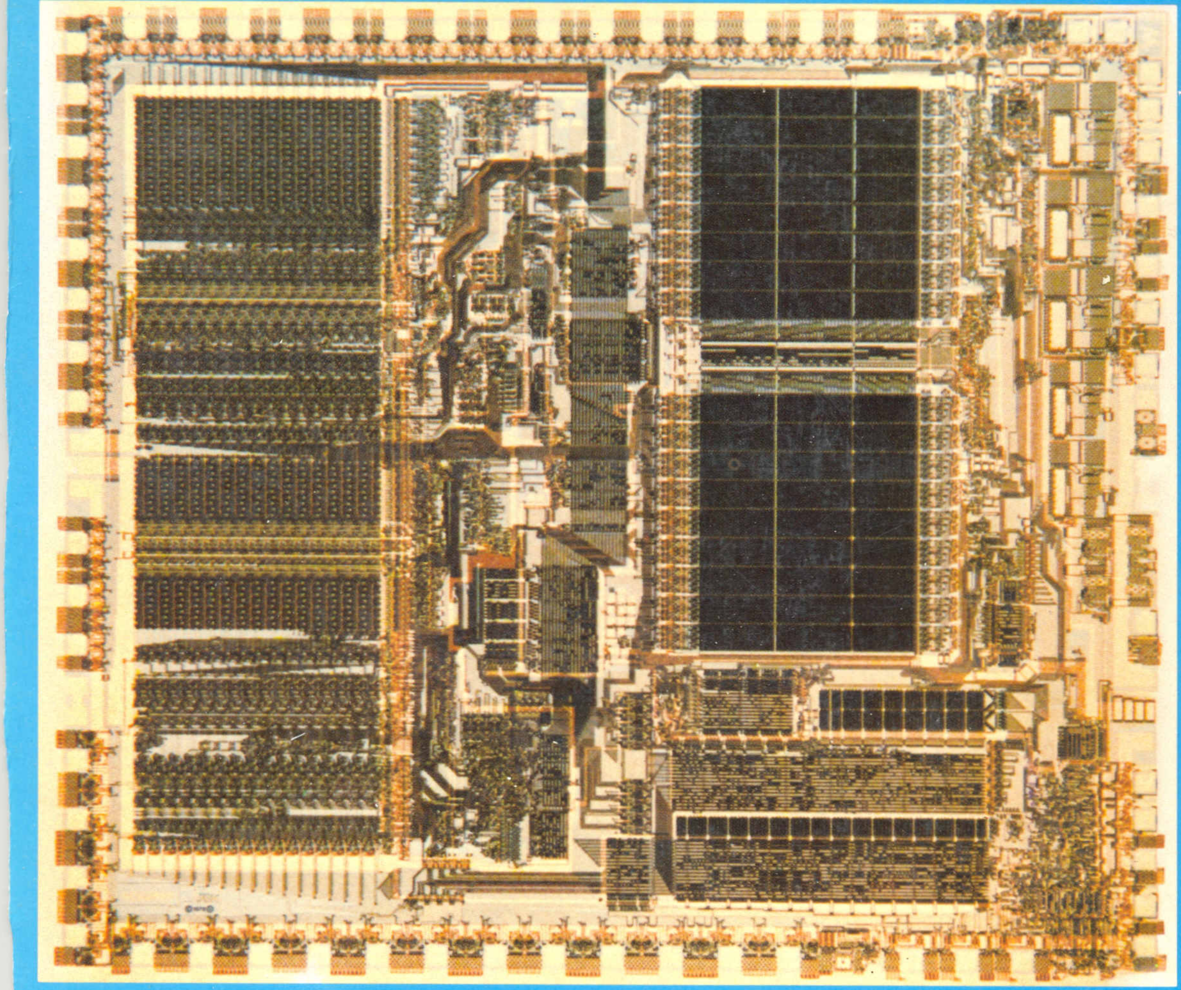
MOTOROLA Semiconductors



MCU/MPU APPLICATIONS MANUAL

MCU/MPU APPLICATIONS MANUAL

This manual contains a compilation of application notes which will help engineers in their system designs. They cover a broad range of Motorola 8/16 bits micro-components.

Specific data on Motorola products can be obtained from any Motorola Distributor or Sales office as can full data on all MCU/MPU products in the Microprocessor manual.

This information has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Motorola reserves the right to make changes to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein. No license is conveyed under patent rights in any form. When this document contains information on a new product, specifications herein are subject to change without notice.

Printed in Switzerland - 3.82

INDEX

INDEX (cont.)

LOW-SPEED MODEM SYSTEM DESIGN USING THE MC6860

Prepared by:
Jon M. DeLaune
Computer Applications

This application note describes modem systems for full duplex originate only, automatic answer-answer only, and answer/originate with automatic answer. Described are the peripheral circuits such as limiters and bandpass filters that surround the MC6860 to make it a 100 Series compatible modem system.

LOW-SPEED MODEM SYSTEM DESIGN USING THE MC6860

GENERAL

Low-speed modem designers will find that the MC6860 MOS LSI Modem with its built-in modulator, demodulator, and supervisory control will allow the design of a high performance, low cost 100 Series type modem. The designer, by selecting from different filter configurations and some surrounding support circuitry, may design either an originate only, answer only, or automatic answer/originate modem system. ,

It is the purpose of this note to cover in some detail these surrounding building blocks that comprise the total system. To familiarize the reader with the MC6860 chip operation, a general overview will be included with a more detailed description to be obtained from the MC6860 data sheet.

BASIC MC6860 CIRCUIT OPERATION

As illustrated in Figure 1, the MC6860 Modem contains a digital modulator, demodulator, and a supervisory control section to handle line disciplines for full duplex originate, auto-answer, and auto-disconnect operations.

Modulator

The modulator section converts serial digital data into analog frequencies for output to the telephone network. The analog output from the modem is a digital synthesized sinewave having one of four possible frequencies as listed in Figure 2. The modulation scheme used is frequency shift keying (FSK), where a logic "0" (space) is the lower frequency and a logic "1" (mark) is the upper or higher

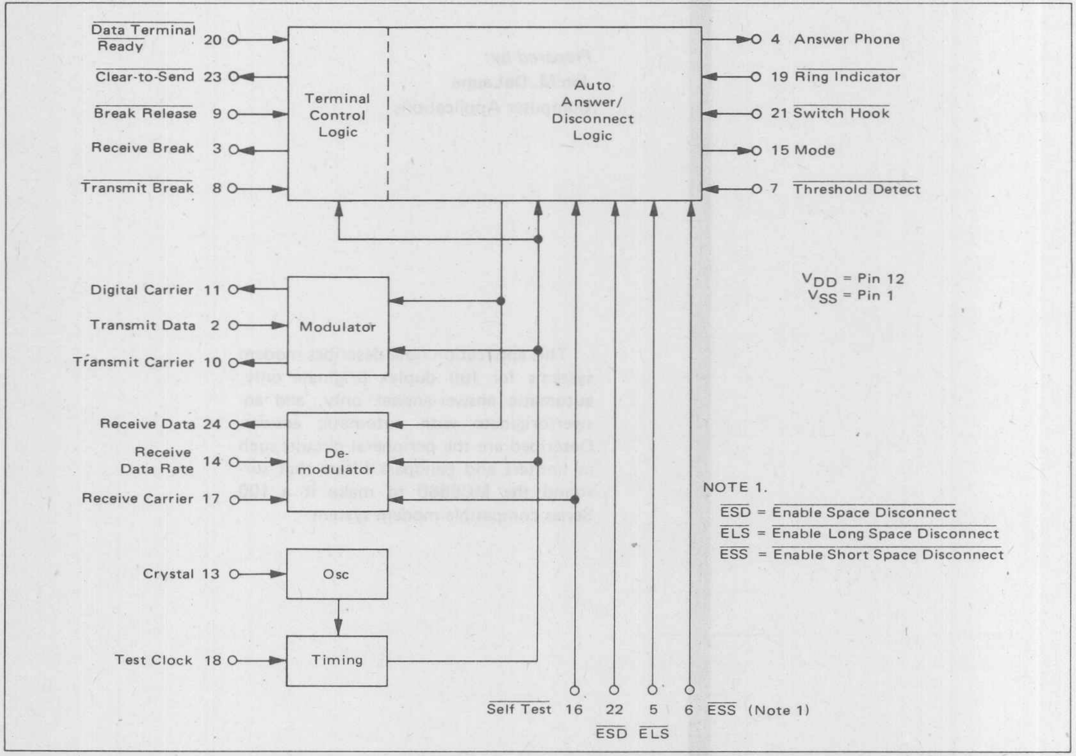


FIGURE 1 — MC6860 Modem

Output	Originate	Answer
Mark	1270 Hz	2225 Hz
Space	1070 Hz	2025 Hz

FIGURE 2 — Output Frequency Shift Keying Pairs

frequency of either the originate or answer frequency pairs. The analog signal output level from the modulator is typically 350 millivolts (rms) into a load of 100 k ohms; therefore, for the MC6860 to interface into a 600 ohm line system such as the telephone network with the necessary signal magnitude, an external transmit buffer will be required.

Demodulator

The demodulator section receives either the lower or upper (answer or originate modem) frequency tone pairs, and by a technique of digital half-cycle detection determines the presence of a mark or a space frequency and will output at the Receive Data pin either a digital logic "1" or "0" to the terminal or computer equipment. The incoming analog signal from the line should be bandlimited (filtered) and limited (amplified/clipped) prior to the demodulator carrier input to remove interfering signals and system noise. The limited input signal presented to the demodulator input should be at 50% duty cycle (±4%) over the full input signal dynamic range and be at a TTL compatible input level in order to maintain low bit-error-rate performance.

Supervisory Control

The supervisory control section of the MC6860 contains the necessary logic to provide initial inter-modem handshaking as well as operational protocol, such as automatic answer, originate only, initiate disconnect, and automatic disconnect. A graphical illustration of these control operations provided by the MC6860 is shown in Figures 3, 4, 5, and 6. Signals provided by the MC6860 for interfacing between a data terminal and either a CBS or a CBT telephone network data coupler are shown at the top right of Figure 1. $\overline{\text{Switch Hook}}$ ($\overline{\text{SH}}$), $\overline{\text{Ring Indicator}}$ ($\overline{\text{RI}}$), and Answer Phone (An Ph) signals will interface directly with a CBT data coupler, or with a CBS data coupler when RS-232 interface circuits are used. Both of these data coupler interface methods will be illustrated in later system implementation examples.

Additional control signals that are provided for data terminal control are: $\overline{\text{Data Terminal Ready}}$ ($\overline{\text{DTR}}$), $\overline{\text{Clear-to-Send}}$ ($\overline{\text{CTS}}$), Receive Break (Rx Brk), $\overline{\text{Transmit Break}}$ ($\overline{\text{Tx Brk}}$), and $\overline{\text{Break Release}}$ ($\overline{\text{Brk R}}$). The Mode output is a control function that is system oriented for the surrounding filter block. This output can be used to control switchable filters to provide a full automatic answer/originate modem system. A logic low level at the Mode output pin indicates the demodulator is in the answer mode of operation and will demodulate 1070 Hz and 1270 Hz incoming signals. When the Mode output is in a high state, the frequencies demodulated will be 2025 Hz and 2225 Hz. A design example using switchable filters will be illustrated in a later section.

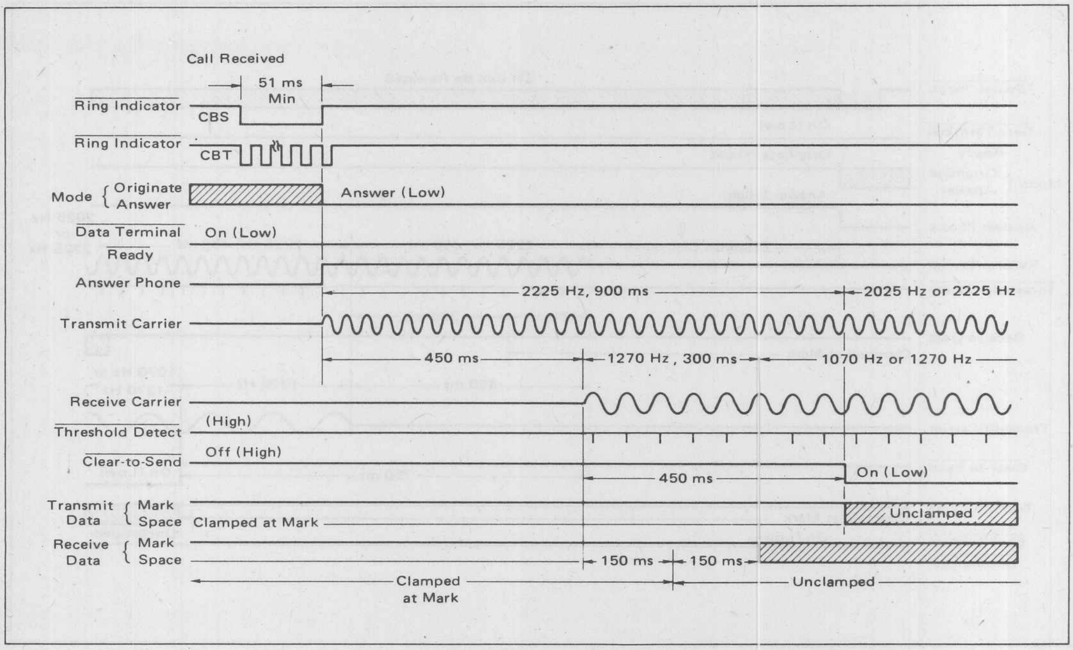


FIGURE 3 — Automatic Answer

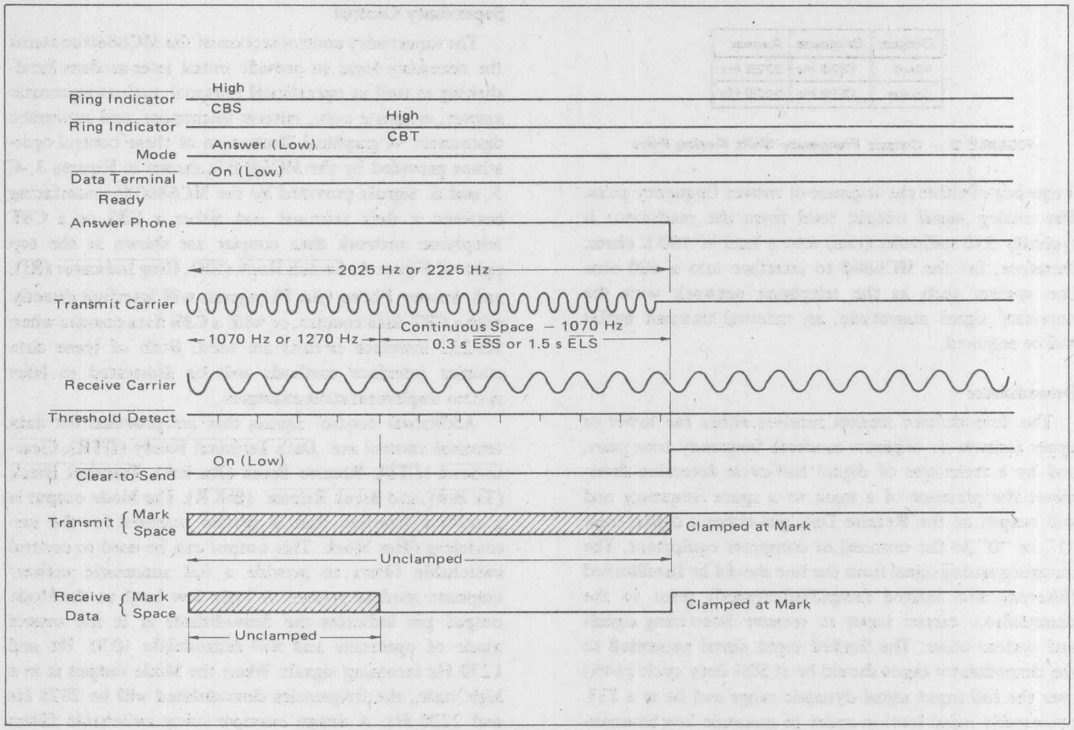


FIGURE 4 — Automatic Disconnect — Long or Short Space

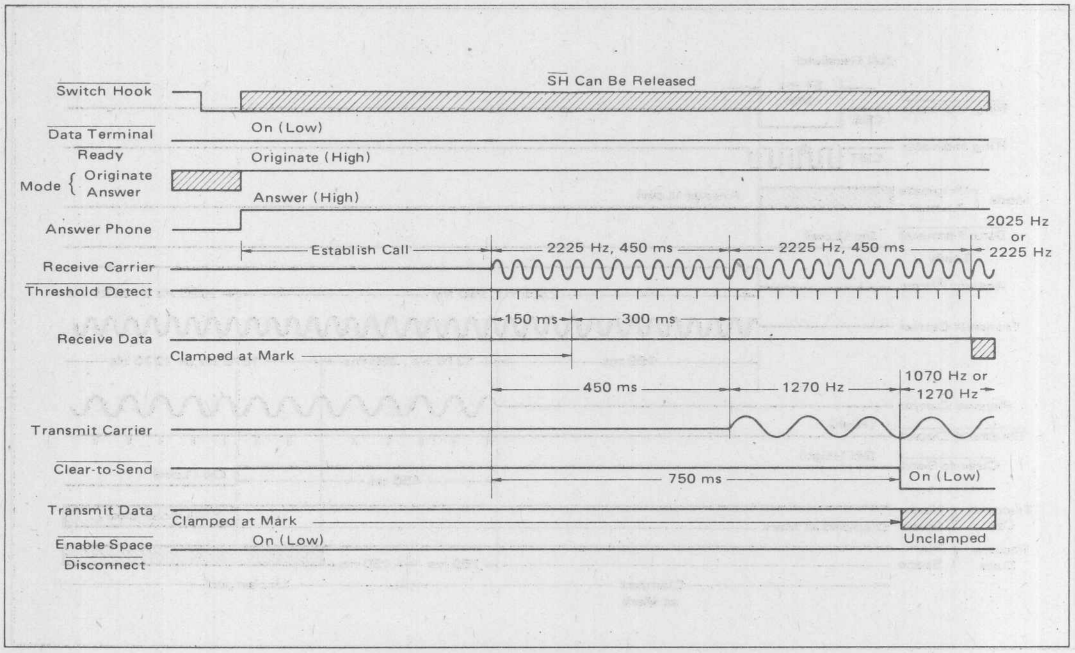


FIGURE 5 — Originate Only

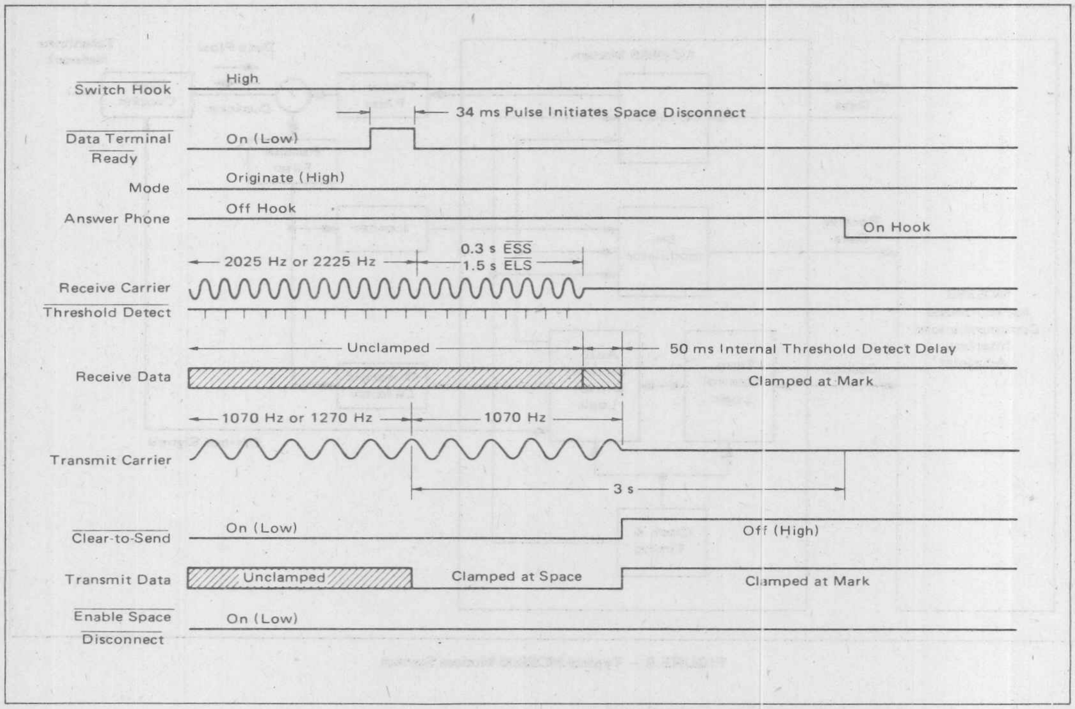


FIGURE 6 – Initiate Disconnect

A self test feature is included in the MC6860 for testing the modulator/demodulator sections. When a low logic level is applied to the $\overline{\text{Self Test}}$ ($\overline{\text{ST}}$) input pin, the demodulator is switched to detect the modulator transmitted frequency pair. Channel establishment obtained during initial handshaking is not lost, with only the Mode output changing state during initiation of self test as shown in Figure 7. This test feature allows the modulator, demodulator, and interval timer circuitry to be checked for proper operation during diagnostic system test.

$\overline{\text{ST}}$	$\overline{\text{SH}}$	$\overline{\text{RI}}$	Mode
H	L	H	H
H	H	L	L
L	L	H	L
L	H	L	H

FIGURE 7 – Mode Control Truth Table

MODEM FILTER DESIGN

Filter networks are among the most important surrounding element blocks in a modem system. As shown in Figure 8, a filter block is used in the receive carrier signal path and another filter block is used in the transmit carrier signal path. The transmit carrier filter may not be required in answer only modem designs but is required for originate mode operation.

The receive filter must provide sufficient adjacent channel rejection to provide good bit-error performance. During answer only operation, the filter must pass the receive frequencies of 1070 and 1270 Hz, but reject the adjacent channel local transmit frequencies of 2025 and 2225 Hz.

Typically, the receive carrier bandpass filter should provide greater than 35 dB attenuation to the adjacent channel. During full duplex originate operation, the local transmit signal produces second harmonic energy within the receive filter bandpass (2 x 1070 Hz = 2140 Hz). To reduce this frequency component in the receive filter passband, a transmit carrier filter must be included. This transmit filter may be either a low pass, a high pass, or a bandpass filter dependent upon the designed mode of operation of the modem: originate only, answer only, or auto answer/originate.

The filter design example presented is a bandpass configuration which could be used in either the transmit or receive signal paths with only component value changes. The transmit filter must have a pass frequency of 2025-2225 Hz when the modem is used as an answer only modem (receiving frequencies of 1070-1270 Hz). The opposite configuration is true when the modem is in the originate only mode of operation (transmit frequencies of 1070-1270 Hz and receive frequencies of 2025-2225 Hz).

A design example is presented, with design tables and equations to solve for the modem system bandpass filter

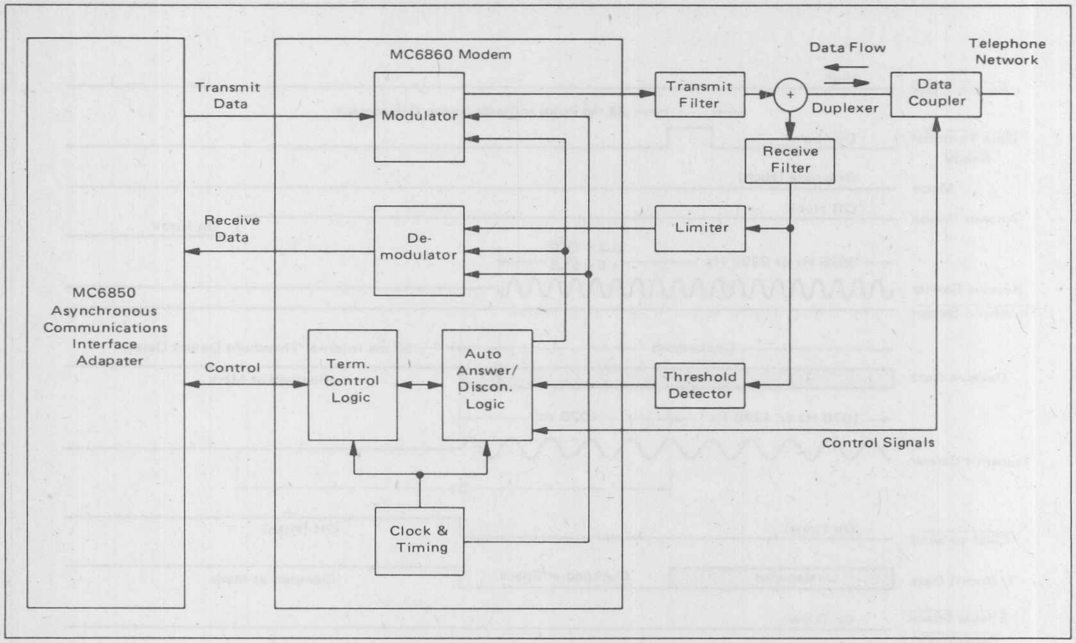


FIGURE 8 – Typical MC6860 Modem System

component values. A 6-pole answer filter is developed in detail in this application note, whereas a 6-pole originate filter has values tabulated only. Also tabulated are component values for 8-pole, 50-dB receive filters and 4-pole, 25-dB transmit filters.

A filter design may take one of many forms. The included design examples use a 0.5 dB ripple Chebyshev approximation. The filter element configuration used is a multiple feedback bandpass as shown in Figure 9. As indicated in Figure 10, the Chebyshev filter will provide a high degree of attenuation in the stop band, but with less phase linearity than a Butterworth or Bessel filter. Linear phase or group delay in the passband is an important design consideration for modem filter design. Error performance and demodulator phase/bias distortion of the modem system is affected by unequal delay of data frequencies within the filter passband. Therefore, it is important to provide filters that not only provide sharp stopband attenuation, but also provide some degree of phase linearity in the passband. By designing the Chebyshev filter to have a wider bandwidth than required for FSK (frequency shift keyed) data recovery, the designer can maximize phase linearity within the required passband.

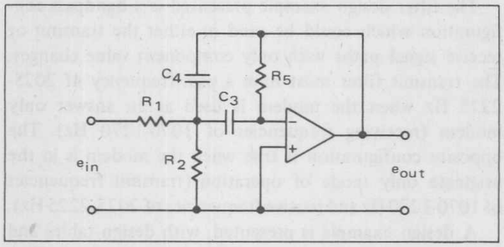


FIGURE 9 – Multiple Feedback Bandpass Filter Element

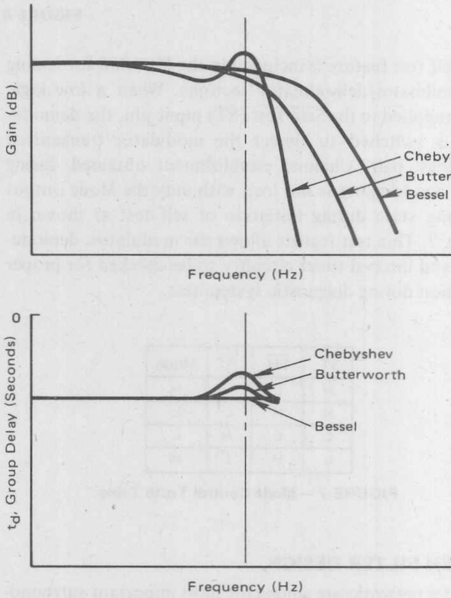


FIGURE 10 – Filter Approximation Characteristics

Determining the minimum filter bandwidth comes by investigating the received signal characteristics. Data communication theory states that data transmitted by FSK can be recovered by detecting the data carrier and the first sidebands. At a data rate of 300 bits per second and a data format of alternate mark and space, the first sidebands occur ±150 Hz from the carrier which is located halfway between the mark and space frequencies. Therefore, the minimum bandwidth for the receive bandpass filter is 300 Hz. Typically, frequencies within this 300 Hz bandwidth should undergo no greater than 0.8 millisecond change in group delay. Group delay is defined by:

$$t_d = \frac{\triangle\phi}{\triangle F}\,\frac{1}{360^o/\text{cycle}}$$

where $\triangle\phi$ = change in phase in degrees
$\triangle F$ = change in frequency in Hz

To maintain less than 0.8 millisecond group delay at a data rate of 300 bits per second requires an overall filter bandpass of 400 Hz. This results in the low frequency pair (answer) filter passband being between 970 Hz and 1370 Hz (6-pole, 0.5 dB ripple Chebyshev).

Filter Design Steps

The modem bandpass filter examples will be designed using the following procedural steps:

(1) Determine the required prototype low pass filter shape factor from the passband width and stopband attenuation.

(2) Enter Table 1 with the shape factor, passband ripple (A_{max}), and stopband attenuation (A_{min}), to determine the order of the prototype lowpass filter.

(3) From Table 2, determine the location of the prototype low pass filter poles opposite the determined filter order.

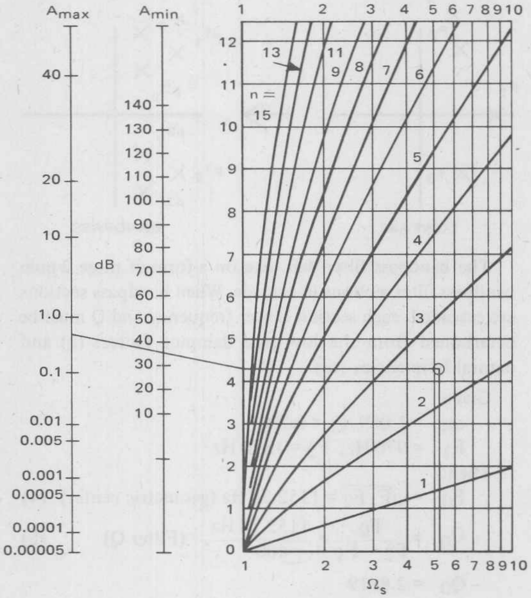
(4) From the low pass filter poles, determine their natural frequency (ω) and damping factor (ξ).

(5) Transform the low pass filter section parameters to cascaded second order bandpass filter design section Q and center frequency values.

(6) Determine the active element operational amplifier gain by solving for center frequency loss and system filter passband gain (A_{VO}).

(7) Use each section Q, frequency, and gain to solve for the bandpass filter passive component values.

TABLE 1 – Complexity Nomograph for Chebyshev Filters (Zverev)



Step (1) – Filter Shape Factor

Figure 11 shows a design example for a typical 6-pole answer modem receive filter design. From this data, it is possible to calculate the filter shape factor (Ω_s) for the prototype filter.

$$\Omega_s = \frac{F_4 - F_3}{F_2 - F_1} = \frac{2225 - 115}{1370 - 970} \qquad (1)$$

$$\Omega_s = \frac{2110}{400} = 5.28$$

TABLE 2 – Pole Locations and Quadratic Factors ($s^2 + a_1s + a_0$) for Chebyshev 0.5 dB Ripple Filter

	0.5 dB Ripple		
Order	Poles	a_0	a_1
2	-0.71281 ± j 1.00404	1.51620	1.42562
3	-0.31323 ± j 1.02193	1.14245	0.62646
	-0.62646		
4	-0.17535 ± j 1.01625	1.06352	0.35071
	-0.42334 ± j 0.42095	0.35641	0.84668
5	-0.11196 ± j 1.01156	1.03578	0.22393
	-0.29312 ± j 0.62518	0.47677	0.58625
	-0.36232		
6	-0.07765 ± j 1.00846	1.02302	0.15530
	-0.21214 ± j 0.73824	0.59001	0.42429
	-0.28979 ± j 0.27022	0.15700	0.57959
7	-0.05700 ± j 1.00641	1.01611	0.11401
	-0.15972 ± j 0.80708	0.67688	0.31944
	-0.23080 ± j 0.44789	0.25388	0.46160
	-0.25617		
8	-0.04362 ± j 1.00500	1.01193	0.08724
	-0.12422 ± j 0.85200	0.74133	0.24844
	-0.18591 ± j 0.56929	0.35865	0.37182
	-0.21929 ± j 0.19991	0.08805	0.43859
9	-0.03445 ± j 1.00400	1.00921	0.06891
	-0.09920 ± j 0.88291	0.78936	0.19841
	-0.15199 ± j 0.65532	0.45254	0.30397
	-0.18644 ± j 0.34869	0.15634	0.37288
	-0.19841		
10	-0.02790 ± j 1.00327	1.00734	0.05580
	-0.08097 ± j 0.90507	0.82570	0.16193
	-0.12611 ± j 0.71826	0.53181	0.25222
	-0.15891 ± j 0.46115	0.23791	0.31781
	-0.17615 ± j 0.15890	0.05628	0.35230

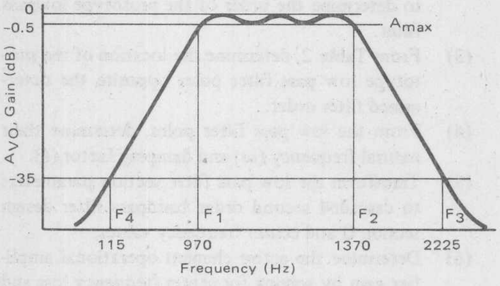


FIGURE 11 – Answer Filter Design Goals

where:

F_1 = lower passband frequency in Hz
F_2 = upper passband frequency in Hz
F_3 = lower stopband frequency in Hz
F_4 = upper stopband frequency in Hz

NOTE:

F_1 and F_2 are ripple bandwidth frequencies, i.e., gain down 0.5 dB.

Steps (2) and (3) – Filter Order and Pole Location

The second step of the filter design process was to determine the complexity of the filter. To determine this complexity, the following information is required:

1. The passband ripple, A_{max}.
2. The minimum stopband attenuation, A_{min}.
3. The ratio of the ripple bandwidth and the first frequency of minimum attenuation, shape factor Ω_s.

With A_{max} = 0.5 dB, A_{min} = -35 dB, and Ω_s = 5.28 enter the nomograph in Table 1 to determine the filter complexity or order.

The nomograph is used by locating the passband ripple A_{max} and the minimum stopband attenuation A_{min} and drawing a line from A_{max} through A_{min} to the left-hand side of the graph. From this point, a horizontal line is drawn to an intersection of the vertical line value of Ω_s. The minimum complexity or order, n, will be the n curve that passes through or above this intersection. In our example, the order n equals 3. This implies that the low pass prototype filter will have 3 poles and, consequently, the final bandpass filter will have 3 pole-pairs.

Table 2 gives the pole locations and quadratic factors for a third order 0.5 dB passband ripple Chebyshev low pass filter.

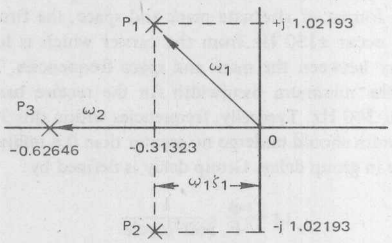
The values obtained from Table 2 are:

$-0.31323 \pm j1.02193$	Complex conjugate pole
$-0.62646 + j0$	Real pole
$a_0 = 1.14245$	Characteristic of non s term
$a_1 = 0.62646$	Characteristic of s term

where the s term equation = $(s^2 + a_1 s + a_0)$

Step (4) – Lowpass Prototype Filter Natural Frequencies and Damping Factors

Using the following relationships, solve for the natural frequencies (ω) and damping factors (ξ):



$$\omega_1{}^2 = (1.02193)^2 + (-0.31323)^2 \quad (2)$$

$$\omega_1 = 1.069$$

$$\text{also, } \omega_1\,\xi_1 = 0.31323 \quad (3)$$

$$\xi_1 = \frac{0.31323}{1.069}$$

$$\xi_1 = 0.293$$

$$\omega_2{}^2 = (0)^2 + (-0.62646)^2$$

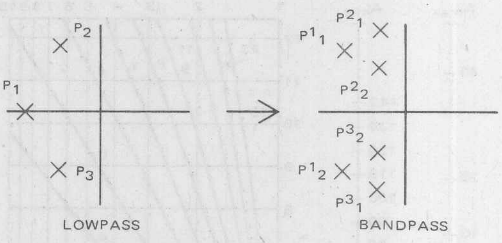
$$\omega_2 = 0.62646$$

$$\text{also, } \omega_2\,\xi_2 = 0.62646$$

$$\xi_2 = 1$$

Step (5) – Filter Section Q and Center Frequency

The complex conjugate pole of the low pass prototype is transformed into a pair of complex conjugate bandpass poles, whereas the real pole of the low pass prototype is transformed into a complex conjugate pair of bandpass poles.



The bandpass filter will take on a form of three 2-pole bandpass filter sections in cascade. When bandpass sections are cascaded, each section center frequency and Q must be determined from the low pass damping factors (ξ) and natural frequencies (ω).

Given:

ω_1 = 1.069, ξ_1 = 0.293
F_1 = 970 Hz, F_2 = 1370 Hz

Then:

$$F_0 = \sqrt{F_1 F_2} = 1152.78 \text{ Hz (geometric center)} \quad (4)$$

$$Q_0 = \frac{F_0}{F_2 - F_1} = \frac{1152.78 \text{ Hz}}{400} \quad \text{(Filter Q)} \quad (5)$$

$$Q_0 = 2.8819$$

$$2\left(\frac{2\xi_1\omega_1}{Q_0}\right)^2$$

Yielding:

$$Q_1 = 9.345$$

Section 2 is a reflected image about F_0 of section 1 for a 3 section cascaded filter (odd order). Recall that a third order low pass when transformed to a bandpass results in two pairs of complex poles (sections 1 and 2) from the low pass complex pole and one pair of complex poles (section 3) from the low pass real pole.

$$Q_1 = Q_2 = 9.345$$

For section 3:

$$Q_3 = \frac{Q_0}{\xi_2\omega_2} = \frac{2.882}{(1)\,(0.627)} = 4.596 \tag{7}$$

Center Frequencies:

$$F_1 = MF_0 \tag{8}$$

where:

$$M = \frac{\xi_1\omega_1 Q_1}{Q_0} + \sqrt{\left(\frac{\xi_1\omega_1 Q_1}{Q_0}\right)^2 -1} \tag{9}$$

$$M = \frac{(0.293)\,(1.069)\,(9.345)}{2.882} + \sqrt{\left[\frac{(0.293)\,(1.069)\,(9.345)}{2.882}\right]^2 -1}$$

$$M = 1.1932$$

$$F_1 = (1.1932)\,(1152.78) = 1375.52 \text{ Hz}$$

$$F_3 = F_0 = 1152.78 \text{ Hz} \tag{11}$$

Step (6) – Center Frequency Loss and Filter Passband Gain

The gain produced by the active elements in the bandpass filter should overcome loss due to the stagger tuned filter sections. Each section of a cascade bandpass filter, except the section centered about ω_0, has a loss as represented by Equation 12. The overall filter center angular frequency ω_0 (Equation 13), section Q, and section center angular frequency ω_n (Equation 14) are required to determine each section's center frequency loss. Once the individual losses are determined, they are summed to arrive at the total cascaded filter loss A_{VO} $(j\omega_0)$.

This value is used in determining filter section gain such that the designed bandpass filter meets design gain goals. The receive filter block must amplify the minimum input line signal to a minimum required limiter input signal.

$$A_{VO_n}\,(j\omega_0)\text{ dB loss} = 20 \log \frac{\dfrac{\omega_n\omega_0}{Q_n}}{\sqrt{(\omega_n{}^2 - \omega_0{}^2)^2 + \left(\dfrac{\omega_n\omega_0}{Q_n}\right)^2}} \tag{12}$$

$$\omega_0 = 2\,\pi\sqrt{F_1F_2} \tag{13}$$

$$\omega_n = 2\pi F_n \tag{14}$$

The following will illustrate the use of Equation 12 to solve for the center frequency loss of the modem answer filter example.

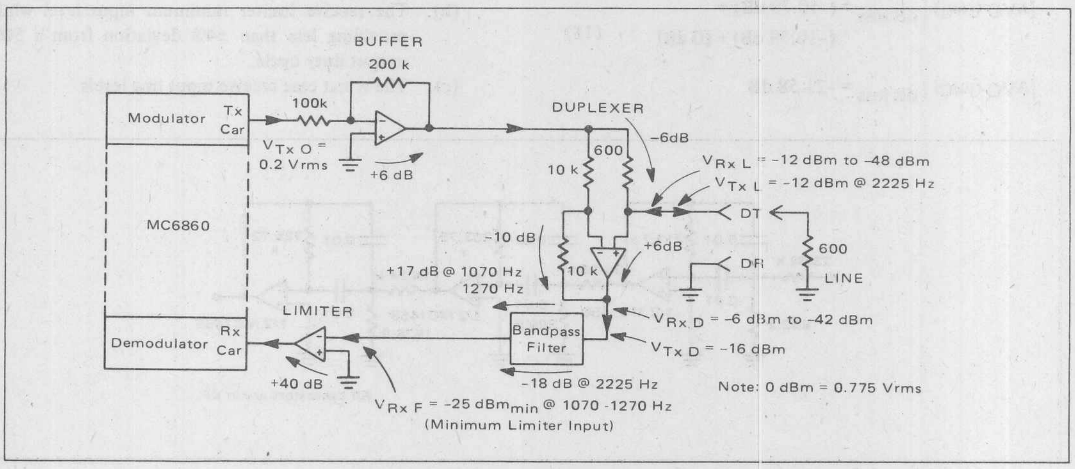


FIGURE 12 – System Level Constraints

Section 1

$$\omega_0 = 2\pi\sqrt{(970)(1370)} = 7.2431 \times 10^3 \text{ rad/s} \qquad (15)$$

$$\omega_1 = 2\pi(1375.52) = 8.6426 \times 10^3 \text{ rad/s} \qquad (16)$$

$$Q_1 = 9.345$$

$$|AVO1\,(j\omega_0)|_{\text{dB loss}} = 20 \log \left[\frac{\dfrac{(8.6426 \times 10^3)(7.243 \times 10^3)}{9.345}}{\sqrt{\left[(8.642 \times 10^3)^2 - (7.243 \times 10^3)^2\right]^2 + \left[\dfrac{(8.642 \times 10^3)(7.243 \times 10^3)}{9.345}\right]^2}} \right] \qquad (17)$$

$$|AVO1\,(j\omega_0)|_{\text{dB loss}} = 20 \log (0.2886)$$

$$|AVO1\,(j\omega_0)|_{\text{dB loss}} = -10.794 \text{ dB}$$

Section 2

$$\omega_0 = 7.243 \times 10^3 \text{ rad/s}$$

$$\omega_2 = 2\pi(966.1) = 6.07 \times 10^3 \text{ rad/s}$$

$$Q_2 = 9.345$$

$$|AVO2\,(j\omega_0)|_{\text{dB loss}} = 20 \log (0.2886)$$

$$|AVO2\,(j\omega_0)|_{\text{dB loss}} = -10.794 \text{ dB}$$

Section 3

$$\omega_0 = 7.243 \times 10^3 \text{ rad/s}$$

$$\omega_3 = 7.243 \times 10^3 \text{ rad/s}$$

$$Q_3 = 4.596$$

$$|AVO3\,(j\omega_0)|_{\text{dB loss}} = 20 \log (1)$$

$$|AVO3\,(j\omega_0)|_{\text{dB loss}} = 0 \text{ dB, due to } \omega_n = \omega_0$$

The total filter center frequency loss is equal to the sum of all sectional losses.

$$|AVO\,(j\omega_0)|_{\text{dB loss}} = (-10.79 \text{ dB}) + (-10.79 \text{ dB}) + (0 \text{ dB}) \qquad (18)$$

$$|AVO\,(j\omega_0)|_{\text{dB loss}} = -21.58 \text{ dB}$$

Figure 12 illustrates the design goals that are used to determine the receive filter passband gain for the answer only modem system. The answer filter provides 35 dB of attenuation to 2225 Hz relative to the filter passband. This results in -34 dBm of unwanted signal level being present at the limiter input. To maintain a probability of error (P_e) $\leqslant 1 \times 10^{-5}$, a signal-to-noise ratio at the limiter input must be greater than +12.12 dB. The theoretical probability of error (P_e) curve for non-coherent FSK is determined by:

$$P_e = 1/2\, e^{-\left[\frac{\left(\frac{V_s}{V_n}\right)^2 \left(\frac{BW_n}{BW_s}\right)}{2}\right]} \qquad (19)$$

where V_s = signal level
V_n = noise level
BW_n = noise bandwidth (400 Hz)
BW_s = signal bandwidth (300 Hz)

In calculating the voltage gain required by the receive active filter block, the following constraints should be considered:

(a) The signal to noise performance required by the modem system.

(b) The receive limiter minimum input level while providing less than ±4% deviation from a 50% output duty cycle.

(c) The worst case receive input line levels.

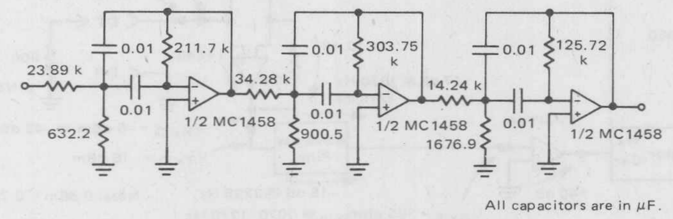


FIGURE 13a – Answer Filter Component Values

(d) At the maximum input line levels, the designed filter gain should not saturate any active stage of the filter.

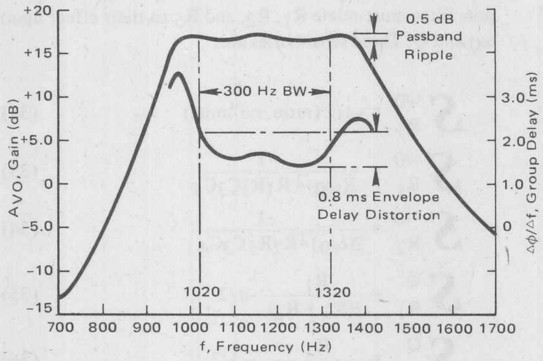


FIGURE 13b – Answer Filter Gain and Group Delay

The use of the MLM311 as a receive signal limiter provides 40 dB of signal gain while maintaining a limited output level having less than ±2% deviation from a 50% duty cycle with a -25 dBm applied input level ($V_{Rx\ F}$).

The telephone line receive level for the answer only example ranges between -12 dBm and -48 dBm. An active duplexer provides 6 dB of signal gain to these line levels resulting in filter input levels ($V_{Rx\ D}$) between -6 dBm and -42 dBm.

From the above information, the active filter must provide the following passband gain.

$$A_{VO} = |V_{Rx\ D}\text{min}| - |V_{Rx\ F}\text{max}| \qquad (20)$$

$$A_{VO} = 42 \text{ dB} - 25 \text{ dB} = 17 \text{ dB passband gain}$$

The amount of operational amplifier gain used in the filter design is based on both the passband gain requirements and the filter center frequency loss.

$$A_{VOtotal} = |A_{VO}\text{ (passband)}| + |A_{VO}\text{ (center frequency loss)}| \qquad (21)$$

$$A_{VOtotal} = 17 \text{ dB} + 21.58 \text{ dB} = +38.58 \text{ dB}$$

This requires that each of the three filter sections provide a gain of:

$$A_{VO} = \frac{+38.58 \text{ dB}}{3} = +12.86 \text{ dB or } 4.41 \text{ volts/volt.} \quad (22)$$

Step (7) – Filter Component Values

Now that each section gain, center frequency, and design Q is known, the actual filter component values can be calculated (reference Figure 9).

Section 1:

$F_1 = 1375.52$ Hz

$\omega_1 = 8.6426 \times 10^3$ rad/s

$Q_1 = 9.345$

$A_{VO1} = 4.41$ (gain of section)

$C_3 = C_4 = 0.01\ \mu F$ (using equal value capacitors)

$$R_5\text{ (uncorrected)} = \frac{2Q_1}{\omega_1 C} = \frac{2\,(9.35)}{2\pi(1375.5)\,(1 \times 10^{-8})}$$

$$= 216.4 \text{ k}\Omega \qquad (23)$$

$$R_1\text{ (uncorrected)} = \frac{R_5}{2\,A_{VO1}} = \frac{216.4 \text{ k}}{2(4.41)}$$

$$= 24.5 \text{ k}\Omega \qquad (24)$$

$$R_2\text{ (uncorrected)} = \frac{R_1 R_5}{4Q_1{}^2 R_1 - R_5}$$

$$= \frac{(24.5 \text{ k})\,(216.4 \text{ k})}{4(9.35)^2\,(24.5 \text{ k}).-216.4 \text{ k}}$$

$$= 634.9\ \Omega \qquad (25)$$

These three resistor values, if used to initially implement the first bandpass section, would not produce exact design goals. Filter response will shift due to non-ideal operational amplifier parameters such as dc gain (A_{VOL}), gain bandwidth product (GBW), and input impedance (z_{in}).

To offset any shift in filter response, new values for selection Q, gain and frequency should be calculated taking into account the operational amplifier parameters. These corrected values will be used to obtain new values for R_5, R_1, and R_2, resulting in a filter response very near design goals.

Corrected values for ω_n, Q_n, and A_{VO_n} are calculated using the following MC1458 operational amplifier parameters.

$A_{VOL} = 1 \times 10^5$ volts/volt

GBW $= 1 \times 10^6$ Hz, 6.283×10^6 rad/s

$z_{in} = 1 \times 10^6$ ohms

$$\omega_{C_1} = \frac{\omega_1}{1 - Q_1\left(\frac{\omega_1}{GBW}\right)} \qquad (26)$$

$\omega_{C_1} = 8.755 \times 10^3$ rad/s, 1393.4 Hz

$$Q_{C_1} = \frac{Q_1}{1 - Q_1\left[\frac{2Q_1}{A_{VOL}} + \left(\frac{R_5}{z_{in}} - 1\right)\ \frac{\omega_1}{GBW}\right]} \qquad (27)$$

Plugging in values we obtain:

$Q_{C_1} = 9.27$

$$A_{VOC_1} = \frac{A_{VO1}}{1 - Q_1\left[\frac{2Q_1}{A_{VOL}} + \left(\frac{R_5}{z_{in}}\right)\ \frac{\omega_1}{GBW}\right]} \qquad (28)$$

$A_{VOC_1} = 4.43$

Using these corrected values of section center frequency, Q, and section gain, solve for the corrected values of R_1, R_2, and R_5:

$$R_5 = \frac{2Q_{C1}}{\omega_{C1} C} \tag{29}$$

$$R_5 = \frac{2\,(9.27)}{(8.755 \times 10^3)\,(1 \times 10^{-8})} = 211.7\text{ k}\Omega$$

$$R_1 = \frac{R_5}{2A_{VOC}} \tag{30}$$

$$R_1 = \frac{2.117 \times 10^5}{2\,(4.43)} = \ 23.89\text{ k}\Omega$$

$$R_2 = \frac{R_1 R_5}{4Q_{C1}{}^2 R_1 - R_5} \tag{31}$$

$$R_2 = \frac{(2.389 \times 10^4)\,(2.117 \times 10^5)}{4(9.27)^2\,(2.389 \times 10^4) - 2.117 \times 10^5}$$

$$R_2 = 632.2\ \Omega$$

Section 2:

$F_2 = 966.1$ Hz
$\omega_2 = 6.07 \times 10^3$ rad/s
$Q_2 = 9.345$
$A_{VO2} = 4.43$
$C_3 = C_4 = 1 \times 10^{-8}$ F

Solving as in Section 1 using Equations 23 through 31, we obtain:

$\omega_{C2} = 6.1255 \times 10^3$ rad/s, 974.9 Hz
$Q_{C2} = 9.30$
$A_{VOC2} = 4.43$
$R_5 = 303.75$ kΩ
$R_1 = 34.28$ kΩ
$R_2 = 900.5$ Ω

Section 3:

$F_3 = 1152.73$ Hz
$\omega_3 = 7.243 \times 10^3$ rad/s
$Q_3 = 4.596$
$A_{VO3} = 4.41$
$C_3 = C_4 = 1 \times 10^{-8}$ F

Solving as in section 1 and 2, we obtain:

$\omega_{C3} = 7.281 \times 10^3$ rad/s, 1158.87 Hz
$Q_{C3} = 4.58$
$A_{VOC3} = 4.41$
$R_5 = 125.72$ kΩ
$R_1 = 14.24$ kΩ
$R_2 = 1676.9$ Ω

The complete answer filter is shown in Figure 13a with the filter response and envelope delay curves shown in Figure 13b. If the filter is not optimum after construction, it may be fine tuned by the following method.

In tuning filters, one of the most useful parameters is the sensitivity of the filter to element variations. Sensitivity is defined as a measure of the dependence of a network upon the change of some parameter of the network. The sensitivities of importance to the multiple-feedback bandpass filter must relate R_1, R_2, and R_5 to their effect upon ω_0 and Q. These sensitivities are:

$$S^{\omega_0}_{R_5} = -1/2 \text{ (ratio, no units)} \tag{32}$$

$$S^{\omega_0}_{R_1} = \frac{-1}{2(\omega_0)^2 R_1 R_5 C_3 C_4} \tag{33}$$

$$S^{\omega_0}_{R_2} = \frac{-1}{2(\omega_0)^2 R_2 R_5 C_3 C_4} \tag{34}$$

$$S^{Q}_{R_1} = \frac{R_1}{2(R_1 + R_2)} - 1/2 \tag{35}$$

$$S^{Q}_{R_2} = \frac{R_2}{2(R_1 + R_2)} - 1/2 \tag{36}$$

$$S^{Q}_{R_5} = +1/2 \tag{37}$$

In practice, $R_1 \gg R_2$ such that

$$S^{Q}_{R_1} \rightarrow 0$$

$$S^{Q}_{R_2} \rightarrow -1/2$$

These sensitivities imply that to change section Q, R_2 should be adjusted. If R_2 were increased, for example 20%, section Q will decrease 10%. Notice that the sensitivity of Q to changes in R_2 and R_5 is equal and opposite in magnitude. This implies that if R_2 and R_5 are changed by the same percentage, but in opposite directions, section Q will not change. Also, as R_5 is adjusted, it changes the section center frequency by a ratio of -1/2.

Filter Tuning Procedure

Section Center Frequency:

(a) Increase/decrease R_5 for a corresponding decrease/increase in section center frequency ω_0.

(b) Increase/decrease R_2 by the same percentage of increase/decrease applied to R_5 in step (a) to maintain constant sectionQ.

Section Q:

(a) Increase/decrease R_2 for a corresponding decrease/increase in section Q.

ORIGINATE FILTER DESIGN

Basically, the originate receiving filter design procedures are identical to the answer filter example. The one major difference is that the filter center frequency is shifted to accept 2025 – 2225 Hz signals. One might also note that the second harmonics of the local transmit signals in the originate mode (1070 – 1270 Hz) fall within and just values for the 6-pole originate receive filter are:

Section 1:

$F_1 = 2425.81$ Hz
$Q_1 = 16.56$
$A_{VO1} = 4.48$
$C_3 = C_4 = 1 \times 10^{-8}$F
$R_1 = 24.26$ kΩ
$R_2 = 199.76$ Ω
$R_5 = 217.258$ kΩ

Section2:

$F_2 = 1985.62$ Hz
$Q_2 = 16.67$
$A_{VO2} = 4.48$
$C3 = C4 = 1 \times 10^{-8}$F
$R_1 = 29.85$ k
$R_2 = 242.36$ Ω
$R_5 = 267.23$ kΩ

Section 3:

$F_3 = 2154.01$ Hz
$Q_3 = 8.32$
$A_{VO3} = 4.43$
$C_3 = C_4 = 1 \times 10^{-8}$F
$R_1 = 13.88$ kΩ
$R_2 = 458.85$ Ω
$R_5 = 122.913$ kΩ

The complete 6-pole receive originate filter is shown in Figure 14a, with the response and envelope delay curves shown in Figure 14b.

8-POLE, -50 dB RECEIVE AND 4-POLE, -25 dB TRANSMIT FILTER DESIGN

A complete full duplex modem system will most likely require operation with input signals down to -50 dBm at the line input. This requires a receive filter network having at least 8 poles to provide the necessary attenuation to adjacent duplex channel interference and a local transmit filter having 4 poles to provide 25 dB local transmit signal harmonic rejection. The construction of an 8-pole or 4-pole filter takes on the same cascaded form as the illustrated

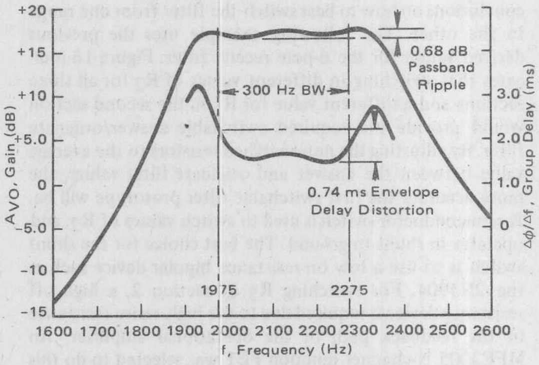


FIGURE 14b – Originate Filter Gain and Group Delay

RECEIVE ORIGINATE

Section	1	2	3	4
R_1 (Ω)	31.42 k	39.54 k	14.71 k	16.1 k
R_2 (Ω)	146.8	181.15	396.29	432.32
R_5 (Ω)	288.64 k	363.27 k	132.15 k	144.66 k

RECEIVE ANSWER

Section	1	2	3	4
R_1 (Ω)	31.08 k	46.34 k	14.51 k	17.1 k
R_2 (Ω)	468.48	690.57	1397.94	1643.88
R_5 (Ω)	283.33 k	422.31 k	131.38 k	154.8 k

TRANSMIT ORIGINATE

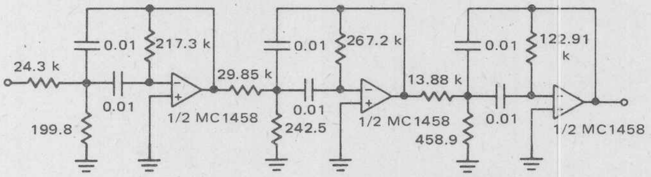
Section	1	2
R_1 (Ω)	15.73 k	20.56 k
R_2 (Ω)	1218.55	1586.55
R_5 (Ω)	130.47 k	170.47 k

TRANSMIT ANSWER

Section	1	2
R_1 (Ω)	16.17 k	18.78 k
R_2 (Ω)	366.95	423.79
R_5 (Ω)	133.25 k	154.81 k

Note: All Capacitors = 0.01 μF

FIGURE 15 – 8-Pole, -50 dB Receive and 4-Pole, -25 dB Transmit Filter Values



All capacitors are in μF.

FIGURE 14a – Originate Filter Component Values

AUTOMATIC ANSWER/ORIGINATE MODEM SYSTEM

The filter design for a fully automatic answer/originate modem system must have switchable bandpass characteristics. By tabulating the previous component values for both the answer and originate filters, one can draw some conclusions on how to best switch the filter from one range to the other. The following example uses the previous derived values for the 6-pole receive filter. Figure 16 indicates that switching in different values of R_2 for all three sections and a different value for R_5 in the second section would provide the required switchable answer/originate filter. By adjusting the non-switched resistors to the average value between the answer and originate filter values, the more accurate the first switchable filter prototype will be. A semiconductor switch is used to switch values of R_2, and operates in shunt to ground. The best choice for the shunt switch is to use a low on-resistance bipolar device such as the 2N3904. For switching R_5 of section 2, a high off resistance device is required due to the high series resistance in the feedback path of the operational amplifier. An MFE2005 N-channel junction FET was selected to do this job. Figure 17a illustrates the fully automatic answer/originate switchable filter system. Also shown are the transmit buffer, duplexer, threshold detector, limiter, and mode control level translator sections. The level translator, which provides the correct on/off voltage levels to the bipolar FET switches, receives its answer/originate command from the MC6860 modem mode control output pin. The measured response and envelope delay for the switchable 6-pole receive filter design is shown in Figure 17b.

Figure 18 illustrates the complete modem system with the RS-232 interface to the CBS data coupler, and the direct interface to a CBT data coupler. Automatic disconnect option inputs are handled by PC board mounted switches. The complete automatic modem, less the power supply, may be easily constructed on a single 4 x 5 printed circuit board.

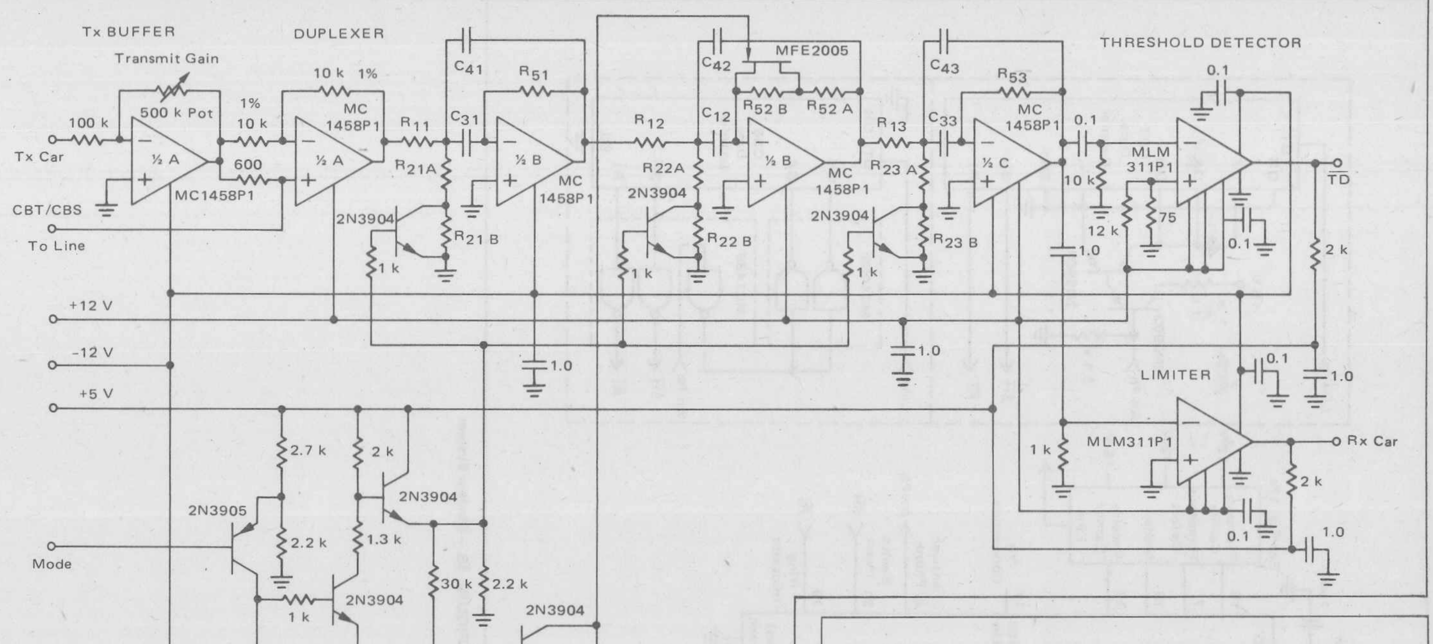
CONCLUSION

A low-speed modem design has been presented using the MC6860 LSI MOS digital Modem integrated circuit. Included has been a system design example using filter design tables and equations to develop a complete modem system. Also included have been component values for filter designs which may be used to develop full duplex modem systems.

The availability of this LSI modem circuit along with the presented filter designs should provide a very useful building block for the OEM modem and terminal designers by providing him precise digital modulation, demodulation, and supervisory control. The modem designer will find that a design approach using the MC6860 modem will also provide an impressive system size reduction as well as a better price-performance choice for his present and future low speed modem designs.

Resistor	Answer 1070-1270 Hz	Originate 2025-2225 Hz	Average or △ Value	Answer Switched	Originate Switched
R_{11}	23.89 k	24.26 k	24.08 k	24.1 k	24.1 k
R_{21}	632.2	199.76	△ 432.4	632	200
R_{51}	211.7 k	217.26 k	214.48 k	214.5 k	214.5 k
R_{12}	34.28 k	29.85 k	32.07 k	32.1 k	32.1 k
R_{22}	900.5	242.36	△ 658.2	900	242
R_{52}	303.75 k	267.23 k	△ 36.5 k	304 k	267 k
R_{13}	14.24 k	13.88 k	14.06 k	14.06 k	14.06 k
R_{23}	1676.9	458.85	1218.05	1677	459
R_{53}	125.72 k	122.91 k	124.32 k	124.3 k	124.3 k

FIGURE 16 – Switchable Modem Filter Values



All Capacitors are in μF.
C_{3n}, C_{4n} = 0.01 μF ± 1%.
Resistors R11 thru R53 are ±1%.

R11 = 24.1 kΩ	R12 = 32.1 kΩ	R52B = 37 kΩ
R21A = 200 Ω	R22A = 242 Ω	R13 = 14.06 kΩ
R21B = 432 Ω	R22B = 658 Ω	R23A = 459 Ω
R51 = 214.5 kΩ	R52A = 267 kΩ	R23B = 1218 Ω
		R53 = 124.3 kΩ

FIGURE 17a – Switchable Filter/Duplexer

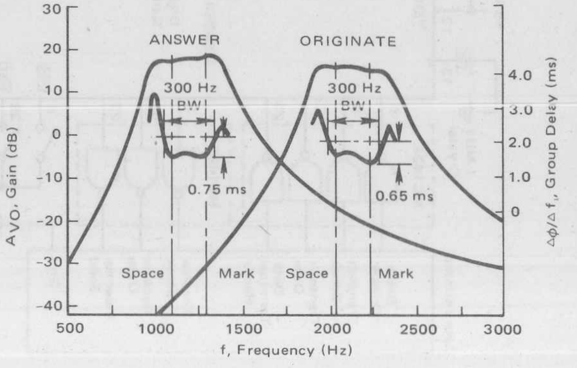


FIGURE 17b – Switchable Filter Response

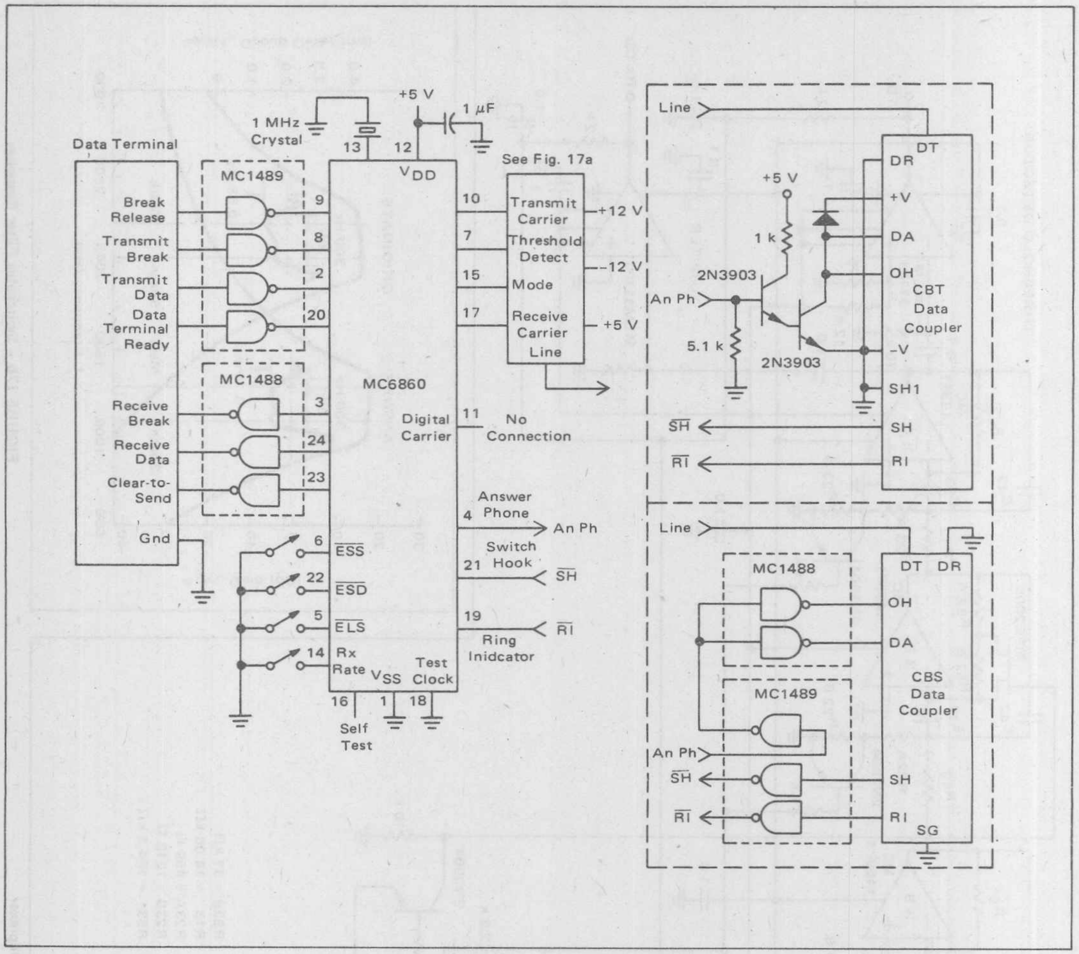


FIGURE 18 – Modem System

AN-754

DEVICE OPERATION AND SYSTEM IMPLEMENTATION OF THE ASYNCHRONOUS COMMUNICATIONS INTERFACE ADAPTER (MC6850)

Prepared by
Karl Fronheiser
Computer Applications Engineering

This application note provides ACIA operational information beyond that included in the data sheet, specifically, information on power-on reset/master reset operation and status register operation. System implementation examples and their associated software are also illustrated and discussed. One of these examples is a data communication application using the MC6860 Modem.

DEVICE OPERATION AND SYSTEM IMPLEMENTATION OF THE ASYNCHRONOUS COMMUNICATIONS INTERFACE ADAPTER (MC6850)

INTRODUCTION

Microcomputer systems must be provided with an efficient means of communicating with peripheral equipment such as modems, teletypes, CRT terminals, and keyboard/printers. The microcomputer manipulates parallel data byte information at high speeds relative to the slow speed asynchronous data format required for communicating with peripherals. Therefore, an efficient interface adapter to convert the processor parallel data byte information into a serial asynchronous data format and vice-versa is a highly desirable system function. This relieves the microprocessor of this time-consuming task. A device providing the above data formatting/interface function is the MC6850 Asynchronous Communications Interface Adapter (ACIA). One side of the ACIA is directly compatible with Motorola's MC6800 (MPU) microprocessor bus while the other side is compatible with peripherals that use an asynchronous data format.

The asynchronous data format characteristics are used by the ACIA to establish bit and character synchronization in the absence of a clock that has been pre-synchronized to the data. An asynchronous data format consists of a serial bit stream with the data bits preceded by a start bit and followed by one or more stop bits. The ACIA converts a character which was serially received from peripheral equipment to a parallel byte with the start, stop, and parity bits deleted from the character. Also, the parallel bytes from the microprocessor are converted to a serial form with start, stop, and optional parity bits appended to the character. Performing these functions in hardware outside of the processor enables the microprocessor to more efficiently communicate with peripheral equipment by using a minimum of software overhead.

The ACIA consists of control, status, transmit data and receive data registers; data bus buffers; transmit and receive shift registers; and peripheral control as shown in the block diagram of Figure 1. Since basic operational information on the ACIA is contained in the ACIA data sheet, this application note will provide additional information to supplement the data sheet with a minimum of repetitive information. The first section of this note provides a description of the operation of the transmitter and receiver portions of the ACIA with reference to appropriate timing diagrams. The second section covers the aspects of the power-on and master reset functions for initialization of the ACIA. The third section covers a detailed description of the ACIA status register bits. The fourth section covers a system implementation of the ACIA as a data communications link in a microcomputer based system. The last section provides examples of the software requirements for initializing the ACIA, and the transmit/receive subroutines for the transmission of data. Additional application information on Motorola's MPU family is available in the "M6800 Microprocessor Applications Manual."

TRANSMITTER/RECEIVER OPERATION

This section covers the internal transmitter/receiver operation of the ACIA as well as the timing relationship between characters being transmitted or received and their associated status register bits. It should be noted that prior to the transmission and/or reception of data, the ACIA must be initialized as described in the "Power-on Reset/Master Reset" section.

Data is transferred to/from the four internal registers of the ACIA on the trailing transition (negative edge) of the signal on the enable input (E). For example, a write data command (RS = 1, R/W = 0) transfers data into the transmit data register on the trailing transition of the enable input signal. In a typical MPU based system, the enable input signal is generated from the ANDing of the Valid Memory Address (VMA) and $\phi 2$.

Transmitter

In a typical transmitting sequence, a character is written into the Transmit Data Register (TDR) if a status read operation indicated the TDR was Empty (TDRE). The write data command (trailing edge of the enable pulse) causes the TDRE status bit to go "low" indicating a transmitter data register full condition. During an idling (absence of data transmission) condition, the transfer of data from the TDR to the transmit shift register will take place within one data bit time. This results in a delay (due to internal operation of the ACIA) in the transmission of the character from the Transmit Data Output with respect to the write data command of one to two data bit times as shown in Figure 2. The trailing edge of the internal transfer signal returns the TDRE status bit to a "high" level indicating a Transmitter Data Register empty condition. The transmitter shift register serializes the data and transmits the data bits, starting with data bit D0, preceded by a start bit and followed by one or two stop bits. Also, internal parity (odd or even) can be optionally added by the ACIA and will occur between the last data bit and the first stop bit.

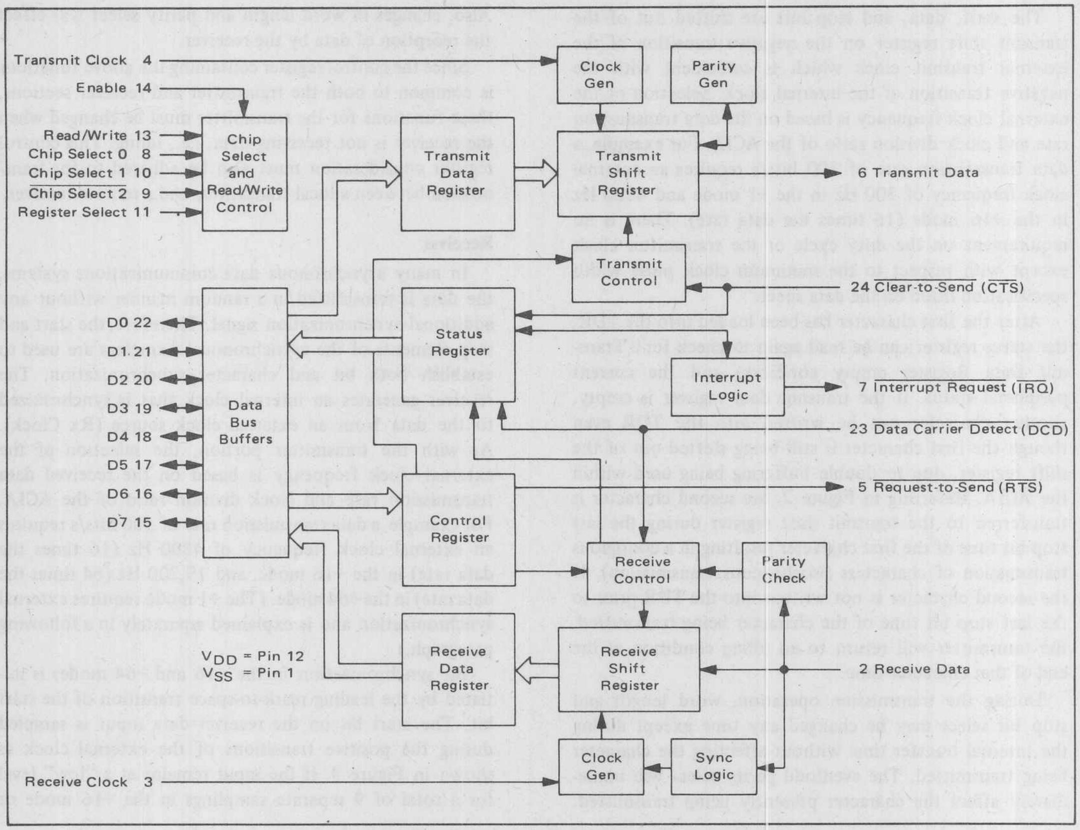


FIGURE 1 — ACIA Block Diagram

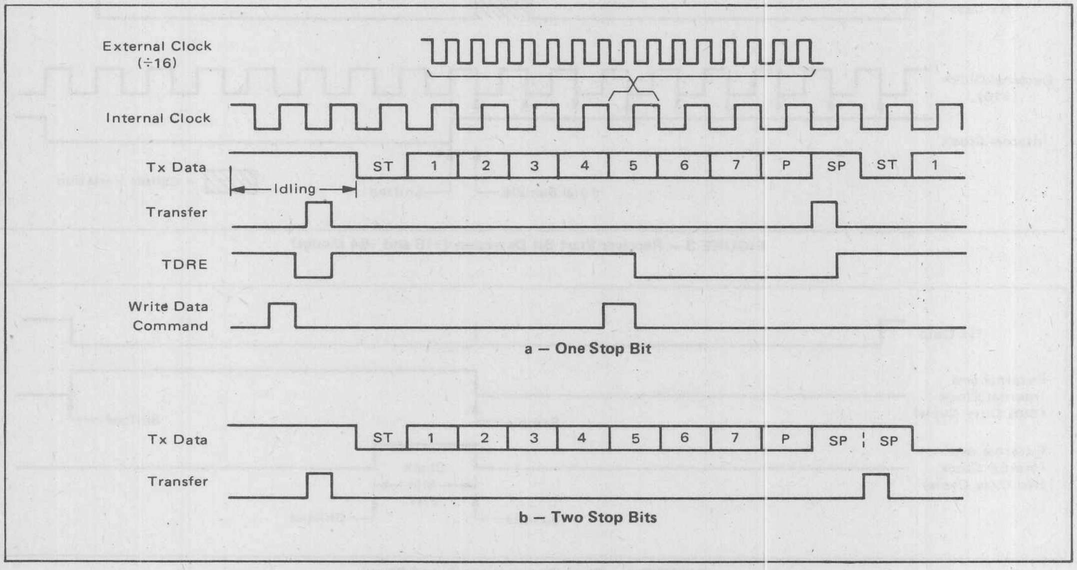


FIGURE 2 — Transmitter Asynchronous Operation

The start, data, and stop bits are shifted out of the transmit shift register on the negative transition of the external transmit clock which is coincident with the negative transition of the internal clock. Selection of the external clock frequency is based on the data transmission rate and clock division ratio of the ACIA. For example, a data transmission rate of 300 bits/s requires an external clock frequency of 300 Hz in the ÷1 mode and 4800 Hz in the ÷16 mode (16 times the data rate). There is no requirement on the duty cycle of the transmitter clock except with respect to the minimum clock pulse width specification listed on the data sheet.

After the first character has been loaded into the TDR, the status register can be read again to check for a Transmit Data Register empty condition and the current peripheral status. If the transmit data register is empty, another character can be written into the TDR even though the first character is still being shifted out of the shift register, due to double buffering being used within the ACIA. Referring to Figure 2, the second character is transferred to the transmit shift register during the last stop bit time of the first character resulting in a contigious transmission of characters (isochronous transmission). If the second character is not written into the TDR prior to the last stop bit time of the character being transmitted, the transmitter will return to an idling condition at the end of that character time.

During the transmission operation, word length and stop bit select may be changed any time except during the internal transfer time without affecting the character being transmitted. The even/odd parity select will immediately affect the character presently being transmitted. Also, changes in word length and parity select will effect the reception of data by the receiver.

Since the control register containing the above functions is common to both the transmitter and receiver sections, these functions for the transmitter must be changed when the receiver is not receiving data, i.e., idling. This control register consideration must also be adhered to for transmission between a local transmitter and a remote receiver.

Receiver

In many asynchronous data communications systems, the data is transmitted in a random manner without any additional synchronization signal. Therefore, the start and stop elements of the asynchronous characters are used to establish both bit and character synchronization. The receiver generates an internal clock that is synchronized to the data from an external clock source (Rx Clock). As with the transmitter portion, the selection of the external clock frequency is based on the received data transmission rate and clock division ratio of the ACIA. For example, a data transmission rate of 300 bits/s requires an external clock frequency of 4800 Hz (16 times the data rate) in the ÷16 mode, and 19,200 Hz (64 times the data rate) in the ÷64 mode. (The ÷1 mode requires external synchronization and is explained separately in a following paragraph.)

Bit synchronization in the ÷16 and ÷64 modes is initiated by the leading mark-to-space transition of the start bit. The start bit on the receiver data input is sampled during the positive transitions of the external clock as shown in Figure 3. If the input remains at a "low" level for a total of 9 separate samplings in the ÷16 mode or

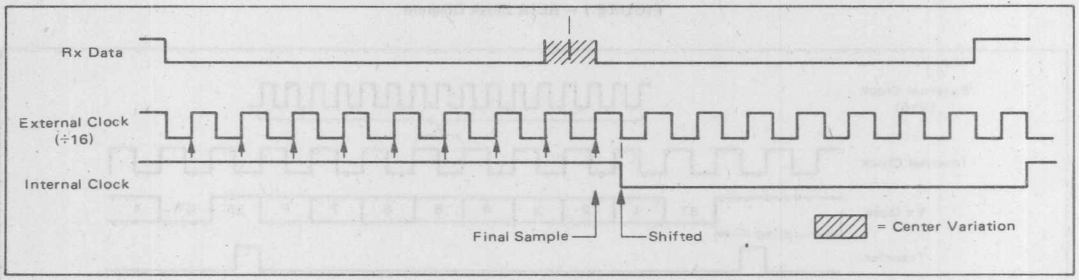


FIGURE 3 – Receiver Start Bit Detection (÷16 and ÷64 Modes)

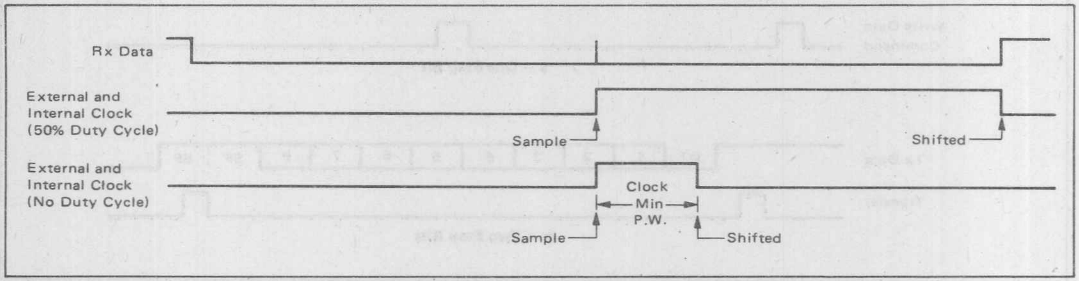

Rx Data
External and Internal Clock (50% Duty Cycle)
Sample
Shifted
External and Internal Clock (No Duty Cycle)
Clock Min P.W.
Sample
Shifted

FIGURE 4 – Clock Requirement for ÷1 Mode

33 samplings in the ÷64 mode, which is equivalent to more than 50% of a bit time, the bit is assumed to be a valid start bit. This start bit is shifted into the shift register on the negative edge of the internal clock. Once a valid start bit has been detected, bit and character synchronization are obtained and the remaining bits are shifted into the shift register at their approximate midpoints.

If the receiver input returns to a mark state during the start bit sampling period, this false start bit is ignored and the receiver resumes looking for the mark-to-space transition of a valid start bit; this technique is referred to as false start bit deletion. The ACIA monitors the start bit on an incremental sampling basis rather than on a continuous sampling basis. This technique is a desirable feature for operation within a noisy environment and stems from the fact that a noise pulse occurring anywhere in a continuous sampling technique would initialize the monitoring logic; whereas in an incremental sampling technique, the noise pulse must occur during the sample to initialize the monitoring logic. The receiver will repeat this process for synchronization of each character in the message.

Divide-by-1 mode selection will not provide internal bit synchronization within the receiver. Therefore, the external receive clock must be synchronized to the data under the following considerations. The sampling of the start bit occurs on the positive edge of the external clock and the start bit is shifted into the shift register on the negative edge of the external clock, as shown in Figure 4. For higher reliability of sampling, the positive transition of the external clock (sampling point) should occur at the approximate midpoint of the bit interval. There is no requirement on the duty cycle of the external receive clock except that the clock must meet the minimum pulse width requirement as noted on the ACIA data sheet.

After the start bit has been detected, the remaining portion of the character being received is checked for parity, framing, and overrun errors. The complete reception of the character produces a "high" on the Receiver Data Register Full (RDRF) status bit, indicating that the receiver data register is full (Figure 5). The received character is transferred to the Receive Data Register (RDR) with the start, stop, and parity bits stripped from the character. At the same time, any receive data errors (parity, overrun, framing) are available in the status register in accordance with the status register definitions. The RDR is oriented such that the first data bit received is available on the D0 output. The receiver is double buffered so that one character may be read from the data register as another character is being received in the shift register. During the reception of data characters, the absence of the first stop bit of the character will not result in the receiver losing character synchronization but will indicate a framing error. The above receive process is repeated for each character in the total message.

POWER-ON RESET/MASTER RESET

The ACIA contains an internal power-on reset circuit to detect the power line turn-on transition and to hold the ACIA in a reset state until initialization is complete to prevent any erroneous output transitions from occurring. In addition to initializing the transmitter and receiver sections, the power-on reset circuit holds the CR5 and CR6 bits of the control register at a logic 0 and logic 1, respectively. When CR5 = 0 and CR6 = 1 as defined by the ACIA data sheet, the $\overline{\text{Request-to-Send}}$ ($\overline{\text{RTS}}$) output is held "high" and an interrupt from the transmitter is disabled. The power-on reset logic is sensitive to the shape

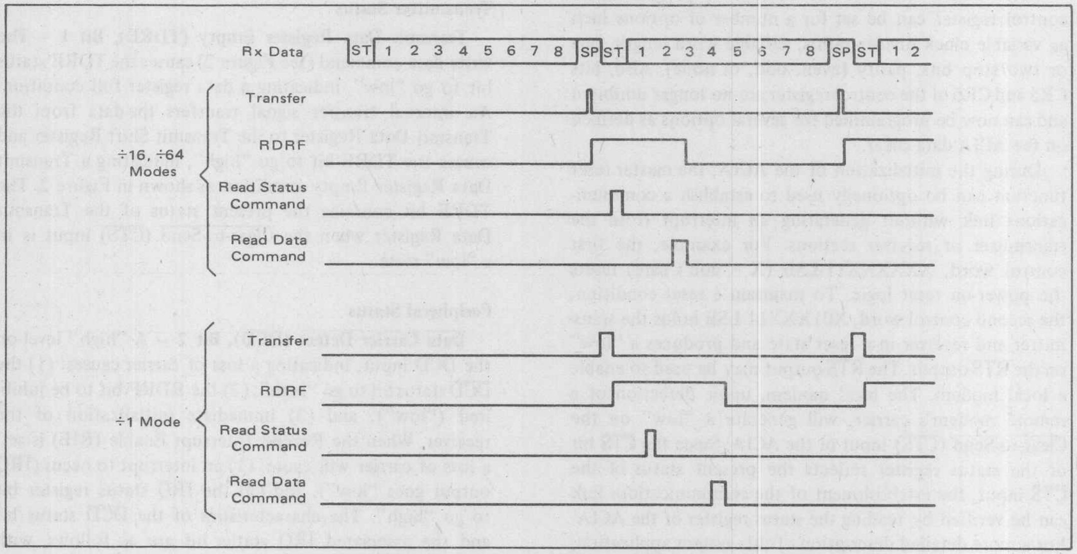


FIGURE 5 — Receiver Asynchronous Operation

TABLE 1 – Reset Functions

	POWER-ON RESET	MASTER RESET (Release Power-On Reset)	MASTER RESET (General)
	b7 b6 b5 b4 b3 b2 b1 b0	b7 b6 b5 b4 b3 b2 b1 b0	b7 b6 b5 b4 b3 b2 b1 b0
Status Register	0 0 0 0 X X 0 0	0 0 0 0 X X 0 0	0 0 0 0 X X 0 0
$\overline{IRQ}$ Output	1	1	1
$\overline{RTS}$ Output	1	1	X
Transmit Break Capability	Inhibit	Inhibit	Optional
Internal: RIE	0	X	X
TIE	0	0	X

Held by Power-On Reset

Defined by Control Register

(X-Independent of Reset function)

of the V_{DD} power supply turn-on transition. To insure correct operation of the reset function, the power turn-on transition must have a positive slope throughout its transition. The conditions of the status register and other outputs during a power-on reset or software master reset are shown in Table I.

The internal ACIA power-on reset logic must be released prior to the transmission of data by performing a software master reset function via the control register. Control Register bits CR0 and CR1 are used to program a master reset condition while the remaining control register bits provide other functions in accordance with the ACIA data sheet. The internal power-on reset logic will inhibit any change in bits CR5 and CR6 of the control register. Therefore, the control word that generates the master reset function clearing the internal power-on reset will not change the $\overline{RTS}$ output or the Internal Transmit Interrupt Enable (TIE), as reflected in Table I. Also, the state of the Receiver Interrupt Enable (RIE) bit of the control register has no external effect because the receiver is initialized by the master reset function.

After master reset of the ACIA, the programmable control register can be set for a number of options such as variable clock divider ratios, variable word length, one or two stop bits, parity (even, odd, or none). Also, bits CR5 and CR6 of the control register are no longer inhibited and can now be programmed for several options as defined on the ACIA data sheet.

During the initialization of the ACIA, the master reset function can be optionally used to establish a communications link without generating an interrupt from the transmitter or receiver sections. For example, the first control word, XXXXXX11-LSB (X = don't care) resets the power-on reset logic. To maintain a reset condition, the second control word, X01XXX11-LSB holds the transmitter and receiver in a reset state and produces a "low" on the $\overline{RTS}$ output. The $\overline{RTS}$ output may be used to enable a local modem. The local modem, upon detection of a remote modem's carrier, will generate a "low" on the $\overline{\text{Clear-to-Send}}$ ($\overline{CTS}$) input of the ACIA. Since the $\overline{CTS}$ bit of the status register reflects the present status of the $\overline{CTS}$ input, the establishment of the communications link can be verified by reading the status register of the ACIA. For a more detailed description of this system application, refer to the system implementation section.

STATUS REGISTER

ACIA status information is available to the MPU through the bus interface by means of the ACIA Status Register. Status information comes from three sources: the receiving section, the transmitting section, and the peripheral status inputs.

Receiver Status

Receive Data Register Full (RDRF), Bit 0 – A logic "high" level on the RDRF bit indicates that data has been transferred to the Receive Data Register and that the received data can be read from the ACIA. Reading the Receive Data Register causes the RDRF status bit to go "low", as shown in Figure 5. A "low" on the $\overline{\text{Data Carrier Detect}}$ ($\overline{DCD}$) input enables the RDRF status bit to be generated from a Receive Data Register full condition. A "high" on the $\overline{DCD}$ input or a master reset condition will force the RDRF status bit to a "low" state until the $\overline{DCD}$ input returns to a "low" state. This is independent of the state of the status register $\overline{DCD}$ bit.

Transmitter Status

Transmit Data Register Empty (TDRE), Bit 1 – The write data command (see Figure 2) causes the TDRE status bit to go "low", indicating a data register full condition. An internal transfer signal transfers the data from the Transmit Data Register to the Transmit Shift Register and causes the TDRE bit to go "high", indicating a Transmit Data Register Empty condition as shown in Figure 2. The TDRE bit contains the present status of the Transmit Data Register when the $\overline{\text{Clear-to-Send}}$ ($\overline{CTS}$) input is in a "low" state.

Peripheral Status

$\overline{\text{Data Carrier Detect}}$ ($\overline{DCD}$), Bit 2 – A "high" level on the $\overline{DCD}$ input, indicating a loss of carrier causes: (1) the $\overline{DCD}$ status bit to go "high"; (2) the RDRF bit to be inhibited ("low"); and (3) immediate initialization of the receiver. When the Receive Interrupt Enable (RIE) is set, a loss of carrier will cause: (1) an interrupt to occur ($\overline{IRQ}$ output goes "low"), and (2) the $\overline{IRQ}$ status register bit to go "high". The characteristics of the $\overline{DCD}$ status bit and the associated $\overline{IRQ}$ status bit are as follows, with reference to the six segments in Figure 6, where each

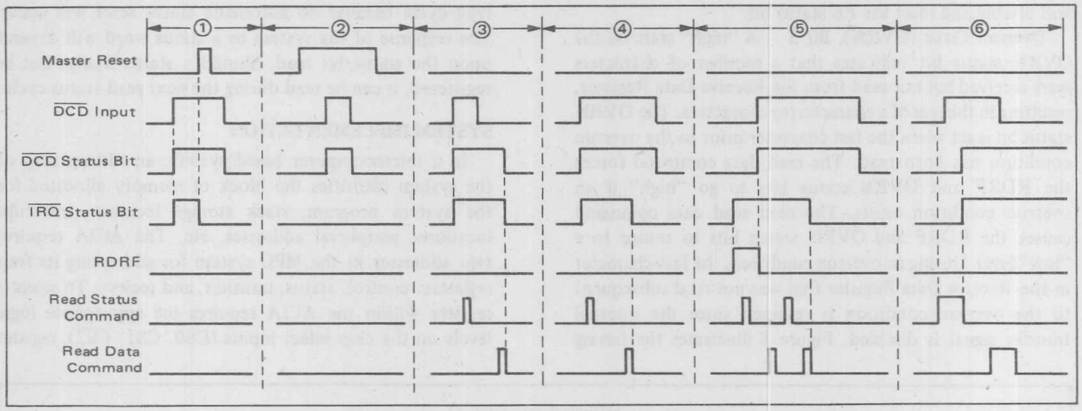

①
②
③
④
⑤
⑥
Master Reset
$\overline{\text{DCD}}$ Input
$\overline{\text{DCD}}$ Status Bit
$\overline{\text{IRQ}}$ Status Bit
RDRF
Read Status Command
Read Data Command

FIGURE 6 – Data Carrier Detect Variations

segment represents a specific condition. (Note: The $\overline{\text{IRQ}}$ output is the inverse logic level of the $\overline{\text{IRQ}}$ status bit.) Segment (1) – A master reset of the ACIA resets the interrupt status bit ($\overline{\text{IRQ}}$) generated by a loss of carrier. Segment (2) – If the $\overline{\text{DCD}}$ input goes "high" during a master reset condition, the $\overline{\text{DCD}}$ status bit will reflect the state of the $\overline{\text{DCD}}$ input. Segment (3) – After an interrupt has occurred from a loss of carrier, the $\overline{\text{IRQ}}$ and $\overline{\text{DCD}}$ status bits (provided the $\overline{\text{DCD}}$ input has returned to a "low" level) are reset by first reading the Status Register and then reading the Data Register. Segment (4) – If the $\overline{\text{DCD}}$ input remains "high" after a read status and a read data, the $\overline{\text{IRQ}}$ bit will be cleared but the $\overline{\text{DCD}}$ status bit remains "high" and will follow the state of the $\overline{\text{DCD}}$ input. Segment (5) – If a read status occurs when the $\overline{\text{DCD}}$ input is "low" followed by a loss of carrier ($\overline{\text{DCD}}$ input goes "high") prior to the read data command, this read data command will not reset either the $\overline{\text{IRQ}}$ or $\overline{\text{DCD}}$ status bits. The next read status followed by a read data will reset the $\overline{\text{IRQ}}$ status bit. Segment (6) – A transition of the $\overline{\text{DCD}}$ input during a read status or read data command is not recognized until the trailing edge of the read command. The $\overline{\text{DCD}}$ input to the ACIA must be tied "low" if it is not used.

Clear-to-Send ($\overline{\text{CTS}}$), Bit 3 – The $\overline{\text{CTS}}$ status bit continuously reflects the state of the $\overline{\text{CTS}}$ input. A "high" on the $\overline{\text{CTS}}$ input will inhibit the TDRE status bit and associated interrupt status bit ($\overline{\text{IRQ}}$). The $\overline{\text{CTS}}$ input has no effect on a character being transmitted from the shift register or the character in the Transmit Data Register (the transmitter is not initialized). Also, the $\overline{\text{CTS}}$ bit is not affected by a master reset. The $\overline{\text{CTS}}$ input to the ACIA must be tied "low" if it is not used.

Framing Error (FE), Bit 4 – A framing error indicates the absence of the first stop bit of a character resulting from a loss of character synchronization, faulty transmission, or a "break" (all spaces) condition. If one of the above conditions is present, the internal receiver transfer signal will cause the FE bit to go "high". The next internal transfer signal will cause the FE status bit to be updated for the error status of the next character, as shown in Figure 7. A "high" on the $\overline{\text{DCD}}$ input or a master reset

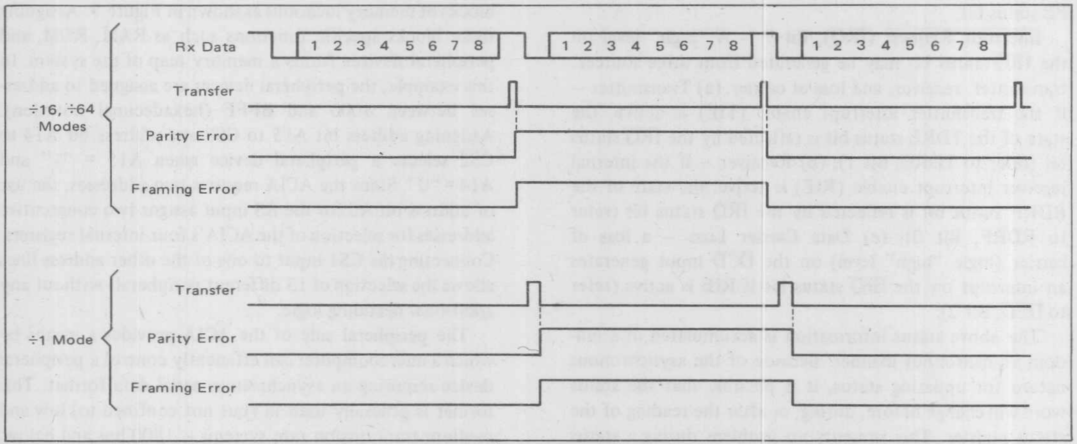


FIGURE 7 – Parity and Framing Errors

will disable and reset the FE status bit.

Overrun Error (OVRN), Bit 5 – A "high" state on the OVRN status bit indicates that a number of characters were received but not read from the Receive Data Register, resulting in the loss of a character/or characters. The OVRN status bit is set when the last character prior to the overrun condition has been read. The read data command forces the RDRF and OVRN status bits to go "high" if an overrun condition exists. The next read data command causes the RDRF and OVRN status bits to return to a "low" level. During an overrun condition, the last character in the Receive Data Register that was not read subsequent to the overrun condition is retained since the internal transfer signal is disabled. Figure 8 illustrates the timing events during an overrun error condition. A "high" state on the $\overline{\text{DCD}}$ input or a master reset disables and resets the OVRN status bit.

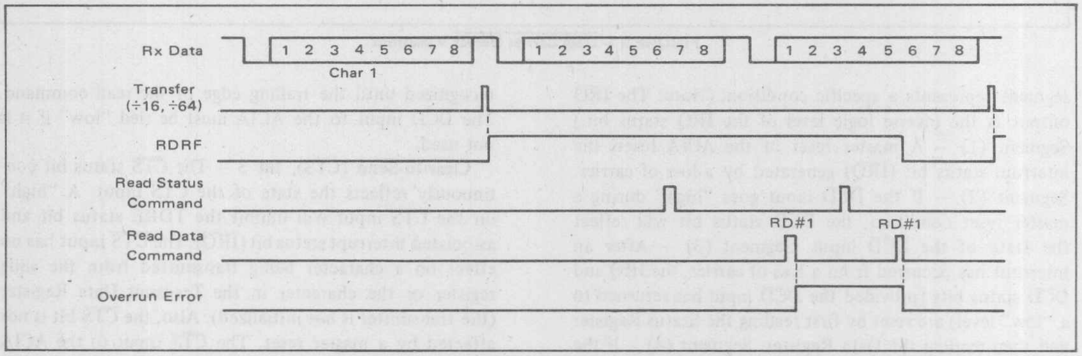


FIGURE 8 – Overrun Error

Parity Error (PE), Bit 6 – If the parity check function is enabled, the internal transfer signal causes the PE status bit to go "high" if a parity error condition exists. The parity error status bit is updated by the next internal transfer signal, as shown in Figure 7. A "high" state on the $\overline{\text{DCD}}$ input or a master reset disables and resets the PE status bit.

Interrupt Request ($\overline{\text{IRQ}}$), Bit 7 – A "high" level on the $\overline{\text{IRQ}}$ status bit may be generated from three sources: transmitter, receiver, and loss of carrier. (a) Transmitter – if the transmitter interrupt enable (TIE) is active, the state of the TDRE status bit is reflected by the $\overline{\text{IRQ}}$ status bit (refer to TDRE, Bit 1); (b) Receiver – if the internal receiver interrupt enable (RIE) is active, the state of the RDRF status bit is reflected by the $\overline{\text{IRQ}}$ status bit (refer to RDRF, Bit 0); (c) Data Carrier Loss – a loss of carrier (logic "high" level) on the $\overline{\text{DCD}}$ input generates an interrupt on the $\overline{\text{IRQ}}$ status bit if RIE is active (refer to $\overline{\text{DCD}}$, Bit 2).

The above status information is accumulated in a random asynchronous manner. Because of the asynchronous nature for updating status, it is possible that the status word will change before, during, or after the reading of the status register. This presents no problem during a status read cycle because no automatic status reset will occur. The response of the system to a status word will depend upon the status bit read. Should a status change not be registered, it can be read during the next read status cycle.

SYSTEM IMPLEMENTATION

In a microcomputer based system, an address map of the system identifies the block of memory allocated for the system program, stack storage location, interrupt locations, peripheral addresses, etc. The ACIA requires two addresses in the MPU system for addressing its four registers: control, status, transmit, and recieve. To select a register within the ACIA requires the appropriate logic levels on the chip select inputs (CS0, CS1, $\overline{\text{CS2}}$), register select input (RS), and read/write control input (R/W). The R/W output line provided by the MPU (MC6800) is used to control writing to and reading from peripheral interface devices or memory. In addition, the R/W control selects one of the read or write registers in the ACIA. A combination of the chip selects and register select inputs can be used to minimize the amount of address decoding logic required for each peripheral. For example, the four Boolean combinations of address lines A14 and A15 select blocks of memory locations as shown in Figure 9. Assigning these blocks specific functions such as RAM, ROM, and peripheral devices forms a memory map of the system. In this example, the peripheral devices are assigned to addresses between 8000 and BFFF (hexadecimal notation). Assigning address bit A15 to CS0 and address bit A14 to $\overline{\text{CS2}}$ selects a peripheral device when A15 = "1" and A14 = "0". Since the ACIA requires two addresses, the use of address bit A0 for the RS input assigns two consecutive addresses for selection of the ACIA's four internal registers. Connecting the CS1 input to one of the other address lines allows the selection of 13 different peripherals without any additional decoding logic.

The peripheral side of the ACIA provides a means by which a microcomputer can efficiently control a peripheral device requiring an asynchronous serial data format. This format is generally used in (but not confined to) low and medium transmission rate systems – 1800 bps and below.

Peripheral	8000 – BFFF	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X
RAM	4000 – 7FFF	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X
RAM	0000 – 3FFF	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X
ACIA #1	8400 – 8401	CS0	$\overline{\text{CS2}}$	0	0	0	CS1	0	0	0	0	0	0	0	0	0	RS
ACIA #2	8020 – 8021	CS0	$\overline{\text{CS2}}$	0	0	0	0	0	0	0	0	CS1	0	0	0	0	RS

X = 1 or 0
CS0 = 1
$\overline{\text{CS2}}$ = 0
CS1 = 1
RS = 1 or 0

FIGURE 9 – Address Specification

A teletype is an example system which has a transmission rate of 110 bps or 10 char/s. An interface device is required between a teletype and an ACIA to convert the TTL compatible levels of the ACIA to the 20 mA current loop of the teletype. A non-complemented data teletype interface requires an optical coupler (4N33) with the addition of logic inverters, as shown in Figure 10. Other teletype options such as RS-232 can be easily interfaced to the ACIA with RS-232 interface devices (MC1488, MC1489), as shown in Figure 11.

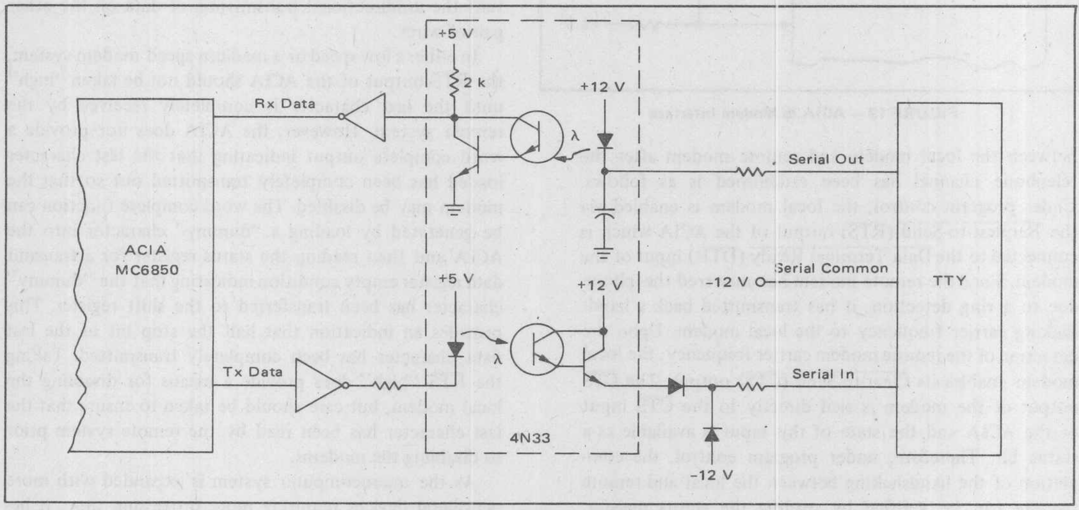


FIGURE 10 – Teletype Interface

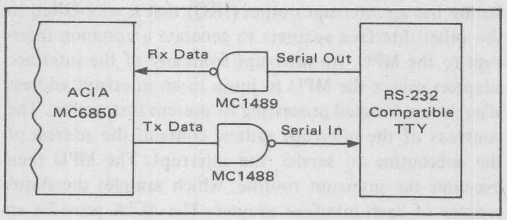


FIGURE 11 – RS-232 Interface

The ACIA provides a means to control a modem for the transmission of data to and from a remote terminal, such as a teletype, over the telephone lines. The modem function can be implemented with a low speed modem, MC6860, as shown in Figure 12. The MC6860 modem uses a frequency shift keying (FSK) technique for the transmission of data up to a maximum data rate of 600 bps. A typical system consists of a local modem and a similar modem at the remote terminal. The local modem converts the digital data from the ACIA to analog form for transmission over the telephone lines. Likewise, analog data received from the remote modem is converted back to digital form by the local modem for use by the microcomputer system via the ACIA. Refer to application notes, AN-731, AN-747 and the MC6860 data sheet for a detailed description of the modem and its operation.

Since the MC6860 does not provide automatic dialing, the telephone channel must be established by manual means or through the use of external automatic dialing equipment. The procedure for accomplishing the handshaking

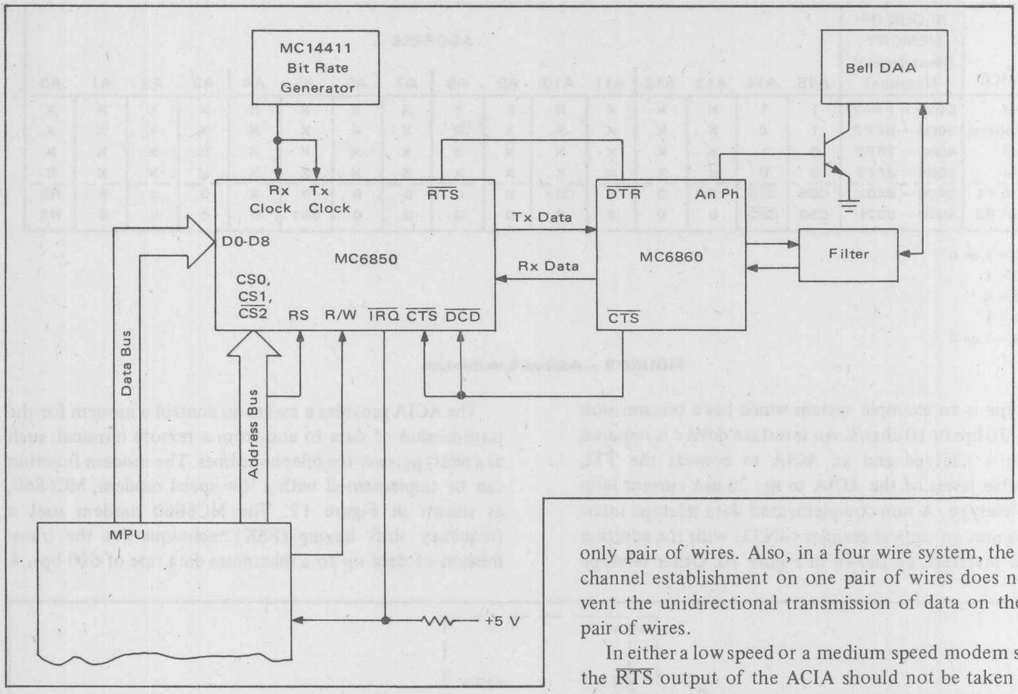


FIGURE 12 – ACIA to Modem Interface

between the local modem and remote modem after the telephone channel has been established is as follows. Under program control, the local modem is enabled via the $\overline{\text{Request-to-Send}}$ ($\overline{RTS}$) output of the ACIA which is connected to the $\overline{\text{Data Terminal Ready}}$ ($\overline{DTR}$) input of the modem. Since the remote modem has answered the phone due to a ring detection, it has transmitted back a handshaking carrier frequency to the local modem. Upon the detection of the remote modem carrier frequency, the local modem enables its $\overline{\text{Clear-to-Send}}$ ($\overline{CTS}$) output. The $\overline{CTS}$ output of the modem is tied directly to the $\overline{CTS}$ input of the ACIA and the state of this input is available as a status bit. Therefore, under program control, the completion of the handshaking between the local and remote modem can be verified by reading the status register. After modem handshaking is completed, data can be transmitted and received over the telephone lines under program control of the microcomputer system. Since a low speed modem such as the MC6860 provides only a $\overline{CTS}$ output, the $\overline{CTS}$ and $\overline{DCD}$ inputs of the ACIA in this example were tied together such that a communications link disconnect could be detectable in either the transmitting or receiving subroutines. The software section of this note provides additional information on the handling of the $\overline{CTS}$ and $\overline{DCD}$ status bits.

Medium speed modem systems may independently utilize both the $\overline{CTS}$ and $\overline{DCD}$ inputs. In a four wire system, the $\overline{CTS}$ input indicates the status of a transmit-only pair of wires and the $\overline{DCD}$ input indicates the status of a receive-only pair of wires. Also, in a four wire system, the loss of channel establishment on one pair of wires does not prevent the unidirectional transmission of data on the other pair of wires.

In either a low speed or a medium speed modem system, the $\overline{RTS}$ output of the ACIA should not be taken "high" until the last character is completely received by the remote system. However, the ACIA does not provide a word complete output indicating that the last character loaded has been completely transmitted out so that the modem may be disabled. The word complete function can be generated by loading a "dummy" character into the ACIA and then reading the status register for a transmit data register empty condition indicating that the "dummy" character has been transferred to the shift register. This provides an indication that half the stop bit of the last data character has been completely transmitted. Taking the $\overline{RTS}$ "high" does provide a means for disabling the local modem, but care should be taken to ensure that the last character has been read by the remote system prior to disabling the modems.

As the microcomputer system is expanded with more peripheral devices requiring more processing time, it becomes increasingly difficult to service each peripheral in the time available. One method to increase the efficiency of the system is to use an interrupt driven system. In an interrupt driven system, each interface adapter of the MPU family has an interrupt output ($\overline{IRQ}$) that is wire-ORed to the other interface adapters to generate a common interrupt to the MPU. An interrupt from any of the interface adapters causes the MPU to jump to an interrupt address after it has finished processing its present instruction. The contents of the interrupt address contains the address of the subroutine to service the interrupt. The MPU then executes the interrupt routine, which samples the status register of each interface adapter. The ACIA provides an $\overline{IRQ}$ status bit that is located in the D7 position of the status register (sign position for numbers) such that only one MPU instruction, BMI (Branch if minus), is required to determine if the transmitter or receiver portion of that particular ACIA was generating the interrupt. Once it is determined that an ACIA is generating the interrupt, the TDRE and RDRF status bits can be checked within their individual subroutines to determine the specific reason for interrupt. The control register can be programmed to inhibit an interrupt from either the transmitter or receiver portions depending on the intended use of the ACIA.

An MC14411 CMOS Bit Rate Generator (BRG), which has 16 standard communication clock rates available, provides a clock source for the ACIA. The receiver and transmitter sections of the ACIA have separate clock inputs to provide independent transmission rates, if desired.

SOFTWARE

Since the internal registers of the ACIA and other MPU interface adapters look like memory locations to the MPU, there is no need for special instructions in the MPU instruction set when using interface adapters. The MPU instructions most commonly used for writing information into the ACIA and reading information out of the ACIA are the store (STA) and load (LDA) instructions, respectively. A store instruction causes the read/write (R/W) output of the MPU to go "low" while a load instruction causes the R/W output to go "high". Assigning consecutive addresses with address bus bit A0 tied to the ACIA Register Select input (RS) along with the R/W input allows access of one of the four ACIA internal registers in accordance with Table II. For example, an STA instruction to address 8400 (hexadecimal notation) performs a write to the ACIA control register whereas an LDA instruction to the same address performs a read of the ACIA status register.

ADDRESS* LOCATION (Hexadecimal Notation)	STA INSTRUCTION (R/W = 0)	LDA INSTRUCTION (R/W = 1)
8400 RS = A0 = 0	Control Register	Status Register
8401 RS = A0 = 1	Transmit Register	Receive Register

*A0 tied to RS

TABLE 2 — ACIA Register Selection

Software Initialization Routine

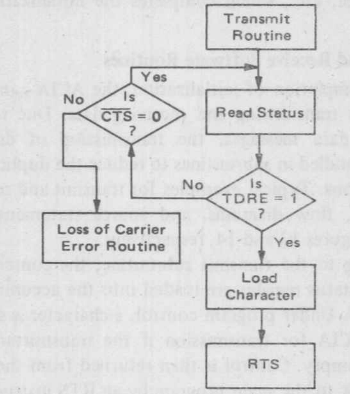
The ACIA must be initialized prior to transmitting and receiving data. During a power-on transition, an internal power-on chip reset (latch) holds the $\overline{\text{IRQ}}$ output "high" to prevent the ACIA from interrupting the MPU or transmitting erroneous information(Ref. Table I). The power-on reset function is released by master resetting the ACIA. A master reset is accomplished by storing a word with bits B0 and B1 equal to "one" into the Control Register. After master resetting, the control register is programmed to set the counter divide ratio, word length, parity, interrupt control, etc., which completes the initialization of the ACIA.

Transmit and Receive Software Routines

After completion of initialization, the ACIA can then be used for transmitting and receiving data. Due to the length of data messages, the transmission of data is normally handled in subroutines to reduce the duplication of instructions. Typical examples for transmit and receive subroutines, flow diagrams, and source statements are shown in Figures 13 and 14, respectively.

Referring to the transmit subroutine, the contents of the ACIA status register are loaded into the accumulator of the MPU. Under program control, a character is stored into the ACIA for transmission if the transmitter data register is empty. Control is then returned from the subroutine back to the main program by an RTS instruction. If the transmitter data register is not empty (TDRE = 0) indicating the transmit data register is full or that the $\overline{\text{CTS}}$ input is "high", inhibiting the TDRE status, the $\overline{\text{CTS}}$ status information which was previously loaded into the accumulator should be checked for its condition. This step is not required when the $\overline{\text{CTS}}$ input is permanently held "low". In a system using a modem, a "high" on the $\overline{\text{CTS}}$ input indicates that the modem data carrier is not present or was lost, requiring the re-establishment of the communications channel. A "low" on the $\overline{\text{CTS}}$ status register bit indicates the TDR is full and allows the status register to be re-read in a loop manner until the TDR becomes empty. When a TDR empty indication occurs, the character stored in the TDR and control is returned to the main program.

Referring to the receive subroutine, there is a similarity of its flow diagram to the transmit routine. The first step in the receive routine is to load the contents of the status register into the accumulator of the MPU. If the receive data register is not full (RDRF = 0), it indicates that the register is empty or that the receiver is inhibited by the $\overline{\text{DCD}}$ input being "high". Therefore, the $\overline{\text{DCD}}$ status bit which was previously loaded into the accumulator should be checked under program control for its condition (this step is not required when the $\overline{\text{DCD}}$ input is permanently held "low"). In a medium speed modem system, a "high" on the $\overline{\text{DCD}}$ input during character reception indicates that the receive carrier was lost and the communications channel would have to be re-established. The $\overline{\text{DCD}}$ status bit is reset back to a "low" state when: (1) the $\overline{\text{DCD}}$ input has returned "low"; (2) by a master reset; or (3) by reading the Receive Data Register after having read the status register. If the $\overline{\text{DCD}}$ status bit is "low", the status is re-read in a loop manner until the receive data register is full. When a logic "1" is read from the RDRF status bit position (B0), indicating that a character was received, the status regarding the framing, overrun, and parity errors of the received character is available. A received character status error could provide for re-transmission of the message, or implement error correction techniques. If there are no errors in the character received, the Receive Data Register is read and control is returned from the



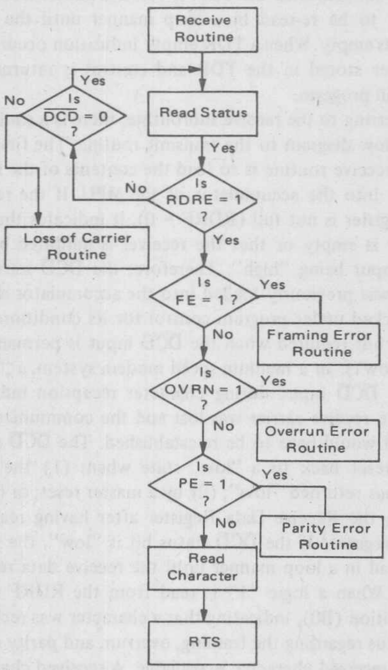
```
NEXT     LDA A STACON    Load Status
         ASR A
         ASR A           Shift TDRE Bit to C-Bit Position
         BCS TX Data     Check TDRE Bit
         ASR A
         ASR A           Shift CTS Bit to C-Bit Position
         BCC NEXT        Check CTS
         BR ERROR 1      Carrier Loss-Branch to Error Routine
TX Data  STA B TXRX      Store Character in ACIA
         RTS             Return from Subroutine
```

FIGURE 13 — Flow Diagram and Source Statements for Transmit Subroutine

subroutine to the main program via an RTS instruction.

In an interrupt driven system, the ACIA can be programmed to generate an interrupt from the transmitter or receiver sections independently. For example, an interrupt from only the transmitter section can be achieved by enabling the transmitter interrupt enable (CR5 = 1, CR6 = 0), and disabling the receiver interrupt enable (CR7 = 0). This results in an interrupt being generated when the Transmit Data Register is empty (TDRE = 1). Therefore, the condition of the Transmit Data Register Empty bit is known and there is no need to examine the condition of this bit as shown in the transmit data subroutine in Figure 13.

As demonstrated in the program examples, only the STA and LDA instructions are required to access one of the four internal registers within the ACIA. However, there are other instructions such as the CMP (compare) instruction that can be used with the ACIA. Since the registers in the ACIA are either write-only or read-only registers, the MPU instructions that perform an automatic rewrite should not be used with the ACIA; this would result in the selection of two registers from one instruction. The MPU instructions that should not be used during ACIA operation are: ASL, ASR, COM, DEC, INC, LSR, NEG, ORA, ROR, and ROL.



```
NEXT 1   LDA A STACON    Load Status
         ASR A           Shift RDRF Bit to C-Bit Position
         BCS FRAM        Check RDRF Bit
         ASR A
         ASR A           Shift DCD Bit to C-Bit Position
         BCC NEXT 1      Check DCD Bit
         BR ERROR 2      Carrier Loss — Branch to Error Routine
FRAM     ASR A
         ASR A           Shift FE Bit to C-Bit Position
         BCC OVRN        Check FE Bit
         BR ERROR 3      Framing Error — Branch to Error Routine
OVRN     ASR A           Shift OVRN Bit to C-Bit Position
         BCC PAR         Check OVRN Bit
         BR ERROR 4      Overrun Error-Branch to Error Routine
PAR      ASR A           Shift PE Bit to C-Bit Position
         BCC R Data      Check PE Bit
         BR ERROR 5      Parity Error-Branch to Error Routine
R DATA   LDA B TXRX      Load B Register with Data
         RTS             Return From Subroutine
```

FIGURE 14 — Flow Diagram and Source Statements for Receive Subroutine

CONCLUSION

The ACIA provides a cost effective approach for adding asynchronous data communications links to computer, minicomputer, and microcomputer systems. The memory-like registers of the ACIA enable a processor to transmit and receive data via the ACIA without the need of special I/O instructions. The ACIA also provides modem I/O control for the transmission of data to remote sites over the telephone network. Included has been a detailed discussion on the ACIA status register along with software program examples, such that the ACIA user can effectively and efficiently apply this part in his data communications system.

AN-757

ANALOG-TO-DIGITAL CONVERSION TECHNIQUES WITH THE M6800 MICROPROCESSOR SYSTEM

Prepared by:
Don Aldridge
Applications Engineering

This application note describes several analog-to-digital conversion systems implemented with the M6800 microprocessor and external linear and digital IC's. Systems consisting of an 8- and 10-bit successive approximation approach, as well as dual ramp techniques of 3½- and 4½-digit BCD and 12-bit binary, are shown with flow diagrams, source programs and hardware schematics. System tradeoffs of the various schemes and programs for binary-to-BCD and BCD-to-7 segment code are discussed.

Analog-To-Digital Conversion Techniques with the M6800 Microprocessor System

INTRODUCTION

The MPU (microprocessing unit) is rapidly replacing both digital and analog circuitry in the industrial control environment. It provides a convenient and efficient method of handling data; controlling valves, motors and relays; and in general, supervising a complete processing machine. However, much of the information required by the MPU for the various computations necessary in the processing system may be available as analog input signals instead of digitally formatted data. These analog signals may be from a pressure transducer, thermistor or other type of sensor. Therefore, for analog data an A/D (analog-to-digital) converter must be added to the MPU system.

Although there are various methods of A/D conversion, each system can usually be divided into two sections – an analog subsystem containing the various analog functions for the A/D and a digital subsystem containing the digital functions. To add an A/D to the MPU, both of the sections may be added externally to the microprocessor in the form of a PC card, hybrid module or monolithic chip. However, only the analog subsystem of the A/D need be added to the microprocessor, since by adding a few instructions to the software, the MPU can perform the function of the digital section of the A/D converter in addition to its other tasks. Therefore, a system design that already contains an MPU and requires analog information needs only one or two additional inexpensive analog components to provide the A/D. The microprocessor software can control the analog section of the A/D, determine the digital value of the analog input from the analog section, and perform various calculations with the resulting data. In addition, the MPU can control several analog A/D sections in a timeshare mode, thus multiplexing the analog information at a digital level.

Using the MPU to perform the tasks of the digital section provides a lower cost approach to the A/D function than adding a complete A/D external to the MPU. The information presented in this note describes this technique as applied to both successive approximation (SA) A/D and dual ramp A/D. With the addition of a DAC (digital-to-analog converter), a couple of operational amplifiers, and the appropriate MPU software, an 8- or 10-bit successive approximation A/D is available. Expansion to greater accuracies is possible by modifying the software and adding the appropriate D/A converter. The technique of successive approximation A/D provides medium speed with accuracies compatible with many systems. The second technique adds an MC1405 dual ramp analog subsystem to the MPU system and, if desired, a digital display to produce a 12-15 bit binary or a 3½- or 4½-digit BCD A/D conversion with 7-segment display readout. This A/D technique has a relatively slow conversion rate but produces a converter of very high accuracy. In addition to the longer conversion time, the MPU must be totally devoted to the A/D function during the conversion period. However, if maximum speed is not required this technique of A/D allows an inexpensive and practical method of handling analog information.

Figure 1 shows the relative merits of each A/D conversion technique. Listed in this table are conversion time, accuracy and whether interrupts to the MPU are allowed during the conversion cycle.

This note describes each method listed in Figure 1 and provides the MPU software and external system hardware schematics along with an explanation of the basic A/D technique and system peculiarities. In addition, the MPU interface connections for the external A/D hardware schemes are shown. These schemes are a complete 8-bit successive approximation and a 3½-digit dual ramp A/D system, both of which externally perform the conversion and transfer the digital data into the MPU system through a PIA.

For additional information on the MC6800 MPU system or A/D systems, the appropriate data sheets or other available literature should be consulted.

MPU

The Motorola microprocessor system devices used are the MC6800 MPU, MCM6810 RAM, MCM6830 ROM and MC6820 PIA (peripheral interface adapter). The following is a brief description of the basic MPU system as it pertains to the A/D systems presented later in this application note.

The Motorola MPU system uses a 16-bit address bus and an 8-bit data bus. The 16-bit address bus provides 65,536 possible memory locations which may be either storage devices (RAM, ROM, etc.) or interface devices (PIA, etc.). The basic MPU contains two 8-bit accumulators, one 16-bit index register, a 16-bit program counter, a 16-bit stack pointer, and an 8-bit condition code register. The condition code register indicates carry, half carry, interrupt, zero, minus, and 2's complement overflow. Figure 2 shows a functional block of the MC6800 MPU.

The MPU uses 72 instructions with six addressing modes which provide 197 different operations in the MPU. A summary of each instruction and function with the appropriate addressing mode is shown in Appendix A of this note.

Characteristic	Successive Approximation 8-Bit Software	Successive Approximation 10-Bit Software	Successive Approximation 8-Bit Hardware	Dual Ramp 12-Bit Software	Dual Ramp 3½-Digit Software	Dual Ramp 4½-Digit Software	Dual Ramp 3½-Digit Hardware
External Hardware	8-Bit DAC Op Amp Comparator	10-Bit DAC Op Amp Comparator	8-Bit DAC SAR* Op Amp Comparator	MC1405	MC1405	MC1405	MC1405 MC14435 MC14558 (for 7-segment display)
Conversion Rate	700 μs Constant	1.25 ms Constant	60 μs for MPU, plus A/D Conversion Time	165 ms (max) Variable	60 ms (max) Variable	600 ms (max) Variable	183 μs (min) for MPU, plus A/D Conversion Time
Interrupt Capability	Allowed	Allowed	Allowed	Not Allowed	Not Allowed	Not Allowed	Allowed
Number of Memory Locations Required (Including PIA Configuration)	106	145	42	84	296	328	58
Serial Output Available	Yes	Yes	Yes	No	No	No	No

*Successive Approximation Register

FIGURE 1 — Relative Merits of A/D Conversion Techniques

The RAMs used in the system are static and contain 128 8-bit words for scratch pad memory while the ROM is mask programmable and contains 1024 8-bit words. The ROM and RAM, along with the remainder of the MPU system components, operate from a single +5 volt power supply; the address bus, data bus and PIAs are TTL compatible.

The MPU system requires a 2ϕ non-overlapping clock with a lower frequency limit of 100 kHz and an upper limit of 1 MHz.

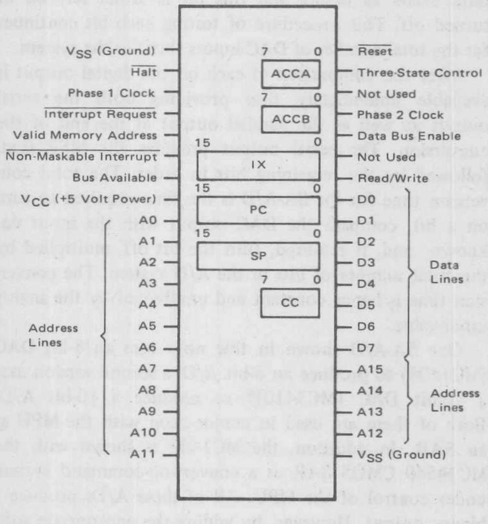


FIGURE 2 — MPU Pin Functions

The PIA is the interface device used between the address and data buses and the analog sections of the A/D. Each PIA contains two essentially identical 8-bit interface ports. These ports (A side, B side) each contain three internal registers that include the data register which is the interface from the data bus to the A/D, the data direction register which programs each of the eight lines of the data register as either an input or an output, and the control register which, in addition to other functions, switches the data bus between the data register and the data direction register. Each port to the PIA contains two addition pins, CA1 and CA2, for interrupt capability and extra I/O lines. The functions of these lines are programmable with the remaining bits in the control register. Figure 3 shows a functional block of the MC6820 PIA.

Each PIA requires four address locations in memory. Two addresses access either of the two (A or B sides) data/data direction registers while the remaining two addresses access either of the two control registers. These addresses are decoded by the chip select and register select lines of the PIA which are connected to the MPU address bus. Selection between the data register and data direction register is made by programming a "1" or "0" in the third least significant bit of each control register . A logic "0" accesses the data direction register while a logic "1" accesses the data register.

By programming "0"s in the data direction register each corresponding line performs as an input, while "1"s in the data direction register make corresponding lines act as outputs. The eight lines may be intermixed between inputs and outputs by programming different combinations of "1"s and "0"s into the data direction register. At the beginning of the program the I/O configuration is programmed into the data direction register, after which the control register is programmed to select the data register for I/O operation.

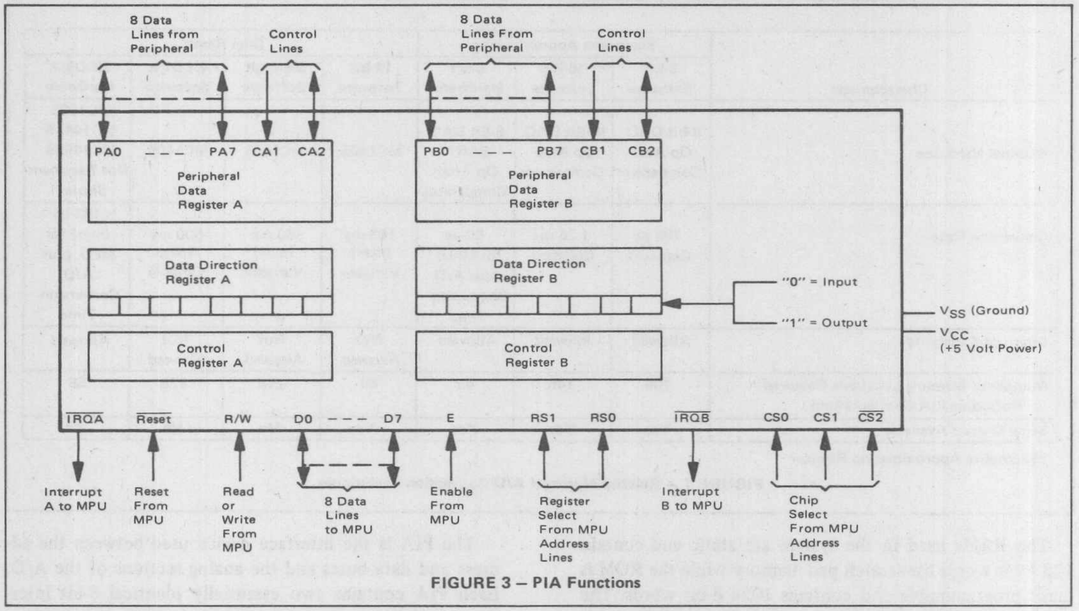


FIGURE 3 – PIA Functions

The printouts shown for each A/D program are the source instructions for the cross assembler from the Motorola timeshare. Since the MPU contains a 16-bit address bus and an 8-bit data bus, the hexadecimal number system provides a convenient representation of these numbers. Although the assembler output is in hexadecimal, the source input may be either binary, octal, decimal or hexadecimal. A dollar sign ($) preceding a number in the source instructions indicates hexadecimal, a percent sign (%) indicates binary and an at sign (@) indicates octal. No prefix indicates the decimal number system.

Only the beginning addresses of the program and labels are shown in the source programs. These beginning addresses may be changed prior to assembling the total system program or the programs may be relocated after assembly with little or no modification.

SUCCESSIVE APPROXIMATION TECHNIQUES

General

One of the more popular methods of A/D conversion is that of successive approximation. This technique uses a DAC (digital-to-analog converter) in a feedback loop to generate a known analog signal to which the unknown analog input is compared. In addition to medium speed conversion rates, it has the advantages of providing not only a parallel digital output after the conversion is completed but also the serial output during the conversion.

Figure 4 shows the block diagram and waveform of the SA-A/D. The DAC inputs are controlled by the successive approximation register (SAR) which is, as presented here, the microprocessor. The DAC output is compared to the analog input (V_{in}) by the analog comparator and its output controls the SAR. At the start of a conversion the MSB of the DAC is turned on by the SAR, producing an output from the DAC equal to half of the full scale value. This output is compared to the analog input and if the DAC output is greater than the input unknown, the SAR turns the MSB off. However, if the DAC output is less than the input unknown, the MSB remains on. Following the trial of the MSB the next most significant bit is turned on and again the comparison is made between the DAC output and the input unknown. The same criteria exists as before and this bit is either left on or turned off. This procedure of testing each bit continues for the total number of DAC inputs (bits) in the system.

After the comparison of each bit the digital output is available immediately thus providing both the serial output as well as the parallel output at the end of the conversion. The serial output provides the MSB first, followed by the remaining bits in order. The total conversion time for the SA-A/D is the time required to turn on a bit, compare the DAC output with the input unknown and, if required, turn the bit off, multiplied by the total number of bits in the A/D system. The conversion time is hence constant and unaffected by the analog input value.

One SA-A/D shown in this note uses an 8-bit DAC (MC1408) to produce an 8-bit A/D; a second version uses a 10-bit DAC (MC3410)* to produce a 10-bit A/D. Both of these are used in conjunction with the MPU as an SAR. In addition, the MC1408 is shown with the MC14549 CMOS SAR as a convert-on-command system under control of the MPU. All of these A/Ds produce a binary output. However, by adding the appropriate software a BCD output or 7-segment-display outputs are available. Also by using a BCD-weighted DAC, the BCD output can be produced directly.

*MC3410 to be announced

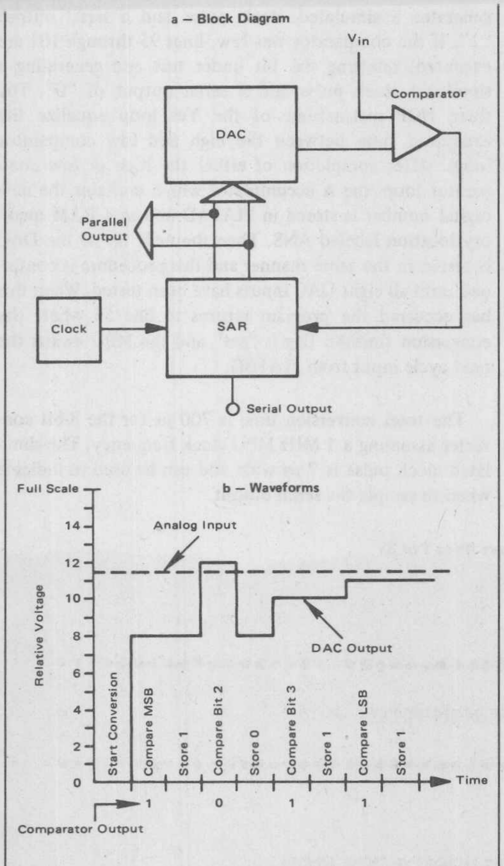


FIGURE 4 – 4-Bit Successive Approximation Converter

8-Bit SA Program

The flow chart for the 8-bit MPU A/D system is shown in Figure 5; Figures 6 and 7 show the software and the hardware external to the microprocessor. The DAC used is the MC1408L-8 which has active high inputs and a current sink output. An uncompensated MLM301A operational amplifier is used as a comparator while an externally compensated MLM301A or internally compensated MC1741 operational amplifier is used as a buffer amplifier for the input voltage. The output voltage compliance of the DAC is ±0.5 volt; if the current required by the D/A does not match that produced from the output of the buffer amplifier through R1 and R2, then the DAC output will saturate at 0.5 volt above or below ground, thus toggling the comparator. The system is calibrated by adjusting R1 for 1 volt full scale, and zero calibration is set by adjusting R3.

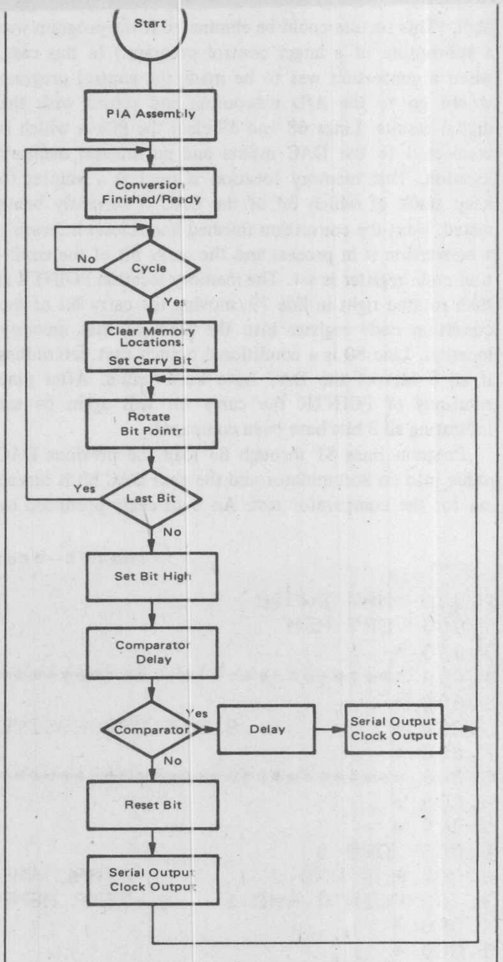

Start
PIA Assembly
Conversion Finished/Ready
No
Cycle
Yes
Clear Memory Locations. Set Carry Bit
Rotate Bit Pointr
Yes
Last Bit
No
Set Bit High
Comparator Delay
Yes
Comparator
Delay
Serial Output Clock Output
No
Reset Bit
Serial Output Clock Output

FIGURE 5 – 8-Bit Successive Approximation A/D Flow Diagram

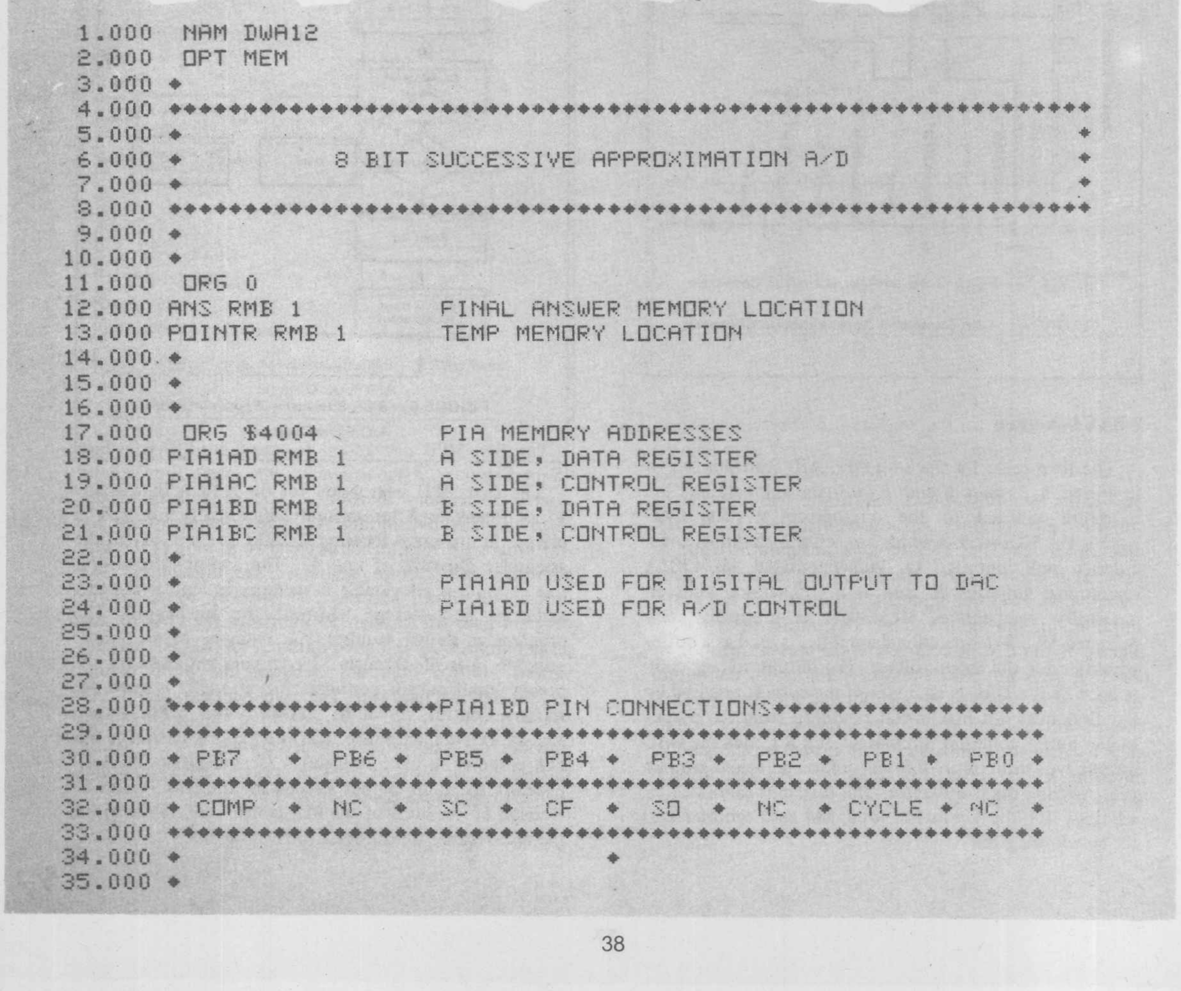
The first MPU instruction for the 8-bit A/D is in line 45 of Figure 6. After assembly, this instruction will be placed in memory location $0A00 as defined in the assembler directive of line 42. The assembled code for this program is relocatable in memory as long as the PIA addresses and storage addresses are unchanged. The program as shown requires 106 memory bytes. Source program lines 45 through 53 configure the PIAs for the proper input/output configuration. PIA1BD is used for various control functions between the MPU system and the external hardware. The exact configuration of this PIA is shown in lines 28 through 33 of Figure 6. PIA1AD provides the 8-bit output needed for the DAC. Lines 51 through 53 set bit 3 of the PIA control register to access the data register for the actual A/D program.

Lines 55 and 56 set the conversion finished flag, which consists of a LED on the hardware schematic, after which the program enters a loop in lines 63-65 which causes the MPU to wait until the cycle input line goes high. (This feature could be eliminated if the program was a subroutine of a larger control program.) In this case, when a conversion was to be made the control program would go to the A/D subroutine and return with the digital results. Lines 68 and 69 clear the PIA-A which is connected to the DAC inputs and an internal memory location. This memory location is used as a pointer to keep track of which bit of the DAC is currently being tested. Next the conversion finished line is reset indicating a conversion is in process and the carry bit of the condition code register is set. The memory location POINTR is then rotated right in line 79, moving the carry bit of the condition code register into the MSB of that memory location. Line 80 is a conditional branch that determines if all 8 bits of the DAC have been tested. After nine rotations of POINTR the carry bit will again be set indicating all 8 bits have been compared.

Program lines 81 through 83 load the previous DAC value into an accumulator and the next DAC bit is turned on for the comparator test. An 8 μs delay produced by the NOP instruction of lines 87 through 90 allows the DAC and comparator to settle to a final value before the comparator test of lines 91 and 92. At this point if the comparator was high the Yes loop is executed, which generates a simulated clock pulse and a serial output "1". If the comparator was low, lines 95 through 101 are executed, resetting the bit under test and generating a simulated clock pulse and a serial output of "0". The three NOP instructions of the Yes loop equalize the execution time between the high and low comparator loops. After completion of either the high or low comparator loop, the A accumulator which contains the new digital number is stored in PIA1AD and in a RAM memory location labeled ANS. Then the next bit of the DAC is tested in the same manner and this procedure is continued until all eight DAC inputs have been tested. When this has occurred the program returns to line 55 where the conversion finished flag is "set" and the MPU awaits the next cycle input from PIA1BD.

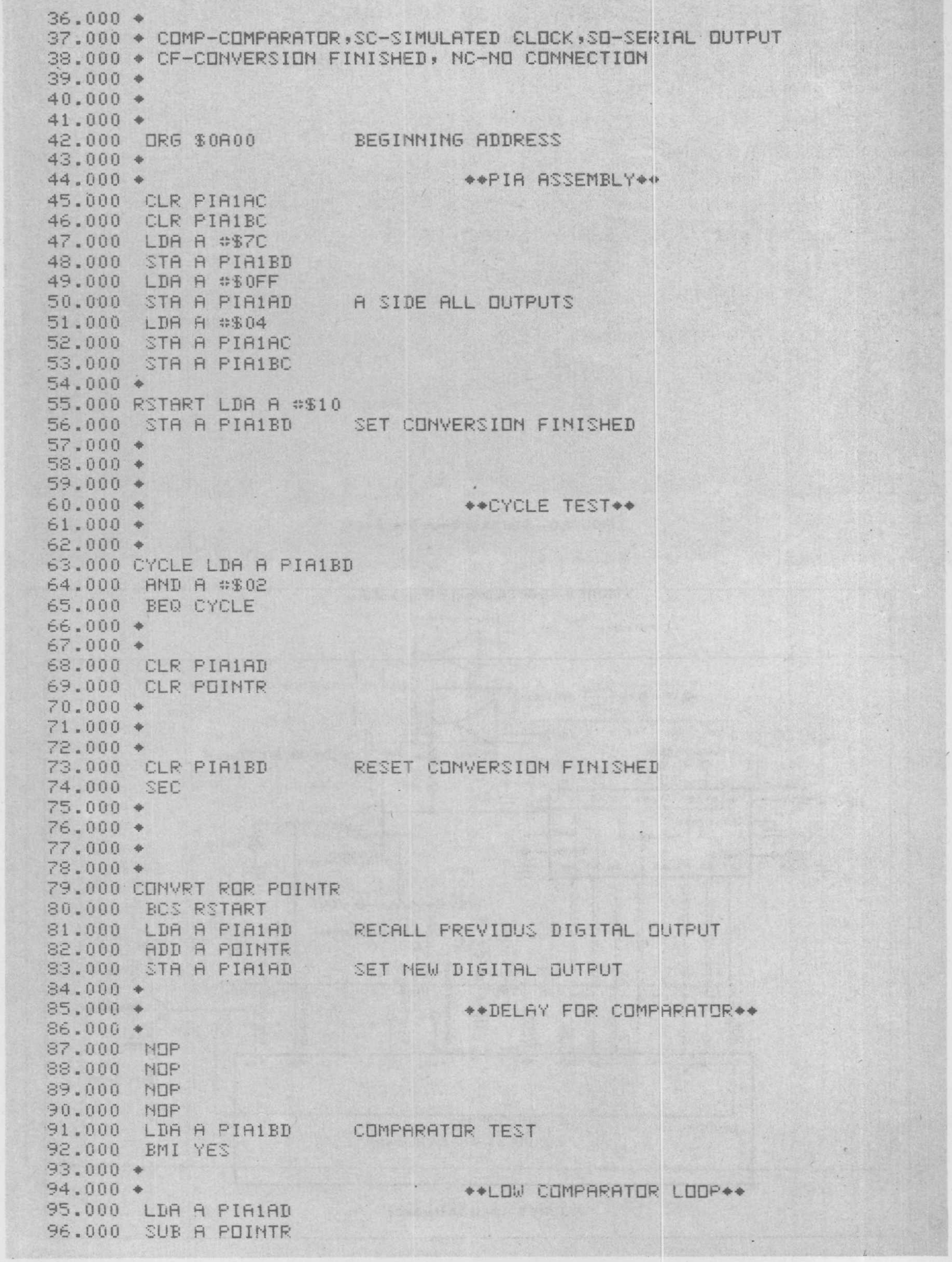
The total conversion time is 700 μs for the 8-bit converter assuming a 1 MHz MPU clock frequency. The simulated clock pulse is 7 μs wide and can be used to indicate when to sample the serial output.

FIGURE 6 — 8-Bit SA Software (Page 1 of 3)

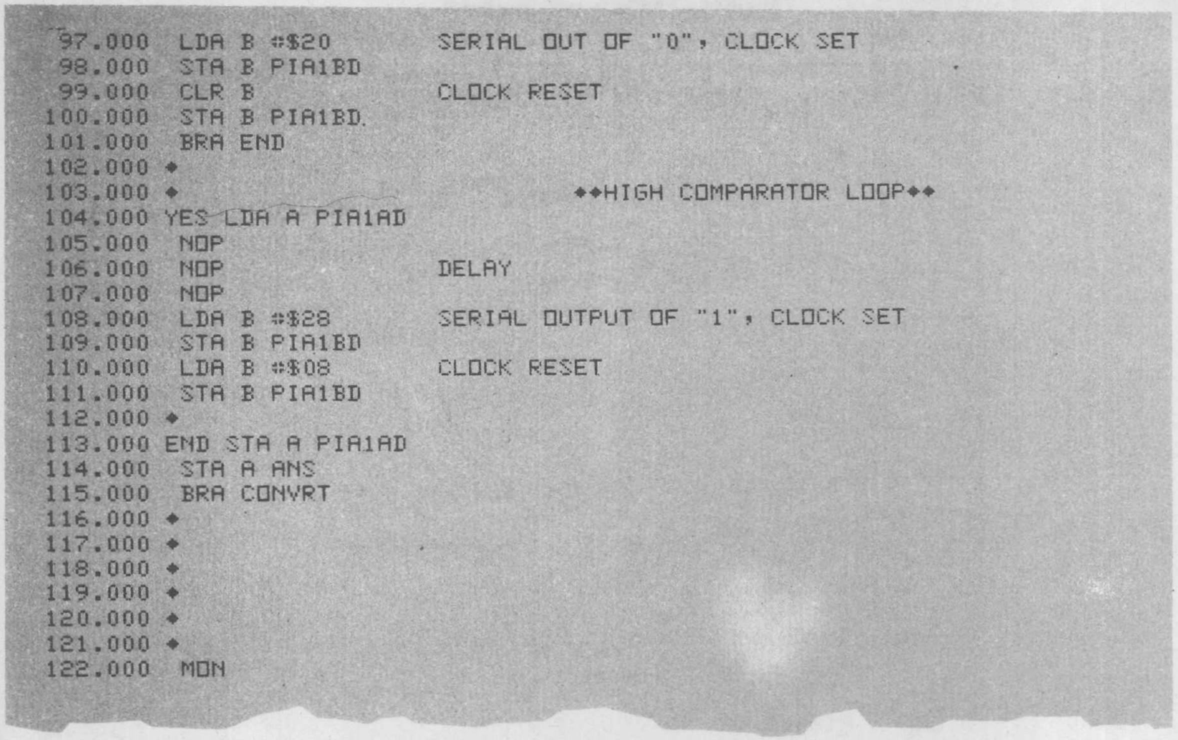


```
 1.000  NAM DWA12
 2.000  OPT MEM
 3.000 *
 4.000 ************************************************************
 5.000 *                                                          *
 6.000 *           8 BIT SUCCESSIVE APPROXIMATION A/D             *
 7.000 *                                                          *
 8.000 ************************************************************
 9.000 *
10.000 *
11.000  ORG 0
12.000 ANS RMB 1           FINAL ANSWER MEMORY LOCATION
13.000 POINTR RMB 1        TEMP MEMORY LOCATION
14.000 *
15.000 *
16.000 *
17.000  ORG $4004          PIA MEMORY ADDRESSES
18.000 PIA1AD RMB 1        A SIDE, DATA REGISTER
19.000 PIA1AC RMB 1        A SIDE, CONTROL REGISTER
20.000 PIA1BD RMB 1        B SIDE, DATA REGISTER
21.000 PIA1BC RMB 1        B SIDE, CONTROL REGISTER
22.000 *
23.000 *                   PIA1AD USED FOR DIGITAL OUTPUT TO DAC
24.000 *                   PIA1BD USED FOR A/D CONTROL
25.000 *
26.000 *
27.000 *
28.000 *****************PIA1BD PIN CONNECTIONS*****************
29.000 *******************************************************
30.000 * PB7   * PB6 * PB5 * PB4 * PB3 * PB2 * PB1 * PB0 *
31.000 *******************************************************
32.000 * COMP  * NC  * SC  * CF  * SO  * NC  * CYCLE * NC  *
33.000 *******************************************************
34.000 *                      *
35.000 *
```

FIGURE 6 — 8-Bit SA Software (Page 2 of 3)



```
*
* COMP-COMPARATOR,SC-SIMULATED CLOCK,SO-SERIAL OUTPUT
* CF-CONVERSION FINISHED, NC-NO CONNECTION
*
*
*
 ORG $0A00        BEGINNING ADDRESS
*
*                          **PIA ASSEMBLY**
 CLR PIA1AC
 CLR PIA1BC
 LDA A #$7C
 STA A PIA1BD
 LDA A #$0FF
 STA A PIA1AD     A SIDE ALL OUTPUTS
 LDA A #$04
 STA A PIA1AC
 STA A PIA1BC
*
RSTART LDA A #$10
 STA A PIA1BD     SET CONVERSION FINISHED
*
*
*
*                          **CYCLE TEST**
*
*
CYCLE LDA A PIA1BD
 AND A #$02
 BEQ CYCLE
*
*
 CLR PIA1AD
 CLR POINTR
*
*
*
 CLR PIA1BD       RESET CONVERSION FINISHED
 SEC
*
*
*
*
CONVRT ROR POINTR
 BCS RSTART
 LDA A PIA1AD     RECALL PREVIOUS DIGITAL OUTPUT
 ADD A POINTR
 STA A PIA1AD     SET NEW DIGITAL OUTPUT
*
*                          **DELAY FOR COMPARATOR**
*
 NOP
 NOP
 NOP
 NOP
 LDA A PIA1BD     COMPARATOR TEST
 BMI YES
*
*                          **LOW COMPARATOR LOOP**
 LDA A PIA1AD
 SUB A POINTR
```



```
 97.000  LDA B #$20         SERIAL OUT OF "0", CLOCK SET
 98.000  STA B PIA1BD
 99.000  CLR B              CLOCK RESET
100.000  STA B PIA1BD.
101.000  BRA END
102.000 *
103.000 *                            **HIGH COMPARATOR LOOP**
104.000 ~~YES LDA A PIA1AD~~
105.000  NOP
106.000  NOP                DELAY
107.000  NOP
108.000  LDA B #$28         SERIAL OUTPUT OF "1", CLOCK SET
109.000  STA B PIA1BD
110.000  LDA B #$08         CLOCK RESET
111.000  STA B PIA1BD
112.000 *
113.000 END STA A PIA1AD
114.000  STA A ANS
115.000  BRA CONVRT
116.000 *
117.000 *
118.000 *
119.000 *
120.000 *
121.000 *
122.000  MON
```

FIGURE 6 — 8-Bit SA Software (Page 3 of 3)

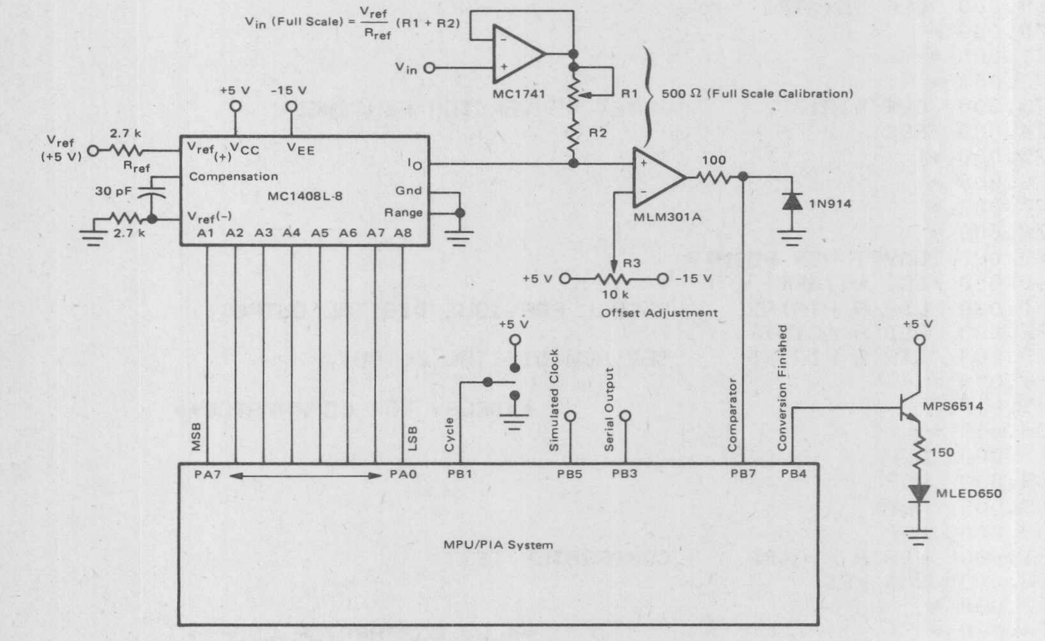
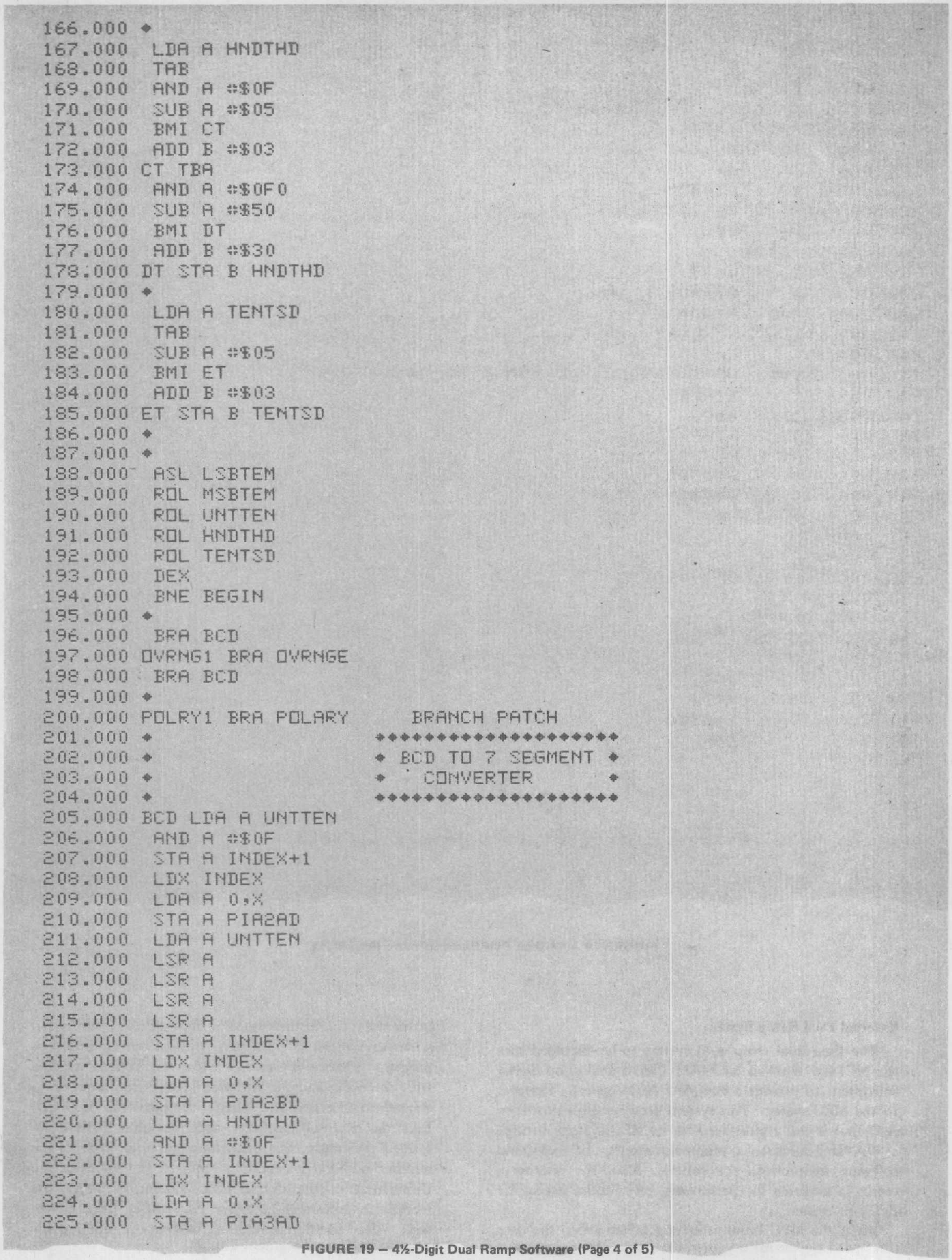
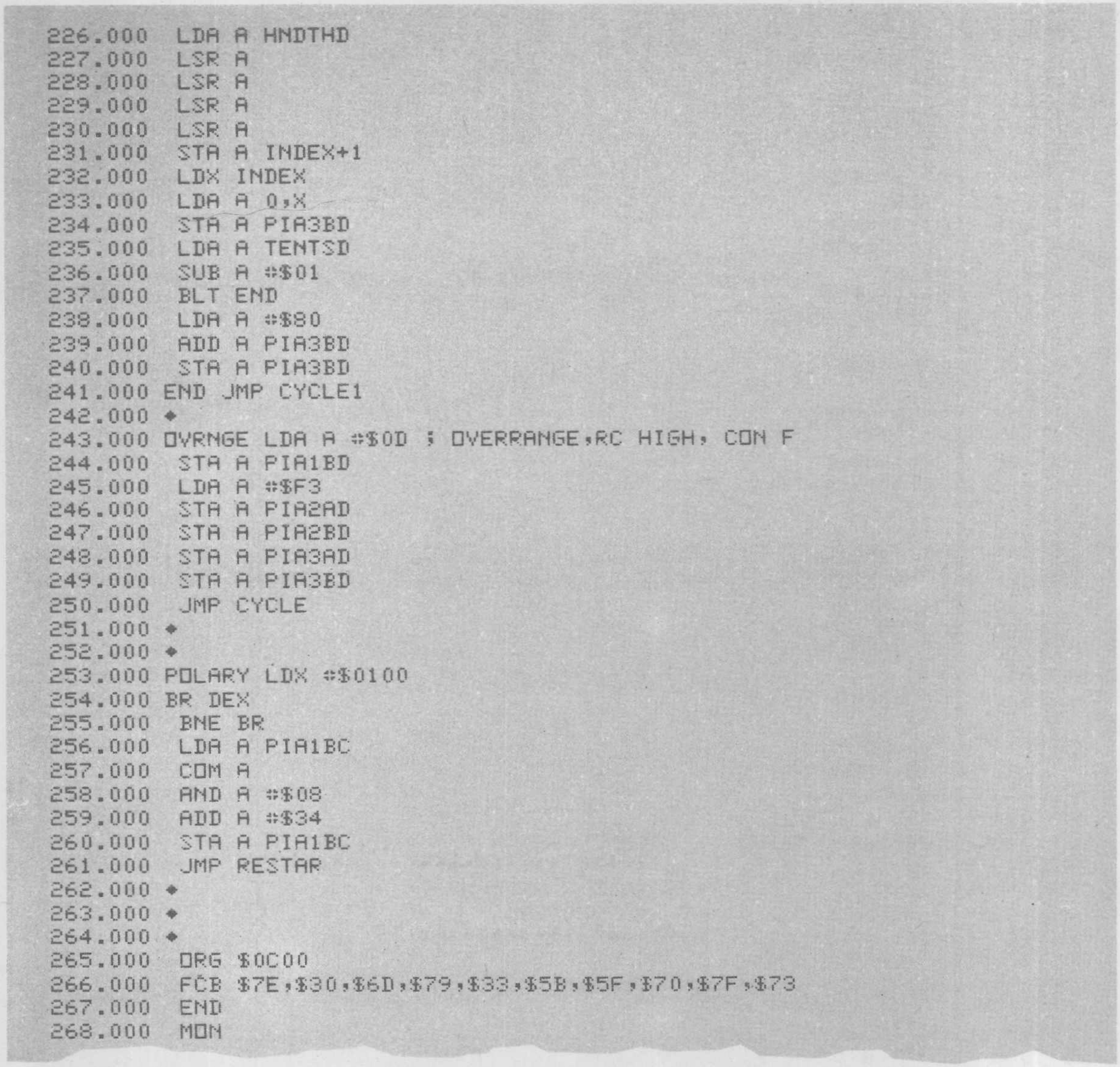


FIGURE 7 — 8-Bit SA Hardware



```
*
 LDA A HNDTHD
 TAB
 AND A #$0F
 SUB A #$05
 BMI CT
 ADD B #$03
CT TBA
 AND A #$0F0
 SUB A #$50
 BMI DT
 ADD B #$30
DT STA B HNDTHD
*
 LDA A TENTSD
 TAB
 SUB A #$05
 BMI ET
 ADD B #$03
ET STA B TENTSD
*
*
 ASL LSBTEM
 ROL MSBTEM
 ROL UNTTEN
 ROL HNDTHD
 ROL TENTSD
 DEX
 BNE BEGIN
*
 BRA BCD
OVRNG1 BRA OVRNGE
 BRA BCD
*
POLRY1 BRA POLARY     BRANCH PATCH
*                  ********************
*                  * BCD TO 7 SEGMENT *
*                  *   CONVERTER      *
*                  ********************
BCD LDA A UNTTEN
 AND A #$0F
 STA A INDEX+1
 LDX INDEX
 LDA A 0,X
 STA A PIA2AD
 LDA A UNTTEN
 LSR A
 LSR A
 LSR A
 LSR A
 STA A INDEX+1
 LDX INDEX
 LDA A 0,X
 STA A PIA2BD
 LDA A HNDTHD
 AND A #$0F
 STA A INDEX+1
 LDX INDEX
 LDA A 0,X
 STA A PIA3AD
```

FIGURE 19 – 4½-Digit Dual Ramp Software (Page 4 of 5)



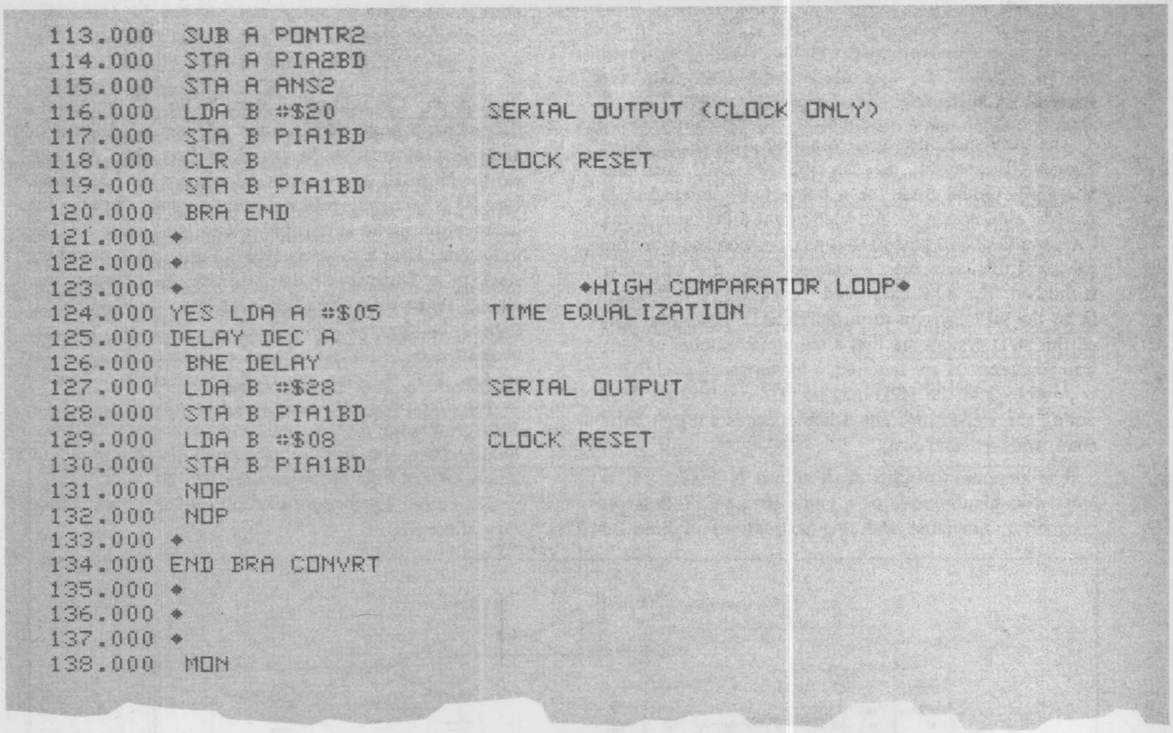
```
 LDA A HNDTHD
 LSR A
 LSR A
 LSR A
 LSR A
 STA A INDEX+1
 LDX INDEX
 LDA A 0,X
 STA A PIA3BD
 LDA A TENTSD
 SUB A #$01
 BLT END
 LDA A #$80
 ADD A PIA3BD
 STA A PIA3BD
END JMP CYCLE1
*
OVRNGE LDA A #$0D ; OVERRANGE,RC HIGH, CON F
 STA A PIA1BD
 LDA A #$F3
 STA A PIA2AD
 STA A PIA2BD
 STA A PIA3AD
 STA A PIA3BD
 JMP CYCLE
*
*
POLARY LDX #$0100
BR DEX
 BNE BR
 LDA A PIA1BC
 COM A
 AND A #$08
 ADD A #$34
 STA A PIA1BC
 JMP RESTAR
*
*
*
 ORG $0C00
 FCB $7E,$30,$6D,$79,$33,$5B,$5F,$70,$7F,$73
 END
 MON
```

FIGURE 19 — 4½-Digit Dual Ramp Software (Page 5 of 5)

External Dual Ramp System

The final dual ramp A/D system to be discussed uses the MC1405 with an MC14435 CMOS dual ramp digital subsystem to provide a complete A/D converter external to the MPU system. This system provides an inexpensive A/D that is easily interfaced to an MPU system through a PIA and requires a minimum number of additional software instructions for control. Also, the microprocessor is available for performing other tasks during the A/D conversion.

When the MPU requires analog information, the data is brought into the MPU system through a PIA and placed in memory for further use. The flow of this information is under control of the MPU system via an interrupt program. Figures 20 and 21 show the external devices with the MPU and the software instructions required to start the conversion and transfer the data from the A/D. Like the external successive approximation method described previously, this dual ramp technique reduces the number of MPU instructions required and increases the throughput of the overall MPU system. However, the increase in exterrnal hardware may offset these advantages. Also, additional external hardware is required for autopolarity and a 7-segment display.



```
 SUB A PONTR2
 STA A PIA2BD
 STA A ANS2
 LDA B #$20           SERIAL OUTPUT (CLOCK ONLY)
 STA B PIA1BD
 CLR B                CLOCK RESET
 STA B PIA1BD
 BRA END
*
*
*                           *HIGH COMPARATOR LOOP*
YES LDA A #$05        TIME EQUALIZATION
DELAY DEC A
 BNE DELAY
 LDA B #$28           SERIAL OUTPUT
 STA B PIA1BD
 LDA B #$08           CLOCK RESET
 STA B PIA1BD
 NOP
 NOP
*
END BRA CONVRT
*
*
*
 MON
```

FIGURE 8 – 10-Bit SA Software (Page 3 of 3)

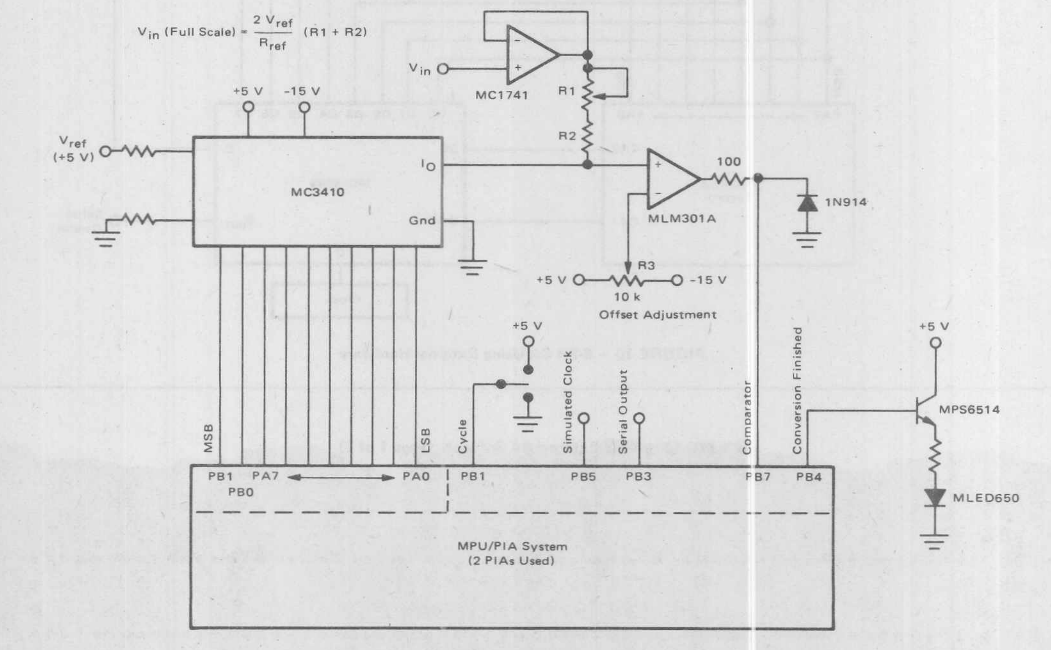


FIGURE 9 – 10-Bit SA Hardware

External SA System

The third successive approximation program, shown in Figures 10 and 11, uses an MC1408 DAC with the MC14549 CMOS SAR for a convert-on-command A/D system. This system is controlled by the MPU through the CA1 and CA2 PIA pins to start a conversion and store the results of this conversion in memory when the conversion is finished. The 8-bit data word from the A/D is brought in to the MPU system through PIA1AD. The advantages of this A/D system are that a minimum number of software instructions are required, a higher speed conversion is possible, and the MPU may be performing other tasks during the conversion. The disadvantage is a higher parts count and increased cost.

The program for this A/D, shown in Figure 11, is written as a subroutine of a larger program. This larger program is simulated with the instructions of lines 28 through 31. The subroutine starts in line 34, unmasking the interrupt input on CA1 and setting CA2 high. (For additional information on use of the CA1 and CA2 lines, see the MC6820 data sheet.) CA2 initiates the conversion. Line 35 is a dummy read statement necessary to clear the data register of the interrupt bit associated with the CA1 input line. Then a wait for interrupt instruction stores the stack in anticipation of the A/D conversion being completed. When the conversion is finished the CA1 line is toggled by the EOC output of the MC14549 and the program goes to line 43 where CA1 is masked and CA2 is set low, thus stopping any further conversion sequences by the A/D. The digital results are loaded into the A accumulator through PIA-A and stored in memory location TEMP. Then the MPU returns from the interrupt and finally returns from the subroutine.

The entire sequence requires 60 μs plus the conversion time of the A/D.

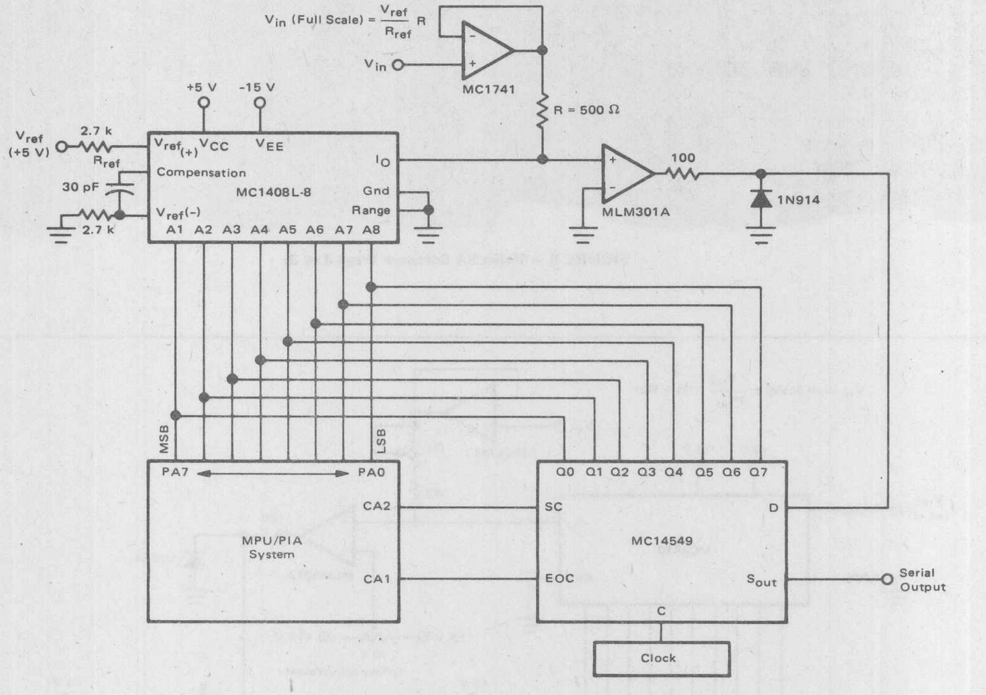
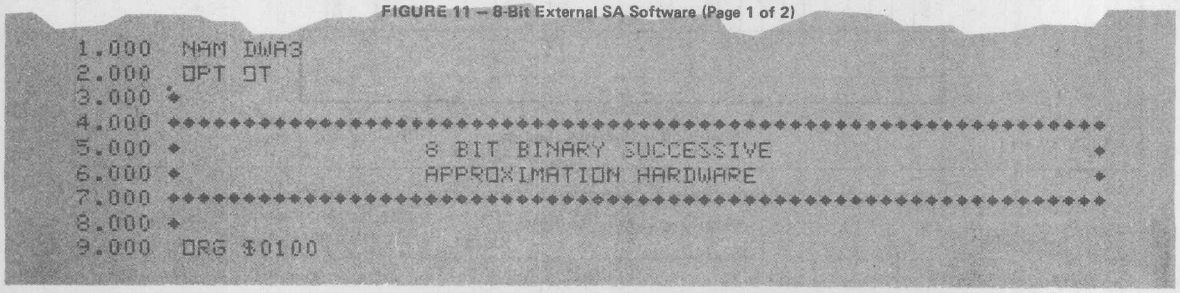
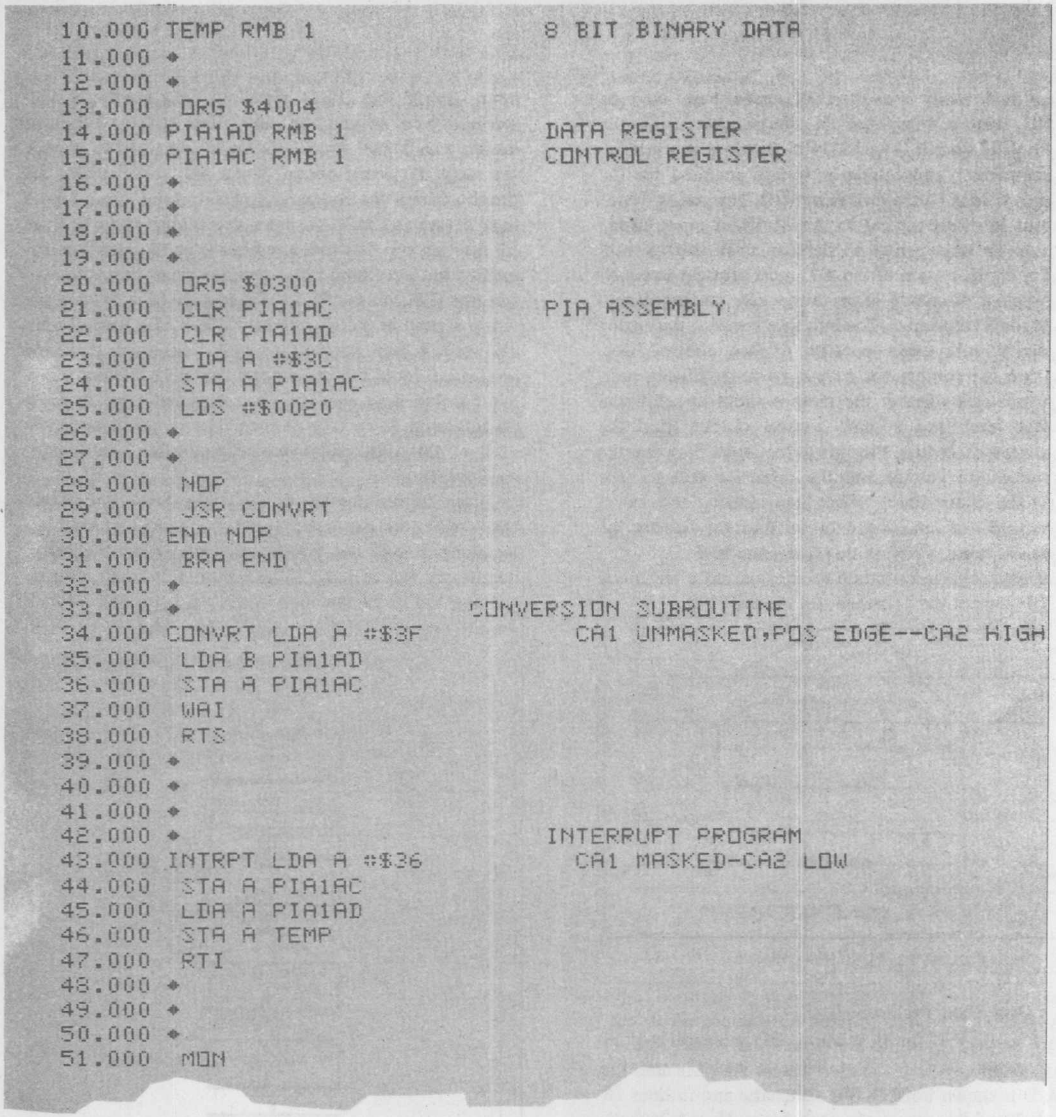


FIGURE 10 – 8-Bit SA Using External Hardware

FIGURE 11 – 8-Bit External SA Software (Page 1 of 2)



```
1.000  NAM DWA3
2.000  OPT OT
3.000 *
4.000 *****************************************************************
5.000 *                  8 BIT BINARY SUCCESSIVE                      *
6.000 *                  APPROXIMATION HARDWARE                       *
7.000 *****************************************************************
8.000 *
9.000  ORG $0100
```



```
TEMP RMB 1                8 BIT BINARY DATA
*
*
 ORG $4004
PIA1AD RMB 1              DATA REGISTER
PIA1AC RMB 1              CONTROL REGISTER
*
*
*
*
 ORG $0300
 CLR PIA1AC               PIA ASSEMBLY
 CLR PIA1AD
 LDA A #$3C
 STA A PIA1AC
 LDS #$0020
*
*
 NOP
 JSR CONVRT
END NOP
 BRA END
*
*                    CONVERSION SUBROUTINE
CONVRT LDA A #$3F           CA1 UNMASKED,POS EDGE--CA2 HIGH
 LDA B PIA1AD
 STA A PIA1AC
 WAI
 RTS
*
*
*
*                         INTERRUPT PROGRAM
INTRPT LDA A #$36           CA1 MASKED-CA2 LOW
 STA A PIA1AC
 LDA A PIA1AD
 STA A TEMP
 RTI
*
*
*
 MON
```

FIGURE 11 — 8-Bit External SA Software (Page 2 of 2)

DUAL RAMP TECHNIQUES

General

Another commonly used method for A/D conversion is the dual ramp or dual slope technique. This approach has a longer conversion time than that of the successive approximation method. The conversion time period is also variable and input voltage dependent. However, this method yields an A/D converter of high accuracy and low cost.

As the name implies the dual ramp method consists of two ramp periods for each conversion cycle. Figure 12 shows the basic waveforms for the dual ramp A/D. The ratio in time of the ramp lengths provides a value representing the difference between a reference and an unknown voltage. During time period T1, the input unknown is integrated for a fixed time period (fixed number of clock cycles). The integrator voltage increases from the reference level to a voltage which is proportional to the input voltage. At the end of this time period a reference voltage is applied to the input of the integrator causing the integrator output voltage to decrease until the reference level is again reached. The number of clock cycles that are required to bring the integrator output voltage back to the reference level is proportional to the input unknown voltage.

The dual ramp converters discussed here use the MC1405 analog subsystem in conjunction with the M6800 MPU system. The MC1405 provides the integrator, comparator and reference voltage required for the analog functions of the dual ramp A/D. The analog device also adds an offset current to the integrator input during the ramp up time period to stabilize small voltage readings. The digital section of the A/D must subtract an equivalent number of counts to produce a zero reading display output for a zero input. The interface between the analog and digital subsystems consists of two control lines. These are the comparator output from the analog part, which indicates whether the ramp is above or below the reference level, and a ramp control output from the digital part to switch the integrator input between the input unknown voltage and the reference voltage. The control of these lines, offset subtraction, and calculations with the resulting data must be handled by the digital subsystem, which in this case is the MPU.

For additional information on the dual ramp technique for A/D, consult the data sheet for the MC1405.

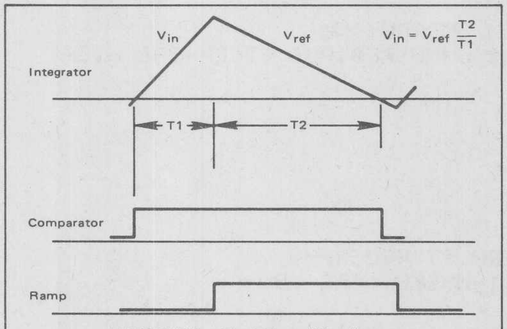


FIGURE 12 – Dual Ramp Waveforms

12-Bit Dual Ramp Program

This version of the dual ramp A/D generates a 12-bit binary output from a 1 volt full scale analog input. Figures 13, 14 and 15 show the flow chart, MPU software and external hardware. The interface of the PIAs used for this A/D is shown both on the schematic and in lines 16 through 22 of the source program. Lines 25 and 26 indicate the two memory locations where the final 12-bit binary result is stored. These locations are $0000 and $0001. The four most significant bits are in location $0000 while the remaining eight bits are in $0001.

Referring to the software of Figure 14, the first instructions (lines 37 through 42) initialize the PIA for its input/output configuration. Source program lines 46 through 49 set the ramp control line of the MC1405 and check the comparator output from the MC1405 to insure that the integrator output is below the reference level at the start of a conversion. Next the "conversion finished" flag is set indicating a conversion ready status. Then the MPU enters a loop (lines 55 through 57) waiting for a cycle input (PB1) from the PIA. When this condition occurs the conversion finished flag is reset while the ramp control line (PB2) goes low, thus starting a conversion cycle. In addition, the index register has been loaded with $2000 which will be decremented to provide the ramp up timing period. When the ramp crosses the threshold level the comparator (PB7) change from low to high causes the MPU to enter the timing cycle of lines 67 through 69. The index register is continuously decremented until reaching zero, at which point the ramp control line (PB2) to the MC1405 is set high (line 74) and the index register is incremented (line 75). This loop continues until the integrator output again reaches the threshold level. Line 76 of the ramp down cycle is a dummy statement included to equalize the timing between the ramp up and ramp down time periods. The proper timing ratio (2:1 in this example) must be maintained for correct A/D operation.

After the termination of the ramp down time period the content of the index register is stored in memory locations $0000 and $0001 (line 82). Next the offset counts are subtracted (512_{10}) from this result by subtracting $01 from memory location $0000. The result is

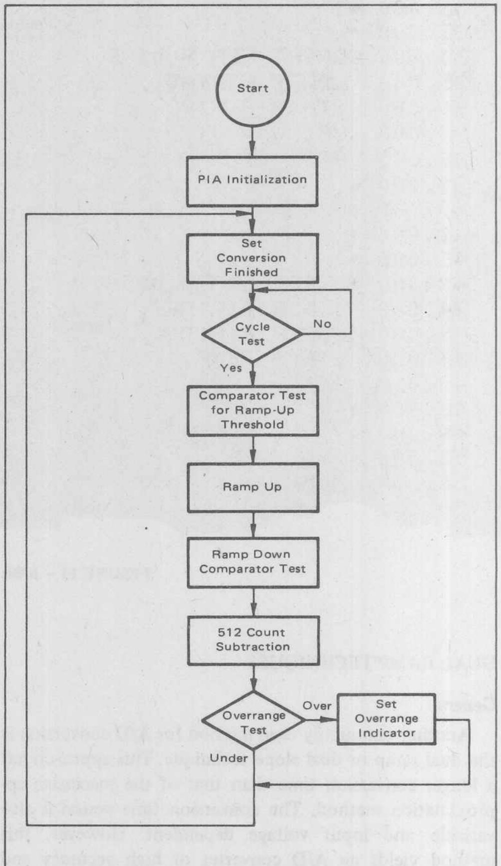


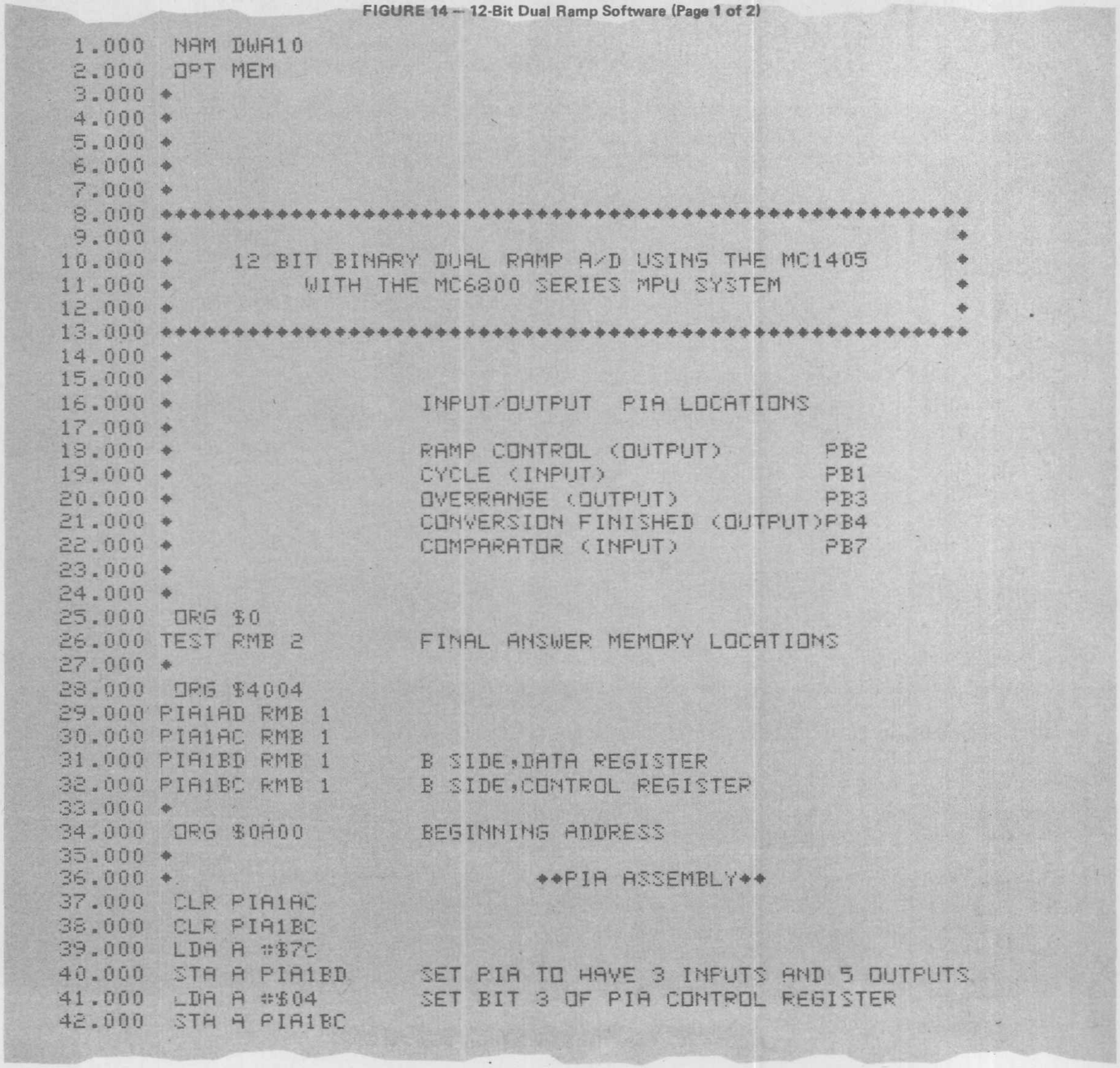
FIGURE 13 – 12-Bit Binary Dual Ramp A/D Flow Diagram

then stored back into the same memory location. Lines 86 and 87 check the contents of memory location TEST for a number greater than 4095_{10}. If this condition occurs, the overrange, conversion finished, and ramp control bits are set high. Otherwise the MPU branches back to line 50 where only the conversion finished and ramp control bits are set high. The program then checks the status of the cycle input waiting for the next conversion.

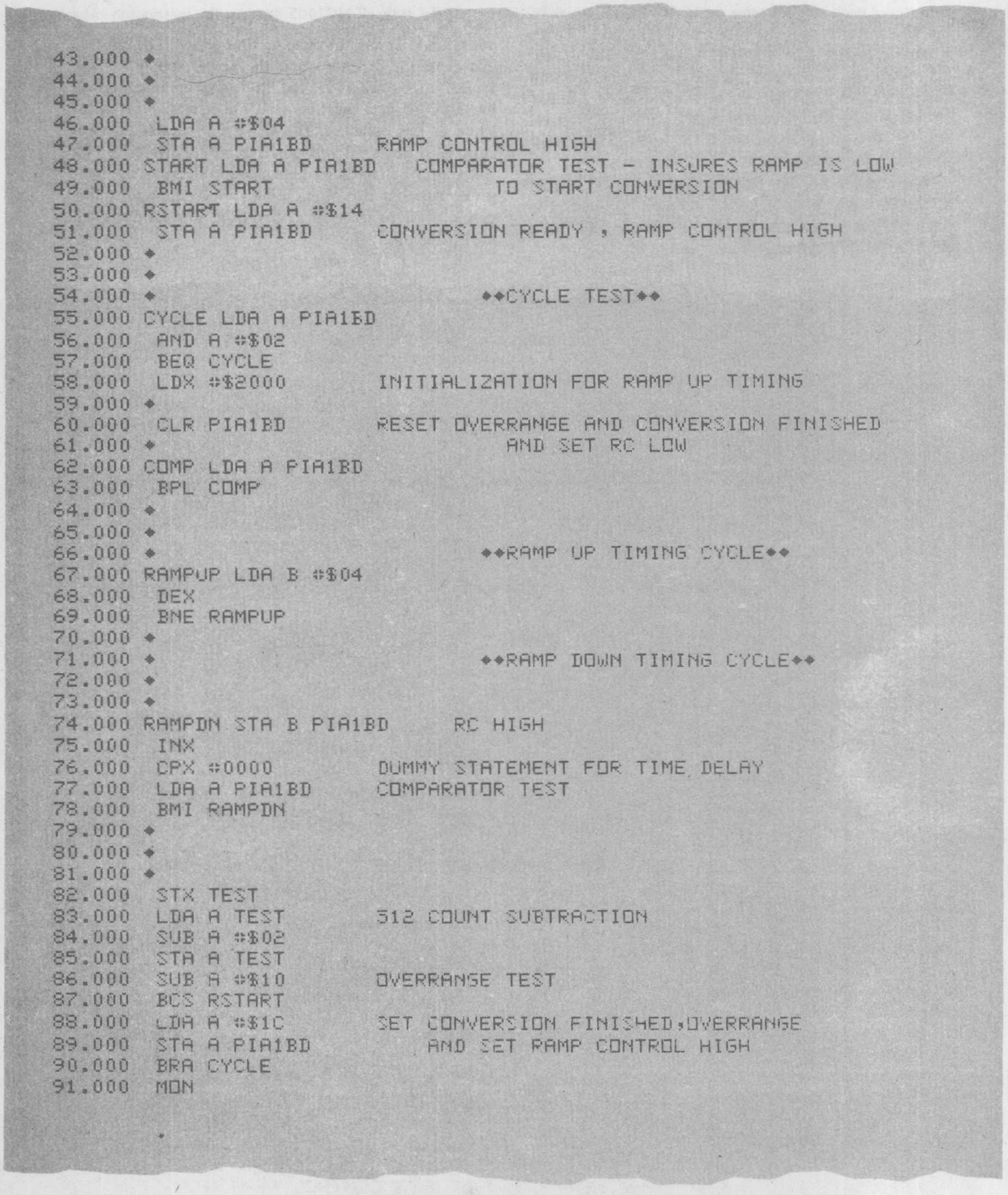
When assembled, the first instruction will be located at $0A00 with 84_{10} memory locations required. The full scale conversion time is 165 ms assuming a 1 MHz clock in the MPU system.

As with all MC1405 designs, the integration capacitor must be large enough to insure that the integrator does not saturate during the ramp up time period. The value of this capacitor depends upon the power supply voltage applied to the MC1405 and the ramp up time period. The MC1405 data sheet contains the equations for calculation of this capacitor. The MC1405 is capable of operating on a single +5 volt power supply; however, a +15 volt supply voltage is recommended to decrease the integrator capacitor size. When using 15 volts the comparator output must be clamped at 5 volts to prevent damaging the PIA inputs.

FIGURE 14 — 12-Bit Dual Ramp Software (Page 1 of 2)



```
  1.000  NAM DWA10
  2.000  OPT MEM
  3.000 *
  4.000 *
  5.000 *
  6.000 *
  7.000 *
  8.000 ********************************************************
  9.000 *                                                      *
 10.000 *     12 BIT BINARY DUAL RAMP A/D USING THE MC1405     *
 11.000 *          WITH THE MC6800 SERIES MPU SYSTEM           *
 12.000 *                                                      *
 13.000 ********************************************************
 14.000 *
 15.000 *
 16.000 *                  INPUT/OUTPUT  PIA LOCATIONS
 17.000 *
 18.000 *                  RAMP CONTROL (OUTPUT)        PB2
 19.000 *                  CYCLE (INPUT)                PB1
 20.000 *                  OVERRANGE (OUTPUT)           PB3
 21.000 *                  CONVERSION FINISHED (OUTPUT)PB4
 22.000 *                  COMPARATOR (INPUT)           PB7
 23.000 *
 24.000 *
 25.000  ORG $0
 26.000 TEST RMB 2         FINAL ANSWER MEMORY LOCATIONS
 27.000 *
 28.000  ORG $4004
 29.000 PIA1AD RMB 1
 30.000 PIA1AC RMB 1
 31.000 PIA1BD RMB 1       B SIDE,DATA REGISTER
 32.000 PIA1BC RMB 1       B SIDE,CONTROL REGISTER
 33.000 *
 34.000  ORG $0A00         BEGINNING ADDRESS
 35.000 *
 36.000 *                          **PIA ASSEMBLY**
 37.000  CLR PIA1AC
 38.000  CLR PIA1BC
 39.000  LDA A #$7C
 40.000  STA A PIA1BD      SET PIA TO HAVE 3 INPUTS AND 5 OUTPUTS
 41.000  LDA A #$04        SET BIT 3 OF PIA CONTROL REGISTER
 42.000  STA A PIA1BC
```



```
43.000 *
44.000 *
45.000 *
46.000  LDA A #$04
47.000  STA A PIA1BD     RAMP CONTROL HIGH
48.000 START LDA A PIA1BD   COMPARATOR TEST - INSURES RAMP IS LOW
49.000  BMI START                 TO START CONVERSION
50.000 RSTART LDA A #$14
51.000  STA A PIA1BD     CONVERSION READY , RAMP CONTROL HIGH
52.000 *
53.000 *
54.000 *                         **CYCLE TEST**
55.000 CYCLE LDA A PIA1BD
56.000  AND A #$02
57.000  BEQ CYCLE
58.000  LDX #$2000       INITIALIZATION FOR RAMP UP TIMING
59.000 *
60.000  CLR PIA1BD       RESET OVERRANGE AND CONVERSION FINISHED
61.000 *                           AND SET RC LOW
62.000 COMP LDA A PIA1BD
63.000  BPL COMP
64.000 *
65.000 *
66.000 *                         **RAMP UP TIMING CYCLE**
67.000 RAMPUP LDA B #$04
68.000  DEX
69.000  BNE RAMPUP
70.000 *
71.000 *                         **RAMP DOWN TIMING CYCLE**
72.000 *
73.000 *
74.000 RAMPDN STA B PIA1BD     RC HIGH
75.000  INX
76.000  CPX #0000        DUMMY STATEMENT FOR TIME DELAY
77.000  LDA A PIA1BD     COMPARATOR TEST
78.000  BMI RAMPDN
79.000 *
80.000 *
81.000 *
82.000  STX TEST
83.000  LDA A TEST       512 COUNT SUBTRACTION
84.000  SUB A #$02
85.000  STA A TEST
86.000  SUB A #$10       OVERRANSE TEST
87.000  BCS RSTART
88.000  LDA A #$1C       SET CONVERSION FINISHED,OVERRANGE
89.000  STA A PIA1BD         AND SET RAMP CONTROL HIGH
90.000  BRA CYCLE
91.000  MON

        *
```

FIGURE 14 – 12-Bit Dual Ramp Software (Page 2 of 2)

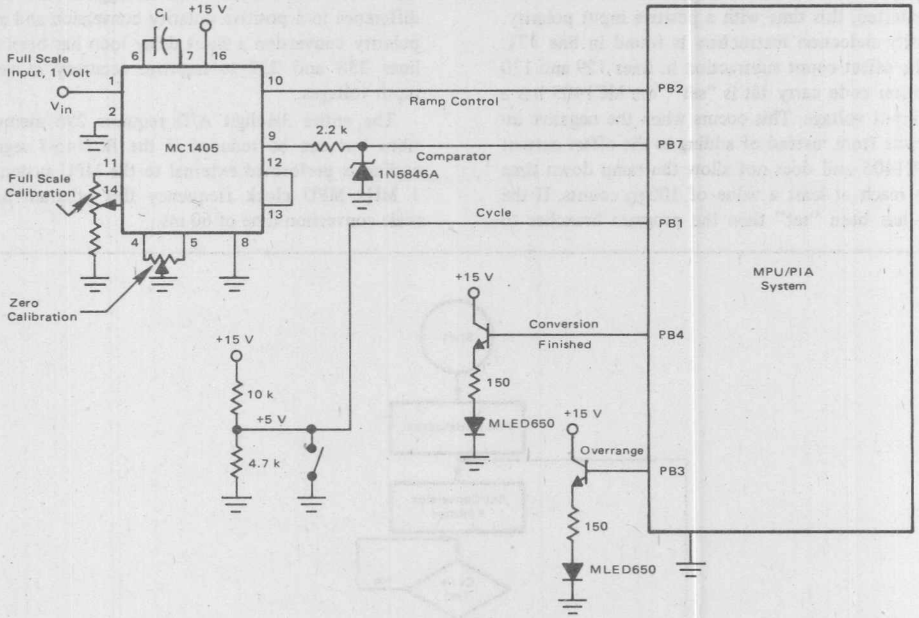


FIGURE 15 – 12-Bit Binary Dual Ramp A/D Hardware

3½-Digit Dual Ramp Program

The flow chart, source program and hardware for a 3½-digit system are shown in Figures 16, 17, and 18 respectively. Referring to Figure 17, the basic conversion routine of lines 96 through 135 in this program is similar to that of the previously discussed 12-bit binary system. The initialization of the index register in line 108 has been changed to increase the ramp up time period. The basic conversion results in a binary number as did the 12-bit version previously discussed. This binary result is converted by the software routine in lines 144 through 180 to produce 3½-digit BCD output. This routine converts up to a 16-bit binary number to the equivalent BCD value. Also the BCD result is converted to a 7-segment display code for use in a LED or LCD readout system. Another feature of the 3½-digit A/D program shown here is a polarity detection scheme. This allows the A/D to handle both positive and negative input voltages.

The external hardware for the 3½-digit A/D requires two full PIAs; one of the four ports is used for interface to the MC1405, cycle input, overrange flag, etc. An I/O configuration similar to that of the 12-bit binary A/D is used. The remaining three ports of the PIAs are used for the 3½-digit display, as shown in Figure 18b.

The conversion initially produces a binary result which is stored in memory locations MSB and MSB+1. This result has 100_{10} offset counts subtracted, and then a polarity check is made. If the polarity that is currently being applied to the input of the MC1405 is positive, the binary number is converted to a BCD number. The technique used for binary-to-BCD conversion is described in Appendix B. The BCD results are stored in memory locations UNTTEN and HNDTHD. Each of these memory locations contains two BCD words. Following the conversion, an overrange test is made in lines 183 through 186 which checks for a maximum of a BCD "1" in the upper four bits of memory location HNDTHD. If an overrange condition occurs, the program branches to lines 227 through 234 where a 1999_{10} is placed in the display and the overrange flag in PIA1BD is "set".

After the overrange test the BCD code is converted to a 7-segment code and stored in the memory location for each PIA port. Segments A through G use PIA outputs 0 through 6 while the half digit output uses PIA2BD output PB7. The conversion technique for BCD-to-7 segment utilizes a look-up table in line 251 with the indexed mode of addressing to access the table. Each of the three full BCD digits is converted to the 7-segment code by first separating the lower BCD and upper BCD word and using the BCD code as the least significant byte of a two byte address for the look-up table. This address is then loaded into the index register and used to locate the corresponding 7-segment code. In the case of the upper BCD digit of each BCD, the memory must be shifted left four times for correct addressing of the look-up table. Finally, the half digit output is added to PIA2BD in lines 197 through 226.

Should the MC1405 have the incorrect polarity on its input, a polarity reversing relay is operated by toggling the

CA2 output of PIA1BC control register. Then the conversion is restarted, this time with a positive input polarity. The polarity detection instruction is found in line 131. If after the offset count subtraction in lines 129 and 130 the condition code carry bit is "set", the MC1405 has a negative input voltage. This occurs when the negative input subtracts from instead of adding to the offset current in the MC1405 and does not allow the ramp down time period to reach at least a value of 100_{10} counts. If the carry bit has been "set" then the program branches to line 236 where the CA2 line is toggled. Also due to the difference in a positive polarity conversion and a negative polarity conversion a short delay loop has been added in lines 238 and 239 to improve accuracy at very small input voltages.

The entire 3½-digit A/D requires 296 memory locations but can be reduced if the BCD-to-7 segment decoding is performed external to the MPU system. With a 1 MHz MPU clock frequency this program has a full scale conversion time of 60 ms.

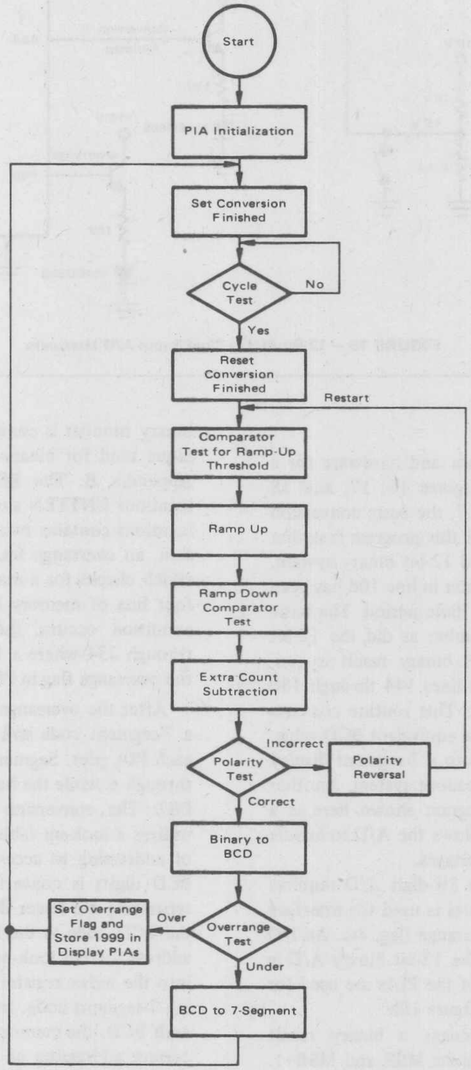
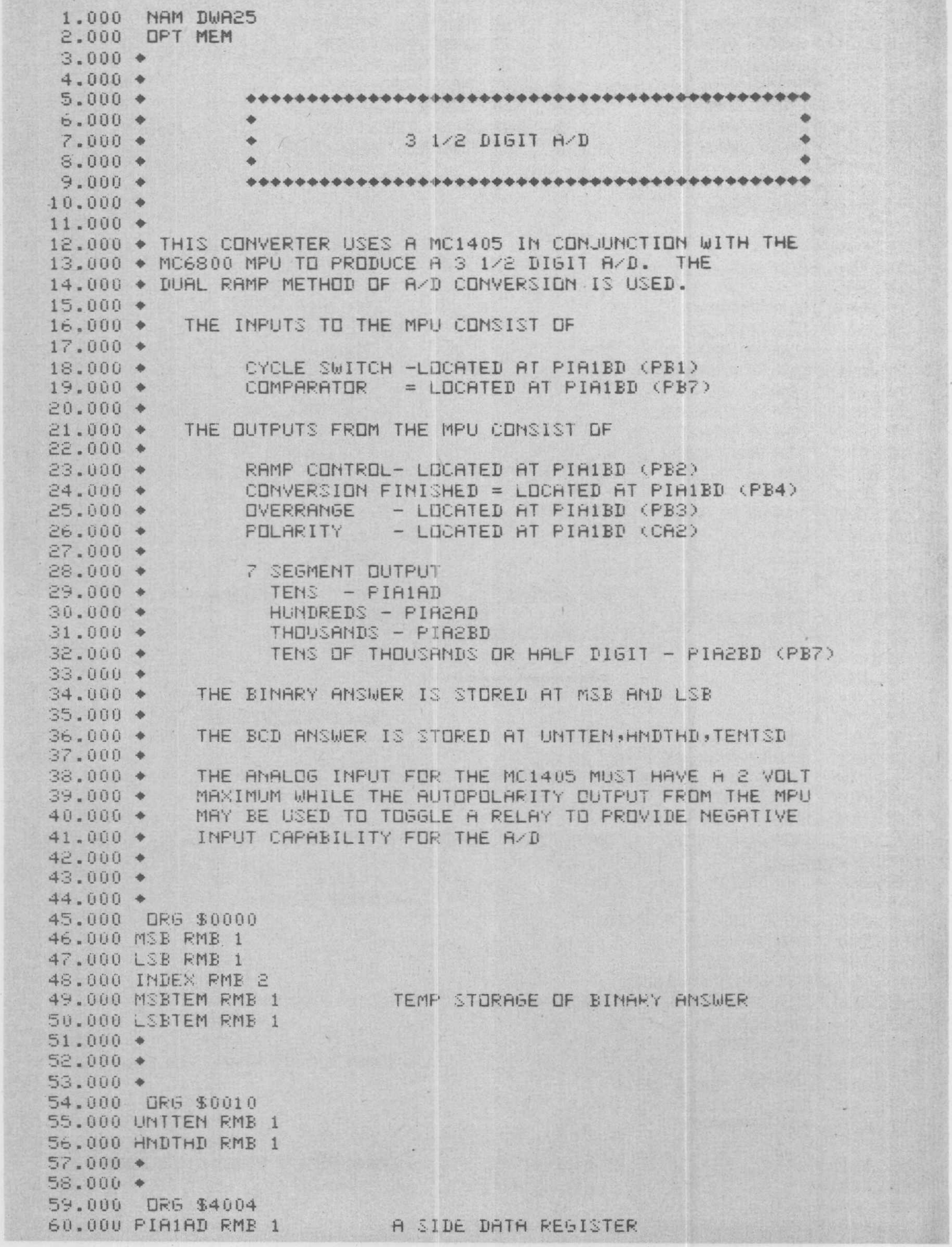


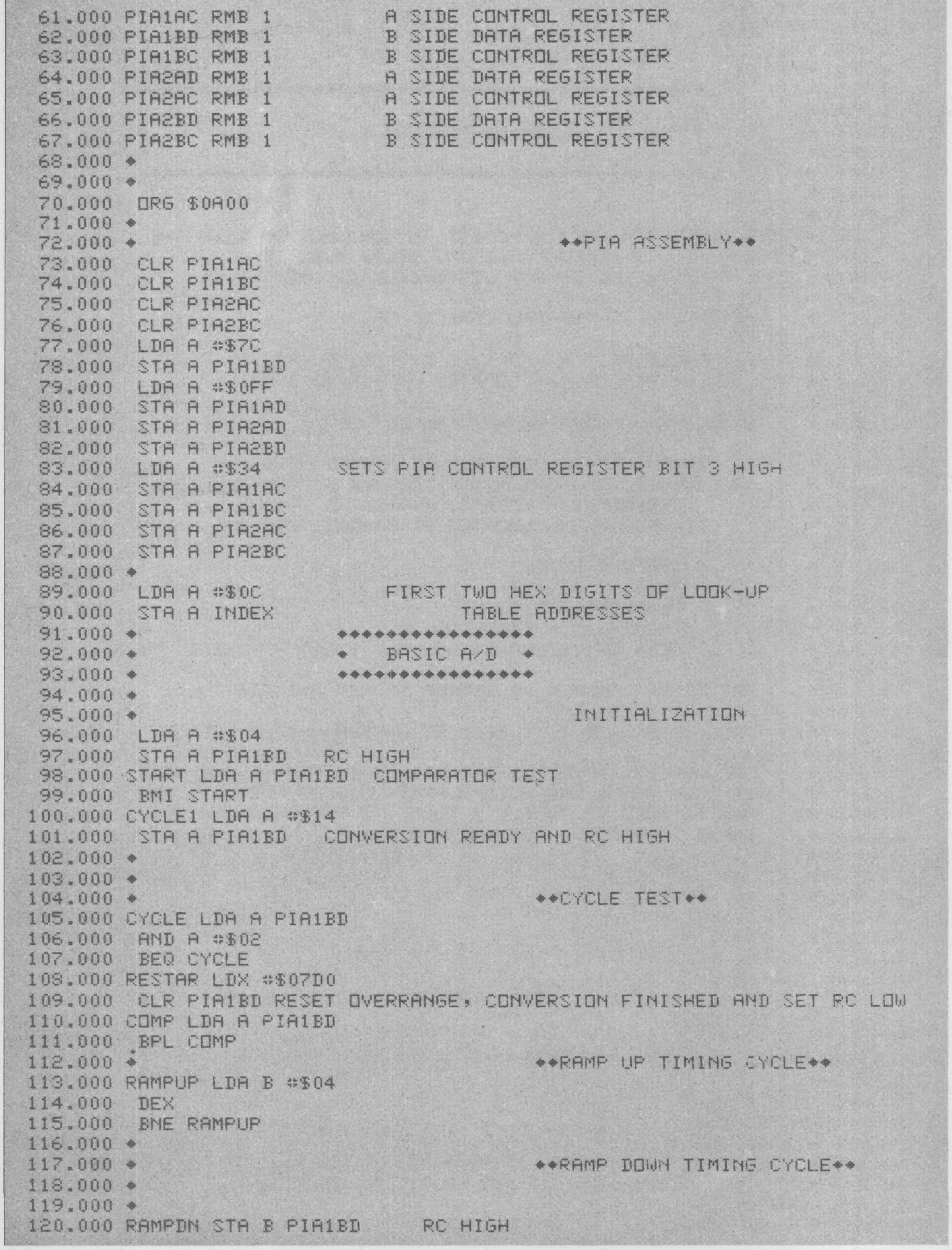
FIGURE 16 – 3½-Digit Dual Ramp A/D Flow Diagram

FIGURE 17 — 3½-Digit Dual Ramp Software (Page 1 of 5)

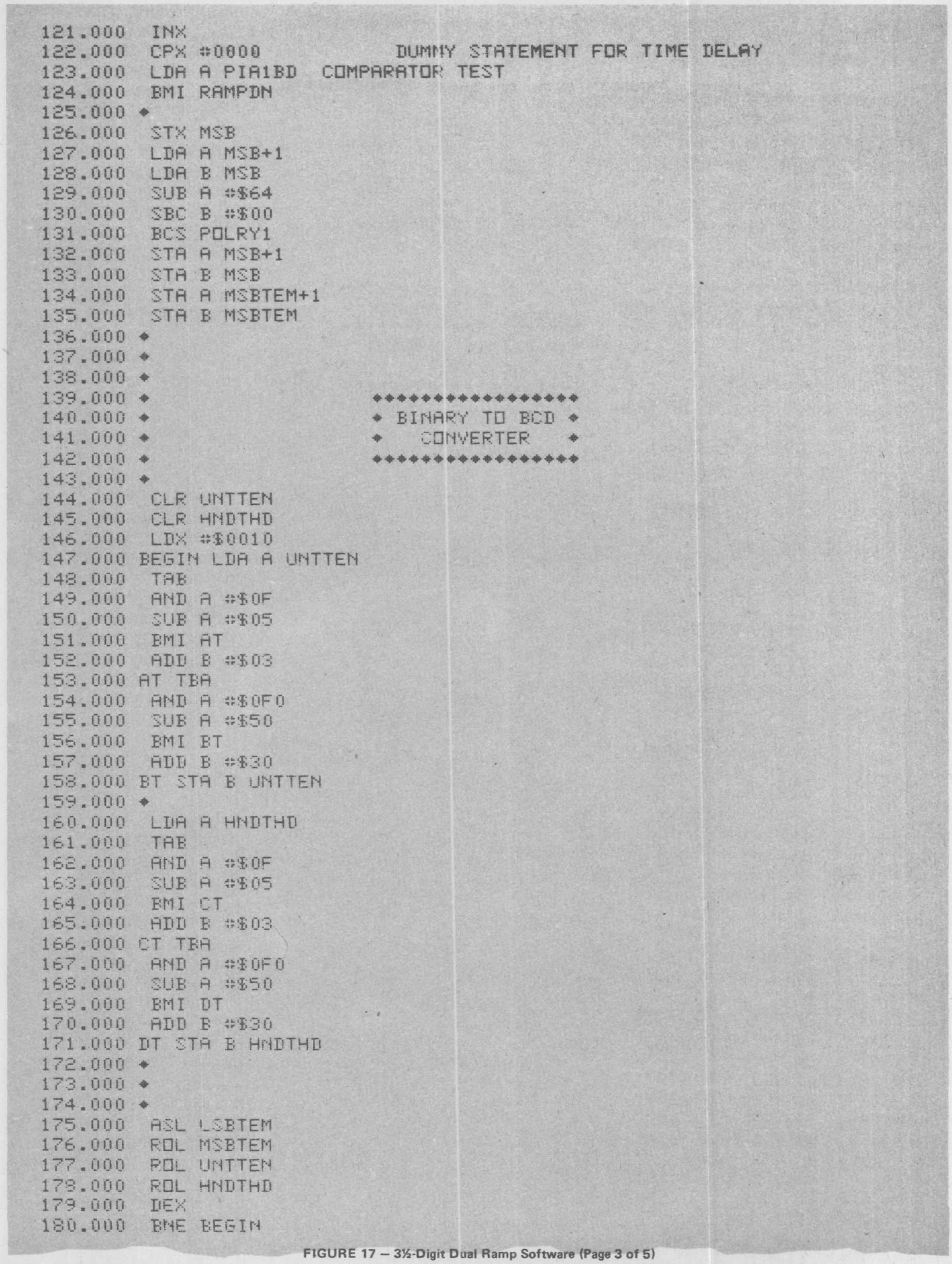


```
 NAM DWA25
 OPT MEM
*
*
*        **********************************************
*        *                                            *
*        *            3 1/2 DIGIT A/D                 *
*        *                                            *
*        **********************************************
*
*
* THIS CONVERTER USES A MC1405 IN CONJUNCTION WITH THE
* MC6800 MPU TO PRODUCE A 3 1/2 DIGIT A/D.  THE
* DUAL RAMP METHOD OF A/D CONVERSION IS USED.
*
*   THE INPUTS TO THE MPU CONSIST OF
*
*        CYCLE SWITCH -LOCATED AT PIA1BD (PB1)
*        COMPARATOR   = LOCATED AT PIA1BD (PB7)
*
*   THE OUTPUTS FROM THE MPU CONSIST OF
*
*        RAMP CONTROL- LOCATED AT PIA1BD (PB2)
*        CONVERSION FINISHED = LOCATED AT PIA1BD (PB4)
*        OVERRANGE   - LOCATED AT PIA1BD (PB3)
*        POLARITY    - LOCATED AT PIA1BD (CA2)
*
*        7 SEGMENT OUTPUT
*          TENS  - PIA1AD
*          HUNDREDS - PIA2AD
*          THOUSANDS - PIA2BD
*          TENS OF THOUSANDS OR HALF DIGIT - PIA2BD (PB7)
*
*    THE BINARY ANSWER IS STORED AT MSB AND LSB
*
*    THE BCD ANSWER IS STORED AT UNTTEN,HNDTHD,TENTSD
*
*    THE ANALOG INPUT FOR THE MC1405 MUST HAVE A 2 VOLT
*    MAXIMUM WHILE THE AUTOPOLARITY OUTPUT FROM THE MPU
*    MAY BE USED TO TOGGLE A RELAY TO PROVIDE NEGATIVE
*    INPUT CAPABILITY FOR THE A/D
*
*
*
 ORG $0000
MSB RMB 1
LSB RMB 1
INDEX RMB 2
MSBTEM RMB 1         TEMP STORAGE OF BINARY ANSWER
LSBTEM RMB 1
*
*
*
 ORG $0010
UNTTEN RMB 1
HNDTHD RMB 1
*
*
 ORG $4004
PIA1AD RMB 1         A SIDE DATA REGISTER
```

FIGURE 17 — 3½-Digit Dual Ramp Software (Page 2 of 5)



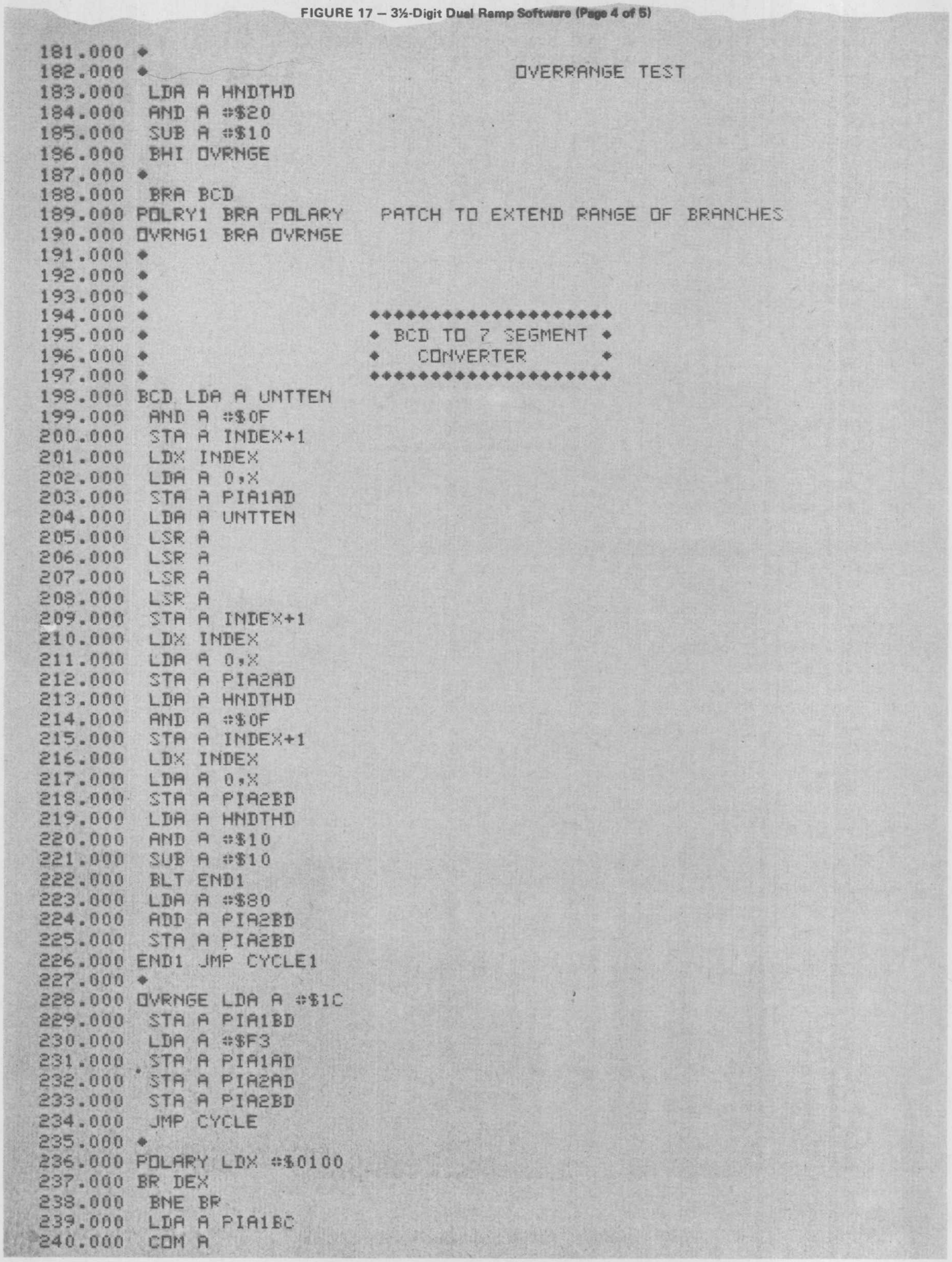
```
PIA1AC RMB 1         A SIDE CONTROL REGISTER
PIA1BD RMB 1         B SIDE DATA REGISTER
PIA1BC RMB 1         B SIDE CONTROL REGISTER
PIA2AD RMB 1         A SIDE DATA REGISTER
PIA2AC RMB 1         A SIDE CONTROL REGISTER
PIA2BD RMB 1         B SIDE DATA REGISTER
PIA2BC RMB 1         B SIDE CONTROL REGISTER
*
*
 ORG $0A00
*
*                                  **PIA ASSEMBLY**
 CLR PIA1AC
 CLR PIA1BC
 CLR PIA2AC
 CLR PIA2BC
 LDA A #$7C
 STA A PIA1BD
 LDA A #$0FF
 STA A PIA1AD
 STA A PIA2AD
 STA A PIA2BD
 LDA A #$34      SETS PIA CONTROL REGISTER BIT 3 HIGH
 STA A PIA1AC
 STA A PIA1BC
 STA A PIA2AC
 STA A PIA2BC
*
 LDA A #$0C          FIRST TWO HEX DIGITS OF LOOK-UP
 STA A INDEX               TABLE ADDRESSES
*               ****************
*               *   BASIC A/D  *
*               ****************
*
*                                   INITIALIZATION
 LDA A #$04
 STA A PIA1BD   RC HIGH
START LDA A PIA1BD  COMPARATOR TEST
 BMI START
CYCLE1 LDA A #$14
 STA A PIA1BD   CONVERSION READY AND RC HIGH
*
*
*                                **CYCLE TEST**
CYCLE LDA A PIA1BD
 AND A #$02
 BEQ CYCLE
RESTAR LDX #$07D0
 CLR PIA1BD RESET OVERRANGE, CONVERSION FINISHED AND SET RC LOW
COMP LDA A PIA1BD
 BPL COMP
*                                **RAMP UP TIMING CYCLE**
RAMPUP LDA B #$04
 DEX
 BNE RAMPUP
*
*                                **RAMP DOWN TIMING CYCLE**
*
*
RAMPDN STA B PIA1BD     RC HIGH
```



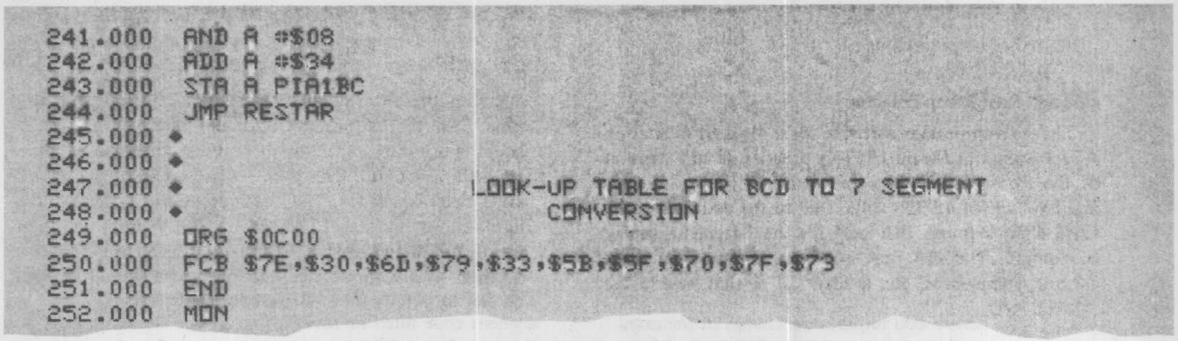
```
121.000  INX
122.000  CPX #0000           DUMMY STATEMENT FOR TIME DELAY
123.000  LDA A PIA1BD  COMPARATOR TEST
124.000  BMI RAMPDN
125.000 *
126.000  STX MSB
127.000  LDA A MSB+1
128.000  LDA B MSB
129.000  SUB A #$64
130.000  SBC B #$00
131.000  BCS POLRY1
132.000  STA A MSB+1
133.000  STA B MSB
134.000  STA A MSBTEM+1
135.000  STA B MSBTEM
136.000 *
137.000 *
138.000 *
139.000 *                  *****************
140.000 *                  * BINARY TO BCD *
141.000 *                  *   CONVERTER   *
142.000 *                  *****************
143.000 *
144.000  CLR UNTTEN
145.000  CLR HNDTHD
146.000  LDX #$0010
147.000 BEGIN LDA A UNTTEN
148.000  TAB
149.000  AND A #$0F
150.000  SUB A #$05
151.000  BMI AT
152.000  ADD B #$03
153.000 AT TBA
154.000  AND A #$0F0
155.000  SUB A #$50
156.000  BMI BT
157.000  ADD B #$30
158.000 BT STA B UNTTEN
159.000 *
160.000  LDA A HNDTHD
161.000  TAB
162.000  AND A #$0F
163.000  SUB A #$05
164.000  BMI CT
165.000  ADD B #$03
166.000 CT TBA
167.000  AND A #$0F0
168.000  SUB A #$50
169.000  BMI DT
170.000  ADD B #$30
171.000 DT STA B HNDTHD
172.000 *
173.000 *
174.000 *
175.000  ASL LSBTEM
176.000  ROL MSBTEM
177.000  ROL UNTTEN
178.000  ROL HNDTHD
179.000  DEX
180.000  BNE BEGIN
```

FIGURE 17 — 3½-Digit Dual Ramp Software (Page 3 of 5)

FIGURE 17 — 3½-Digit Dual Ramp Software (Page 4 of 5)

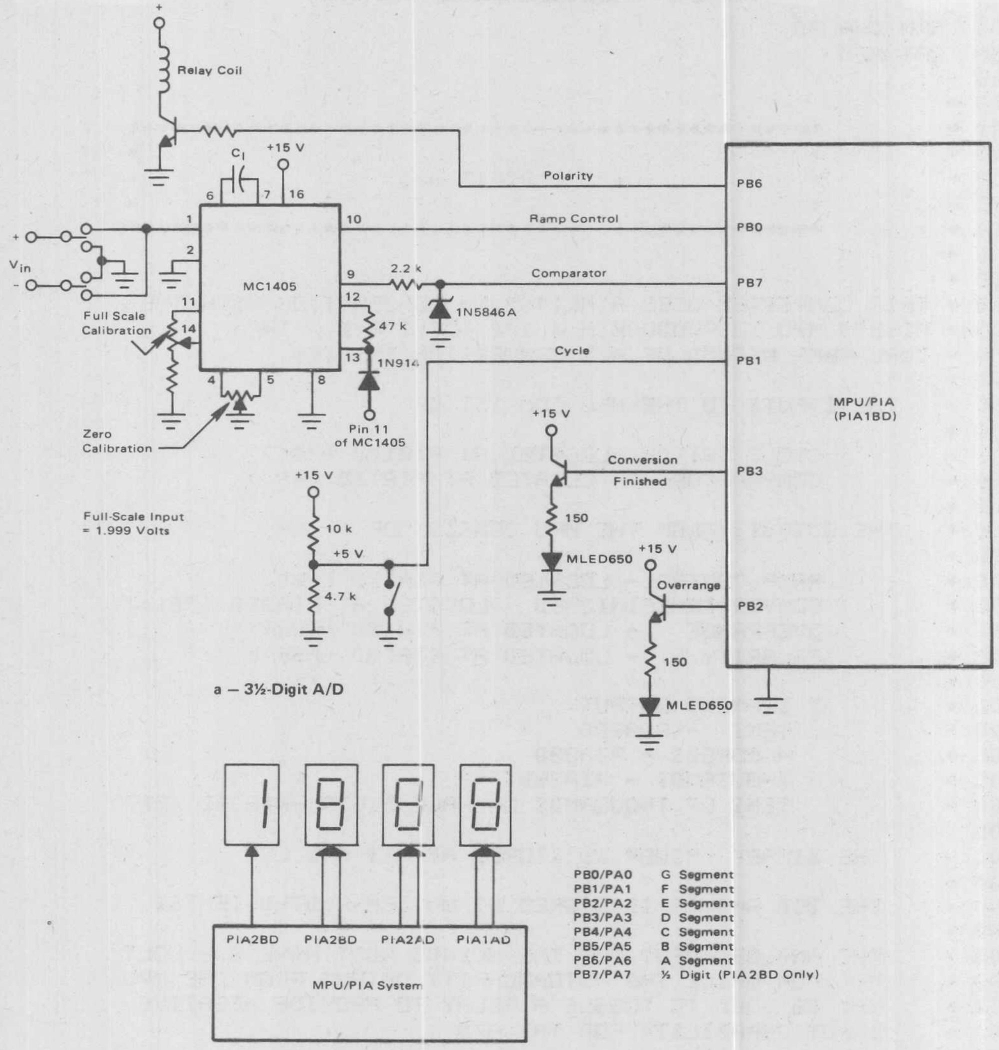


```
*
*                              OVERRANGE TEST
 LDA A HNDTHD
 AND A #$20
 SUB A #$10
 BHI OVRNGE
*
 BRA BCD
POLRY1 BRA POLARY   PATCH TO EXTEND RANGE OF BRANCHES
OVRNG1 BRA OVRNGE
*
*
*
*                   ********************
*                   * BCD TO 7 SEGMENT *
*                   *   CONVERTER      *
*                   ********************
BCD LDA A UNTTEN
 AND A #$0F
 STA A INDEX+1
 LDX INDEX
 LDA A 0,X
 STA A PIA1AD
 LDA A UNTTEN
 LSR A
 LSR A
 LSR A
 LSR A
 STA A INDEX+1
 LDX INDEX
 LDA A 0,X
 STA A PIA2AD
 LDA A HNDTHD
 AND A #$0F
 STA A INDEX+1
 LDX INDEX
 LDA A 0,X
 STA A PIA2BD
 LDA A HNDTHD
 AND A #$10
 SUB A #$10
 BLT END1
 LDA A #$80
 ADD A PIA2BD
 STA A PIA2BD
END1 JMP CYCLE1
*
OVRNGE LDA A #$1C
 STA A PIA1BD
 LDA A #$F3
 STA A PIA1AD
 STA A PIA2AD
 STA A PIA2BD
 JMP CYCLE
*
POLARY LDX #$0100
BR DEX
 BNE BR
 LDA A PIA1BC
 COM A
```



```
241.000  AND A #$08
242.000  ADD A #$34
243.000  STA A PIA1BC
244.000  JMP RESTAR
245.000 *
246.000 *
247.000 *                       LOOK-UP TABLE FOR BCD TO 7 SEGMENT
248.000 *                               CONVERSION
249.000  ORG $0C00
250.000  FCB $7E,$30,$6D,$79,$33,$5B,$5F,$70,$7F,$73
251.000  END
252.000  MON
```

FIGURE 17 – 3½-Digit Dual Ramp Software (Page 5 of 5)



a – 3½-Digit A/D

b – PIA Displays

FIGURE 18 – 3½-Digit Dual Ramp A/D Hardware

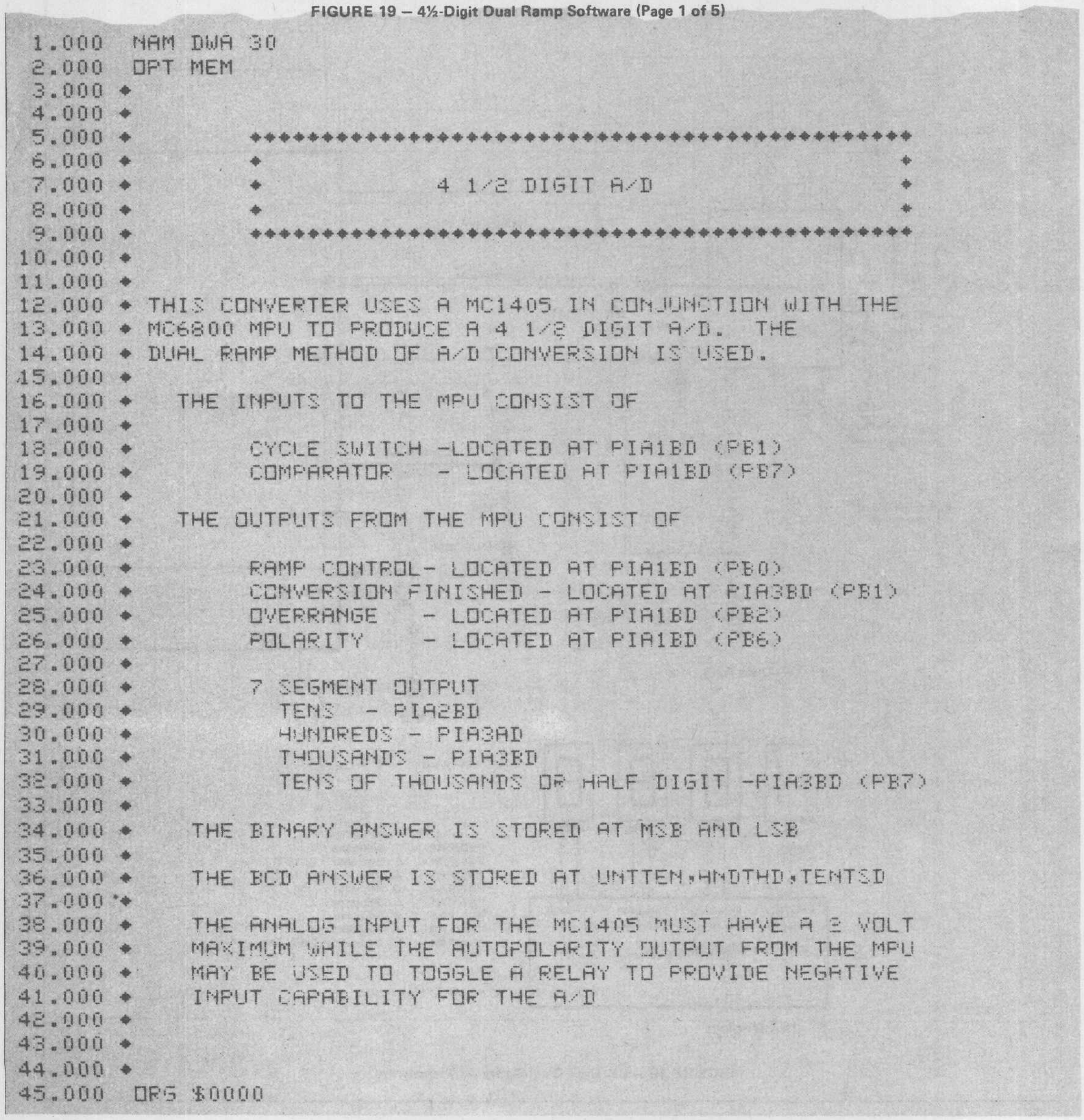
4½-Digit Dual Ramp Program

The microprocessor software for a 4½-digit dual ramp A/D is shown in Figure 19. This program in an extension of the 3½-digit A/D just discussed and has a full scale input voltage of 1.9999 volts. Due to the addition of the extra digit, a fourth PIA port for the 7-segment display is required. The PIA port configuration used for ramp control, comparator, etc. is identical to that used in the 3½-digit A/D.

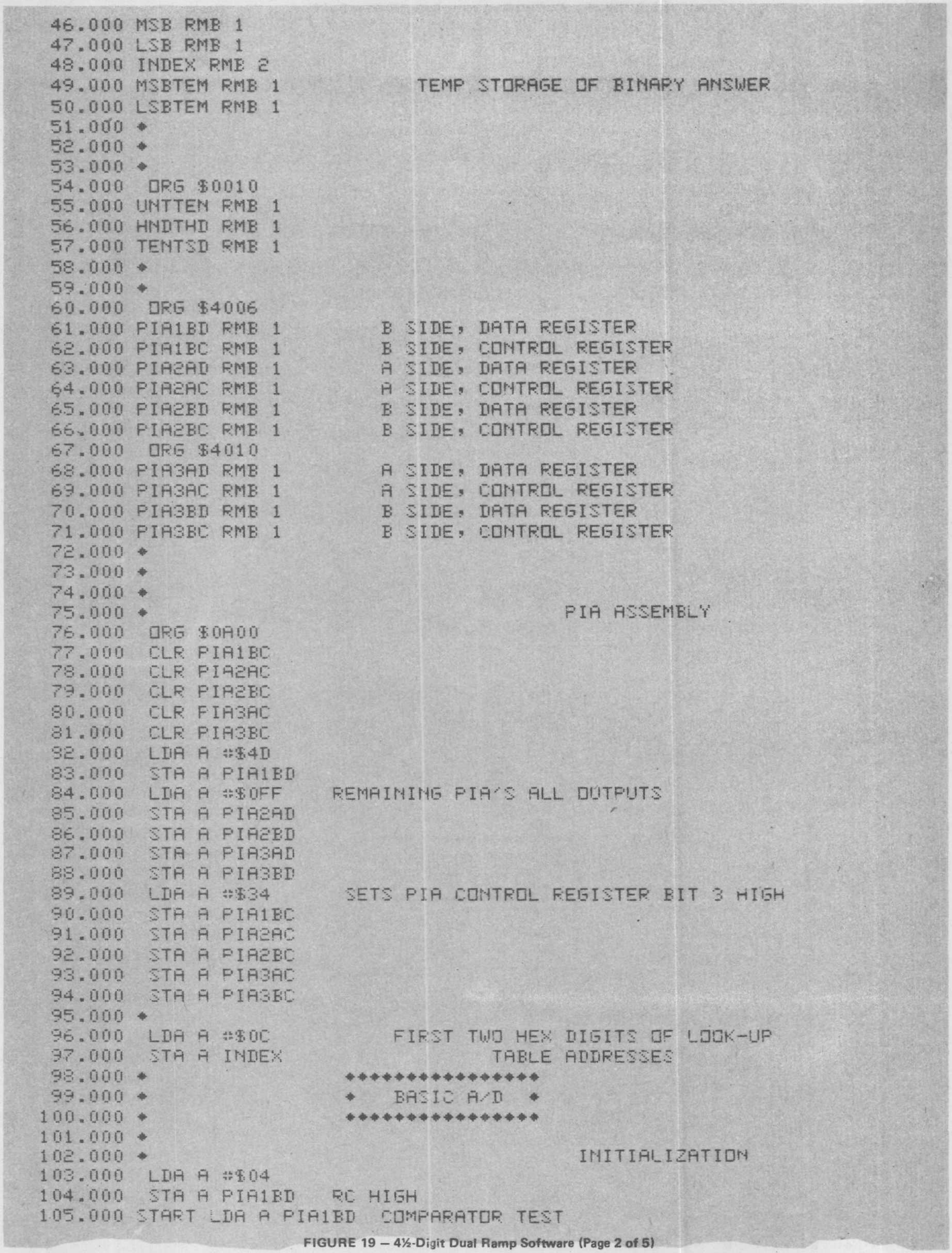
The addition of the extra digit also implies a longer ramp up time period which is produced by increasing the initialization of the index register in line 115. This longer ramp up time period also requires the change of the extra count subtraction statements of lines 137 and 138 to maintain the extra count subtraction of 10% ramp up time. Also, the longer ramp up time period will require a larger integration capacitor to prevent saturation of the MC1405 integrator. This is of course, assuming the same MPU clock frequency. The remainder of the A/D external hardware is unchanged except for the addition of the fourth full digital display. Figure 18a can be used for the 4½-digit A/D without modification, and Figure 18b can be used with only the addition of another digit.

The software for the binary-to-BCD converter remains the same for the 4½-digit A/D since it is capable of handling up to 16 bits. The conversion routine for BCD-to-7 segment code must be modified to handle the extra digit although the same basic technique is retained.

FIGURE 19 – 4½-Digit Dual Ramp Software (Page 1 of 5)



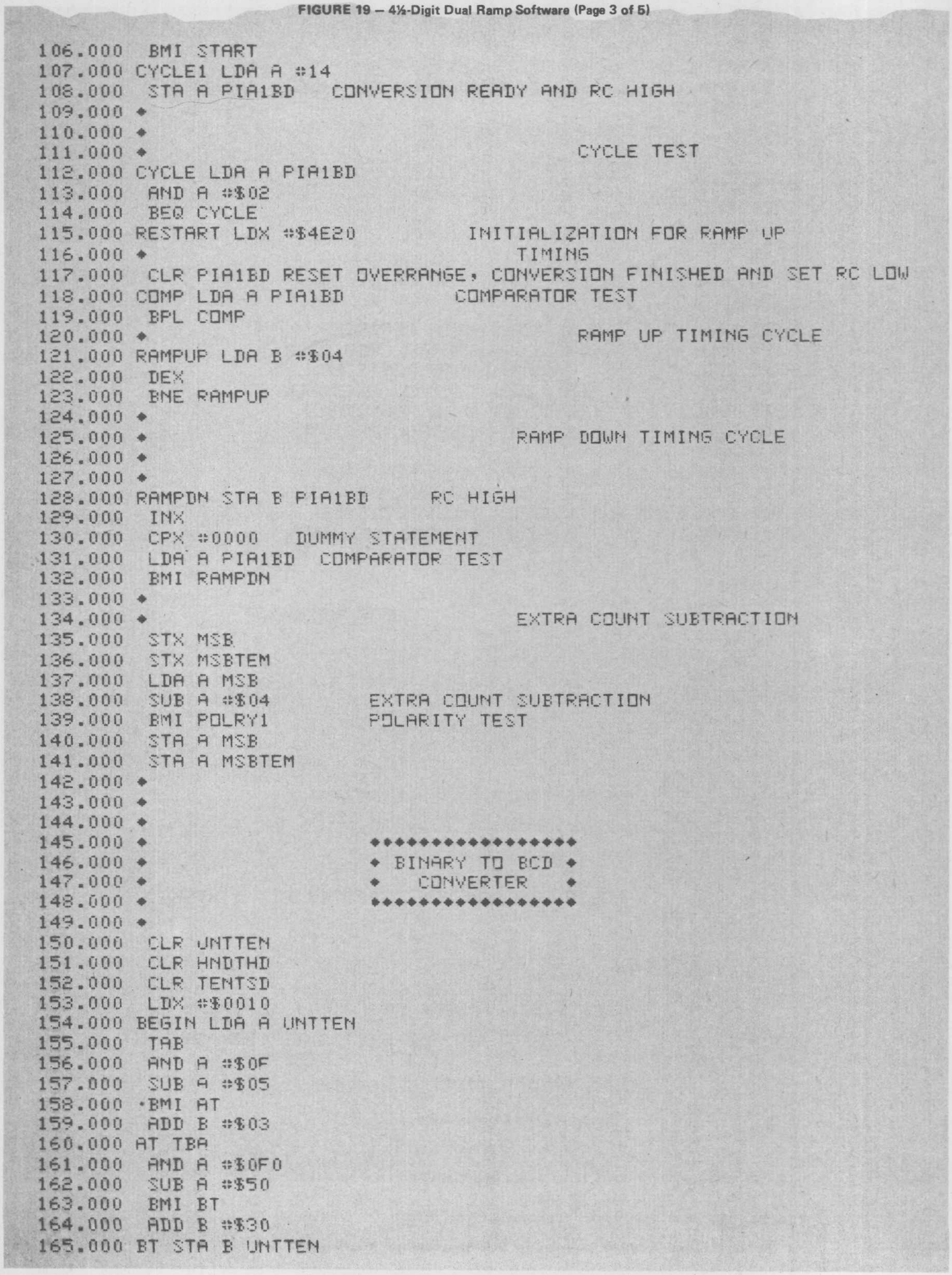
```
 1.000  NAM DWA 30
 2.000  OPT MEM
 3.000 *
 4.000 *
 5.000 *        **********************************************
 6.000 *        *                                            *
 7.000 *        *            4 1/2 DIGIT A/D                 *
 8.000 *        *                                            *
 9.000 *        **********************************************
10.000 *
11.000 *
12.000 * THIS CONVERTER USES A MC1405 IN CONJUNCTION WITH THE
13.000 * MC6800 MPU TO PRODUCE A 4 1/2 DIGIT A/D.  THE
14.000 * DUAL RAMP METHOD OF A/D CONVERSION IS USED.
15.000 *
16.000 *   THE INPUTS TO THE MPU CONSIST OF
17.000 *
18.000 *        CYCLE SWITCH -LOCATED AT PIA1BD (PB1)
19.000 *        COMPARATOR   - LOCATED AT PIA1BD (PB7)
20.000 *
21.000 *   THE OUTPUTS FROM THE MPU CONSIST OF
22.000 *
23.000 *        RAMP CONTROL- LOCATED AT PIA1BD (PB0)
24.000 *        CONVERSION FINISHED - LOCATED AT PIA3BD (PB1)
25.000 *        OVERRANGE    - LOCATED AT PIA1BD (PB2)
26.000 *        POLARITY     - LOCATED AT PIA1BD (PB6)
27.000 *
28.000 *        7 SEGMENT OUTPUT
29.000 *          TENS  - PIA2BD
30.000 *          HUNDREDS - PIA3AD
31.000 *          THOUSANDS - PIA3BD
32.000 *          TENS OF THOUSANDS OR HALF DIGIT -PIA3BD (PB7)
33.000 *
34.000 *    THE BINARY ANSWER IS STORED AT MSB AND LSB
35.000 *
36.000 *    THE BCD ANSWER IS STORED AT UNTTEN,HNDTHD,TENTSD
37.000 *
38.000 *    THE ANALOG INPUT FOR THE MC1405 MUST HAVE A 3 VOLT
39.000 *    MAXIMUM WHILE THE AUTOPOLARITY OUTPUT FROM THE MPU
40.000 *    MAY BE USED TO TOGGLE A RELAY TO PROVIDE NEGATIVE
41.000 *    INPUT CAPABILITY FOR THE A/D
42.000 *
43.000 *
44.000 *
45.000  ORG $0000
```



```
MSB RMB 1
LSB RMB 1
INDEX RMB 2
MSBTEM RMB 1           TEMP STORAGE OF BINARY ANSWER
LSBTEM RMB 1
*
*
*
 ORG $0010
UNTTEN RMB 1
HNDTHD RMB 1
TENTSD RMB 1
*
*
 ORG $4006
PIA1BD RMB 1        B SIDE, DATA REGISTER
PIA1BC RMB 1        B SIDE, CONTROL REGISTER
PIA2AD RMB 1        A SIDE, DATA REGISTER
PIA2AC RMB 1        A SIDE, CONTROL REGISTER
PIA2BD RMB 1        B SIDE, DATA REGISTER
PIA2BC RMB 1        B SIDE, CONTROL REGISTER
 ORG $4010
PIA3AD RMB 1        A SIDE, DATA REGISTER
PIA3AC RMB 1        A SIDE, CONTROL REGISTER
PIA3BD RMB 1        B SIDE, DATA REGISTER
PIA3BC RMB 1        B SIDE, CONTROL REGISTER
*
*
*
*                                  PIA ASSEMBLY
 ORG $0A00
 CLR PIA1BC
 CLR PIA2AC
 CLR PIA2BC
 CLR PIA3AC
 CLR PIA3BC
 LDA A #$4D
 STA A PIA1BD
 LDA A #$0FF    REMAINING PIA'S ALL OUTPUTS
 STA A PIA2AD
 STA A PIA2BD
 STA A PIA3AD
 STA A PIA3BD
 LDA A #$34      SETS PIA CONTROL REGISTER BIT 3 HIGH
 STA A PIA1BC
 STA A PIA2AC
 STA A PIA2BC
 STA A PIA3AC
 STA A PIA3BC
*
 LDA A #$0C          FIRST TWO HEX DIGITS OF LOOK-UP
 STA A INDEX                 TABLE ADDRESSES
*                ****************
*                *   BASIC A/D  *
*                ****************
*
*                                    INITIALIZATION
 LDA A #$04
 STA A PIA1BD   RC HIGH
START LDA A PIA1BD  COMPARATOR TEST
```

FIGURE 19 — 4½-Digit Dual Ramp Software (Page 2 of 5)

FIGURE 19 — 4½-Digit Dual Ramp Software (Page 3 of 5)



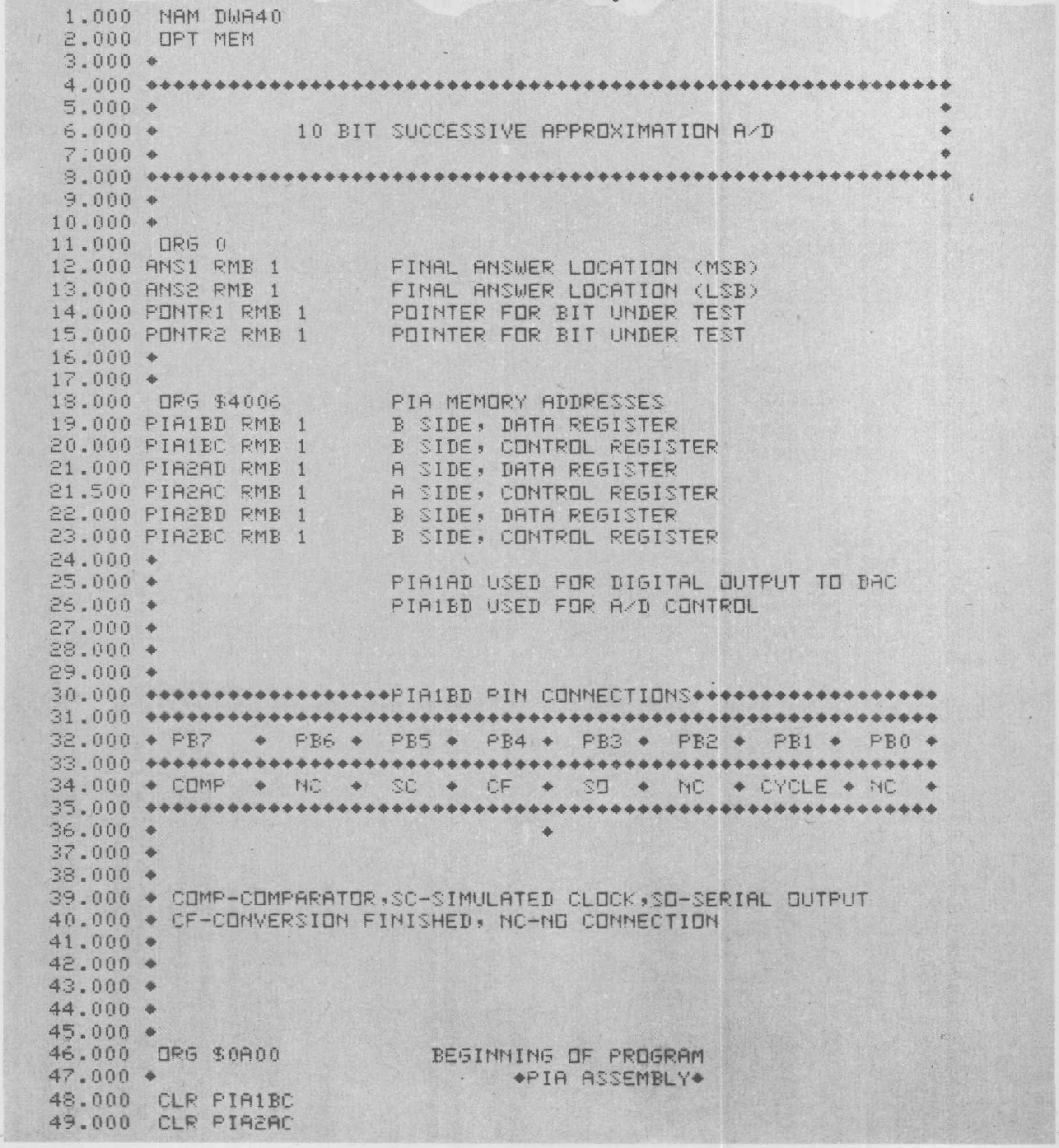
```
 BMI START
CYCLE1 LDA A #14
 STA A PIA1BD   CONVERSION READY AND RC HIGH
*
*
*                                   CYCLE TEST
CYCLE LDA A PIA1BD
 AND A #$02
 BEQ CYCLE
RESTART LDX #$4E20         INITIALIZATION FOR RAMP UP
*                              TIMING
 CLR PIA1BD RESET OVERRANGE, CONVERSION FINISHED AND SET RC LOW
COMP LDA A PIA1BD         COMPARATOR TEST
 BPL COMP
*                                   RAMP UP TIMING CYCLE
RAMPUP LDA B #$04
 DEX
 BNE RAMPUP
*
*                              RAMP DOWN TIMING CYCLE
*
*
RAMPDN STA B PIA1BD     RC HIGH
 INX
 CPX #0000   DUMMY STATEMENT
 LDA A PIA1BD  COMPARATOR TEST
 BMI RAMPDN
*
*                              EXTRA COUNT SUBTRACTION
 STX MSB
 STX MSBTEM
 LDA A MSB
 SUB A #$04        EXTRA COUNT SUBTRACTION
 BMI POLRY1        POLARITY TEST
 STA A MSB
 STA A MSBTEM
*
*
*
*                  *****************
*                  * BINARY TO BCD *
*                  *   CONVERTER   *
*                  *****************
*
 CLR UNTTEN
 CLR HNDTHD
 CLR TENTSD
 LDX #$0010
BEGIN LDA A UNTTEN
 TAB
 AND A #$0F
 SUB A #$05
 BMI AT
 ADD B #$03
AT TBA
 AND A #$0F0
 SUB A #$50
 BMI BT
 ADD B #$30
BT STA B UNTTEN
```

10-Bit SA Program

Figures 8 and 9 show the MPU software and external hardware for a 10-bit successive approximation A/D using the MC3410 DAC. The operation of this A/D is very similar to that of the 8-bit A/D. Both the A and B halves of a PIA are required for the DAC output while the control lines (comparator, conversion finished, etc.) are also identical to that of the 8-bit A/D previously discussed. The pointer for indicating which bit is currently under test is contained in two memory locations, PONTR1 and PONTR2. The pointer is initialized in lines 63 and 64 and as before, it is continuously shifted to the left as each bit is tested. Lines 72 through 77 and lines 89 through 101 operate on both halves of the PIA, "setting" and "resetting" the DAC bits under test. The final answer is stored in the two PIA memory locations as well as two internal memory locations (ANS1 and ANS2).

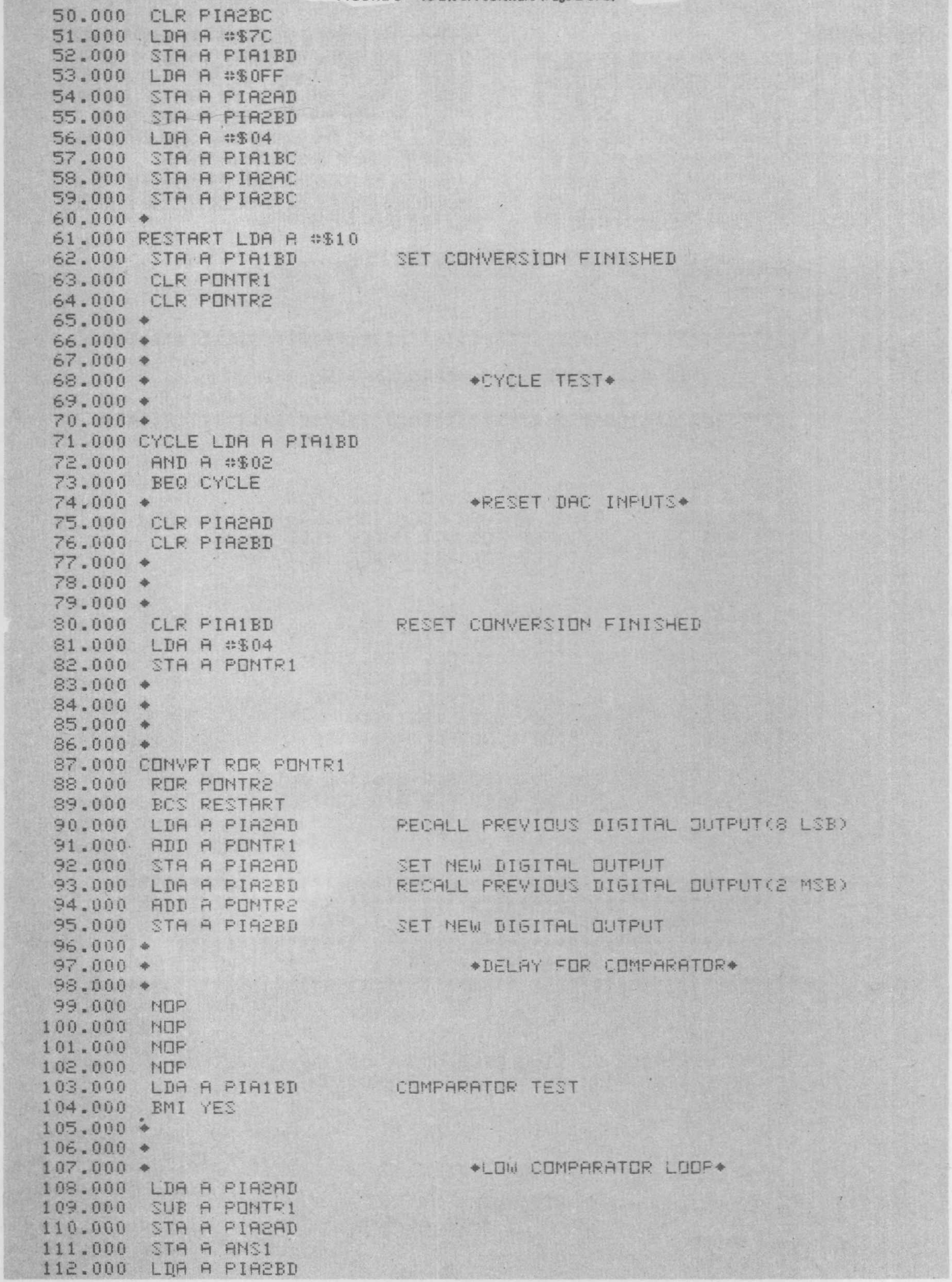
By using the appropriate DAC and changing line 63 of the software program, the 10-bit SA D/A can be modified for 9-16 bit A/D operation.

FIGURE 8 — 10-Bit SA Software (Page 1 of 3)



```
 1.000  NAM DWA40
 2.000  OPT MEM
 3.000 *
 4.000 ***************************************************************
 5.000 *                                                             *
 6.000 *          10 BIT SUCCESSIVE APPROXIMATION A/D                *
 7.000 *                                                             *
 8.000 ***************************************************************
 9.000 *
10.000 *
11.000  ORG 0
12.000 ANS1 RMB 1           FINAL ANSWER LOCATION (MSB)
13.000 ANS2 RMB 1           FINAL ANSWER LOCATION (LSB)
14.000 PONTR1 RMB 1         POINTER FOR BIT UNDER TEST
15.000 PONTR2 RMB 1         POINTER FOR BIT UNDER TEST
16.000 *
17.000 *
18.000  ORG $4006           PIA MEMORY ADDRESSES
19.000 PIA1BD RMB 1         B SIDE, DATA REGISTER
20.000 PIA1BC RMB 1         B SIDE, CONTROL REGISTER
21.000 PIA2AD RMB 1         A SIDE, DATA REGISTER
21.500 PIA2AC RMB 1         A SIDE, CONTROL REGISTER
22.000 PIA2BD RMB 1         B SIDE, DATA REGISTER
23.000 PIA2BC RMB 1         B SIDE, CONTROL REGISTER
24.000 *
25.000 *                    PIA1AD USED FOR DIGITAL OUTPUT TO DAC
26.000 *                    PIA1BD USED FOR A/D CONTROL
27.000 *
28.000 *
29.000 *
30.000 *****************PIA1BD PIN CONNECTIONS*****************
31.000 ***********************************************************
32.000 * PB7    * PB6 * PB5 * PB4 * PB3 * PB2 * PB1 * PB0 *
33.000 ***********************************************************
34.000 * COMP   * NC  * SC  * CF  * SO  * NC  * CYCLE * NC  *
35.000 ***********************************************************
36.000 *                          *
37.000 *
38.000 *
39.000 * COMP-COMPARATOR,SC-SIMULATED CLOCK,SO-SERIAL OUTPUT
40.000 * CF-CONVERSION FINISHED, NC-NO CONNECTION
41.000 *
42.000 *
43.000 *
44.000 *
45.000 *
46.000  ORG $0A00              BEGINNING OF PROGRAM
47.000 *                           *PIA ASSEMBLY*
48.000  CLR PIA1BC
49.000  CLR PIA2AC
```

FIGURE 8 — 10-Bit SA Software (Page 2 of 3)



```
 CLR PIA2BC
 LDA A #$7C
 STA A PIA1BD
 LDA A #$0FF
 STA A PIA2AD
 STA A PIA2BD
 LDA A #$04
 STA A PIA1BC
 STA A PIA2AC
 STA A PIA2BC
*
RESTART LDA A #$10
 STA A PIA1BD        SET CONVERSION FINISHED
 CLR PONTR1
 CLR PONTR2
*
*
*
*                          *CYCLE TEST*
*
*
CYCLE LDA A PIA1BD
 AND A #$02
 BEQ CYCLE
*                          *RESET DAC INPUTS*
 CLR PIA2AD
 CLR PIA2BD
*
*
*
 CLR PIA1BD          RESET CONVERSION FINISHED
 LDA A #$04
 STA A PONTR1
*
*
*
*
CONVRT ROR PONTR1
 ROR PONTR2
 BCS RESTART
 LDA A PIA2AD        RECALL PREVIOUS DIGITAL OUTPUT(8 LSB)
 ADD A PONTR1
 STA A PIA2AD        SET NEW DIGITAL OUTPUT
 LDA A PIA2BD        RECALL PREVIOUS DIGITAL OUTPUT(2 MSB)
 ADD A PONTR2
 STA A PIA2BD        SET NEW DIGITAL OUTPUT
*
*                          *DELAY FOR COMPARATOR*
*
 NOP
 NOP
 NOP
 NOP
 LDA A PIA1BD        COMPARATOR TEST
 BMI YES
*
*
*                          *LOW COMPARATOR LOOP*
 LDA A PIA2AD
 SUB A PONTR1
 STA A PIA2AD
 STA A ANS1
 LDA A PIA2BD
```

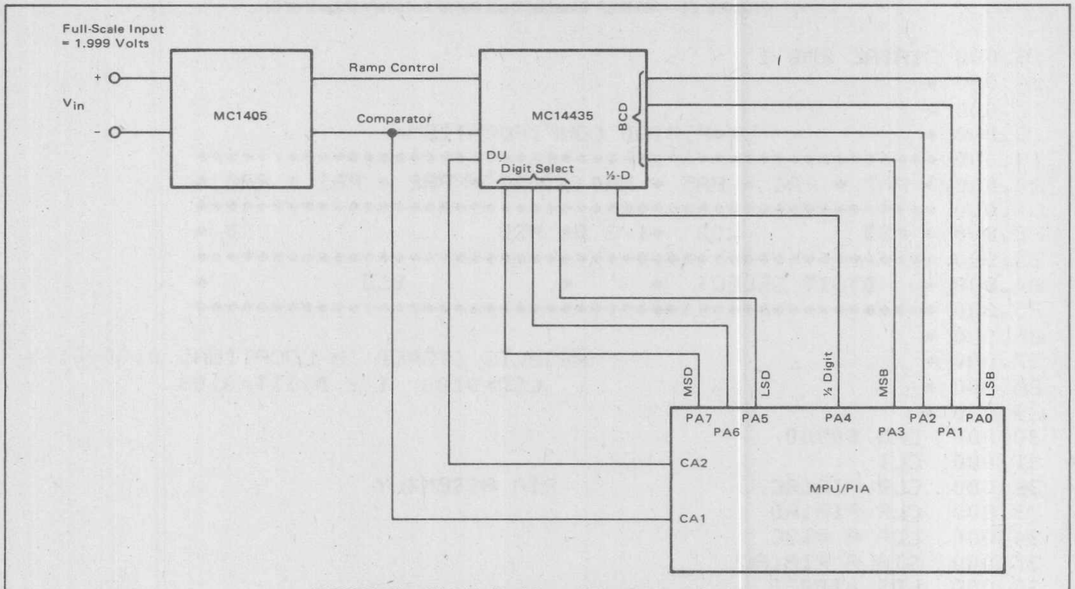


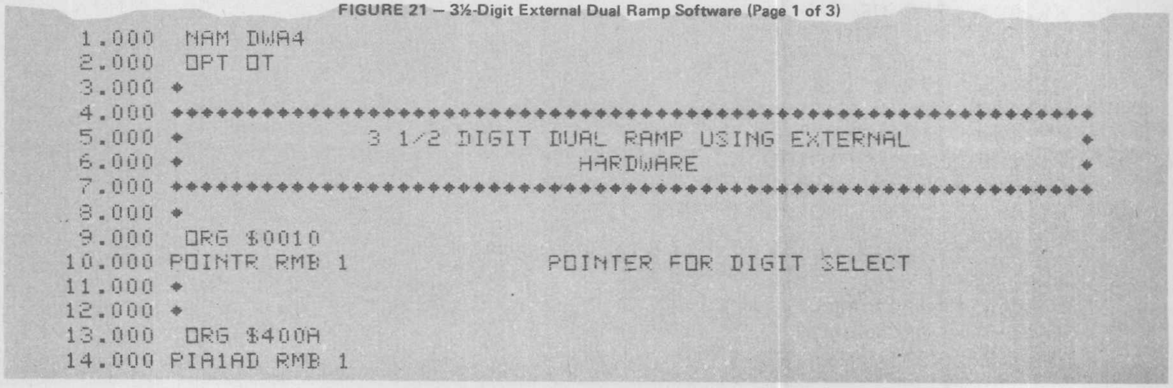
FIGURE 20 — 3½-Digit Dual Ramp Using External Hardware

One port of a PIA is required for the interface to the MPU. The I/O configuration of this PIA is shown in lines 18 through 25 of the source program (Figure 21). The output of the MC14435 digital subsystem consists of three multiplexed BCD digits with the half digit output provided on a separate pin. The three most significant bits of the PIA port are connected to the digit select lines of the MC14435 while the four LSBs are connected to the BCD lines of the MC14435. The remaining PIA bit is connected to the half digit output. Lines 36 through 39 simulate the main MPU program which branches to the A/D subroutine starting in line 42. When this occurs the display update pin of the MC14435 (CA2) is set low which allows only the next data update to enter the MC14435 output latches. The wait for interrupt (WAI) instruction (line 44) stores the MPU stack and waits until the comparator output causes an interrupt on CA1.

At this point the processor is interrupted and vectored to the program beginning at line 50 causing it to demultiplex the BCD data on the output of the MC14435. The least significant digit (LSD) is first selected by the pointer of lines 50 and 51. When a low condition on this LSD line occurs, the BCD data is stored via the indexed mode of addressing in memory location $0100. The pointer is then shifted to the next position (line 57) and when the digit select line goes low the BCD data is stored in the next sequential memory location ($0101). Then the MSD BCD value is placed in memory location $0102 when the MSD digit select goes low. After the multiplexed BCD data has been placed in memory, the half digit is placed in memory location $0103. At this point the display update line to the MC14435 is returned to a high position and the MPU returns from the interrupt and then from the subroutine back to the main program which requested the data.

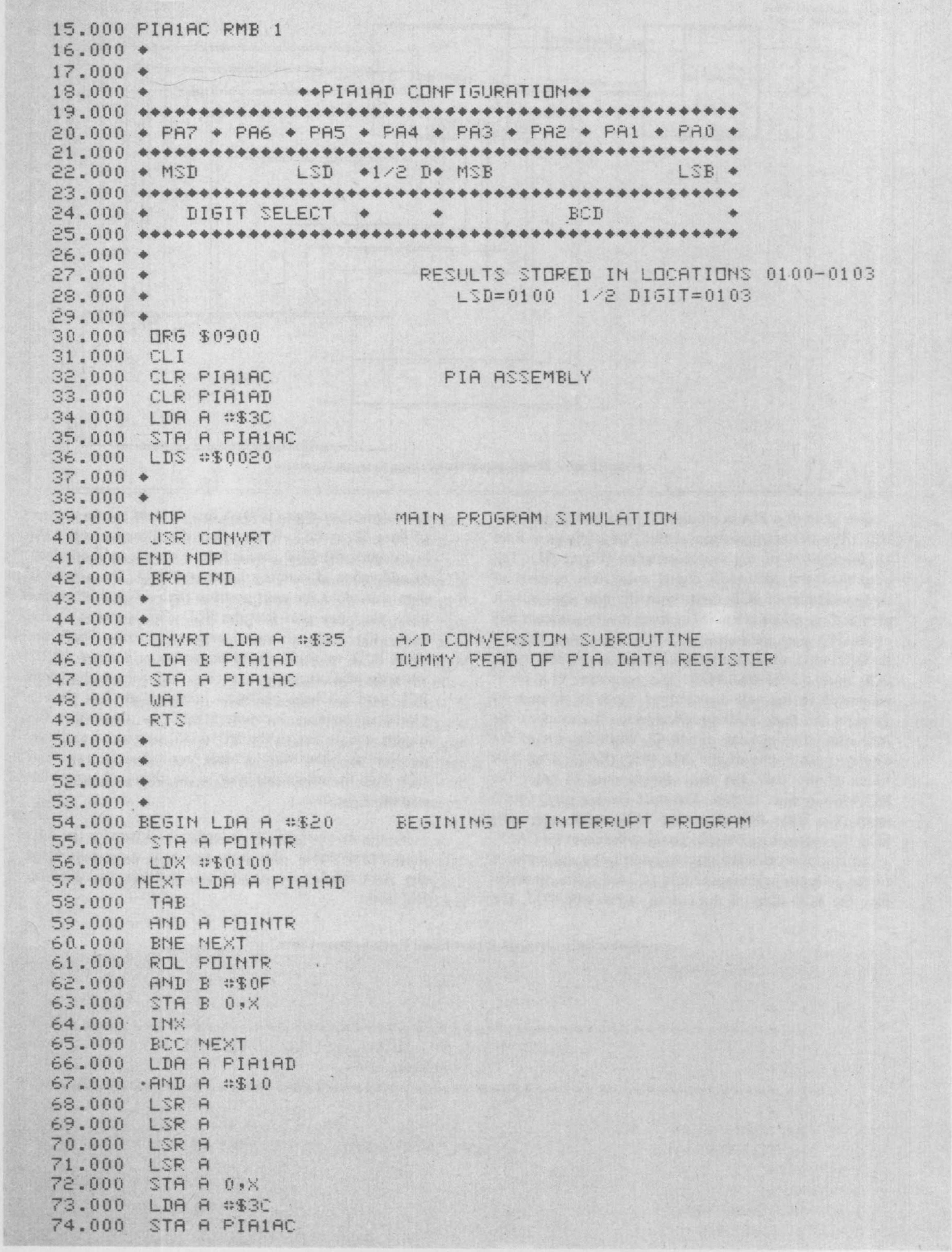
A minimum of 183 μs is required to transfer the A/D data to the MPU. This time period is dependent upon the A/D clock frequency which controls the digit select lines.

FIGURE 21 — 3½-Digit External Dual Ramp Software (Page 1 of 3)

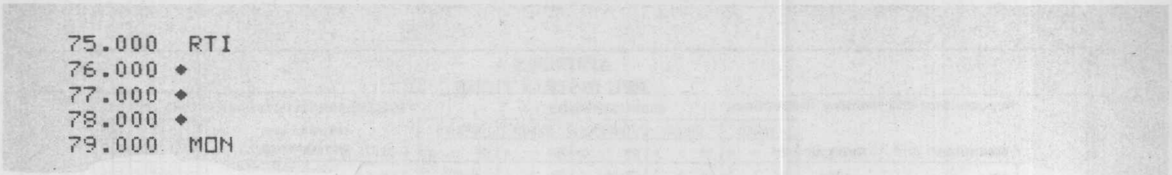


```
 1.000  NAM DWA4
 2.000  OPT OT
 3.000 *
 4.000 **********************************************************
 5.000 *              3 1/2 DIGIT DUAL RAMP USING EXTERNAL       *
 6.000 *                        HARDWARE                         *
 7.000 **********************************************************
 8.000 *
 9.000  ORG $0010
10.000 POINTR RMB 1                 POINTER FOR DIGIT SELECT
11.000 *
12.000 *
13.000  ORG $400A
14.000 PIA1AD RMB 1
```

FIGURE 21 — 3½-Digit External Dual Ramp Software (Page 2 of 3)



```
15.000 PIA1AC RMB 1
16.000 ◆
17.000 ◆
18.000 ◆            ◆◆PIA1AD CONFIGURATION◆◆
19.000 ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
20.000 ◆ PA7 ◆ PA6 ◆ PA5 ◆ PA4 ◆ PA3 ◆ PA2 ◆ PA1 ◆ PA0 ◆
21.000 ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
22.000 ◆ MSD         LSD  ◆1/2 D◆ MSB               LSB ◆
23.000 ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
24.000 ◆   DIGIT SELECT   ◆    ◆          BCD          ◆
25.000 ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
26.000 ◆
27.000 ◆                      RESULTS STORED IN LOCATIONS 0100-0103
28.000 ◆                         LSD=0100  1/2 DIGIT=0103
29.000 ◆
30.000  ORG $0900
31.000  CLI
32.000  CLR PIA1AC                PIA ASSEMBLY
33.000  CLR PIA1AD
34.000  LDA A #$3C
35.000  STA A PIA1AC
36.000  LDS #$0020
37.000 ◆
38.000 ◆
39.000  NOP                   MAIN PROGRAM SIMULATION
40.000  JSR CONVRT
41.000 END NOP
42.000  BRA END
43.000 ◆
44.000 ◆
45.000 CONVRT LDA A #$35      A/D CONVERSION SUBROUTINE
46.000  LDA B PIA1AD          DUMMY READ OF PIA DATA REGISTER
47.000  STA A PIA1AC
48.000  WAI
49.000  RTS
50.000 ◆
51.000 ◆
52.000 ◆
53.000 ◆
54.000 BEGIN LDA A #$20       BEGINING OF INTERRUPT PROGRAM
55.000  STA A POINTR
56.000  LDX #$0100
57.000 NEXT LDA A PIA1AD
58.000  TAB
59.000  AND A POINTR
60.000  BNE NEXT
61.000  ROL POINTR
62.000  AND B #$0F
63.000  STA B 0,X
64.000  INX
65.000  BCC NEXT
66.000  LDA A PIA1AD
67.000  AND A #$10
68.000  LSR A
69.000  LSR A
70.000  LSR A
71.000  LSR A
72.000  STA A 0,X
73.000  LDA A #$3C
74.000  STA A PIA1AC
```



```
75.000  RTI
76.000 ◆
77.000 ◆
78.000 ◆
79.000  MON
```

FIGURE 21 – 3½-Digit External Dual Ramp Software (Page 3 of 3)

SUMMARY

Many MPU systems require analog information, which necessitates the use of an A/D converter in the microprocessor design. This note has presented two popular A/D techniques used in conjunction with the M6800 microprocessor system. These techniques, successive approximation and dual ramp, were shown using the MPU as the digital control element for the A/D system. This required dedication of the MPU to the A/D function during the conversion. Also shown were systems using the MPU to control the flow of data from an external A/D allowing the MPU to perform other tasks during the conversion.

The variety of programs presented allow the designer to make a selection based upon hardware cost, conversion speed, memory locations and interrupt capability. Although the A/D programs shown here are complete designs, they are general designs and may be tailored to fit each individual application. Also a variety of digital outputs are available including binary, BCD, and 7-segment. In conjunction with the BCD output a 16-bit binary to BCD conversion routine is presented in Appendix B.

REFERENCES

Aldridge, Don: "Autopolarity Circuits for the MC1405 Dual-Slope A-D Converter System", EB-35, Motorola Semiconductor Products Inc.

Aldridge, Don: "Input Buffer Circuits for the MC1505 Dual Ramp A-to-D Converter Subsystem", EB-24, Motorola Semiconductor Products Inc.

Kelley, Steve: "4½-Digit DVM System Using the MC1505 Dual-Slope Converter", EB-36, Motorola Semiconductor Products Inc.

M6800 Microprocessor Applications Manual, Motorola Semiconductor Products Inc.

M6800 Microprocessor Programming Manual, Motorola Semiconductor Products Inc.

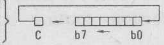
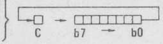
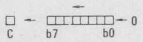
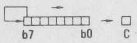
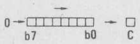
MC1505/1405 Data Sheet, Motorola Semiconductor Products Inc.

MC6800, MC6820 Data Sheets, *M6800 Systems Reference and Data Sheets*, Motorola Semiconductor Products Inc.

MC14435 Data Sheet, Motorola Semiconductor Products Inc.

APPENDIX A
MPU INSTRUCTIONS

Accumulator and Memory Instructions

		ADDRESSING MODES															BOOLEAN/ARITHMETIC OPERATION	COND. CODE REG.					
		IMMED			DIRECT			INDEX			EXTND			IMPLIED			(All register labels refer to contents)	5	4	3	2	1	0
OPERATIONS	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#		H	I	N	Z	V	C
Add	ADDA	8B	2	2	9B	3	2	AB	5	2	BB	4	3				A + M → A	↕	●	↕	↕	↕	↕
	ADDB	CB	2	2	DB	3	2	EB	5	2	FB	4	3				B + M → B	↕	●	↕	↕	↕	↕
Add Acmltrs	ABA													1B	2	1	A + B → A	↕	●	↕	↕	↕	↕
Add with Carry	ADCA	89	2	2	99	3	2	A9	5	2	B9	4	3				A + M + C → A	↕	●	↕	↕	↕	↕
	ADCB	C9	2	2	D9	3	2	E9	5	2	F9	4	3				B + M + C → B	↕	●	↕	↕	↕	↕
And	ANDA	84	2	2	94	3	2	A4	5	2	B4	4	3				A · M → A	●	●	↕	↕	R	●
	ANDB	C4	2	2	D4	3	2	E4	5	2	F4	4	3				B · M → B	●	●	↕	↕	R	●
Bit Test	BITA	85	2	2	95	3	2	A5	5	2	B5	4	3				A · M	●	●	↕	↕	R	●
	BITB	C5	2	2	D5	3	2	E5	5	2	F5	4	3				B · M	●	●	↕	↕	R	●
Clear	CLR							6F	7	2	7F	6	3				00 → M	●	●	R	S	R	R
	CLRA													4F	2	1	00 → A	●	●	R	S	R	R
	CLRB													5F	2	1	00 → B	●	●	R	S	R	R
Compare	CMPA	81	2	2	91	3	2	A1	5	2	B1	4	3				A − M	●	●	↕	↕	↕	↕
	CMPB	C1	2	2	D1	3	2	E1	5	2	F1	4	3				B − M	●	●	↕	↕	↕	↕
Compare Acmltrs	CBA													11	2	1	A − B	●	●	↕	↕	↕	↕
Complement, 1's	COM							63	7	2	73	6	3				$\overline{M}$ → M	●	●	↕	↕	R	S
	COMA													43	2	1	$\overline{A}$ → A	●	●	↕	↕	R	S
	COMB													53	2	1	$\overline{B}$ → B	●	●	↕	↕	R	S
Complement, 2's (Negate)	NEG							60	7	2	70	6	3				00 − M → M	●	●	↕	↕	①	②
	NEGA													40	2	1	00 − A → A	●	●	↕	↕	①	②
	NEGB													50	2	1	00 − B → B	●	●	↕	↕	①	②
Decimal Adjust, A	DAA													19	2	1	Converts Binary Add. of BCD Characters into BCD Format	●	●	↕	↕	↕	③
Decrement	DEC							6A	7	2	7A	6	3				M − 1 → M	●	●	↕	↕	4	●
	DECA													4A	2	1	A − 1 → A	●	●	↕	↕	4	●
	DECB													5A	2	1	B − 1 → B	●	●	↕	↕	4	●
Exclusive OR	EORA	88	2	2	98	3	2	A8	5	2	B8	4	3				A⊕M → A	●	●	↕	↕	R	●
	EORB	C8	2	2	D8	3	2	E8	5	2	F8	4	3				B⊕M → B	●	●	↕	↕	R	●
Increment	INC							6C	7	2	7C	6	3				M + 1 → M	●	●	↕	↕	⑤	●
	INCA													4C	2	1	A + 1 → A	●	●	↕	↕	⑤	●
	INCB													5C	2	1	B + 1 → B	●	●	↕	↕	⑤	●
Load Acmltr	LDAA	86	2	2	96	3	2	A6	5	2	B6	4	3				M → A	●	●	↕	↕	R	●
	LDAB	C6	2	2	D6	3	2	E6	5	2	F6	4	3				M → B	●	●	↕	↕	R	●
Or, Inclusive	ORAA	8A	2	2	9A	3	2	AA	5	2	BA	4	3				A + M → A	●	●	↕	↕	R	●
	ORAB	CA	2	2	DA	3	2	EA	5	2	FA	4	3				B + M → B	●	●	↕	↕	R	●
Push Data	PSHA													36	4	1	A → M_{SP}, SP − 1 → SP	●	●	●	●	●	●
	PSHB													37	4	1	B → M_{SP}, SP − 1 → SP	●	●	●	●	●	●
Pull Data	PULA													32	4	1	SP + 1 → SP, M_{SP} → A	●	●	●	●	●	●
	PULB													33	4	1	SP + 1 → SP, M_{SP} → B	●	●	●	●	●	●
Rotate Left	ROL							69	7	2	79	6	3				M 	●	●	↕	↕	⑥	↕
	ROLA													49	2	1	A	●	●	↕	↕	⑥	↕
	ROLB													59	2	1	B	●	●	↕	↕	⑥	↕
Rotate Right	ROR							66	7	2	76	6	3				M 	●	●	↕	↕	⑥	↕
	RORA													46	2	1	A	●	●	↕	↕	⑥	↕
	RORB													56	2	1	B	●	●	↕	↕	⑥	↕
Shift Left, Arithmetic	ASL							68	7	2	78	6	3				M 	●	●	↕	↕	⑥	↕
	ASLA													48	2	1	A	●	●	↕	↕	⑥	↕
	ASLB													58	2	1	B	●	●	↕	↕	⑥	↕
Shift Right, Arithmetic	ASR							67	7	2	77	6	3				M 	●	●	↕	↕	⑥	↕
	ASRA													47	2	1	A	●	●	↕	↕	⑥	↕
	ASRB													57	2	1	B	●	●	↕	↕	⑥	↕
Shift Right, Logic	LSR							64	7	2	74	6	3				M 	●	●	R	↕	⑥	↕
	LSRA													44	2	1	A	●	●	R	↕	⑥	↕
	LSRB													54	2	1	B	●	●	R	↕	⑥	↕
Store Acmltr.	STAA				97	4	2	A7	6	2	B7	5	3				A → M	●	●	↕	↕	R	●
	STAB				D7	4	2	E7	6	2	F7	5	3				B → M	●	●	↕	↕	R	●
Subtract	SUBA	80	2	2	90	3	2	A0	5	2	B0	4	3				A − M → A	●	●	↕	↕	↕	↕
	SUBB	C0	2	2	D0	3	2	E0	5	2	F0	4	3				B − M → B	●	●	↕	↕	↕	↕
Subtract Acmltrs.	SBA													10	2	1	A − B → A	●	●	↕	↕	↕	↕
Subtr. with Carry	SBCA	82	2	2	92	3	2	A2	5	2	B2	4	3				A − M − C → A	●	●	↕	↕	↕	↕
	SBCB	C2	2	2	D2	3	2	E2	5	2	F2	4	3				B − M − C → B	●	●	↕	↕	↕	↕
Transfer Acmltrs	TAB													16	2	1	A → B	●	●	↕	↕	R	●
	TBA													17	2	1	B → A	●	●	↕	↕	R	●
Test, Zero or Minus	TST							6D	7	2	7D	6	3				M − 00	●	●	↕	↕	R	R
	TSTA													4D	2	1	A − 00	●	●	↕	↕	R	R
	TSTB													5D	2	1	B − 00	●	●	↕	↕	R	R
																		H	I	N	Z	V	C

LEGEND:

OP	Operation Code (Hexadecimal);
~	Number of MPU Cycles;
#	Number of Program Bytes;
+	Arithmetic Plus;
−	Arithmetic Minus;
·	Boolean AND;
M_{SP}	Contents of memory location pointed to be Stack Pointer;
+	Boolean Inclusive OR;
⊕	Boolean Exclusive OR;
$\overline{M}$	Complement of M;
→	Transfer Into;
0	Bit = Zero;
00	Byte = Zero;

Note – Accumulator addressing mode instructions are included in the column for IMPLIED addressing

CONDITION CODE SYMBOLS:

H	Half-carry from bit 3;
I	Interrupt mask
N	Negative (sign bit)
Z	Zero (byte)
V	Overflow, 2's complement
C	Carry from bit 7
R	Reset Always
S.	Set Always
↕	Test and set if true, cleared otherwise
●	Not Affected

Index Register and Stack Manipulation Instructions

		IMMED			DIRECT			INDEX			EXTND			IMPLIED			BOOLEAN/ARITHMETIC OPERATION	COND. CODE REG. 5	4	3	2	1	0
POINTER OPERATIONS	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	BOOLEAN/ARITHMETIC OPERATION	H	I	N	Z	V	C
Compare Index Reg	CPX	8C	3	3	9C	4	2	AC	6	2	BC	5	3				$X_H - M, X_L - (M+1)$	●	●	⑦	↕	⑦	●
Decrement Index Reg	DEX													09	4	1	$X - 1 \rightarrow X$	●	●	●	↕	●	●
Decrement Stack Pntr	DES													34	4	1	$SP - 1 \rightarrow SP$	●	●	●	●	●	●
Increment Index Reg	INX													08	4	1	$X + 1 \rightarrow X$	●	●	●	↕	●	●
Increment Stack Pntr	INS													31	4	1	$SP + 1 \rightarrow SP$	●	●	●	●	●	●
Load Index Reg	LDX	CE	3	3	DE	4	2	EE	6	2	FE	5	3				$M \rightarrow X_H, (M+1) \rightarrow X_L$	●	●	⑨	↕	R	●
Load Stack Pntr	LDS	8E	3	3	9E	4	2	AE	6	2	BE	5	3				$M \rightarrow SP_H, (M+1) \rightarrow SP_L$	●	●	⑨	↕	R	●
Store Index Reg	STX				DF	5	2	EF	7	2	FF	6	3				$X_H \rightarrow M, X_L \rightarrow (M+1)$	●	●	⑨	↕	R	●
Store Stack Pntr	STS				9F	5	2	AF	7	2	BF	6	3				$SP_H \rightarrow M, SP_L \rightarrow (M+1)$	●	●	⑨	↕	R	●
Indx Reg → Stack Pntr	TXS													35	4	1	$X - 1 \rightarrow SP$	●	●	●	●	●	●
Stack Pntr → Indx Reg	TSX													30	4	1	$SP + 1 \rightarrow X$	●	●	●	●	●	●

Jump and Branch Instructions

		RELATIVE			INDEX			EXTND			IMPLIED				COND. CODE REG. 5	4	3	2	1	0
OPERATIONS	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#	BRANCH TEST	H	I	N	Z	V	C
Branch Always	BRA	20	4	2										None	●	●	●	●	●	●
Branch If Carry Clear	BCC	24	4	2										C = 0	●	●	●	●	●	●
Branch If Carry Set	BCS	25	4	2										C = 1	●	●	●	●	●	●
Branch If = Zero	BEQ	27	4	2										Z = 1	●	●	●	●	●	●
Branch If ≥ Zero	BGE	2C	4	2										N ⊕ V = 0	●	●	●	●	●	●
Branch If > Zero	BGT	2E	4	2										Z + (N ⊕ V) = 0	●	●	●	●	●	●
Branch If Higher	BHI	22	4	2										C + Z = 0	●	●	●	●	●	●
Branch If ≤ Zero	BLE	2F	4	2										Z + (N ⊕ V) = 1	●	●	●	●	●	●
Branch If Lower Or Same	BLS	23	4	2										C + Z = 1	●	●	●	●	●	●
Branch If < Zero	BLT	2D	4	2										N ⊕ V = 1	●	●	●	●	●	●
Branch If Minus	BMI	2B	4	2										N = 1	●	●	●	●	●	●
Branch If Not Equal Zero	BNE	26	4	2										Z = 0	●	●	●	●	●	●
Branch If Overflow Clear	BVC	28	4	2										V = 0	●	●	●	●	●	●
Branch If Overflow Set	BVS	29	4	2										V = 1	●	●	●	●	●	●
Branch If Plus	BPL	2A	4	2										N = 0	●	●	●	●	●	●
Branch To Subroutine	BSR	8D	8	2										See Special Operations	●	●	●	●	●	●
Jump	JMP				6E	4	2	7E	3	3					●	●	●	●	●	●
Jump To Subroutine	JSR				AD	8	2	BD	9	3					●	●	●	●	●	●
No Operation	NOP										02	2	1	Advances Prog. Cntr. Only	●	●	●	●	●	●
Return From Interrupt	RTI										3B	10	1		⑩					
Return From Subroutine	RTS										39	5	1	See Special Operations	●	●	●	●	●	●
Software Interrupt	SWI										3F	12	1		●	●	●	●	●	●
Wait for Interrupt	WAI										3E	9	1		●	⑪	●	●	●	●

Condition Code Register Manipulation Instructions

		IMPLIED				COND. CODE REG. 5	4	3	2	1	0
OPERATIONS	MNEMONIC	OP	~	#	BOOLEAN OPERATION	H	I	N	Z	V	C
Clear Carry	CLC	0C	2	1	0 → C	●	●	●	●	●	R
Clear Interrupt Mask	CLI	0E	2	1	0 → I	●	R	●	●	●	●
Clear Overflow	CLV	0A	2	1	0 → V	●	●	●	●	R	●
Set Carry	SEC	0D	2	1	1 → C	●	●	●	●	●	S
Set Interrupt Mask	SEI	0F	2	1	1 → I	●	S	●	●	●	●
Set Overflow	SEV	0B	2	1	1 → V	●	●	●	●	S	●
Acmltr A → CCR	TAP	06	2	1	A → CCR	⑫					
CCR → Acmltr A	TPA	07	2	1	CCR → A	●	●	●	●	●	●

CONDITION CODE REGISTER NOTES:

(Bit set if test is true and cleared otherwise)

1 (Bit V) Test: Result = 10000000?
2 (Bit C) Test: Result = 00000000?
3 (Bit C) Test: Decimal value of most significant BCD Character greater than nine? (Not cleared if previously set.)
4 (Bit V) Test: Operand = 10000000 prior to execution?
5 (Bit V) Test: Operand = 01111111 prior to execution?
6 (Bit V) Test: Set equal to result of N⊕C after shift has occurred.
7 (Bit N) Test: Sign bit of most significant (MS) byte = 1?
8 (Bit V) Test: 2's complement overflow from subtraction of MS bytes?
9 (Bit N) Test: Result less than zero? (Bit 15 = 1)
10 (All) Load Condition Code Register from Stack. (See Special Operations)
11 (Bit I) Set when interrupt occurs. If previously set, a Non-Maskable Interrupt is required to exit the wait state.
12 (All) Set according to the contents of Accumulator A.

APPENDIX B

BINARY-TO-BCD CONVERSION

A standard technique for binary-to-BCD conversion is that of the Add 3 algorithm. Figures B1 and B2 show a flow diagram and example of this algorithm. The technique requires a register containing the N-bit binary number and enough 4-bit BCD registers to contain the maximum equivalent BCD number for the initial binary number. The conversion starts by checking each BCD register for a value of 5 or greater. If this condition exists in one or all of these registers (initially this condition cannot exist), then a 3 is added to those registers where this condition exists. Next the registers are shifted left with the carry out of the previous register being the carry in to the next register. Again each BCD register is checked for values of 5 or greater. This sequence continues until the registers have been shifted N times, where N is the number of bits in the initial binary word. The BCD registers then contain the resulting BCD equivalent to the initial binary word. The example in Figure B2 starts with an 8-bit binary word consisting of all "1's." This word is converted to the BCD equivalent of 255 by this technique. After 8 shifts the last binary bit has been shifted out of the binary register and the hundreds, tens, and units registers contain a 255.

Figure B3 shows an MC6800 software routine for performing this technique of binary to BCD conversion. The initial binary number is a 16-bit number and occupies memory locations MSB and LSB; this binary number is converted to the equivalent BCD number in memory locations TENTSD, HNDTHD and UNTTEN. Each of these memory locations contains two BCD digits. Eighty-three memory locations are required for program storage with a maximum conversion taking 1.8 ms.

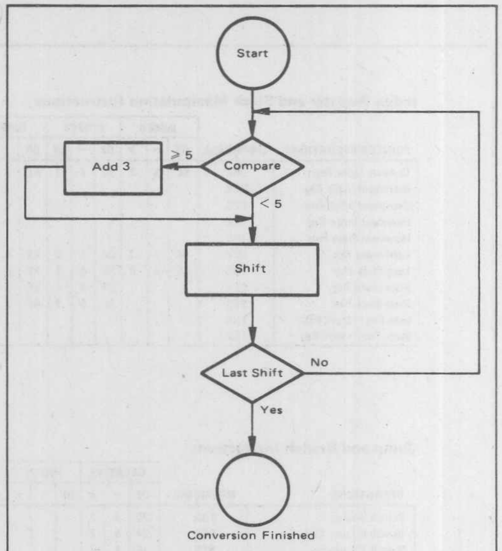
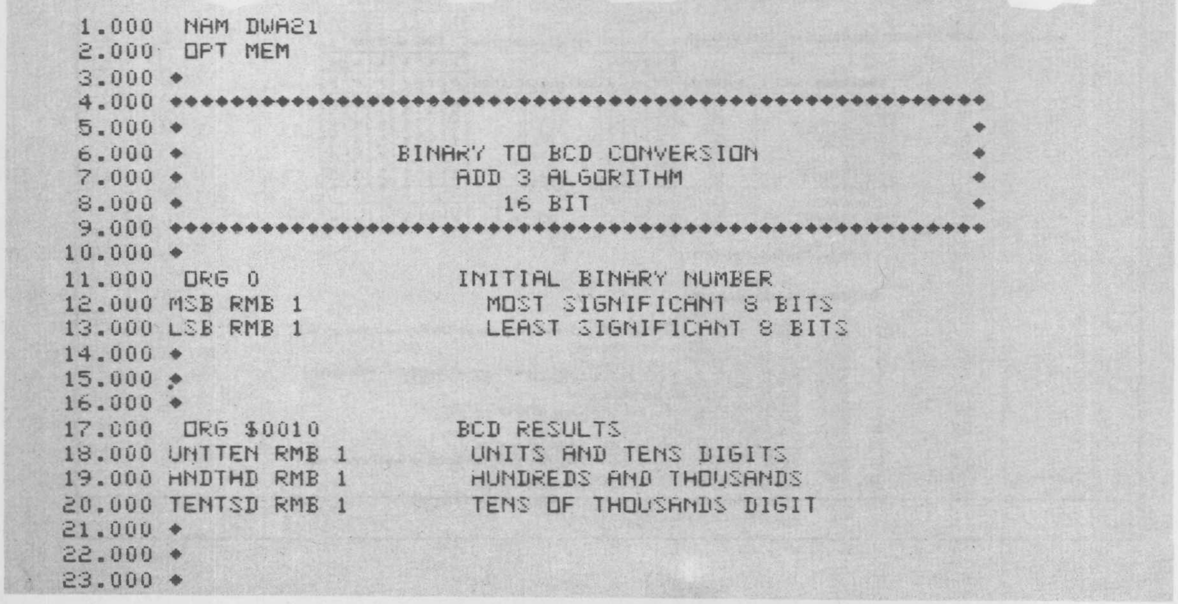


FIGURE B1 – Binary to BCD Conversion Flow Diagram

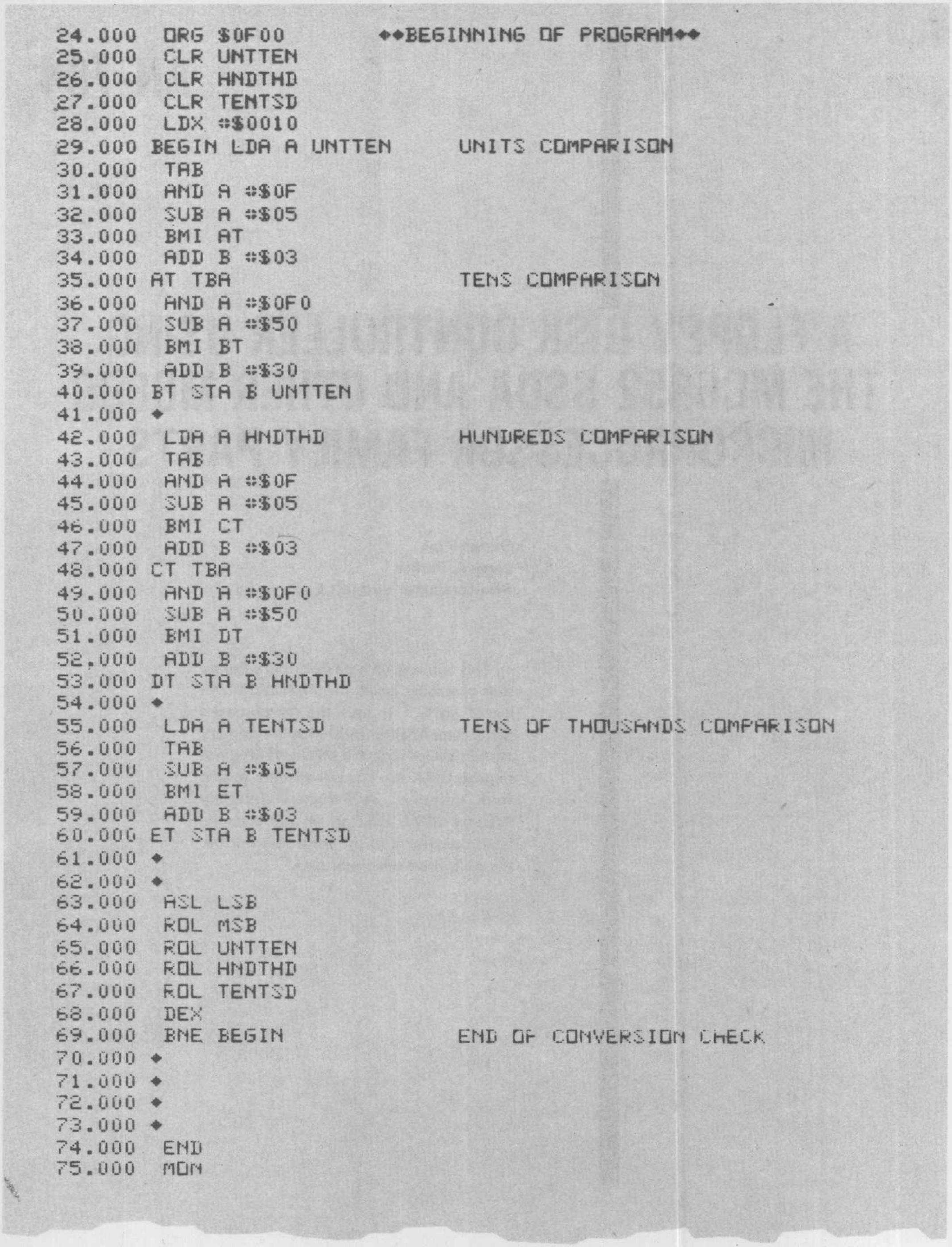
Hundreds	Tens	Units	8-Bit Binary	
			1 1 1 1 1 1 1 1	
		1	1 1 1 1 1 1 1	Shift
		1 1	1 1 1 1 1 1	Shift
		1 1 1	1 1 1 1 1	Shift
		1 0 1 0	1 1 1 1 1	Add 3 to Units
	1	0 1 0 1	1 1 1 1	Shift
	1	1 0 0 0	1 1 1 1	Add 3 to Units
	1 1	0 0 0 1	1 1 1	Shift
	1 1 0	0 0 1 1	1 1	Shift
	1 0 0 1	0 0 1 1	1 1	Add 3 to Tens
1	0 0 1 0	0 1 1 1	1	Shift
1	0 0 1 0	1 0 1 0	1	Add 3 to Units
1 0	0 1 0 1	0 1 0 1		Shift
2	5	5		Total Shifts 8

FIGURE B2 – Binary to BCD Conversion

FIGURE B3 – Binary-to-BCD Conversion Software (Page 1 of 2)



```
 NAM DWA21
 OPT MEM
*
**************************************************
*                                                *
*          BINARY TO BCD CONVERSION              *
*              ADD 3 ALGORITHM                   *
*                  16 BIT                        *
**************************************************
*
 ORG 0              INITIAL BINARY NUMBER
MSB RMB 1            MOST SIGNIFICANT 8 BITS
LSB RMB 1            LEAST SIGNIFICANT 8 BITS
*
*
*
 ORG $0010          BCD RESULTS
UNTTEN RMB 1         UNITS AND TENS DIGITS
HNDTHD RMB 1         HUNDREDS AND THOUSANDS
TENTSD RMB 1         TENS OF THOUSANDS DIGIT
*
*
*
```



```
 ORG $0F00         **BEGINNING OF PROGRAM**
 CLR UNTTEN
 CLR HNDTHD
 CLR TENTSD
 LDX #$0010
BEGIN LDA A UNTTEN      UNITS COMPARISON
 TAB
 AND A #$0F
 SUB A #$05
 BMI AT
 ADD B #$03
AT TBA                  TENS COMPARISON
 AND A #$0F0
 SUB A #$50
 BMI BT
 ADD B #$30
BT STA B UNTTEN
*
 LDA A HNDTHD           HUNDREDS COMPARISON
 TAB
 AND A #$0F
 SUB A #$05
 BMI CT
 ADD B #$03
CT TBA
 AND A #$0F0
 SUB A #$50
 BMI DT
 ADD B #$30
DT STA B HNDTHD
*
 LDA A TENTSD           TENS OF THOUSANDS COMPARISON
 TAB
 SUB A #$05
 BMI ET
 ADD B #$03
ET STA B TENTSD
*
*
 ASL LSB
 ROL MSB
 ROL UNTTEN
 ROL HNDTHD
 ROL TENTSD
 DEX
 BNE BEGIN              END OF CONVERSION CHECK
*
*
*
*
 END
 MON
```

FIGURE B3 – Binary-to-BCD Conversion Software (Page 2 of 2)

AN-764

A FLOPPY DISK CONTROLLER USING THE MC6852 SSDA AND OTHER M6800 MICROPROCESSOR FAMILY PARTS

Prepared by:
Larry A. Parker
Semiconductor Systems Engineering

This application note describes a floppy disk controller based on the M6800 family of parts. It uses the Synchronous Serial Data Adapter (SSDA) as the primary data interface with the MPU and does not require DMA for transfer of data to and from memory. A Peripheral Interface Adapter (PIA) controls all non-date related operations in the controller (including seek, drive selection, etc.).

A FLOPPY DISK CONTROLLER USING THE MC6852 SSDA AND OTHER M6800 MICROPROCESSOR FAMILY PARTS

INTRODUCTION

With the introduction of the MC6852 SSDA, the task of interfacing synchronous serial peripherals such as floppy disks, tape cassettes or cartridges, and bi-sync or HDLC data channels, has been reduced significantly.

Described in this application note is an efficient and flexible floppy disk controller design. Various features of this design include:

- Controller operates one to four daisy-chained drives
- Four drive radial configuration possible with additional multiplexing
- Flexible drive interfacing
- MPU controls data transfer allowing:
  - Store only desired data from a sector into memory
  - Search disk for pattern match without transferring data into memory until pattern is found
  - Read or write entire track in one revolution; consecutive tracks on consecutive revolutions
- DMA not required when using host MPU
- Interrupt MPU system operations on address mark match to start operations, allowing increased throughput
- Seek interlaced with R/W when using radial configuration
- Hard or soft sectoring
- IBM or user programmable sync patterns and format
- Write format blank disks
- Cost competitive
- Effective use of MPU leaves time available for additional tasks (see Table 1).
- Low parts count

Controller: (MPU and RAM shared with system)

Formatter		14 TTL SSI, MSI Devices 1 SSDA 1 PIA 1 CRCCG
Data Recovery		5 TTL + filter SSI, MSI Devices
Drive Interface		
	Buffers and Receivers	5-10 TTL/CMOS Devices + Termination
	MPU System Interface	2-5 TTL/Three-state Devices

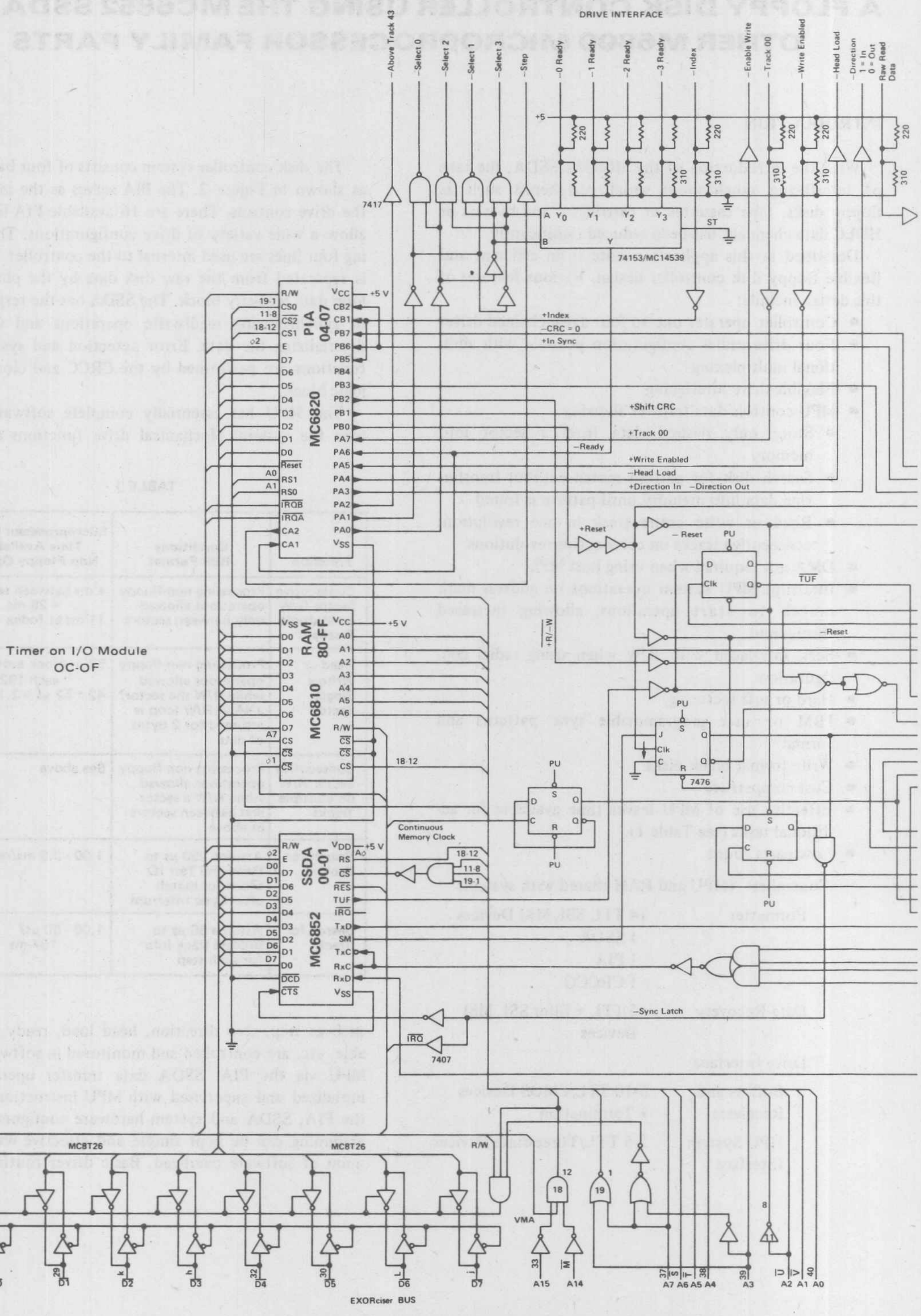
The disk controller system consists of four basic blocks as shown in Figure 2. The PIA serves as the interface to the drive controls. There are 16 available PIA lines which allow a wide variety of drive configurations. The remaining four lines are used internal to the controller. The clock is separated from the raw disk data by the phase-locked loop data recovery block. The SSDA has the responsibility of synchronizing read/write operations and serializing/deserializing the data. Error detection and system clock functions are performed by the CRCC and clock control logic block.

TABLE 1

Function	Conditions IBM Format	Microprocessor Processing Time Available for Non-Floppy Operations
Consecutive Sector R/W on Multiple Tracks	Processing non-floppy operations allowed only between sectors	1 ms between sectors = 25 ms 21.6% 11 ms at Index
Read or Write a Single Sector	Processing non-floppy operations allowed while R/W the sector; a 44 μs R/W loop is assumed for 2 bytes of data	52 μs block available each 192 μs 43.7% 42 · 52 μs = 2.184 ms
Consecutive Sector R/W on Multiple Tracks	Processing non-floppy operations allowed while R/W a sector and between sectors as above	See above 65.3%
Search for Sector	Assume 250 μs to Read and Test ID block for match after Sync Interrupt	1.00 - 3.9 ms/rev 96.1%
Search for Track	Assume 50 μs to process track info for each step	1.00 - 50 μs/ 167 ms 99.97%

The MPU has essentially complete software control over the system. Mechanical drive functions and status such as step, step direction, head load, ready, write enable, etc. are controlled and monitored in software by the MPU via the PIA. SSDA data transfer operations are initialized and supervised with MPU instructions. Due to the PIA, SSDA and system hardware configuration, programming can be kept simple and effective with a minimum of software overhead. Basic driver routines can be

FIGURE 1 – M6800 Floppy Disc Controller Schematic



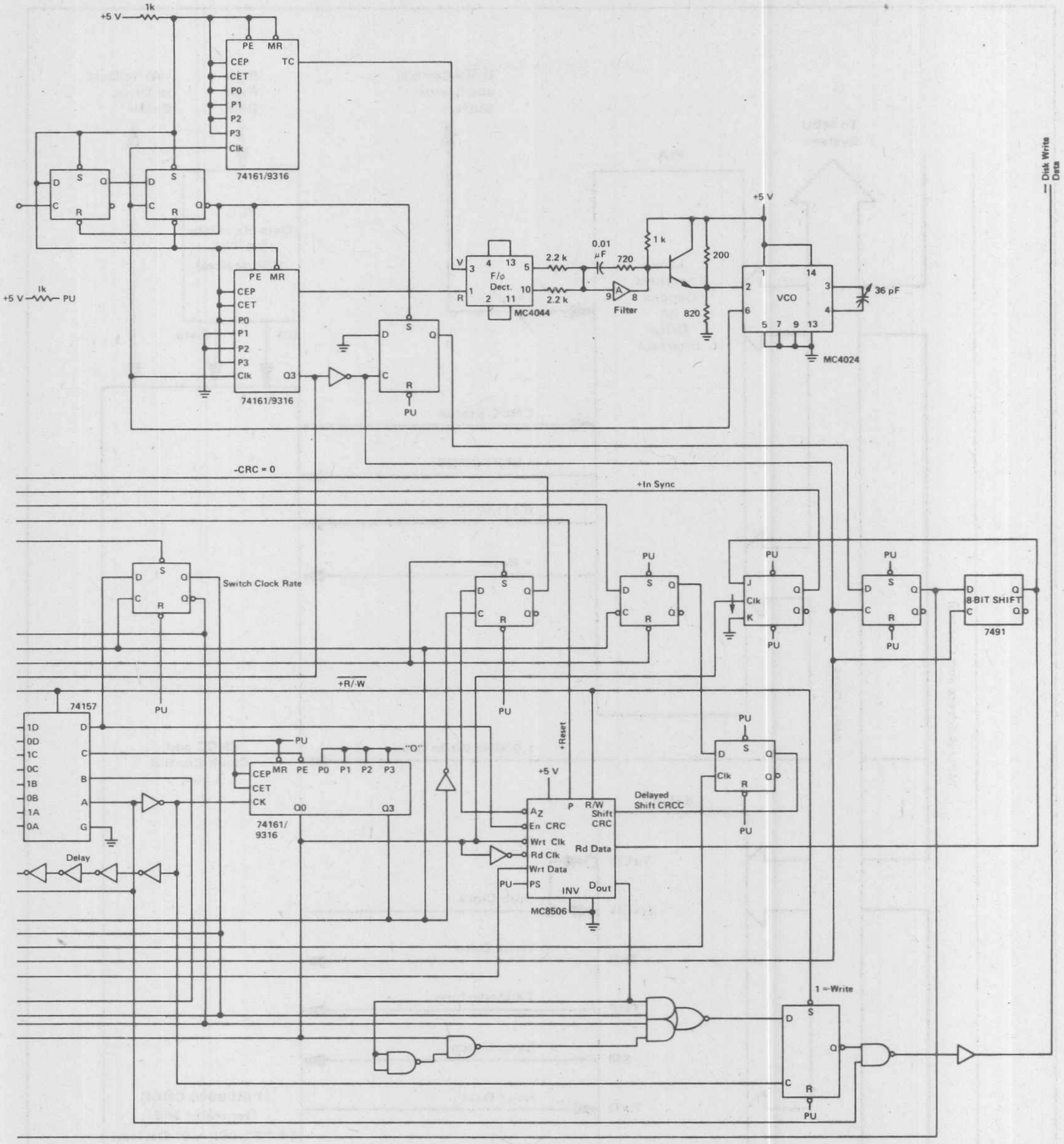
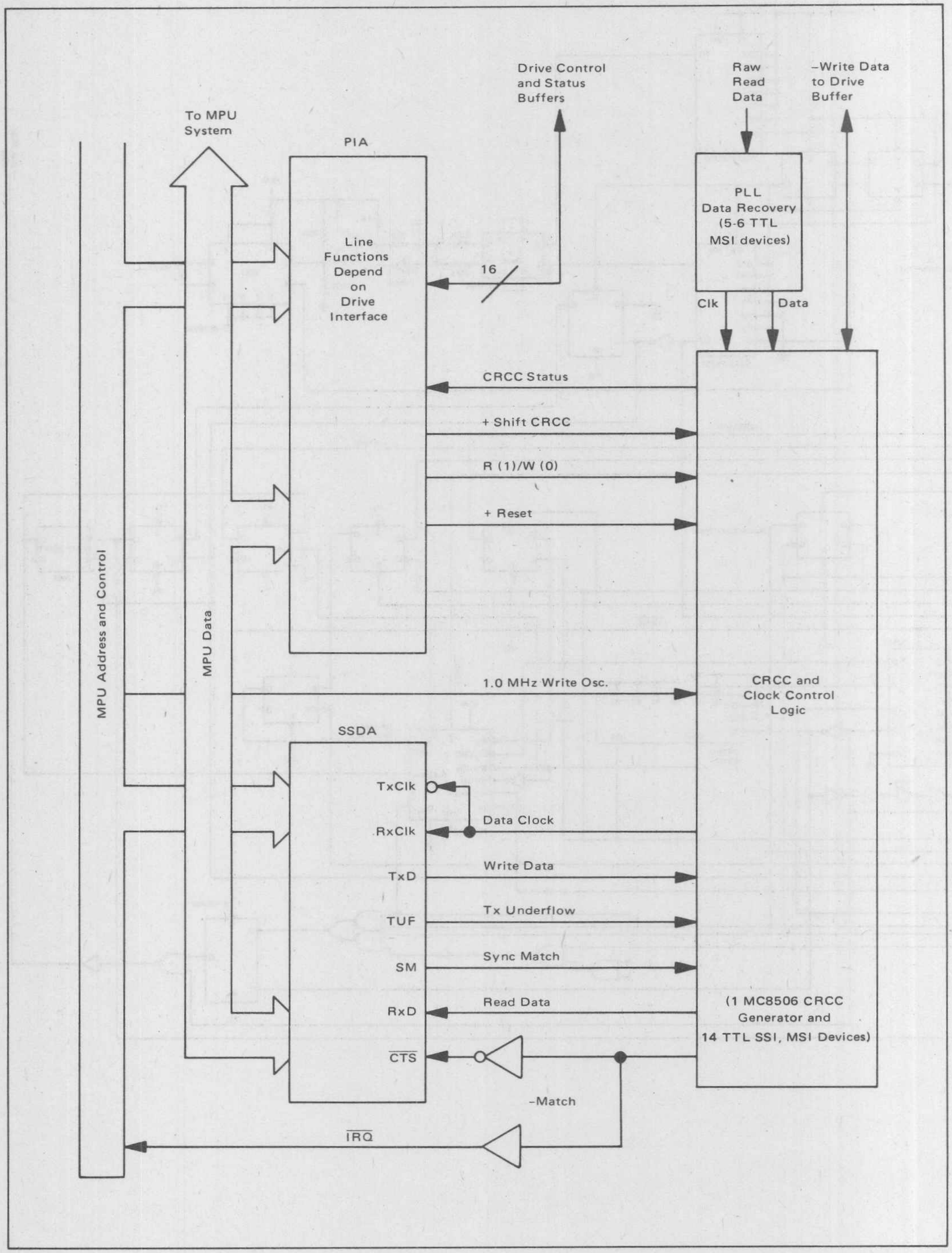

+5 V
1k
PE
MR
CEP
CET
P0
P1
P2
P3
Clk
TC
74161/9316
D
S
Q
C
R
+5 V
1k
PU
Q3
V
R
F/ϕ
Dect.
MC4044
2.2 k
0.01 μF
720
1 k
200
820
A
Filter
+5 V
VCO
MC4024
36 pF
Disk Write Data
-CRC = 0
+In Sync
Switch Clock Rate
J
Clk
K
8-BIT SHIFT
7491
+R/-W
74157
1D
0D
1C
0C
1B
0B
1A
0A
G
Delay
"O"
CK
Q0
74161/
9316
+Reset
+5 V
AZ
En CRC
Wrt Clk
Rd Clk
Wrt Data
PS
P
R/W
Shift CRC
Rd Data
INV
Dout
MC8506
Delayed
Shift CRCC
1 = Write

FIGURE 2—Floppy Disk Controller Block Diagram



written with fewer than 600 bytes of code. Operating systems suitable to most user needs can be done within two to four kilobytes.

Specific descriptions of the data recovery circuit and read and write operations are discussed in the following pages. Simplified logic diagrams are used in the circuit descriptions. The actual system schematic is shown in Figure 1.

It is important to be familiar with the operation of the SSDA, PIA and the IBM 3740 format in order to understand the controller design. A review of the MC6820 and MC6852 data sheets is recommended. A discussion of drive interfacing and IBM format can be found in the *M6800 Microprocessor Applications Manual.* A description of the software drivers and the software for the controller is available upon request.

DATA RECOVERY CIRCUIT DESCRIPTION

The raw data from the drive (clock and data) is terminated and buffered before clocking the first D flip-flop (Figure 3-B and Figure 4-B). Flip-Flops 1 and 2 generate a negative pulse 1 VCO period wide (Figure 3-D and Figure 4-D) which is used to load the reference counter with 9 and to set the data flip-flop 3.

IBM 3740 data can have only one consecutive pulse missing in the stream. By loading the reference counter with 9, Q3 will have a positive transition within 15 VCO periods generating a clock edge even if the data pulse is missing (Figure 3-E and Figure 4-E). Carryouts will occur every 2 μs ($16/f_O$), nominally providing a fundamental reference for the frequency/phase detector (Figure 3-F and Figure 4-F). The variable input to the frequency/phase detector is generated by dividing f_O by 16, using the carryout to give a pulse similar in duration to the reference.

Negative transitions on Q3 are inverted and clock flip-flop 3, whose output goes low (Figure 3-G, H and Figure 4-G, H). If a data pulse is present, the flip-flop is set by pulse from flip-flop 2 (Figure 3-D, Figure 4-D). If no data pulse is present, the output of flip-flop 3 remains low until set by a data pulse which must occur within 32 μs of the last one. The output of flip-flop 3 is then clocked one Q3 period later by Q3 to generate the NRZ data required by the formatter circuitry (Figure 3-J, Figure 4-J).

The 8-bit shift register provides 16 μs delayed data which is fed to the CRCCG. The SSDA clocks in 8 bits of data at 500 kHz before sync occurs and the read operation starts; because the sync data is included in the CRCC permutation, this sync data must be included in the CRCC field.

Phased-locked loop design is described in Motorola Application Note AN-535.

READ OPERATION

The sync code register of the SSDA is used to synchronize read operations by testing the incoming data stream, clocked at 500 kHz (2X clock), for the first half clock and data pattern of the desired address mark. When a match is found, the external circuitry is released by the Sync Match (SM) output and the second half of the address mark (clock and data) is read from the SSDA Rx FIFO (when it becomes available) and tested for a match with the desired type. If it does not match the sequence is restarted. If the second half of the address mark matches, the desired data transfer is initiated. The external circuitry switches the SSDA read clock to 250 kHz (1X clock) after the second half of the address mark has been received so that only the data portion of the remaining Rx FIFO information is recovered. The external circuitry also controls the CRCC generator (CRCCG) timing so that only the data portion of the recovered information is clocked into the generator.

After the data block has been transferred, the CRCC status is made available to the MPU for 32 μs at a PIA peripheral line.

READ DATA LOGIC

Figure 5 is a simplified logic diagram of the read data logic. Figure 6 is a timing diagram which shows the signal timing relationship when a read operation is begun.

This explanation of the read data logic assumes that system initialization has been completed. This includes the completion of the seek and head load operation. The enable read line is set and the formatter reset line has been toggled to reset the sync match latch and set the switch clock rate latch. These two previous operations are initiated in software and are executed via the system Peripheral Data Adapter (PIA). Initialization of the Synchronous Serial Data Adapter, SSDA, has been completed as described in the SSDA Read Preparation section.

Raw serial data is processed by the data recovery circuit which provides the separated read data and 500 kHz clock to the read data logic, Figure 5-A, B and Figure 6-A, B.

The 500 kHz clock from the data recovery circuit is inverted, delayed and then fed to the SSDA read clock input (RxD), via combinational AND/OR selector logic controlled by the switch clock rate latch, Figure 5, Figure 6-C. Inverting the clock provides the correctly phased positive transition to load the read data in the SSDA receiver shift register. The delay (4 inverters) is necessary to prevent a possible timing glitch which will be discussed later. The 500 kHz, 2X clock rate will load the receiver register with both clock and data bits from the read data line.

When the bit pattern loaded in the receiver shift register is equal to the pattern present in the SSDA sync code register, the SSDA synch match output, SM, will go high for one read clock period, Figure 6-D. When the sync match occurs, the SSDA receive data FIFO is internally enabled and will begin to store the read data, Figure 6-D.

FIGURE 3—Data Recovery Circuit

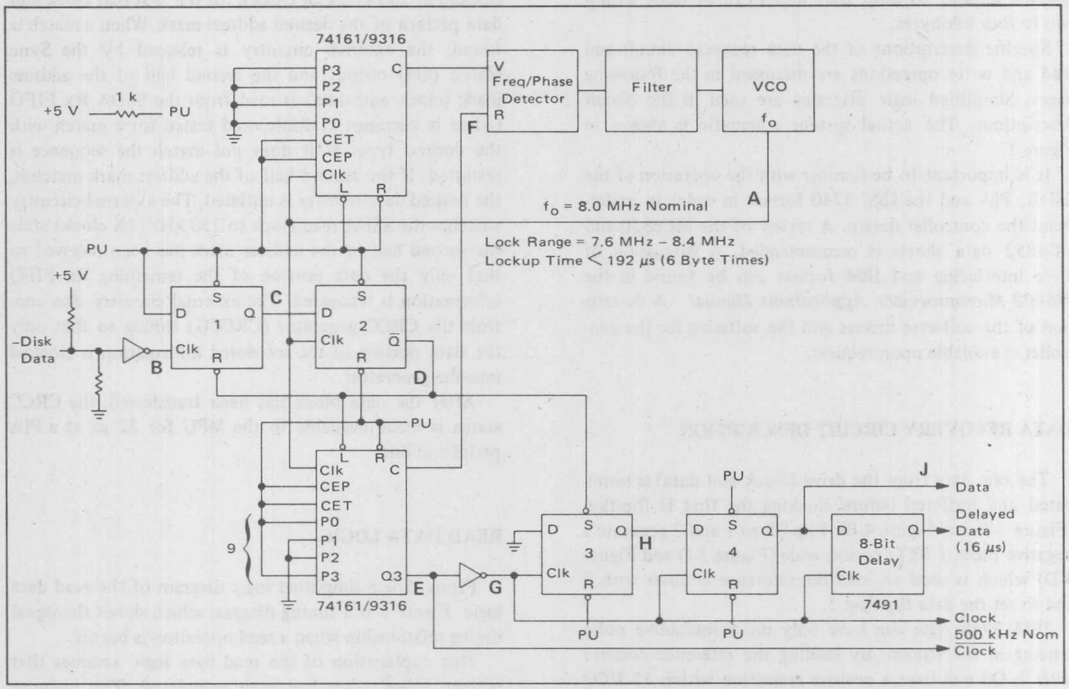


FIGURE 4—Data Recovery Timing

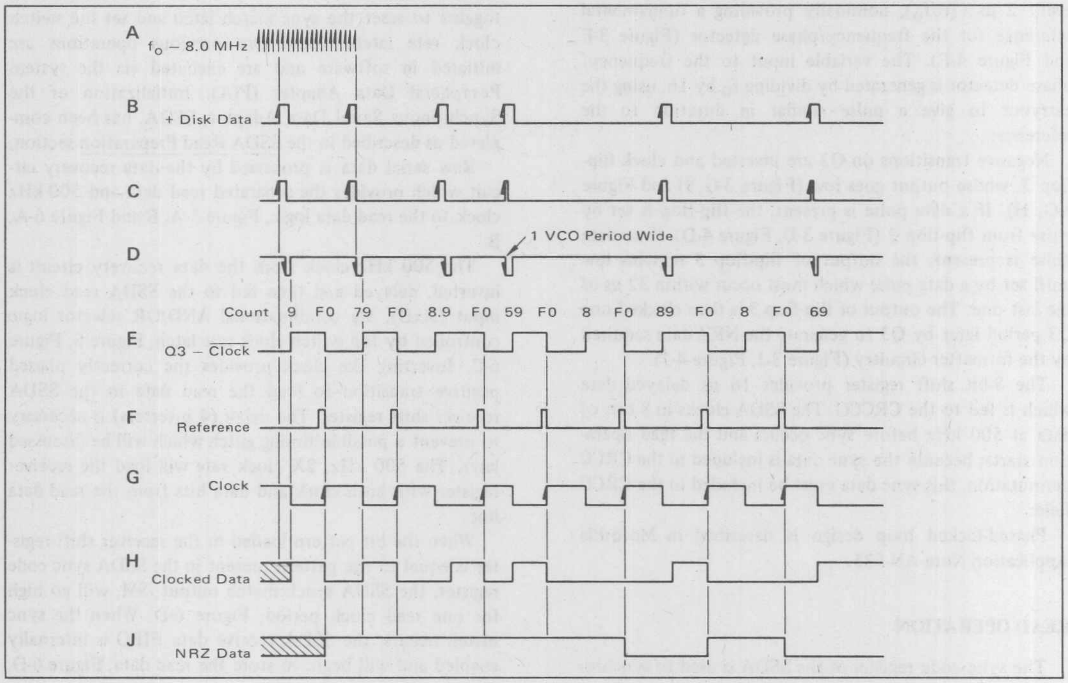


FIGURE 5—Read Data Simplified Logic Diagram

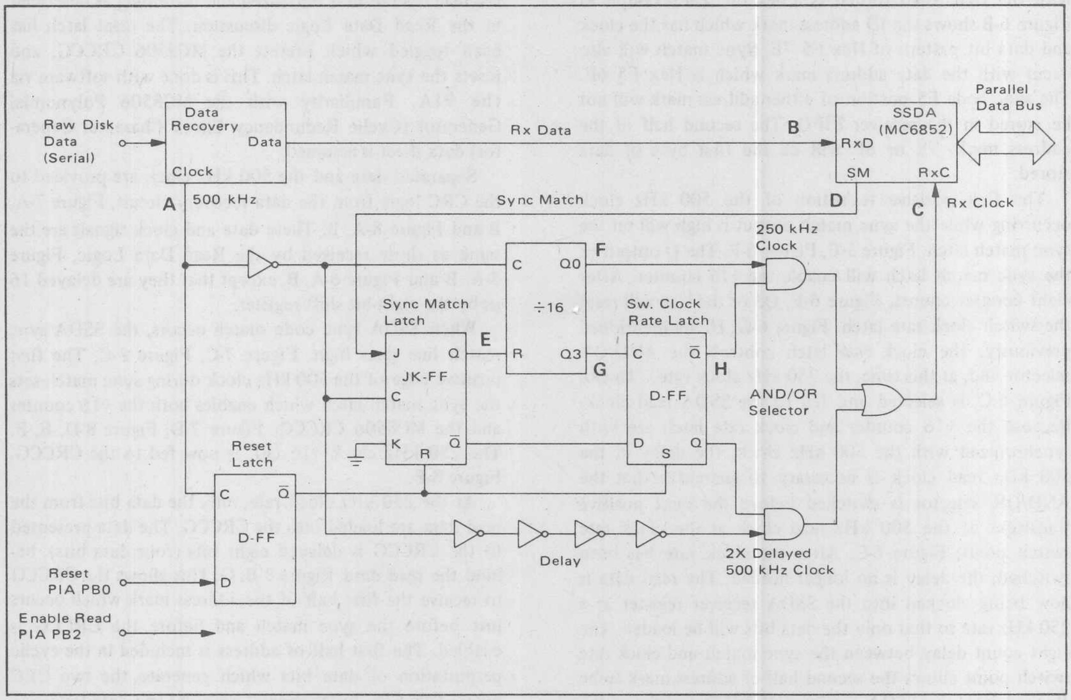
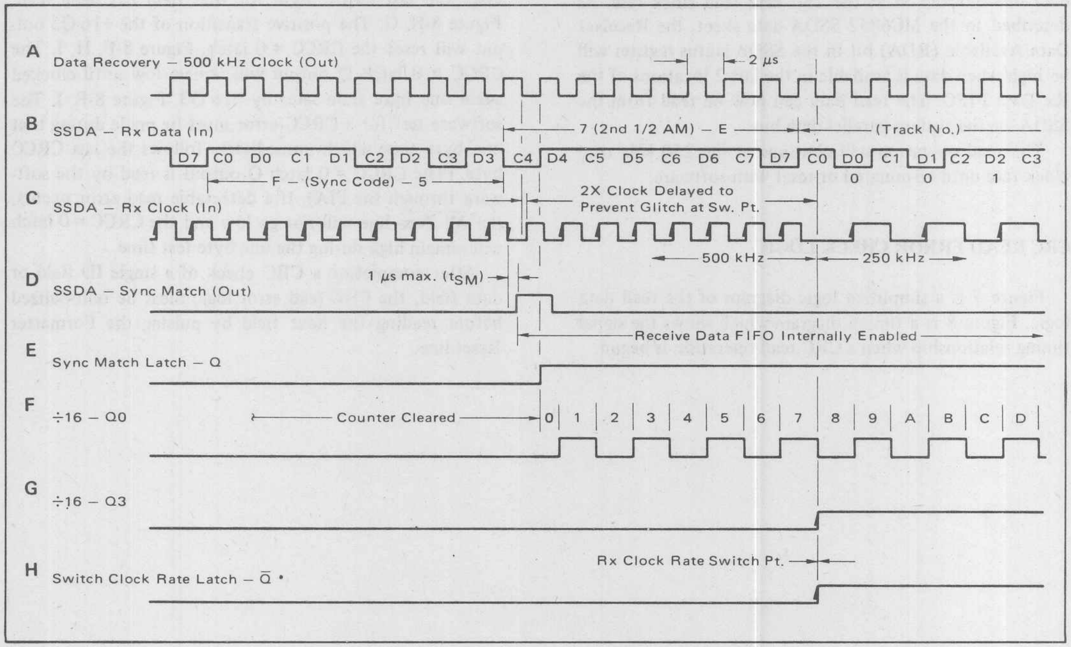


FIGURE 6—Read Data Logic Timing Diagram



A
Data Recovery — 500 kHz Clock (Out)
2 µs
B
SSDA — Rx Data (In)
7 (2nd 1/2 AM) — E
(Track No.)
D7 C0 D0 C1 D1 C2 D2 C3 D3 C4 D4 C5 D5 C6 D6 C7 D7 C0 D0 C1 D1 C2 D2 C3
0 1
F — (Sync Code) — 5
C
SSDA — Rx Clock (In)
2X Clock Delayed to Prevent Glitch at Sw. Pt.
500 kHz
250 kHz
D
SSDA — Sync Match (Out)
1 µs max. (t_{SM})
Receive Data FIFO Internally Enabled
E
Sync Match Latch — Q
F
÷16 — Q0
Counter Cleared
0 1 2 3 4 5 6 7 8 9 A B C D
G
÷16 — Q3
H
Switch Clock Rate Latch — $\overline{Q}$ •
Rx Clock Rate Switch Pt.

The clock and data bit pattern used for sync match with the IBM 3740 format is a hex F5. The example in Figure 6-B shows the ID address mark which has the clock and data bit pattern of Hex F5 7E. Sync match will also occur with the data address mark which is Hex F5 6F. The sync code F5 portion of either address mark will not be stored in the receiver FIFO. The second half of the address mark, 7E or 6F will be the first byte of data stored.

The first positive transition of the 500 kHz clock occurring while the sync match output is high will set the sync match latch, Figure 5-E, Figure 6-E. The Q output of the sync match latch will enable the ÷16 counter. After eight counter counts, Figure 6-F, Q3 of the ÷16 will reset the switch clock rate latch, Figure 6-G, H. As mentioned previously, the clock rate latch controls the AND/OR selector and, at this time, the 250 kHz clock rate, ÷16-Q0, Figure 6-C, is selected and fed to the SSDA read clock. Because the ÷16 counter and clock rate latch are both synchronized with the 500 kHz clock, the delay in the 500 kHz read clock is necessary to guarantee that the AND/OR selector is switched before the next positive transition of the 500 kHz read clock at the clock rate switch point, Figure 6-C. After the clock rate has been switched, the delay is no longer needed. The read data is now being clocked into the SSDA receiver register at a 250 kHz rate so that only the data bits will be loaded. The eight count delay between the sync match and clock rate switch point allows the second half of address mark to be clocked into the SSDA receive data FIFO register at the 2X clock rate.

The receive data FIFO will now continue to fill with data bits clocked in at the 250 kHz read clock rate. As described in the MC6852 SSDA data sheet, the Receiver Data Available (RDA) bit in the SSDA status register will be high when data is available in the last 2 locations of the Rx Data FIFO. The read data can now be read from the SSDA via the system parallel data bus.

The read operation will continue at the 250 kHz read clock rate until terminated or reset with software.

CRC READ ERROR CHECK LOGIC

Figure 7 is a simplified logic diagram of the read data logic. Figure 8 is a timing diagram which shows the signal timing relationship when a CRC read operation is begun.

This explanation of the CRC read logic assumes that the read operation is initialized and is running as described in the Read Data Logic discussion. The reset latch has been toggled which presets the MC8506 CRCCG, and resets the sync match latch. This is done with software via the PIA. Familiarity with the MC8506 Polynomial Generator (Cyclic Redundancy Check Character Generator) data sheet is assumed.

Separated data and the 500 kHz clock are provided to the CRC logic from the data recovery circuit, Figure 7-A, B and Figure 8-A, B. These data and clock signals are the same as those received by the Read Data Logic, Figure 5-A, B and Figure 6-A, B, except that they are delayed 16 μs by the eight-bit shift register.

When SSDA sync code match occurs, the SSDA sync match line goes high, Figure 7-C, Figure 8-C. The first positive edge of the 500 kHz clock during sync match sets the sync match latch which enables both the ÷16 counter and the MC8506 CRCCG, Figure 7-D, Figure 8-D, E, F. The 250 kHz clock ÷16 $\overline{\text{Q0}}$, is now fed to the CRCCG, Figure 8-E.

At the 250 kHz clock rate, only the data bits from the read data are loaded into the CRCCG. The data presented to the CRCCG is delayed eight bits (four data bits), behind the read data, Figure 8-B, G. This allows the CRCCG to receive the first half of the address mark which occurs just before the sync match and before the CRCCG is enabled. The first half of address is included in the cyclic permutation of data bits which generate the two CRC bytes. Two CRC bytes append every ID and data field.

If the complete address mark and ID or data field has been read correctly, the CRCCG $\overline{\text{All Zero}}$ line will go low after the last CRCC byte for that field has been read, Figure 8-H, G. The positive transition of the ÷16 $\overline{\text{Q3}}$ output will reset the CRCC = 0 latch, Figure 8-F, H, I. The CRCC = 0 latch Q output will remain low until clocked again one byte time later by ÷16 $\overline{\text{Q3}}$, Figure 8-F, I. The software test for a CRCC error must be made during that one byte time which immediately follows the last CRCC byte. (The CRCC = 0 latch Q output is read by the software through the PIA). If a detectable read error occurs, the $\overline{\text{All Zero}}$ line will not go low and the CRCC = 0 latch will remain high during the one byte test time.

After completing a CRC check of a single ID field or data field, the CRC read error logic must be reinitialized before reading the next field by pulsing the Formatter Reset line.

FIGURE 7—CRC Read Error Check—Simplified Circuit Diagram

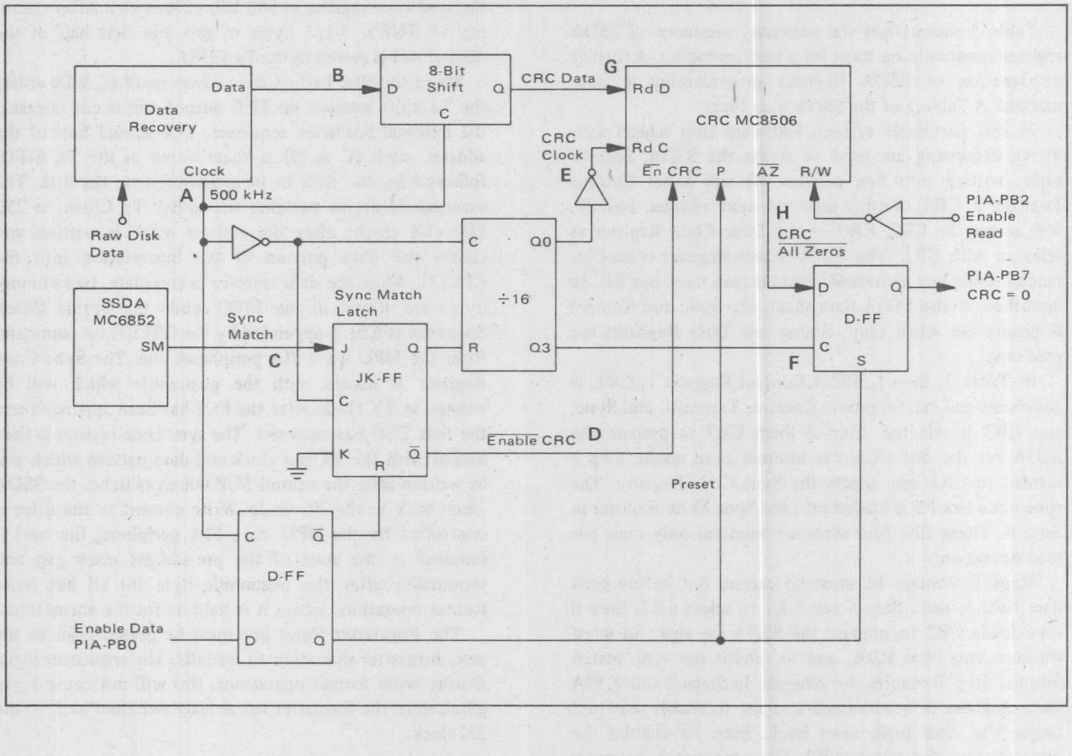
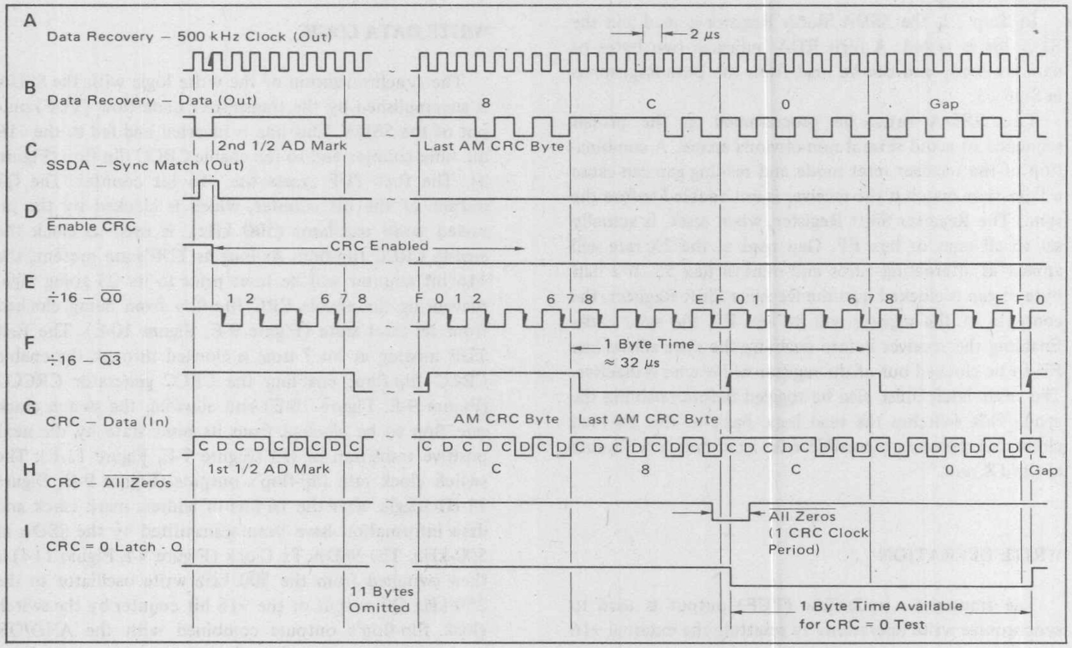


FIGURE 8—CRC Read Error Check—Timing Diagram



SSDA PREPARATION FOR READ

Table 2 summarizes the necessary sequence of SSDA register programming steps for a read operation. A further explanation of SSDA Register programming is summarized in Table 1 of the SSDA data sheet.

In this particular system, hardware chip selects with direct addressing are used to access the SSDA. Specifically, writing into hex address 00 will select Control Register 1, CR1. Writing into the next address, hex 01, will access the CR2, CR3 or the Sync Code Register as selected with CR1. The SSDA Status Register is read by reading from hex address 00. Data is read from hex 01. As described in the SSDA data sheet, the Sync and Control Registers are write only. Status and Data Registers are read only.

In Table 2, Step 1, SSDA Control Register 1, CR1, is addressed and set to inhibit Receive, Transmit, and Sync, and CR3 is selected. Step 2 loads CR3 to prepare the SSDA for the one character internal sync mode. Step 3 returns to CR1 and selects the Sync Code Register. The sync code hex F5 is loaded into the Sync Code Register in Step 4. These first four steps are required only once per read operation.

Steps 5 through 11 must be carried out before each new field is read. Step 5 sets CR1 to select CR2. Step 6 then loads CR2 to prepare the SSDA for eight bit word transfer, two byte RDA, and to inhibit the sync match output. Step 7 enables the receiver. In Steps 8 and 9, PIA Data Register B is addressed and set to enable read and toggle the read logic reset latch. Step 10 enables the internal sync and selects CR2. The sync match output is enabled in Step 11.

In Step 12, the SSDA Status Register is read and the RDA bit is tested. A high RDA indicates two bytes of data are ready and can be read from the Data Register as in Step 13.

The SSDA must be programmed in the proper sequence to avoid several non-obvious errors. A combination of the receiver reset mode and reading gap can cause a false sync match if the receiver is not enabled before the sync. The Receiver Shift Register, when reset, is actually set to all ones or hex FF. Gap read at the 2X rate will appear as alternating zeros and ones or hex 55. If a half byte of gap is clocked into the Receiver Shift Register, the contents of the register will be hex F5, the sync code. Enabling the receiver before enabling the sync allows hex FF to be clocked out of the register while sync is inactive. The reset latch must also be toggled before enabling the sync. This switches the read logic back to the 2X read clock and prevents a possible false sync match with data at the 1X rate.

WRITE OPERATION

The transmitter underflow (TUF) output is used to synchronize write operations by resetting the external ÷16 bit counter while writing the pre-address mark gap from the sync code register at 500 kHz (2X clock). After counting 11 TUF's, 5-1/2 bytes of gap, the first half of the desired AM is stored in the Tx FIFO.

When the first half of the address mark (C & D) enters the Tx Shift register, no TUF output will occur releasing the external hardware sequence. The second half of the address mark (C & D) is then stored in the Tx FIFO, followed by the data to be transferred to the disk. The external hardware switches the SSDA Tx Clock to 250 kHz (1X clock) after the address mark is written and clocks the data portion of the information into the CRCCG. When the data transfer is complete, two dummy bytes are stored in the FIFO while the Frame Check Sequence (FCS) is appended by the CRCCG on command from the MPU via a PIA peripheral line. The Sync Code Register is loaded with the postamble which will be written at 1X clock after the FCS has been appended and the first TUF has occurred. The sync code register is then loaded with the 2X gap clock and data pattern which will be written after the second TUF which switches the SSDA clock back to the 2X mode. Write current to the drive is controlled by the MPU via a PIA peripheral line and is initialed at the start of the pre-address mark gap and terminated after the postamble byte for all but write format operations where it is held on for the entire track.

The Formatter Reset line must be pulsed prior to the next formatter operation to initialize the sequencer logic. During write format operations, this will not cause a gap glitch since the formatter has already switched back to the 2X clock.

WRITE DATA LOGIC

The synchronization of the write logic with the SSDA is accomplished by the transmitter underflow (TUF) output of the SSDA. This line is inverted and fed to the ÷16 bit time counter and to the enable CRCC flip flop (Figure 9). The first TUF resets the ÷16 bit counter. The Q3 output of the bit counter, which is clocked by the inverted write oscillator (500 kHz), is used to clock the enable CRCC flip-flop. As long as TUF's are present, the ÷16 bit counter will be reset prior to its Q3 going high, preventing the enable CRC flip-flop from being clocked from its reset state (Figure 9-E, Figure 10-E). The first TUF missing at bit 7 time is clocked through the enable CRCC flip-flop, enabling the CRCC generator CRCCG (Figure 9-E, Figure 10-E) and allowing the switch clock rate flop to be clocked from its reset state by the next positive transition of Q3 (Figure 9-E, Figure 11-E). The switch clock rate flip-flop's outputs (Figure 9-H, Figure 11-H) toggle after the 16 bits of address mark clock and data information have been transmitted by the SSDA at 500 kHz. The SSDA Tx Clock (Figure 9-I, Figure 11-I) is then switched from the 500 kHz write oscillator to the 250 kHz Q0 output of the ÷16 bit counter by the switch clock flip-flop's outputs combined with the AND/OR

TABLE 2 – SSDA PREPARATION FOR READ

	Register		Data								Function & Comments
Step	Address	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
1	SSDACR CR1		Rx Rs	Tx Rs	Strip Sync	Clear Sync	TIE	RIE	AC1	AC2	Inhibit: Sync Tx Rx
	00	W	1	1	0	1	0	0	1	0	Select CR3
2	SSDADR CR3		Intrn Sync	1 Sync Char	Clear $\overline{\text{CTS}}$	Clear TUF	X	X	X	X	1 Sync Character Internal Sync Clear: TUF $\overline{\text{CTS}}$
	01	W	0	1	1	1	0	0	0	0	
3	SSDACR CR1		Rx Rs	Tx Rs	Strip Sync	Clear Sync	TIE	RIE	AC1	AC2	Select Sync Code Register
	00	W	1	1	0	1	0	0	0	1	
4	SSDADR Sync Code Reg		D7	D6	D5	D4	D3	D2	D1	D0	Set Sync Code to hex F5
	01	W	1	1	1	1	0	1	0	1	
5	SSDACR CR1		Rx Rs	Tx Rs	Strip Sync	Clear Sync	TIE	RIE	AC1	AC2	Select CR2
	00	W	1	1	0	1	0	0	0	0	
6	SSDADR CR2		PC1	PC2	2 Byte RDA	WS1	WS2	WS3	Tx Syn	EIE	Inhibit SM 2-Byte RDA 8-Bit Word
	01	W	1	1	0	1	1	0	0	0	
7	SSDACR CR1		Rx	Tx Rs	Strip Sync	Clear Sync	TIE	RIE	AC1	AC2	Enable Read
	00	W	0	1	0	1	0	0	0	0	
8	PIADRB		CRC = 0 (In)	Int Sync (In)	Shift CRC (Out)	Index Drive (In)	X	Enable Read (Out)	Enable Write (Drive)	Reset (Out)	Enable Read Logic Toggle Reset High
	05	W	0	0	0	0	0	1	1	1	
9	PIADRB										Toggle Reset Low
	05	W	0	0	0	0	0	1	1	0	
10	SSDACR CR1		Rx	Tx Rs	Strip Sync	Clear Sync	TIE	RIE	AC1	AC2	Enable Sync Select CR2
	00	W	0	1	0	0	0	0	0	0	
11	SSDADR CR2		PC1	PC2	2 Byte RDA	WS1	WS2	WS3	Tx Sync	EIE	Enable SM Output
	01	W	1	0	0	1	1	0	0	0	
12	SSDASR Status Reg		RDA	TDRA	DCD	$\overline{\text{CTS}}$	TUF	Rx DVR	PE	IRQ	Test RDA for 2 Bytes Ready
	00	R	0→1	X	X	X	X	X	X	X	
13	SSDADR 01	R	0	1	1	1	1	1	1	0	Read Data Exp: $7E_{16}$ = 2nd ½ IDAM

logic. The switch clock rate flip-flop's outputs also control the selection of 2X write data and clock or 1X write data and clock being fed to the write data formatting circuit (Figure 9-R).

The CRCCG has been clocked from the first missing TUF by the ÷16 Q0 output so that only the data portion of the transferred information is accumulated during the write data operation, including the data portion of the address mark (Figure 9-D, Figure 10-D).

Once the last data byte has been transferred from the SSDA's Tx FIFO into the Tx Shifter, the MPU enables the shift CRCC line via the PIA (Figure 13-K and 14-K). This signal is then clocked by the next positive transition of Q3 at the end of the last data byte. The shift CRCC command to the CRCCG is then delayed 1 μs (Figure 13-P, S, A; Figure 14-P, S, A; Figure 15-P, S, A) by the write oscillator clock to allow for the last data bit to be transferred into the CRCCG registers before switching into the shift mode.

Two dummy data bytes are written from the SSDA while FCS is being appended to the data field. This allows the MPU to keep track of the FCS status and disable the Shift CRCC command at the proper time, i.e., while the last dummy byte is to be shifted out of the SSDA. The disable shift CRCC command is clocked by the positive transition of Q3 at the end of the second dummy byte and is again delayed 1 μs before reaching the CRCCG. This switches the CRCCG data out back to the SSDA TxD.

When the last bit of the second dummy byte is being transmitted, a TUF occurs indicating that the Tx FIFO is empty and the content of the sync code register will be transmitted next. The sync code register was previously loaded with the 1X postamble data field and it will immediately follow the FCS field. The first TUF following the FCS is clocked by the positive transition Q3 to disable the CRCCG (Figure 13-C, E, F; Figure 16-C, E, F) starting the write termination sequence. The next positive transition of Q3 restores the clock to the 2X mode which allows the second TUF to reset the ÷16 bit counter (Figure 13-F, H, N; Figure 16-F, H, N). While the postamble is being written, the sync code register is loaded with the 2X gap clock and data pattern which will be written until the FIFO is loaded with the next address mark restarting the operation or the transmitter section is reset by software. Using this technique, the entire track may be formatted without write current to the drive being shut off and without any glitches at the switchover points.

TUF's may be counted to determine gap sizes when write formatting the disk.

The write data format logic takes the NRZ clock and data information (Figure 9-R and Figure 12-R) and generates the raw unseparated clock and data format required by the drive electronics (Figure 9-M and Figure 12-M). The NRZ data is clocked and delayed for one-half a write oscillator period (Figure 9-L and Figure 12-L) to remove any delays generated in the previous logic. The inverted and delayed NRZ data is NANDed with the 500 kHz write oscillator (Figure 9-A and Figure 12-A) to generate the -Write Data for the drive electronics.

ACKNOWLEDGEMENT

Special appreciation is given to Mark Eidson who is responsible for the original text of this article. Also, credit and thanks for initial circuit design are given to Mark Eidson and Tom Daly.

FIGURE 9—Write Synchronization Simplified Circuit Diagram

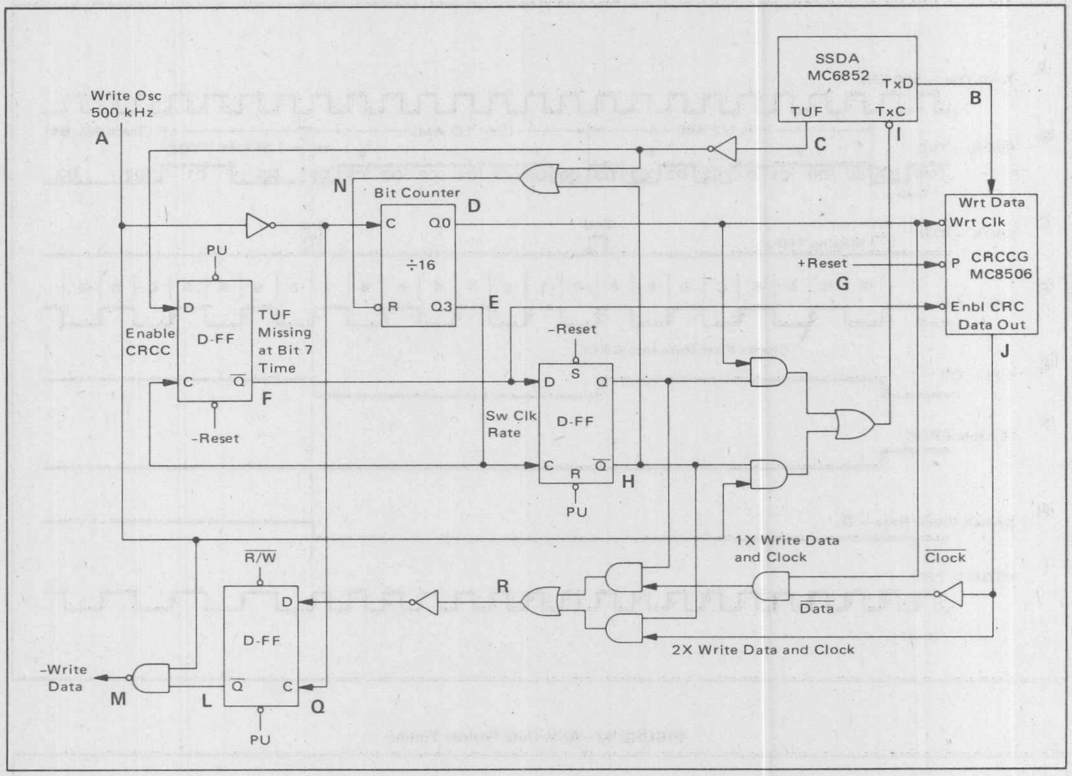


FIGURE 10—Write Synchronization Timing Diagram

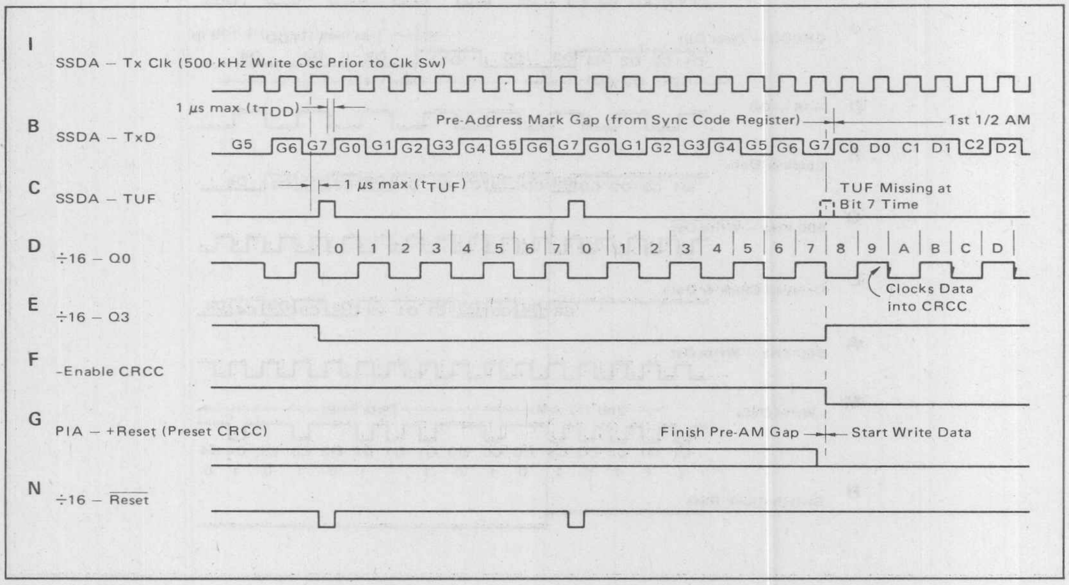


FIGURE 11—Write Switch Clock Rate Timing Diagram

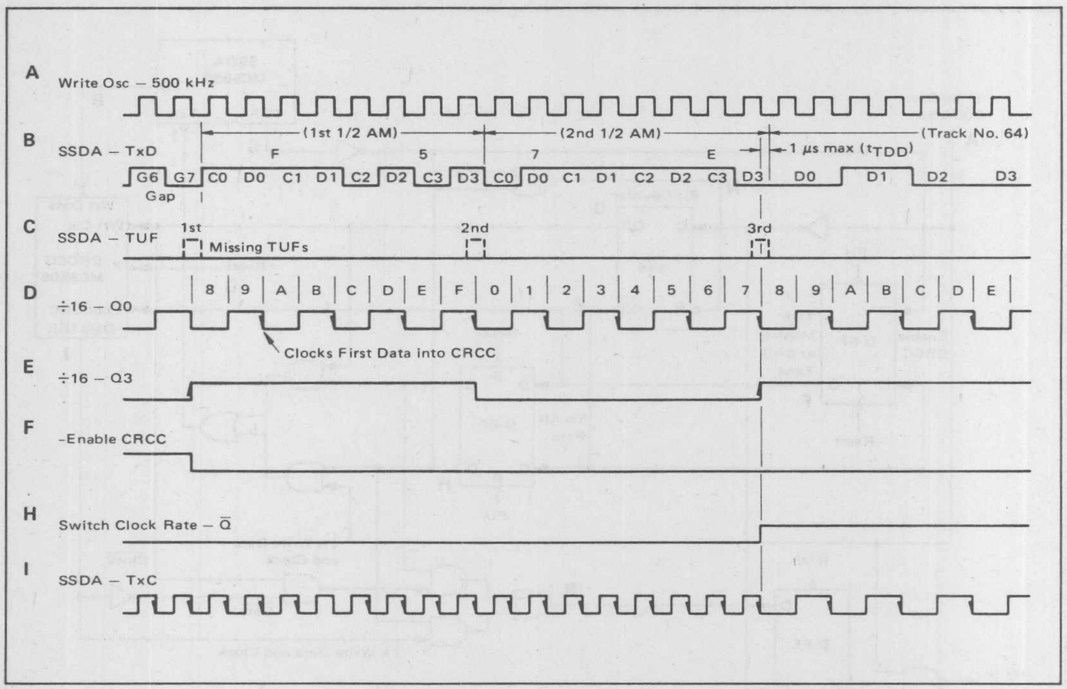


FIGURE 12—Write Data Format Timing

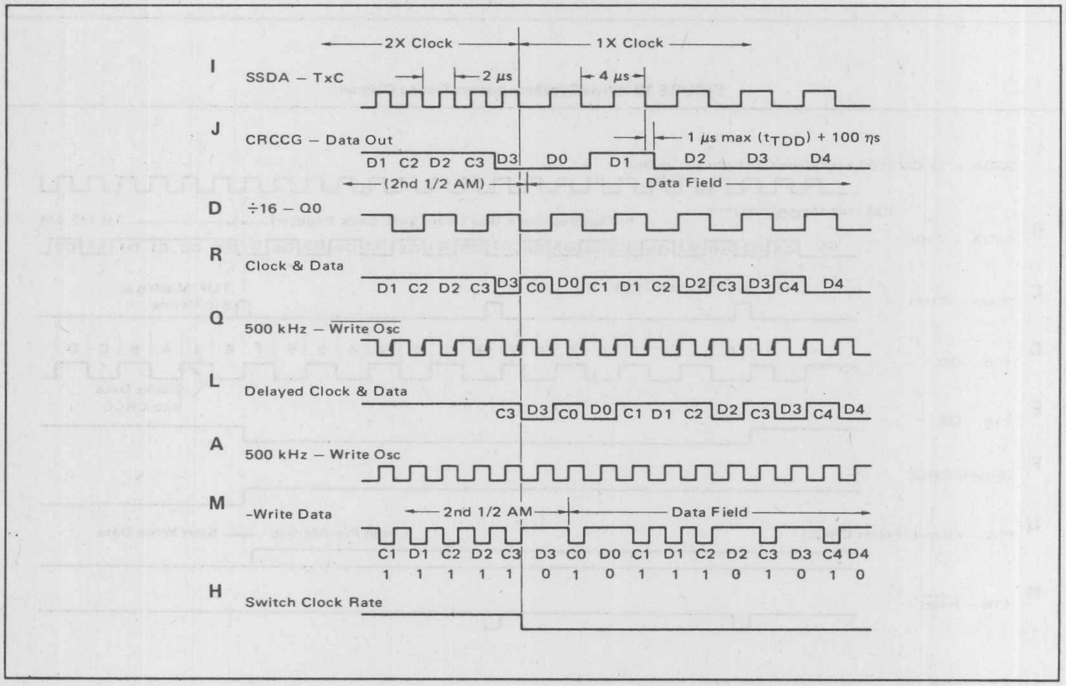


FIGURE 13—Write CRCC Simplified Block Diagram

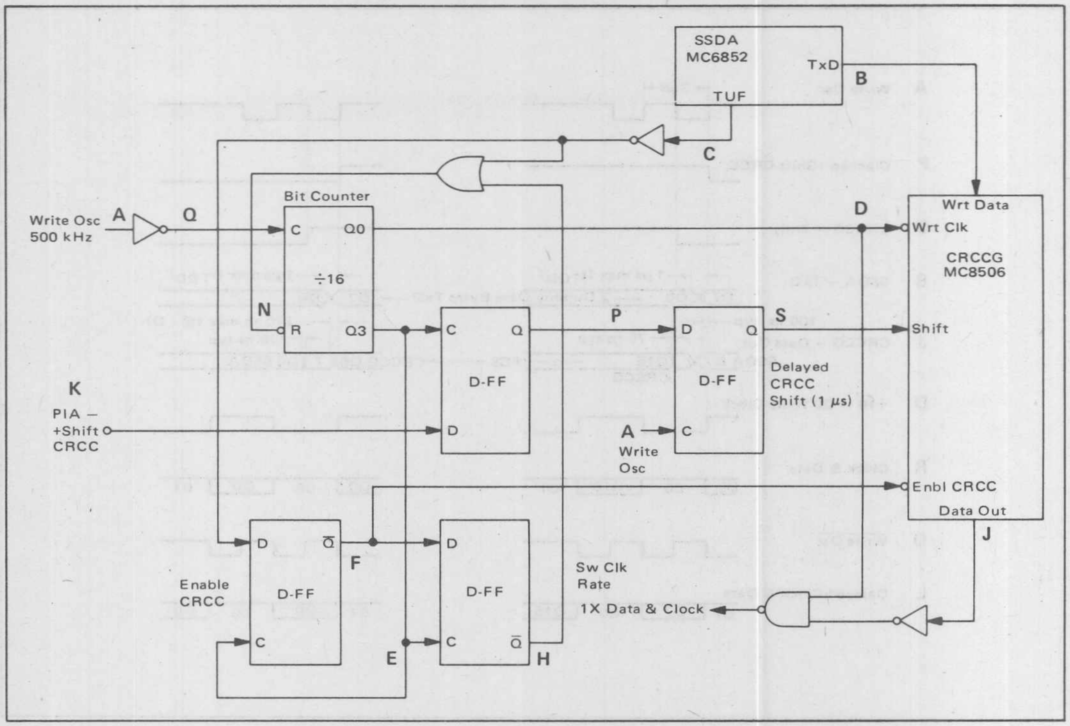


FIGURE 14—Write CRCC Timing Diagram

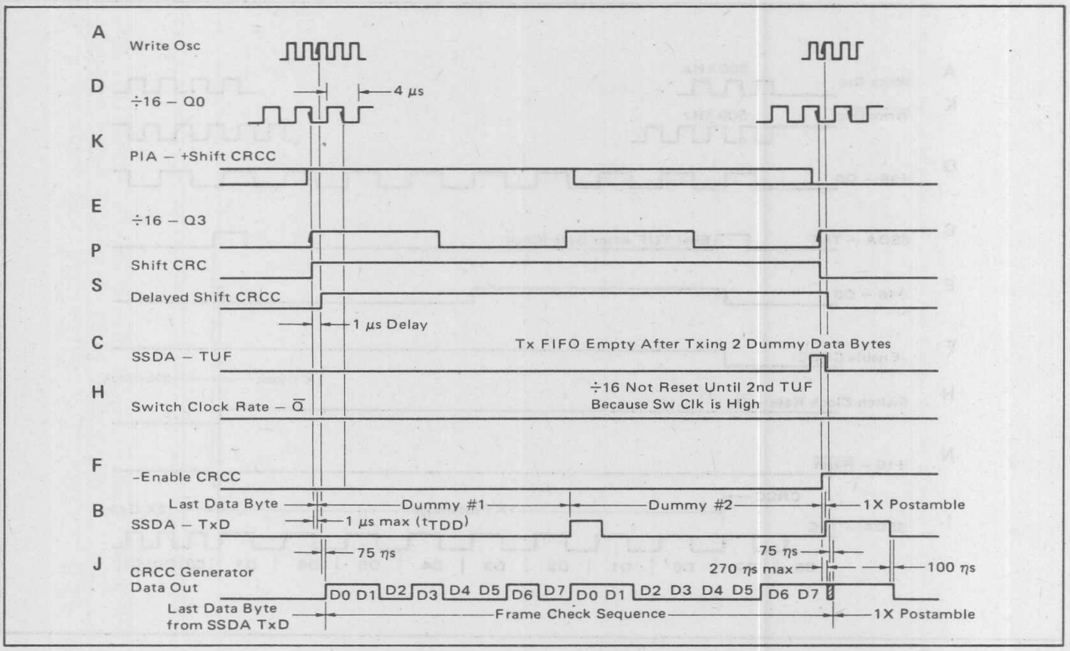


FIGURE 15—Write CRCC Detailed Timing

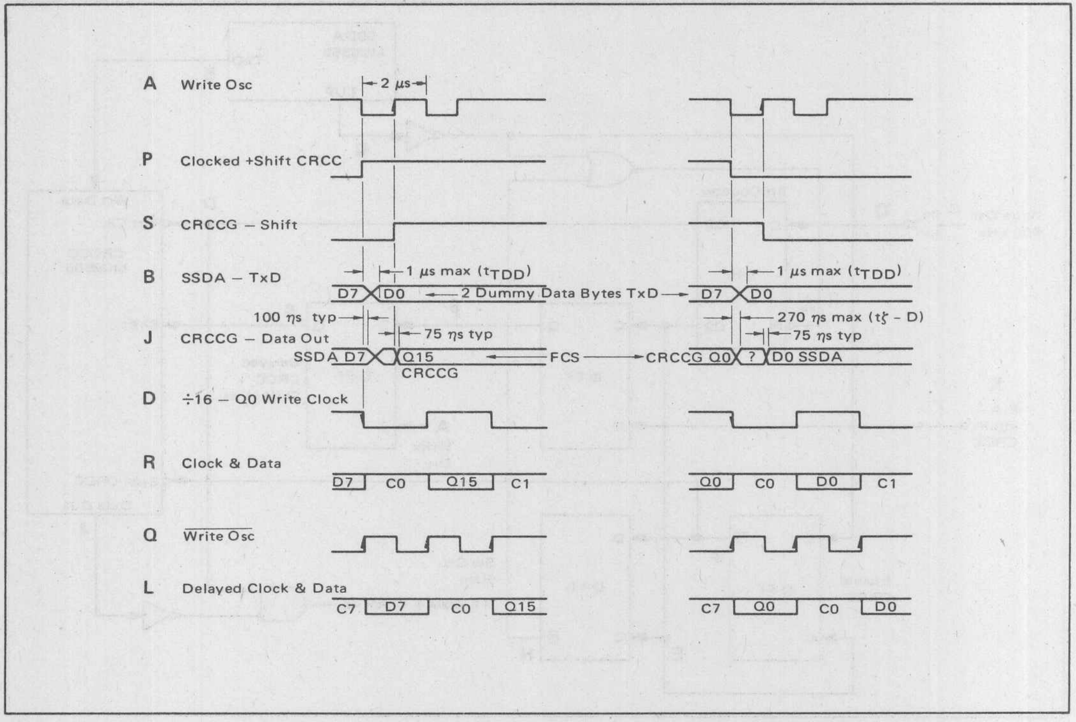
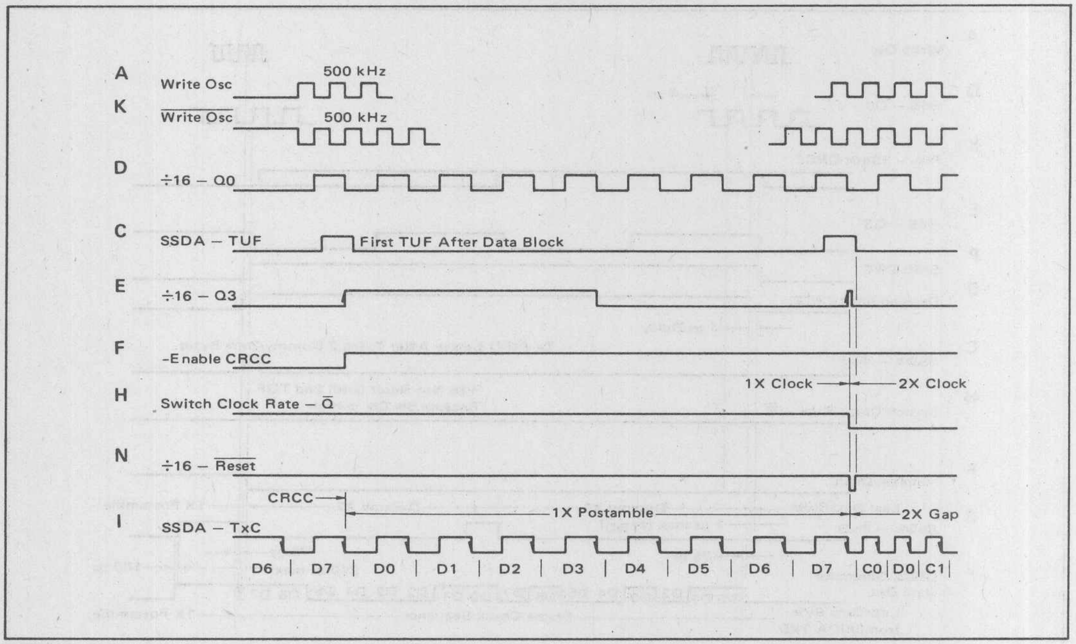


FIGURE 16—Write Termination Timing Diagram



AN-771

MEK6800D2 MICROCOMPUTER KIT SYSTEM EXPANSION TECHNIQUES

Prepared By
Wayne Harrington
Microprocessor Applications/Systems Engineering

The bus architecture of the MEK6800D2 Kit Microcomputer provides straightforward design options for memory or I/O port expansion. This note outlines techniques for interfacing an 8K or 16K memory array with the kit. A technique is also outlined whereby a data terminal-based ROM monitor such as MINIBUG may co-reside with the basic kit ROM JBUG Monitor. The resulting two-monitor system allows the user to switch between either the JBUG I/O port or the MINIBUG I/O port for moving data to and from RAM.

MEK6800D2 MICROCOMPUTER KIT SYSTEM EXPANSION TECHNIQUES

INTRODUCTION

The Motorola MEK6800D2 kit microcomputer system (hereafter referred to as MEK/D2) is a complete computer requiring only a +5 V power supply to begin microprocessor evaluation. It features a hexadecimal keyboard for data and command entry and seven-segment LED array for data display. In addition, the MEK/D2 provides an audio cassette I/O data transfer capability. Figure 1 presents a functional block diagram of the basic system. The intent of this note is to describe some useful system expansion techniques which exploit the architecture of MEK/D2 computer. This note is intended to supplement the information provided in the MEK/D2 manual and is divided into sections which discuss memory expansion, data I/O port expansion and expanded system application considerations.

Off-board memory expansion involves only minor changes in addressing and control logic plus certain elementary control-handshake logic to support both dynamic memory arrays and provide MPU control for slow memory arrays.

The inclusion of an I/O port to add data terminal communication in addition to the keyboard module function is accomplished by inserting control logic which converts MEK/D2 into a dual-monitor microcomputer system. This modification allows the basic MEK/D2 JBUG monitor ROM and its ACIA to co-reside with a MINIBUG ROM/ACIA combination. The JBUG-ROM/ACIA pair support keyboard and audio cassette data I/O transfer while MINIBUG, along with its ACIA, supports RS-232 or current loop-configured data terminals. Each ROM/ACIA pair may be manually initialized or software-accessed from the user program.

The capability to select, initialize, or address locations in either monitor ROM at will provides useful system application benefits. These include moving data between various storage media, directly addressing proven subroutines in either ROM from user program and manually selecting either monitor as desired to exploit the most useful commands of each during a software or system development phase. These modifications convert the MEK/D2 into a powerful software development tool.

RANDOM ACCESS MEMORY EXPANSION

Functional Design

The basic MEK/D2 Microcomputer Module provides for a maximum of 512 bytes of On-Board static RAM. Expansion for additional memory is accomplished by providing address and data bus buffers as well as some Off-Board control logic.

Figure 2 presents a functional block diagram summary of the supplemental logic necessary to support Off-Board memory expansion. Shaded blocks represent logic available with the basic MEK/D2 system. This convention holds for all schematics and diagrams in this note.

Certain static RAMs require up to 100 ns of data hold following chip deselect. The 10 ns data hold specified for the MC6800 MPU is insufficient to meet this requirement. The data bus enable (DBE) stretch network shown must be added if this type of RAM is utilized in the Off-Board expansion array. The Memory Control Handshake Logic provides control and timing signals between logic resident on Off-Board memory systems and the MPU clock module. Data transceivers, with a control logic block, are required to buffer bidirectional data to the Off-Board memory array as shown. The block labled "Array Select Decoder" represents logic for converting high-order address decode signals to Memory-Block enabling signals. These activitate either the On-Board or Off-Board array within the appropriate addressing range of a memory reference instruction.

Logic Design

Figure 3 shows a network which exploits the propagation delay of non-inverting CMOS buffers to generate a "stretched" $\phi 2$ for processor and peripheral data bus enable. This network delays the falling edge of DBES approximately 125 ns with respect to DBE. This meets the data hold time requirement of most static RAMs. Trim capacitor C_t may be added for fine adjustments to account for device variations in accordance with the equation shown.

Memory Control Handshake Logic is shown in Figure 4. Clocked latches E17A and E17B provide signals to control either dynamic memory refresh or slow-memory access on a synchronous basis with respect to MPU timing.

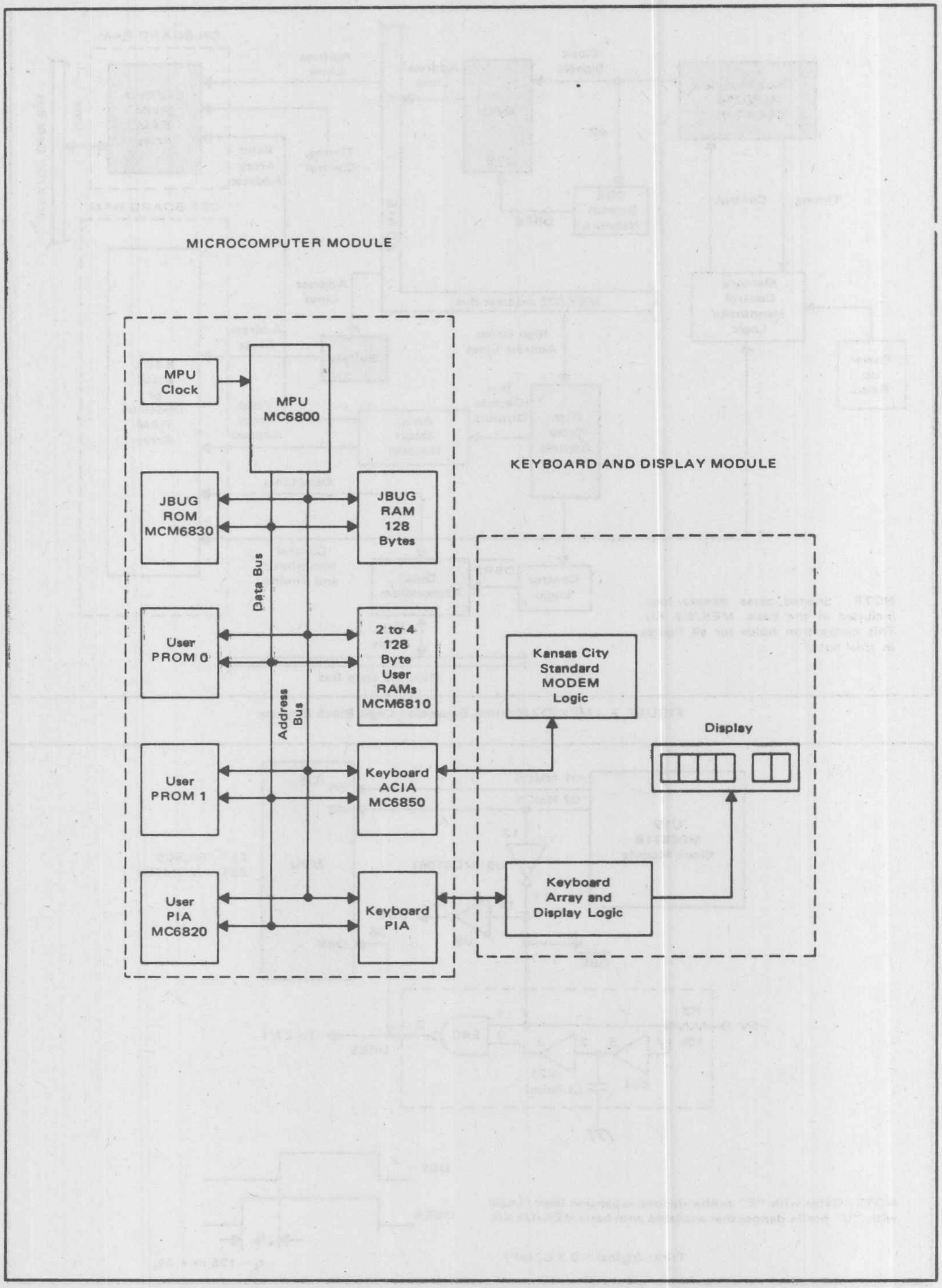
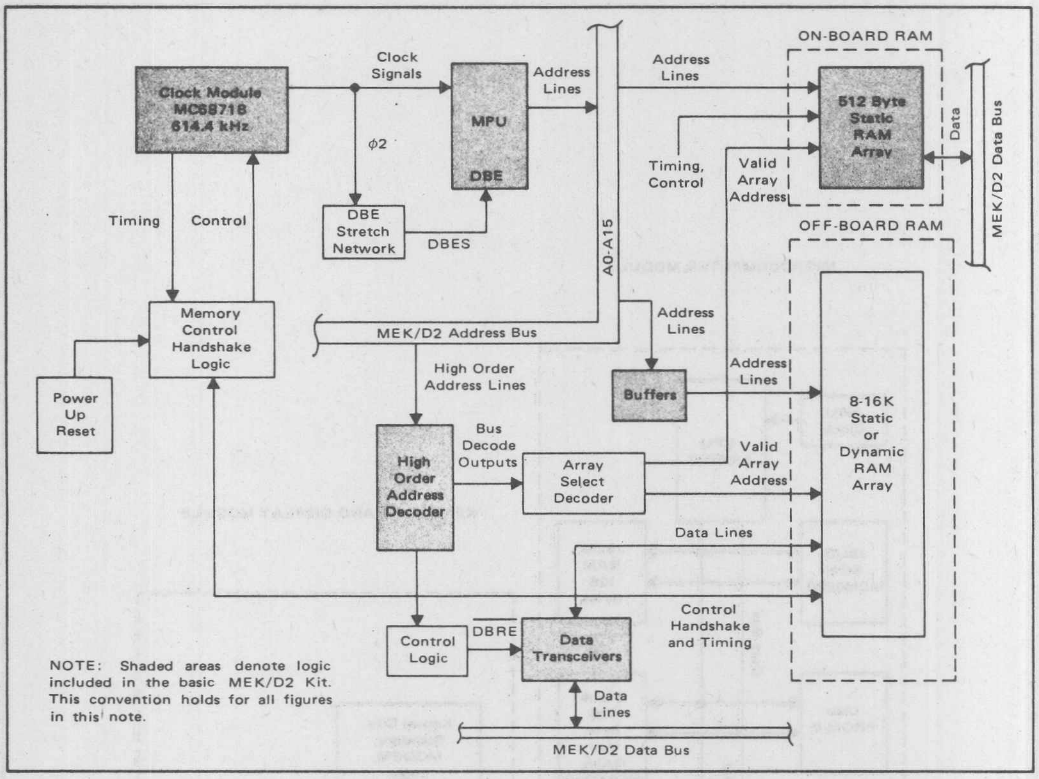
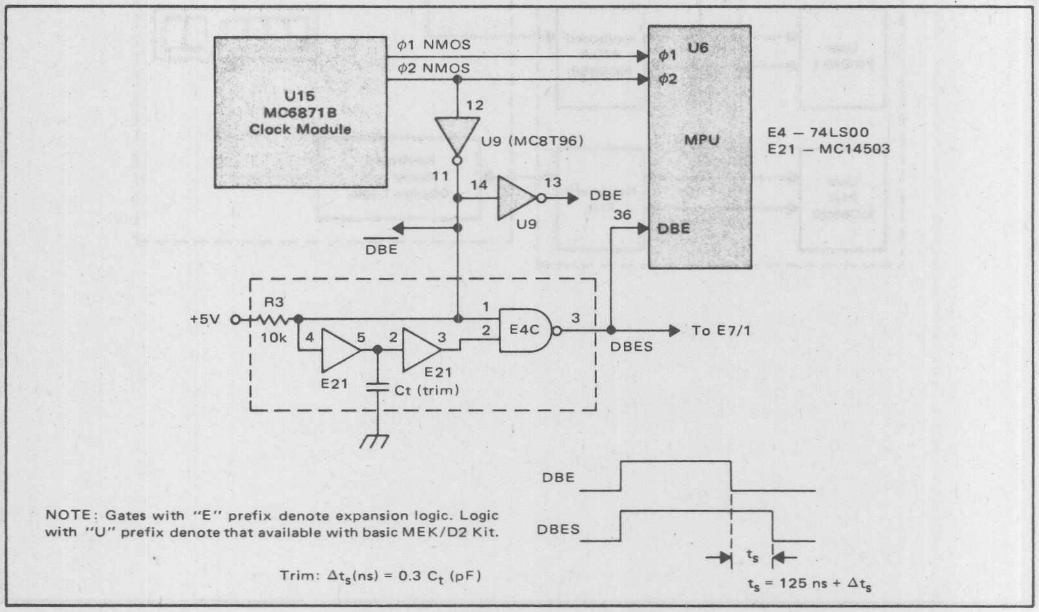


FIGURE 1 – MEK6800D2 Block Diagram



NOTE: Shaded areas denote logic included in the basic MEK/D2 Kit. This convention holds for all figures in this note.

FIGURE 2 – MEK/D2 Memory Expansion Logic Block Diagram



NOTE: Gates with "E" prefix denote expansion logic. Logic with "U" prefix denote that available with basic MEK/D2 Kit.

Trim: Δt_s(ns) = 0.3 C_t (pF)

FIGURE 3 – DBE Stretch Network for Memories with Non-Zero Data Hold Time Requirement

Dynamic memory cells store data in the form of electronic charge on the capacitance inherent in MOS transistor junctions. This charge must be periodically "refreshed." This is accomplished in most dynamic memories by performing a "dummy" read or write operation on each cell. In the case of the 8K Dynamic RAM Module (MEX6815-3), complete memory refresh is accomplished by a modified internal read operation on each of 32 columns once every 64 μs. (memory system organization is 128 rows X 32 columns). The columns are accessed by an address multiplexer which is pulsed by the Refresh Grant (RG) handshake signal once every 64 μs.

The power-up reset network, composed of E9 and E4D, sets latch E17A on power-up to insure a proper initialization of the refresh-handshake logic. E9 also automatically initializes the MPU system on power-up by pulsing E6/12.

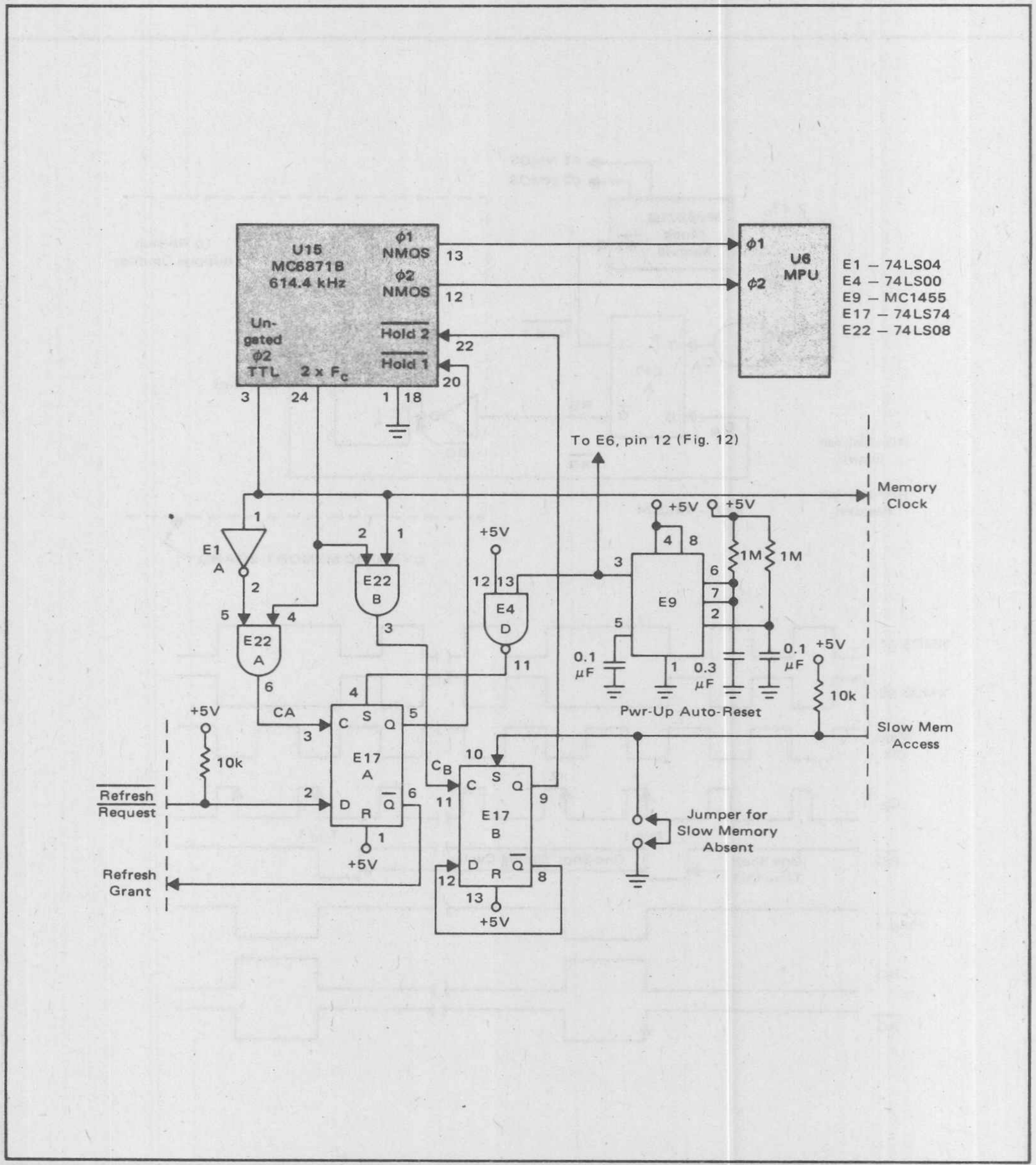


FIGURE 4 – Memory Control Handshake Logic

Figure 5 presents an example of refresh-handshake timing between latch E17A and logic on a dynamic memory system. The latch is clocked by AND-gate output C_A. The first low-to-high transistion of C_A (pulse 1) following time-out of the refresh-period one-shot (8602) samples the logical zero state appearing at the D input of E17A. This state and its complement are transferred by the rising edge of C_A to the Q and Q outputs of E17A as the signals Hold 1 and Refresh Grant (RG), respectively. The resultant falling edge of RG retriggers the 8602 to start a new timing cycle as shown in the diagram. This action returns the Request Refresh (RR) signal to logical one. This is sampled by the low-to-high transition of C_A, which returns Hold 1 high. The resulting Hold 1 signal applied to the H1 input of the MPU clock module is correctly phased to meet H1 set-up and release time requirements and "freezes" the MPU clock in the phase relation shown. The resulting RG pulse automatically increments the refresh address counter for the next refresh cycle.

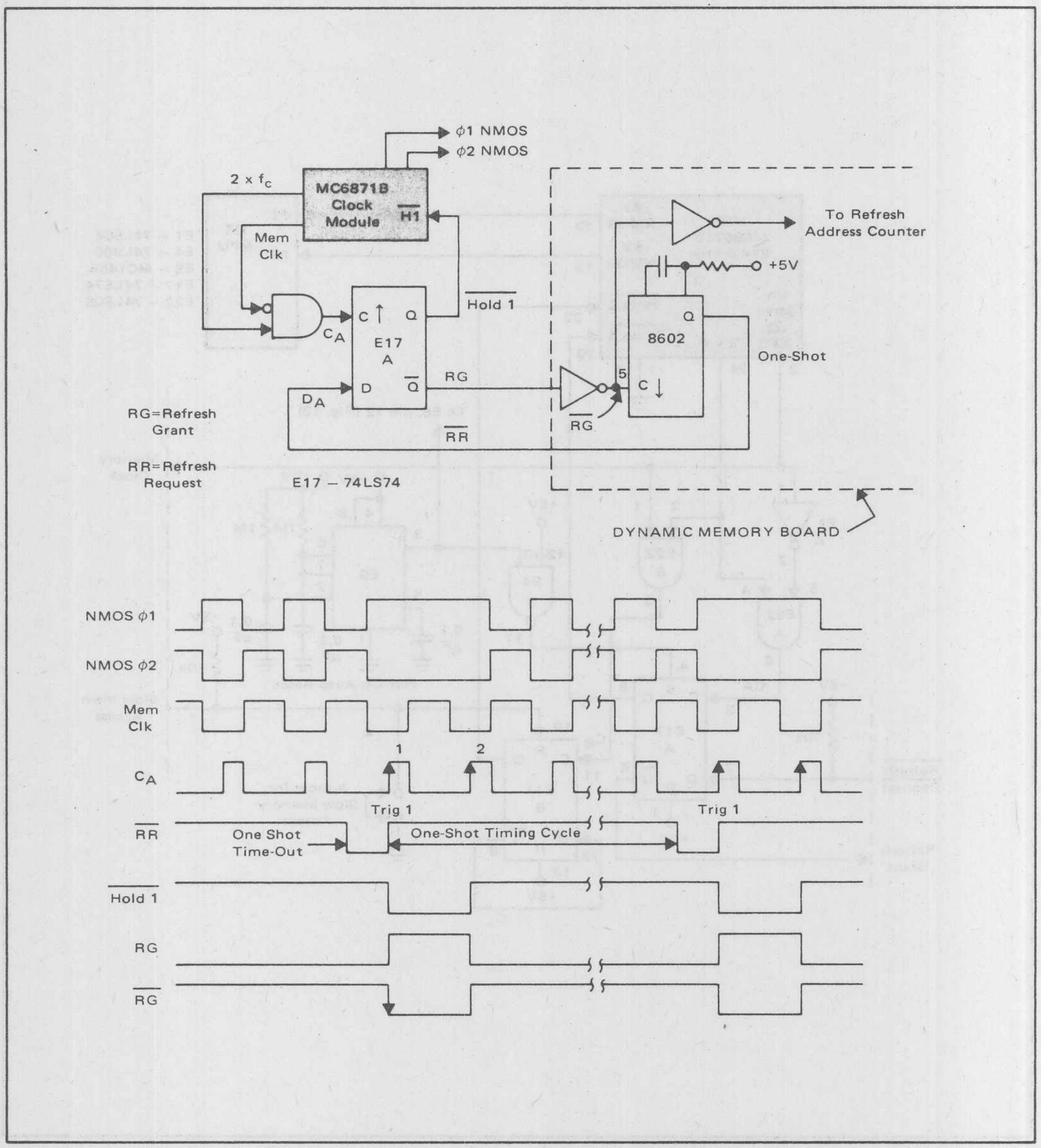


FIGURE 5 – Memory Refresh Handshake Logic and Waveforms

Figure 6 presents a typical example of slow memory control with handshake-timing between latch E17B and memory control logic on a slow memory board. Slow memory control signals are required to account for memories (or peripherals) whose access times are in the range of 540 to 4500 ns. The control signals provide proper slow memory data acquisition by freezing the MPU clock. This effectively allows the MPU to "wait" for memory data to return and still meet the maximum MPU bus memory access time specification of 540 ns. The access time upper limit of 4500 ns is determined by the maximum allowable clock phase 2 high time of 4500 ns. High times in excess of this value will introduce data loss within the MPU dynamic registers. These registers use the MPU clock for refresh, just as with memory cells in dynamic RAM. The sequence of events for a slow memory access are described in the waveform timing diagram. The array decoder output, AS, goes high following the low-to-high transition of $\phi 2$ for a memory reference within the addressing range of the array. The high-state of AS (or Slo Mem Acc) applied to the asynchronous-set input of latch E17B releases the hold-set condition on the latch and allows it to be clocked by the first C_B pulse. This forces Hold 2 (Q) low, which freezes the MPU clock in the phase relation shown. Hold 2 is returned high with the low-to-high transition of the next C_B pulse, since latch E17B is connected as a toggle flip-flop. Since Hold 2 is returned to logic 1, the clock is allowed to resume as shown, and the cycle is complete. The resulting freeze of the clock cycle with $\phi 2$ high and $\phi 1$ low adds a 1-clock-cycle delay to the normal access time available. This scheme may be extended with additional counters and logic in place of the toggle flip-flop to hold the clock a multiple-number of MEM Clk cycles for very slow memories. The total hold time must not exceed the 4500 ns maximum limit.

A key integrated circuit for generating system bus chip-select or enabling signals in the MEK/D2 is the high-order address decoder U11 – a 2-line to 4-line decoder/demultiplexer. This logic element decodes the three-most-significant bits, A15-A13, of the address bus in accordance with the following truth table.

A15	A14	A13	Bus Enable Term	Comments
0	0	0	$\overline{\text{RAM}}$ = 0	Enables 512 byte array On-Board RAM
0	0	1	$\overline{\text{2/3}}$ = 0	Enables 8K range of user stack
0	1	0	$\overline{\text{4/5}}$ = 0	Fnables 8K range of user stack
0	1	1	$\overline{\text{PROM 1}}$ = 0	Enables user PROM located at $6000^{*}{}_{16}$
1	0	0	$\overline{\text{I/O}}$ = 0	Enables ACIA located at 8008_{16}
1	0	1	$\overline{\text{STACK}}$ = 0	Enables 128 byte RAM used by JBUG monitor
1	1	0	$\overline{\text{PROM Ø}}$ = 0	Enables user PROM located at $C000_{16}$
1	1	1	$\overline{\text{ROM}}$ = 0	Enables JBUG ROM located at $E000_{16}$

*Denotes Base 16 (hexadecimal) number

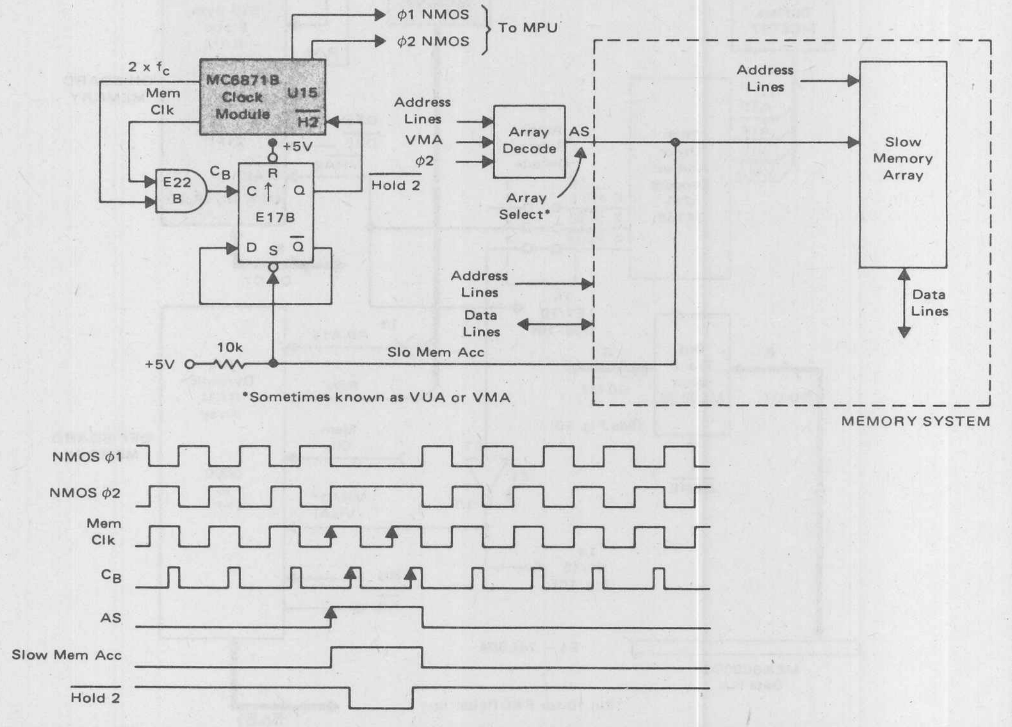


FIGURE 6 – Slow Memory Handshake Logic and Waveforms

This scheme divides the 64K addressing range of the MPU into eight 8K blocks. The 512 byte static RAM array is placed in the bottom 8K range, the next two 8K blocks are reserved for expansion RAM, the fourth contains a user PROM, etc.

Figure 7 presents the Bus Peripheral Allocation Map for the basic MEK/D2 system. Exact address boundaries of the bus peripherals described in the decoder truth table are defined in this map. The decoder output terms which enable the first three 8K blocks of memory, beginning with address zero, are RAM, 2/3 and 4/5. Inspection of the map shows that within the first addressable 8K block, only 512 bytes are dedicated to static RAM. This produces a memory addressing "gap" in the range 0200 to 1FFF as far as continuous addressing within the first 8K block is concerned. This problem may be solved by additional decoding of the three RAM select signals above so as to place an 8K expansion RAM in the first 8K addressing block, or a 16K expansion RAM within the first two 8K addressing blocks. The 512

	Address Ranges
JBUG ROM	E000-E3FF
User PROM	C000-C3FF
JBUG Stack/RAM	A000-A07F
Keyboard Module PIA	8020-8023
Keyboard Module ACIA	8008-8009
User PIA	8004-8007
User PROM	6000-7FFF
User RAM	4000-5FFF
User RAM	2000-3FFF
User Static RAM	0000-01FF

FIGURE 7 – MEK/D2 Bus Peripheral Allocation Map

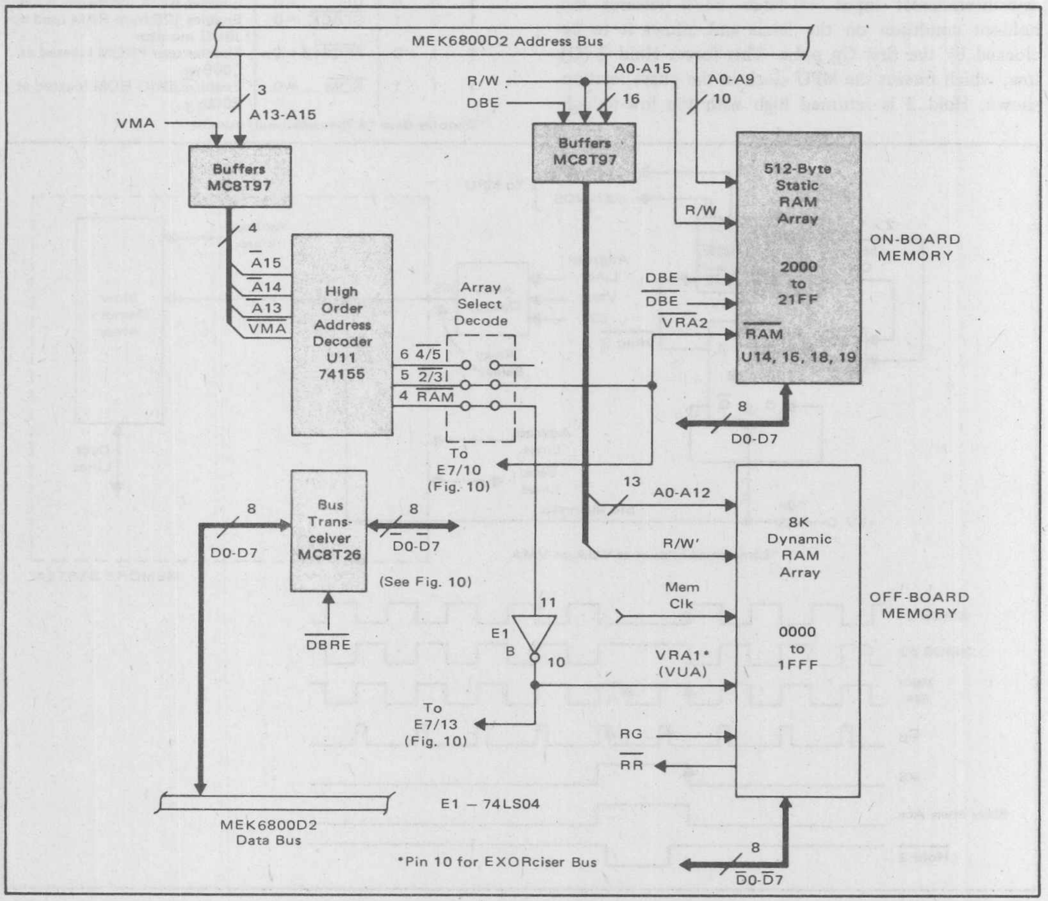


FIGURE 8 – Addressing for 8.5K Memory Configuration

byte static array is then placed in either the second or third block, respectively, "on top" of the expansion RAM. Figures 8 and 9 show the additional decode required to form either an 8.5K or 16.5K memory configuration. Control and Timing signals necessary to support these arrays are also shown.

Data flow direction to Off-Board memory is determined by the decode/control logic shown in Figure 10. This logic asserts DBRE (Data Bus Receive Enable) for any MPU read cycle involving Off-Board memory. This enabling scheme should be used with any additional Off-Board memory, whether static or dynamic.

Recent developments in semiconductor dynamic RAM system design have provided compact, cost-effective arrays such as the MMS68100 and MMS68103 produced by Motorola Memory Systems. These are available in 4K x 8, 8K x 8, or 16K x 8 size. The most notable feature of these memories is that the usual refresh-handshake logic, such as shown in Figure 4, is not required since refresh is processed by memory board logic during MPU phase 1.

I/O DATA PORT EXPANSION/MODIFICATION

Dual Monitor System – Functional Description

The basic MEK/D2 system with keyboard data entry and seven-segment light-emitting-diode display may be expanded to include a co-resident data terminal I/O capability which may be evoked manually or from user program. The software necessary to support data terminal operations is provided in firmware using a MINIBUG ROM. This ROM monitor co-resides with the JBUG monitor ROM supplied with the basic MEK/D2. ROM access and initialization is controlled by the logic shown in functional block diagram form in Figure 11. With this scheme, peripheral chip-select signals derived from the high-order address decoder (U11) are steered to the desired ROM-ACIA pair as a function of the state of the Chip Select Control signal, CSC. CSC is generated from either the manual ROM select switch (S_R) or by user-program command from the PIA. Control from user program automatically overrides the manual input but does not initiate an MPU reset cycle as does a manual

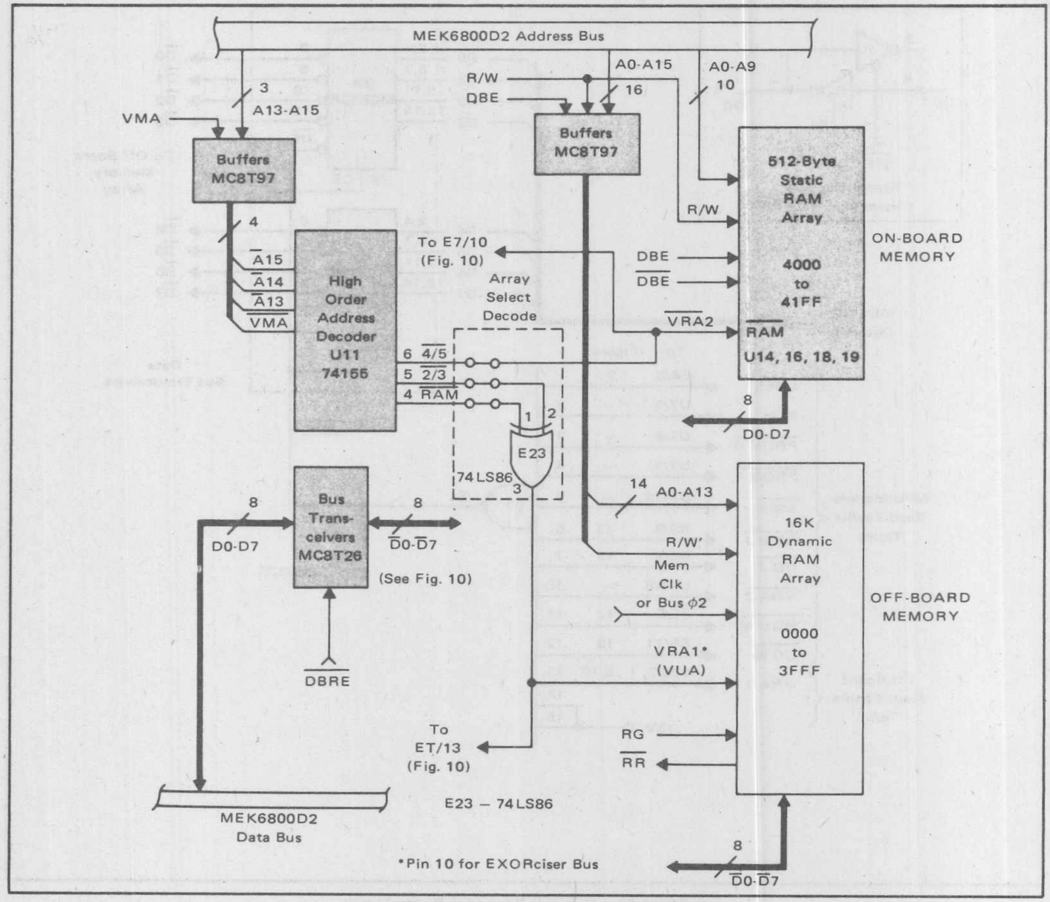


FIGURE 9 – Addressing for 16.5K Memory Configuration

select from S_R. With S_R in position J (for JBUG enable), the ROM and I/O chip-select signals (ROM and I/O are steered, respectively, only to the JBUG ROM or ACIA, while the MINIBUG ROM and ACIA are held deselected. The converse actions occur for S_R at position M (for MINIBUG enable). Each toggle of S_R generates an MPU Reset pulse via the State Change Detect Logic. This has the effect of automatically initializing each monitor ROM when manually selected. Nine standard data terminal baud rates may be derived from existing MEK/D2 logic and are used to provide transmit and receive clocks for the MINIBUG ACIA.

Logic Design

Logic realizations of the system functions depicted in Figure 11 are presented in Figures 12, 14, 15, 16 and 17. Figure 12 shows the Chip Select Steering Logic, MPU Cycle-Sync Logic and State Change Detect Logic. Chip-select steering is accomplished by the network composed of gates E5 and E1C. The clocked-latch network (E3A – E3B) which generates the chip-select steering control signal, provides two design benefits. First, monitor switching occurs only after MPU reset is asserted and prior to a $\phi 2$ cycle, thus assuring that data will not be erroneously written or read as a result of a manual monitor select. In addition, latch E3A, under the control of the PIA, provides an asynchronous-override to the manual select switch control. This feature allows direct access to subroutines in either ROM or addresses in either ACIA from the user program. A subroutine to accomplish this access is described in a following section.

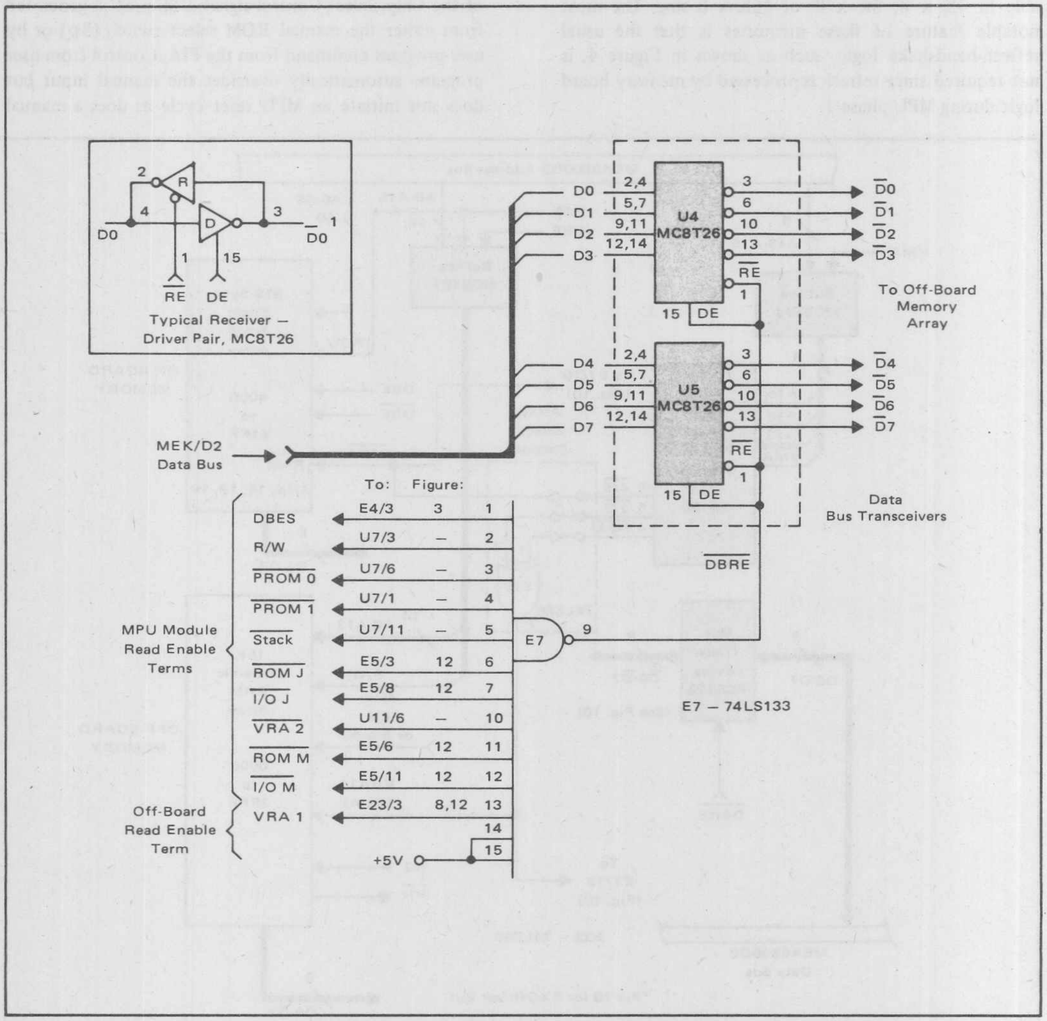


FIGURE 10 – Data Bus Expansion Control Logic

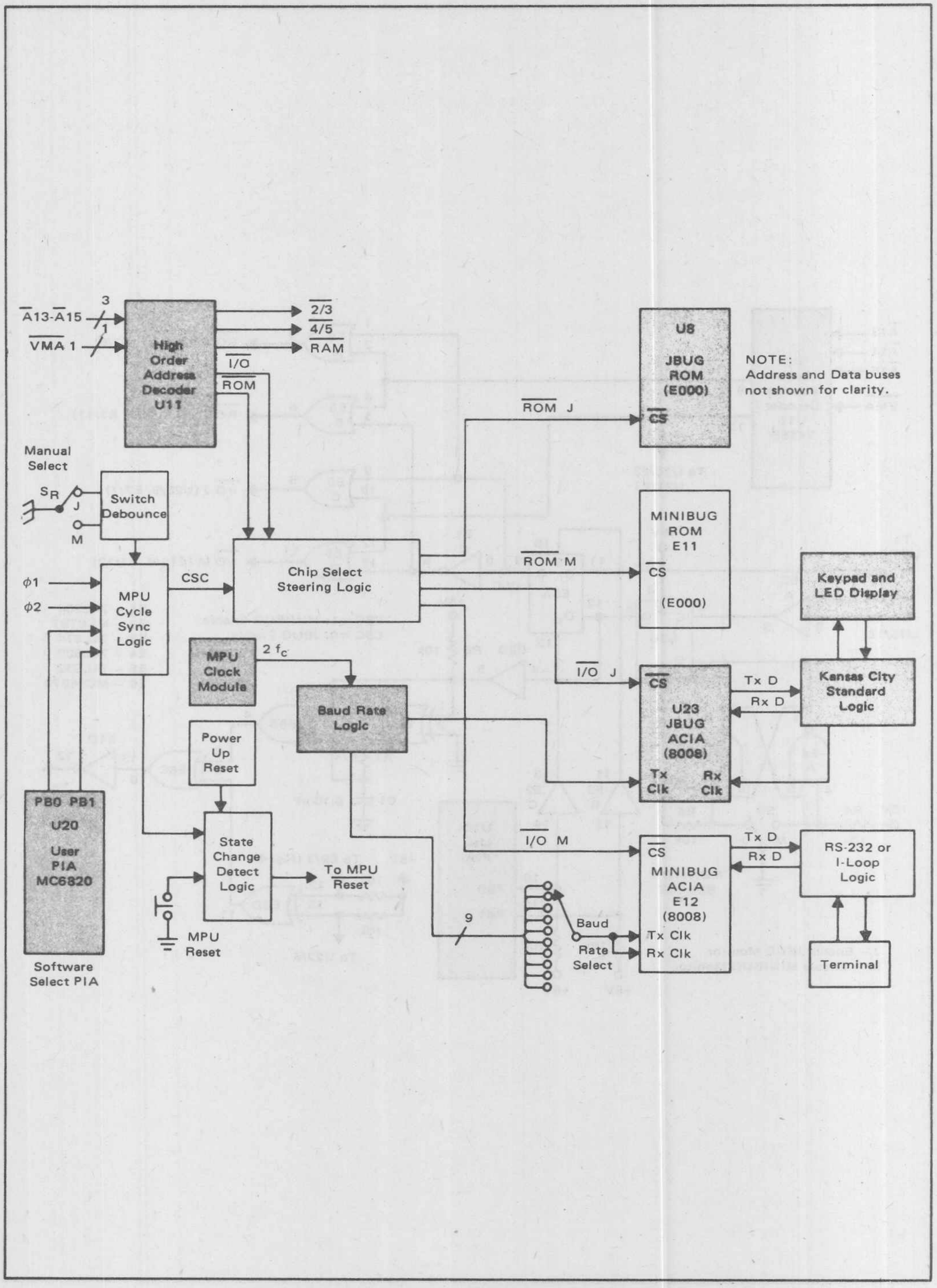


FIGURE 11 – Dual-Monitor Switching Logic Block Diagram

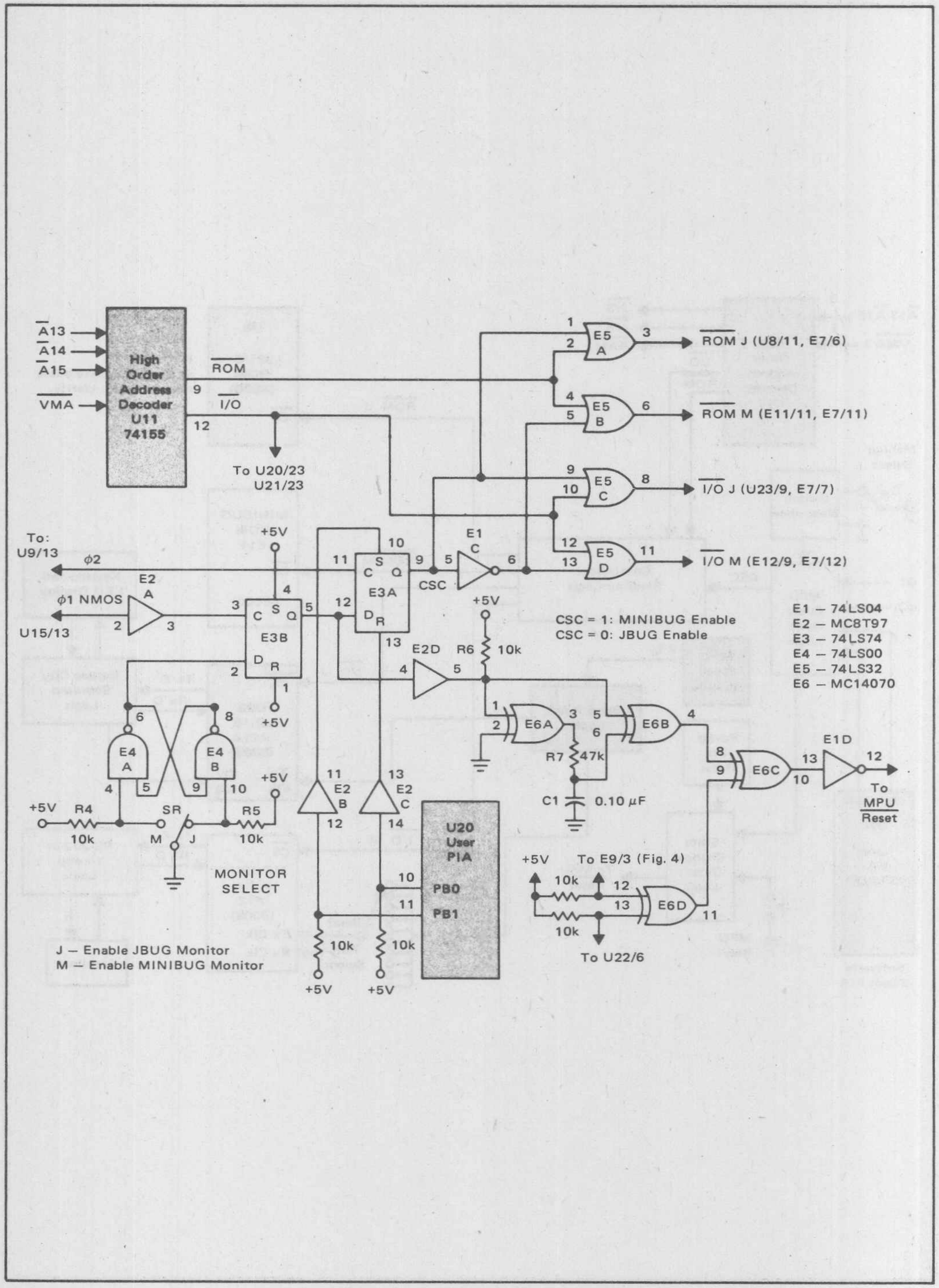


FIGURE 12 – Dual-Monitor Switching Logic

Figure 13 shows the chip-select timing for a manual command conversion from JBUG to MINIBUG via toggle switch S_R. Exclusive-OR gates E6 in Figure 12 form the state-change detection circuit which generates a 4 ms reset pulse for automatic MPU initialization whenever the monitor select switch is thrown in either direction. Note that provision for direct push-button reset of the MPU is also retained via E6D to pin 6 of U22.

Figure 14 shows address, data and control signal interconnection to the MINIBUG ROM and its ACIA. Note that even though these peripherals reside at the same bus address as the JBUG pair, the two pairs are never simultaneously selected due to the complementary' control nature of the chip select steering logic.

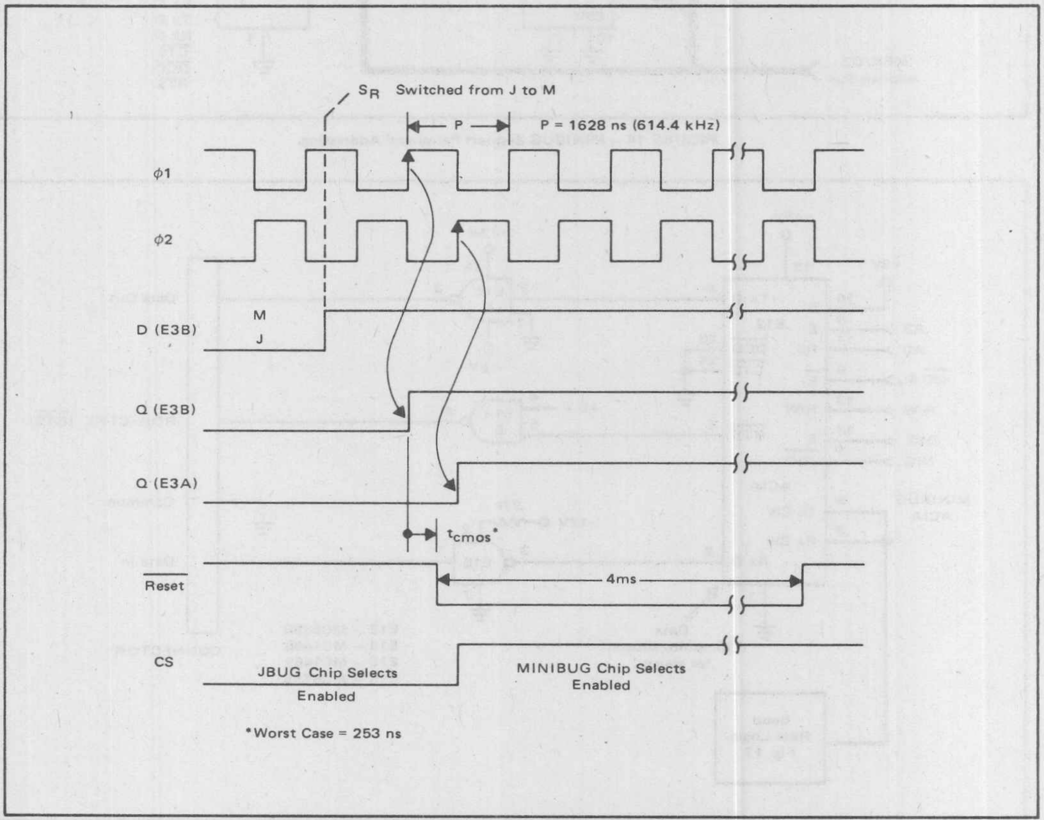
Figures 15 and 16 show circuitry necessary for interfacing with data terminals using either RS-232 or current-loop I/O configuration. Data terminal baud-rate clocks may be taken from the existing MC14040 binary counter (U17) outputs as shown in Figure 17. An MC1455 connected as an astable multivibrator (E13) is utilized to generate a baud-rate clock consistent with current-loop TTYs.

Software Control Considerations

Software access to addresses in either Monitor ROM or ACIA is gained through a subroutine which controls the output states of PB0 and PB1 of the user PIA. The four possible states of PB0 – PB1 produce the following control functions with respect to latch E3A, Figure 12:

PB1	PB0	Monitor Control Function
0	0	Illegal state
0	1	Enable MINIBUG ROM/ACIA user addressing
1	0	Enable JBUG ROM/ACIA user addressing
1	1	Addressing controlled by Monitor Select Switch, S_R

The 1-1 state is automatically entered upon system power-up or manual reset, since following the power-up reset pulse the PIA Data-Direction-Registers are programmed as inputs (all registers cleared). PB0 – PB1 appear as high-impedance inputs and both terms are held at logic 1 by the 10 kΩ pullup resistors.



S_R Switched from J to M
P
P = 1628 ns (614.4 kHz)
φ1
φ2
D (E3B)
M
J
Q (E3B)
Q (E3A)
t_{cmos}*
Reset
4ms
CS
JBUG Chip Selects Enabled
MINIBUG Chip Selects Enabled

*Worst Case = 253 ns

FIGURE 13 – Monitor Chip-Select Timing – Manual Select
Select MINIBUG, Deselect JBUG

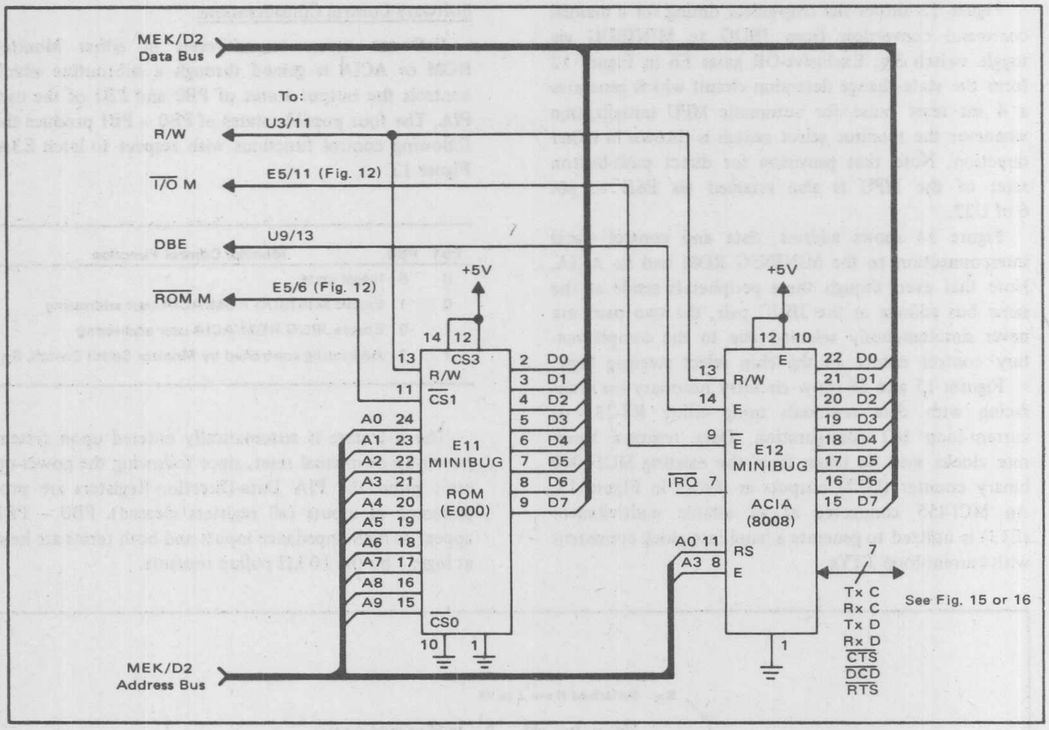


FIGURE 14 – MINIBUG Support Peripheral Addressing

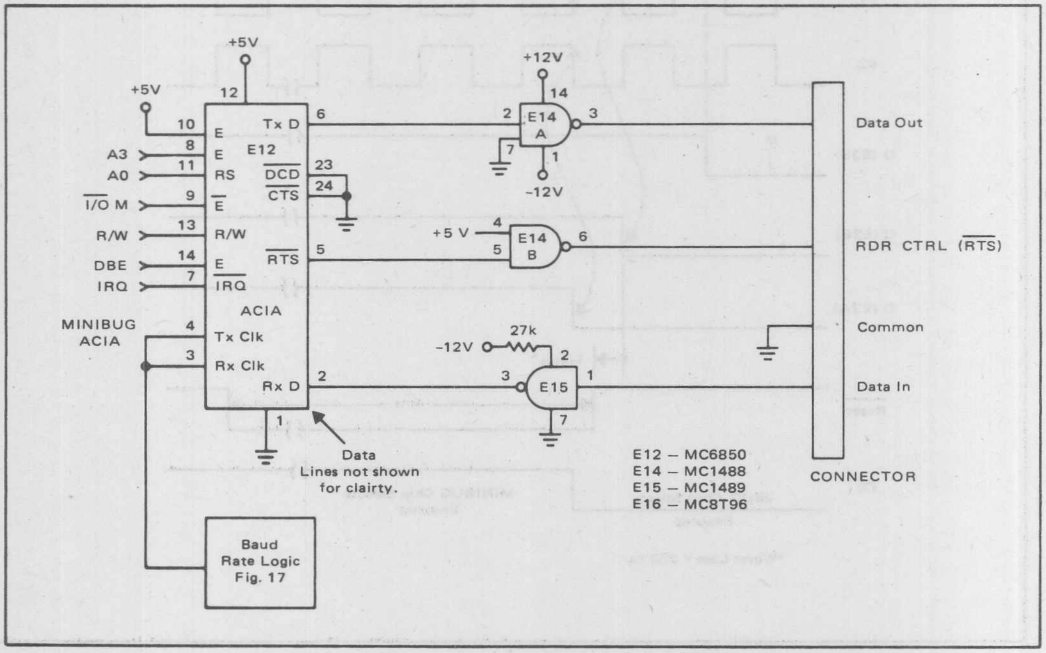


FIGURE 15 – Terminal Interface Logic
RS-232 Channel

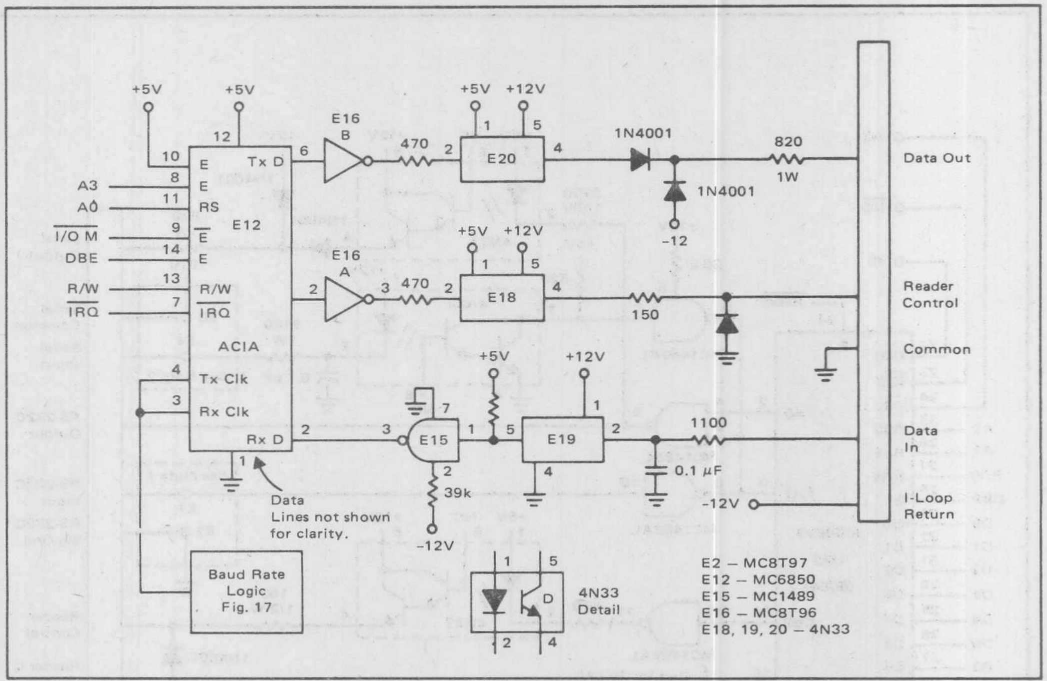


FIGURE 16 – Terminal Interface Logic
Current Loop Channel

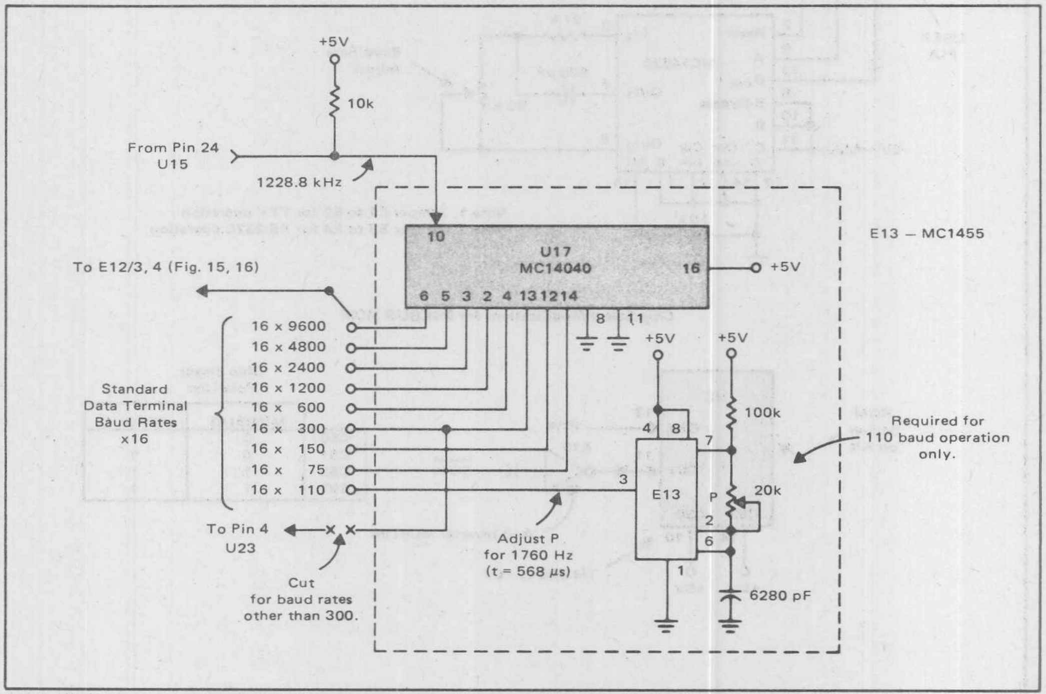
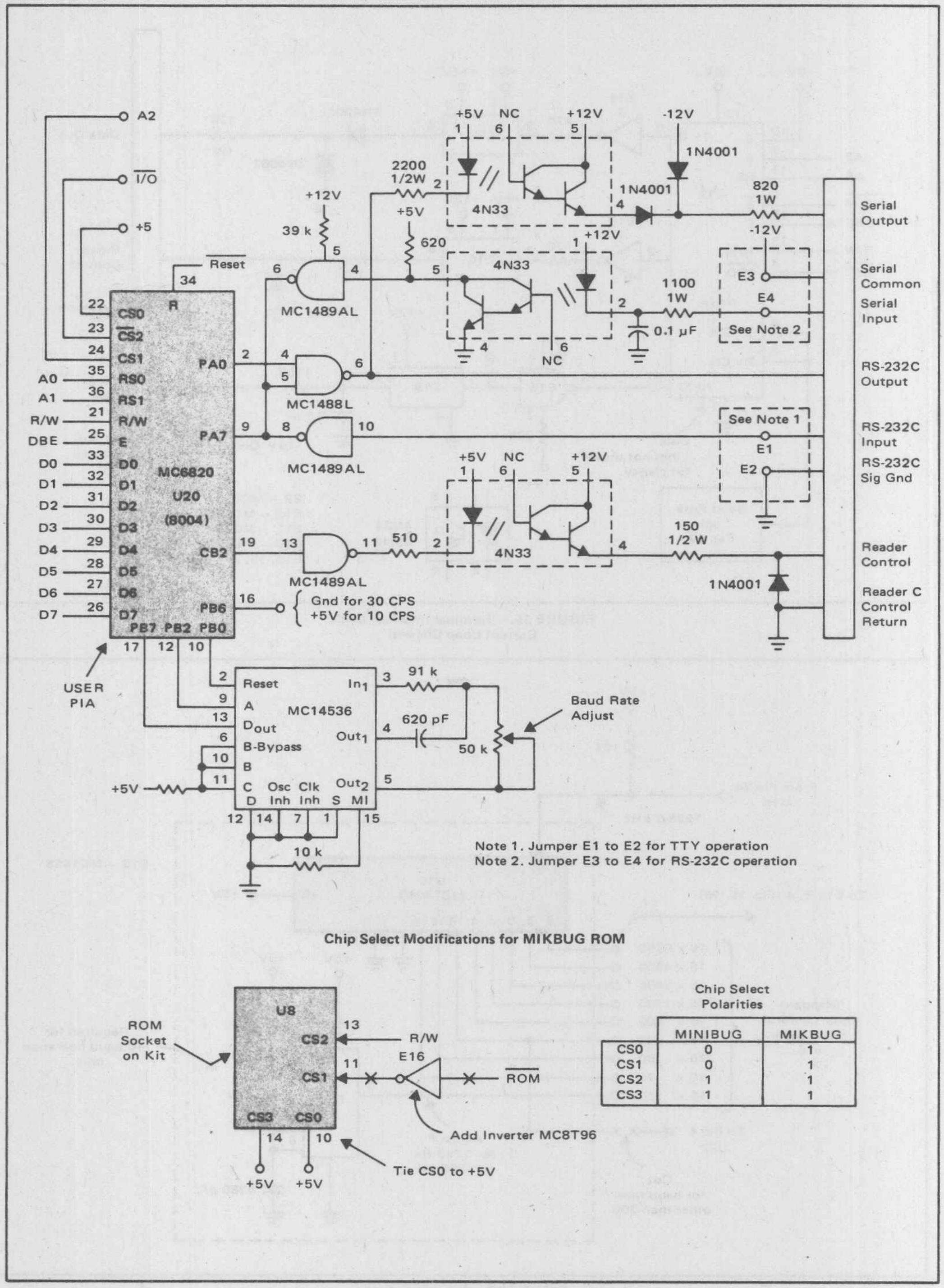


FIGURE 17 – Baud Rate Logic



Note 1. Jumper E1 to E2 for TTY operation
Note 2. Jumper E3 to E4 for RS-232C operation

Chip Select Modifications for MIKBUG ROM

Chip Select Polarities

	MINIBUG	MIKBUG
CS0	0	1
CS1	0	1
CS2	1	1
CS3	1	1

FIGURE 18 – Terminal Interface Logic for MIKBUG

Data Terminal-Only Configurations

A configuration which employs data-terminal communication interface only may be easily implemented by inserting MINIBUG or MIKBUG Monitor ROMs into the JBUG ROM socket (U8). Foil path modifications and additional logic necessary to support these ROMs are as follows:

Modifications for MINIBUG
1. Cut foil path at U17, pin 13.
2. Connect pins 3 and 4 of U23 (ACIA for Audio Cassette).
3. Add terminal I/O interface logic as shown by Figures 15 or 16 and Figure 17. Connect U17 output to pin 3 or 4 of U23 as shown. U17/3 need not be cut (as shown in Figure 17) if 300 baud operation is desired.

Modifications for MIKBUG
1. Add terminal I/O interface logic to the user PIA (U20) as described by the schematic of Figure 18.
2. Cut foil paths at U8/10 and U8/11 and connect per Figure 18.

The I/O logic and discrete components described in these figures may be mounted in the wire-wrap area provided on the microcomputer module board.

SYSTEM APPLICATION CONSIDERATIONS

A subroutine which controls the monitor-selection latch (E3, Figure 12) through the PIA is presented in Figure 19. User program access to subroutines in ROM or addresses in ACIA is accomplished by first calling the monitor access subroutine (MONACC) shown in Figure 19 and then executing a memory reference instruction to the ROM or ACIA address desired. As an example, the subroutine calling sequence:

```
        .
        .
        .
LDAA # $ 41 Form ASCII "A"
LDAB # $ 01 Get subroutine constant
JSR MONACC Enable MINIBUG ROM/
                ACIA addressing
JSR $ E108 Output ASCII char to
                terminal
        .
        .
        .
```

causes the character "A" to be printed on a terminal as a result of MINIBUG monitor access from the subroutine MONACC. In this example, the hex address E108 is the start vector of the MINIBUG II subroutine OUTCH which outputs one ASCII character to a terminal. The following is a list of useful data-moving subroutines contained in MINIBUG II and III along with their starting addresses, entry and exit conditions:

MINIBUG ROUTINES

( ) – Addresses in MINIBUG II
⟨ ⟩ – Addresses in MINIBUG III

BADDR ($E0D9) ⟨$E0F8⟩* – Build a 16-bit hexadecimal address from four digits entered from the keyboard.
Entry requirements: none
Exit: X-register contains the 16-bit address. The A & B registers are destroyed.

BYTE ($E0E7) ⟨$E106⟩ – Input two hex characters from the keyboard and form a 1-byte number.
Entry requirements: none
Exit: A-register contains the 8-bit number. B-register is destroyed.

OUTHL ($E0FA) ⟨$E118⟩ – Output left digit of hex number to console.
Entry requirements: A-register contains hex number.
Exit: A-register is destroyed.

OUTHR ($E0FE) ⟨E11C⟩ – Output right digit of hex number to console.
Entry requirements: A-register contains hex number.
Exit: A-register is destroyed.

OUTCH ($E108) ⟨$E126⟩ – Output one ASCII character to terminal.
Entry requirements: A-register contains ASCII character to output.
Exit: No change

INCHP ($E115) ⟨$133⟩ – Input one character, with parity, from terminal to A-register.
Entry requirements: None
Exit: A-register contains character input.

INCH ($E11F) ⟨$E133⟩ – Input one character from terminal to A-register and set parity bit = 0. If character is a delete ($7F) it is ignored. Location $A00C should be equal to zero if the character should be echoed (MINIBUG II only).
Entry requirements: none
Exit: A-register contains character without parity.

PDATA1 ($E130) ⟨$E14B⟩ – Print at terminal the ASCII data string pointed to by X-register. Data string must contain an ASCII EOT ($04) as a terminator.
Entry requirements: X-register contains the address of the 1st byte of the data string. The data string is terminated with a $04 character.
Exit: A-register is destroyed. X-register contains address of $04 character.

OUT2H ($E173) ⟨$E18D⟩ – Output two hex characters, pointed to by X-register to the terminal.
Entry requirements: X-register contains the address of the characters to be output.
Exit: A-register is destroyed. X-register is incremented.

* $ is Motorola Resident Assembler syntax for a hexadecimal number.

OUT2HA ($E175) ⟨$E10F⟩ – Output two hex character in A-register to the terminal.

Entry requirements: A-register contains the characters to output.

Exit: A-register is destroyed. X-register is incremented.

OUT4HS ($E17C) ⟨$E196⟩ – Output four hex characters (2 bytes) plus a space to the terminal.

Entry requirements: X-register contains address of first byte.

Exit: A-register is destroyed. X-register contains address of second byte.

OUT2HS ($E17E) ⟨$E198⟩ – Output two hex characters (1 byte) and a space to the terminal.

Entry requirements: X-register contains address of byte to output.

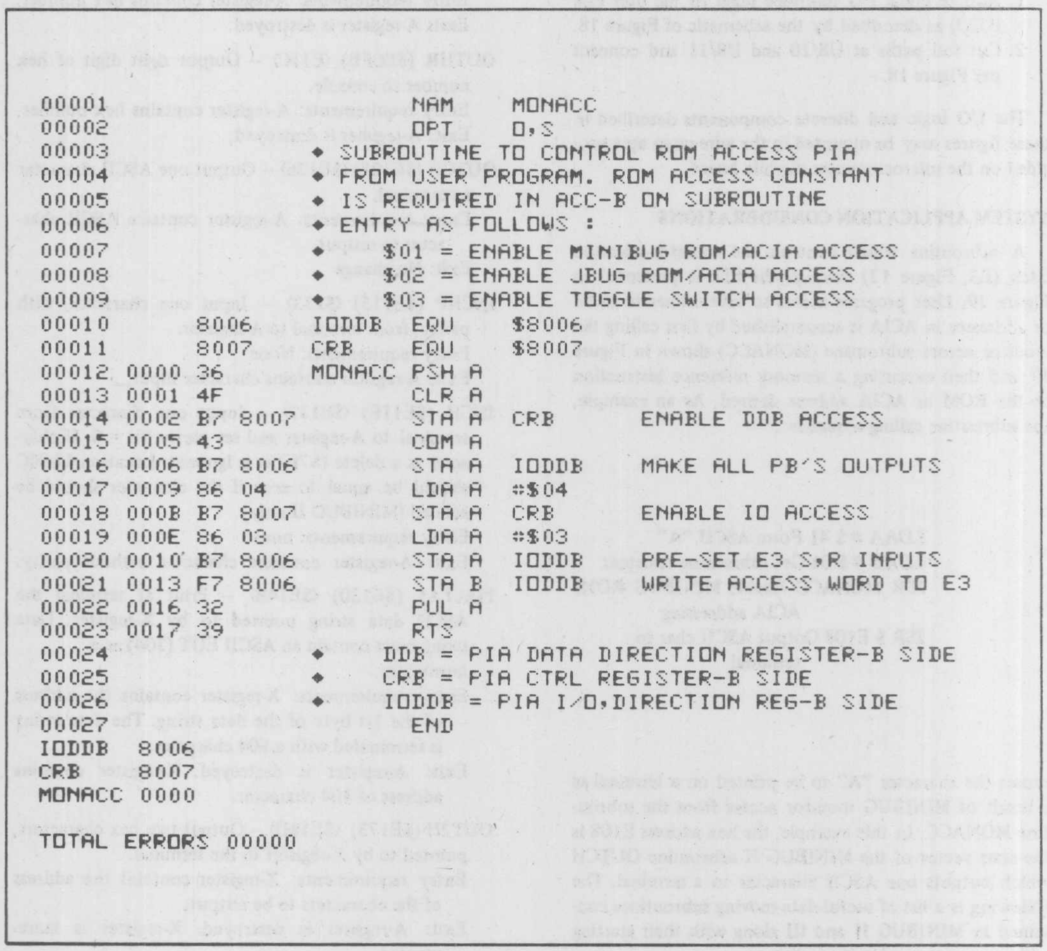
Exit: A-register is destroyed. X-register is incremented.

OUTS ($E180) ⟨$E19A⟩ – Output a space.

Entry requirements: none

Exit: A-register destroyed.

The ability to gain access to two co-residing monitor ROMs via manual or software commands combined with keyboard, audio cassette, or data terminal I/O capability provides opportunity for moving program data between various storage media. It is possible, for instance, to create and assemble a program under the control of MINIBUG II or III using an RS-232-compatible digital cassette terminal. The resulting object code is loaded to MEK/D2 RAM using the MINIBUG "L" command. The Monitor Control Switch may now be used to initialize the JBUG Monitor in order to move the object code in RAM to an audio cassette tape with a JBUG "P" command.



```
00001                      NAM    MONACC
00002                      OPT    O,S
00003                    ◆ SUBROUTINE TO CONTROL ROM ACCESS PIA
00004                    ◆ FROM USER PROGRAM. ROM ACCESS CONSTANT
00005                    ◆ IS REQUIRED IN ACC-B ON SUBROUTINE
00006                    ◆ ENTRY AS FOLLOWS :
00007                    ◆    $01 = ENABLE MINIBUG ROM/ACIA ACCESS
00008                    ◆    $02 = ENABLE JBUG ROM/ACIA ACCESS
00009                    ◆    $03 = ENABLE TOGGLE SWITCH ACCESS
00010        8006  IODDB  EQU    $8006
00011        8007  CRB    EQU    $8007
00012 0000 36      MONACC PSH A
00013 0001 4F             CLR A
00014 0002 B7 8007        STA A  CRB        ENABLE DDB ACCESS
00015 0005 43             COM A
00016 0006 B7 8006        STA A  IODDB      MAKE ALL PB'S OUTPUTS
00017 0009 86 04          LDA A  #$04
00018 000B B7 8007        STA A  CRB        ENABLE IO ACCESS
00019 000E 86 03          LDA A  #$03
00020 0010 B7 8006        STA A  IODDB      PRE-SET E3 S,R INPUTS
00021 0013 F7 8006        STA B  IODDB      WRITE ACCESS WORD TO E3
00022 0016 32             PUL A
00023 0017 39             RTS
00024                    ◆     DDB = PIA DATA DIRECTION REGISTER-B SIDE
00025                    ◆     CRB = PIA CTRL REGISTER-B SIDE
00026                    ◆     IODDB = PIA I/O,DIRECTION REG-B SIDE
00027                     END
 IODDB   8006
 CRB     8007
 MONACC  0000

 TOTAL ERRORS 00000
```

FIGURE 19 – ROM Access Subroutine

* $ is Motorola Resident Assembler syntax for a hexadecimal number.

Figure 20 presents a tabular comparison of command sets for JBUG, MINIBUG and MIKBUG monitors. Any two pairs of these monitors may be used to configure the MEK/D2 computer to maximum advantage to suit the application through use of the dual monitor access logic described in Figure 12. A comparison of the commands of Figure 20 reveals that an excellent combination might be a MINIBUG II/MINIBUG III configuration. This would provide capability for memory test, punching and loading of binary tapes as well as access to the powerful software edit functions of Trace and Breakpoint insertion.

Monitor Function	JBUG	MINIBUG II	MINIBUG III	MIKBUG	Notes
Display Internal Registers	R	R	R	R	1
Load RAM from Tape	L	L	L	L	
Dump RAM to Tape (Punch)	P	P	P	P	2
Memory Examine/Change	M	M	M	M	3
Go to Entered Address and Execute	G	G	G	G	4
Set Terminal Baud Rate	—	S	S	—	S
Test Memory	—	W	—	—	6
Punch Binary Tape from RAM	—	Y	—	—	7
Load Binary Tape to RAM	—	Z	—	—	7
Abort Program Execution (Escape)	E	—	—	—	
Trace One Instruction	N	—	N	—	
Set a Breakpoint	V	—	V	—	8
Reset a Breakpoint	V	—	U	—	
Continue Execute from Breakpoint	E, G	—	C	—	
Delete All Breakpoints	V	—	D	—	8
Print Addresses of All Breakpoints	—	—	B	—	
Trace N Instructions	—	—	T	—	

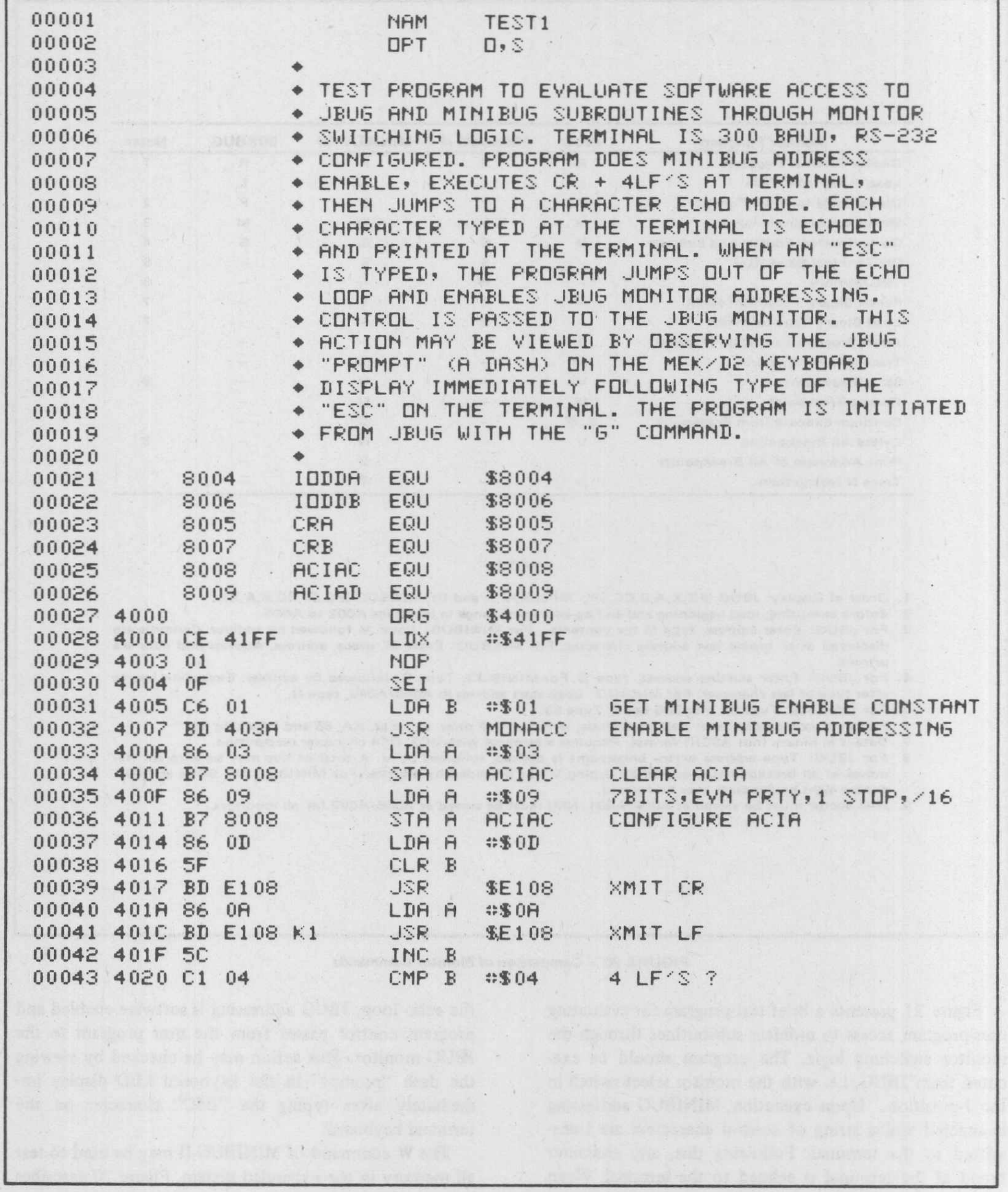
NOTES

1. Order of Display: JBUG (PC,X,A,B,CC,SP); MINIBUG II and III, MIKBUG (PC,SP,CC,B,A,X).
2. Before executing, load beginning and ending address of range in locations A002 to A005.
3. For JBUG: Enter address, type M for contents. For MINIBUG: Enter M followed by address. Contents are displayed after typing last address character. For MIKBUG: Enter M, space, address. Address and data are printed.
4. For JBUG: Enter starting address, type G. For MINIBUG: Type G, followed by address. Execution begins after type of last character. For MIKBUG: Load start address in A048/A049, type G.
5. For 110 Baud: Type S1. For 300 Baud: Type S3.
6. Performs six memory tests: walking address, write/read all ones, all zeros, AA, 55 and "Walking Bit."
7. Data is in binary (not ASCII) format. Requires a terminal with DC2, DC4 character recognition.
8. For JBUG: Type address where breakpoint is desired, followed by V. A total of five may be entered. Removal of all breakpoints executed by typing V not preceded by address. For MINIBUG III: Same as JBUG except eight breakpoints may be entered.
9. IRQ vector must be stored at A000/A001, NMI must be stored at A006/A007 for all monitors.

FIGURE 20 — Comparison of Monitor Commands

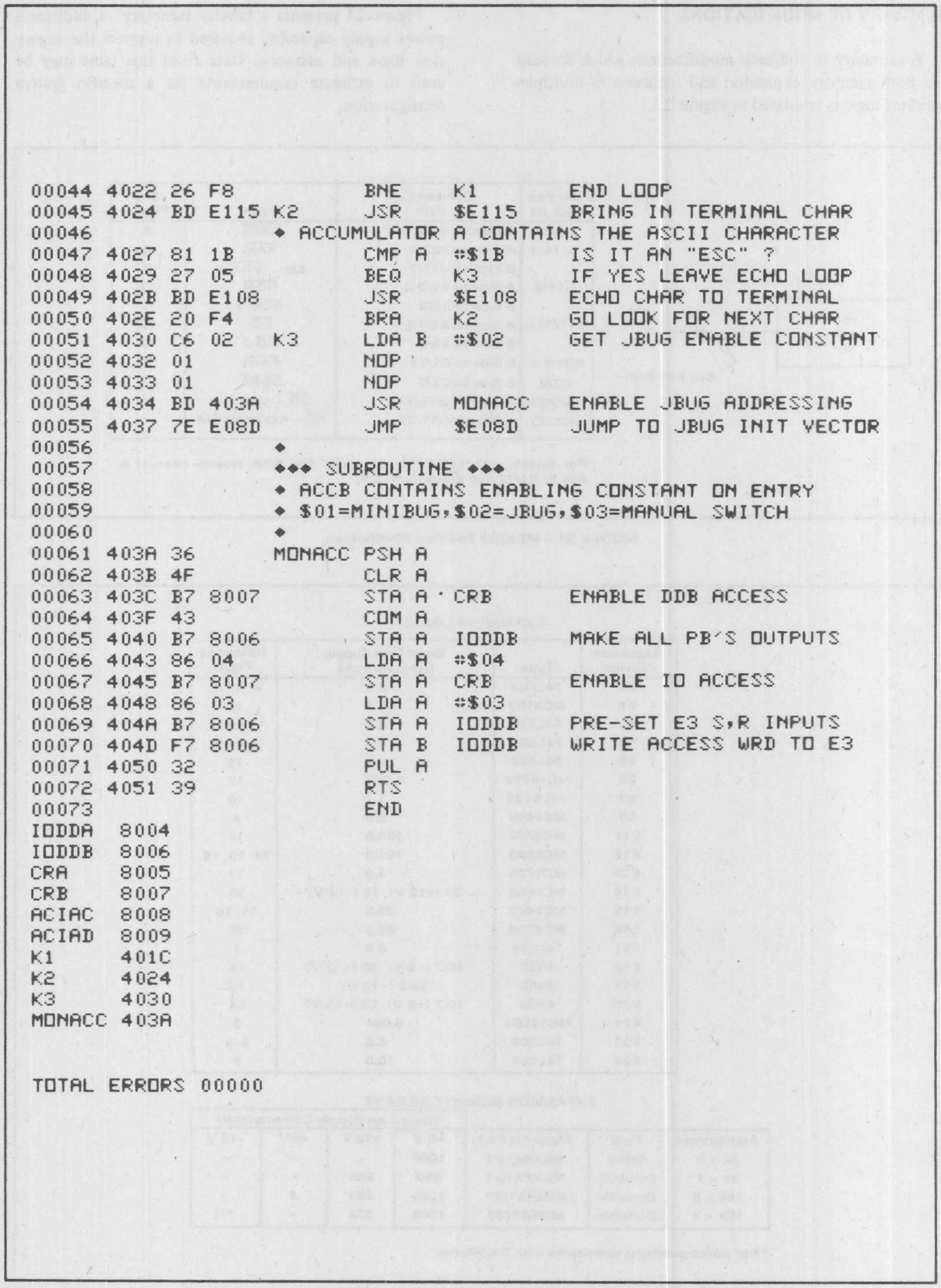
Figure 21 presents a brief test program for evaluating user-program access to monitor subroutines through the monitor switching logic. The program should be executed from JBUG, i.e. with the monitor select switch in the J-poisition. Upon execution, MINIBUG addressing is enabled and a string of control characters are transmitted to the terminal. Following this, any character typed at the terminal is echoed to the terminal. When the character "ESC" is typed, the program jumps from the echo loop, JBUG addressing is software enabled and program control passes from the user program to the JBUG monitor. This action may be checked by viewing the dash "prompt" in the keyboard LED display immediately after typing the "ESC" character on the terminal keyboard.

The W command of MINIBUG II may be used to test all memory in the expanded system. Figure 20 describes the use of this command.



```
                 NAM   TEST1
                 OPT   O,S
           ◆
           ◆ TEST PROGRAM TO EVALUATE SOFTWARE ACCESS TO
           ◆ JBUG AND MINIBUG SUBROUTINES THROUGH MONITOR
           ◆ SWITCHING LOGIC. TERMINAL IS 300 BAUD, RS-232
           ◆ CONFIGURED. PROGRAM DOES MINIBUG ADDRESS
           ◆ ENABLE, EXECUTES CR + 4LF'S AT TERMINAL,
           ◆ THEN JUMPS TO A CHARACTER ECHO MODE. EACH
           ◆ CHARACTER TYPED AT THE TERMINAL IS ECHOED
           ◆ AND PRINTED AT THE TERMINAL. WHEN AN "ESC"
           ◆ IS TYPED, THE PROGRAM JUMPS OUT OF THE ECHO
           ◆ LOOP AND ENABLES JBUG MONITOR ADDRESSING.
           ◆ CONTROL IS PASSED TO THE JBUG MONITOR. THIS
           ◆ ACTION MAY BE VIEWED BY OBSERVING THE JBUG
           ◆ "PROMPT" (A DASH) ON THE MEK/D2 KEYBOARD
           ◆ DISPLAY IMMEDIATELY FOLLOWING TYPE OF THE
           ◆ "ESC" ON THE TERMINAL. THE PROGRAM IS INITIATED
           ◆ FROM JBUG WITH THE "G" COMMAND.
           ◆
    8004   IODDA EQU   $8004
    8006   IODDB EQU   $8006
    8005   CRA   EQU   $8005
    8007   CRB   EQU   $8007
    8008   ACIAC EQU   $8008
    8009   ACIAD EQU   $8009
4000             ORG   $4000
4000 CE 41FF     LDX   #$41FF
4003 01          NOP
4004 0F          SEI
4005 C6 01       LDA B #$01     GET MINIBUG ENABLE CONSTANT
4007 BD 403A     JSR   MONACC   ENABLE MINIBUG ADDRESSING
400A 86 03       LDA A #$03
400C B7 8008     STA A ACIAC    CLEAR ACIA
400F 86 09       LDA A #$09     7BITS,EVN PRTY,1 STOP,/16
4011 B7 8008     STA A ACIAC    CONFIGURE ACIA
4014 86 0D       LDA A #$0D
4016 5F          CLR B
4017 BD E108     JSR   $E108    XMIT CR
401A 86 0A       LDA A #$0A
401C BD E108 K1  JSR   $E108    XMIT LF
401F 5C          INC B
4020 C1 04       CMP B #$04     4 LF'S ?
```

FIGURE 21 – Test Program



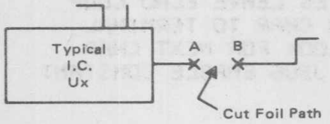
```
00044 4022 26 F8          BNE    K1       END LOOP
00045 4024 BD E115 K2     JSR    $E115    BRING IN TERMINAL CHAR
00046              ◆ ACCUMULATOR A CONTAINS THE ASCII CHARACTER
00047 4027 81 1B          CMP A  #$1B     IS IT AN "ESC" ?
00048 4029 27 05          BEQ    K3       IF YES LEAVE ECHO LOOP
00049 402B BD E108        JSR    $E108    ECHO CHAR TO TERMINAL
00050 402E 20 F4          BRA    K2       GO LOOK FOR NEXT CHAR
00051 4030 C6 02   K3     LDA B  #$02     GET JBUG ENABLE CONSTANT
00052 4032 01             NOP
00053 4033 01             NOP
00054 4034 BD 403A        JSR    MONACC   ENABLE JBUG ADDRESSING
00055 4037 7E E08D        JMP    $E08D    JUMP TO JBUG INIT VECTOR
00056              ◆
00057              ◆◆◆ SUBROUTINE ◆◆◆
00058              ◆ ACCB CONTAINS ENABLING CONSTANT ON ENTRY
00059              ◆ $01=MINIBUG,$02=JBUG,$03=MANUAL SWITCH
00060              ◆
00061 403A 36      MONACC PSH A
00062 403B 4F             CLR A
00063 403C B7 8007        STA A  CRB      ENABLE DDB ACCESS
00064 403F 43             COM A
00065 4040 B7 8006        STA A  IODDB    MAKE ALL PB'S OUTPUTS
00066 4043 86 04          LDA A  #$04
00067 4045 B7 8007        STA A  CRB      ENABLE IO ACCESS
00068 4048 86 03          LDA A  #$03
00069 404A B7 8006        STA A  IODDB    PRE-SET E3 S,R INPUTS
00070 404D F7 8006        STA B  IODDB    WRITE ACCESS WRD TO E3
00071 4050 32             PUL A
00072 4051 39             RTS
00073                     END
IODDA  8004
IODDB  8006
CRA    8005
CRB    8007
ACIAC  8008
ACIAD  8009
K1     401C
K2     4024
K3     4030
MONACC 403A

TOTAL ERRORS 00000
```

FIGURE 21 (Continued) — Test Program

SUMMARY OF MODIFICATIONS

A summary of foil-path modifications which account for both memory expansion and inclusion of multiple-monitor logic is tabulated in Figure 22.

Figure 23 presents a tabular summary of additional power supply capability required to support the expansion logic and memory. Data from this table may be used to estimate requirements for a specific system configuration.



Cut Foil Path At	Connect Cut Path	Term	See Figure
U6/36	A Side to E4/3, E7/1	$\overline{\text{DBES}}$	3
*U11/4	A Side to E23/1	$\overline{\text{RAM}}$	9
	B Side to U11/6	4/5 – $\overline{\text{VRA}}$ 2	9
U11/9	A Side to E5/2,4	$\overline{\text{ROM}}$	12
	B Side to E5/3	$\overline{\text{ROM}}$ J	12
U11/12A	A Side to E5/10;12	$\overline{\text{I/O}}$	12
	B Side to E5/8	$\overline{\text{I/O}}$ J	12
U23/3,4	B Side to E1/12	$\overline{\text{Reset}}$	12
U7/8	B Side to E7/9	$\overline{\text{DBRE}}$	10
U20/23	A Side to U11/12	$\overline{\text{I/O}}$ – User PIA	–
U21/23	A Side to U11/12	$\overline{\text{I/O}}$– Keyboard PIA	–

*For system with 16.5K memory only. For 8.5K system, connect A side to E1/11 and B side to U11/5.

FIGURE 22 – MEK/D2 Foil Path Modification

EXPANSION LOGIC

Expansion Device	Type	Worst Case Supply Currents (mA)	Reference Figure
E1	74LS04	6.6	4, 8, 12
E2	MC8T97	98.0	12
E3	74LS74	8.0	12
E4	74LS00	4.4	3, 4, 12
E5	74LS32	9.8	12
E6	MC14507	0.008	12
E7	74LS133	1.1	10
E9	MC1455	6.0	4
E11	MC6830	130.0	14
E12	MC6850	105.0	14, 15, 16
E13	MC1455	6.0	17
E14	MC1488	25 (+12 V), 15 (–12 V)	15
E15	MC1489	26.0	15, 16
E16	MC8T96	89.0	16
E17	74LS74	8.0	4
E18	4N33	10.7 (+5 V), 80 (+12 V)	16
E19	4N33	20.0 (+12 V)	16
E20	4N33	10.7 (+5 V), 20 (+12 V)	16
E21	MC14503	0.004	3
E22	74LS08	8.8	4, 6
E23	74LS86	10.0	9

EXPANSION MEMORY ARRAYS

			Worst-Case Supply Currents (mA)			
Architecture	Type	Motorola Part	+5 V	+12 V	-5 V	-12 V
2k x 8	Static	MEX6812-1	1000	–	–	–
8k x 8	Dynamic	MEX6815-1	860	300	–	–
16k x 8	Dynamic	MMS68100*	1200	333	4	–
16k x 8	Dynamic	MMS68103	1200	333	–	1.7

*Not board-geometry compatible with EXORciser.

FIGURE 23 – DC Power Supply Requirements for MEK/D2 Expansion

CONCLUSION

The technqiues discussed in this note add the following capability to the basic MEK/D2 kit microcomputer

* Power-up auto-reset
* Switch-selectable monitor operation
* RS-232 or current-loop data terminal operation at all standard baud-rates
* RAM expansion to 16.5K bytes
* ROM-resident subroutine acquisition by user program
* Operation with JBUG, MINIBUG II and III or MINIBUG ROM monitors

AN-773

A CRT TERMINAL USING THE M6800 FAMILY

Prepared by:
Joe Roy and Dusty Morris
Systems Engineering

This Note describes an M6800-based CRT Controller. A display format of 24 rows of 80 characters is featured. A Motorola M3000 Video Monitor is utilized.

A CRT Terminal Using the M6800 Family

By Joe Roy and Dusty Morris

This article describes a versatile M6800 based CRT Controller for "glass-teletype," smart, programmable, and intelligent CRT terminals. While a complete duplication of the entire package may be beyond the capabilities of most readers, some of the design features should be of particular utility in other construction projects. Of particular interest is the exploitation of the biphase clock architecture of the M6800 system, providing higher throughput, more I/O handling capability, less interference patterns during refresh memory accesses, and easier task-orientated multiple processor implementation in a CRT terminal than is possible with other approaches. Let's look at the features of this CRT terminal:

1. 24 rows of 80 characters.
2. 7 X 9 uppercase characters in a 9 X 12 dot block; shifted lower case through the use of a custom programmed MCM6832 16K Binary ROM.
3. Conventional non-interlaced raster scan.
4. Blink, half-intensity, video invert, underline, and non-display FACS (Field Attribute Codes); embedded FACS with optional widened memory capability.
5. Alternating inverted/non-inverted cursor.
6. 50/60 Hz field rate is logic selectable; display is centered for both 50 and 60 Hz.
7. Transparent accesses of Refresh Memory by VIA and MPU.
8. Limited graphics implementation with no changes in basic design philosophy.
9. Design philosophy facilitates up-grading a simple economical terminal with upward compatible software and hardware to a task-oriented multiple processor intelligent terminal.
10. MPU is unburdened from overhead of refresh memory contention and is free to service keyboard, edit functions, serial communications, and high speed control such as floppy disk.

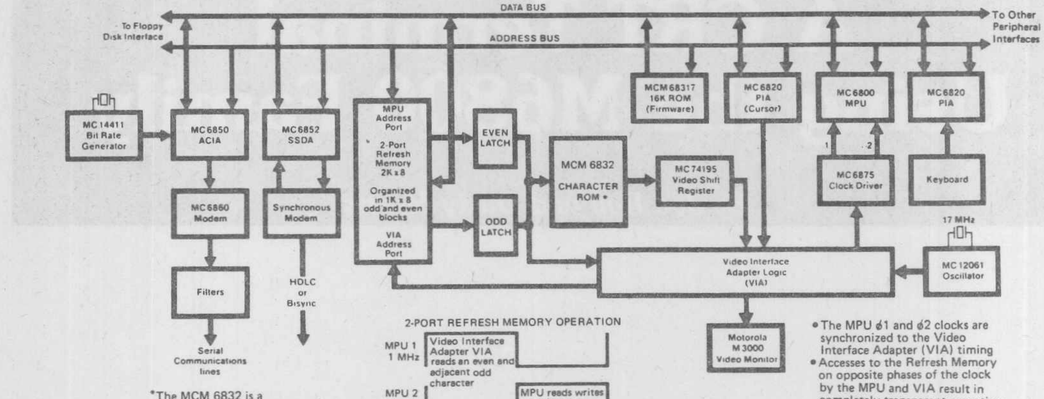
The basic configuration for a microprocessor-controlled CRT terminal is shown in Figure 1. An MC6800 microprocessor executes the CRT terminal executive firmware routine and jumps to driver subroutines when servicing the keyboard, serial synchronous or asynchronous interface, floppy disk formatter, and other peripherals. Cursor movements, R/W, and all editing functions are programmable and under microprocessor control. Actual refresh of the CRT monitor display is done with a configuration of SSI/MSI hardware called a VIA (Video Interface Adapter). The VIA provides video, vertical sync, and horizontal sync to the Motorola M3000 (or equivalent) monitor. The monitor must meet the requirements specified in Figure 2.

The MPU and VIA share the CRT Refresh Memory. Since the processor clock is derived from the VIA, both are synchronized. As shown in Figure 1 timing diagram, the VIA accesses memory for CRT refresh during clock phase Ø1, while the MPU accesses memory during Ø2. The Refresh Memory is organized in an odd address block and an even address block. Both an even and the adjacent odd address characters are transferred during a Ø1 access, whereas a single character is transferred to the MPU during a Ø2 access. The "odd/even memory" concept allows characters to be pulled from memory at a rate (≃ 2 MHz) sufficient to update the CRT. The "interleaved clocking" of memory makes it look transparent to both the MPU and the VIA. That is, neither delays the access of the other to memory. Consequently, less MPU overhead results than in other approaches.

Note that the interleaved clocking of memory is unaffected by cycle stretching of either MPU Ø1 or Ø2 . . . techniques used for refreshing dynamic memories, synchronizing other I/O, or interfacing slow memory.

The Refresh Memory provides a bidirectional data bus to the MPU and a two byte output bus for screen refresh. The two bytes of display data are pipelined to even and odd latches which are alternately enabled as data to the address inputs of an MCM6832 character ROM. The address is an ASCII character which the

- Single M6800 Microprocessor Controls CRT, Keyboard, Serial Communications Interface, and Floppy Disk in Smart/Intelligent CRT terminal.
- Unique 2-Port Refresh Memory Configuration eliminates contention overhead for memory accesses.


DATA BUS
ADDRESS BUS
To Floppy Disk Interface
To Other Peripheral Interfaces
MC 14411 Bit Rate Generator
MC 6850 ACIA
MC 6852 SSDA
MPU Address Port
2-Port Refresh Memory 2K x 8
Organized in 1K x 8 odd and even blocks
VIA Address Port
EVEN LATCH
ODD LATCH
MCM 6832 CHARACTER ROM *
MC 74195 Video Shift Register
MCM 68317 16K ROM (Firmware)
MC 6820 PIA (Cursor)
MC 6800 MPU
MC 6820 PIA
1
2
MC 6875 Clock Driver
Keyboard
17 MHz
MC 12061 Oscillator
Video Interface Adapter Logic (VIA)
Motorola M 3000 Video Monitor
MC 6860 Modem
Synchronous Modem
Filters
HDLC or Bisync
Serial Communications lines
2-PORT REFRESH MEMORY OPERATION
MPU 1 1 MHz
Video Interface Adapter VIA reads an even and adjacent odd character
MPU 2 1MHz
MPU reads writes any charactor location

*The MCM 6832 is a custon 16K binary ROM. selected for 450 ns access, and programmed for row select.

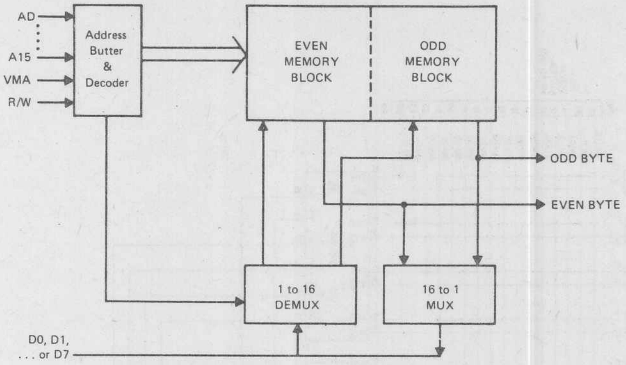
Note: DMA circuitry can grab the Address Data bus from the MPU during 2 d required

- The MPU ϕ1 and ϕ2 clocks are synchronized to the Video Interface Adapter (VIA) timing
- Accesses to the Refresh Memory on opposite phases of the clock by the MPU and VIA result in completely transparent operation.
- There is no annoying flickering or blanking of the screen during data transfers.

Figure 1. M6800 Terminal Block Diagram

1. Character Matrix	(Columns		7	7	
2.	(Rows		9	9	
3. Character Block	(Columns		9	9	
4.	(Rows		12	12	
5. Frame (refresh) Rate			50 HZ	60 HZ	
6. Rows of Characters			24	24	
7. Active Scan Lines (line 6 times line 4)			288	288	
8. Delay before Vertical Sync (No. of scan lines)			1667 μs (31)	0	
9. Vertical Sync Width (No. of scan lines)		4.52 ms	215 μs (4)	215 μs (4)	1.18 ms
10. Delay after Vertical Sync (No. of scan lines)			2634 μs (49)	968 μs (18)	
11. Total Scan Lines (Line 7+8+9+10)			372	310	
12. Horizontal Frequency (line rate) (line 11 times line 5)			18.6 KHZ (53.76 μs)	18.6 KHZ (53.76 μs)	
13. Character Rate (line 12 times line 18)			1.8972 MHZ	1.8972 MHZ	
14. Characters/Row			80	80	
15. Delay before Horiz. Sync. μsec:	(Character times)		2.1 μs (4)	2.1 μs (4)	
16. Horizontal Sync Width μsec:	(Character times)		4.7 μs (9)	4.7 μs (9)	
17. Delay after Horiz. Sync. μsec:	(Character times)		4.7 μs (9)	4.7 μs (9)	
18. Total character times (line 14+15+16+17)			11.5 μs (102)	11.5 μs (102)	
19. Clock Rate (line 13 times line 3)			17.074800 MHZ	17.074800 MHZ	
20. Display size			12″	12″	
21. Character Time (reciprocal of line 13)			527 ms	527 ms	
22. MPU Clock (line 19÷2X line 3)			948.6 KHZ	948.6 KHZ	
23. Clock Time (reciprocal of line 19)			58.6 μs	58.6 μs	

Figure 2 Video Monitor Timing (Motorola Display Products M3000 Monitor meets these requirements)



2K x 8 equals one page

	Hex Address
Even Byte	0000
Odd Byte	0001
Even Byte	0002
Odd Byte	0003
Even Byte	
Odd Byte	07FF

Figure 3.

ROM maps into a block of dots. The particular row of dots (0 to 11) in the block is determined by four "row select" inputs from the VIA. In a raster scan system, each ASCII character is presented 12 times to the character ROM with a sequential row select each time in order to paint the complete character on the screen.

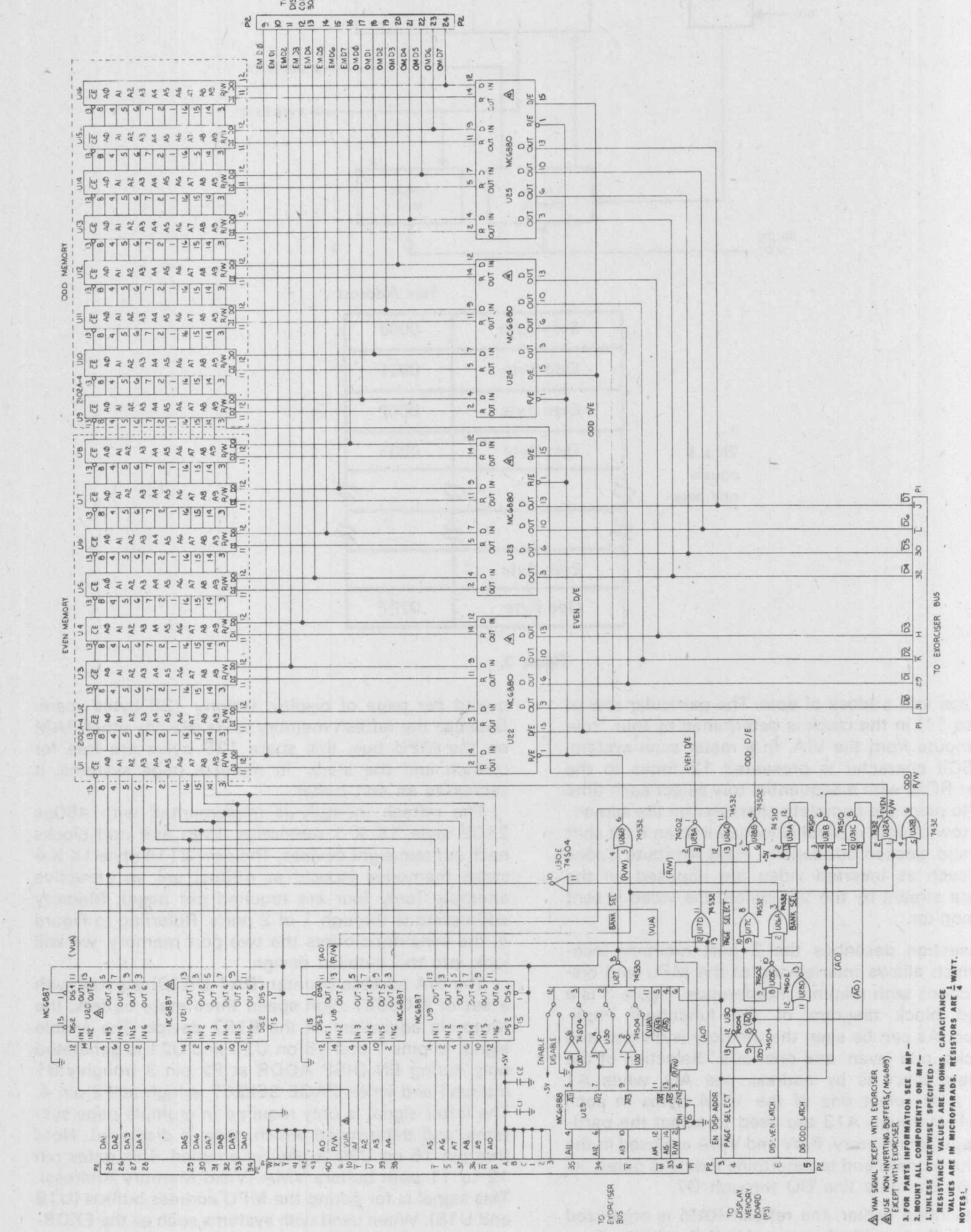
Each row of dots is parallel loaded into an 8 bit shift register and clocked out serially. Field Attribute Codes (FACS) such as inverted video are imposed on the serial data stream by the VIA before the video is sent to the monitor.

This section describes the 2 port memory technique which allows interleaving of the MPU and display functions with minimal interference. Figure 3 is a simplified block diagram of the functional implimentation. As can be seen the memory is divided into two blocks; one 'even' and one 'odd.' Selection of the appropriate block is by address line AO, while A1 through A10 select one of the 1024 bytes in each block. A11 through A13 are used to select the particular "page" of memory. R/W and VMA are used in the read/write process and to determine if data is gated 'in' or 'out' on the data line DO through D7.

As described earlier, the refresh RAM is organized as a 1K X 8 even block and a 1K X 8 odd block. Even and odd blocks are interleaved to form a 2K X 8 "page." Of the 2048 bytes, 1920 (80 times 24) are required per page of display, leaving 128 bytes spare. Because the refresh memory looks like any other RAM on the MPU bus, this spare 128 bytes are free for scratch and the stack. In multiple page systems, it serves as an edit buffer.

The refresh memory is implemented with 450ns 2102 style 1K X 1 memories. Even and odd blocks each contain eight devices. The new 2114 style 1K X 4 static memories (spec'd at 450ns) are an attractive alternate (only four are required per page). Memory addressing is through 1 of 2 ports. Referring to Figure 4, the schematic offers the two port memory, we will look into the detailed design.

The VIA address counter (DA1-DA10) is gated with a set of MC6887 high speed three state buffers, the Motorola equivalent of the 8797 device. The enable signal to pins 1 and 15 on U20 and U21 is generated only during EN DISP ADDR at P2 pin 3 (roughly 01 interval) and when PAGE SELECT is high at P2 pin 4. The latter signal is only required in multiple page systems, and determined which page is displayed. Note that pin 15 on U20 is always enabled. This gates pin 12 to 11 path buffers VMA (Valid Memory Address). This signal is for gating the MPU address buffers (U18 and U19). When used with systems such as the EXORcisor where a block of addresses must be unconditionally protected, this signal should be VUA (Valid Users Address).



NOTES:

1. UNLESS OTHERWISE SPECIFIED: RESISTANCE VALUES ARE IN OHMS. CAPACITANCE VALUES ARE IN MICROFARADS. RESISTORS ARE $\frac{1}{4}$ WATT.
2. MOUNT ALL COMPONENTS ON MP–
3. FOR PARTS INFORMATION SEE AMP–

4. USE NON-INVERTING BUFFERS (MC6889) EXCEPT WITH EXORCISER

5. VMA SIGNAL EXCEPT WITH EXORCISER

Figure 4. Display Memory Board

The selection of U18 and U19 is decoded by U27 which generates BANK Select. The particular bank (1 of 8 pages) is strap selectable (U29 and U30). The ENABLE/DISABLE switch on U27 provides a means of overlaying other chunks of memory with the same address. The other gating for BANK SELECT is VMA (described above), $\overline{A14}$, $\overline{A15}$, and $\overline{EN}$ $\overline{DISP}$ $\overline{ADDR}$ (EN MPU or roughly Ø2 interval). AO and R/$\overline{W}$ from the MPU are gated at all times because pin 15 on U18 is tied to ground.

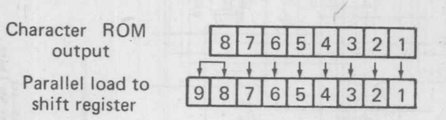
Devices U22 through U25 are MC6880 high speed bidirectional data buffers for multiplexing the Even and Odd Memory bytes into the MPU data bus ($\overline{DO}$ - $\overline{D7}$) and vice versa. In systems requiring a non-inverted data bus, MC6880 is replaced with MC6889. During an MPU read, either U22/U23 or U24/U25 is enabled by EVEN D/E or ODD D/E respectively. The buffered LSB of the MPU address, AO, determines which signal is active. During an MPU write, both U22/U23 and U24/U25 are enabled into their respective Even and Odd Memory blocks. The buffered LSB of the MPU address, AO, determines whether EVEN R/W or ODD R/W is active.

The refresh data is an even byte (EMDØ-7) and an odd byte (OMDØ-7).

Memory for Graphics Applications

Alphanumeric refresh memories are organized on a character basis. Each code stored in memory represents a 7 X 9 pattern in a 9 X 12 dot block. The dot pattern is stored in a character ROM addressed by the character code in the RAM. The repetition of a limited set of symbols on the screen to construct messages makes it possible to use a smaller amount of memory than would be required in a full graphics application where every dot is addressable as a memory location.

A "limited" graphics set of symbols for line drawings and forms is usually implemented with a special character ROM. The nine horizontal dots per character are provided by the ROM. However, most ROMS are organized by eight. It is usually acceptable to get the ninth bit from one of the eight ROM outputs, by parallel load of the 8th bit of ROM into both the eighth and ninth bits of the shift register.



A full graphics capability requires every possible dot on the screen to be stored in memory. Since the pattern is stored directly in RAM, all alphanumeric patterns are generated external to the refresh loop. Accordingly, the character ROM is placed on the MPU bus, and the dual latches drive the shift register directly. As in the alphanumeric controller, the RAM delivers two bytes (even and odd) when addressed by the VIA for refresh. Read and write addressing by the MPU is efficiently handled by bit addressing rather than byte addressing. The complete 64 K address structure of the MC6800 is decoded by hardware; only one of the eight MPU data bits is used for transfers.

A graphics terminal dedicates an MPU to the keyboard and I/O transfers with the refresh memory. All calculations (e.g. vectors), curves, rotations are done in an outboard high speed processor (e.g. microprogrammed bipolar slice). An interface between the MPU and the higher speed processor provides means for control and exchange of input parameters and results.

DISPLAY CONTROL

The DISPLAY CONTROL consists of circuitry to: sequentially access ASCII characters from the 2-PORT REFRESH MEMORY; generate character row selects; load row patterns into the Parallel-to-Serial Shift Register; serially shift the row pattern through Field Attribute circuits to the monitor as video; provide blanking, horizontal sync, and vertical sync signals; perform cursor compare and generate cursor block.

Character ROM

The purpose of the Display Control circuitry is to paint the contents of the character ROM at the designated positions on the CRT screen. Custom character ROMS contain 9 X 16 dot matrix patterns for alpha-numeric characters of various domestic and foreign fonts, limited graphics symbols, control characters, or combinations of all from the above. The addresses of the character dot matrix patterns correspond to their ASCII code representation in the case of alphanumeric and control characters. (Assignment is somewhat arbitrary for graphic symbols.)

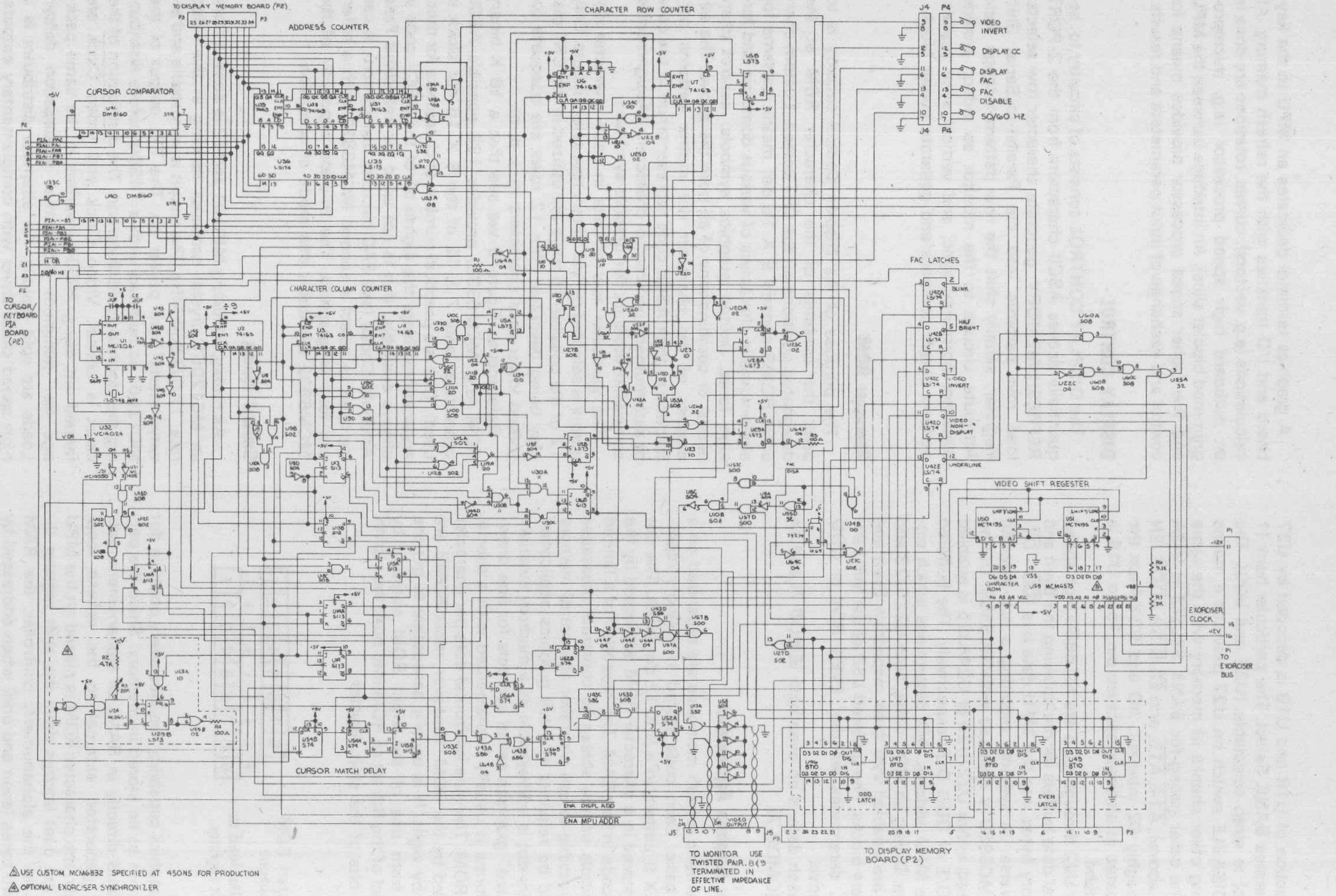
The particular row of dots in each character dot matrix is selected by four binary row select inputs. Only 12 of the possible 16 rows are utilized in the CRT display being described. The 12 rows are adequate for shifted lower-case characters (g, j, p, q).

Referring to Figure 5 we see that a 16 K binary ROM (e.g. MCM6832) provides 128 possible ASCII characters (only 7 X 12 of the 8 X 16 dot block is used). This is a practical number of characters for most applications. Note the eighth bit of the ASCII code is always available. If it is not used for imbedded Field Attribute Codes (see FAC circuitry description), it can be used to select a second MCM6832 with a different font or graphics. The tri-state capability of the MCM6832 facilitates this mode of operation.

General Timing

All timing, including MPU Ø1 and Ø2 is derived from an MC12061 crystal oscillator running at the video rate, 17.074800 MHz. The circuit is very stable and always starts from power up. The TTL output of the MC12061 is buffered with a 74S04 before distributing the clock to avoid distortion. The distribution of the clock is critical. Video Clock and $\overline{\text{Video Clock}}$ are fanned out using 74S04 inverters in the same package (for minimum differential propagation delay). Loads are split equally. An alternate distributor is a high speed clock driver with complementary outputs. An important consideration is to keep all 17 MHz logic close together and in proximity to a ground. This will minimize noise and distortion.

There is another method for arriving at 17.074800 MHz. Although it is more expensive, a phase-locked



NOTES:

1. UNLESS OTHERWISE SPECIFIED: RESISTANCE VALUES ARE IN OHMS. CAPACITANCE VALUES ARE IN MICROFARADS. RESISTORS ARE $\frac{1}{4}$ WATT.
2. MOUNT ALL COMPONENTS ON MP–
3. FOR PARTS INFORMATION SEE AMP–

△4 OPTIONAL EXORCISER SYNCHRONIZER

△5 USE CUSTOM MCM6832 SPECIFIED AT 450NS FOR PRODUCTION

Figure 5. Schematic — Display Control Board

loop or phase-locked oscillator is used. The line frequency is the reference and vertical sync the comparison signal. In areas where the line frequency varies radically from nominal, the more expensive system is often required to reduce the visible beat on the CRT screen.

A ÷ 9 counter divides video rate down to a character rate of 1.9 MHz. On/Off decodes from the ÷ 9 counter are resynchronized in 74S113 high speed flip flops. The signals provide the General Timing in Figure 6.

Decodes off the Character Column Counter provide horizontal timing (Figure 7). Vertical timing is from the Character Row Counter (Figure 8).

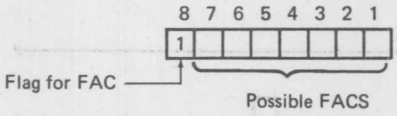
Note the operation of the Address Counter. It must repeat each character address 12 times to paint a complete line of 80 characters on the screen. This requires storing the address of the first character in each line (function of the 74LS latches). Another function of the Address Counter is to advance to address 64 during V blank. (Effectively, this amounts to a start address of 128 since the memory is addressed two bytes at a time.)

Another function of the Display Control Electronics is cursor compare. The contents of the Address Counter are compared with PIA data for coincidence. The DM8160's are exclusive-or comparators. PIA data is a binary address which is manipulated by the MPU for cursor control.

The other circuitry in the Display Control portion generates invert/non-invert cursor block when cursor compare is sensed and furnishes FAC (Field Attribute Code) logic.

FACS (Field Attribute Codes)

There are two popular methods for handling FACS. In the wide memory method, the memory size is increased by adding bits to each character in memory. Each bit controls a different attribute code for that character. The other method imbeds FAC characters in refresh memory. The eighth bit of the ASCII code is usually decoded as a FAC flag.



When it is a logical one, the other seven bits are latched as FACS. Once latched, the FAC applies to all subsequent characters until another FAC code is decoded. The only exception is at the end of a character line; all FACS are hardware reset. This scheme's advantage is low cost to implement; the disadvantage is the use of a memory location per FAC code. Not only is character density decreased, but the individual characters in a string can not be accented with FACS. It is possible to get around this drawback by stripping FACS from the memory before display, but requires extra hardware and is a programming nightmare. When individually accented characters are required, it is usually better to implement a wider memory.

The wide memory approach to FACS may seem clumsy at first glance . . . the MPU has an 8 bit bus. How are 8 bit MPU transfers done? Construct two pages of memory — a page of ASCII characters 2K X 8, and a page of attributes (2K X 1, 2, 3, 4, . . ., 8). The attribute page is a mask and need only be accessed when the attribute changes or must be read.

For simplicity, the imbedded FAC method was implemented in the CRT terminal. Few changes in Display Control circuitry are required for a wide memory approach.

CURSOR/KEYBOARD

Referring to Figure 9 we can see how to add a cursor/keyboard interface to this "glass teletype."

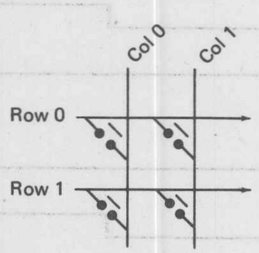
Cursor

The cursor address is stored as the contents of a PIA. The eight least significant binary bits are PBO - PB7 and the three most significant binary bits are PAO - PA2. There is a one-to-one correspondence between the binary cursor address and a memory location on the screen. The assignment of bits in the PIA is for programming convenience. An STX instruction to the A side of the PIA writes the higher order byte of the index register into the A side of the cursor PIA and the lower order byte into the B side of the cursor PIA (provided the address lines to RSO and RS1 are reversed).

The cursor address is binary rather than X-Y because the binary address manipulations are more frequent. When X-Y addressing is required (e.g. communication interface), a conversion subroutine is called.

Keyboard

Either a fully-encoded or non-encoded keyboard can be designed. Whichever, a PIA provides the interface to the MPU. In a fully-encoded keyboard, the keyboard hardware generates a strobe and an ASCII encoded character corresponding to the depressed key. All debounce is handled by the keyboard hardware. The strobe pulse applied to the PIA CA or CB inputs causes an interrupt to the MPU. The MPU reads the ASCII character through the PIA and performs the appropriate function. For an alphanumeric character, the MPU writes the data at the present cursor location and increments cursor PIA contents. For control characters, the corresponding commands are executed. (e.g. space, carriage return, insert, delete, etc.)



A non-encoded keyboard is a set of switches wired in a column/row matrix.

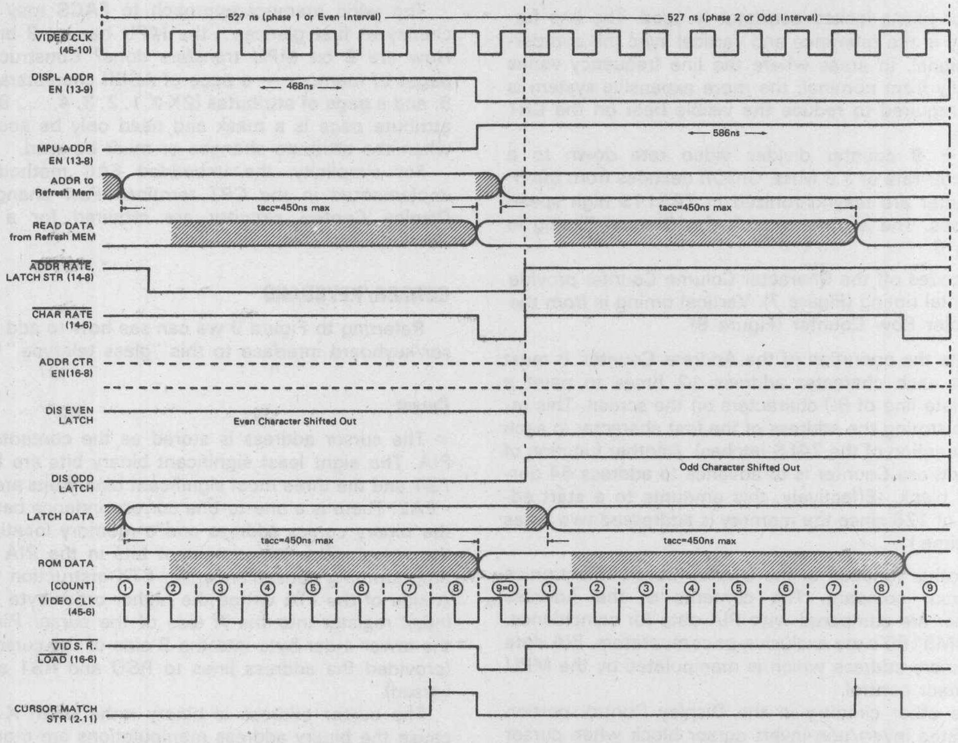


Figure 6. General Timing

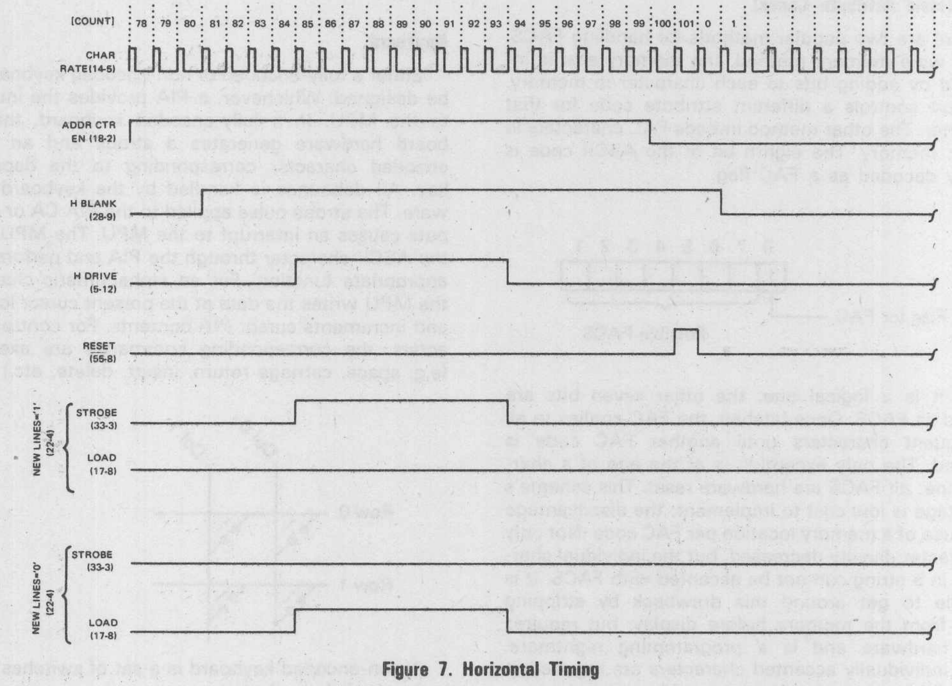


Figure 7. Horizontal Timing

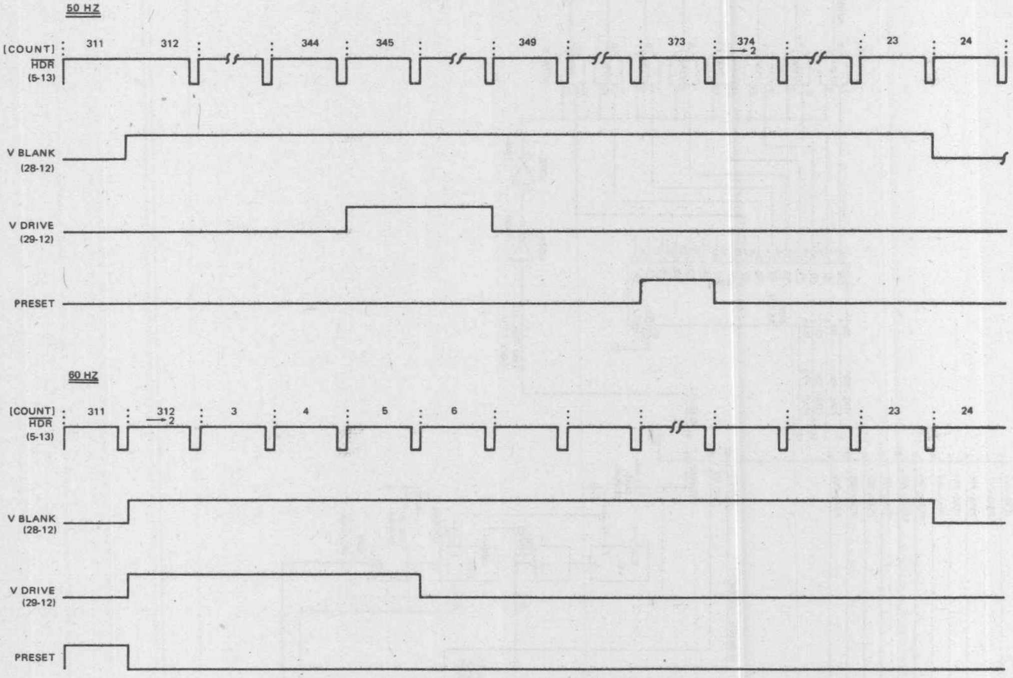
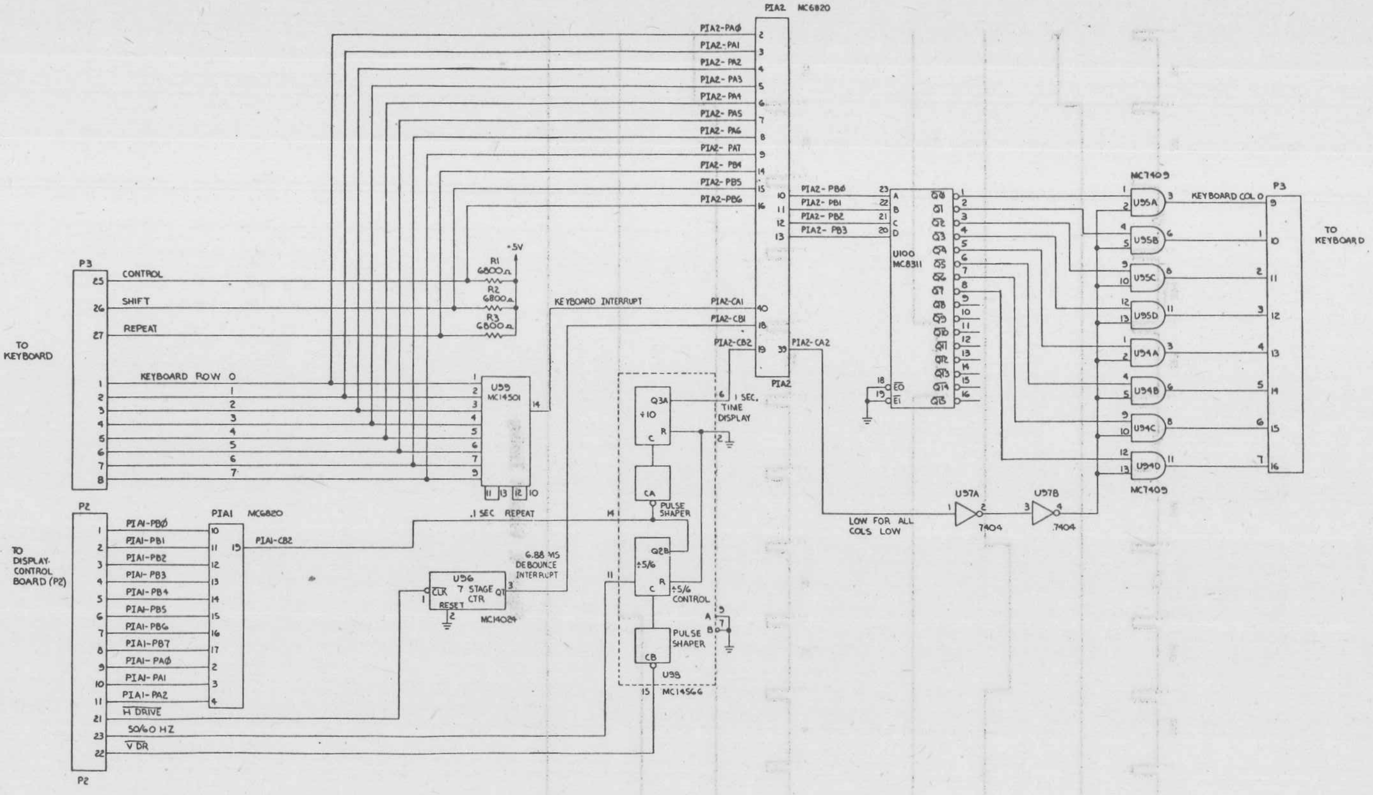


Figure 8. Vertical Timing



4. ADDITION TO MEX6820 I/O MODULE
3. FOR PARTS INFORMATION SEE AMP-
2. MOUNT ALL COMPONENTS ON MP-
1. UNLESS OTHERWISE SPECIFIED:
RESISTANCE VALUES ARE IN OHMS. CAPACITANCE
VALUES ARE IN MICROFARADS. RESISTORS ARE $\frac{1}{4}$ WATT.
NOTES:

Figure 9. Cursor/Keyboard PIA Board

If the columns are scanned one at a time, and the rows read back, the simultaneous depression of any one or two keys can be discerned. For n keys, diodes are wired in series with the switch contacts.

The scanning of columns, reading of rows, and switch debounce are under software control. Keys are strategically placed in the matrix so the column and row location easily translate into an ASCII code. Cost savings and flexibility of this non-encoded keyboard versus a fully-encoded keyboard sometimes justifies the additional MPU overhead in a basic CRT terminal.

A non-encoded approach is described here. Referring to the schematic, the keyboard columns are normally held low. When key(s) are depressed, lows appear on the keyboard row inputs. A keyboard interrupt is generated, which starts a scan. Keyboard column outputs are brought low in sequential order. Together with the Shift and Control PIA inputs, the active columns and rows are encoded as an ASCII character and temporarily stored. A 6.88 msec interrupt is provided for debounce and a PIA input for REPEAT (also .1 sec interrupt for this function and a 1 second interrupt for clock functions).

The software for a non-encoded keyboard ranges from simple to quite complex depending on how foolproof the algorithm is.

Comparison of CRT Terminal Architectures

The central design criteria of a modern CRT terminal is the method used for multiplexing refresh memory between the MPU and the display refresh circuitry. In this section we will discuss time-division multiplexing and priority multiplexing. Throughput versus hardware complexity for different techniques will be analyzed. Schemes which result in missing a character refresh during an MPU transfer are not considered. These techniques result in operator annoyance in many applications (e.g., Key-to-Disk), and are not appropriate for the modern CRT terminal.

Priority Multiplexing of Refresh Memory

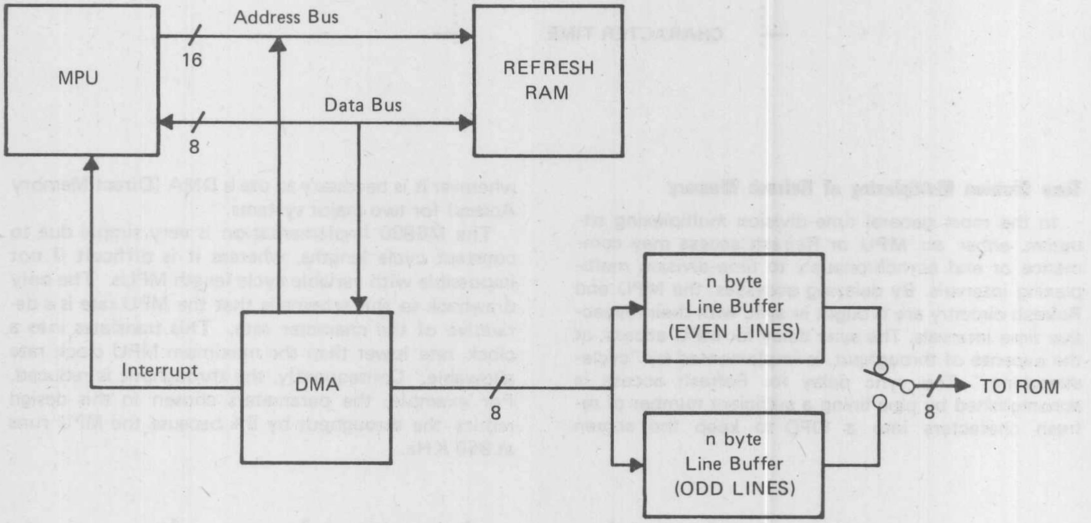
These techniques fall into two general categories:

A. MPU grabs Refresh Memory and locks out Refresh Circuitry.

B. Refresh circuitry grabs Refresh Memory and locks out MPU.

Method A results in zero burden on the MPU; however, a sufficient number of refresh characters must be pipe-lined into FIFO to keep the screen refreshed during an MPU access. The fast memory and complex hardware to accomplish this task is unattractive.

Method B is probably the most widespread technique in use today. The dual line buffer approach is

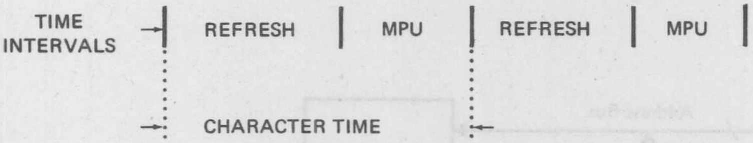

MPU
Address Bus
16
REFRESH RAM
Data Bus
8
Interrupt
DMA
8
n byte Line Buffer (EVEN LINES)
TO ROM
8
n byte Line Buffer (ODD LINES)

Dual Line Buffering Technique

representative of this class. All data for a line on the screen is stored in FIFO's (usually 80 bytes).

While the Odd line buffer is keeping the display refreshed, the Even line buffer is being filled with the next displayed line from Refresh RAM through a DMA channel. The functions are alternately reversed. In an alphanumeric terminal, average MPU burden is between 15 and 30% with peak burden on the MPU of 100% for 80-100 microseconds when loading either line buffer. The MPU is stalled during this transfer and cannot service high speed interfaces (e.g. Floppy Disk). With the addition of extra hardware, the Refresh page of the RAM is isolated from the processor bus during DMA, thus allowing the MPU to continue processing provided it attempts no accesses to memory. In conclusion, the dual-line buffers and DMA configuration is more expensive, has higher parts count, and imposes a severe burden on the MPU compared to the time-division multiplexed technique.

Furthermore, the DMA approach is very inefficient for full graphics, since there are no recirculations of the line buffer as in an alphanumeric display.



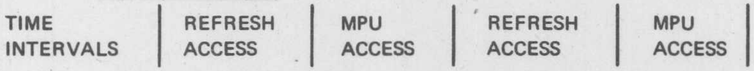
Time Division Multiplexing of Refresh Memory

In the most general time-division multiplexing situation, either an MPU or Refresh access may commence or end asynchronously to time-division multiplexing intervals. By delaying accesses, the MPU and Refresh circuitry are brought in sync with their respective time intervals. The sync delay for MPU access, at the expense of throughput, is implemented by "cycle-stretching." The sync delay for Refresh access is accomplished by pipe lining a sufficient number of refresh characters into a FIFO to keep the screen refreshed during an MPU synchronization delay. The FIFO is typically two bytes deep. The disadvantage of this technique is the requirement for fast memory. It is considerably simplified if the time-division multiplexing is made synchronous with character rate. Sync delay is no longer required for refresh and the FIFO circuitry is eliminated (a single byte latch is still required). Also memory speed is reduced. This simplification is not always acceptable; e.g. it may result in excessive MPU "cycle-stretching" at slow character rates.

The technique for sharing CRT Refresh Memory described in this article is a special case of time-division multiplexing. The access intervals for Refresh and MPU are $\phi 1$ and $\phi 2$, respectively.

As $\phi 1$ and $\phi 2$ clocking signals are outputted even during wait states, no matter how long this MPU is forced into a wait state the screen will remain refreshed. Since there is no overlap of accesses, no contention circuitry is required and the MPU burden is zero at all times. While you may not want or need to duplicate the entire circuitry described in this article, the bi-phase memory access technique may be used wherever it is necessary to use a DMA (Direct Memory Access) for two major systems.

The M6800 implementation is very simple due to constant cycle lengths, whereas it is difficult if not impossible with variable cycle length MPUs. The only drawback to this scheme is that the MPU rate is a derivative of the character rate. This translates into a clock rate lower than the maximum MPU clock rate allowable. Consequently, the throughput is reduced. For example, the parameters chosen in this design reduce the throughput by 5% because the MPU runs at 950 KHz.



AN-774

A SIMPLE HIGH SPEED BIPOLAR MICROPROCESSOR ILLUSTRATES SYSTEM DESIGN AND MICROPROGRAM TECHNIQUES

Prepared by:
Bill Blood
Applications Engineering

High speed bipolar LSI 4-bit slice circuits can significantly reduce processor system package count. This is shown with a microprocessor which uses only 10 integrated circuit packages to perform 2's complement add, subtract, multiply and divide.

INTRODUCTION

The engineer familiar with MOS microprocessors learns new skills when designing a higher performance bipolar LSI system. Bipolar LSI 4-bit slice circuits are building blocks allowing the designer to configure a processor architecture, size, and instruction set for optimum performance. Additional flexibility is gained through the use of microprogramming because the processor can perform more complicated instructions such as multiply and divide or, alternately, can be designed to take advantage of existing software.

The following text goes through a complete bipolar LSI system design. Steps include define the system, block out system sections, set up the microprogram structure, pick bipolar LSI parts, generate a microprogram, and finalize the system design. Three system goals are used throughout the project.

1. Maximize the use of bipolar LSI.
2. Show the flexibility of microprogramming.
3. Maintain the bipolar LSI speed advantage.

The system is being designed as a demonstration and technical support tool. However, the design flow and decisions involved represent a wide range of bipolar LSI processor systems.

DEFINE THE SYSTEM

System functional requirements must be defined prior to hardware design. This seems obvious, but changes later in the design cycle complicate programming and often increase package count. Figure 1 shows the I/O structure and program functions of this processor project. The system has 16 input lines which enter a 16-bit data word or two 8-bit words in parallel. Similarly, 16 output lines display the results as either one 16-bit word or two 8-bit words.

A program select input port selects which processor program is executed.

1. ADD, SUBTRACT, and EXCLUSIVE OR read two 8-bit operand inputs and generate an 8-bit answer. The eight most significant bits are held at zero.
2. MULTIPLY reads an 8-bit multiplicand and 8-bit multiplier. The answer is a 16-bit product.
3. DIVIDE requires two data inputs. First a 16-bit dividend is entered, then the 8-bit divisor. The answer is an 8-bit quotient and 8-bit remainder.

The final processor input is a start signal given after input lines and program select are present. Since start could be from a pushbutton or toggle switch, bounce elimination and pulse shaping are included in the system design.

BLOCK OUT THE SYSTEM

The major blocks of a bipolar LSI processor are ALU, I/O, microprogram sequencing control, and microprogram memory. These functions are interconnected as shown in Figure 2. The ALU block handles all arithmetic, logic, and shift operations. It also includes working registers for temporary storage and a means for the input of new data and output of results. The MICROPROGRAM SEQUENCING block generates the microprogram memory address and provides a method for sequencing through microprogram. It combines with BRANCH LOGIC to make tests for conditional jumps in program.

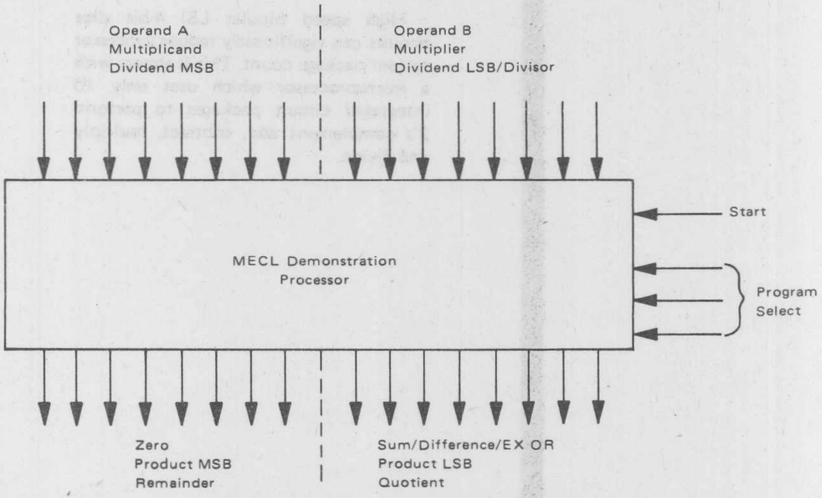


FIGURE 1 — I/O Description

The key to building a bipolar LSI system is in partitioning the MICROPROGRAM MEMORY block. The memory is divided into sections called fields, each capable of an independent function. Any combination of these fields can be selected to execute a microinstruction. In Figure 2, the PROGRAM FLOW field is separated into two parts. The INSTRUCTION section tells the microprogram sequencing logic how to generate the next program address. Instructions include increment, jump, branch on condition, jump to and from subroutine, etc. The NEXT ADDRESS section of the program flow field provides a destination address for direct jump or conditional branch instructions.

The BRANCH field sets up test parameters for conditional jumps in program. For example, a jump if ALU results equal zero would be accomplished by gating zero detect from the ALU to a branch input on the microprogram sequencer. The corresponding INSTRUCTION would be branch on condition and NEXT ADDRESS contains the conditional branch destination. In this manner it is possible to select any number or combination of test signals.

DATA and ALU fields go to the ALU and I/O block. The DATA field controls data transfers between the input ports, output ports, and internal working registers. The ALU field controls the various arithmetic, logic, and shift operations required to execute a program.

The system being developed here is designed around three LSI circuits as shown in Figure 2. Two MC10803s handle all ALU operations, provide working registers, and control the 16 input and 16 output lines. A single MC10801 provides logic for microprogram flow and addressing. Additional MC10801 and MC10803 information is given in the following key LSI parts section.

Processor programs are stored in the microprogram memory. This system uses four 10139 PROMs in a 32-word by 32-bit organization. At first it may seem surprising that add, subtract, multiply, divide, and exclusive OR are all stored in 32 memory words. However, each word is 32 bits wide so several processor functions can be performed in parallel. These seven integrated circuits form the processor nucleus. Three additional SSI parts provide for crystal oscillator, power-up reset, and start pulse shaping. Branch logic gates route test signals into the MC10801 for conditional jumps in program. The result is a high-speed bipolar processor with only ten integrated circuit packages.

Figure 2 can be used to define the term microinstruction. A microinstruction executes all functions in a microporgram memory word. A clock signal into the MC10801 microprogram sequencer logic puts a program address location on the address lines. The microprogram memory then sets up select lines on the MC10803 to read data in, operate on the data, and display or

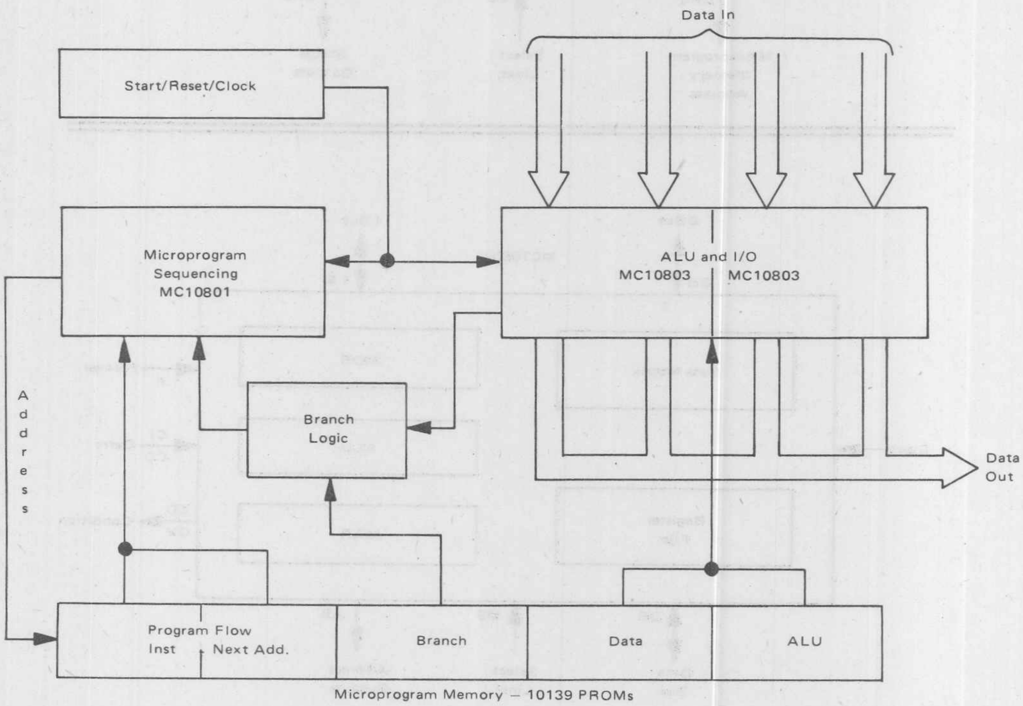


FIGURE 2 – Processor Block Diagram

store the results. In parallel with the ALU, a new microprogram address is being generated from microprogram memory and branch logic inputs to the MC10801. A second clock pulse gives the microprogram a new word address and clocks any ALU function into the appropriate storage register. System performance is measured by the microinstruction time and by the number of microinstructions required to execute a program.

The flexibility of microprogramming allows the processor to perform a wide variety of system functions. The five programs–add, subtract, exclusive OR, multiply, and divide–being developed are only a small sample of the possible combinations. The processor could be expanded for process control, data formatting, digital filtering, minicomputer design, peripheral controllers, etc. Four MC10803s will directly operate on 16-bit words and give a corresponding increase in I/O lines. Two MC10801s allow microprogram memory expansion to 4K words for more comprehensive programming.

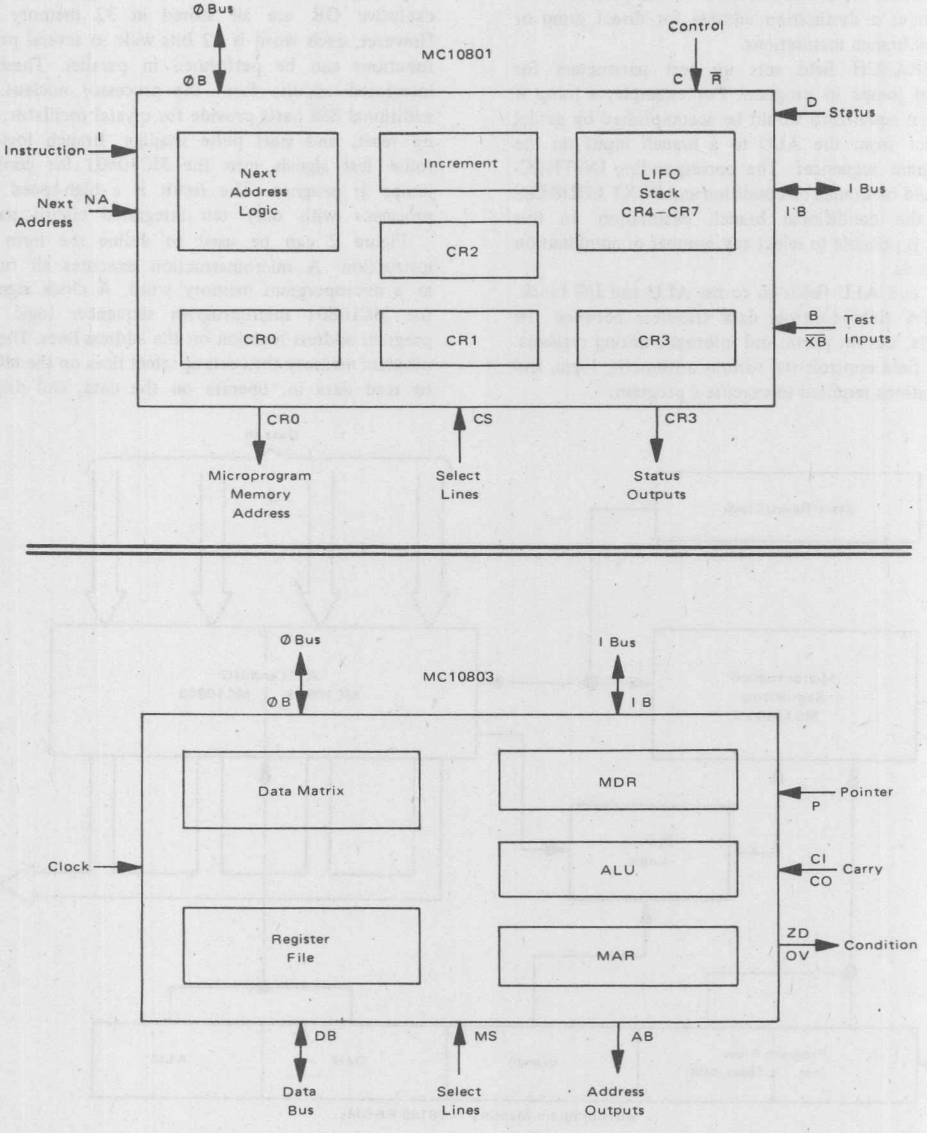


FIGURE 3 – Key LSI Parts

KEY LSI PARTS

Prior to continuing it is important to look at the MC10801 and MC10803 LSI parts. The various internal sections of each part are shown in Figure 3. MC10801 CR0 register holds the microprogram memory address. Sequencing information goes into the Next Address Logic block where it is decoded and the correct next address routed to CR0. An incrementer is used with several sequencing commands. For example, an increment command routes the CR0 outputs through the incrementer to CR0 inputs. Each clock pulse then advances the microprogram memory one location.

CR1 is used with program flow repeat cycles. The repeat count is loaded into CR1 enabling an individual instruction or sequence of instructions to be repeated until the cycle count is reached. CR2 is a general purpose register that can be used to store machine instructions or interrupt vectors. CR3 is a status or condition code register. Individual bits can be tested within the MC10801 for conditional jumps in program.

A 4-deep LIFO stack is included in the part for storing subroutine return address. A jump to subroutine command takes the present microprogram address through the incrementer and pushes it into the stack. Next Address inputs go to CR0 for the subroutine destination. A subroutine return pops the LIFO and routes the return address to CR0. A total of 16 different instructions are built into the MC10801 for various program flow requirements.

The MC10803 performs both data transfers and ALU operations. There are five I/O ports (I Bus, ϕ Bus, A Bus, D Bus, and P inputs) for transferring data to or from the part. Internally there are six storage registers. The MDR can be used to hold incoming or outgoing data. It also functions as an accumulator for the ALU. The MAR holds outgoing data for the A Bus. Four additional registers are contained in the internal register file block. A Data Matrix accepts data transfer commands and routes data between the various I/O ports and internal registers. The final block is an ALU which performs arithmetic, logic, and shift operations. Both the MC10801 and MC10803 are controlled by select lines which in turn are controlled by microprogram memory bits.

PROGRAM FLOW

Program flow charts describe the various processor operations and determine a microprogram instruction sequence. Figure 4 shows system data paths available to a programmer. These data paths, the ALU, and working registers are used in the following program flow diagrams and later to write the microprogram. All Figure 4 data paths and logic except the link bit storage are contained within the MC10803. Link bits are used with multiply and divide to hold shift in and shift out status. Available CR3 register bits in the MC10801 provide this function.

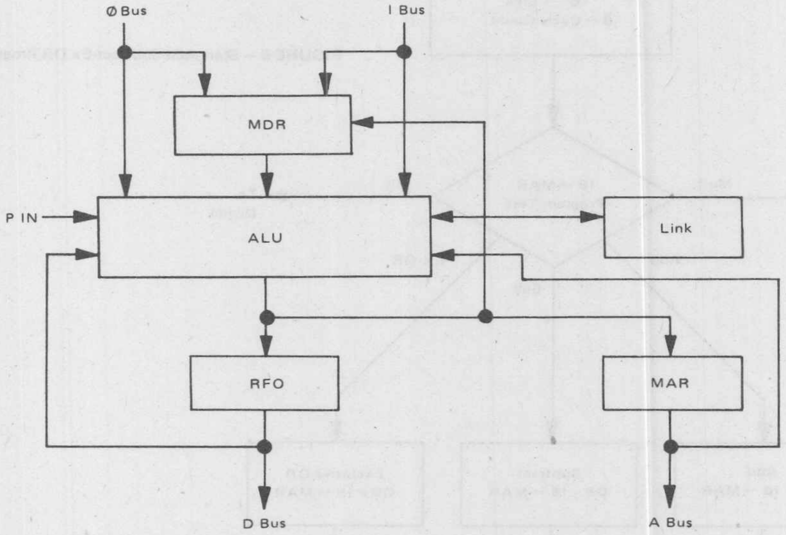


FIGURE 4 – Processor Data Paths

Figure 5 shows the flow patterns for add, subtract, and exclusive OR. Initially the processor is random on power-up or displaying the results of a previous calculation. It is in a continuous loop waiting for a start signal. After receiving start, the program performs several initialize functions. RFO is zeroed as required for add, subtract, and exclusive OR, see Figures 1 and 4. Setting up for a possible multiply or divide the link bit is set to zero, the ,complement of 8 is loaded in the MC10801 program cycle counter CR1, and I Bus data is transferred to the MAR. These functions are common to both multiply and divide. Location at this point in program saves microinstructions.

Program flow continues to a main decision point. Here, the processor looks at the program select inputs and picks one of five possible program directions. Add, subtract, and exclusive OR are each one step programs, Figure 5. The ALU looks at the ϕ Bus and I Bus inputs, performs the selected operation, and transfers the answer to MAR. The program jumps back to start and displays the answer.

Multiply is an implementation of Booth algorithm as shown below:

1. Load multiplier into MAR
2. Load multiplicand into MDR
3. Zero RFO and link bit
4. Set cycle counter to 8
5. Test MAR LSB and link bit

LSB	LINK	
0	0	GO TO 8
0	1	GO TO 6
1	0	GO TO 7
1	1	GO TO 8

6. Subtract RFO - MDR → RFO, go to 8
7. Add RFO + MDR → RFO
8. Arithmetic shift right RFO → MAR → LINK
9. Decrement cycle counter; if ≠ zero, go to 5
10. End.

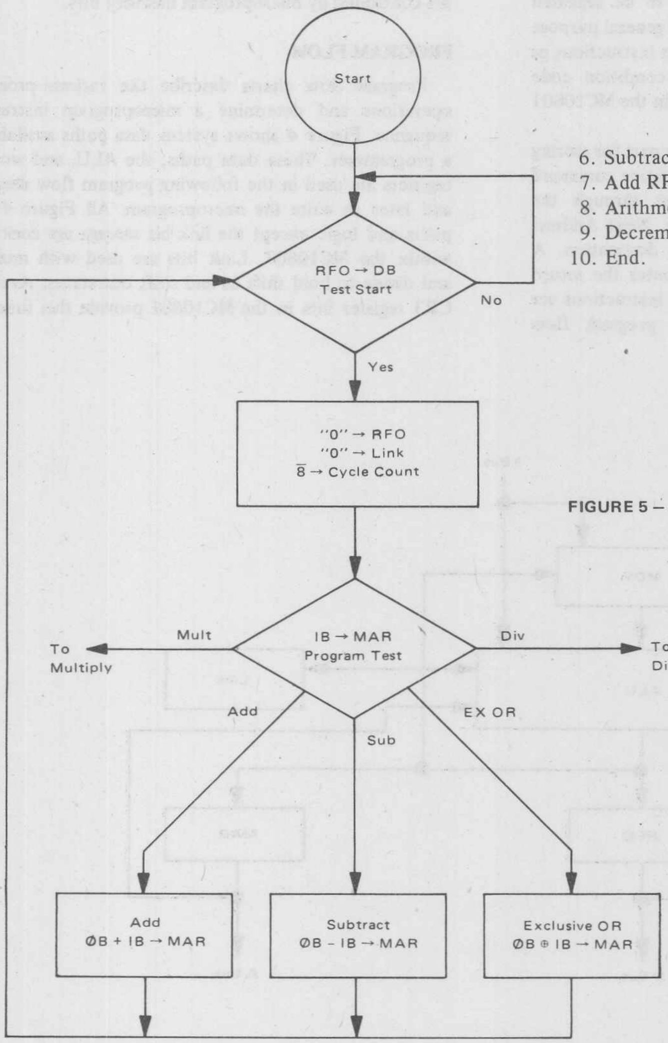


FIGURE 5 — Start-Add-Subtract-Ex OR Program Flow Diagram

The multiply flow diagram is given in Figure 6. Figure 5 sets up the multiplier, RFO, cycle counter, and link bit. Figure 6 loads the multiplicand and contains the program paths for add-shift, subtract-shift, or shift-only as required for 2's complement multiplication. As seen in the figure, the processor alternately tests the link bit and MAR LSB to select a program flow path. This simplifies the branch select compared with testing both status points in parallel. The program cycle counter is incremented after the double precision RFO/MAR shift right and tested for the end count. After 8 cycles, the program is complete. The result is a 16-bit answer with the 8 least significant bits in MAR and the 8 most significant bits in RFO.

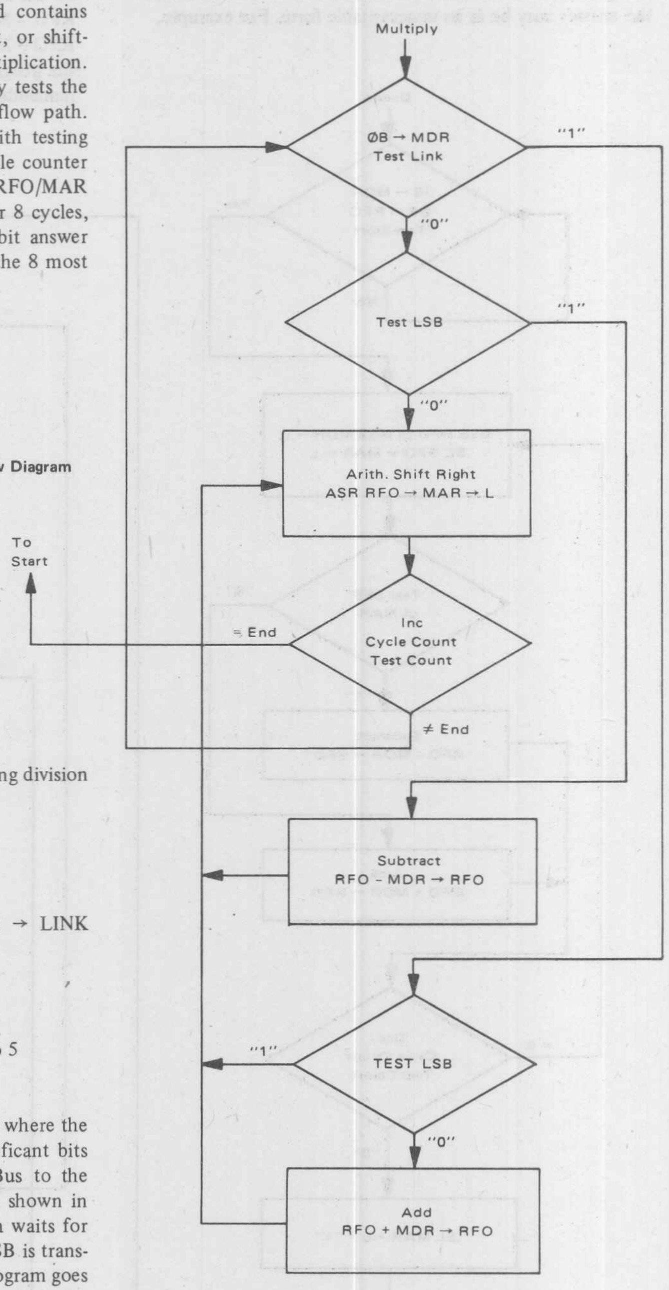


FIGURE 6 – Multiply Flow Diagram

Divide is an implementation of a non-restoring division algorithm as shown below:

1. Load dividend LSB into MAR
2. Load divided MSB into RFO
3. Load divisor into MDR
4. Set cycle counter to 8
5. MSB RFO exclusive NOR MSB MDR → LINK
6. Shift left RFO ← MAR ← LINK
7. Test MAR LSB; if zero, go to 9
8. Subtract RFO - MDR → RFO, go to 10
9. Add RFO + MDR → RFO
10. Decrement cycle counter; if ≠ zero, go to 5
11. Shift right MAR, link = "1"
12. Go to format correction.

The program flow diagram starts in Figure 5 where the program cycle counter is set and the least significant bits of the dividend are transferred from the I Bus to the MAR. The remaining divide program flow is shown in Figure 7. The first block is a start loop which waits for the divisor data on the I Bus. The dividend MSB is transferred to RFO and the divisor to MDR. The program goes through eight repetitive cycles, each setting link bit status, shifting left the MAR and RFO, and performing the divisor add or subtract with RFO. After the eighth cycle MAR is shifted left with a logic "1" forced into the LSB through the link bit.

At this point, the answer is numerically correct with the quotient in MAR and the remainder in RFO. However, the answer may be in an unacceptable form. For example, 37 ÷ 6 comes out 7, remainder -5. The program flow on the right side of Figure 7 checks the format by testing RFO - MDR = 0, RFO = 0, RFO + MDR = 0, and MSB of RFO = MSB of dividend on ϕ Bus. If the format is correct, the program jumps to start. If incorrect, the quotient and remainder are corrected as shown in Figure 7.

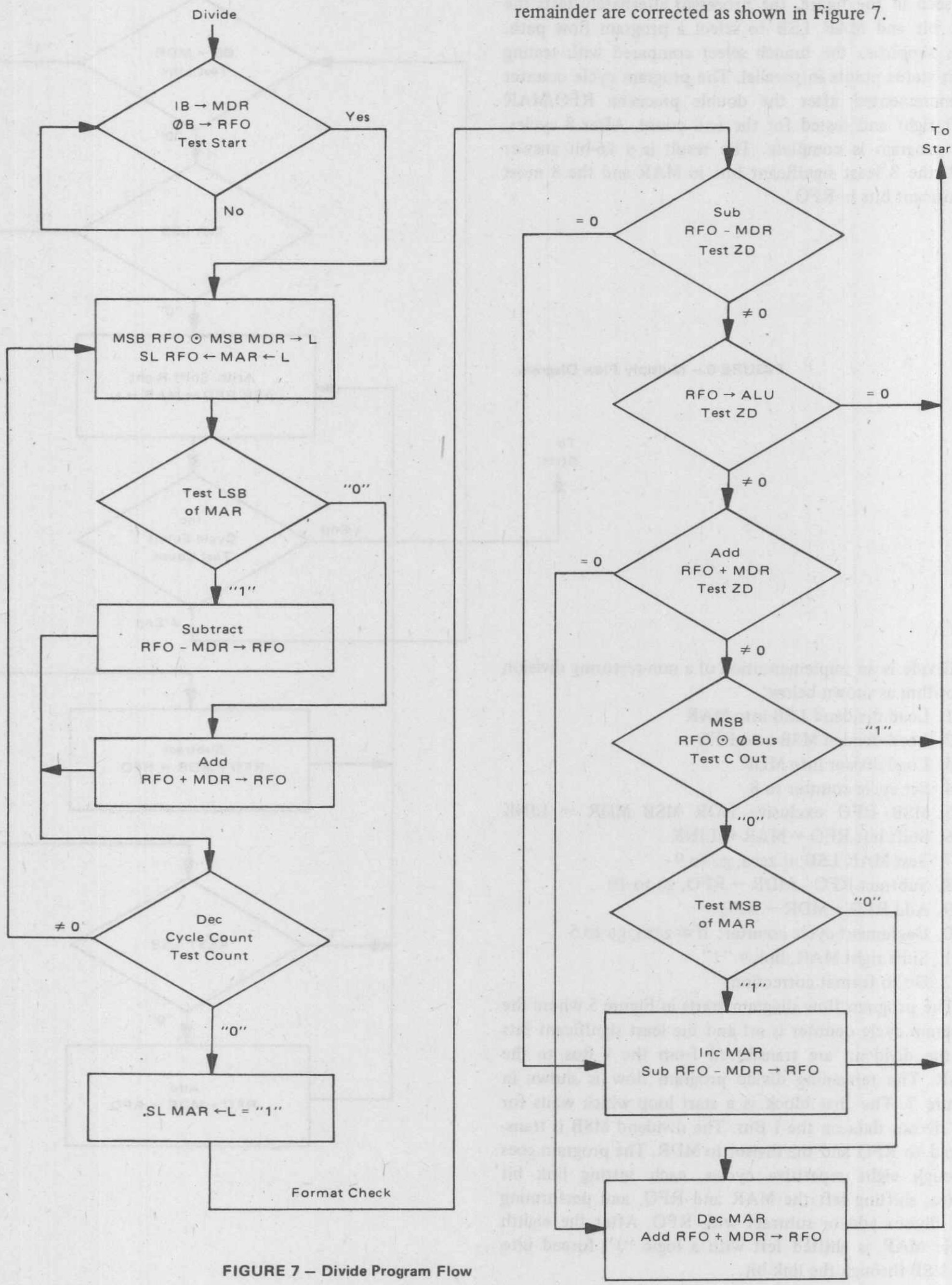


FIGURE 7 – Divide Program Flow

TIE IT ALL TOGETHER

Various test parameters can be identified from the program flow diagrams. These are 1) start, 2) program select, 3) shift right link bit, 4) MAR LSB, 5) cycle count, 6) MSB of RFO EX-NOR MDR (available on C out), and 7) zero detect. From the flow diagrams and the basic block diagram in Figure 2, the complete system is tied together as shown in Figure 8. Start and program select go directly to the MC10801. Cycle count is a feature of the MC10801 and requires no special treatment. Zero detect, MAR LSB, and C out are gated to the MC10801 as required for program test. Shift right link bit must be stored between microinstructions and uses bit 1 of register CR3. This bit can be tested internal to the MC10801 for program flow decisions.

Register CR3 holds three program status bits. CR3, bit 0, is a page address representing the fifth microprogram memory address bit. CR3, bit 1, is the shift right link bit and can be gated to the ALU C out (shift right input). CR3, bit 2 performs the same function for shift left and can be gated to ALU C in.

The Figure 8 microprogram memory fields have been refined from Figure 2. Individual bits control the program test inputs, shift link bits, C in, and MC10803 P inputs. The three LSI parts have been previously defined. The remaining blocks in Figure 8 are SSI. A dual flip-flop (10131) supplies bounce elimination and start signal pulse shaping. Leftover NAND and OR gates are used for power-up reset and a crystal oscillator clock.

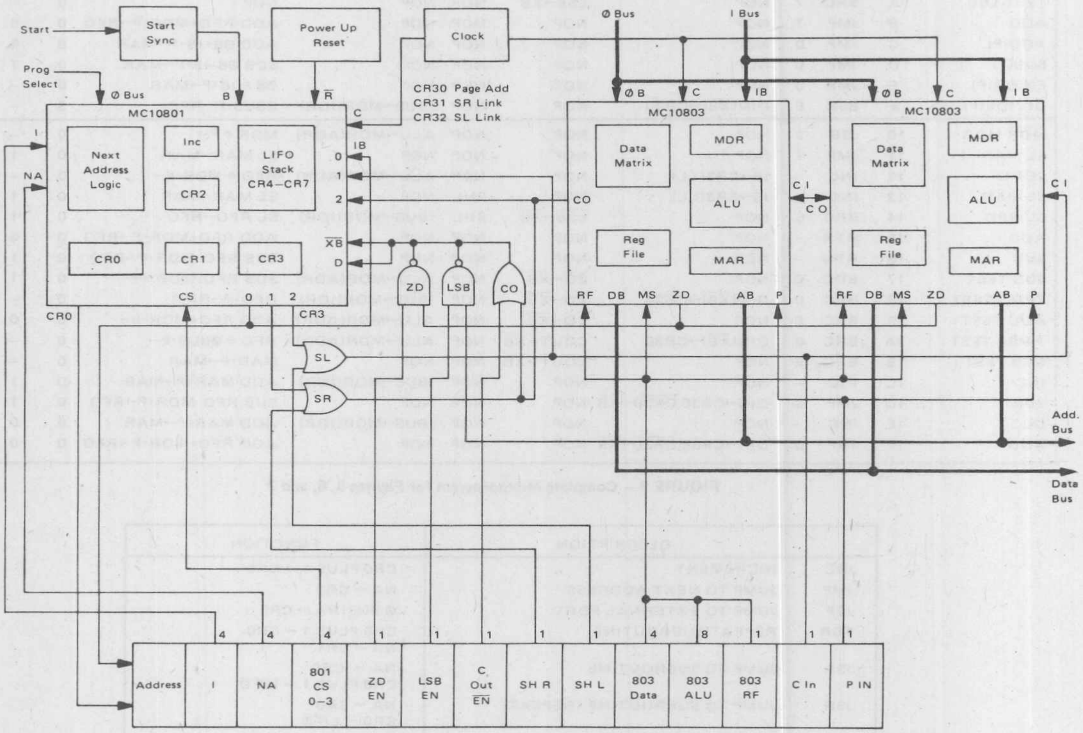


FIGURE 8 – Processor Logic Diagram

MICROPROGRAM THE MEMORY

The complete microprogram for Figures 5, 6, and 7 is given in Figure 9. A function column briefly describes each program step. Add column is the microprogram word address in hexadecimal format. Table 1 shows MC10801 sequencing instructions. INC and JMP are direct transfers of a new address to the microprogram address register, CR0. JEP is used for the five-way program select jump. Program select inputs are connected to the MC10801 ϕ Bus port and program select information corresponds to the individual program starting address. RSR loads the repeat cycle count into CR1. JSR jumps to the NA input address for a subroutine location and pushes a return address into the LIFO stack. If the subroutine is to be repeated, CR0 is pushed into the stack. Otherwise, CR0 plus 1 is loaded in the stack. This repeat function is automatically controlled by the cycle counter internal to the MC10801. RTN pops the LIFO stack into CR0 for a subroutine return. In a repeat mode, the

FUNCTION	ADD	I	NA	801 CR3	$\overline{XB}$	SH	DATA	ALU	RF	C_{IN}	P
READ	0	BSR	0	NOP	NOP	NOP	RFO→DB (FDB)	NOP	0	–	–
"0" RFO,L	1	RSR	8	IB→CR3(L)	NOP	NOP	IBUS→MDR (IDR)	RFO•P→RFO	0	0	0
PROG TEST	2	JEP	F	NOP	NOP	NOP	NOP	MDR•P→MAR	0	–	1
MULTIPLY(P)	3	JSR	5	NOP	NOP	NOP	ØBUS→MDR (ØDR)	NOP	0	–	–
END	4	JMP	0	NOP	NOP	NOP	NOP	NOP	0	–	–
TEST LINK	5	BRC	A	$\overline{CR31}$→$\overline{XB}$(L)	NOP	NOP	NOP	NOP	0	–	–
TEST LSB	6	BRC	9	NOP	$\overline{LSB}$→$\overline{XB}$	NOP	NOP	NOP	0	–	–
SHR MSB	7	INC	–	IB→CR3(L)	NOP	NOP	NOP	ASR RFO→RFO	0	1	–
SHR LSB	8	RTN	–	IB→CR3(L)	NOP	SHR	NOP	LSR MAR→MAR	0	1	–
SUB	9	JMP	7	NOP	NOP	NOP	NOP	SUB RFO-MDR•P→RFO	0	1	1
TEST LSB	A	BRC	7	NOP	$\overline{LSB}$→$\overline{XB}$	NOP	NOP	NOP	0	–	–
ADD	B	JMP	7	NOP	NOP	NOP	NOP	ADD RFO+MDR•P→RFO	0	0	1
ADD(P)	C	JMP	0	NOP	NOP	NOP	NOP	ADD ØB+IB•P→MAR	0	0	1
SUB(P)	D	JMP	0	NOP	NOP	NOP	NOP	SUB ØB-IB•P→MAR	0	1	1
EX OR(P)	E	JMP	0	NOP	NOP	NOP	NOP	ØB ⊕ IB•P→MAR	0	–	1
DIVIDE(P)	F	BSR	F	DIN($\overline{XB}$)→CR30	NOP	NOP	IBUS→MDR(IDR)	ØBUS•P→RFO	0	–	1
$\overline{MDR}$ MDR	10	JSR	2	NOP	NOP	NOP	ALU→MDR(ADR)	MDR ⊕ P→	0	–	1
SL MAR←1	11	JMP	7	NOP	NOP	NOP	NOP	SL MAR→MAR	0	1	–
SET L	12	INC	–	IB→CR3(LL)	NOP	NOP	ALU→MDR(ADR)	RFO ⊕ MDR•P→	0	–	1
SL MAR	13	INC	–	IB→CR3(LL)	NOP	SHL	NOP	SL MAR→MAR	0	1	–
SL RFO	14	BRC	6	NOP	$\overline{LSB}$→$\overline{XB}$	SHL	IBUS→MDR(IDR)	SL RFO→RFO	0	1	–
ADD	15	RTN	–	NOP	NOP	NOP	NOP	ADD RFO+MDR•P→RFO	0	0	1
SUB	16	RTN	–	NOP	NOP	NOP	NOP	SUB RFO-MDR•P→RFO	0	1	1
SUB TEST	17	BRC	C	NOP	$\overline{ZD}$→$\overline{XB}$	NOP	ALU→MDR(ADR)	SUB RFO-MDR•P→	0	1	1
ZERO TEST	18	BRC	0	DIN($\overline{XB}$)→CR30	$\overline{ZD}$→$\overline{XB}$	NOP	IBUS→MDR(IDR)	RFO•P→RFO	0	–	1
ADD TEST	19	BRC	E	NOP	$\overline{ZD}$→$\overline{XB}$	NOP	ALU→MDR(ADR)	ADD RFO+MDR•P→	0	0	1
⊕ MSB TEST	1A	BRC	0	DIN($\overline{XB}$)→CR30	COUT→$\overline{XB}$	NOP	ALU→MDR(ADR)	RFO ⊕ ØBUS•P→	0	–	1
MSB TEST	1B	BRC	E	NOP	COUT→$\overline{XB}$	NOP	NOP	MAR•P→MAR	0	–	1
INC	1C	INC	–	NOP	NOP	NOP	IBUS→MDR(IDR)	ADD MAR+P→MAR	0	1	0
SUB	1D	JMP	0	DIN→CR30,$\overline{CR30}$→$\overline{XB}$	NOP	NOP	NOP	SUB RFO-MDR•P→RFO	0	1	1
DEC	1E	INC	–	NOP	NOP	NOP	IBUS→MDR(IDR)	ADD MAR+P→MAR	0	0	1
ADD	1F	JMP	0	DIN→CR30,$\overline{CR30}$→$\overline{XB}$	NOP	NOP	NOP	ADD RFO+MDR•P→RFO	0	0	1

FIGURE 9 – Complete Microprogram for Figures 5, 6, and 7

DESCRIPTION		FUNCTION
INC	INCREMENT	CR0 PLUS 1 → CR0
JMP	JUMP TO NEXT ADDRESS	NA → CR0
JEP	JUMP TO EXTERNAL PORT	Ø BUS•NA → CR0
RSR	REPEAT SUBROUTINE	CR0 PLUS 1 → CR0 NA → CR1
JSR	JUMP TO SUBROUTINE	NA → CR0 CR0 PLUS 1 → LIFO
JSR	JUMP TO SUBROUTINE (REPEAT)	NA → CR0 CR0 → LIFO
RTN	RETURN FROM SUBROUTINE	LIFO → CR0
RTN	RETURN FROM SUBROUTINE (REPEAT)	LIFO → CR0 CR1 PLUS 1 → CR1
BRC	BRANCH ON CONDITION	CR0 PLUS 1 → CR0 (TEST = 0) NA → CR0 (TEST = 1)
BSR	BRANCH TO SUBROUTINE	CR0 PLUS 1 → CR0 (START) NA → CR0 ($\overline{\text{START}}$)

TABLE 1
Microprogram Flow Commands

cycle counter CR1 is automatically incremented on this instruction. BRC looks at the selected test parameter and depending on test status executes either an increment or jump to NA inputs. BSR, normally a conditional branch to subroutine, is used to enable the MC10801 Branch input for a jump on start signal instruction.

The NA column in Figure 9 is a jump destination. It is used with all flow instructions except INC and RTN which require no additional information for program flow. The NA field in this system is four bits wide and operates within a 16-word memory page. Page addressing is part of the 801 CR3 column in Figure 9.

The MC10801 CR3 register holds the memory page address and two shift link bits. This register is parallel loaded on an I Bus to CR3 command, see Table 2. Notice CR3 bit 0 is connected to IB0 in Figure 8. This holds the page address constant on the I Bus parallel load command. The multiply program requires testing status of the shift right link bit for a program flow decision. Testing is accomplished by gating CR3 bit $\overline{1}$ to $\overline{\text{XB}}$ and making a program flow decision with a BRC instruction. The last two Table 2 commands control page address. For conditional jumps between pages CR3 bit 0 is loaded from D In connected to $\overline{\text{XB}}$. Alternately, CR3 bit 0 can be toggled for unconditional jumps.

The remaining fields in Figure 9 are relatively straightforward. $\overline{\text{XB}}$ can be selectively programmed to zero detect, $\overline{\text{LSB}}$, or C Out. These are used with the BRC program flow instruction to make decisions in program. The SH R field gates the shift right link bit in MC10801 CR3 bit 1 to carry out of the MC10803. It is disabled for all other arithmetic functions. The SH L field performs a similar function for the shift left link bit. Additional information on shift operations follows in the paragraph on MC10803 ALU operations.

The data field selects four different MC10803 data transfer functions. FDB transfers the information in RFO to the D Bus for answer display. ØDR and IDR read the processor inputs and transfer data to the MDR accumulator. ADR routes the ALU output to MDR. This is used to modify the MDR contents as in Figure 9, word 10, or to avoid changing information in RFO or MAR (program words 12, 17, 19, and 1A).

The MC10803 ALU selects between RFO, MAR, MDR, ϕ Bus, I Bus, and P for operands. AND and exclusive OR are selected logic functions, with add and subtract selected for arithmetic. These functions combine with the P inputs for special operations: ALU = zero, word 1; MDR invert, word 10; and MAR decrement, word 1E. Five different shift combinations are formed with the ALU, MC10801 CR3, shift, and C In fields, as shown in Table 3. An IB → CR3 in the MC10801 CR3 field connects the shift out to a link bit. The shift field routes link bit to the shift input. Word 11 disables the shift left link allowing C In to become the shift input. Word 13 uses the shift left link for a rotate.

Only selected MC10801 and MC10803 functions have been described as required for this program. They are a small percent of the total combinations available to a system designer. The LSI circuits therefore adapt to a wide range of system architectures and applications. Additional information is available on component data sheets.

FUNCTION	DESCRIPTION
IB → CR3	PARALLEL LOAD CR3
$\overline{\text{CR31}}$ → $\overline{\text{XB}}$	TEST SHIFT RIGHT LINK
DIN → CR30	CHANGE PAGE ADDRESS FROM D IN
DIN → CR30, $\overline{\text{CR30}}$,→ $\overline{\text{XB}}$	TOGGLE PAGE ADDRESS

TABLE 2
MC10801 CR3 Commands

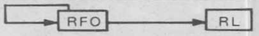
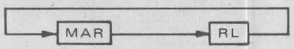
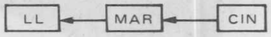
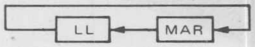
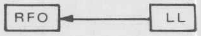
FUNCTION	OPERATION
ASR RFO → RFO	
LSR MAR → MAR	
SL MAR → MAR	
SL MAR → MAR	
SL RFO → RFO	

TABLE 3
ALU Shift Commands

THE FINAL SYSTEM

Figure 10 is a picture of the complete wirewrapped system. The three LSI parts are in 48-pin quad-in-line packages with memories and SSI in standard 16-pin packages. IC headers hold discrete resistors and capacitors for the crystal oscillator, start pulse shaper, and power-up reset. The four remaining packages are pull-down resistors as required for ECL signal lines. A 10 MHz clock (100 ns microinstruction time) gives program execution times of 500 ns for add, subtract, and exclusive OR. The longest multiply is 5.3 μs and the longest divide is 5.2 μs. Additional circuits used as pipeline registers would reduce the microinstruction time, but the goal of this project is to keep part count and cost down. System power is under 14 watts including drive for 16 LEDs used to display the answer.

Are bipolar LSI processors fast? This system is designed with the industry's fastest bipolar LSI circuits, the M10800 family, but does not fully utilize the speed potential to minimize part count. Even so, the system performs the 8-bit 2's complement multiplication approximately 100 times faster than a 1.0 μs MOS microprocessor, 10 times speed improvement is gained with clock time, and 10 times speed improvement from architecture and microprogramming advantages.

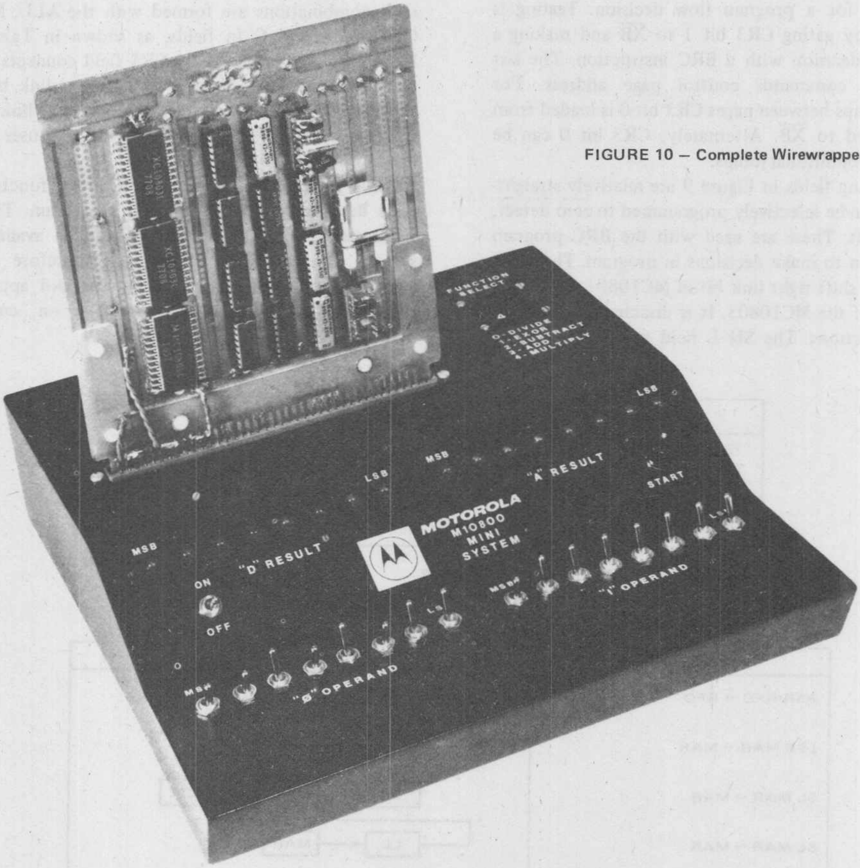


FIGURE 10 — Complete Wirewrapped System

AN-775

M6800 SYSTEMS UTILIZING THE MC6875 CLOCK GENERATOR/DRIVER

Prepared by
Stephen R. Bookout
Microprocessor Applications/Systems Engineering

This application note describes the use of the MC6875 clock generator/driver in M6800 based systems. Design examples will demonstrate the capabilities of the driver in systems using slow and/or dynamic memories. Multiprocessing and DMA methods are also covered.

UTILIZING THE MC6875 CLOCK GENERATOR/DRIVER

INTRODUCTION

Previous methods of implementing MPU clocks ranged from discrete components to expensive hybrid schemes with hardware fixes for dynamic and slow memory handshaking.

The MC6875 is a monolithic MPU clock driver containing the dynamic and slow memory handshaking logic. A series resonant crystal with a center frequency of four times the desired MPU operating frequency is all that is required for most systems. For systems in which frequency stability is not critical, R-C networks may be used.

This application note describes the MC6875 clock generator/driver and illustrates its use in M6800 microprocessor systems. Included in the design examples are slow memory, dynamic memory and DMA (multiprocessor) examples.

PART DESCRIPTION

The MC6875 clock generator/driver is contained in a 16 pin dual-in-line package. The part uses Schottky devices to provide the high speed needed for fast rise and fall times and reduced propagation delays. Frequency is

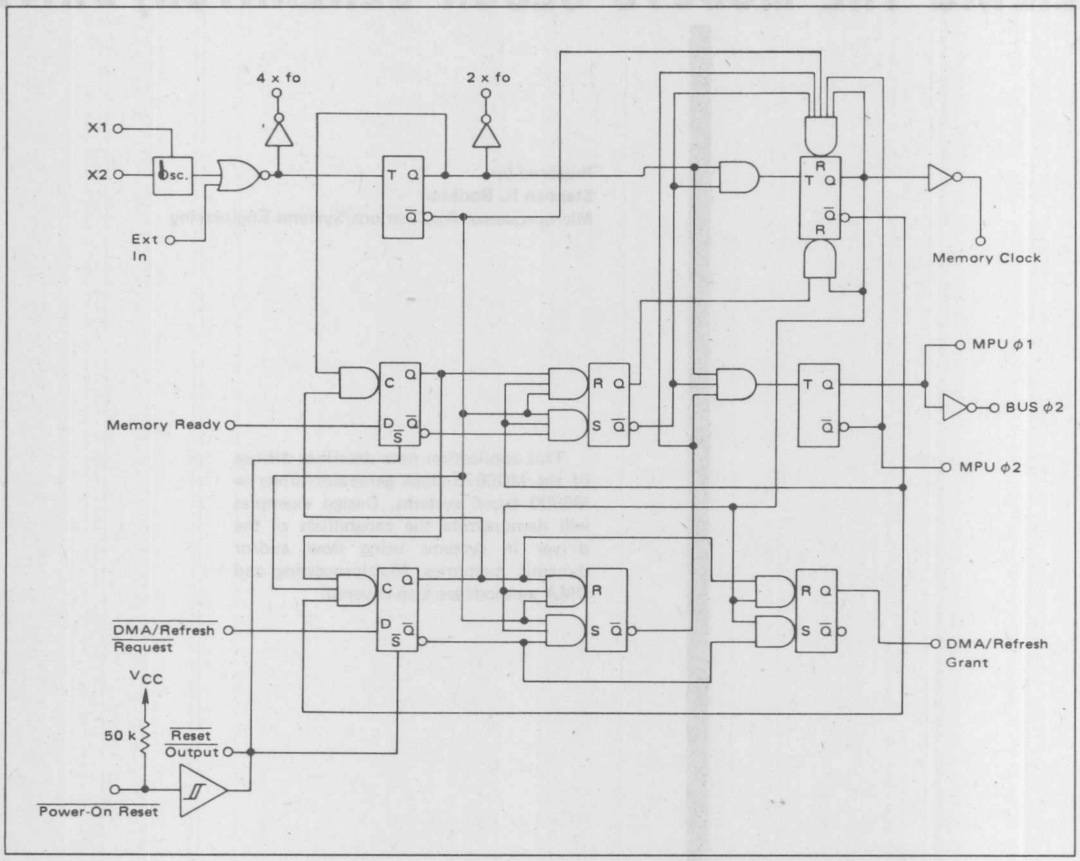


FIGURE 1 – MC6875 Block Diagram

controlled by a series resonant crystal or, alternatively, either an RC or LC network may be used. The resonant frequency of the oscillator is buffered and divided to produce the 4 x fo and 2 x fo outputs. The 2 x fo is then used to produce the Memory Clock, MPU φ1, MPU φ2 and Bus φ2. Memory Ready and $\overline{\text{Refresh Request}}$ inputs are internally sampled on alternate edges of 2 x fo enabling the stretch of either φ1 or φ2. All outputs are capable of driving high capacitance loads typical of unbuffered systems. The MPU φ1 and φ2 signal outputs provide the necessary V_{OH} (V_{CC} - 0.6 V) and V_{OL} (V_{SS} + 0.4 V) capable of driving two MPUs.

The functional block diagram of the internal logic of the MC6875 is illustrated in Figure 1.

SIGNAL DESCRIPTION

4 x fo, 2 x fo

A free running oscillator at four times (two times) the MPU's clock rate useful for a system sync signal.

$\overline{\text{DMA/Ref Req}}$

An asynchronous input used to freeze the MPU clocks in the φ1 high, φ2 low state for dynamic memory refresh or cycle steal DMA (Direct Memory Access).

DMA/Ref Grant

A synchronous output used to synchronize the refresh or DMA operation to the MPU.

Memory Ready

An asynchronous input used to freeze the MPU clocks in the φ1 low, φ2 high state for slow memory interface.

MPU φ1, MPU φ2

Capable of driving the φ1 and φ2 inputs on two MC6800s.

Bus φ2

An output nominally in phase with MPU φ2 having MC8T26 type drive capability which follows MPU φ2.

Memory Clock

An output nominally in phase with MPU φ2 which free runs during a refresh request cycle.

$\overline{\text{Power-On-Reset}}$

A Schmitt trigger input which controls $\overline{\text{Reset}}$. A capacitor to ground is required to set the desired time constant. Internal 50 kΩ resistor to V_{CC}.

$\overline{\text{Reset}}$

An output to the MPU and I/O devices.

X1, X2

Provision to attach a series resonant crystal or RC network.

Ext In

Allows driving by an external TTL signal to synchronize the MPU to an external system.

CAPABILITIES

Slow memory access, dynamic memory refresh and DMA are three areas in which the MC6875 can be used to manipulate and control the MPU timing.

Slow memory access is performed by stretching MPU φ2, Memory Clock and Bus φ2 in the high state while stretching MPU φ1 in the low state. Memory Ready is the control signal used to stretch these signals. Memory Ready is normally high and is active (low) only when addresses are valid to a slow memory or slow peripheral. It should be noted that at higher clock frequencies (above 1 MHz) many of the ROMs, RAMs and peripheral parts with access times sufficient for 1 MHz operation will be classified as Slow Parts needing this interface. The timing relationships are given in Figure 2. Memory Ready is sampled internally on the falling edge of 2 x fo. To stretch φ2, Memory Ready must be in its low state within the required minimum setup time, and held low for the minimum hold time, with respect to the falling edge of 2 x fo corresponding to the leading of φ2 (see MC6875 Data Sheet). Returning Memory Ready to its high state prior to the minimum high setup time referenced to a falling edge of 2 x fo will result in terminating the stretch on the following falling edge of 2 x fo.

A method of generating Memory Ready is illustrated in Figure 3. $\overline{\text{CS}}$ is an active low signal developed from the address decode including VMA used to enable the RAM array. This signal is inverted (CS) and used to hold

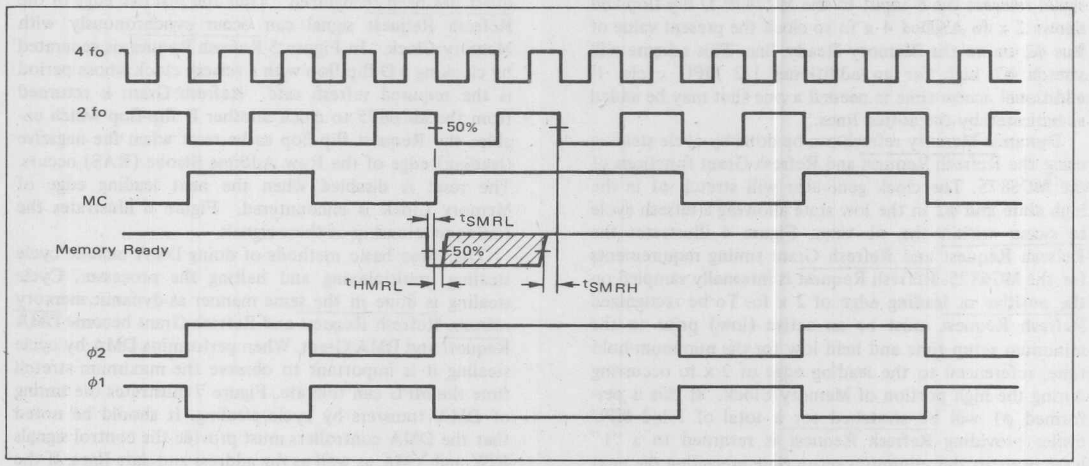


FIGURE 2 — Slow Memory Timing

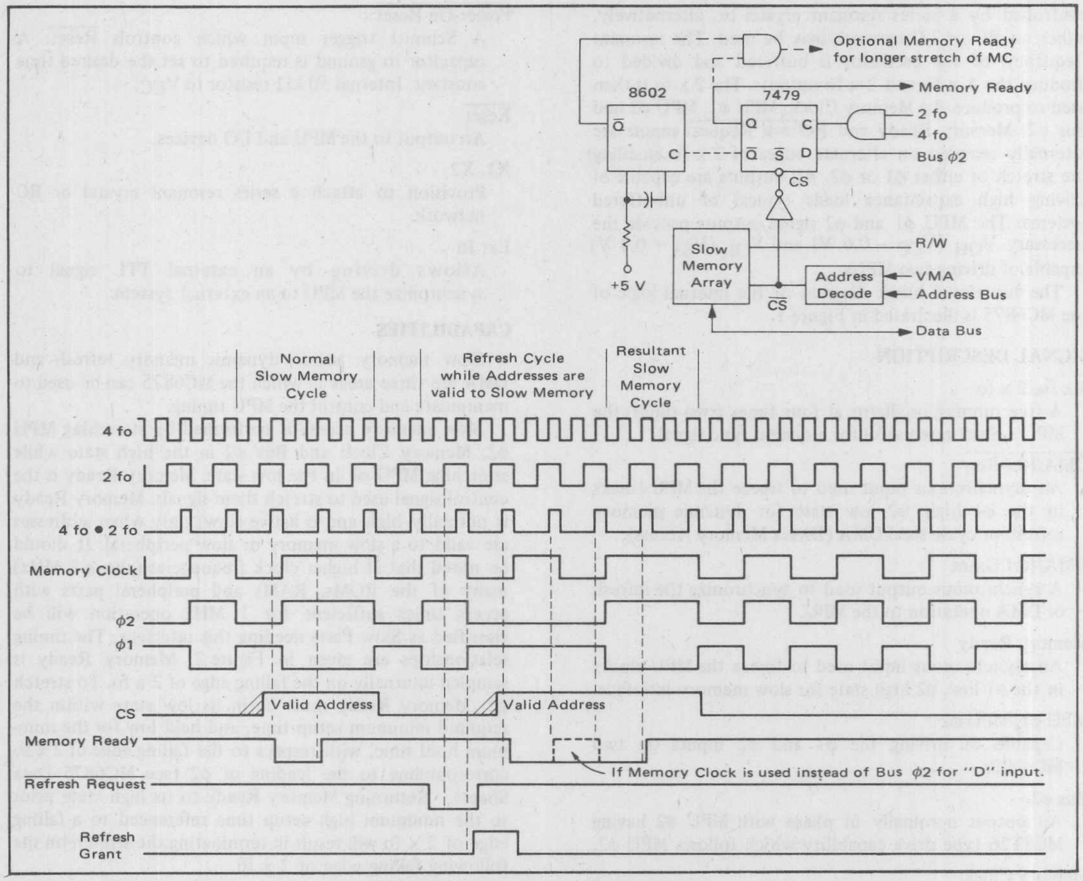


FIGURE 3 — Generation of Memory Ready

Memory Ready at a logic "1" when addresses are not valid to the slow memory. When addresses are valid the CS signal releases the $\overline{S}$ input to the MC7479 D flip-flop and allows 2 x fo ANDed 4 x fo to clock the present value of Bus $\phi 2$ on to the Memory Ready line. This scheme will stretch $\phi 2$ high for an additional 1/2 MPU cycle. If additional access time is needed a one shot may be added as indicated by the dotted lines.

Dynamic Memory refresh can be done by cycle stealing using the $\overline{\text{Refresh Request}}$ and Refresh Grant functions of the MC6875. The clock generator will stretch $\phi 1$ in the high state and $\phi 2$ in the low state allowing a refresh cycle to occur within the $\phi 1$ time. Figure 4 illustrates the Refresh Request and Refresh Grant timing requirements for the MC6875. $\overline{\text{Refresh Request}}$ is internally sampled on the positive or leading edge of 2 x fo. To be recognized Refresh Request must be an active (low) prior to the minimum setup time and held low for the minimum hold time, referenced to the leading edge of 2 x fo occurring during the high portion of Memory Clock. If this is performed $\phi 1$ will be stretched for a total of 1-1/2 MPU cycles providing Refresh Request is returned to a "1" level prior to the minimum setup time preceding the next leading edge of 2 x fo. Since Refresh Request is an asynchronous signal, Refresh Grant is provided by the MC6875 to indicate to the board requesting refresh that the request has been recognized. Thus the inactive edge of the Refresh Request signal can occur synchronously with Memory Clock. In Figure 5 Refresh Request is generated by clocking a D flip-flop with a refresh clock whose period is the required refresh rate. Refresh Grant is returned from the MC6875 to clock another D flip-flop which enables the Request flip-flop to be reset when the negative (leading) edge of the Row Address Strobe (RAS) occurs. The reset is disabled when the next leading edge of Memory Clock is encountered. Figure 6 illustrates the timing relationship of these signals.

The three basic methods of doing DMA include cycle stealing, multiplexing and halting the processor. Cycle stealing is done in the same manner as dynamic memory refresh. Refresh Request and Refresh Grant become $\overline{\text{DMA Request}}$ and DMA Grant. When performing DMA by cycle stealing it is important to observe the maximum stretch time the MPU can tolerate. Figure 7 illustrates the timing of DMA transfers by cycle stealing. It should be noted that the DMA controllers must provide the control signals R/W and VMA as well as the address and data lines. If the DMA Bus interface is wire ORed with the MPU Bus, the

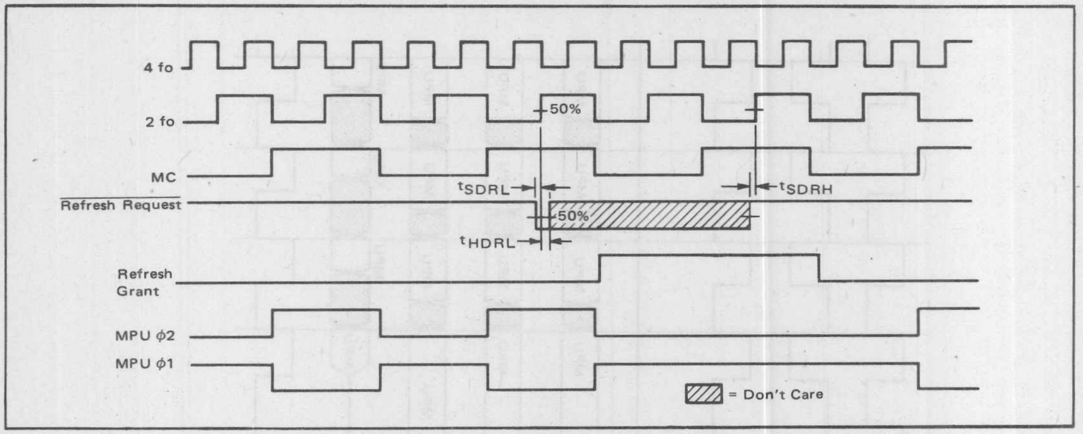


FIGURE 4 – Refresh Request/Refresh Grant Timing

MPU control pin (TSC) can be used to force the MPU Bus drivers to high impedence state. Another alternative would be to use a Bus Switch such as the MC3449.

Multiplexed DMA results in the highest DMA transfer rate since the DMA operation is invisible to the MPU. Multiplexed DMA timing is given in Figure 8. This method requires memory access times fast enough to allow a complete read or write cycle to occur within 1/2 MPU cycle. A sample dual processor design given later will illustrate this technique.

DMA by halting the processor is illustrated in the timing diagram of Figure 9. In this mode the MPU may be halted as long as necessary to perform the DMA as the MPU clock signals are not stretched. This technique is useful when Burst DMA is desired. The DMA controller must provide the halt signal (active low) to the MPU. The MPU will finish the current instruction and respond with a positive transition of BA (Bus available) when the instruction is finished. The DMA controller may then take control of the Bus for transfer.

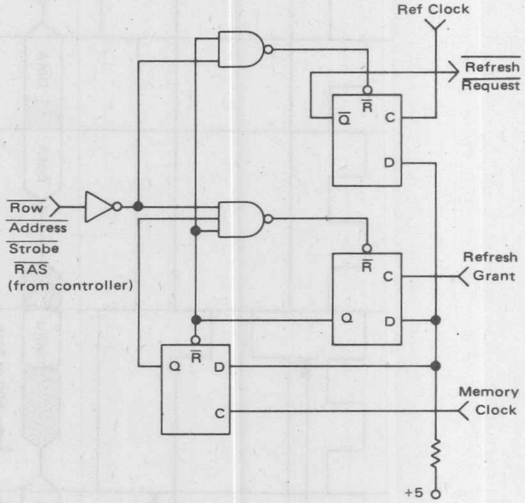


FIGURE 5 – Refresh Request Generation

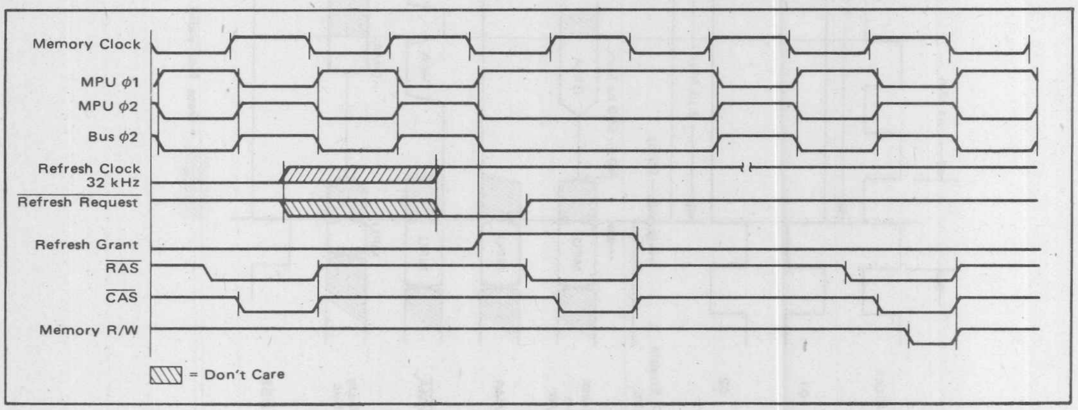


FIGURE 6 – Timing Relationships

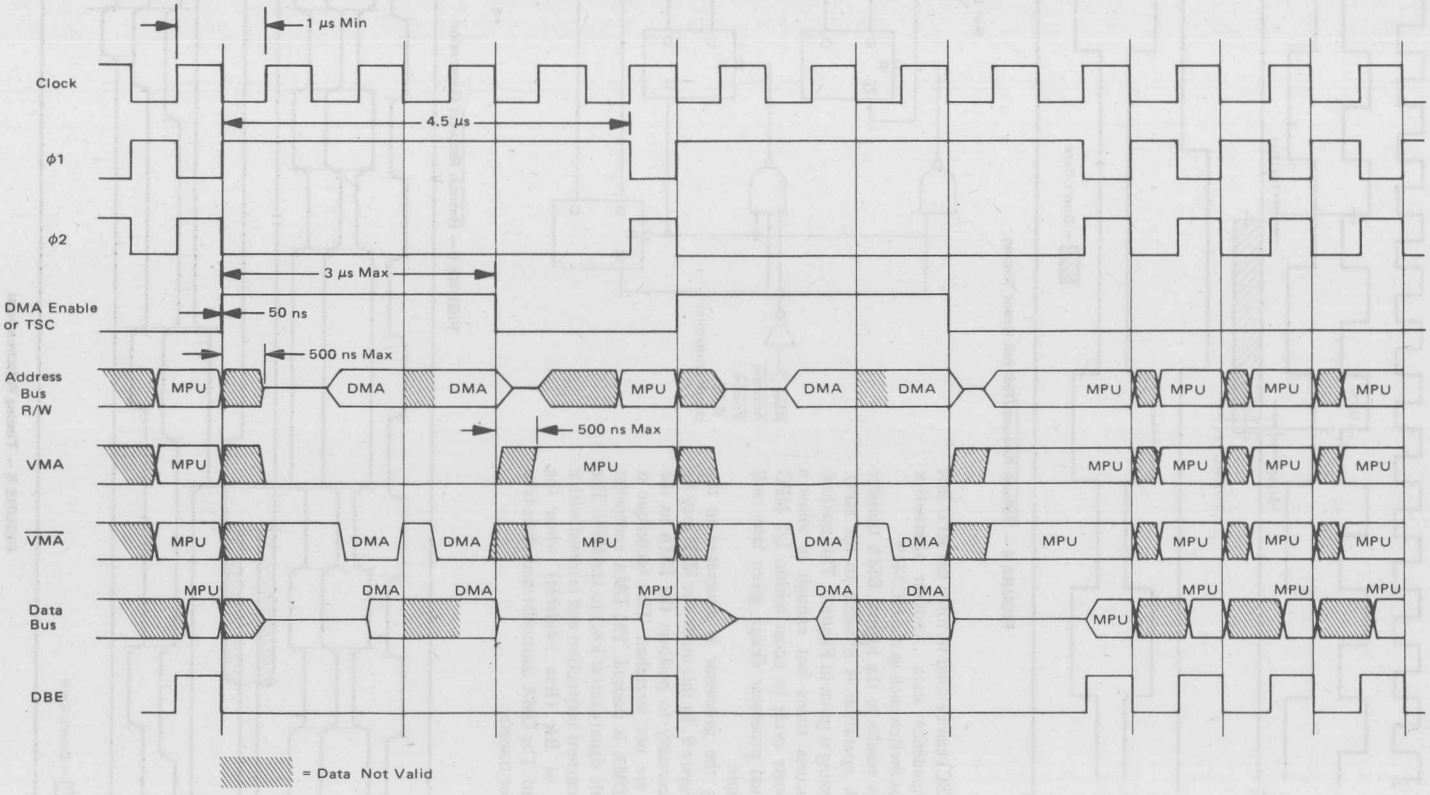


FIGURE 7 – Timing of DMA Transfers by Cycle Stealing

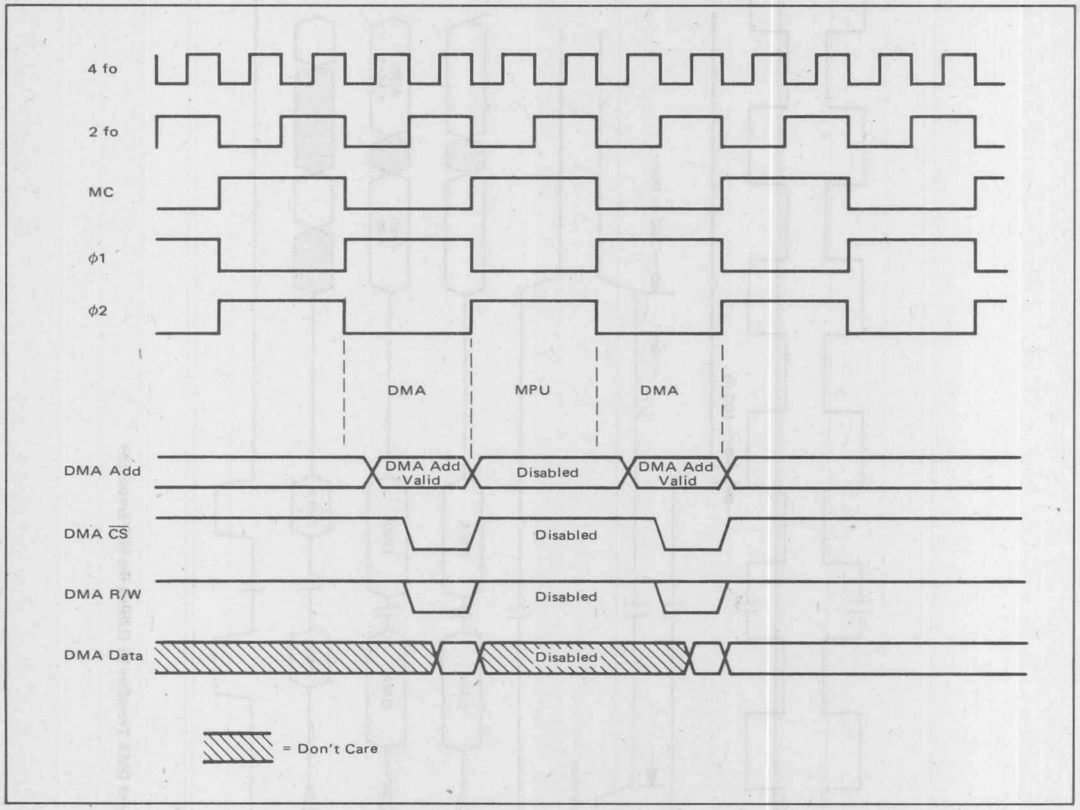


FIGURE 8 – DMA Multiplexed

DESIGN EXAMPLES

Typical Buffered System

In the block diagram of Figure 10 the MC6875 is connected in a typical buffered MPU system. The MC6875 should be located such that the signal path is less than two inches for the MPU ϕ1 and ϕ2. The damping resistors shown at the MPU ϕ1 and ϕ2 inputs should be located as close to the MPU inputs as possible. The value can range from 0 to 30 ohms but the optimum value range is 10 to 20 ohms. These resistors damp the overshoot and ringing typically found in these systems. They also extend the rise and fall times and reduce non-overlap time of the ϕ1 and ϕ2 signals. The block labelled DBE stretch is an optional circuit used with memory peripheral parts requiring longer data hold times. DBE stretch circuits are given in Figures 11a and 11b. These are basic stretch schemes and may be modified to suit the hold-time requirements. DBE may be stretched up to the maximum time allowed by the MPU specification used.

The block labelled Bus Control Logic of Figure 10 is expanded in Figure 12. As shown, this logic controls receiver/driver sections of the MC8T26. The Read Enable (Receiver) is an active low signal enabled only when R/$\overline{W}$ is high and Bus ϕ2 is high (Read Cycle). The write Enable (Driver) is an active high signal enabled only when R/$\overline{W}$ is low, MPU DBE is high and the MPU is not in the halt mode ($\overline{BA}$ high). MPU DBE is used to provide the data hold times required by some memories such as the early 2102.

2048 x 8 Bit Slow Memory Design Using Silicon Gate MOS 2102

The very first step in designing a memory system is to develop a timing diagram showing the relationship of the MPU timing and the required memory timing for both the Read and Write cycles. Once this is done the designer can easily develop the controller which consists of address decode, R/$\overline{W}$ and various strobe signals. Figure 13 illustrates these relationships. The memory $\overline{CE}$ can be presented to the memory array as soon as the MPU addresses are valid. As described earlier, CE should be used to develop the memory ready signal stretching ϕ2 1.5 μs. R/$\overline{W}$ to the memory array must wait at least 200 ns after $\overline{CE}$ for a write cycle. This can be accomplished by incorporating Bus ϕ2 into the Memory R/$\overline{W}$ signal. See schematic of Figure 14.

16K x 8 Dynamic Memory Using MCM6604

Dynamic memory system design has been complicated due to the involved controller. The controller has to

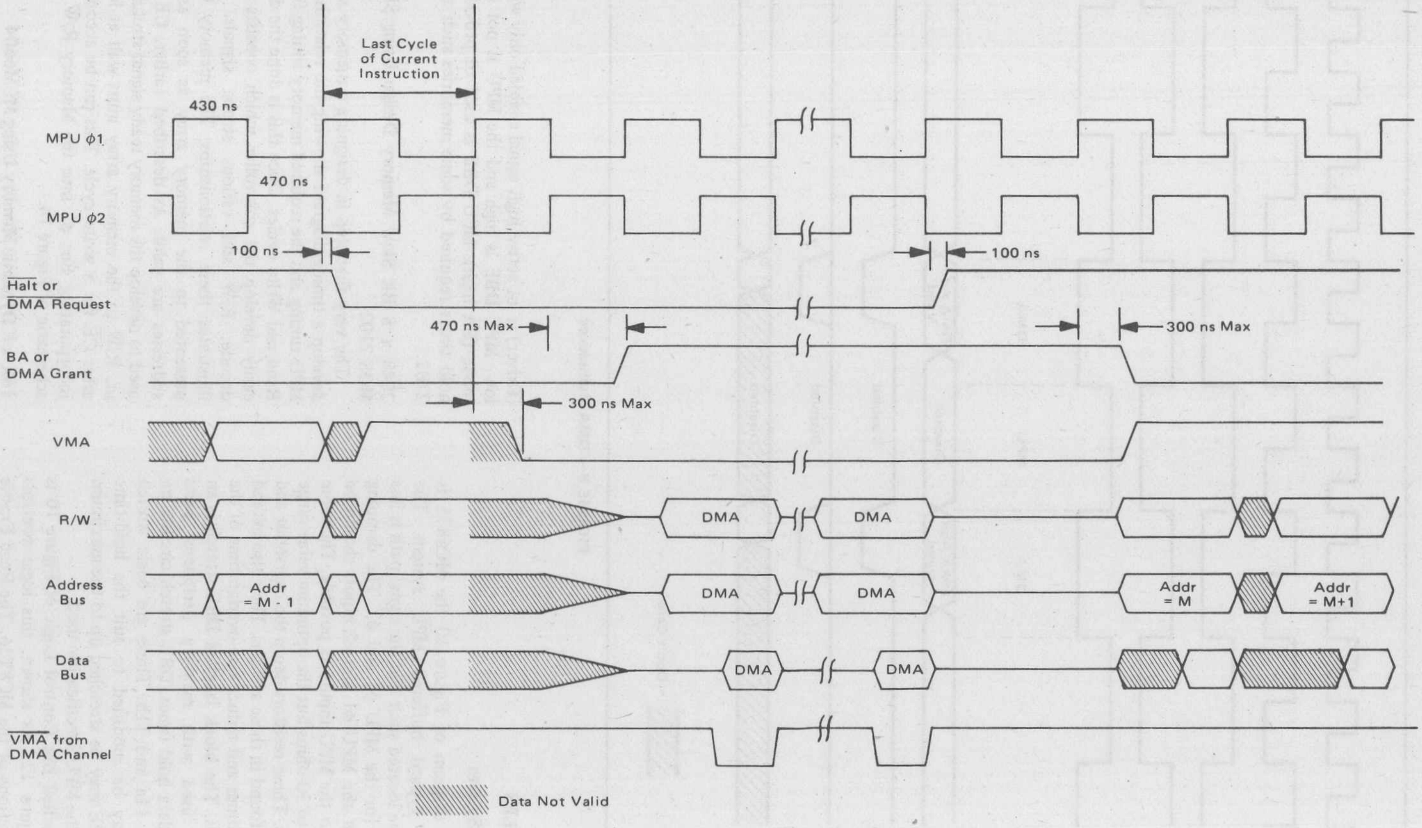


FIGURE 9 – Timing of DMA Transfers by Halting the Microprocessor

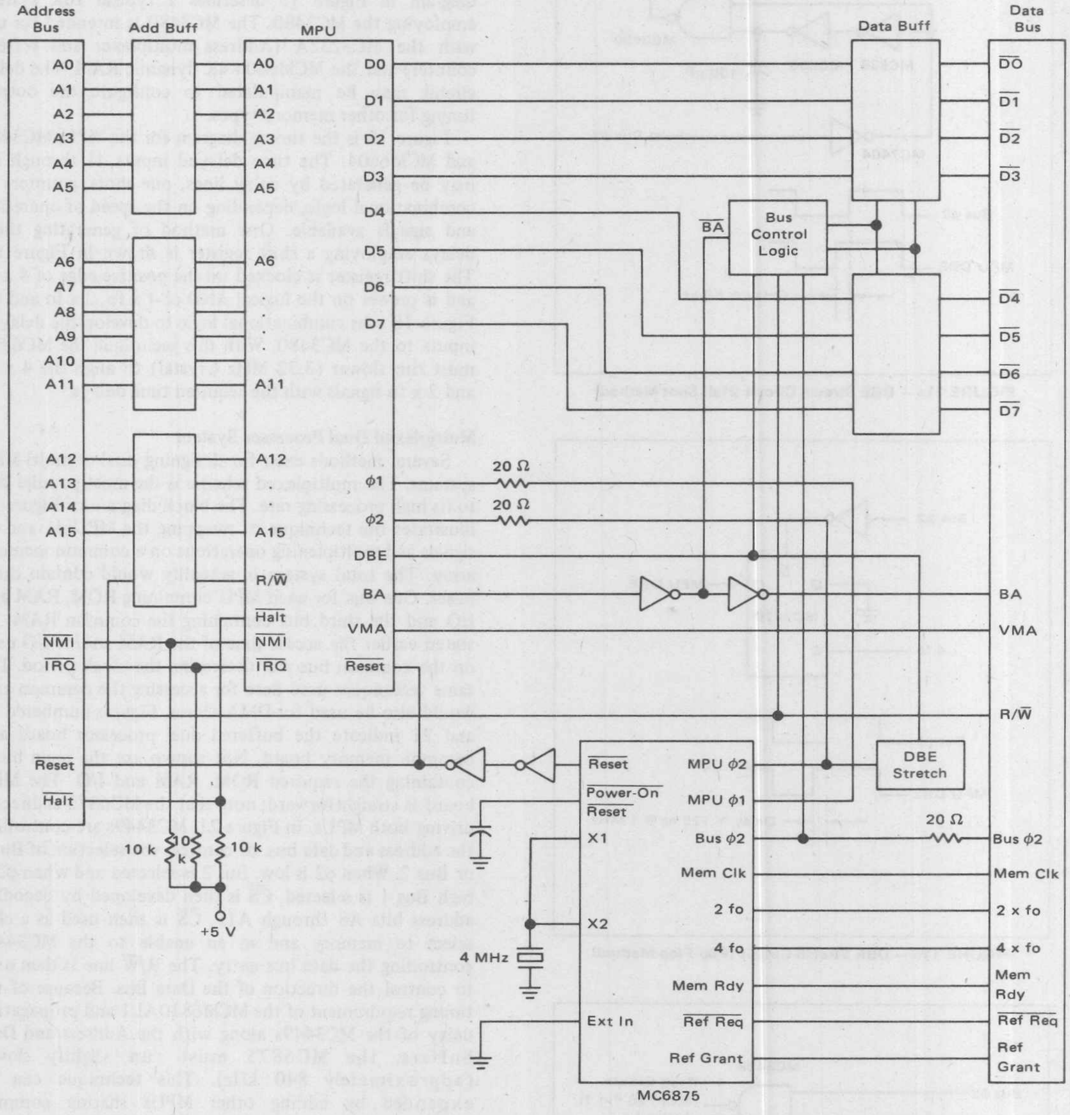


FIGURE 10 – Block Diagram

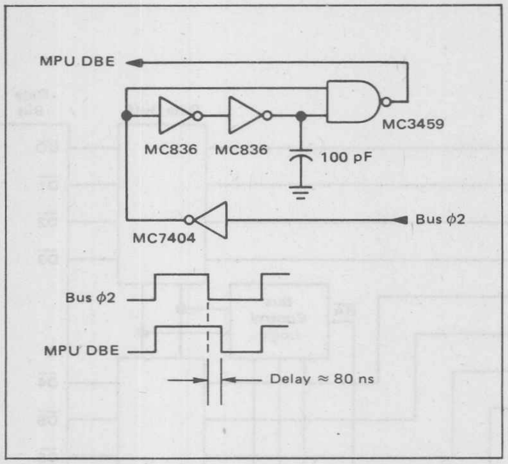


FIGURE 11a – DBE Stretch Circuit (Half Shot Method)

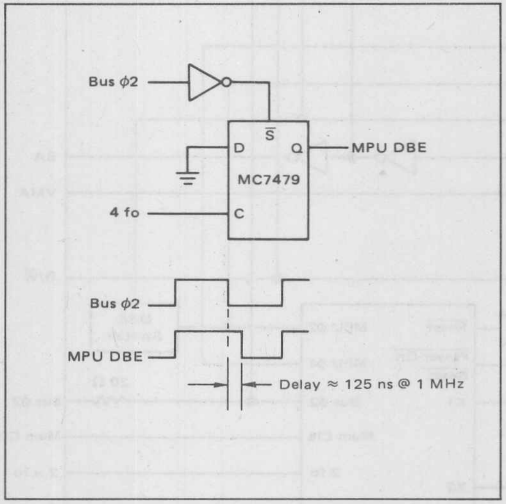


FIGURE 11b – DBE Stretch Circuit (Flip-Flop Method)

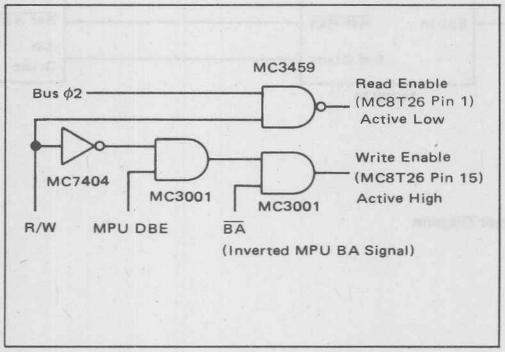


FIGURE 12 – Bus Control Logic

provide refresh, R/W and Row and column address strobes. The MC3480, in conjunction with the MC3232A and MC6875 handles all of this with ease. The block diagram in Figure 15 describes a typical 16K system employing the MC3480. The MC3480 is intended for use with the MC3232A (Address multiplexer and refresh counter) and the MCM6604 4K dynamic RAM. The delay circuit may be manipulated to configure the output timing for other memory types.

Figure 16 is the timing diagram for the MPU, MC3480 and MCM6604. The time delayed inputs, t1 through t5, may be generated by delay lines, one shots, counters or combinational logic, depending on the speed of operation and signals available. One method of generating these delays employing a shift register is shown in Figure 17. The shift register is clocked on the positive edge of 4 x fo and is pre-set on the logical AND of 4 x fo, 2 x fo and fo. Figure 18 uses combinational logic to develop the delayed inputs to the MC3480. With this technique the MC6875 must run slower (3.32 MHz Crystal) to align the 4 x fo and 2 x fo signals with the required time delays.

Multiplexed Dual Processor System

Several methods exist for designing dual or multi-MPU systems. The multiplexed scheme is the most popular due to its high processing rate. The block diagram of Figure 19 illustrates the technique of swapping the MPU ϕ1 and ϕ2 signals and multiplexing operations on a common memory array. The total system in actuality would contain three buses. One bus for each MPU containing ROM, RAM and I/O and the third bus containing the common RAM. As stated earlier the access time of the RAM and/or I/O used on the common bus will determine the clock period. The same techniques used here for accessing the common area would also be used for DMA access. Figures numbered 20 and 21 indicate the buffered dual processor board and common memory board. Not shown are the main buses containing the required ROM, RAM and I/O. The MPU board is straightforward; note that the MC6875 is directly driving both MPUs. In Figure 21, MC3449s are controlling the address and data bus. ϕ2 controls the selection of Bus 1 or Bus 2. When ϕ2 is low, Bus 2 is selected and when ϕ2 is high Bus 1 is selected. $\overline{\text{CS}}$ is then developed by decoding address bits A8 through A15. $\overline{\text{CS}}$ is then used as a chip select to memory and as an enable to the MC3449s controlling the data bus entry. The R/$\overline{\text{W}}$ line is then used to control the direction of the Data Bus. Because of the timing requirement of the MCM6810AL1 and propagation delay of the MC3449s along with the Address and Data buffers, the MC6875 must run slightly slower (approximately 840 kHz). This technique can be expanded by adding other MPUs sharing common memory with the first MPU. Thus MPU1 becomes a master and the other MPUs are slaves sharing common memory only with the master. (Note: All addresses must be unique.)

Dual Processor Using Halt

This technique, as described earlier, is illustrated in the block diagram of Figure 22. For simplicity, it is shown with MPU1 as a master with access to Bus 2 as well as Bus 1. In a user system both MPUs may have access to either bus. In this application access is gained to the second bus

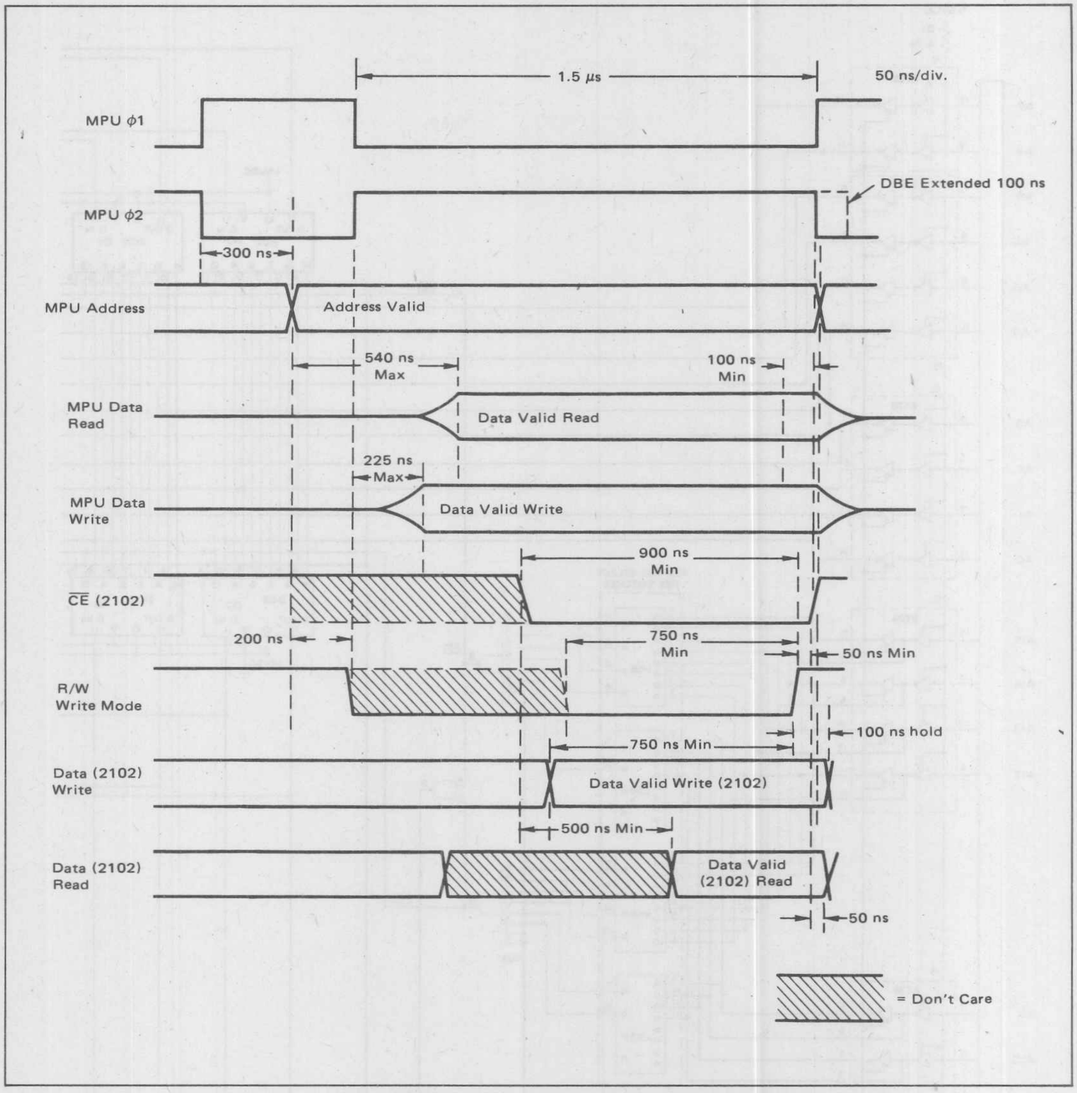


FIGURE 13 – 2102 Timing

by halting MPU2 using a PIA on Bus 1. When MPU2 finishes the current instruction BA will return high switching control of Bus 2 to MPU1 and at the same time indicating to MPU1 that the bus is available. This technique can be used in conjunction with the multiplexed scheme in multiprocessing applications; however, NMOS drivers may have to be used to expand the drive capability of the MC6875.

Design And Layout Considerations

Certain precautions must be taken when designing an MC6875 into an M6800 system. It is recommended that:

1. The MC6875 be located such that the MPU φ1 and φ2 signal paths are within 2" of the MPU.
2. Damping resistors within the ranges of 10 to 30 ohms should be located as close as possible to the MPU φ1 and φ2 input pins of the M6800.
3. Refresh Request and Memory Ready be pulled up when not in use.
4. Crystal, RC or L-C controlling networks be located as close as possible to the corresponding inputs of the MC6875.
5. Ground loops be avoided and high frequency bypass capacitor used directly at the MC6875. (0.1 μF ceramic disk)
6. The External Input be grounded if not being used.

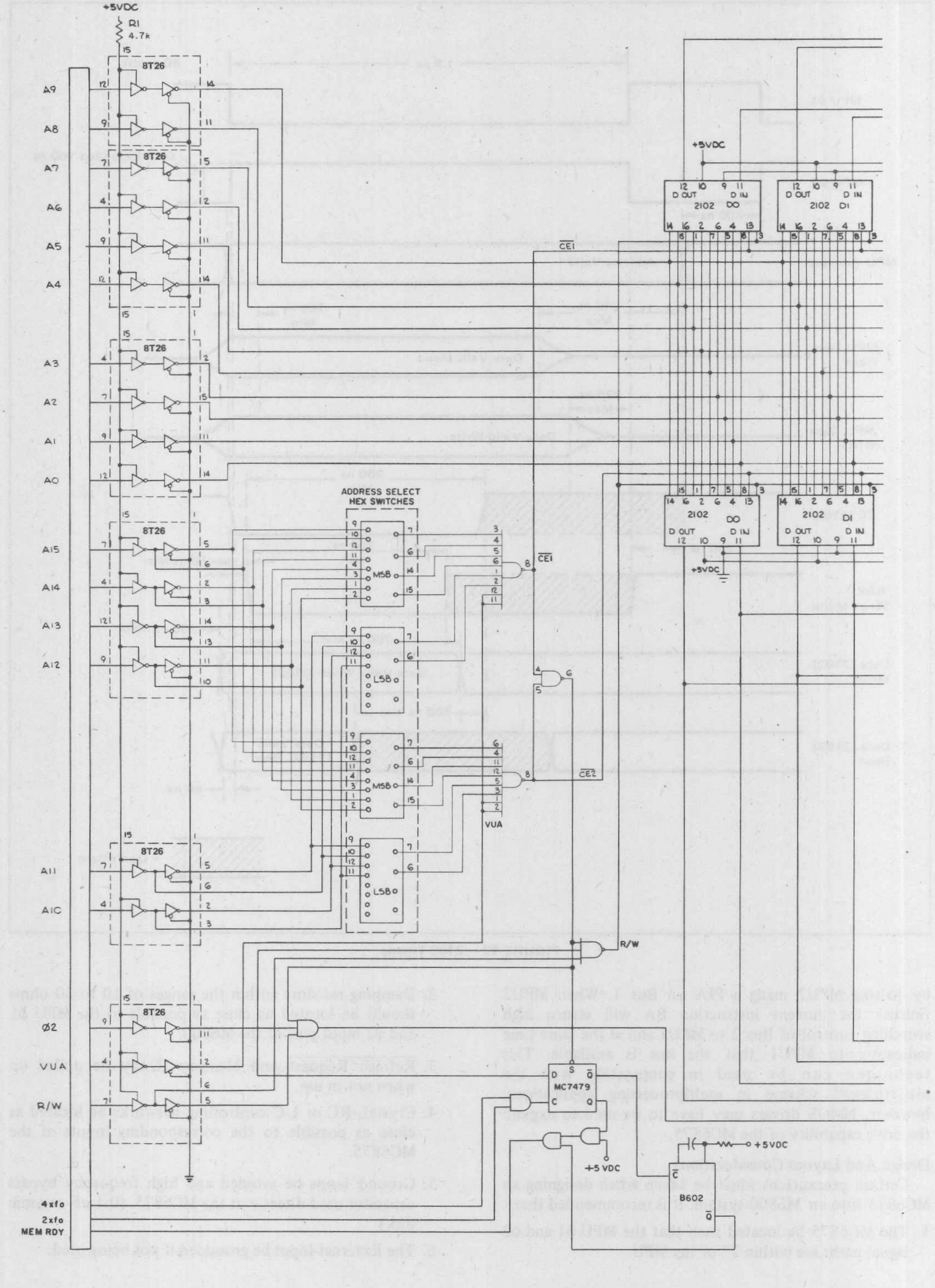
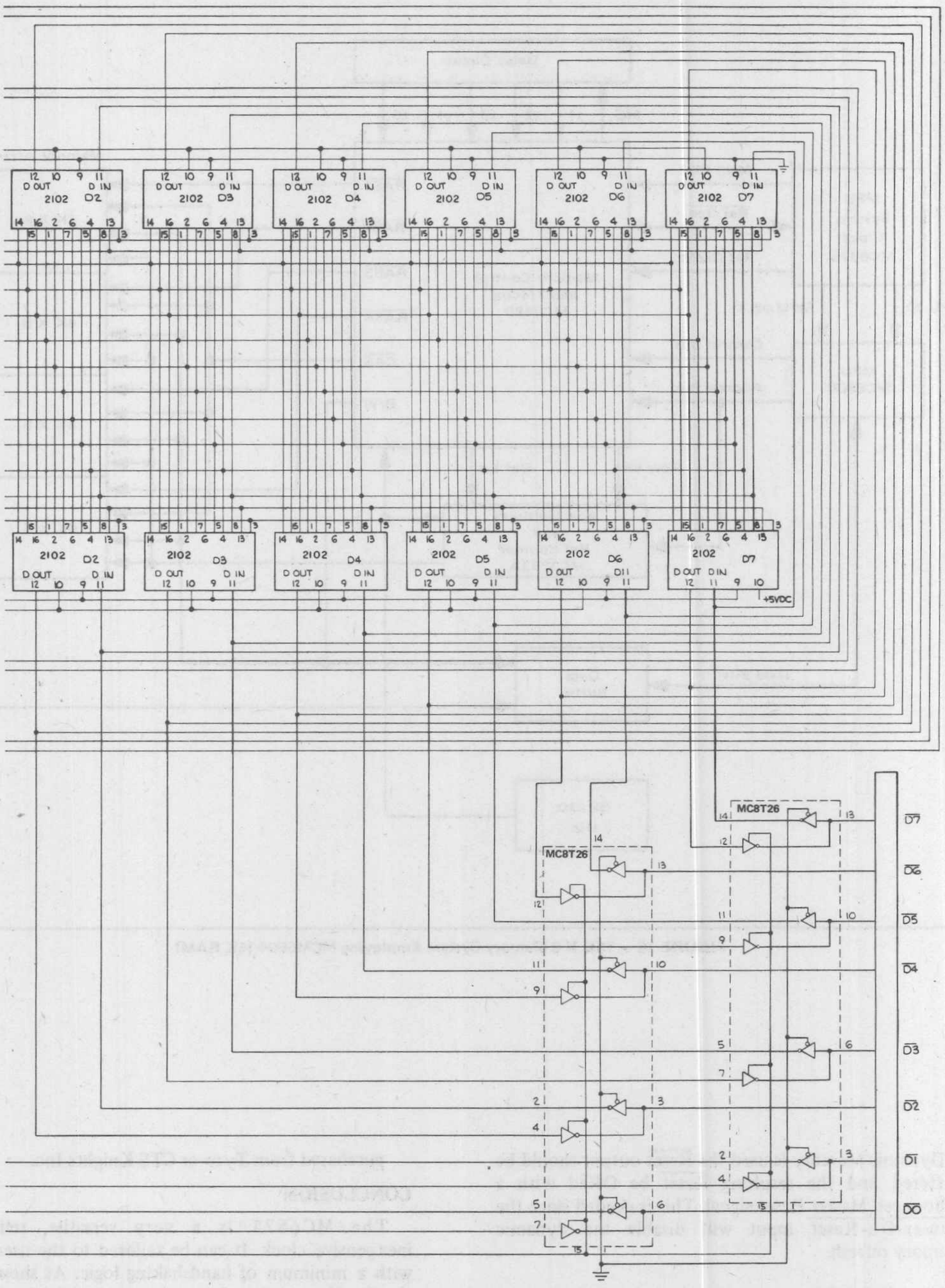


FIGURE 14 — 2K Slow Memory Using 2102's


2102 D2
2102 D3
2102 D4
2102 D5
2102 D6
2102 D7
D OUT
D IN
+5VDC
MC8T26
MC8T26
D7
D6
D5
D4
D3
D2
D1
D0

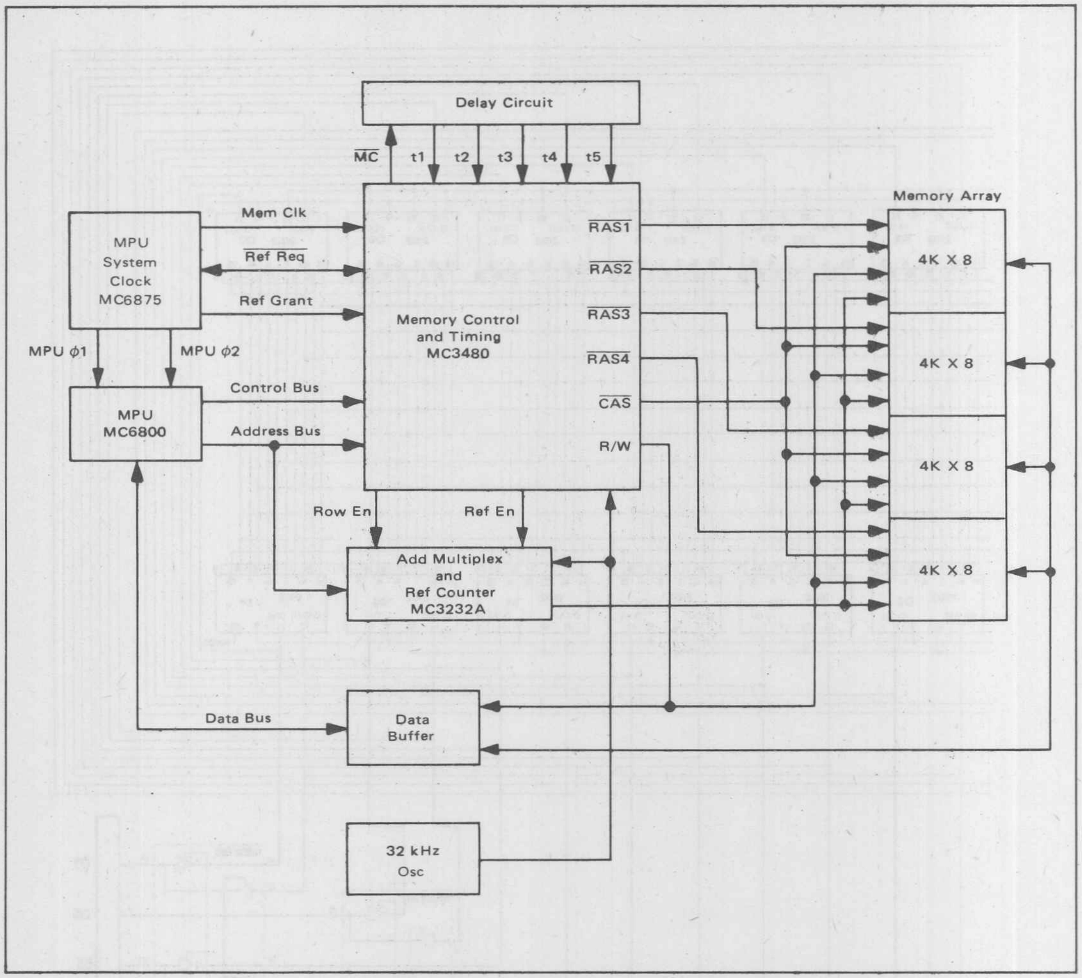


FIGURE 15 – 16K X 8 Memory System Employing MCM6604 (4K RAM)

7. If Dynamic Memory is used the $\overline{\text{Reset}}$ output should be buffered and the resulting $\overline{\text{Reset}}$ be ORed with a debounced Master Reset signal. This is needed since the Power-On-Reset input will disable the dynamic memory refresh.

8. TTL and NMOS loads should not exceed the maximum capability of the MC6875.

9. Crystals be selected with equivalent series resistance of 35 to 60 ohms and that can tolerate a circuit load capacitance of 12.5 to 19 pF. These crystals may be purchased from Tyco or CTS Knights Inc.

CONCLUSION

The MC6875 is a very versatile, reliable and inexpensive clock. It can be tailored to the users' system with a minimum of handshaking logic. As shown earlier, the 2 x fo and 4 x fo outputs are useful in developing various control signals needed with certain memory and I/O parts. The high drive capability make it useful in buffered, unbuffered and dual MPU systems. The clock stretching capability make it useful in Dynamic Memory, slow memory and DMA applications.

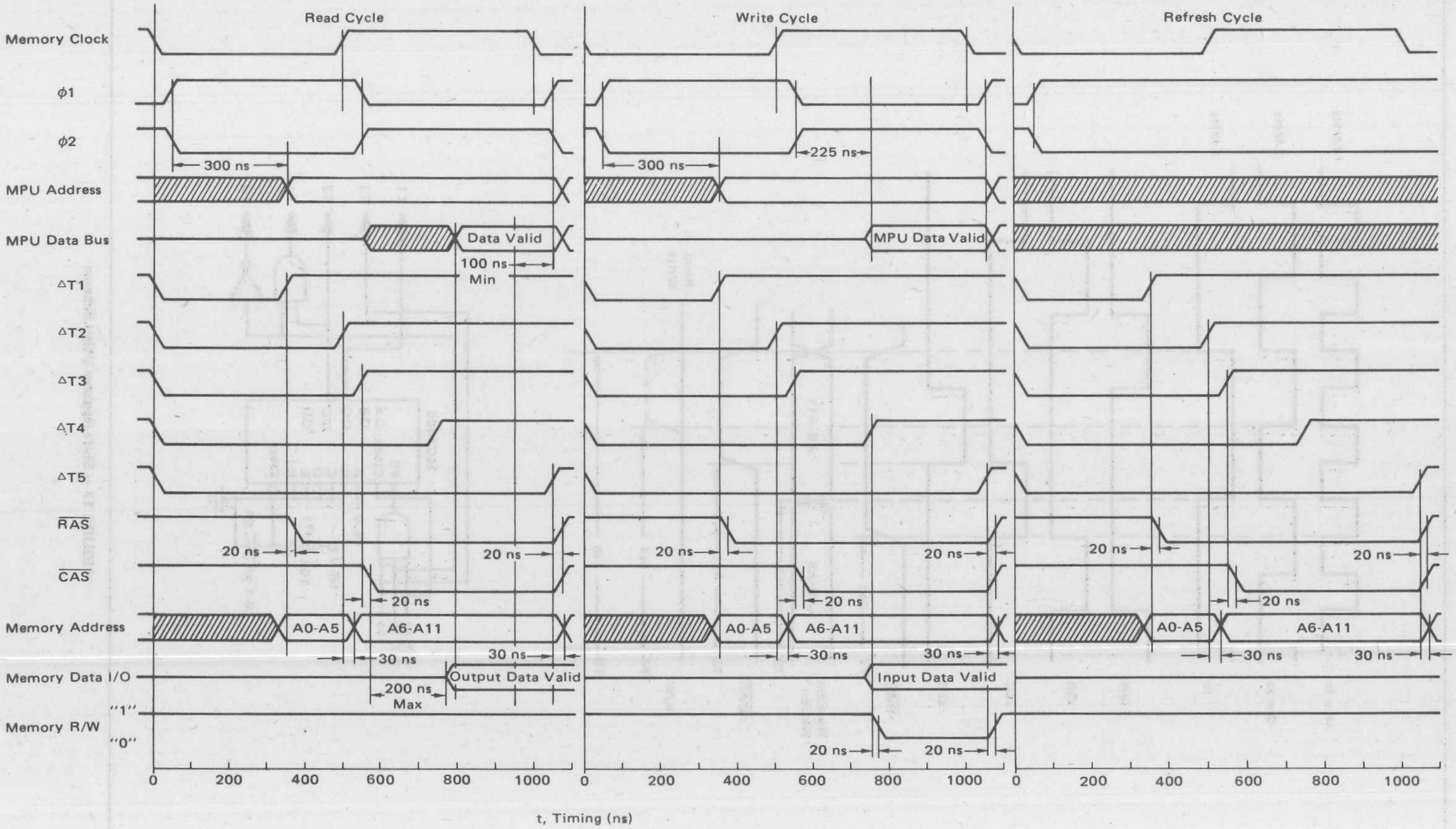

Read Cycle
Write Cycle
Refresh Cycle
Memory Clock
φ1
φ2
MPU Address
MPU Data Bus
ΔT1
ΔT2
ΔT3
ΔT4
ΔT5
$\overline{RAS}$
$\overline{CAS}$
Memory Address
Memory Data I/O
Memory R/W
"1"
"0"
300 ns
225 ns
Data Valid
MPU Data Valid
100 ns
Min
20 ns
30 ns
A0-A5
A6-A11
Output Data Valid
Input Data Valid
200 ns
Max
0
200
400
600
800
1000
t, Timing (ns)

FIGURE 16

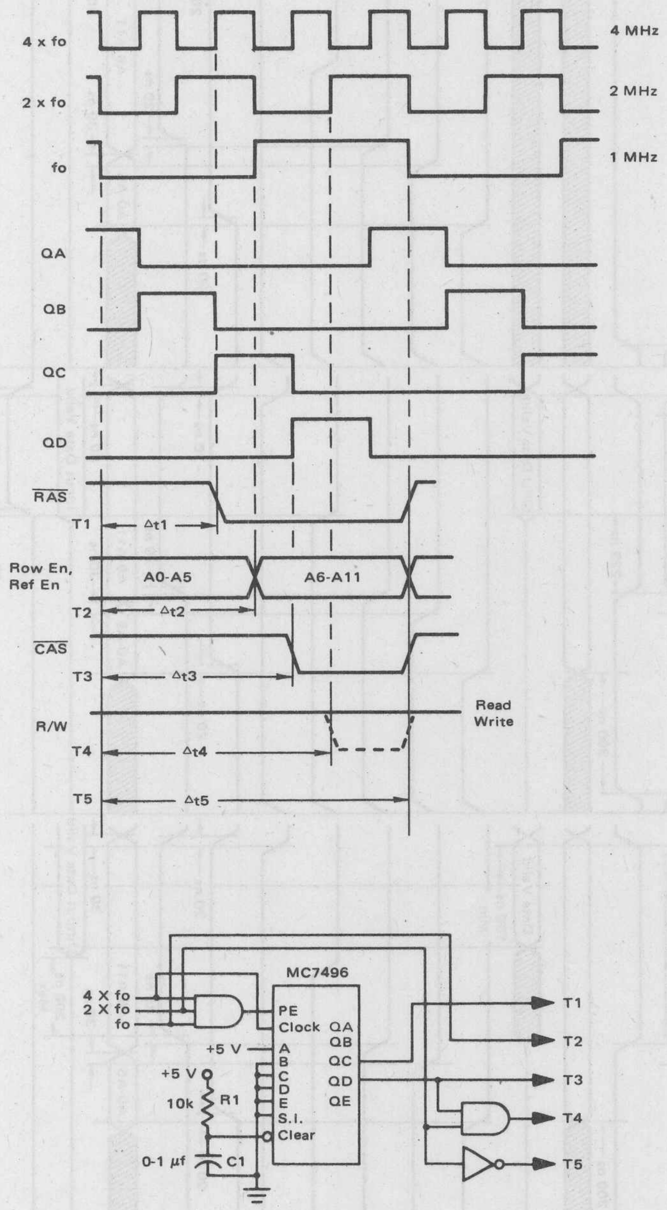


FIGURE 17 — Shift Register Delay Scheme

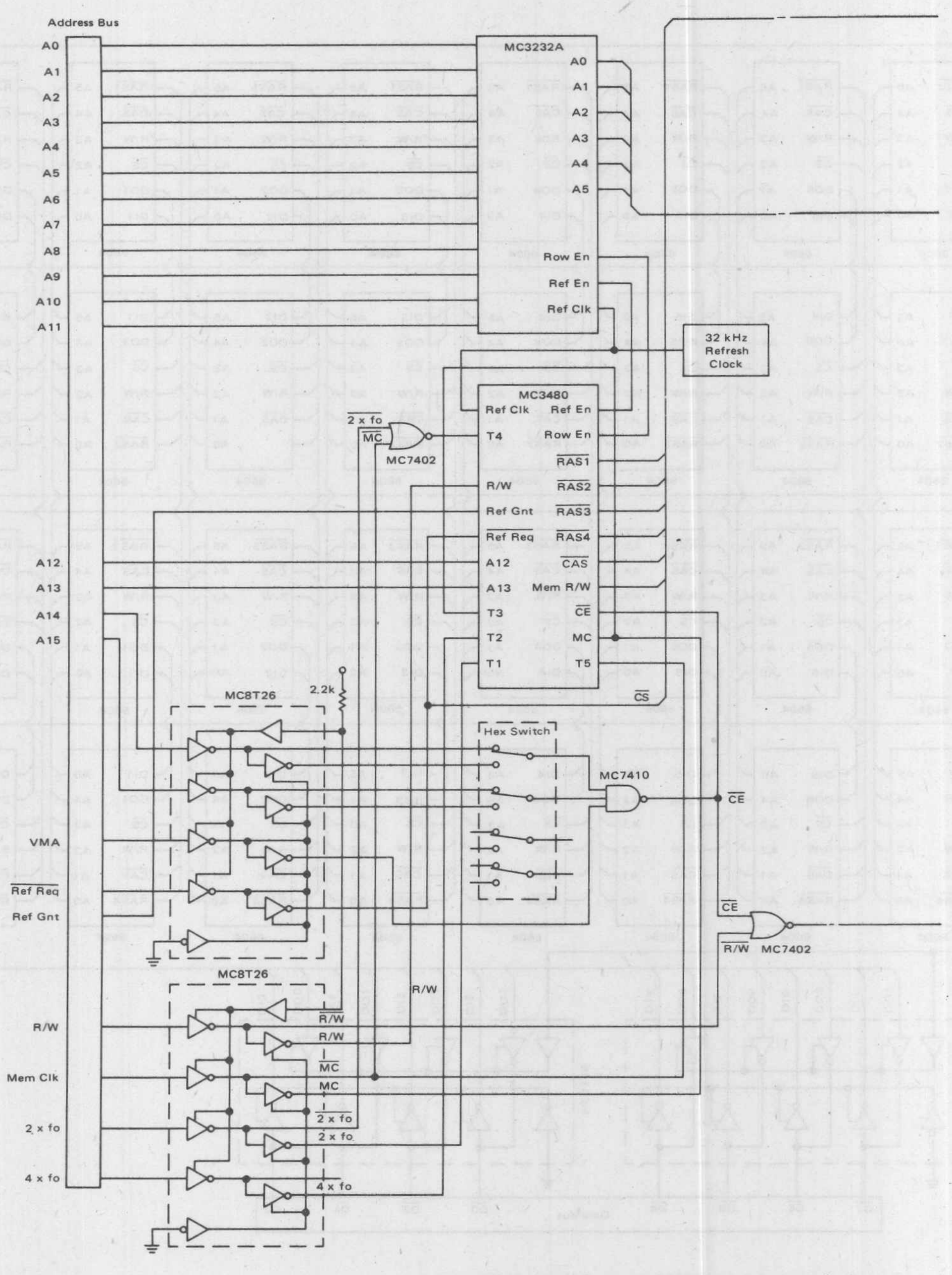
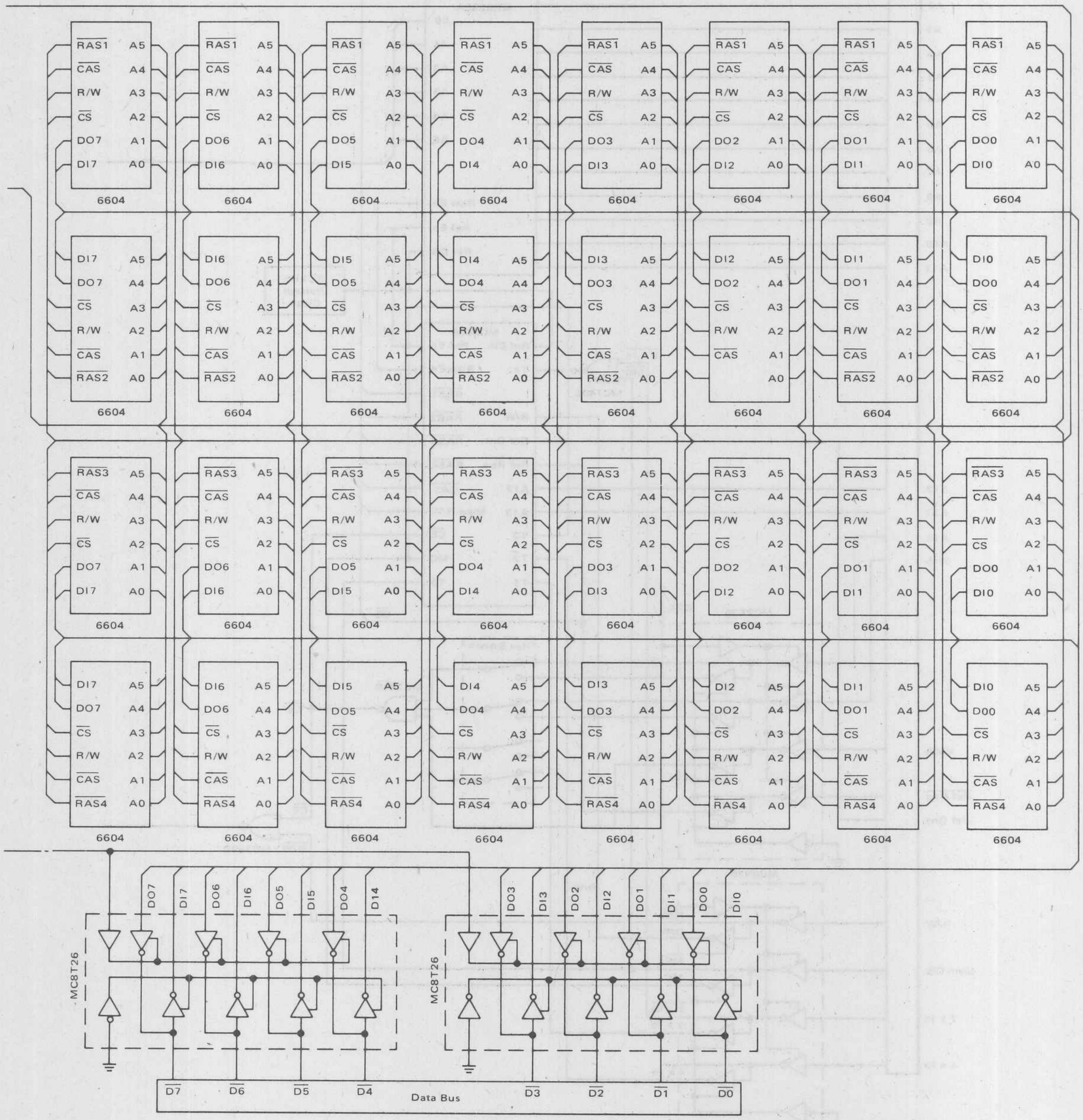


FIGURE 18 – 16K Dynamic RAM Using 4 fo and 2 fo Timing


RAS1
RAS2
RAS3
RAS4
CAS
R/W
CS
DO7
DI7
DO6
DI6
DO5
DI5
DO4
DI4
DO3
DI3
DO2
DI2
DO1
DI1
DO0
DI0
A5
A4
A3
A2
A1
A0
6604
MC8T26
D7
D6
D5
D4
D3
D2
D1
D0
Data Bus

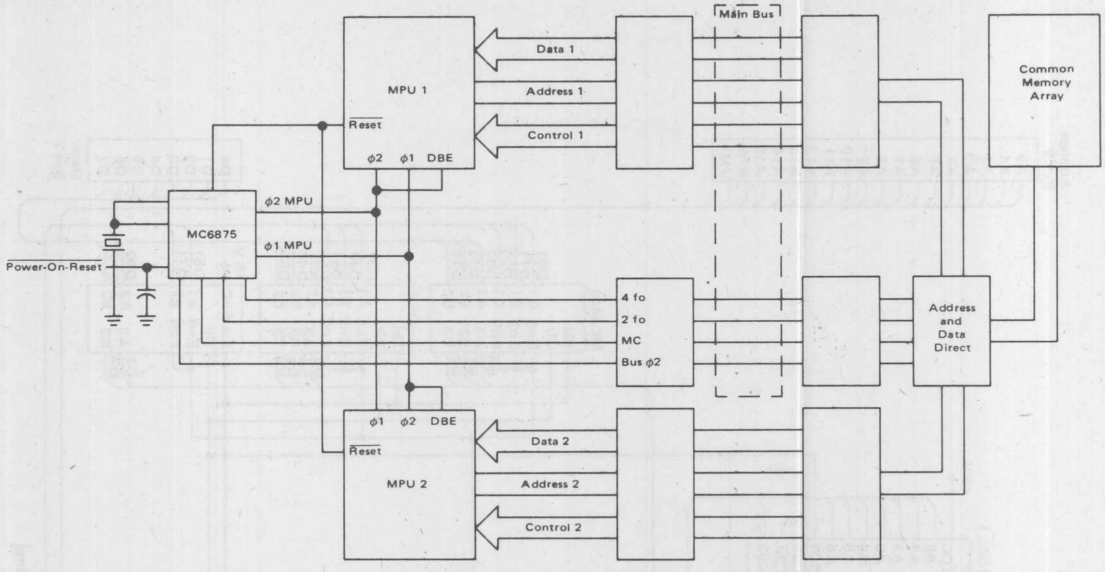


FIGURE 19 – Dual Processor Block Diagram

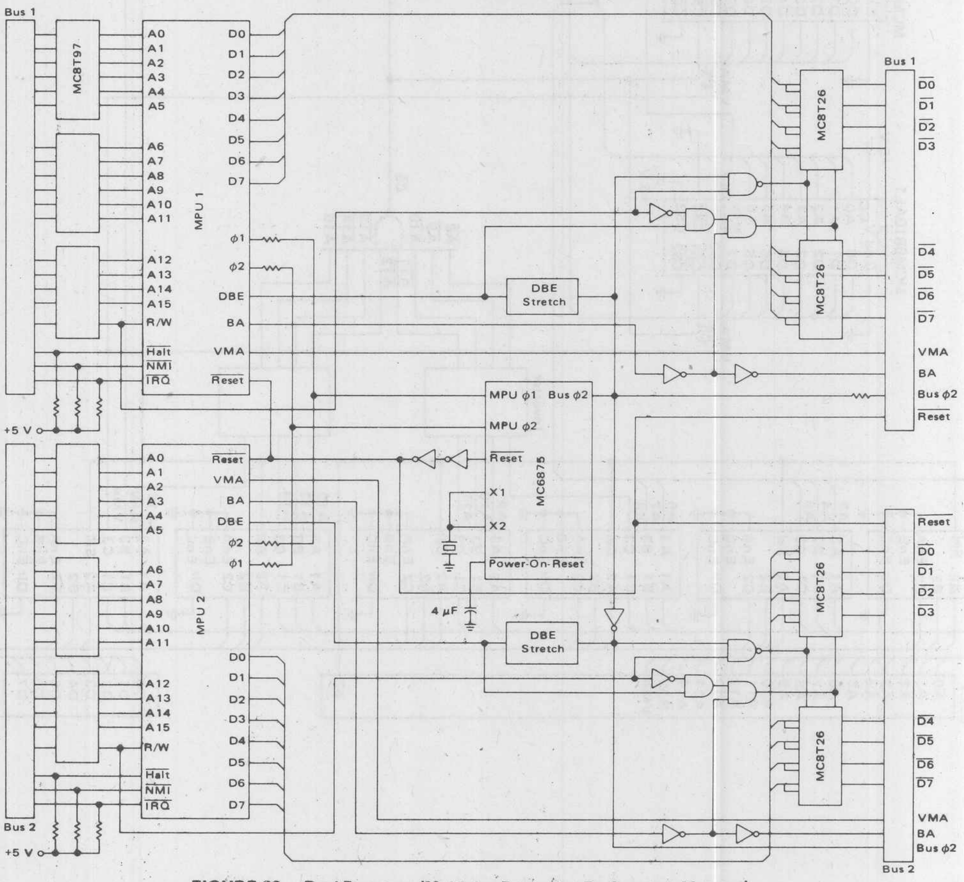


FIGURE 20 – Dual Processor (Multiplex Processing To Common Memory)

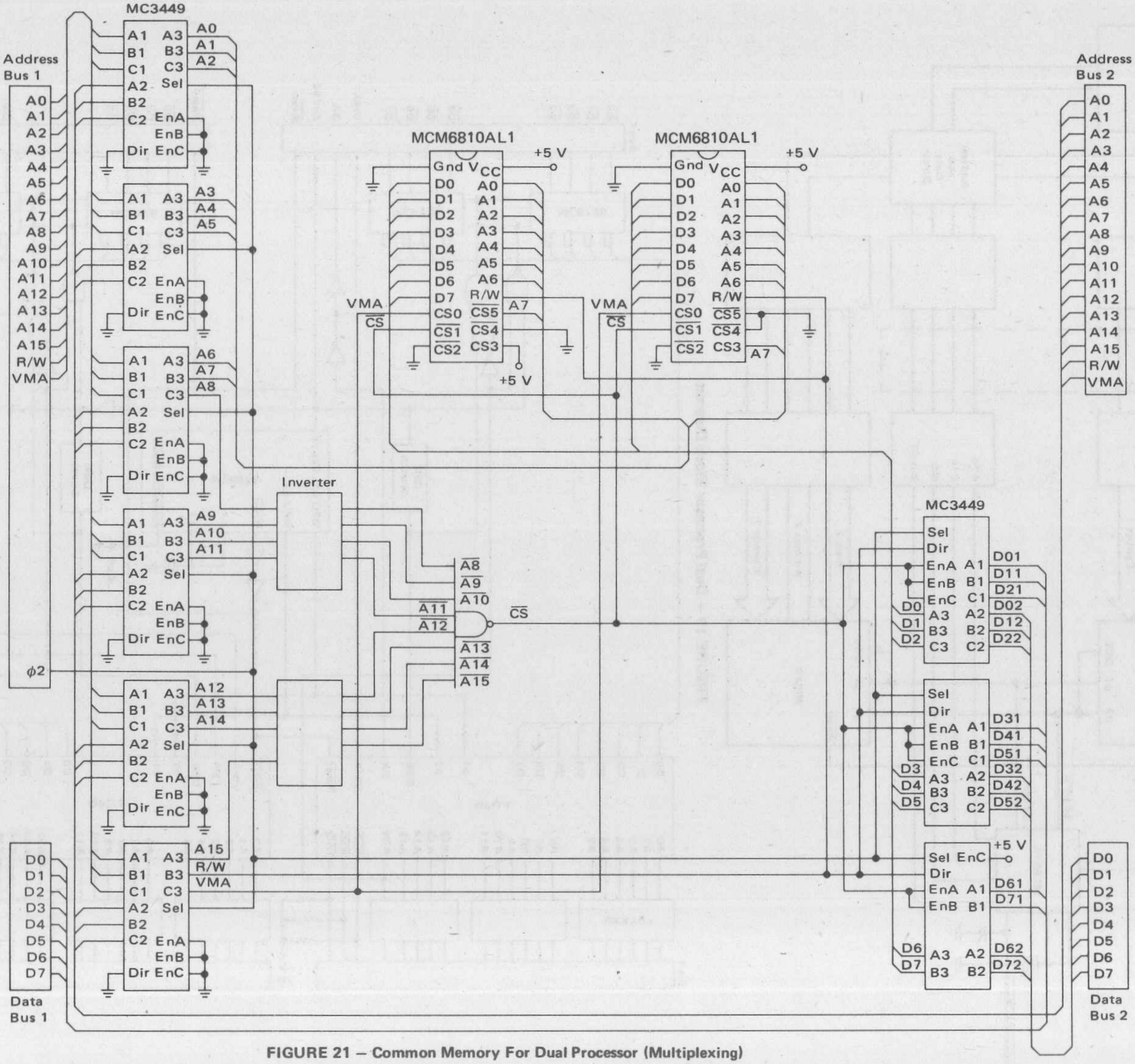


FIGURE 21 — Common Memory For Dual Processor (Multiplexing)

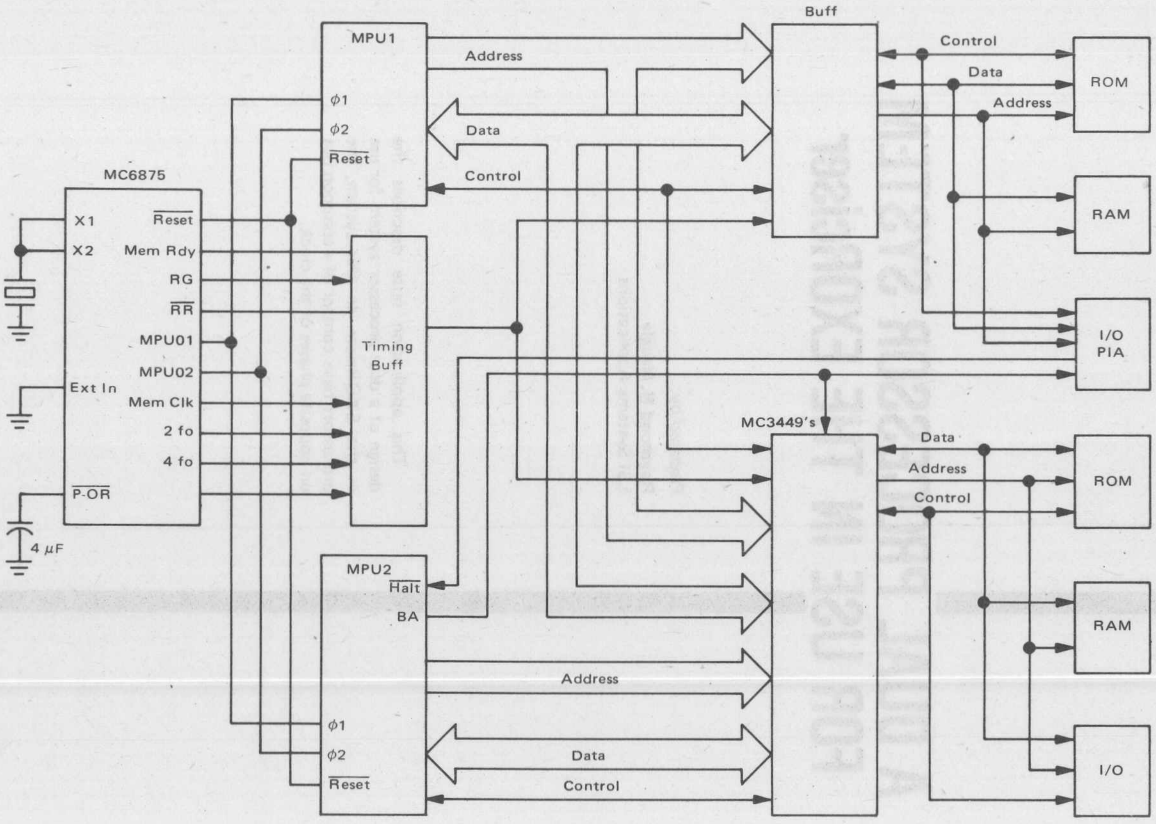
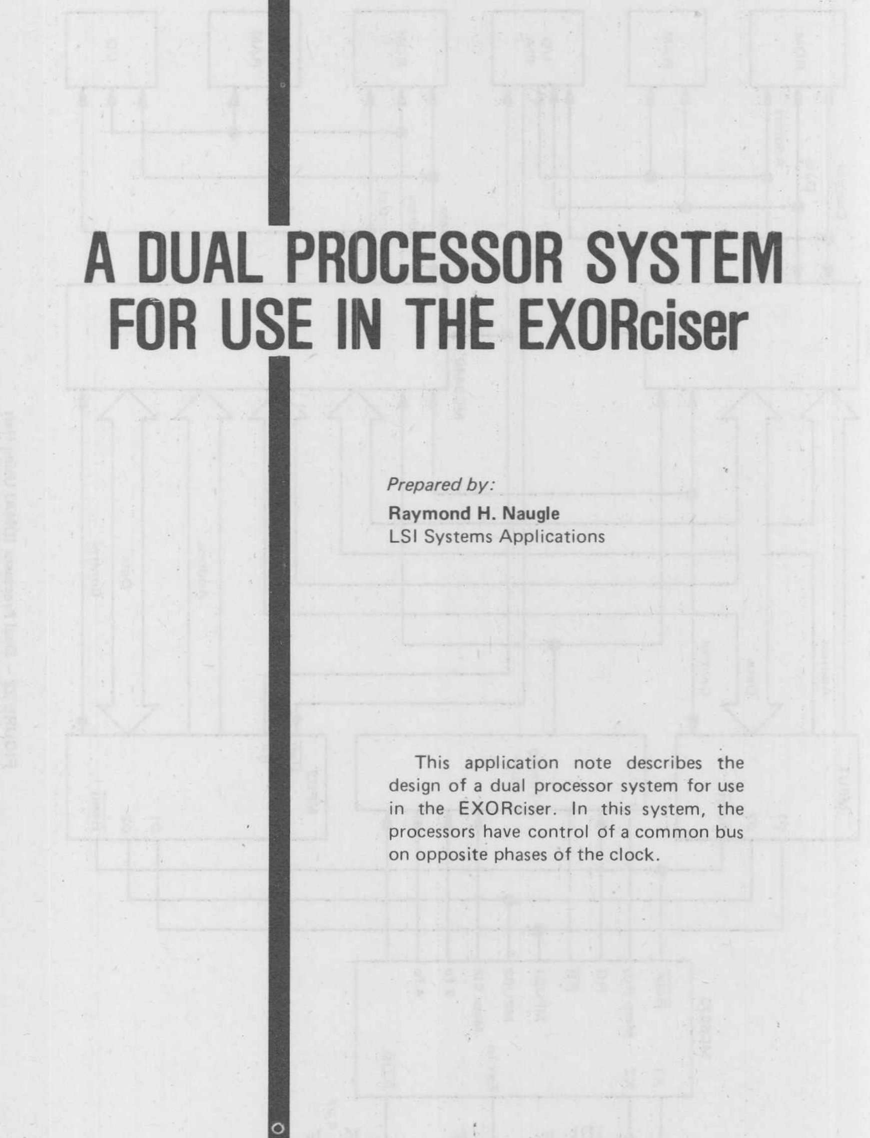


FIGURE 22 – Dual Processor (DMA) Using Halt

AN-777



A DUAL PROCESSOR SYSTEM FOR USE IN THE EXORciser

Prepared by:

Raymond H. Naugle
LSI Systems Applications

This application note describes the design of a dual processor system for use in the EXORciser. In this system, the processors have control of a common bus on opposite phases of the clock.

A DUAL PROCESSOR SYSTEM FOR USE IN THE EXORciser

INTRODUCTION

This paper describes the design of a dual processor system with a minimum of additional circuitry over a single processor system. The system is EXORciser compatible, operating under a common program, EXbug.

DUAL PROCESSING

You can increase your system's throughput by using two processors instead of one. The added speed is accomplished by doubling the available processing power which includes increased interrupt handling capability for the same system. The system may also be designed so the cost will be only a little more than a single processor system.

Many configurations are possible when connecting two processors together. Figures 1 and 2 show two implementations each using two processor subsystems connected by some logic interface. Although these figures show two completely separate systems, they may share some common components such as power supplies, clock circuitry, etc. Also, these two implementations may be extended for use in a system with more than two processors.

In Figure 1, PIAs are used for communication between the two systems. The PIA contains two 8-bit data ports with additional internal logic to take care of the "handshaking" between the two processors when the data is transferred. The PIA can be programmed so that one is an input port and the other is an output port. In this scheme, one processor could handle the peripheral input and some data formatting, then send the data to the second processor for the data maipulation. After the task is complete, the second processor sends formatted data back to the first processor for unformatting and output to a peripheral. Thus, while the first processor is receiving/transmitting data, the second processor could be doing the "number crunching".

Figure 2 shows a FIFO (First-In First-Out Register) being used for transferring data between the processors. (Shown in the diagram is data flow in one direction, but the same logic could be repeated for data transfer in the other direction.) In this configuration, the first processor may stack data for use by the second processor and the second processor may read data as needed, thus saving processing time from answering interrupts for data being transferred in. Here, the processors are allowed to operate more efficiently.

Another approach to the dual-processor implementation is shown in Figure 3. Here both processors utilize the same bus, operating on opposite phases of the clock. This system eliminates the need for a second bus structure and most of the hardware or software for communication between processors. Since the processor transfers data only during the $\phi 2$ portion of its clock cycle, it needs to have access to the bus only during this time and not during $\phi 1$. Therefore, when one processor is in the $\phi 1$ portion of its clock cycle, the other processor is in its $\phi 2$ portion and has control of the bus. This doubles the effective speed of the data transfer rate. (Since the bus cycle time has been cut in half, the response time of the bus parts should be checked to insure proper timing.)

Communication between the two processors may now be done in common RAM. This feature saves adding a couple of PIAs or FIFOs plus extra interface logic.

Some other features of this system include eliminating some ROM where common routines may be utilized. If both processors execute the same program, the cost of the total system's ROM will be cut in half.

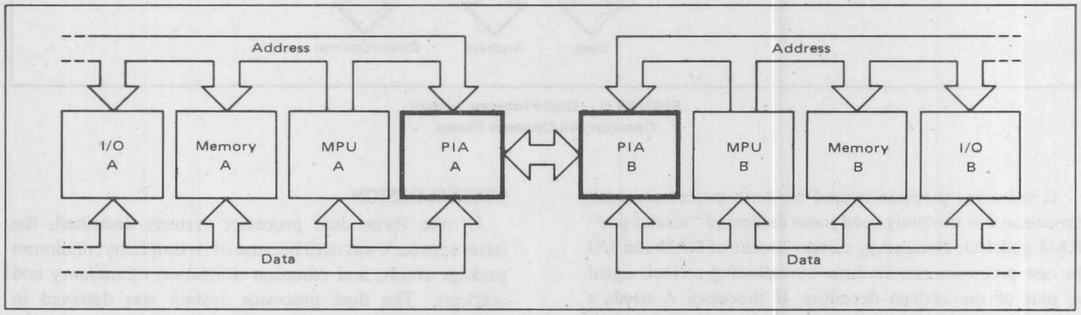


FIGURE 1 — Dual Processor System Using PIAs

EXORciser and EXbug are trademarks of Motorola Inc.

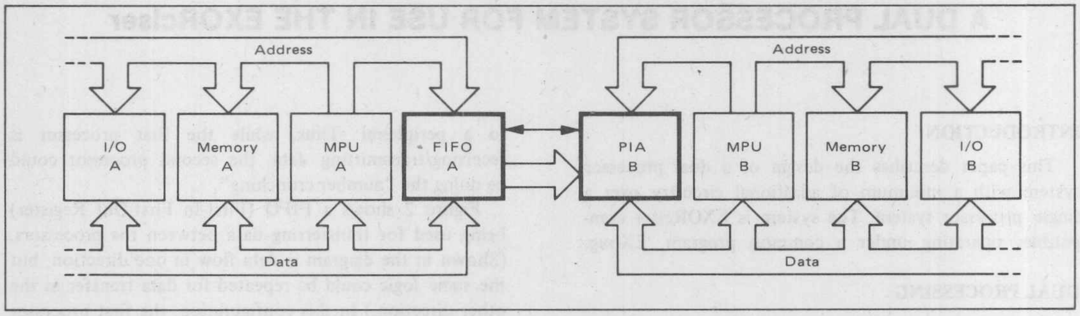


FIGURE 2 — Dual Processor System Using a FIFO

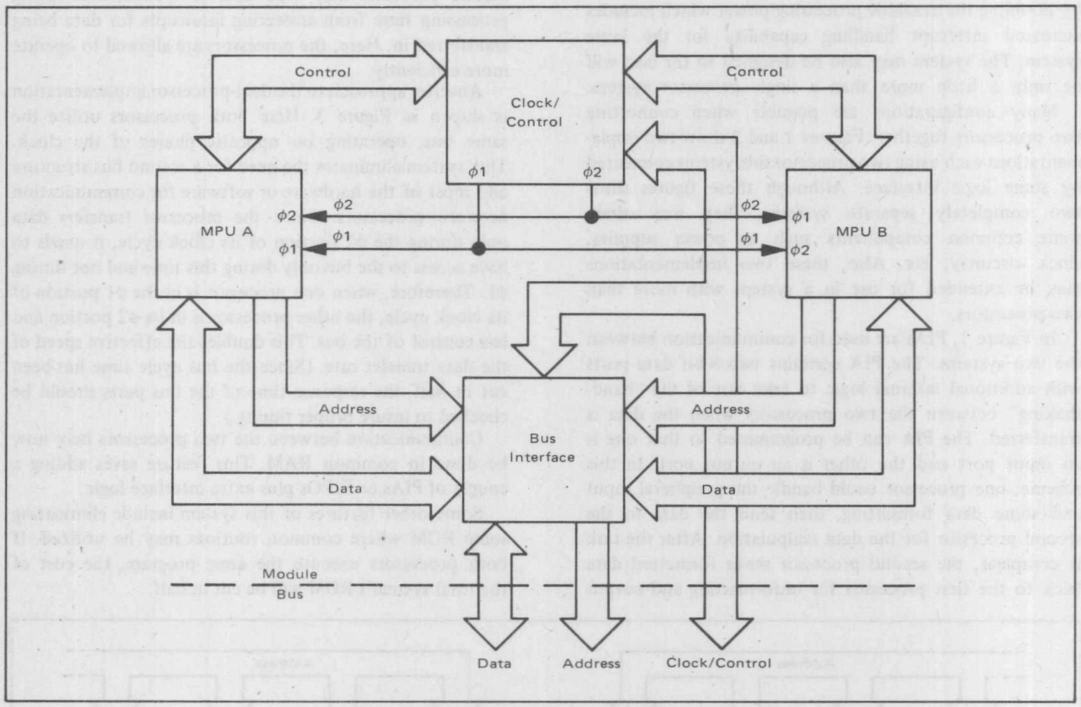


FIGURE 3 - Dual Processor System Operating on Opposite Phases

If the same program is used for both processors, each processor will probably need some dedicated "scratchpad" RAM and I/O. Dedicating certain blocks of RAM and I/O to one processor can be done by including a clock signal as part of the address decoding. If processor A needs a certain block of memory dedicated to it, the clock $\phi 2$ signal should be included in the address decoding for that block. Then when processor B tries to access that block during its $\phi 2$ portion of the clock, the clock's $\phi 2$ is low, thus, that memory block isn't fully addressed and won't respond, while the memory enabled with the clock's $\phi 1$ signal will respond only to processor B.

SYSTEM DESIGN

Of the three dual processor systems described, the latter approach was used because of its simplicity, minimum package count, and common utilization of memory and software. The dual processor system was designed in accordance with the following set of requirements.

First the system will use one EXORciser with little modification, since the EXORciser has the needed bus structure and power supplies.

Second, both processors will execute the software resident to the EXORciser system, namely EXbug. Using this program will eliminate extensive software development for the system, since EXbug has routines to load and punch program tapes, change memory, and start execution of the programs entered, along with some program debugging capability.

Third, each MPU must have dedicated memory and I/O to accommodate the EXbug program. The I/O (serial communication utilizing an ACIA) must interface to both a TTY 20 mA loop and an RS-232C terminal interface. In addition, each MPU must have a ROM for vectoring to a system reset address.

Fourth, each processor will operate at a 1 MHz clock cycle time. With both processors operating at 1 MHz, the bus will operate at 2 MHz or 500 ns cycle time. To be included in the clock design is the ability to refresh memory on a "cycle-stealing" basis, thus transparent to the MPU. Also to be included should be the ability to slow the clock to allow data transfer for the slower responding bus parts.

Fifth, when one or both of the MPUs are halted (either by a WAI instruction or pulling the $\overline{\text{Halt}}$ line low) the address and data bus should go into the high-impedance state, with the exception of VMA. VMA should go to a "0" level so erroneous reading and writing does not occur. An option should be included so control of the bus may be taken over during the time when an MPU is in a halt condition.

CIRCUIT DESCRIPTION

The EXbug program requires a minimum of an ACIA, RAM, and ROM for its hardware support. These items are needed for program operation and are duplicated for each MPU. The addition of a PIA and its supporting hardware will enable the following EXbug commands: (;P), (n;P), (N), (;N), (n;N), ($T), and ($S). (See the EXORciser User's Guide for command descriptions.) Although not included in this system, the address decoding must allow for the PIA so that no other component on the bus will respond to that address.

Figure 4 shows the memory map for this system. Common user memory is allocated addresses $0000 to $EFFF (Hex). The EXbug program occupies addresses $F000 to $FBFF. The address block $FC00 to $FFFF is a block of memory dedicated to each MPU. In this block is the mutually exclusive hardware support for the EXbug program.

The block diagram for this system implemented as one module for the EXORciser is shown in Figure 5. This system is divided into three blocks plus the MPUs. The three blocks consist of Clock/Control, Bus Interface, and Dedicated Memory and I/O.

Clock and Control

The clock and control section is described in two sections: Reset and Clock Design.

Reset. The reset of either the A or the B MPU can be actuated by a number of signals (Figure 6). Both the A and B systems are reset by a power-on reset. When the +5 V power is turned on, the timer (MC1455) is automatically triggered and pulls the $\overline{\text{Master Reset}}$ line low, resetting the whole system. When the timer has timed out (about 400 ms) the reset line is brought high allowing both the A and B MPU to come out of the reset mode and start execution. The $\overline{\text{Master Reset}}$ line may also be pulled low by a signal from the bus. The bus master reset and the power-on reset are wire-ORed so either may cause the whole system to be reset.

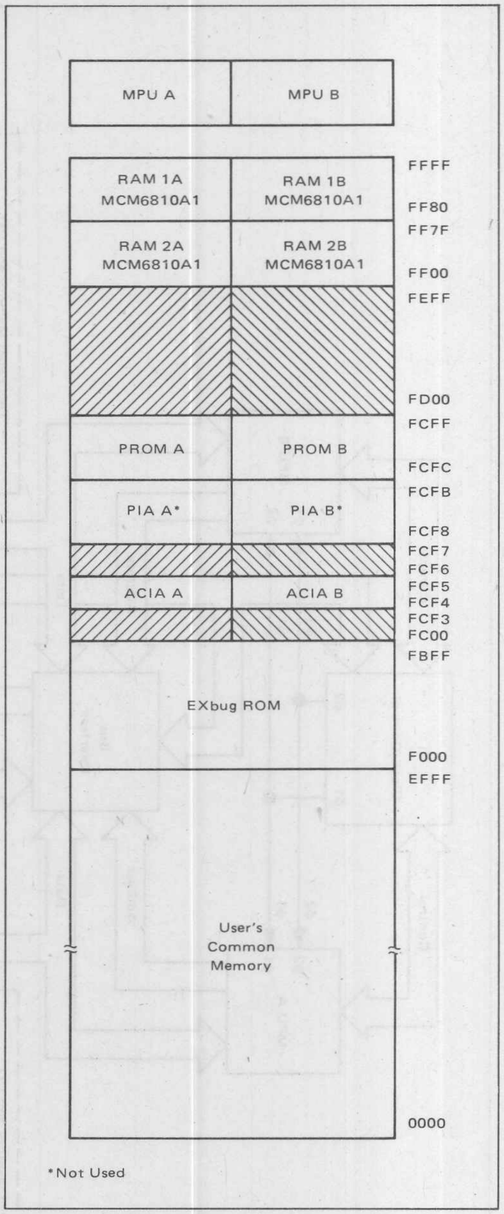


FIGURE 4 — Dual Processor System Memory Map

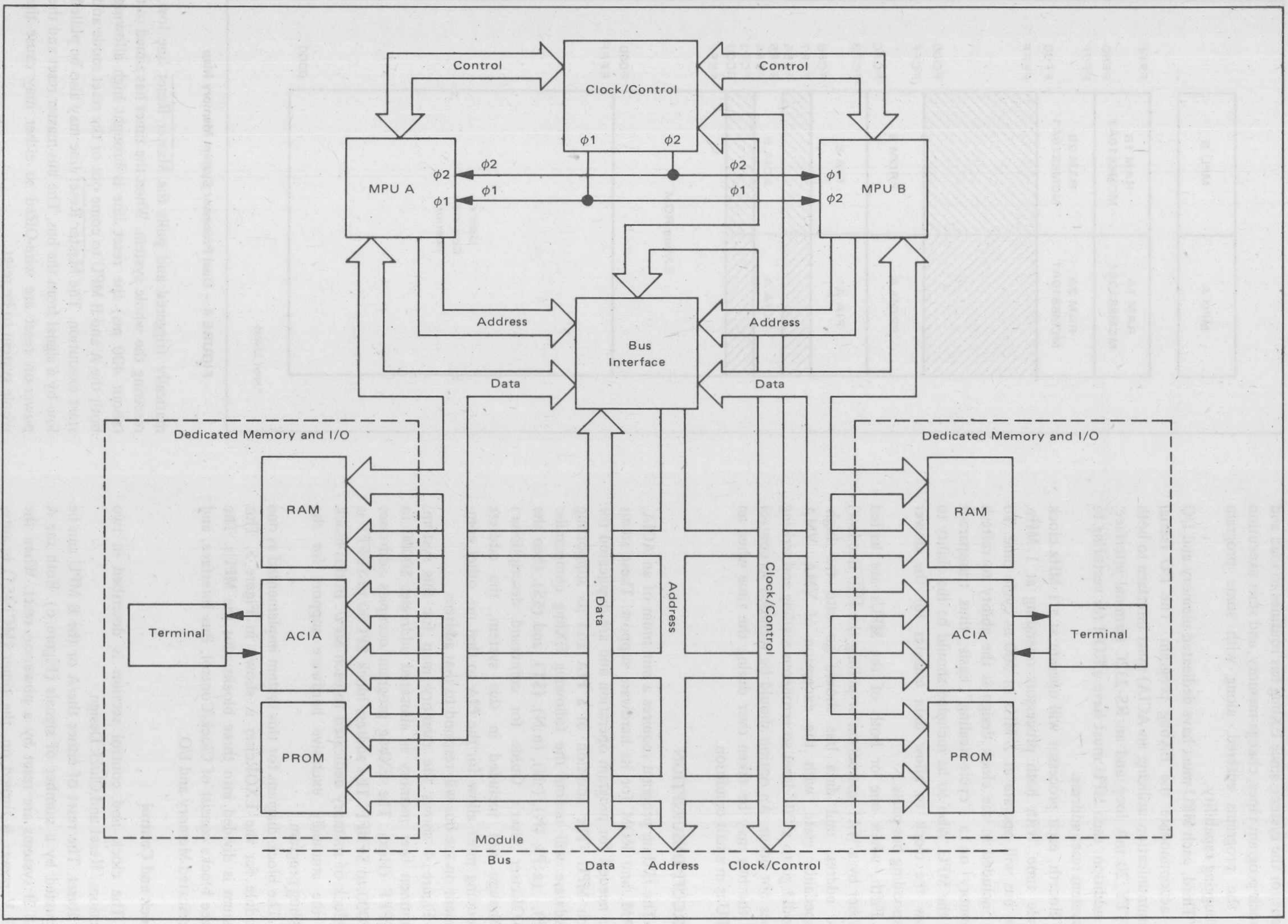


FIGURE 5 – Dual Processor System for Use with EXORciser

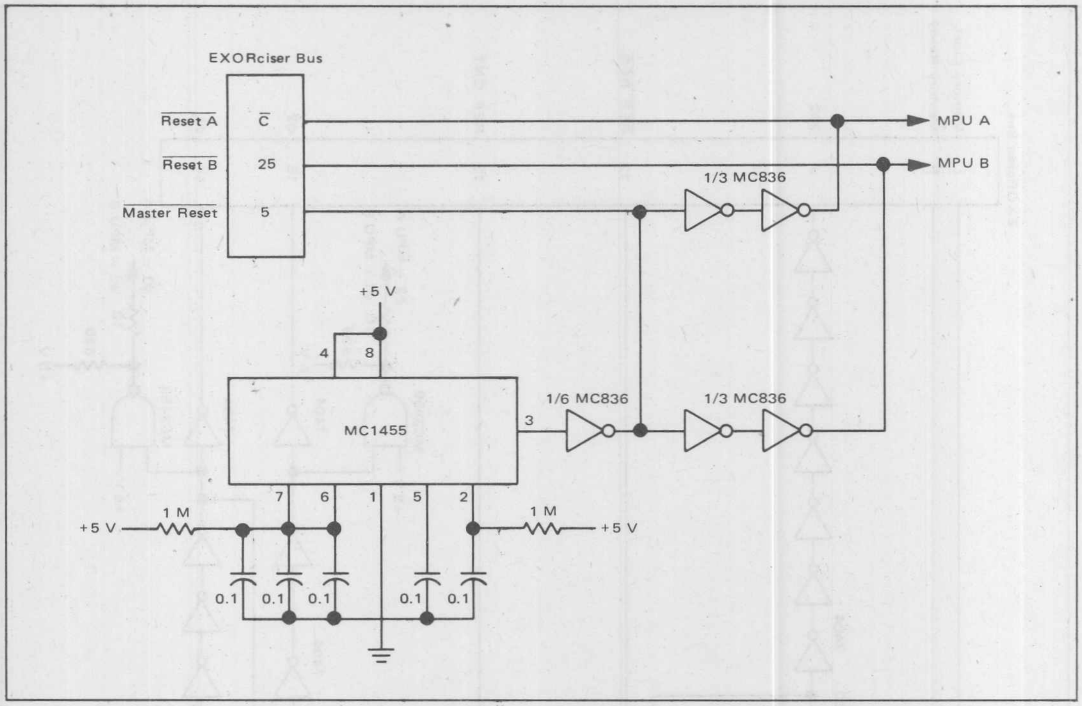


FIGURE 6 – System Reset

Each MPU also has its own reset line (Reset A and Reset B) which will reset the respective MPU and the bus parts connected to it. These two reset lines are pulled low each time the entire system is powered up or when the Master Reset line on the bus is pulled low. Also, an individual MPU may be reset by pulling the reset line associated with it (Reset A or Reset B) to ground. The reset from the bus is wire-ORed with the reset from power-on/Master Reset so either signal may reset the MPU.

Clock Design. The system clock can be analyzed in two parts: The bus clock and the MPU clock (Figures 7 and 8). The bus clock runs at 2 MHz and provides the 2ϕ2 signal for the bus. Additional circuitry is included in the bus clock to allow for the slow response time of some bus parts and for the refreshing of dynamic memory.

When using dynamic memory, a refresh cycle must be provided to recharge the memory cells. To execute a refresh cycle, the memory module generates a request for refresh (REF REQ) signal. This signal asks the clock for a refresh cycle. On the next 2ϕ1 portion of the clock cycle (see Figure 9A), the clock generates a refresh grant (REF GNT) signal telling the bus that a refresh cycle has been granted and it is now taking place. During the refresh cycle, the 2ϕ2 clock remains low for that cycle of the clock. Since some memory modules need a clock for timing during a refresh cycle, another clock signal is also generated called Memory Clock (MEM CLK). This clock signal is the same as 2ϕ2 except during a refresh cycle it continues to cycle while 2ϕ2 remains low for that clock period.

Also included in the bus clock design is the ability to stretch the 2ϕ2 or data transfer portion of the cycle. Since the clock period is 500 ns (2 MHz) some of the logic interfaced to the bus may not have enough time to respond to the fast rate and requires more time for the data transfer. When an MPU addresses such logic, the logic should respond by pulling the Memory Ready

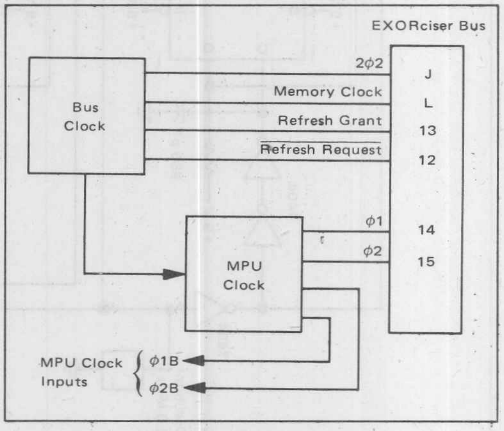


FIGURE 7 – System Clock Block Diagram

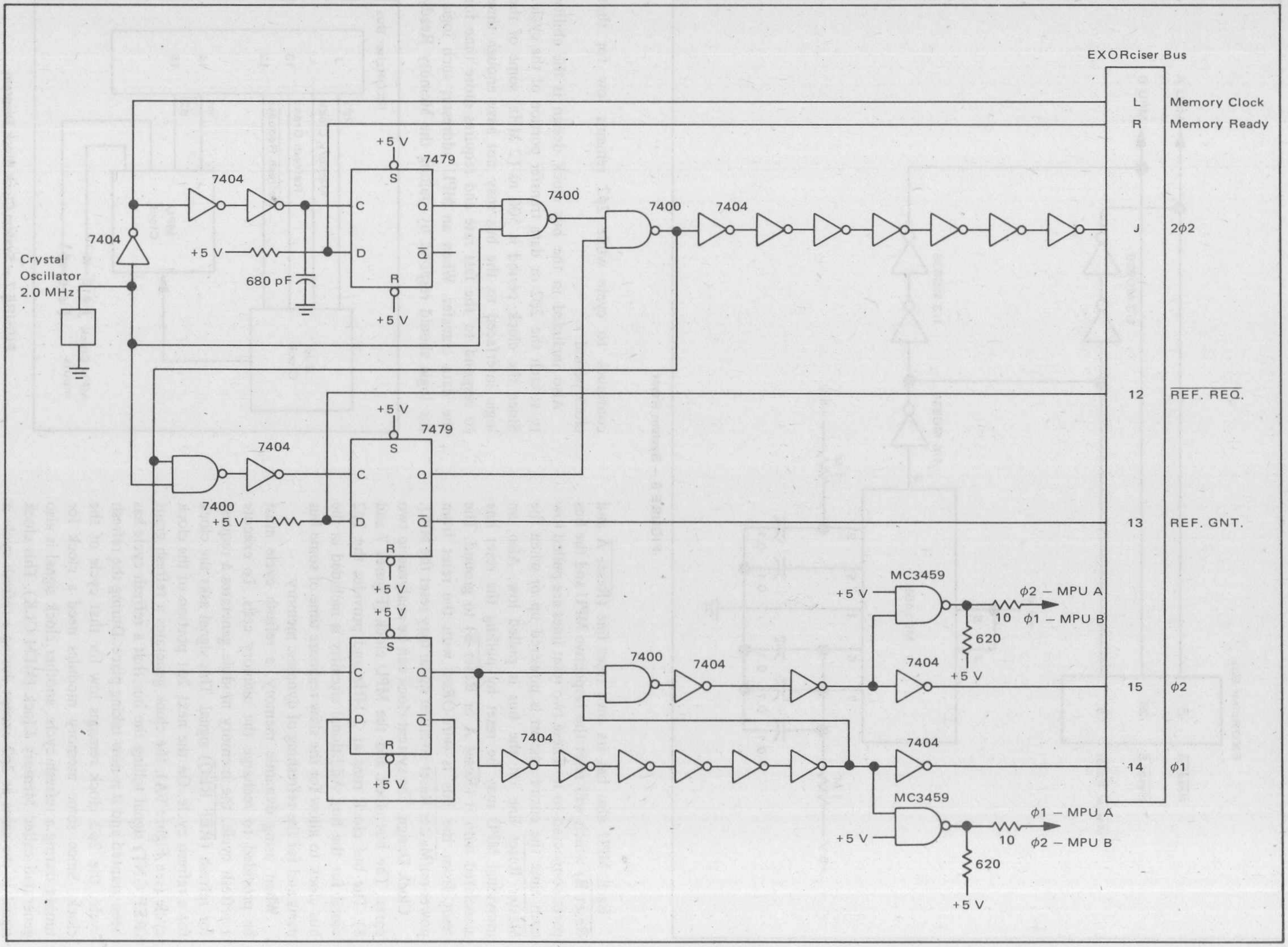


FIGURE 8 – System Clock Schematic

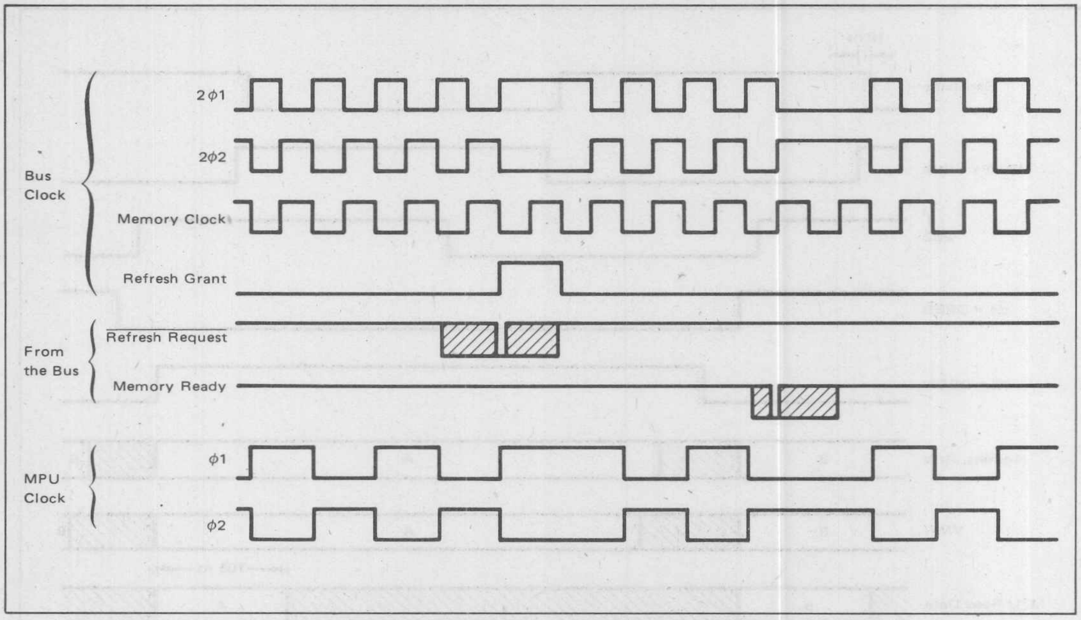


FIGURE 9A – System Clock Timing Diagram – Refresh Cycle, Slow Memory Cycle

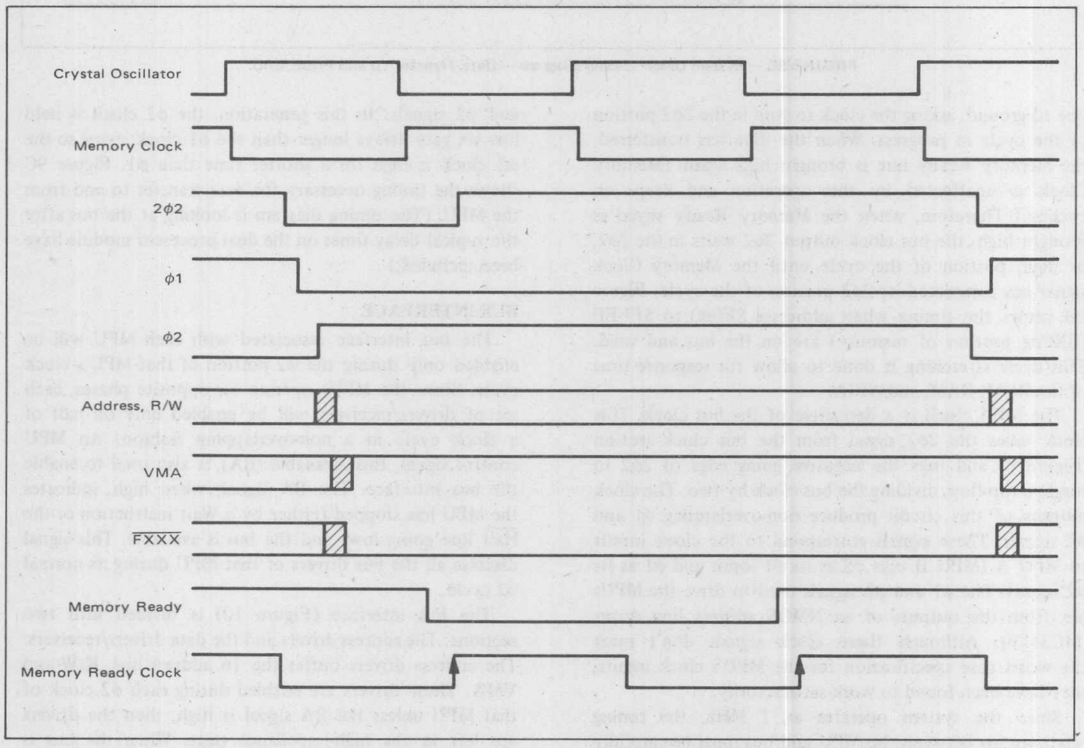


FIGURE 9B – System Clock Timing Diagram – Addresses $F000 to $FFFF on Bus and Valid

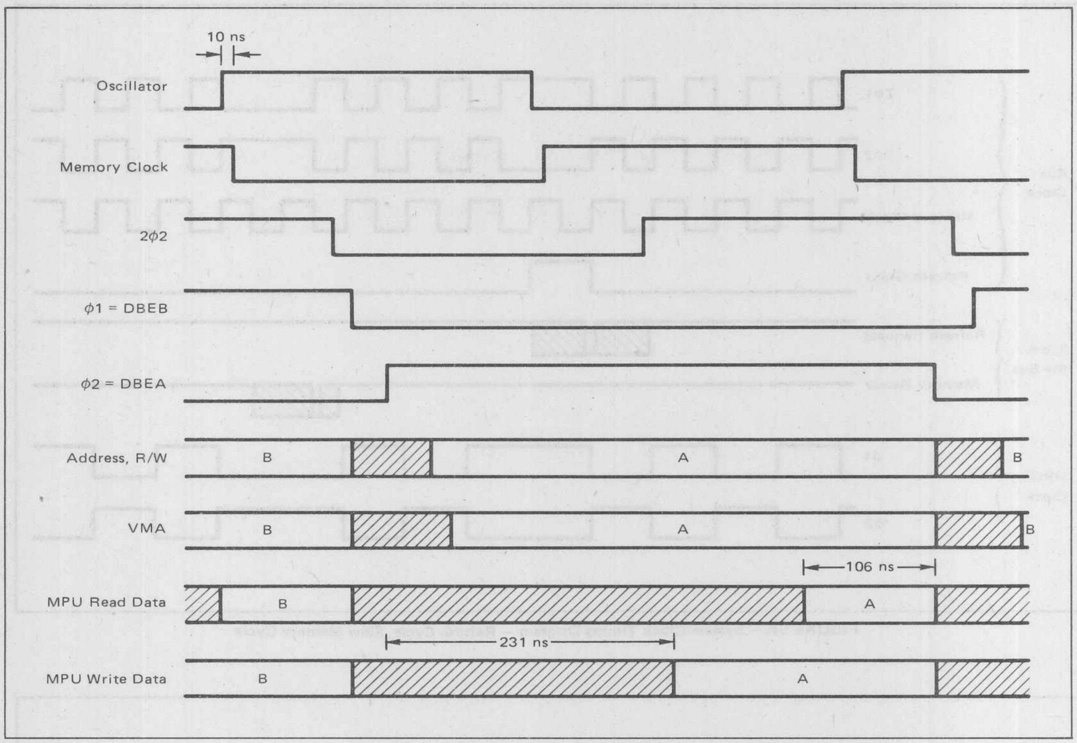


FIGURE 9C — System Clock Timing Diagram — Data Transfer To and From MPU

line to ground, asking the clock to stop in the 2ϕ2 portion of the cycle in progress. When the data has transferred, the Memory Ready line is brought high again. (Memory Clock is unaffected by this operation and keeps on cycling.) Therefore, when the Memory Ready signal is brought high, the bus clock output 2ϕ2 waits in the 2ϕ2, or high, portion of the cycle until the Memory Clock signal has completed its 2ϕ2 portion of the cycle. Figure 9B shows the timing when addresses $F000 to $FFFF (EXbug portion of memory) are on the bus and valid. This cycle stretching is done to allow for response time of the ROM, RAM, and ACIA.

The MPU clock is a derivative of the bus clock. This clock takes the 2ϕ2 signal from the bus clock section (Figure 8) and uses the negative going edge of 2ϕ2 to toggle a flip-flop, dividing the bus clock by two. The clock outputs of this circuit produce non-overlapping ϕ1 and ϕ2 signals. These signals correspond to the clock inputs to MPU A (MPU B uses ϕ2 as its ϕ1 input and ϕ1 as its ϕ2 input). The ϕ1 and ϕ2 signals used to drive the MPUs are from the outputs of an NMOS address line driver (MC3459). Although these clock signals don't meet the worst case specification for the MPU's clock inputs, they have been found to work satisfactorily.

Since the system operates at 2 MHz, the timing relationships between the MPU and bus must be carefully analyzed. The MPU clock generates non-overlapping ϕ1 and ϕ2 signals. In this generation, the ϕ2 clock is held low six gate delays longer than the ϕ1 clock signal so the ϕ2 clock is high for a shorter time than ϕ1. Figure 9C shows the timing necessary for data transfer to and from the MPU. (The timing diagram is looking at the bus after the typical delay times on the dual processor module have been included.)

BUS INTERFACE

The bus interface associated with each MPU will be enabled only during the ϕ2 portion of that MPU's clock cycle. Since the MPUs operate on opposite phases, each set of drivers/receivers will be enabled only on half of a clock cycle in a non-overlapping fashion. An MPU control signal, Bus Available (BA), is also used to enable the bus interface. The BA signal, when high, indicates the MPU has stopped (either by a Wait instruction or the $\overline{\text{Halt}}$ line going low) and the bus is available. This signal disables all the bus drivers of that MPU during its normal ϕ2 cycle.

The bus interface (Figure 10) is divided into two sections: The address drivers and the data drivers/receivers. The address drivers buffer the 16 address bits, R/W and VMA. These drivers are enabled during each ϕ2 clock of that MPU unless the BA signal is high, then the drivers are left in the high-impedance state. When the bus is available, the VMA' (see Figure 11) and VMA are normally

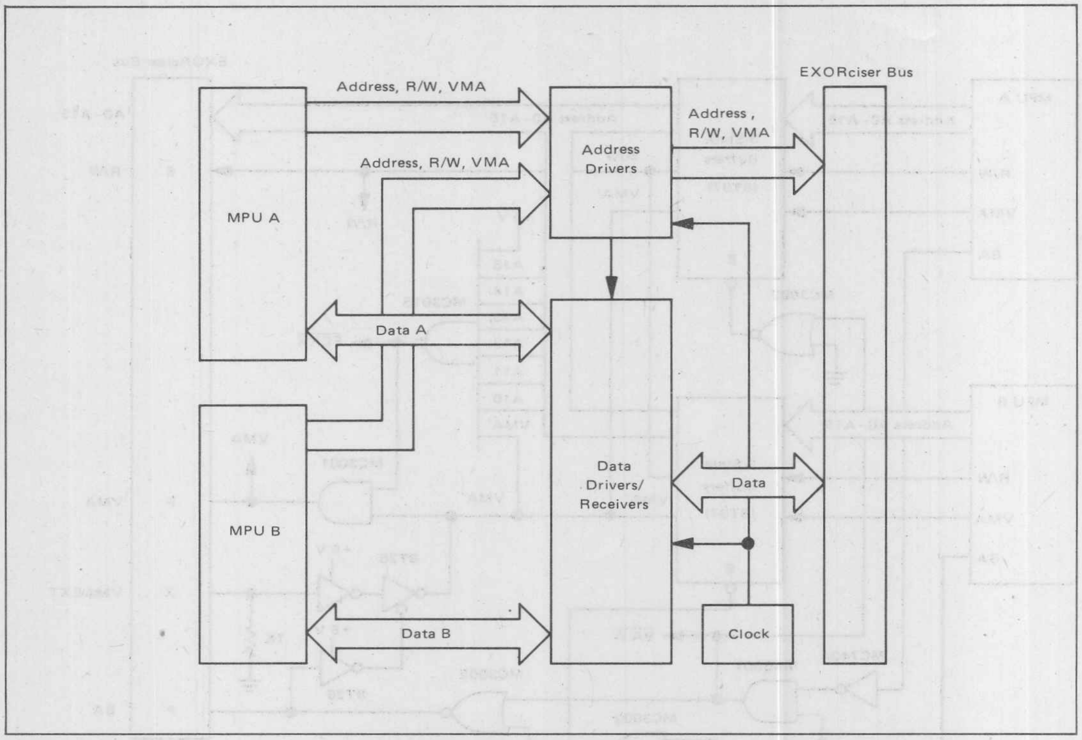


FIGURE 10 — Bus Interface Block Diagram

held at a "0" level. If some peripheral (i.e., Direct Memory Access) needs the bus, the VMAEXT signal must be used to get VMA on the bus to go to a "1" level. Here the peripheral will raise to a "1" level the VMAEXT line when the valid address is on the bus for data transfer.

The data drivers/receivers are bidirectional three-state devices. To enable either the drivers or receivers, it requires a combination of the R/W, VMA, BA and $\phi2$ signal for the MPU being buffered (Figure 12). The bus drivers are enabled when the MPU's clock is in its $\phi2$ portion, R/W is low indicating a write function and BA is low. The bus receivers are enabled when the MPU's clock is the the $\phi2$ portion, R/W is high indicating a read function, VMA is high indicating the address is valid and BA is low.

DEDICATED MEMORY AND I/O

To minimize the logic needed, the two MPUs and the dedicated memory and I/O were incorporated in one module for the EXORciser system. For each MPU, this will eliminate some of the data bus interface, since the ACIA, RAM, and ROM data lines may be wired directly to the respective MPU. In addition, by placing both A and B sides of the memory and I/O on one module, the address decoding redundancy may be eliminated.

Table 1 shows the address decoding necessary to uniquely decode each component (ACIA, PIA, RAM and ROM) as to its bus address. This addressing is enabled with a signal called $\overline{FCXX}$, where address bits A10 to A15 are at a logic 1 level. Also, in the enabling of each component is the clock's $\phi1$ or $\phi2$ signal, defining which MPU is talking to the bus.

The first two machine cycles after an MPU has been reset, the ROM is enabled and the RAM at the restart vector address is disabled. This offset is added to the ROM so the restart may vector to the proper address. After the first two cycles, the ROM is then addressed in its normal memory location (Figure 13).

The ROM also contains the control character necessary to program the ACIA. EXbug programs the ACIA according to the speed of the terminal, so when a teletypewriter is connected (110 baud), the ACIA is programmed for 1 start bit, 8 data bits, and 2 stop bits. Otherwise, the ACIA is programmed for 1 start bit, 8 data bits, and 1 stop bit. To indicate when a teletypewriter is connected, the TTY input line is grounded (Figure 14).

The TTY and terminal interface is shown in Figure 14. This circuit supports a 20 mA loop for TTY interface and a standard RS-232C terminal interface. Included is a bit rate generator to supply both ACIAs with the proper baud rate for the teletype or terminal connected.

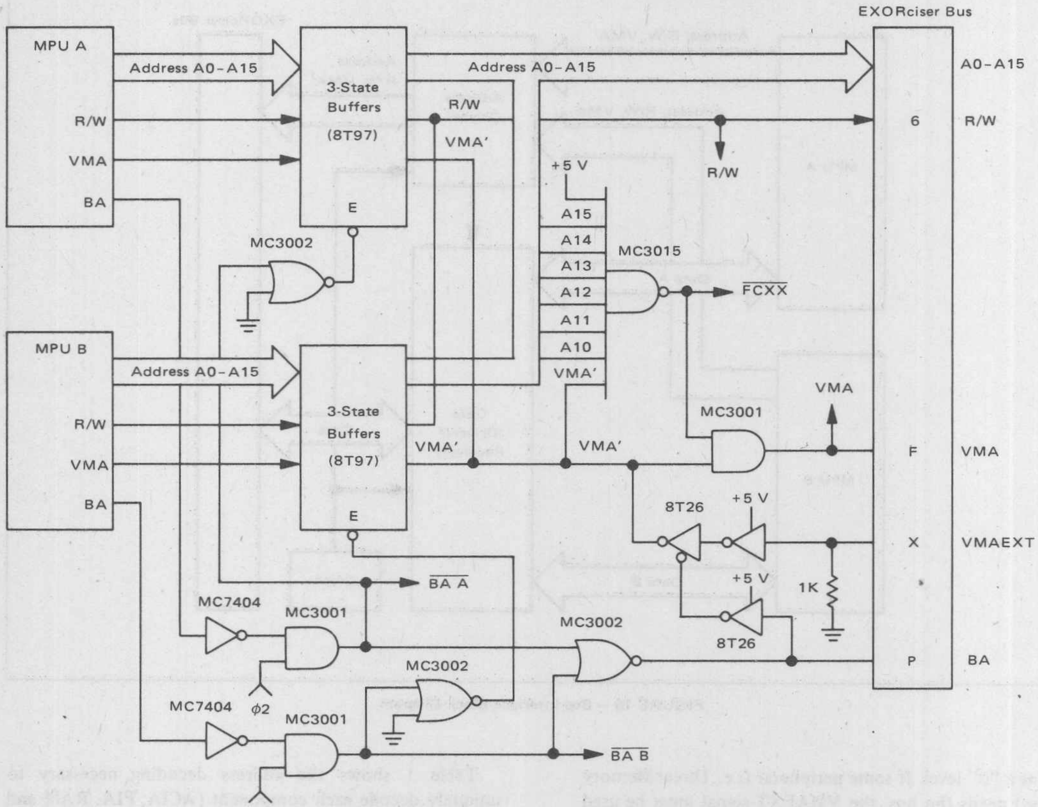


FIGURE 11 — Address Driver Logic

TABLE 1
Address Decoding
(Address Bits A10 to A15 = 1)

Device	Addresses	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
RAM 1	$FF80 – $FFFF	1	–	1	X	X	X	X	X	X	X
RAM 2	$FF00 – $FF7F	1	–	0	X	X	X	X	X	X	X
ROM	$FCFC – $FCFF	0	–	–	–	–	–	1	1	X	X
PIA	$FCF8 – $FCFB	0	–	–	–	–	–	1	0	X	X
ACIA	$FCF4 – $FCF5	0	–	–	–	–	–	0	–	–	X

Where: 1 = High Enable
0 = Low Enable
X = Address/Register Select
– = Don't Care

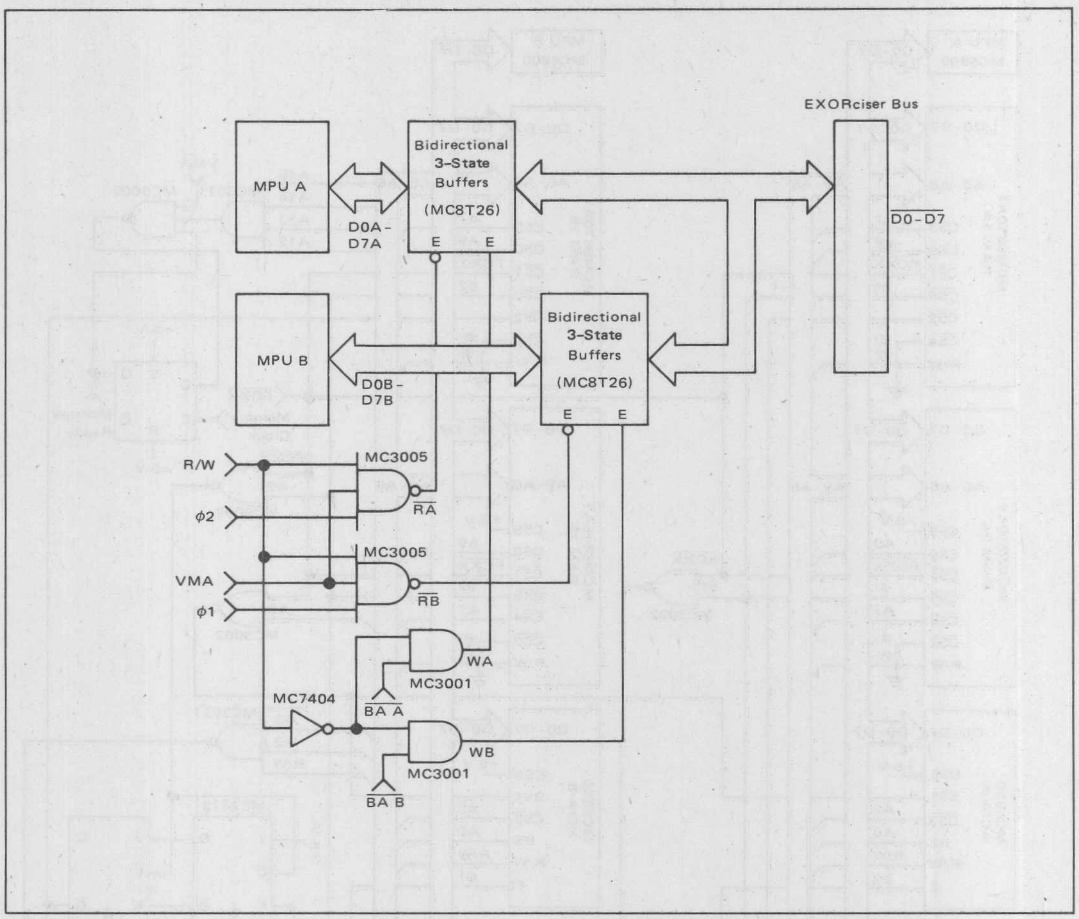


FIGURE 12 – Data Drivers/Receivers

MODIFICATIONS TO THE EXORciser

A few modifications to the basic EXORciser system were needed for the system to operate properly.

First, the MPU module supplied in the EXORciser must be removed. This is obvious, since the dual-processor system contains both of the MPUs for the system.

Second, U20 (MC7473) on the Debug Module must be removed and a short inserted between pins 13 and 11. This will disable the power-up/restart sequence of the module.

SUMMARY

The system as shown has utilized the following features of dual processing.

1. Share common ROM. Both processors share the common EXbug program ROM.

2. Dedicated block of memory. Both processors have exclusively at the same addresses an ACIA, RAM, and ROM.

3. One bus structure. This system shows how a one bus dual-processor system may be implemented. This also includes only one set of power supplies.

4. One clock circuit. This system uses one clock which eliminates duplication of hardware.

5. Both processors operate at 1 MHz. Except when executing the EXbug control program, both processors operate at 1 MHz doubling the overall system throughput. In this system, the EXbug program is used only for loading programs and program debugging. Once the dual-processor system's software has been debugged, the EXbug program is needed only to load the programs and initialize program execution.

After the modifications to the basic EXORciser were completed, the dual processor module was inserted into the EXORciser, terminals connected, and powered up. Both MPUs, automatically reset, started execution of the EXbug progam. A short program was executed in the common user memory to insure both processors operate at the 1 MHz speed.

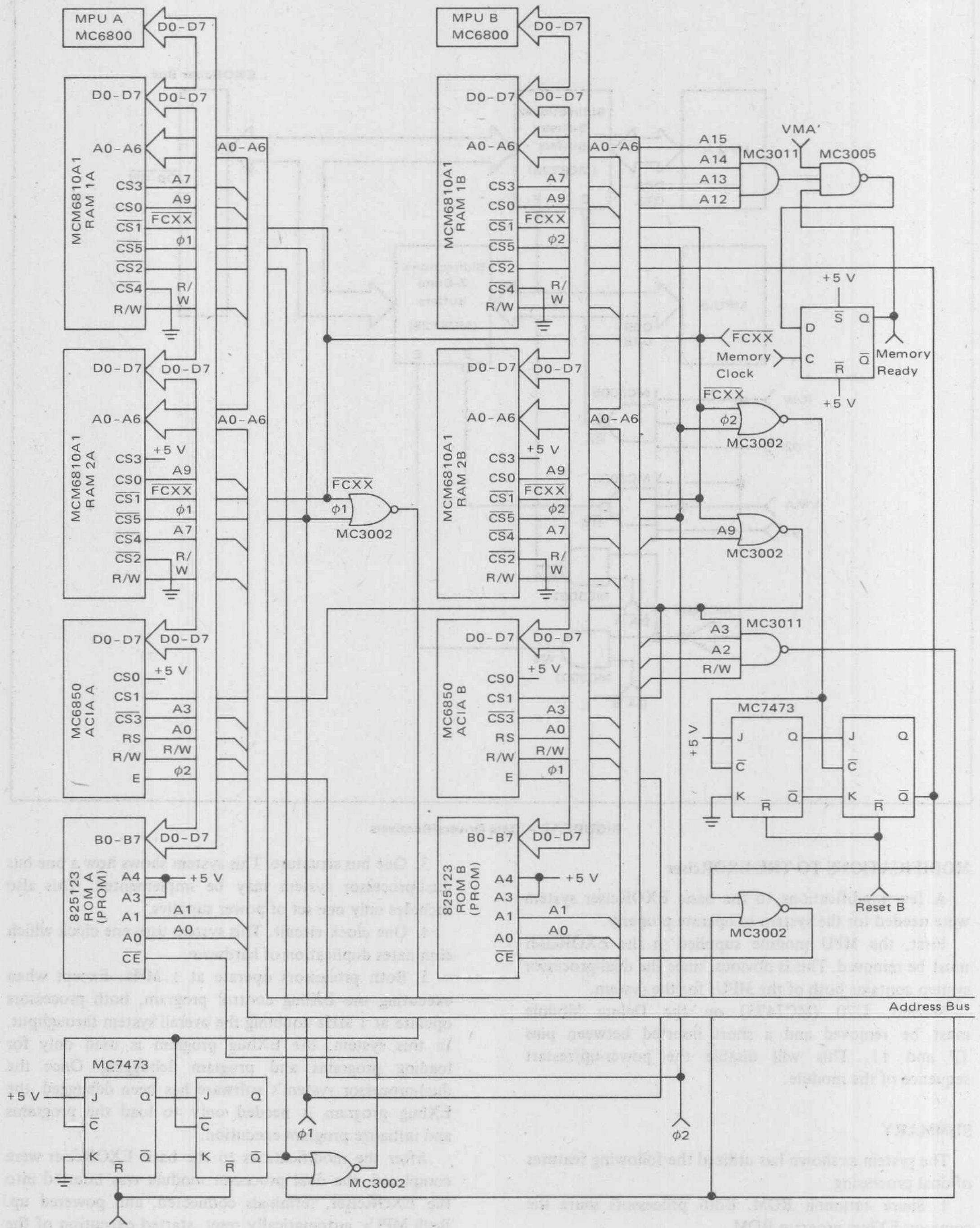


FIGURE 13 — Dedicated Memory and I/O

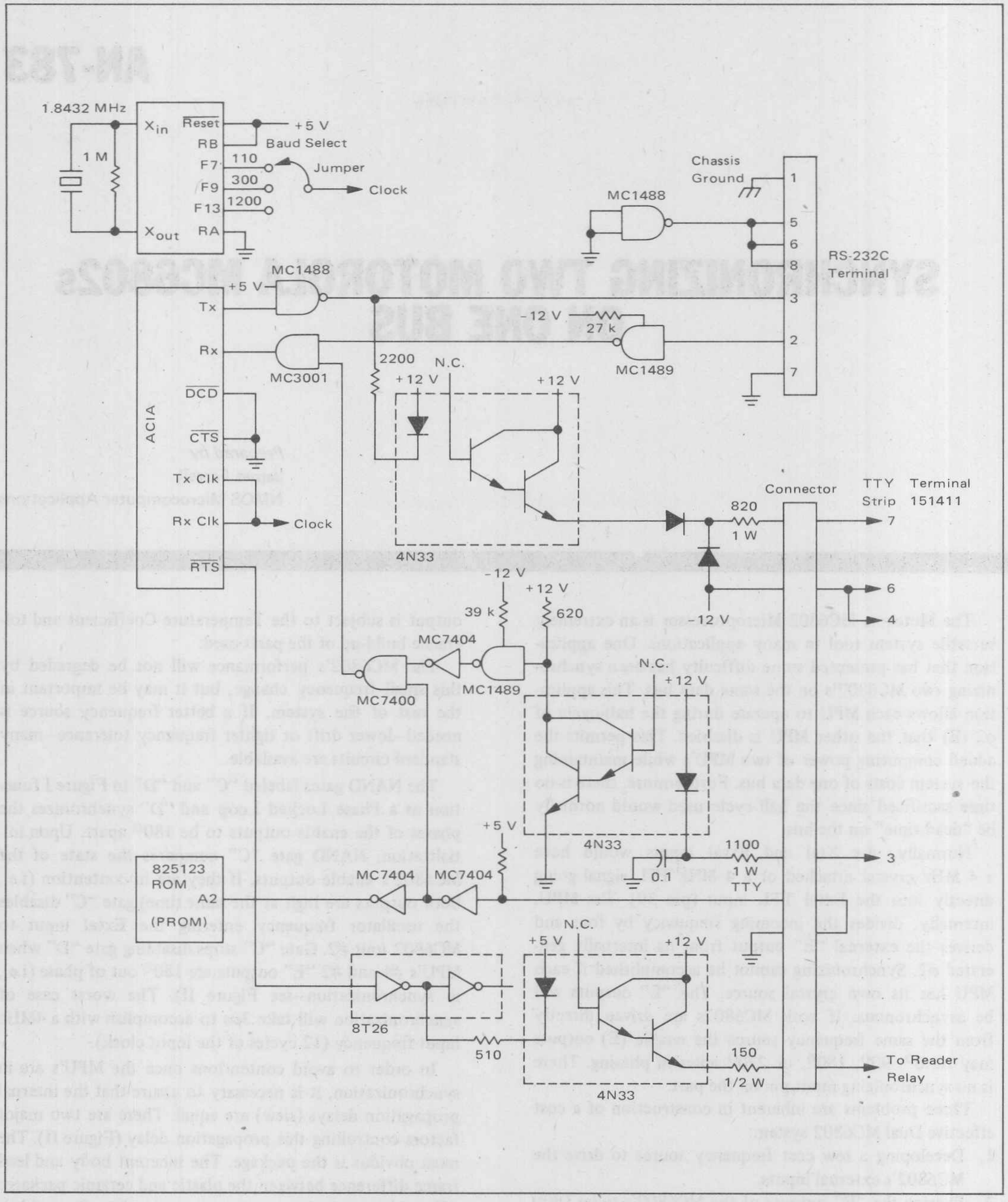


FIGURE 14 – Teletypewriter and Terminal Interface

AN-783

SYNCHRONIZING TWO MOTOROLA MC6802s ON ONE BUS

Prepared by
James Farrell
NMOS Microcomputer Applications

The Motorola MC6802 Microprocessor is an extremely versatile system tool in many applications. One application that has presented some difficulty has been synchronizing two MC6802's on the same data bus. This application allows each MPU to operate during the half-cycle of $\phi2$ (E) that the other MPU is disabled. This permits the added computing power of two MPU's while maintaining the system costs of one data bus. Furthermore, there is no time sacrificed since the half-cycle used would normally be "dead time" on the bus.

Normally, the Xtal and Extal inputs would have a 4 MHz crystal attached or a 4 MHz TTL signal going directly into the Extal TTL input (pin 39). The MPU, internally, divides the incoming frequency by four and derives the external "E" output from its internally generated $\phi2$. Synchronizing cannot be accomplished if each MPU has its own crystal source. The "E" outputs will be asynchronous. If both MC6802s are driven directly from the same frequency source the enable (E) outputs may be 0°, 90°, 180°, or 270° apart in phasing. There is no synchronizing input pin on the part.

Three problems are inherent in construction of a cost effective Dual MC6802 system:

1. Developing a low cost frequency source to drive the MC6802's external inputs.
2. Phasing the "E" outputs of the MC6802's to be 180° apart reliably before the start-up reset is disabled.
3. Insuring the internal propagation delays (sometimes called "slewing") are nearly identical to avoid overlapping of the "E" outputs when they are high.

The NAND gates labeled "A" and "B" on Figure I are used as an extremely low cost frequency source. This approach is reliable and always initializes. The frequency output is subject to the Temperature Coefficient and tolerance build-up of the parts used.

The MC6802's performance will not be degraded by this small frequency change, but it may be important in the rest of the system. If a better frequency source is needed—lower drift or tighter frequency tolerance—many standard circuits are available.

The NAND gates labeled "C" and "D" in Figure I function as a Phase Locked Loop and "D" synchronizes the phases of the enable outputs to be 180° apart. Upon initialization, NAND gate "C" compares the state of the MC6802's enable outputs. If they are in contention (i.e., both outputs are high at the same time) gate "C" disables the oscillator frequency entering the Extal input to MC6802 unit #2. Gate "C" stops disabling gate "D" when MPU's #1 and #2 "E" outputs are 180° out of phase (i.e., in synchronization—see Figure II). The worst case of synchronization will take 3μs to accomplish with a 4MHz input frequency (12 cycles of the input clock).

In order to avoid contentions once the MPU's are in synchronization, it is necessary to assure that the internal propagation delays (slew) are equal. There are two major factors controlling this propagation delay (Figure II). The most obvious is the package. The inherent body and lead frame difference between the plastic and ceramic packages offer different body capacitances to the chip. Since this is a consistent value, no problem will be encountered if the same package is used in both positions. Changes in the chip design will also cause a timing change. It is the nature of the state of the art in the NMOS IC business that the chip will be changed as time goes on. This problem can be avoided by using parts with matching date codes, thereby avoiding using two "different" MC6802's.

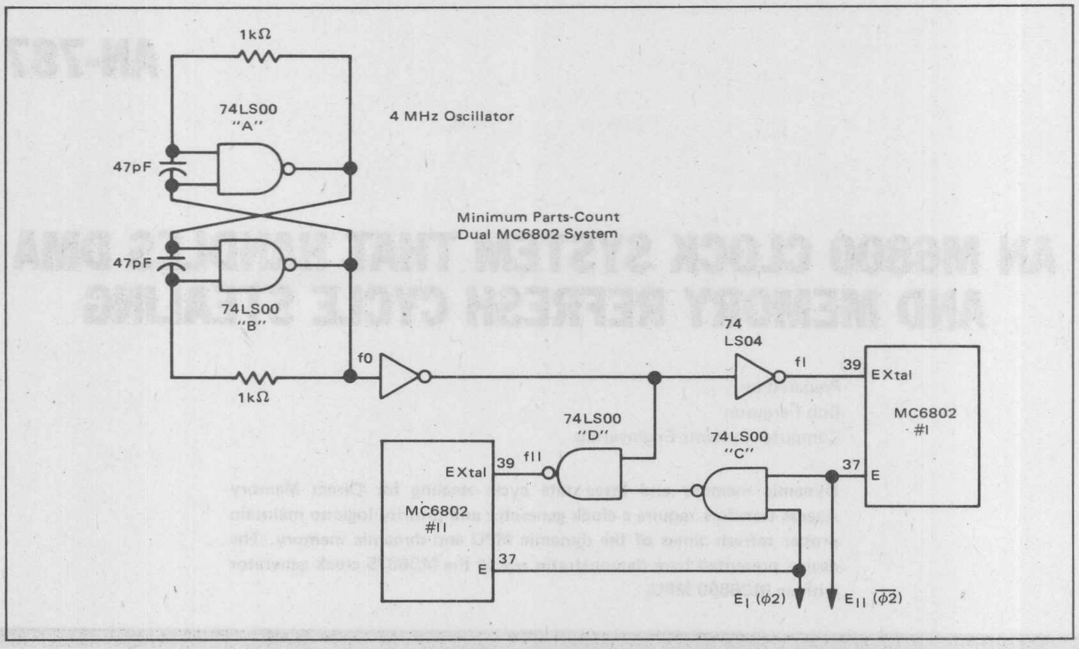


FIGURE I – 4 MHz Oscillator

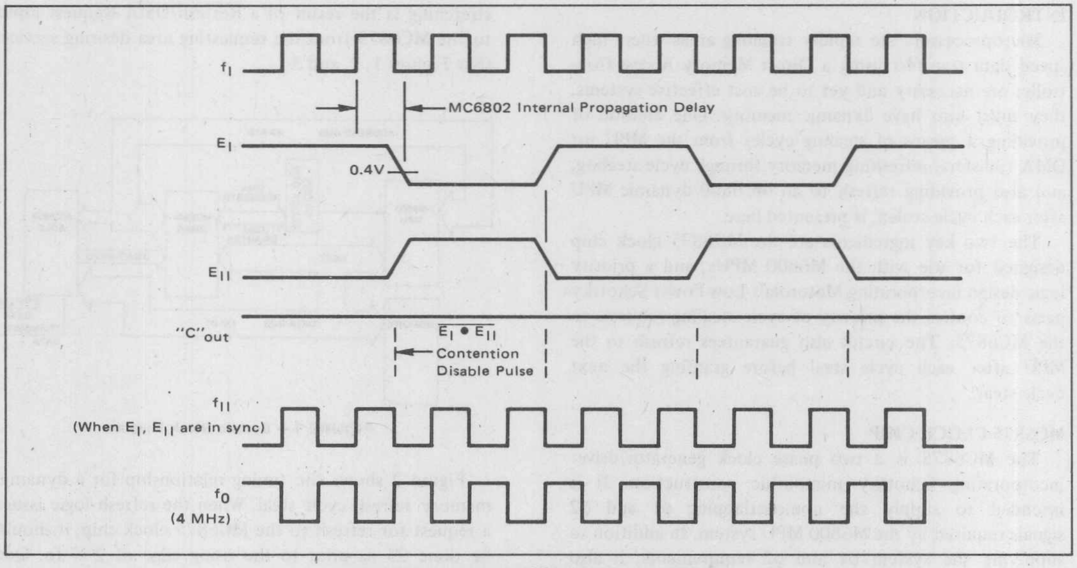


FIGURE II – Dual MC6802 Synchronizer Timing

AN-787

AN M6800 CLOCK SYSTEM THAT HANDLES DMA AND MEMORY REFRESH CYCLE STEALING

Prepared by
Bob Ferguson
Computer Systems Engineering

Dynamic memory and three-state cycle stealing for Direct Memory Access transfers require a clock generator and priority logic to maintain proper refresh times of the dynamic MPU and dynamic memory. The design presented here demonstrates use of the MC6875 clock generator with an MC6800 MPU.

INTRODUCTION

Microprocessors are rapidly reaching areas where high speed data transfers using a Direct Memory Access Controller are necessary and yet to be cost effective systems, they must also have dynamic memory. One method of providing a means of stealing cycles from the MPU for DMA transfers, refreshing memory through cycle stealing, and also providing refresh to an MC6800 dynamic MPU after each cycle stolen, is presented here.

The two key ingredients are the MC6875 clock chip designed for use with the M6800 MPUs, and a priority logic design incorporating Motorola's Low Power Schottky parts to control the priority of cycle stealing requests to the MC6875. The circuit also guarantees refresh to the MPU after each cycle steal before granting the next cycle steal.

MC6875 CLOCK CHIP

The MC6875 is a two phase clock generator/driver incorporating Schottky monolithic construction. It is intended to supply the non-overlapping $\phi 1$ and $\phi 2$ signals required by the M6800 MPU system. In addition to supplying the system $\phi 1$ and $\phi 2$ requirements, it also provides two free running oscillators, one at twice the MPU clock rate (2 × fo) and the second with four times the MPU clock rate (4 × fo). These are useful as system synchronization signals (see Figure 4).

The MC6875 clock chip permits cycle stealing from the MC6800 by holding its MPU $\phi 1$ output high, its MPU $\phi 2$ and Bus $\phi 2$ outputs low, during a cycle steal Grant. Memory $\phi 2$ continues to run during the Grant, allowing memory to be refreshed or DMA transfers to be done during this time. The Grant output and clock stretching is the result of a $\overline{\text{Refresh/DMA}}$ Request input to the MC6875 from the requesting area desiring service. (See Figures 1, 2, and 3.)

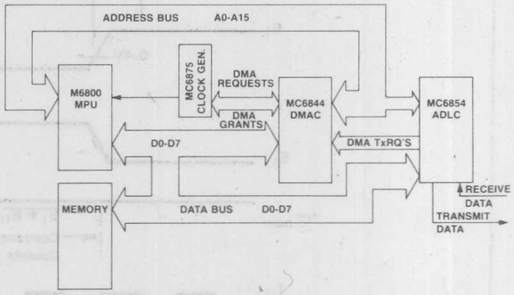
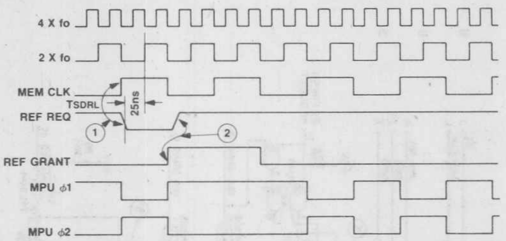


FIGURE 1 – System Block Diagram

Figure 2 shows the timing relationship for a dynamic memory refresh cycle steal. When the refresh logic issues a request for refresh to the MC6875 clock chip, it should be done 25 ns prior to the rising edge of 2 × fo. One means of accomplishing this is to use the rising edge of Memory Clock. This allows time enough for setup and at the same time eliminates the need of special logic. Grant is issued by the MC6875 at the beginning of the next cycle following a request input from the Priority Logic. $\phi 1$ is stretched high during the Grant time plus one half cycle, while Bus $\phi 2$ and MPU $\phi 2$ are stretched low. Should a Refresh Request arrive at the Priority Logic while a DMA Grant is going on, the input latch will hold the request. (See Figure 4, U18.)

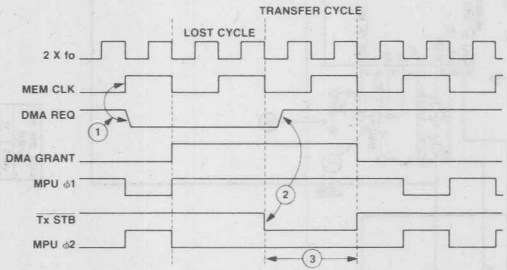


(1) SHOWN BASED ON MEMORY CLOCK

(2) REFRESH IS RELEASED ON THE RECEIPT OF GRANT OR AFTER THE TRAILING EDGE OF MEMORY CLOCK.

FIGURE 2 – MC6875 Interface Timing: Dynamic Memory Refresh

Figure 3 shows the timing of a DMA Controller (MC6844) DMA Request and DMA Grant in the three-state control (TSC) steal mode. DMA Request is timed by the DMAC. Note that the DMA Request does not go high again with the receipt of Grant by the DMAC. Rather the DMA Request remains low until the DMAC issues the $\overline{\text{Tx}}$ $\overline{\text{STB}}$ output. This strobe pulse indicates when a DMA transfer is occurring. The Tx STB time period is when the address valid (VMA) and three-state control (TSC) signals are placed on the system bus. Tx STB also provides the chip select signal, and selects the predetermined address to be input to the peripheral part. (See Figure 5.)



(1) DMA REQUEST THOUGH A FUNCTION OF MEMORY CLOCK, COMES LATER BASED ON INTERNAL TIMING OF THE DMAC.

(2) DMA REQUEST IS RELEASED FOLLOWING $\overline{\text{Tx}}$ $\overline{\text{STB}}$ GOING LOW.

(3) DMA TRANSFER OCCURS DURING CYCLE 2 OF DMA GRANT.

FIGURE 3 – MC6875 Timing: DMA (MC6844)
Three-State Steal Mode

HOOKUP

The MC6875 may be operated with a series resonant crystal having an internal impedance of 35–60 Ω, with an LRC network, or an RC network. In the event a crystal is used, some crystals may require a capacitor in parallel with them. The size necessary would be between 15 to 30 pF. The purpose is to act as a damper so the clock chip does not start at the second or third harmonic of the crystal (Figure 4, point 1).

A power on reset function is built into the MC6875 to enable the M6800 MPU to trap to its power-up vector address. The $\overline{\text{Power On Reset}}$ input should have a capacitor to ground as its only input. Using any type of switching input would result in the loss of memory contents should a Refresh Request be present with the $\overline{\text{Power On Reset}}$ input low. This results because the MC6875 will not service requests for cycle stealing during the time the input is low (Figure 4, point 2).

The two MPU clock outputs ϕ1 and ϕ2 may require a resistor in series with them to the input of the MC6800. This allows for damping. The Bus ϕ2 and the Memory Clock ϕ2 outputs may also require a resistor in series with their outputs to the load. These should all be in the 10 to 30 ohm range (Figure 4, point 3).

PRIORITY LOGIC

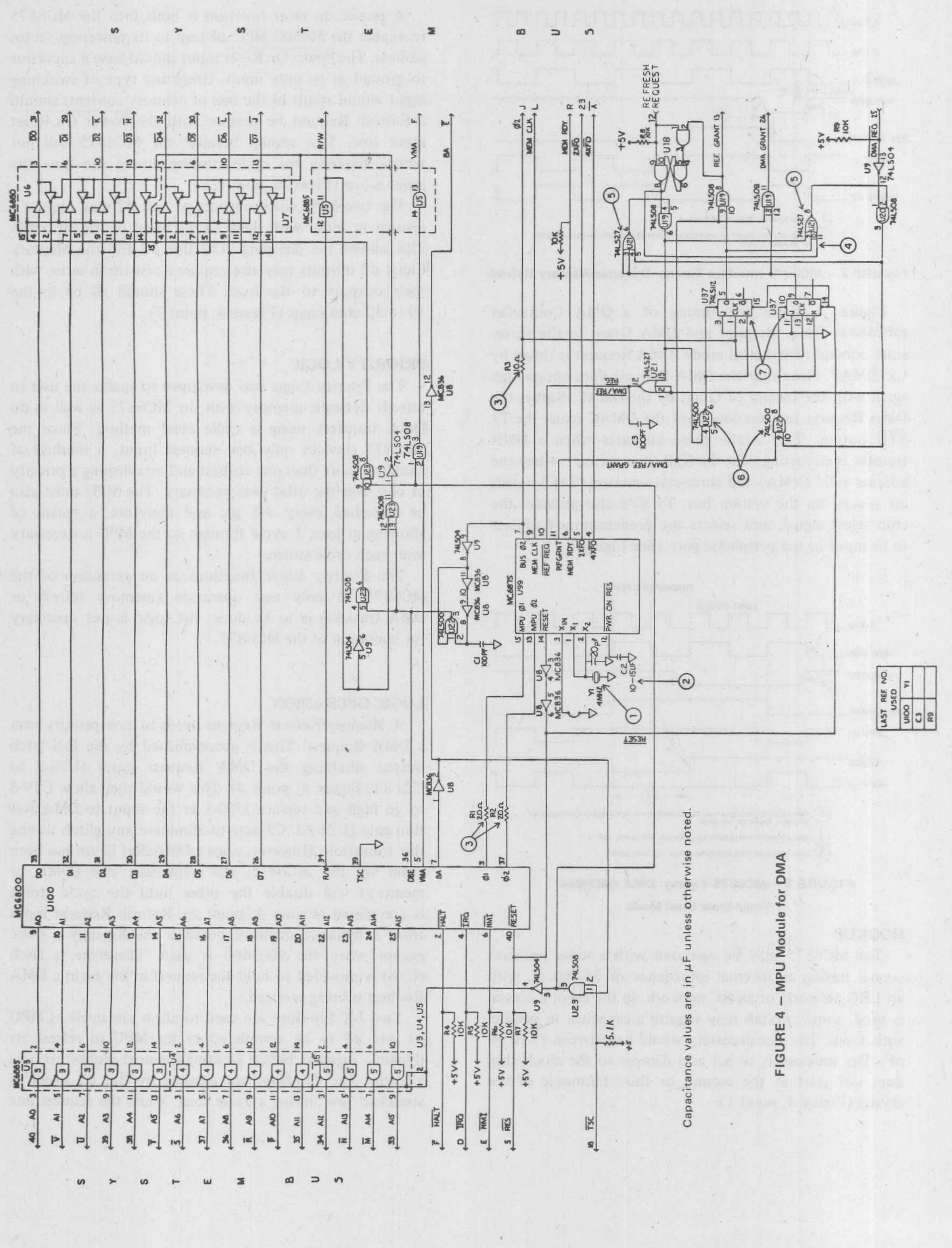
The Priority Logic was developed to enable the user to refresh dynamic memory with the MC6875 as well as do DMA transfers using a cycle steal method. Since the MC6875 services only one request input, a method of handling more than one request and establishing a priority of one over the other was necessary. The MPU must also be refreshed every 4.5 μs; and therefore, a means of allowing at least 1 cycle through to the MPU is necessary after each cycle stolen.

The Priority Logic functions as an extension of the MC6875. If only one operation (memory refresh or DMA transfer) is to be done, the logic is not necessary for operation of the MC6875.

LOGIC OPERATION

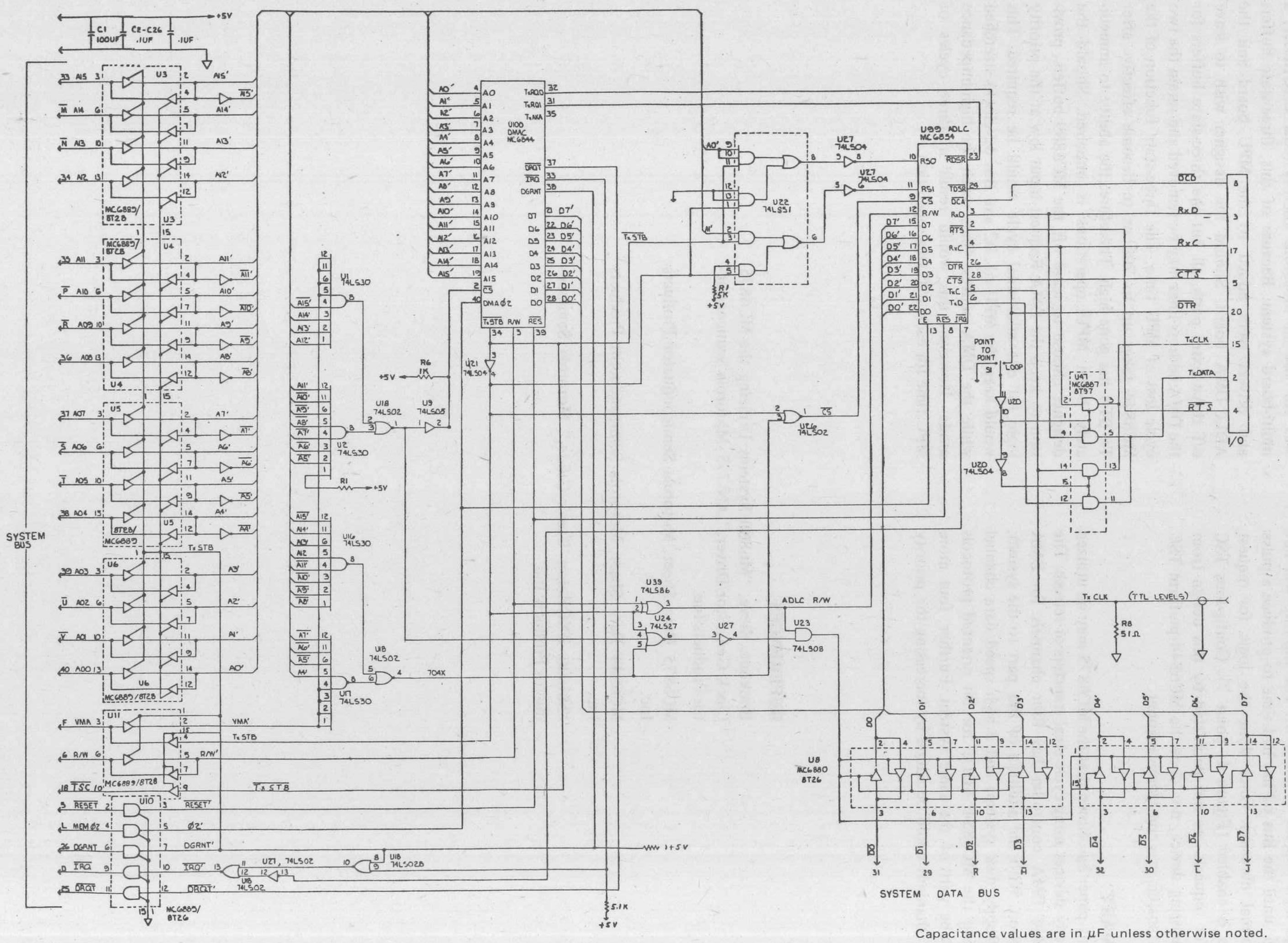
A Memory Refresh Request needs to have priority over a DMA Request. This is accomplished by the R-S latch output disabling the DMA Request input (U18-8 to U21-11; Figure 4, point 4). This would then allow U19-6 to go high and replace U20-3 as the input to DMA/Ref Req gate (U21-A). C3 acts to eliminate any glitch during this transition. However, once a DMA/Ref Grant has been given by the MC6875, the requesting side (DMA or memory) will disable the other until the cycle stolen is completed (Figure 4, point 5). Refresh Request input from a dynamic memory board is usually only a pulse except when the MC3480 is used. Therefore, a latch (U18) is provided to hold the request in the event a DMA Request is being serviced.

Two J-K flip-flops are used to allow one cycle of MPU ϕ1 and ϕ2 to be completed in the MPU to refresh its dynamic registers before granting the next request to steal a cycle. The flip-flops are clocked by Bus ϕ2 which is stretched low during a cycle steal. When the request line



Capacitance values are in µF unless otherwise noted.

FIGURE 4 MPU Module for DMA



Capacitance values are in μF unless otherwise noted.

FIGURE 5 — ADLC-DMA-Loop Board

goes high for the DMA side or the latch is reset for the memory side, the K input of the respective flip-flop will go high (Figure 4, point 6). The flip-flop will not change until the Bus $\phi 2$ has had time to produce a pulse of normal duration. Once done the logic for request inputs is enabled (Figure 4, point 7). Contiguous TSC transfer requests are not permitted by this design from a requesting device, nor does the MC6844 perform TSC steal transfers in a contiguous manner.

SUMMARY

This paper has shown that the MC6875 may be utilized by many devices using cycle steal transfers or refresh. The MC6844 DMA controller has four channels for DMA operation. With the addition of this part to the system, two floppy disk systems and a high speed data channel utilizing the MC6854 ADLC for bit oriented protocols could be part of the same system. Further, four more DMA channels could be added by increasing the priority logic by one more stage. Such a system is unlikely, but still it is possible. The MPU board, as it was designed, was for use in Motorola Microsystems EXORciser, a multi-board system. Because of this, three-state buffers and drivers were added to the MPU board and the ADLC-DMA board. Should the designer wish to leave off these extra parts, all but the three-state buffers for the DMA address lines may be removed and retain the two cycle loss of MPU time. The three-state function of the MC6844 takes up to 700 ns to become effective after Tx STB has gone high. Therefore, the ability to immediately begin MPU operations is impaired. Should the designer choose to leave off the MC6889 buffers, provision to keep the $\overline{\text{DMA Request}}$ input low at the priority logic for one additional cycle would be required. This would keep the MPU $\phi 1$, $\phi 2$ and Bus $\phi 2$ clocks stretched while the DMA address lines reach the high-impedance mode. However, this would result in three cycles of MPU time for each TSC steal operation.

REFERENCES

Bookout, Steve, "M6800 Systems Utilizing the MC6875 Clock Generator/Driver," AN-775, Motorola Semiconductor Products Inc.

MC6875 Data Sheet, Motorola Semiconductor Products Inc.

MC6844 Data Sheet, Motorola Semiconductor Products Inc.

"M6800 EXORciser User's Guide," Motorola Semiconductor Products Inc.

AN-797

MC6801/03 PORT EXPANSION

Prepared by
Arnold J. Morales
Application Engineer

MC6801/MC6803 PORT EXPANSION

The I/O capabilities of the MC6801/03 can be easily expanded. In effect, the number of I/O ports available can be increased to accomodate user needs with simple designs utilizing relatively inexpensive parts.

This application note describes several methods of port expansion.

MC6801/03 EXPANDED MULTIPLEXED MODE PORT EXPANSION

Figure 1 illustrates several methods for increasing the I/O capability of the MC6801/03 in the expanded multiplexed modes.* All these methods utilize ICs interfaced to the data bus.

Figure 1a shows a means of using the SN74LS374 as a latched input port. A strobe from a remote peripheral or MPU is used to latch data into the LS374. The MC6801/03 can read the latched contents of the LS374 by pulling $\overline{\text{O.E.}}$ low utilizing E·I/O Select as shown. When not selected the LS374 is held in a high-impedance state, thus eliminating data bus contention.

The SN74LS374 can also be used as a latched output port, as shown in Figure 1b. Data from the MC6801/03 is latched into the LS374 on the falling edge of E when I/O Select is high. Latched data can be continuously presented to the remote peripheral or MPU by tying $\overline{\text{O.E.}}$ low, as shown, or can be gated by the remote peripheral by an appropriate "I/O Select" logic control of $\overline{\text{O.E.}}$

An unlatched input port can be designed using a SN74LS244, as shown in Figure 1c. The octal three-state buffer presents data to the data bus with E·I/O Select and is particularly suited for MPU read of switches using polling. Once again, when not "selected," this peripheral is held in a high-impedance state, thus eliminating bus contention on the data bus.

A more straightforward approach to port expansion is illustrated in Figure 1d. Here an MC6821 is used to yield a net gain of two handshaking bidirectional ports. The MC6801/03 is compatible with all 6800 family peripherals, including the MC6821. Programming, pinout, and interface information for the MC6821 is provided in its data sheet, readily available from Motorola field offices and distributors.

MC6801/03 SCI PORT EXPANSION

Port expansion is possible using the MC6801/03 SCI (Serial Communications Interface) operated in the standard NRZ format. This format consists of a start bit (low), eight data bits, and one stop bit (high). When no data is being sent the line is held high, indicating an idle line.

All the SCI port expansion schemes described in this application note utilize the SCI clock brought out externally from the MC6801/03 as provided by a software option.

SCI PORT OUTPUT EXPANSION

Figure 2 shows a means of expanding the SCI to 16 sets of 4-bit nibbles yielding 64 I/O output lines.

The key element of this circuit is the start-bit recognition system. The serial bit stream is inverted before reaching the SN74LS164 serial-in, parallel-out shift register. An idle line, therefore, loads the shift register and the two LS74's (U2, U3) with zeros. The SN74LS154 decoder is consequently disabled with $\overline{E}$ high, keeping its outputs high. During this time no clocking is provided to the SN74LS175's, and their data outputs are unchanged.

Data transmitted out of the SCI is preceded by a start bit. This low start bit is inverted and clocked through the LS164 and the LS74's as a high. One-half clock time after the start bit is clocked into F/F U2, it is clocked into F/F U3. At this time the eight SCI data bits are in the LS164. The LS154 is enabled and decodes the four SLB's of the data (transmitted first). The four MSB's of the SCI data byte are I/O output data and are tied to all LS175's with appropriate buffering.

Decoding by the LS154 clocks the addressed 175, latching in the data.

One-half SCI clock time later, the LS164 is cleared by driving the $\overline{MR}$ pin low. F/F U2 is also cleared. The system is now ready for another SCI data byte. If no more data is transmitted, 1's are transmitted indicating an idle line and output data remains unchanged.

Figure 3 illustrates a scheme for expanding the SCI port to an 8-bit output port.

*The MC6803 is limited to expanded multiplexed (modes 2 and 3) operation.

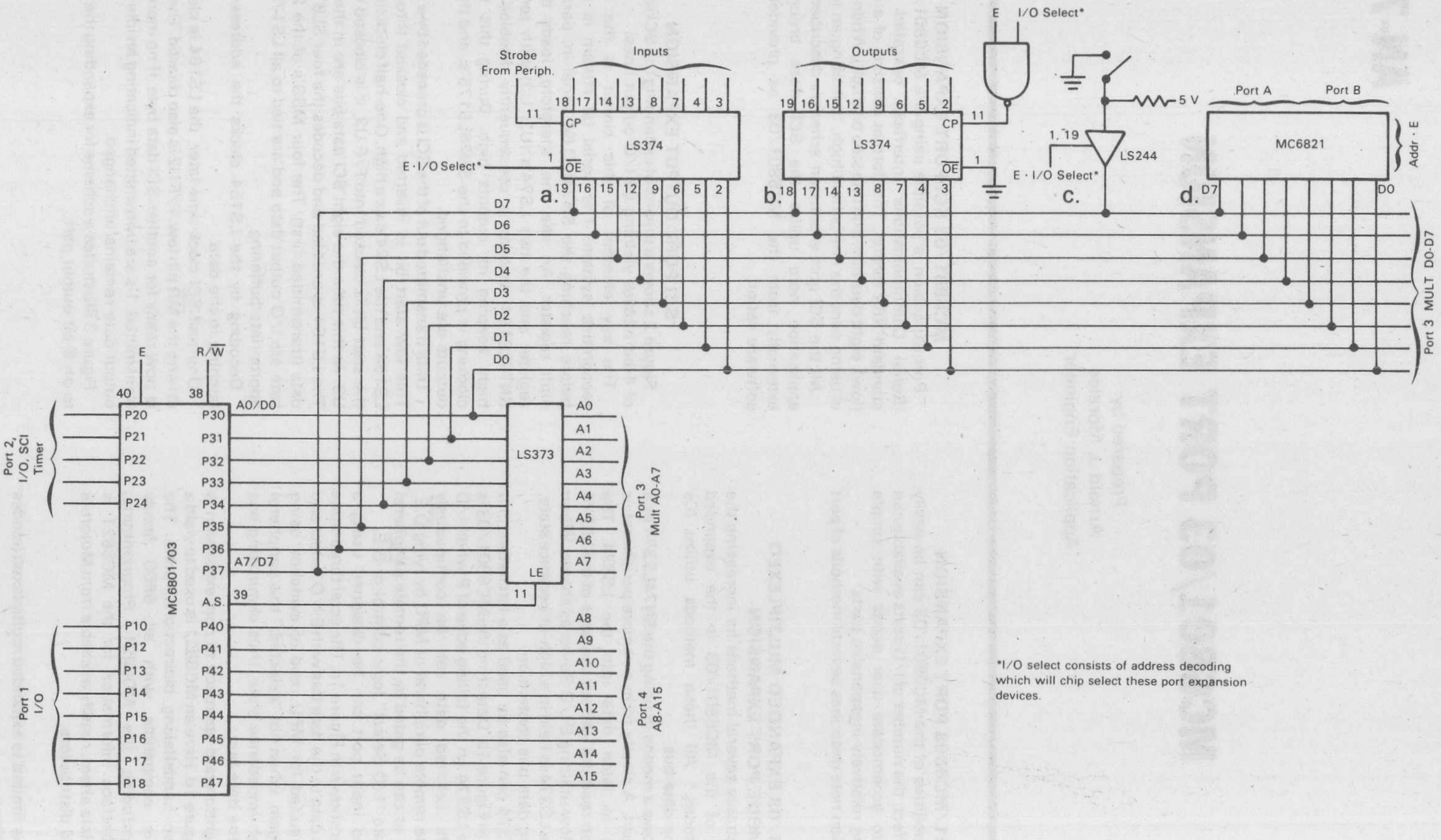


FIGURE 1 — MC6801/MC6803 Expanded Multiplexed Mode Port Expansion

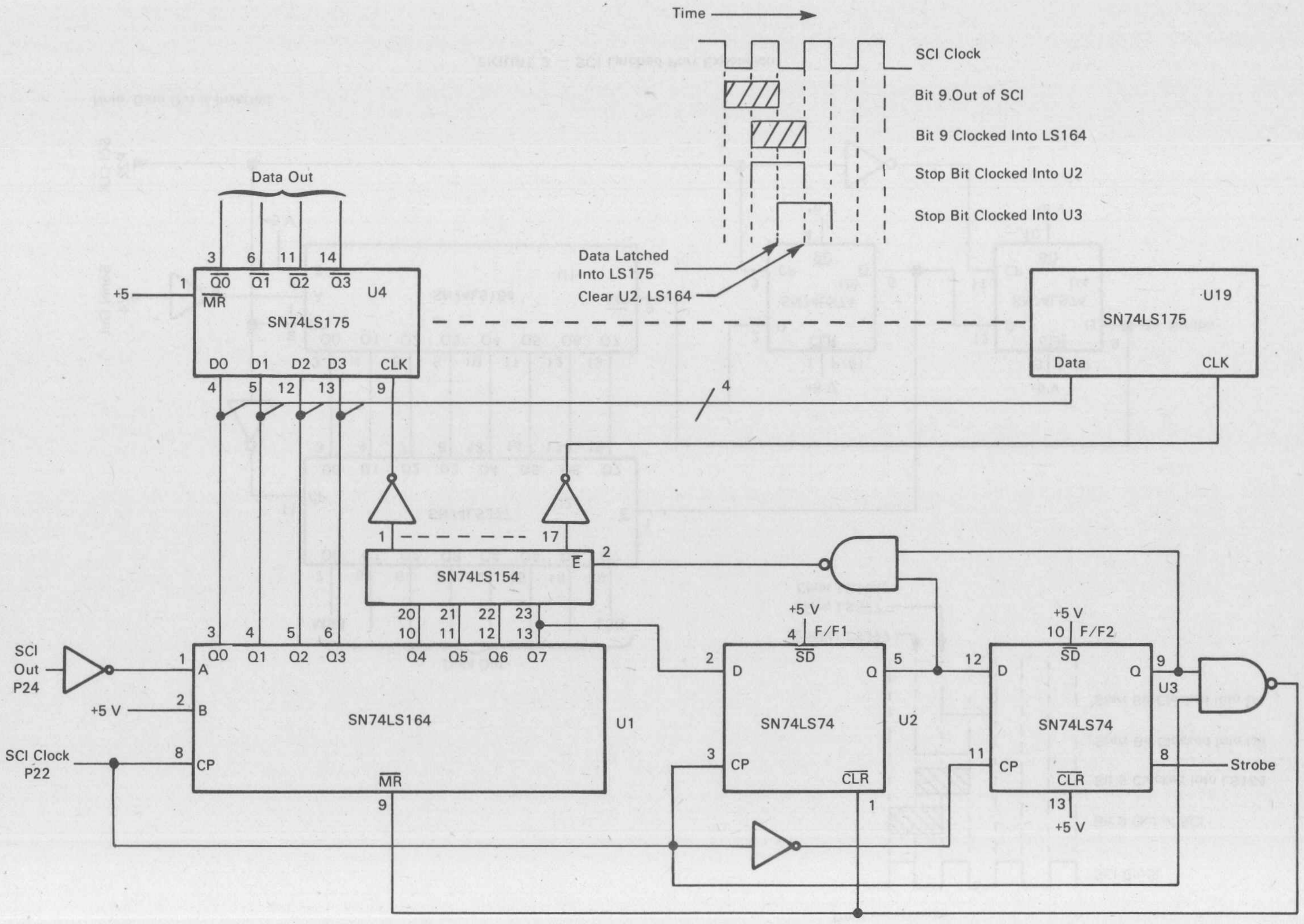
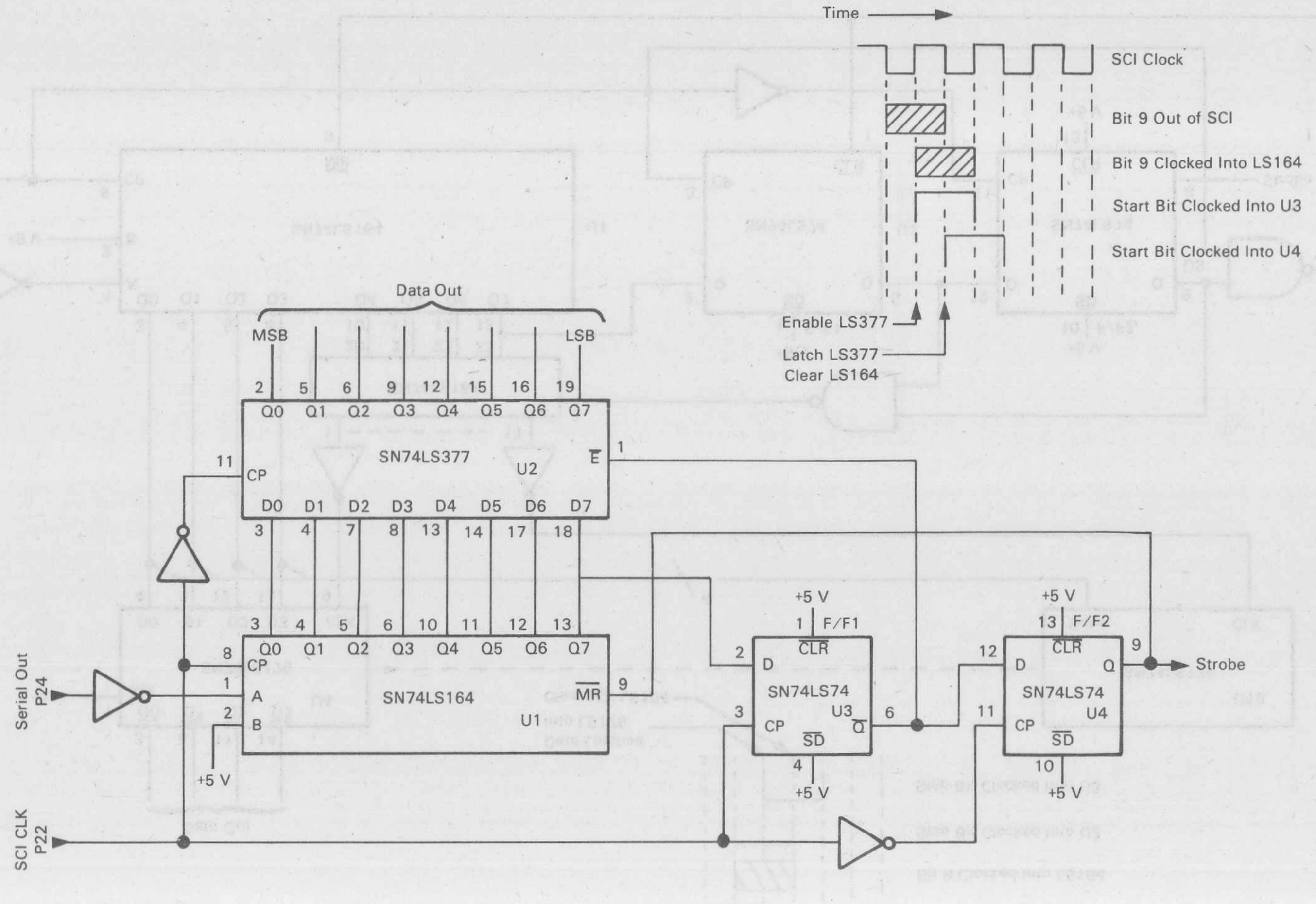


FIGURE 2 — Latched SCI Output Expansion



Note: Data Out is Inverted

FIGURE 3 — SCI Latched Port Expansion

Once again, the SCI bit stream is inverted and an idle line clocks zeros (lows) into the LS164 and LS74's. In this idle state the SN74LS377 octal "D" F/F is disabled with $\overline{\text{E}}$ high so no new data can be latched in, and the $\overline{\text{MR}}$ pin of the LS164 is held high, enabling clocking of serial input data.

When SCI data is transmitted the start bit is inverted and clocked through the LS164 as a one (high). When the start bit is clocked into the first F/F (U3), the LS377 is enabled. One-half SCI clock time later, the eight data bits in the LS164 are latched into the LS377. At the same time the start bit is clocked into the second F/F (U4) and its Q output, with appropriate propagation delay, goes low, pulling LS164 $\overline{\text{MR}}$ low, thus resetting it to all zeros. At this time the system is ready for a new SCI data byte. If the SCI line goes idle (no new SCI data bytes), the LS377 output remains unchanged.

The output of the LS377 is inverted. Non-inverted outputs can be effected by simply complementing data for MC6801/03 SCI transmission, using the appropriate "complement" op code.

SCI PORT INPUT EXPANSION

Figure 4 illustrates a parallel-to-serial interface, designed to input keyboard ASCII characters and clock the data serially into the MC6801/03 SCI port.

The SN74LS165 "serial in" line is tied high so that during an idle period (no keyboard data) 1's are clocked through the LS165 and F/F U4 into the SCI. The SCI thus sees an idle line. F/F U1 and F/F U2 are cleared at this time.

When a key is punched the keyboard strobe clocks a high into F/F U1. This high is clocked into F/F U2 on the falling edge of the SCI clock. When the SCI clock next goes high, F/FU1 and F/FU4 are cleared and LS165 (U3) $\overline{\text{P/L}}$ is driven low, latching the keyboard data. The output of F/F U4, a low, is the start bit. U2 is driven low on the next high-to-low SCI clock transition.

Data is now clocked into the SCI by the SCI clock. Ones (highs) are clocked into the SCI after the eight data bits are clocked in, indicating an idle line. At this time the interface is ready for more data.

Care must be taken to insure that "repeat" characters are not sent by the keyboard while characters are being clocked into the SCI.

MC6801 PORT 3 OUTPUT EXPANSION WITH HANDSHAKING

Port 3 of the MC6801 operated in the single chip mode can be easily expanded using SN74LS377 octal D flip-flops. Figure 5 illustrates one method of expanding port 3 to two 8-bit output ports with handshaking.

The "D" inputs of each LS377 are tied to port 3. Port 4 bit 0 (P40) is used to select one of the LS377s and deselect the other by controlling the respective $\overline{\text{E}}$ inputs.

When data is written to port 3, the port 3 strobe (OS3) clocks both LS377's. Only the selected LS377, however, will latch in the data. Nand gates are used to generate the appropriate strobe, OS3A or OS3B.

Further expansion is possible using two or more port 4 outputs for LS377 selection. Software initialization must include setting bit 4 in the Port 3 Control/Status register ($0F), so that the OS3 strobe is generated by a write to port 3.

MC6801 PORT 3 INPUT EXPANSION WITH HANDSHAKING

Input expansion of port 3 is illustrated in Figure 6 with the MC6801 operated in the single chip mode.

Initially, F/F U3 and U4 are set. When data is strobed into one of the LS374's, one of the LS74 flip-flops is also strobed, clocking its Q output low. When either LS374 is strobed, the $\overline{\text{IS3}}$ pin of the MC6801 is driven low, setting the IS3 flag in the Port 3 Control/Status Register. Setting the IS3 IRQ1 Enable bit in the Port 3 Control/Status Register enables the IS3 flag to generate an IRQ1 interrupt, vectoring the MPU to a routine for servicing port 3.

When data is strobed into port A, F/F U4 is clocked and its Q output goes low, enabling the output of U1 and making the output of U2 high impedance. When data is strobed into port B the Q output of F/FU4 remains high, enabling the output of U2 and making the output of U1 a high impedance. The Q output of F/FU4 is also tied to port 4 bit 0, programmed as an I/O input.

The MC6801 software responds to the IS3 flag by polling port 4 bit 0 to determine which port has data. The software then reads port 3 data, thus generating an OS3 strobe which sets F/F U3 and U4. The system is now ready for new data.

Software initilization must include initializing the Port 3 Control/Status Register ($0F). The IS3 interrupt enable bit (bit 6) may be set if desired. The Output Strobe Select bit (bit 4), cleared during reset, must remain cleared so that the OS3 strobe is generated by an MPU read of port 3. The latch enable bit (bit 3), also cleared during reset, must remain cleared so the port 3 latches remain transparent. Port 3 latching is external in this circuit.

Initialization should also include a read of port 3 to set F/F U3 and U4.

Measures must be taken to insure that data is not strobed into one port while data in another port is being processed. One approach is to inhibit peripheral writes to the port while the output of U5 is low, indicating that port 3 servicing is in progress. Further arbitration logic must be included if the possibility exists of data being strobed into both ports simultaneously.

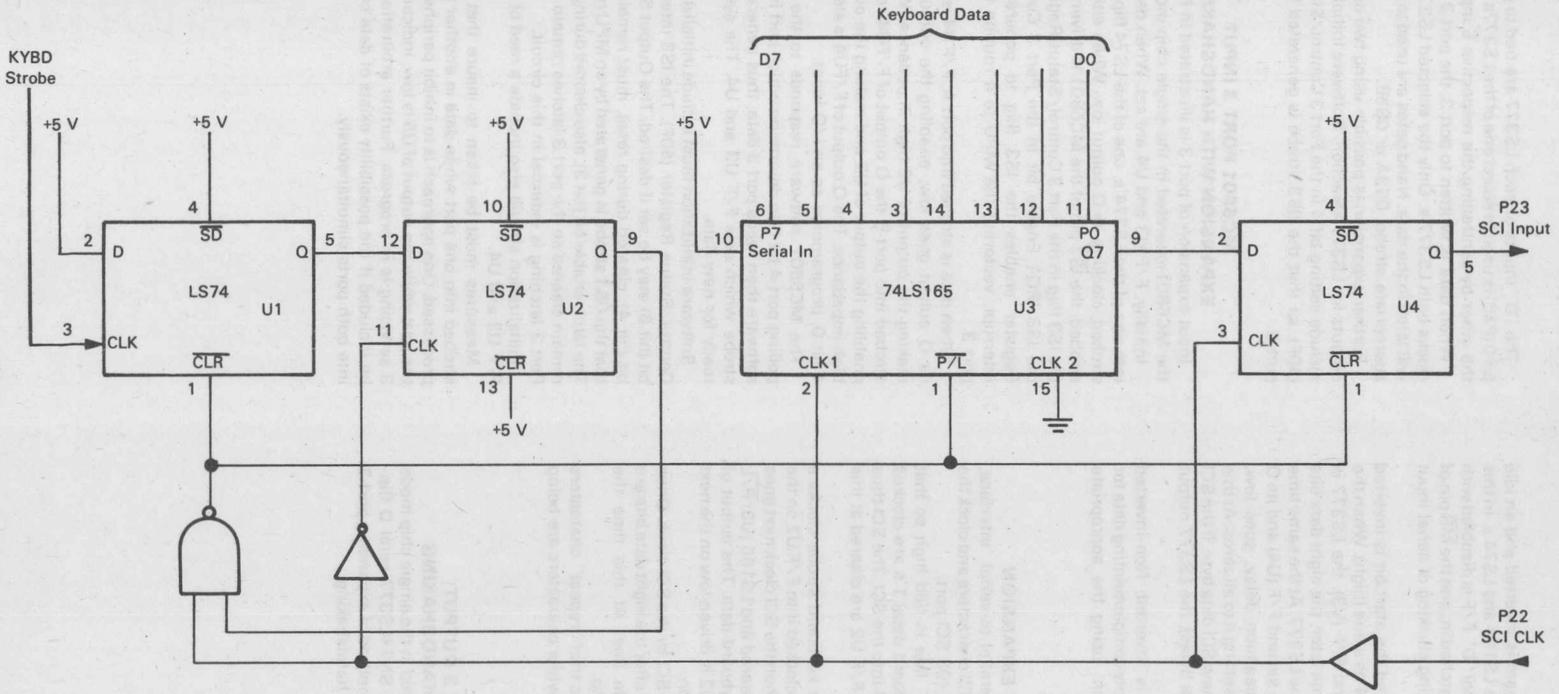


FIGURE 4 — SCI Keyboard Interface

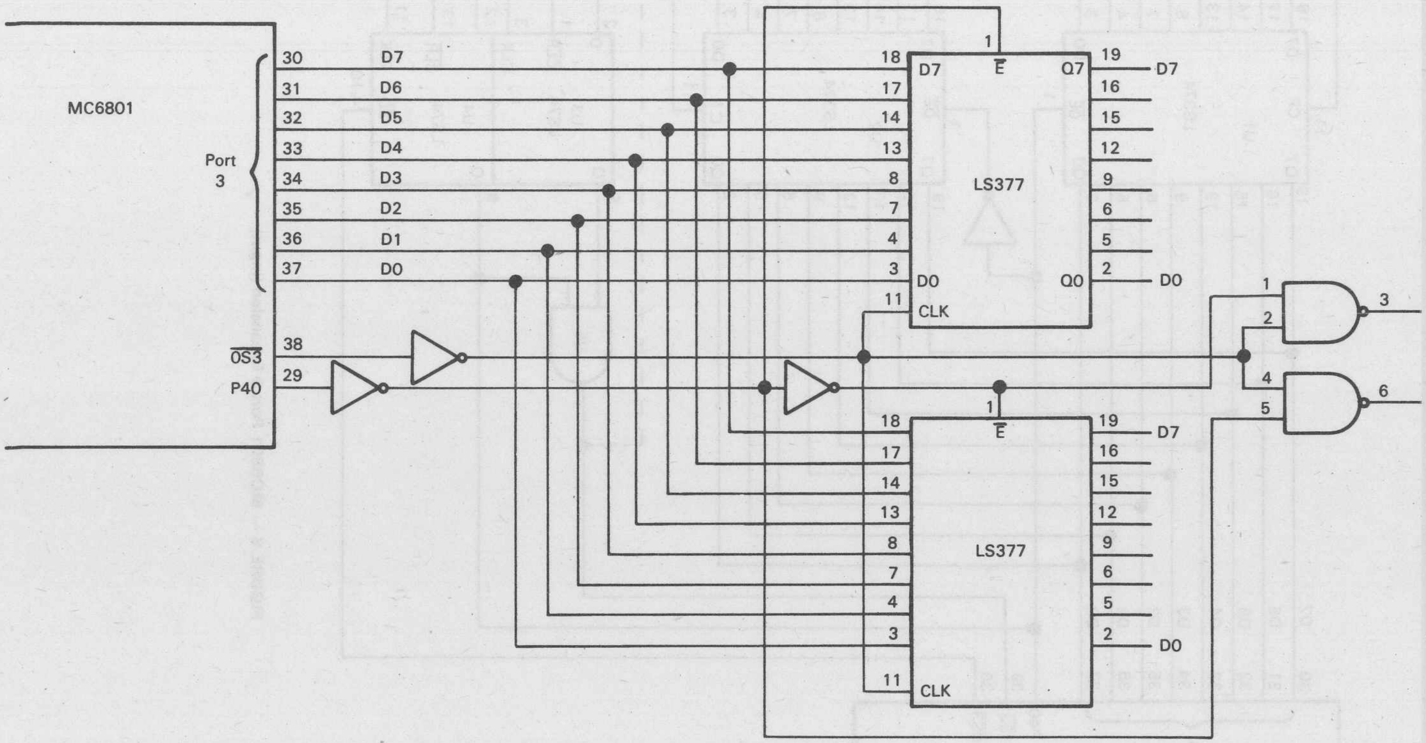


FIGURE 5 — MC6801 Port 3 Expansion (Output)

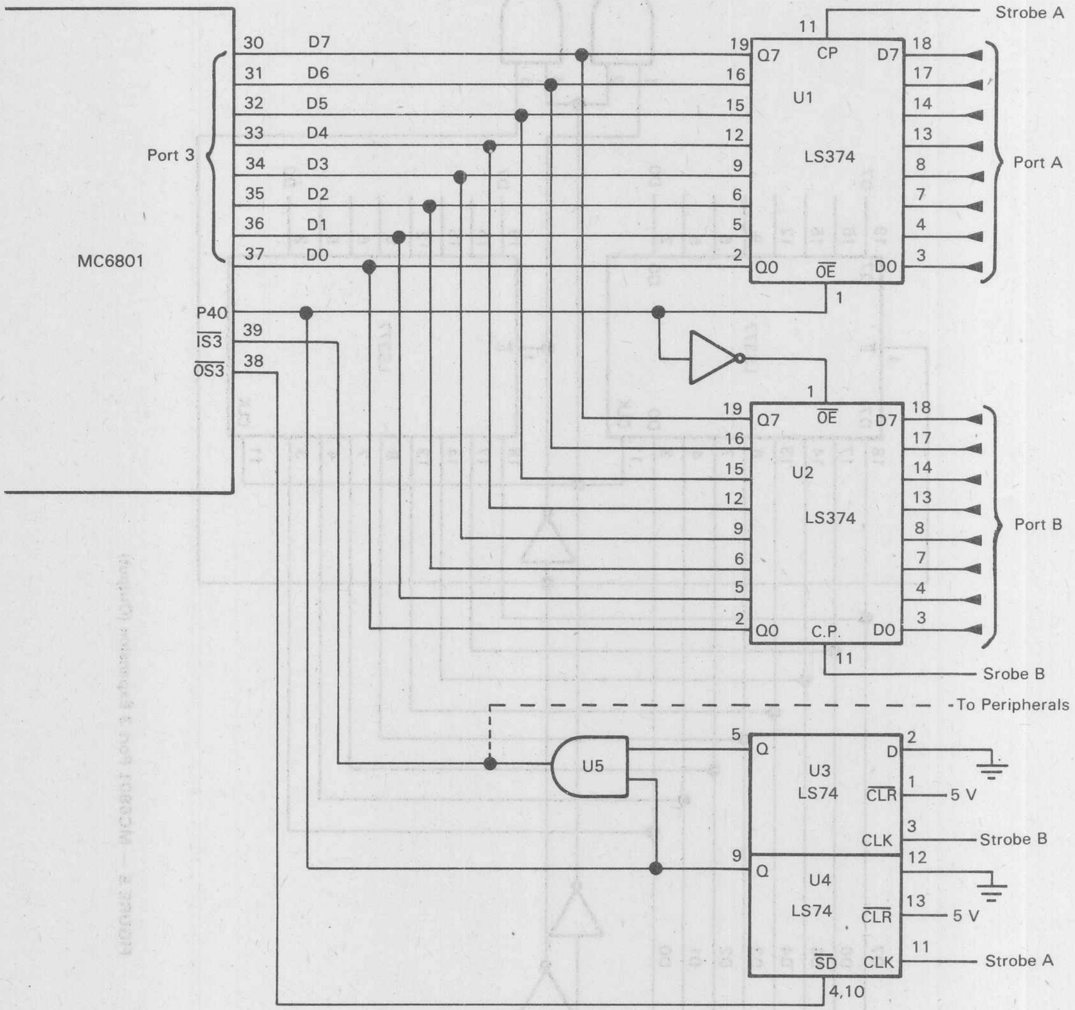


FIGURE 6 — MC6801 Port 3 Expansion (Input)

AN-798

A LOW-COST TERMINAL USING THE MC6801

Prepared by:
ARNOLD J. MORALES
Applications Engineer

An efficient low-cost terminal is now possible using an MC6801 Microcomputer, an MC6847 Video Display Generator (VDG), an MC1372 RF Modulator, and a television set.

The MC6801 is well suited for application as a terminal controller. Its four ports, on-chip programmable timer, ROM, RAM, and SCI (Serial Communications Interface) comprise all the essentials for controlling a terminal.

The four ports allow for easy keyboard and mode-select switch interface while still retaining full 64K byte address capability. The programmable timer is well-suited for cursor and bell timing, and the SCI needs only RS-232 interface buffers for use by the terminal. The on-board ROM is 2 K bytes, allowing plenty of "software room" beyond the 500 bytes necessary for basic terminal operation. The on-chip RAM is 128 bytes, suitable for program stack and scratch-pad memory.

The MC6847 VDG and the MC1372 RF Modulator make possible the use of a conventional, unmodified television as a monitor. The MC1372 interfaces directly with the television 75 ohm antenna terminals.

These Motorola devices form the nucleus of the terminal described in this application note. The terminal uses a total of only 15 devices. This number can be reduced to 13 by using transistors for the RS-232 interface.

DESCRIPTION

The terminal provides the user with the choice of two data formats: Industry Standard Mark/Space (NRZ)* and Bi-phase. As shown in Figure 1, each format consists of a start bit (low), eight data bits, and one stop bit (high).

Most users prefer the NRZ format which is a universal standard. The Bi-phase format, however, has advantages that will be useful in many applications; it is more immune to noise and can tolerate a bit-rate drift of up to 25%.

Four Baud rates per MPU operating frequency are software selectable, as shown in Figure 2. Baud rates not listed can be generated by selecting a crystal to give the desired Baud rate or by using an external clock to drive the SCI, an MC6801 option.

Baud rate, format, and SCI clock source options are controlled by the Rate and Mode Control Register in the MC6801, as shown in Figure 2. For simplicity, user control of this register is determined by a set of four DIP switches (S0-S3) which are set by the user in the same format as called for by the register. Other options, paging or scrolling and full or half duplex, are also selected by DIP switches (S4-S5). These switches are continually polled by software and can be changed "on the fly."

The terminal recognizes most widely-used control characters, including:

Line Feed (LF)
Carriage Return (CR)
Backspace (BS)
Bell (BEL)
Clear Screen (SUB)
Cursor Forward (FF)
Cursor Up (VT)

The cursor blinks and is non-destructive; that is, it can be moved over any displayed character without changing the character. The cursor will simply blink over the character, with the character appearing each time the cursor is "off."

The display format is controlled by the MC6847 VDG which displays a complement of 64 6-bit ASCII characters in a 32 (across) by 16 (down) format.

*Non-Return to Zero

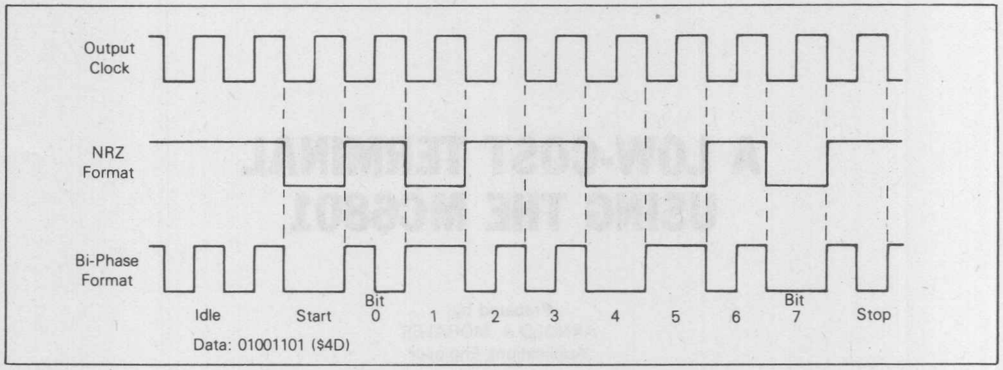


Figure 1. SCI Data Formats

7	6	5	4	3	2	1	0	
X	X	X	X	CC1	CC0	SS1	SS0	ADDR:$0010

Rate and Mode Control Register (RMCR)

	$4f_O \rightarrow$	2.4576 MHz	4.0 MHz	4.9152 MHz
SS1:SS0	E	614.4 kHz	1.0 MHz	1.2288 MHz
0 0	÷ 16	26 μs/38,400 Baud	16 μs/62,500 Baud	13.0 μs/76,800 Baud
0 1	÷ 128	208 μs/4,800 Baud	128 μs/7812.5 Baud	104.2 μs/9,600 Baud
1 0	÷ 1024	1.67 ms/600 Baud	1.024 ms/976.6 Baud	833.3 μs/1,200 Baud
1 1	÷ 4096	6.67 ms/150 Baud	4.096 ms/244.1 Baud	3.33 ms/300 Baud

SCI Bit Times and Rates

CC1:CC0	Format	Clock Source	Port 2 Bit 2
00	Bi-Phase	Internal	Not Used
01	NRZ	Internal	Not Used
10	NRZ	Internal	Output
11	NRZ	External	Input

SCI Format and Clock Source Control

Figure 2. SCI Baud Rate and Format Options

ASSEMBLY OF THE TERMINAL

The MC6801 can be configured to operate in three basic modes: single chip mode (single chip microcomputer), expanded non-multiplexed mode (256 byte external address capability), and expanded multiplexed mode (64K byte external address capability). Several expanded multiplexed modes give the user choices of combinations of on-chip or external RAM, ROM, and interrupt vectors. Table 1 shows a summary of these operating modes.

The terminal presented in this article uses an MC6801 operated in Mode 2. This mode configures the MCU for 128 bytes of on-chip RAM, external ROM, external interrupt vectors, and uses an MCM2708 EPROM for easy software development. Fully developed software can be "masked" into the MC6801 on-chip ROM. The MCU can then be operated in expanded multiplexed Mode 6, which uses the on-chip ROM and RAM, with only an MC6801 mode select design change. As an alternative, the MC68701, with its on-chip 2K EPROM, can be used for development. An MC6803 can be used instead of an MC6801 in terminals using an external EPROM or ROM.

The MC6801 has four ports. In the expanded multiplexed modes, Port 1 is used for general I/O, Port 2 for mode selection and SCI/Timer I/O or for general I/O, Port 3 for multiplexed data and lower order addresses (A0/D0-A7/D7), and Port 4 for higher order addresses (A8-A15).

Port 2 has only five pins associated with it, as shown in the schematic of Figure 3. Pins 8 (P20), 9 (P21), and 10 (P22) of Port 2 are dual purpose pins. On the rising edge of $\overline{\text{RESET}}$, the MC6801 latches the logical state of these pins as the upper three bits of the Port 2 register. These are read-only bits, and have no pins directly associated with them. They are the "mode control" bits and their states determine the operating mode of the MCU. The mode select voltages are applied to pins 8, 9, and 10 through pull-up resistors so that immediately after $\overline{\text{RESET}}$, pin 8 can be used for the Timer input, pin 9 for the Timer output, and pin 10 for the SCI clock input (if used).

Pins 11 and 12 of Port 2 are used by the SCI. MC1488 and MC1489 line drivers are used as the RS-232 interface. The onboard SCI controls all serial communications, serving the function of a UART.

Seven pins of I/O Port 1 are used for keyboard ASCII input. The Timer is used to latch the keyboard strobe.

The keyboard must generate a strobe at least 2 MCU cycles in duration with each keyboard entry. This strobe is tied to pin 10, the Timer input. An input edge detector tied to this pin sets a flag (Input Capture Flag) in the MCU Timer Control/Status Register each time a key is depressed. The software polls this flag for keyboard servicing.

A simple bell circuitry using two NAND gates is used to generate a 4 kHz tone. Software gates the bell using the MCU Timer output, pin 9.

The terminal operating mode switches are read through an MC74LS244 octal three-state driver at address $3800. A simple switch and pull-up arrangement is used for mode selection.

The display circuitry consists of the MC6847 Video Display Generator, the MC1372 Modulator, two MCM2114 RAMs, and associated circuitry. The VDG is operated in the Alphanumeric Internal mode to display characters in a 32 (across) by 16 (down) format and in the Semigraphics 4 mode to display the cursor. Both interlaced (MC6847Y) and non-interlaced (MC6847) versions of the VDG are available.

The VDG is clocked by the MC1372 at a 3.58 MHz colorburst frequency. It reads data sequentially from display RAM and, in this terminal, uses an on-chip character generator to produce the alphanumeric displays. Two MCM2114s are used as the display RAM. Only 512 bytes of display RAM are needed for the Alphanumeric and Semigraphic 4 modes, so only nine VDG address lines are used. A tenth address line (A9), held high by the VDG, is connected to the MCM2114s to ensure proper address decoding.

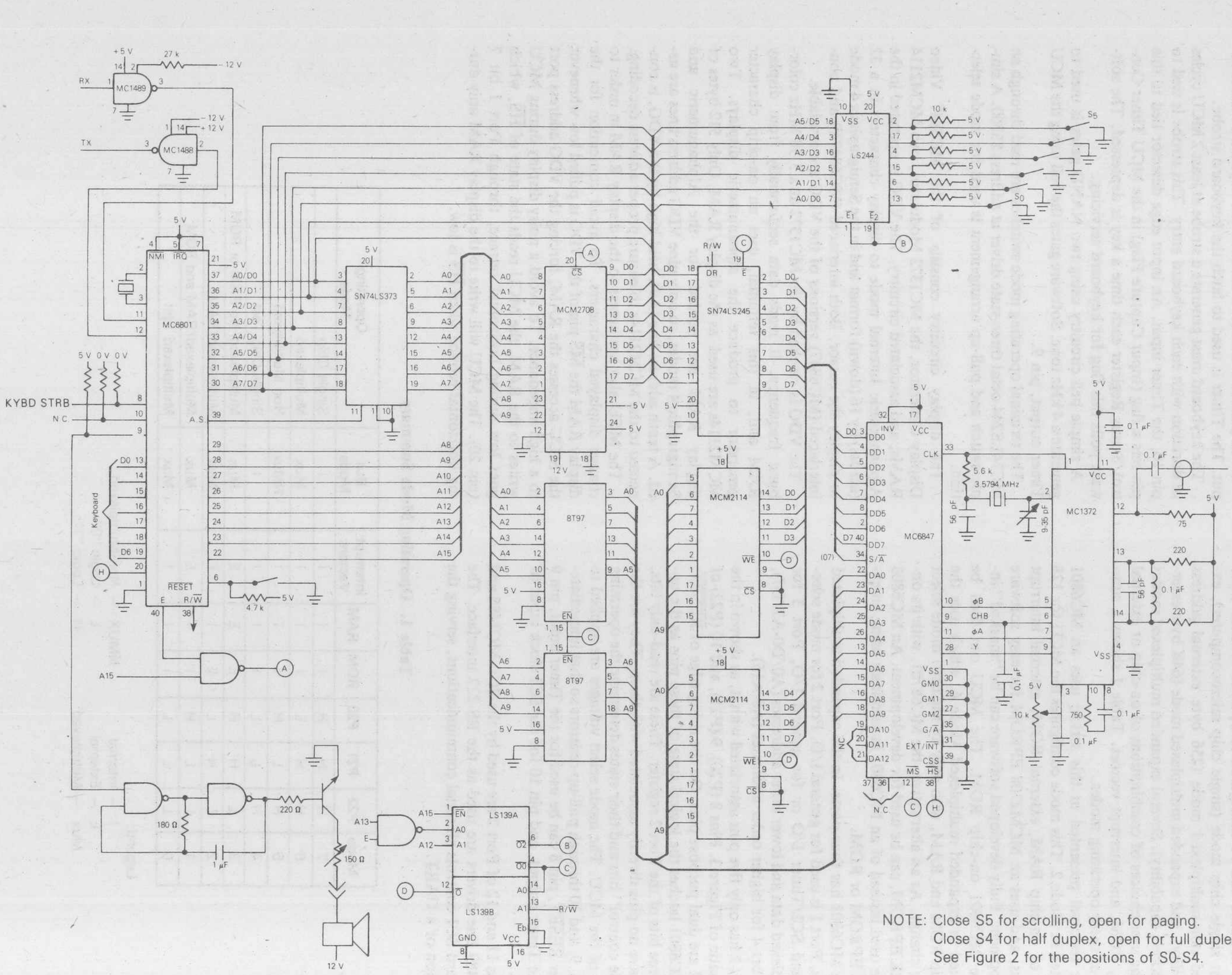
The MCU also has access to the display RAM in order to change displayed characters. To avoid contention for the display RAM the $\overline{\text{MS}}$ pin of the VDG is pulled low whenever the MCU accesses the RAM, forcing the VDG address port to a high impedance. To avoid a noisy display during MCU writes to the RAM, the MCU reads the state of $\overline{\text{HS}}$, which goes low during horizontal retrace, through Port 1 bit 7 (pin 20). The MCU will write to the display RAM only during horizontal retrace when $\overline{\text{HS}}$ is low.

Table 1. Operating Mode Summary

Mode	P22	P21	P20	ROM	RAM	Interrupt Vectors	Bus Mode	Operating Mode
7	H	H	H	I	I	I	I	Single Chip
6	H	H	L	I	I	I	Mux	Multiplexed
5	H	L	H	I	I	I	NMux	Non Multiplexed
4	H	L	L	I	I	I	I	Single Chip Test
3	L	H	H	E	E	E	Mux	Multiplexed/No RAM or ROM
2	L	H	L	E	I	E	Mux	Multiplexed/RAM
1	L	L	H	I	I	E	Mux	Multiplexed/RAM and ROM
0	L	L	L	I	I	I	Mux	Multiplexed Test

Legend:

I – Internal
E – External
Mux – Multiplexed
NMUX – Non-Multiplexed
L – Logic "0"
H – Logic "1"



NOTE: Close S5 for scrolling, open for paging.
Close S4 for half duplex, open for full duplex.
See Figure 2 for the positions of S0-S4.

Figure 3. Schematic

The VDG outputs chrominance (ϕA, ϕB, Y) and chroma bias (CHB) to the MC1372 Modulator which generates composite video for the television. The Modulator is clocked by a 3.58 MHz crystal and its TTL compatible clock output (pin 1) is used to drive the VDG.

C1 is used to adjust the clock frequency; R1 is used to adjust the clock duty cycle.

SOFTWARE

The software initializes the MC6801, services characters from the RS-232 communication link and from the keyboard, controls character display, and controls the bell.

INITIALIZATION

The software first configures Port 1, used to read the keyboard, as a data input port by clearing the Port 1 Data Direction Register. It then clears the Timer Control/Status Register. This disables all Timer interrupts and programs the Timer Input Capture Flag to become set whenever a high-to-low transition is applied to the Timer input pin. This pin is tied to the keyboard strobe.

The software then enables the SCI transmitter and receiver and programs Port 2 for Timer and SCI I/O. It then reads the terminal mode switches and jumps to subroutine CHGWO, which loads the mode word into STATWO, a scratch-pad register, for later comparison.

The display RAM consists of 512 bytes located in addresses $2000 through $21FF. For scrolling purposes the software loads display RAM locations $2200 through $2220 with ASCII "blanks." It then jumps to subroutine BLANK, which clears the screen, then loads the index register with $2000. The index register is used as a screen pointer and $2000 is the first screen location.

MAIN PROGRAM

The main program is essentially a loop which writes the cursor, branches to subroutine TIMER which controls the terminal, erases the cursor, then branches back to TIMER.

Since the terminal interprets only 6-bit ASCII, data bit seven in the display RAM is left free for use as a VDG control bit and is used by this terminal to control $S/\overline{A}$ for displaying the cursor. When $S/\overline{A}$ is low the VDG is in the Alphanumeric Internal mode and will display ASCII characters. When $S/\overline{A}$ is high the VDG is in the Semigraphics 4 mode and will display a color block in one of eight colors. The software used in this terminal selects a green cursor by writing $80 into the display RAM location pointed to by the index register.

Subroutine TIMER first checks for changes in the terminal operating mode by jumping to subroutine CHKSTA. CHKSTA reads the switches and compares their settings to the last setting stored in register STATWO. If switch selections have changed, CHKSTA will load a new value into STATWO and reprogram the SCI. Otherwise, TIMER continues by loading register TEMPX with a value ($7FF) which controls cursor duration. This value will be decremented to zero at which time the cursor will be removed. With each decrement the keyboard and SCI are serviced by subroutines CHKC and SERRX.

The cursor is removed by replacing it with the contents of SAVCHR, a register that stores the character in the location pointed to by the index register (screen pointer). TIMER is used once again to provide delay.

Subroutine CHKC services the keyboard by first checking the Timer Input Capture Flag for the presence of a keyboard strobe. If the flag is not set, the program returns to subroutine TIMER. If the flag is set, CHKC clears the flag and transmits the keyboard character out of the serial port. CHKC then reads STATWO and tests for full duplex selection. If the mode is half duplex, the character is displayed by subroutine DISPL. If the mode is full duplex, no character is displayed at this time but will be displayed by subroutine SERRX, which services the SCI.

CHKC then jumps to subroutine ENDSCN to test for end of screen and pages or scrolls if necessary according to the user mode selection.

Subroutine SERRX services SCI input characters by testing the Receiver Data Register Full flag in the Transmit/Receive Control and Status Register. If no character is present, the program returns to TIMER. If a character is present, it is displayed by DISPL. The program then jumps to ENDSCN to test for end of screen, then returns to TIMER.

Figure 4, the program flowchart, offers a detailed outline of the program. Figure 5 contains the program listing.

FURTHER DEVELOPMENT OF THE TERMINAL

The terminal can be further developed to meet many user requirements with few hardware and software changes. Two improvements are particularly worth considering: interrupt drive and graphics capability.

INTERRUPT DRIVEN TERMINAL

A completely interrupt-driven terminal is possible with very few changes in the software. Polling of the Timer Input Capture Flag is used to detect the presence of keyboard characters. Polling of the SCI Receiver Data Register Full Flag is used to detect the presence of characters received from the serial link. The MC6801 can be programmed to generate a vectored interrupt when each of these flags is set, eliminating the need for polling.

The input capture interrupt is enabled by setting bit 4 in the Timer Control and Status Register. The vector for this interrupt is at $FFF6. The SCI interrupt is enabled by setting bit 4 in the Transmit/Receive Control and Status Register. This interrupt vector is at $FFF0.

Polling of the mode-select switches can be eliminated by reading the switches once during initialization.

COLOR GRAPHICS

The capability for graphics is designed into the VDG. All that is needed is an addressable latch to control the VDG mode pins, more display RAM, and software development.

Table 2 contains a detailed description of the VDG operating modes. The column labeled "VDG PINS" lists all the VDG mode control pins. The terminal presented in this article operates only in the Alphanumeric Internal (internal character generator) and Semigraphics 4 modes. Therefore, only the pins labeled S/$\overline{A}$ and $\overline{MS}$ are used, with the other control pins tied to ground. By tying these unused pins to an addressable latch under software control the terminal can be placed in the graphics modes, and the alphanumeric displays can be inverted.

The column labeled "COMMENTS" gives a description of each mode. Of particular importance in these descriptions is the RAM necessary for each of these modes. As the display density increases (color control of smaller areas), the RAM required increases. The densest modes (Color Graphics Six and Resolution Graphics Six) require 6 K bytes of RAM.

Data in the display RAM controls the display. In the alphanumeric modes the data is interpreted by the VDG as six-bit ASCII code. In the graphics modes the data controls the color of each display element. The VDG can display up to eight colors, excluding black. However, the number of colors selectable varies according to the mode of operation as described in Table 2.

The VDG operating modes can be changed "on the fly" if the mode is changed during $\overline{HS}$ on the proper line count. Therefore, displays combining graphics and alphanumerics are relatively easy. A screen line counter, an addressable latch for VDG mode control, and software are all that is required.

Software for displaying charts and graphs have already been written for demonstration purposes by Motorola, and "3D" graphics (displays in perspective) are being developed.

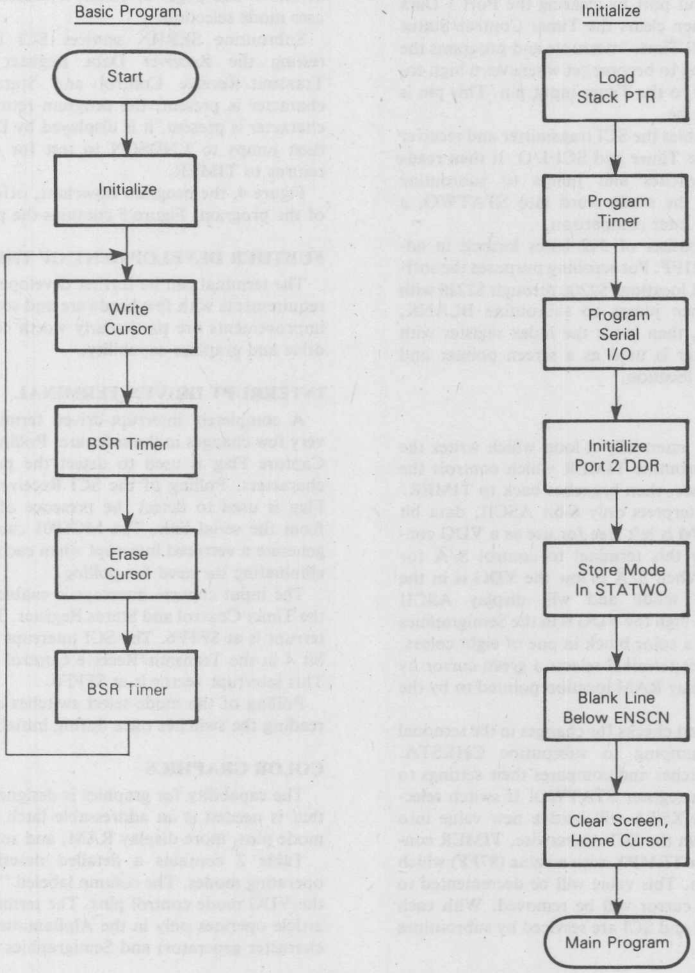


Figure 4. Program Flowchart

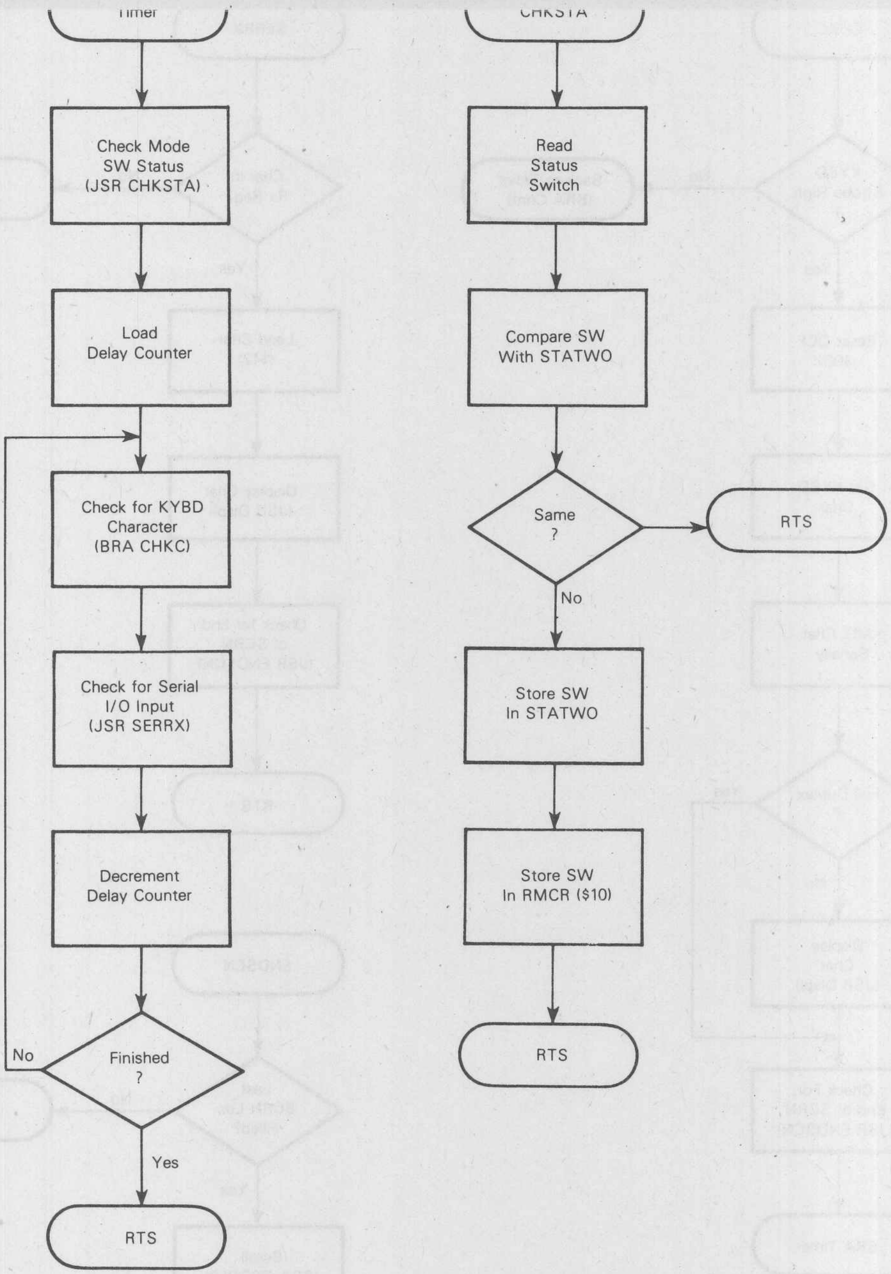


Figure 4. Program Flowchart (Continued)

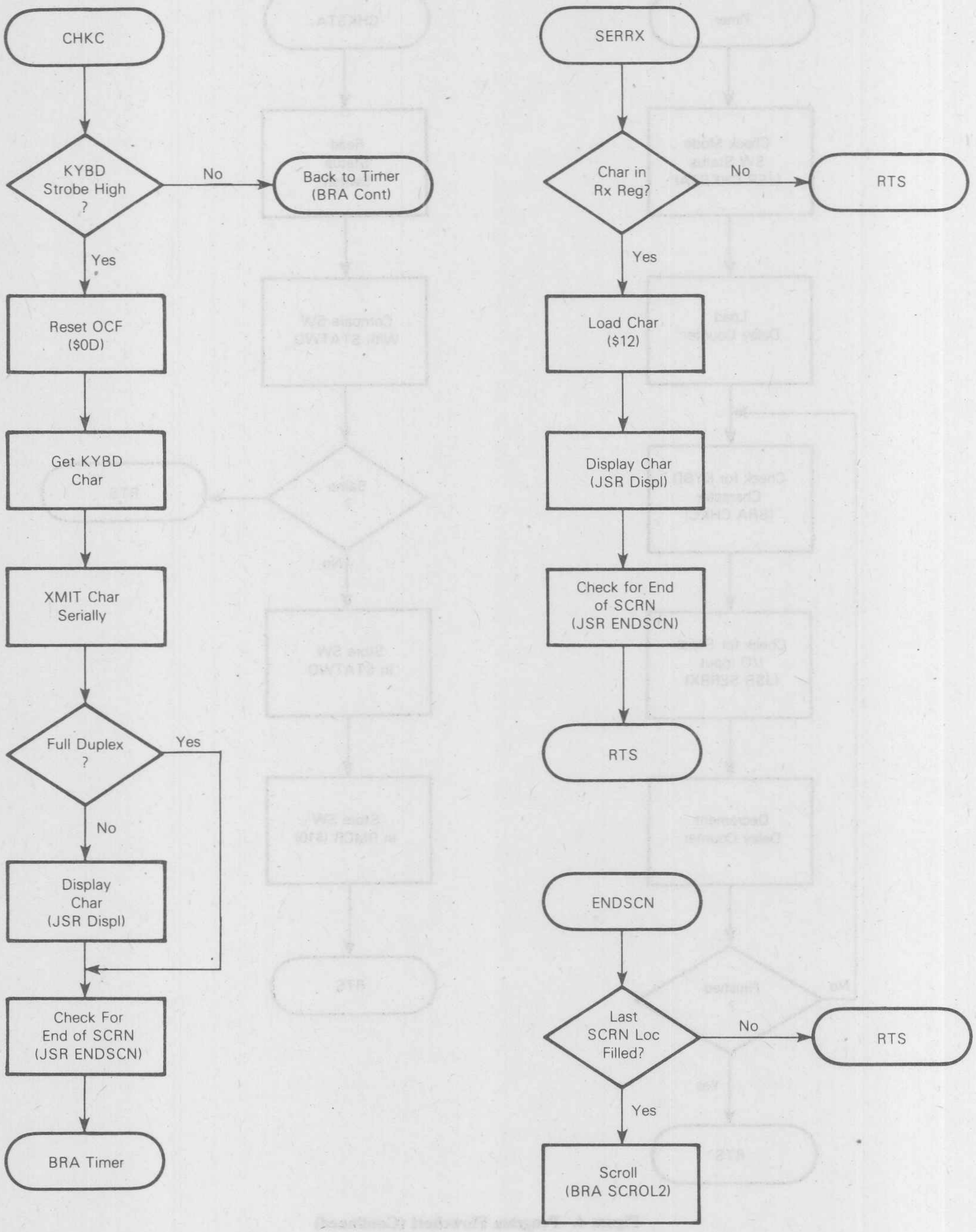


Figure 4. Program Flowchart (Continued)

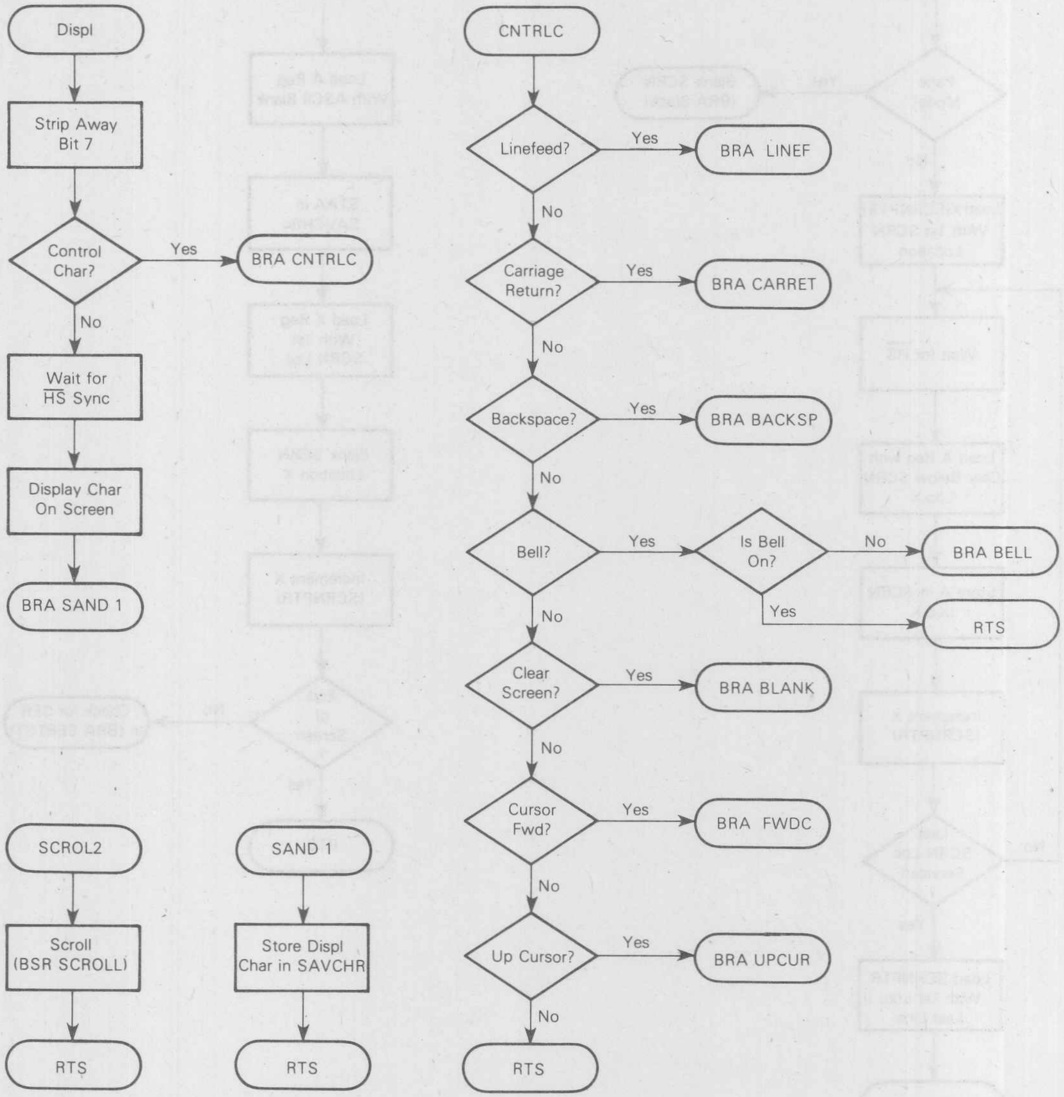


Figure 4. Program Flowchart (Continued)

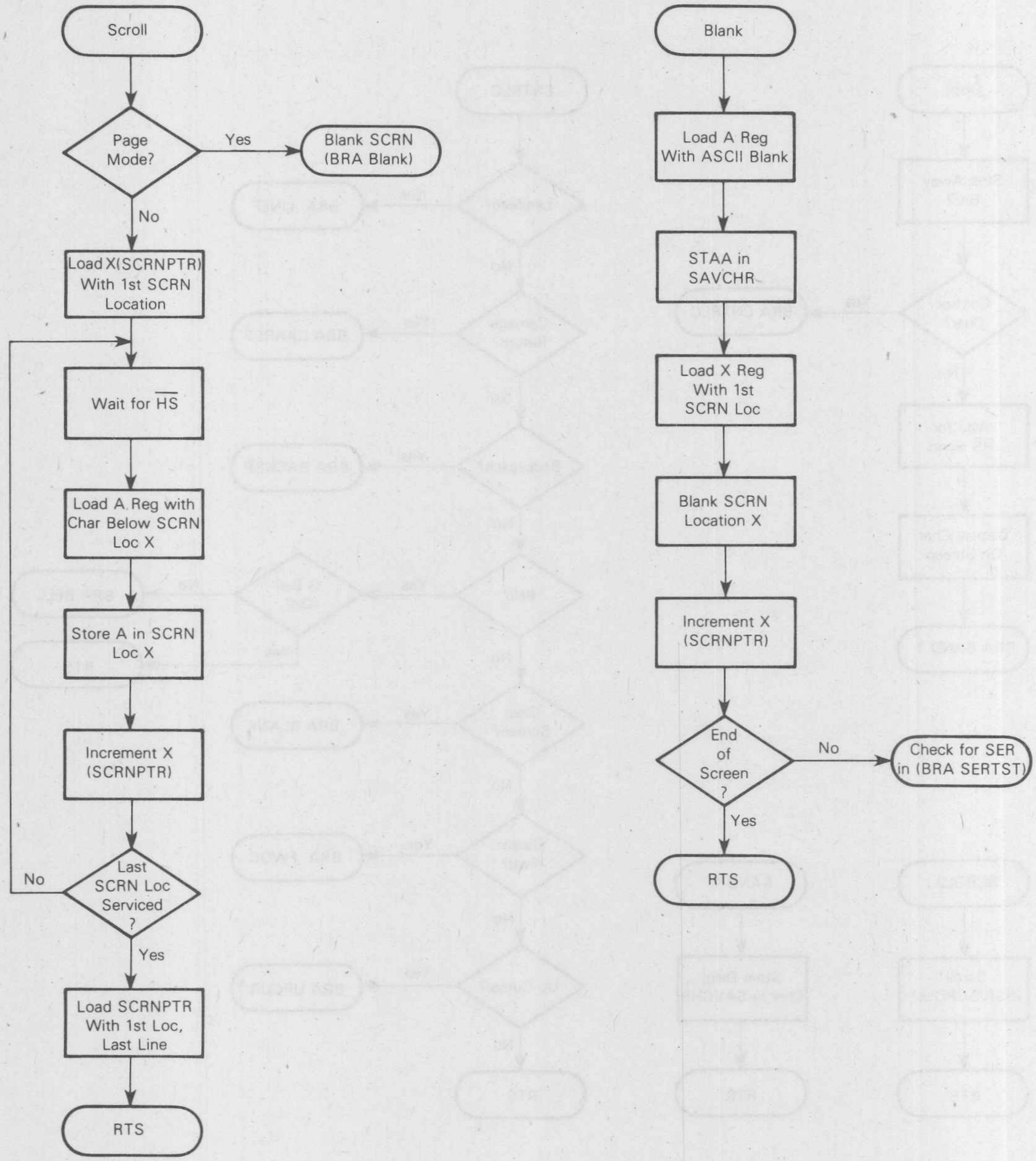


Figure 4. Program Flowchart (Continued)

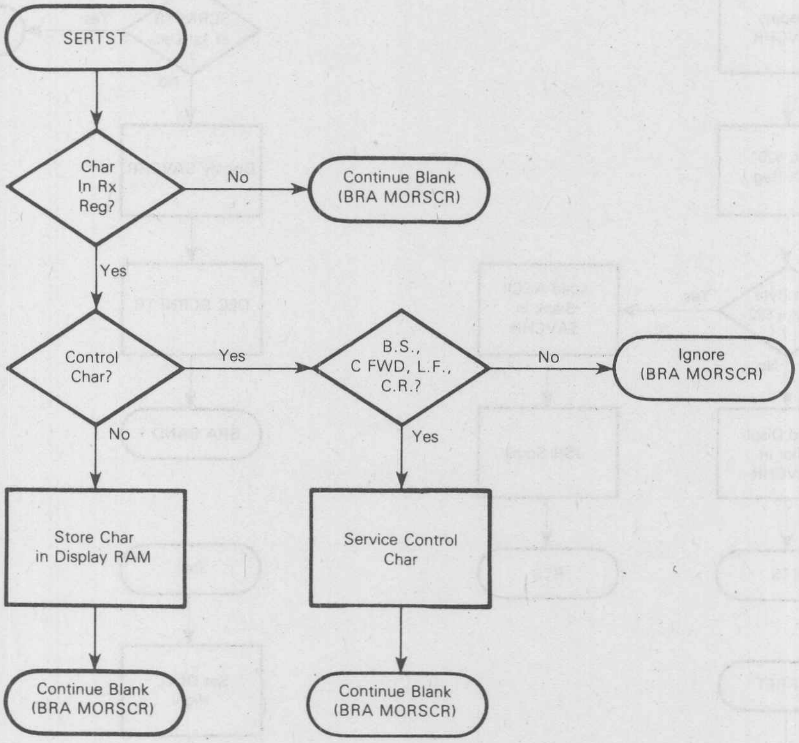


Figure 4. Program Flowchart (Continued)

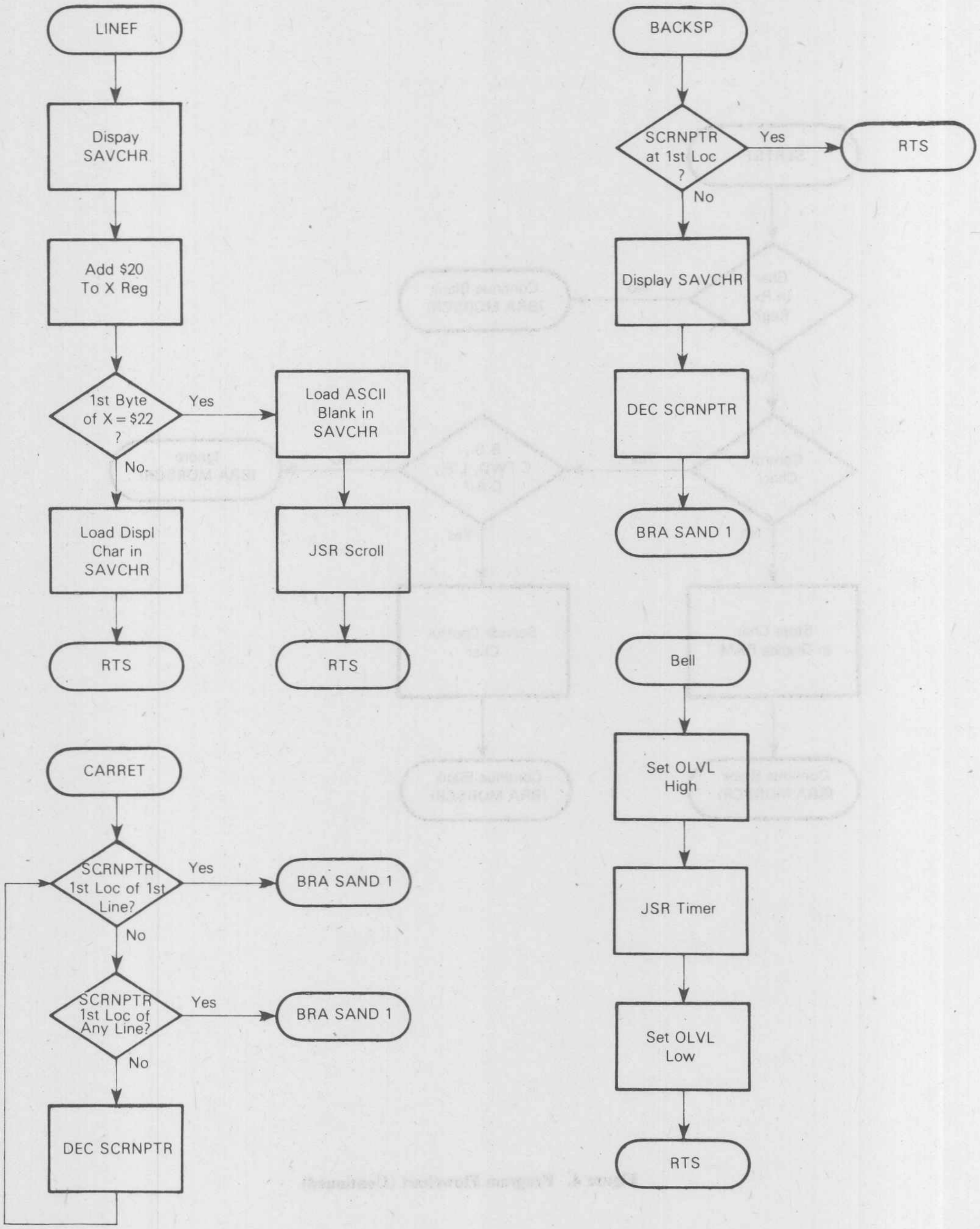


Figure 4. Program Flowchart (Continued)

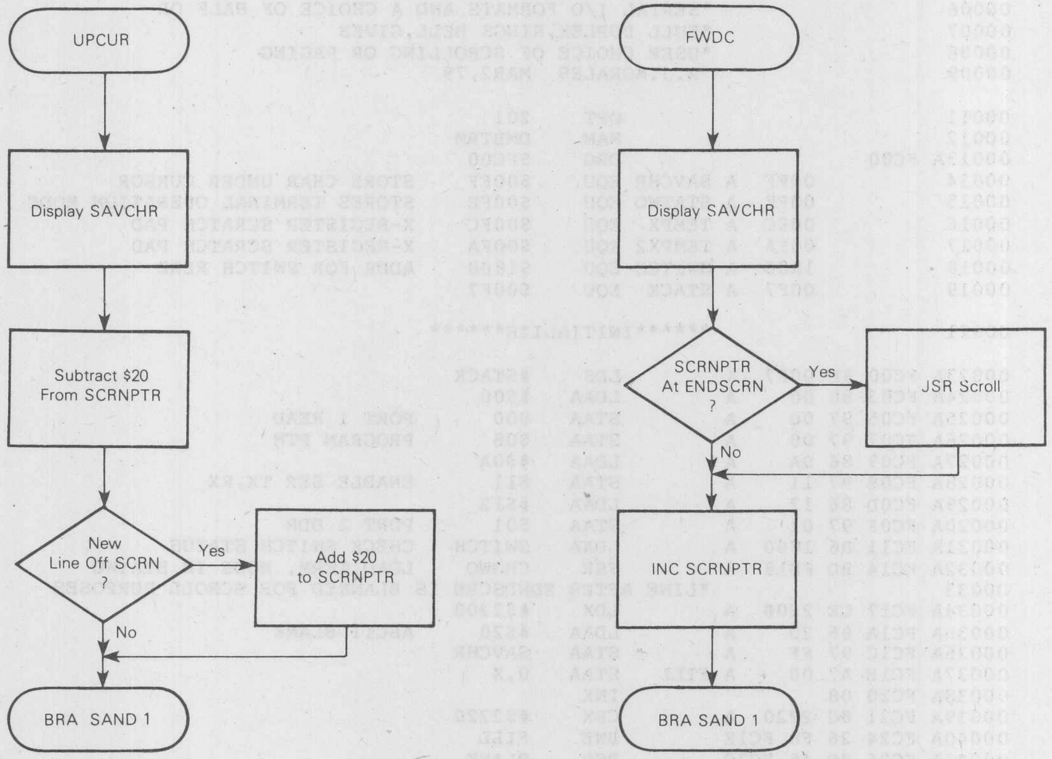


Figure 4. Program Flowchart (Concluded)

```
PAGE  001   DMBTRM

00001                      *THIS PROG DISPLAYS AND XMITS KYBD CHAR,
00002                      *RECEIVES AND DISPLAYS SER INPUT CHAR,
00003                      *SCROLLS, BLINKS CURSOR, LINEFEEDS, MOVES
00004                      *CURSOR FWD, BACKSPACES CURSOR, MOVES CURSOR UP,
00005                      *CARRIAGE RETURNS, GIVES USER CHOICE OF FOUR
00006                      *SERIAL I/O FORMATS AND A CHOICE OF HALF OR
00007                      *FULL DUPLEX,RINGS BELL,GIVES
00008                      *USER CHOICE OF SCROLLING OR PAGING
00009                      *A.J.MORALES  MAR2,79

00011                            OPT    Z01
00012                            NAM    DMBTRM
00013A FC00                      ORG    $FC00
00014         00FF  A SAVCHR EQU    $00FF    STORE CHAR UNDER CURSOR
00015         00FE  A STATWO EQU    $00FE    STORES TERMINAL OPERATION MODE
00016         00FC  A TEMPX  EQU    $00FC    X-REGISTER SCRATCH PAD
00017         00FA  A TEMPX2 EQU    $00FA    X-REGISTER SCRATCH PAD
00018         1800  A SWITCH EQU    $1800    ADDR FOR SWITCH READ
00019         00F7  A STACK  EQU    $00F7

00021                      ******INITIALIZE******

00023A FC00 8E 00F7  A           LDS    #STACK
00024A FC03 86 00    A           LDAA   #$00
00025A FC05 97 00    A           STAA   $00      PORT 1 READ
00026A FC07 97 08    A           STAA   $08      PROGRAM PTM
00027A FC09 86 0A    A           LDAA   #$0A
00028A FC0B 97 11    A           STAA   $11      ENABLE SER TX,RX
00029A FC0D 86 12    A           LDAA   #$12
00030A FC0F 97 01    A           STAA   $01      PORT 2 DDR
00031A FC11 B6 1800  A           LDAA   SWITCH   CHECK SWITCH STATUS
00032A FC14 BD FD18  A           JSR    CHGWO    LOAD TERM. MODE IN STATWO
00033                      *LINE AFTER EDNDSCRN IS BLANKED FOR SCROLL PURPOSES
00034A FC17 CE 2200  A           LDX    #$2200
00035A FC1A 86 20    A           LDAA   #$20     ASCII BLANK
00036A FC1C 97 FF    A           STAA   SAVCHR
00037A FC1E A7 00    A FILL      STAA   0,X
00038A FC20 08                   INX
00039A FC21 8C 2220  A           CPX    #$2220
00040A FC24 26 F8 FC1E           BNE    FILL
00041A FC26 8D 48 FC70           BSR    BLANK

00043                      *******MAIN PROGRAM*******

00045                      *WRITE CURSOR
00046A FC28 86 80    A WRITEC LDAA   #$80
00047A FC2A A7 00    A           STAA   0,X
00048A FC2C 8D 08 FC36           BSR    TIMER
00049                      *ERASE CURSOR
00050A FC2E 96 FF    A ERASEC LDAA   SAVCHR
00051A FC30 A7 00    A           STAA   0,X
00052A FC32 8D 02 FC36           BSR    TIMER
00053A FC34 20 F2 FC28           BRA    WRITEC
00054                      *TIMER ROUTINE PROVIDES TIMING FOR CURSOR BLINK,
00055                      *CHECKS FOR INPUT FROM KYBD AND SERIAL I/O, AND
00056                      *CHECKS FOR OPERATOR CHANGES OF STATUS.
00057A FC36 BD FD10  A TIMER  JSR    CHKSTA   CHECK FOR TRM MODE CHANGES
00058A FC39 3C                   PSHX
```

Figure 5. Program Listing

```
00059A FC3A CE 07FF  A        LDX    #$07FF   TIMER DELAY PARAMETER
00060A FC3D 20 0E FC4D MORE   BRA    CHKC     CHECK FOR INPUT FROM KYBD
00061A FC3F DF FC    A CONT   STX    TEMPX
00062A FC41 38                PULX
00063A FC42 BD FD03  A        JSR    SERRX    CHECK FOR SERIAL INPUT
00064A FC45 3C                PSHX
00065A FC46 DE FC    A        LDX    TEMPX
00066A FC48 09                DEX
00067A FC49 26 F2 FC3D        BNE    MORE     FINISHED TIMING?
00068A FC4B 38                PULX
00069A FC4C 39                RTS
00070                  *CHECK FOR INPUT FROM KYBD
00071A FC4D 96 08    A CHKC   LDAA   $08      TEST FOR STROBE VIA ICF ON PTM
00072A FC4F 2B 02 FC53        BMI    YESC
00073A FC51 20 EC FC3F        BRA    CONT     CONTINUE TIMER ROUTINE
00074                  *THIS ROUTINE SERVICES CHAR FROM KYBD
00075A FC53 38         YESC   PULX
00076A FC54 96 0D    A        LDAA   $0D      RESET ICF
00077A FC56 96 02    A        LDAA   $02      KYBD ASCII
00078A FC58 97 13    A        STAA   $13
00079A FC5A D6 FE    A        LDAB   STATWO
00080A FC5C C4 10    A        ANDB   #$10
00081A FC5E 27 03 FC63        BEQ    FULLD    TEST FOR FULL DUPLEX
00082A FC60 BD FD1D  A        JSR    DISPL
00083A FC63 8D 02 FC67 FULLD  BSR    ENDSCN
00084A FC65 20 CF FC36        BRA    TIMER
00085                  *TEST FOR LAST SCREEN LOC AND MAKE SCROLL DECISION
00086A FC67 8C 2200  A ENDSCN CPX    #$2200   LAST SCRN LOC FILLED?
00087A FC6A 27 01 FC6D        BEQ    SCROL2
00088A FC6C 39                RTS
00089A FC6D 8D 71 FCE0 SCROL2 BSR    SCROLL
00090A FC6F 39                RTS
00091                  *THIS ROUTINE BLANKS ENTIRE SCREEN
00092                  *AND SERVICES INCOMING SERIAL CHAR
00093                  *DURING THE ROUTINE
00094A FC70 86 20    A BLANK  LDAA   #$20
00095A FC72 97 FF    A        STAA   SAVCHR
00096A FC74 CE 2000  A        LDX    #$2000
00097A FC77 DF FC    A        STX    TEMPX
00098A FC79 A7 00    A MORSCR STAA   0,X
00099A FC7B 08                INX
00100A FC7C 8C 2200  A        CPX    #$2200   END OF SCREEN?
00101A FC7F 26 07 FC88        BNE    SERTST   CHECK FOR INPUT SER CHAR
00102A FC81 DE FC    A        LDX    TEMPX
00103A FC83 A6 00    A        LDAA   0,X
00104A FC85 97 FF    A        STAA   SAVCHR
00105A FC87 39                RTS
00106A FC88 D6 11    A SERTST LDAB   $11
00107A FC8A 2A ED FC79        BPL    MORSCR   MORE BLNKING IF NO SER CHAR
00108A FC8C 3C                PSHX
00109A FC8D DE FC    A        LDX    TEMPX
00110A FC8F D6 12    A        LDAB   $12
00111A FC91 C4 7F    A        ANDB   #$7F
00112A FC93 37                PSHB
00113A FC94 C4 60    A        ANDB   #$60
00114A FC96 27 09 FCA1        BEQ    CONTRL   BRA IF CONTROL CHAR
00115A FC98 33                PULB
00116A FC99 E7 00    A        STAB   0,X
```

Figure 5. Program Listing (Continued)

```
00117A FC9B 08                 INX
00118A FC9C DF FC    A         STX    TEMPX
00119A FC9E 38                 PULX
00120A FC9F 20 D8 FC79         BRA    MORSCR
00121A FCA1 33        CONTRL   PULB
00122A FCA2 C1 08    A         CMPB   #$08      TEST FOR BACKSPACE
00123A FCA4 27 0F FCB5         BEQ    BACKS2
00124A FCA6 C1 0C    A         CMPB   #$0C      TEST FOR CURSOR FWD
00125A FCA8 27 15 FCBF         BEQ    FWDC2
00126A FCAA C1 0A    A         CMPB   #$0A      TEST FOR LINE FEED
00127A FCAC 27 16 FCC4         BEQ    LINEF2
00128A FCAE C1 0D    A         CMPB   #$0D      TEST FOR CARRIAGE RETURN
00129A FCB0 27 19 FCCB         BEQ    CARET2
00130A FCB2 38        GOBACK   PULX
00131A FCB3 20 C4 FC79         BRA    MORSCR
00132A FCB5 8C 2000  A BACKS2  CPX    #$2000
00133A FCB8 27 F8 FCB2         BEQ    GOBACK
00134A FCBA 09                 DEX
00135A FCBB DF FC    A         STX    TEMPX
00136A FCBD 20 F3 FCB2         BRA    GOBACK
00137A FCBF 08        FWDC2    INX
00138A FCC0 DF FC    A         STX    TEMPX
00139A FCC2 20 EE FCB2         BRA    GOBACK
00140A FCC4 C6 20    A LINEF2  LDAB   #$20
00141A FCC6 3A                 ABX
00142A FCC7 DF FC    A         STX    TEMPX
00143A FCC9 20 E7 FCB2         BRA    GOBACK
00144A FCCB 8C 2000  A CARET2  CPX    #$2000
00145A FCCE 27 E2 FCB2         BEQ    GOBACK
00146A FCD0 3C                 PSHX
00147A FCD1 DF FA    A         STX    TEMPX2
00148A FCD3 DC FA    A         LDD    TEMPX2
00149A FCD5 C4 1F    A         ANDB   #$1F
00150A FCD7 26 03 FCDC         BNE    MAS8
00151A FCD9 38                 PULX
00152A FCDA 20 D6 FCB2         BRA    GOBACK
00153A FCDC 38        MAS8     PULX
00154A FCDD 09                 DEX
00155A FCDE 20 EB FCCB         BRA    CARET2
00156A FCE0 D6 FE    A SCROLL  LDAB   STATWO
00157A FCE2 C4 20    A         ANDB   #$20
00158A FCE4 27 8A FC70         BEQ    BLANK
00159A FCE6 CE 2000  A         LDX    #$2000
00160                 *THE FOLLOWING LOOP SYNCHS SCROLL TO HORIZ
00161                 *SYNCH FROM VDG TO MINIMIZE NOISE ON SCREEN
00162                 *DURING WRITE TO DISPLAY RAM
00163A FCE9 D6 02    A NOTYET  LDAB   $02       PORT1 I/O
00164A FCEB C4 80    A         ANDB   #$80      ISOLATE H SYNCH
00165A FCED 27 FA FCE9         BEQ    NOTYET    WAIT FOR HIGH
00166A FCEF D6 02    A SAND4   LDAB   $02
00167A FCF1 C4 80    A         ANDB   #$80
00168A FCF3 26 FA FCEF         BNE    SAND4     WAIT FOR LOW
00169A FCF5 A6 20    A         LDAA   32,X
00170A FCF7 A7 00    A         STAA   0,X
00171A FCF9 08                 INX
00172A FCFA 8C 2200  A         CPX    #$2200
00173A FCFD 26 EA FCE9         BNE    NOTYET
00174A FCFF CE 21E0  A         LDX    #$21E0    SCRNPTR TO 1ST LOC,LAST LINE
```

Figure 5. Program Listing (Continued)

```
00175A FD02 39                    RTS
00176                   *CHECK FOR SERIAL INPUT
00177A FD03 96 11    A SERRX  LDAA   $11       TX/RX CONTR STATUS REG
00178A FD05 2B 01 FD08        BMI    SERDIS    TEST FOR BIT 7 (ICF)
00179A FD07 39                RTS
00180A FD08 96 12    A SERDIS LDAA   $12       RECEIVER REGISTER
00181A FD0A 8D 11 FD1D        BSR    DISPL
00182A FD0C BD FC67  A        JSR    ENDSCN
00183A FD0F 39                RTS
00184                   *HERE STATUS SWITCHES ARE CHECKED.  CALLING ADDR
00185                   *$3800 SELECTS NO MEMORY. STATUS SWITCHES
00186                   *ARE READ VIA LS244 LATCH
00187A FD10 B6 1800  A CHKSTA LDAA   SWITCH
00188A FD13 91 FE    A        CMPA   STATWO    HAVE SWITCHES CHANGED?
00189A FD15 26 01 FD18        BNE    CHGWO
00190A FD17 39                RTS
00191                   *CHANGE CONTROL WORD IN STATWO
00192A FD18 97 FE    A CHGWO  STAA   STATWO
00193A FD1A 97 10    A        STAA   $10       SERIAL MODE CONTROL REG
00194A FD1C 39                RTS
00195                   *THE DISPLAY ROUTINE FIRST CHECKS FOR CONTRL
00196                   *CHAR.  IF CONTRL CHAR, CHAR IS TESTED FOR
00197                   *ACTION; OTHERWISE CHAR IS DISPLAYED.
00198A FD1D 84 7F    A DISPL  ANDA   #$7F
00199A FD1F 36                PSHA
00200A FD20 84 60    A        ANDA   #$60
00201A FD22 27 12 FD36        BEQ    CNTRLC    TEST FOR CONTROL CHAR
00202A FD24 32                PULA
00203                   *WAIT FOR HORIZONTAL SYNCH
00204A FD25 D6 02    A SAND2  LDAB   $02
00205A FD27 C4 80    A        ANDB   #$80
00206A FD29 27 FA FD25        BEQ    SAND2
00207A FD2B D6 02    A SAND3  LDAB   $02
00208A FD2D C4 80    A        ANDB   #$80
00209A FD2F 26 FA FD2B        BNE    SAND3
00210A FD31 A7 00    A        STAA   0,X
00211A FD33 08                INX
00212A FD34 20 4C FD82        BRA    SAND1
00213                   *THE FOLLOWING ROUTINE DECODES CONTROL CHAR
00214A FD36 32          CNTRLC PULA
00215A FD37 81 0A    A        CMPA   #$0A      TEST FOR LINE FEED
00216A FD39 27 27 FD62        BEQ    LINEF
00217A FD3B 81 0D    A        CMPA   #$0D      TEST FOR CARRIAGE RETURN
00218A FD3D 27 3A FD79        BEQ    CARRET
00219A FD3F 81 08    A        CMPA   #$08      TEST FOR BACKSPACE
00220A FD41 27 5C FD9F        BEQ    BACKSP
00221A FD43 81 07    A        CMPA   #$07      TEST FOR BELL
00222A FD45 26 07 FD4E        BNE    CLRSCR
00223A FD47 96 08    A        LDAA   $08
00224A FD49 C4 01    A        ANDB   #$01
00225A FD4B 27 5F FDAC        BEQ    BELL      IS BELL ALREADY ON?
00226A FD4D 39                RTS
00227A FD4E 81 1A    A CLRSCR CMPA   #$1A      TEST FOR CLEAR SCREEN
00228A FD50 26 07 FD59        BNE    MORECH
00229A FD52 C6 20    A        LDAB   #$20
00230A FD54 D7 FF    A        STAB   SAVCHR
00231A FD56 7E FC70  A        JMP    BLANK
00232A FD59 81 0C    A MORECH CMPA   #$0C      TEST FOR CURSOR FWD
```

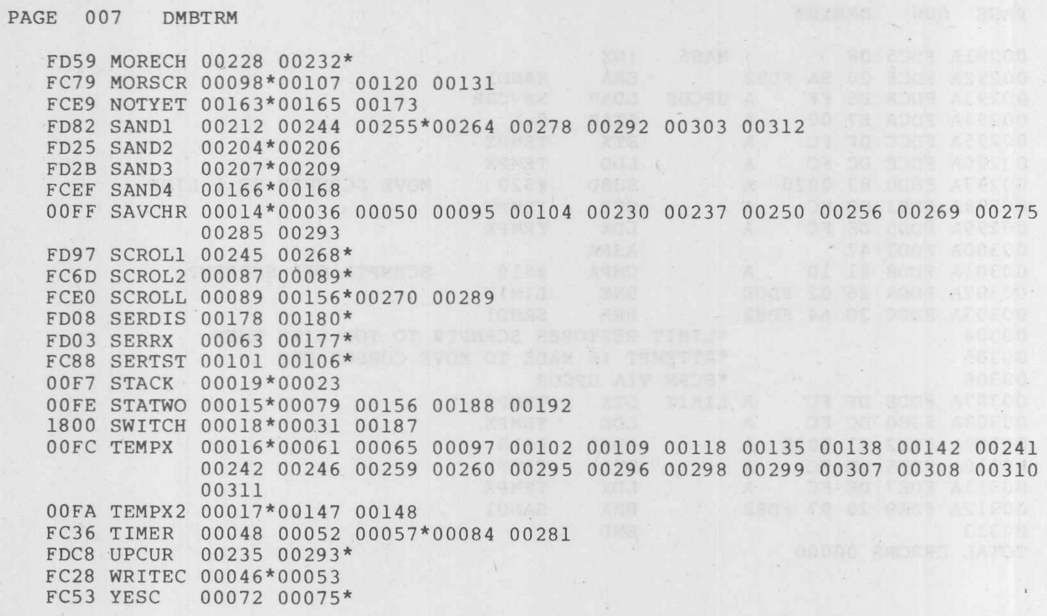
Figure 5. Program Listing (Continued)

```
PAGE  005    DMBTRM

00233A FD5B 27 5B FDB8           BEQ     FWDC
00234A FD5D 81 0B      A         CMPA    #$0B      TEST FOR UP CURSOR
00235A FD5F 27 67 FDC8           BEQ     UPCUR
00236A FD61 39                   RTS               DEFAULT BACK
00237A FD62 D6 FF      A LINEF   LDAB    SAVCHR
00238A FD64 E7 00      A         STAB    0,X
00239A FD66 C6 20      A         LDAB    #$20
00240A FD68 3A                   ABX               INCREMENT SCRNPTR 1 LINE
00241A FD69 DF FC      A         STX     TEMPX
00242A FD6B DC FC      A         LDD     TEMPX
00243A FD6D 81 22      A         CMPA    #$22      SCRNPTR OFF SCREEN?
00244A FD6F 26 11 FD82           BNE     SAND1
00245A FD71 8D 24 FD97           BSR     SCROL1    IF SCRNPTR OFF SCRN,SCROLL
00246A FD73 DC FC      A         LDD     TEMPX
00247A FD75 C4 1F      A         ANDB    #$1F      GET HORIZ POS OF SCRNPTR
00248A FD77 3A                   ABX
00249A FD78 39                   RTS
00250A FD79 D6 FF      A CARRET  LDAB    SAVCHR
00251A FD7B E7 00      A         STAB    0,X
00252A FD7D 8C 2000    A MAS5    CPX     #$2000    SCRNPTR ALREADY AT LIMIT?
00253A FD80 26 05 FD87           BNE     MAS3
00254                    *SAND1  STORES  CHAR UNDER CURSOR
00255A FD82 E6 00      A SAND1   LDAB    0,X
00256A FD84 D7 FF      A         STAB    SAVCHR
00257A FD86 39                   RTS
00258A FD87 3C           MAS3    PSHX
00259A FD88 DF FC      A         STX     TEMPX
00260A FD8A DC FC      A         LDD     TEMPX
00261A FD8C C4 1F      A         ANDB    #$1F      SCRNPTR AT 1ST LINE LOC?
00262A FD8E 26 03 FD93           BNE     MAS4
00263A FD90 38                   PULX
00264A FD91 20 EF FD82           BRA     SAND1
00265A FD93 38           MAS4    PULX
00266A FD94 09                   DEX               DECREMENT SCRNPTR
00267A FD95 20 E6 FD7D           BRA     MAS5      TEST SCRNPTR LOC AGAIN
00268A FD97 C6 20      A SCROL1  LDAB    #$20
00269A FD99 D7 FF      A         STAB    SAVCHR
00270A FD9B BD FCE0    A         JSR     SCROLL
00271A FD9E 39                   RTS
00272A FD9F 8C 2000    A BACKSP  CPX     #$2000
00273A FDA2 26 01 FDA5           BNE     MAS2
00274A FDA4 39                   RTS
00275A FDA5 D6 FF      A MAS2    LDAB    SAVCHR
00276A FDA7 E7 00      A         STAB    0,X
00277A FDA9 09                   DEX
00278A FDAA 20 D6 FD82           BRA     SAND1
00279A FDAC 86 01      A BELL    LDAA    #$01
00280A FDAE 97 08      A         STAA    $08       SET OLVL HIGH NEXT COMPARE
00281A FDB0 BD FC36    A         JSR     TIMER     PROVIDES BELL DURATION
00282A FDB3 86 00      A         LDAA    #$00
00283A FDB5 97 08      A         STAA    $08       SET OLVL LOW NEXT COMPARE
00284A FDB7 39                   RTS
00285A FDB8 D6 FF      A FWDC    LDAB    SAVCHR
00286A FDBA E7 00      A         STAB    0,X
00287A FDBC 8C 21FF    A         CPX     #$21FF    END OF SCREEN?
00288A FDBF 26 04 FDC5           BNE     MAS6
00289A FDC1 BD FCE0    A         JSR     SCROLL
00290A FDC4 09                   DEX
```

Figure 5. Program Listing (Continued)

```
00294A FDCA E7 00      A        STAB   0,X
00295A FDCC DF FC      A        STX    TEMPX
00296A FDCE DC FC      A        LDD    TEMPX
00297A FDD0 83 0020    A        SUBD   #$20       MOVE SCRNPTR UP 1 LINE
00298A FDD3 DD FC      A        STD    TEMPX
00299A FDD5 DE FC      A        LDX    TEMPX
00300A FDD7 47                  ASRA
00301A FDD8 81 10      A        CMPA   #$10       SCRNPTR OFF SCREEN?
00302A FDDA 26 02 FDDE          BNE    LIMIT
00303A FDDC 20 A4 FD82          BRA    SAND1
00304                  *LIMIT RESTORES SCRNPTR TO TOP LINE WHEN
00305                  *ATTEMPT IS MADE TO MOVE CURSOR OFF
00306                  *SCRN VIA UPCUR
00307A FDDE DF FC      A LIMIT  STX    TEMPX
00308A FDE0 DC FC      A        LDD    TEMPX
00309A FDE2 C3 0020    A        ADDD   #$20
00310A FDE5 DD FC      A        STD    TEMPX
00311A FDE7 DE FC      A        LDX    TEMPX
00312A FDE9 20 97 FD82          BRA    SAND1
00313                           END
TOTAL ERRORS 00000


   FCB5 BACKS2 00123 00132*
   FD9F BACKSP 00220 00272*
   FDAC BELL   00225 00279*
   FC70 BLANK  00041 00094*00158 00231
   FCCB CARET2 00129 00144*00155
   FD79 CARRET 00218 00250*
   FD18 CHGWO  00032 00189 00192*
   FC4D CHKC   00060 00071*
   FD10 CHKSTA 00057 00187*
   FD4E CLRSCR 00222 00227*
   FD36 CNTRLC 00201 00214*
   FC3F CONT   00061*00073
   FCA1 CONTRL 00114 00121*
   FD1D DISPL  00082 00181 00198*
   FC67 ENDSCN 00083 00086*00182
   FC2E ERASEC 00050*
   FC1E FILL   00037*00040
   FC63 FULLD  00081 00083*
   FDB8 FWDC   00233 00285*
   FCBF FWDC2  00125 00137*
   FCB2 GOBACK 00130*00133 00136 00139 00143 00145 00152
   FDDE LIMIT  00302 00307*
   FD62 LINEF  00216 00237*
   FCC4 LINEF2 00127 00140*
   FDA5 MAS2   00273 00275*
   FD87 MAS3   00253 00258*
   FD93 MAS4   00262 00265*
   FD7D MAS5   00252*00267
   FDC5 MAS6   00288 00291*
   FCDC MAS8   00150 00153*
   FC3D MORE   00060*00067
```

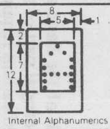
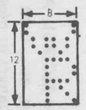
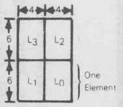
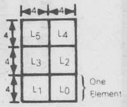
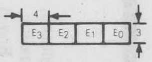
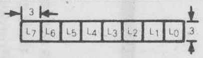
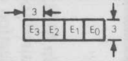
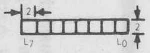
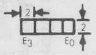
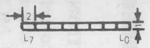
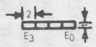
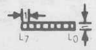
Figure 5. Program Listing (Continued)



```
PAGE  007   DMBTRM

   FD59 MORECH 00228 00232*
   FC79 MORSCR 00098*00107 00120 00131
   FCE9 NOTYET 00163*00165 00173
   FD82 SAND1  00212 00244 00255*00264 00278 00292 00303 00312
   FD25 SAND2  00204*00206
   FD2B SAND3  00207*00209
   FCEF SAND4  00166*00168
   00FF SAVCHR 00014*00036 00050 00095 00104 00230 00237 00250 00256 00269 00275
               00285 00293
   FD97 SCROL1 00245 00268*
   FC6D SCROL2 00087 00089*
   FCE0 SCROLL 00089 00156*00270 00289
   FD08 SERDIS 00178 00180*
   FD03 SERRX  00063 00177*
   FC88 SERTST 00101 00106*
   00F7 STACK  00019*00023
   00FE STATWO 00015*00079 00156 00188 00192
   1800 SWITCH 00018*00031 00187
   00FC TEMPX  00016*00061 00065 00097 00102 00109 00118 00135 00138 00142 00241
               00242 00246 00259 00260 00295 00296 00298 00299 00307 00308 00310
               00311
   00FA TEMPX2 00017*00147 00148
   FC36 TIMER  00048 00052 00057*00084 00281
   FDC8 UPCUR  00235 00293*
   FC28 WRITEC 00046*00053
   FC53 YESC   00072 00075*
```

Figure 5. Program Listing (Concluded)

Table 2. Detailed Description of VDG Modes

VDG Pins									Color			TV Screen		VDG Data Bus	Comments
$\overline{\text{MS}}$	G/$\overline{\text{A}}$	S/$\overline{\text{A}}$	EXT/$\overline{\text{INT}}$	GM2	GM1	GM0	CSS	INV	Character Color	Background	Border	Display Mode	Detail		
1	0	C	C	X	X	X	0 1	0 1 0 1	Green Black Orange Black	Black Green Black Orange	Black Black	32 Characters per row 16 Character rows	 Internal Alphanumerics	□ □ □ □ □ □ □ □ extra ASCII Code	The ALPHANUMERIC INTERNAL mode uses an internal character generator (which contains the following five dot by seven dot characters: .@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←SP !"#$%&'( )*+,-./0123456789:;<=>?) The six bit ASCII code leaves two bits free and these may be externally connected to the mode pins (G/$\overline{\text{A}}$, S/$\overline{\text{A}}$, EXT/$\overline{\text{INT}}$, GM2, GM1, GM0, CSS or INV).
1	0	0	1	X	X	X	0 1	0 1 0 1	Green Black Orange Black	Black Green Black Orange	Black Black	32 Characters per row 16 Character rows		□ □ □ □ □ □ □ □ One Row of Custom Characters	The ALPHANUMERIC EXTERNAL mode uses an external character generator as well as a row counter. Thus, custom character fonts or graphic symbol sets with up to 256 different 8 × 12 dot "characters" may be displayed.
1	0	1	0	X	X	X	X	X	Lx C2 C1 C0 0 X X X 1 0 0 0 1 0 0 1 1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1	Color Black Green Yellow Blue Red Buff Cyan Magenta Orange	Black	64 Display elements per row 32 rows of Display elements	 4 4 6 6 L_3 L_2 L_1 L_0 One Element	□ C_2 C_1 C_0 L_3 L_2 L_1 L_0 extra	The SEMIGRAPHICS FOUR mode uses an internal "coarse graphics" generator in which a rectangle (eight dots by twelve dots) is divided into four equal parts. The luminance of each part is determined by a corresponding bit on the VDG data bus. The color of illuminated parts is determined by three bits.
1	0	1	1	X	X	X	0 1	X	Lx C1 C0 0 X X 1 0 0 1 0 1 1 1 0 1 1 1 0 X X 1 0 0 1 0 1 1 1 0 1 1 1	Color Black Green Yellow Blue Red Black Buff Cyan Magenta Orange	Black	64 Display elements per row 48 rows of Display elements	 4 4 4 4 4 L_5 L_4 L_3 L_2 L_1 L_0 One Element	C_1 C_0 L_5 L_4 L_3 L_2 L_1 L_0	The SEMIGRAPHIC SIX mode is similar to the SEMIGRAPHIC FOUR mode with the following differences. The eight dot by twelve dot rectangle is divided into six equal parts. Color is determined by the two remaining bits.
1	1	X	X	0	0	0	0 1	X	C1 C0 0 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1	Color Green Yellow Blue Red Buff Cyan Magenta Orange	Green Buff	64 Display elements per row 64 rows of Display elements		C_1 C_0 C_1 C_0 C_1 C_0 C_1 C_0	The COLOR GRAPHICS ONE mode uses a maximum of 1024 bvtes of display RAM in which one pair of bits specifies one picture element
1	1	X	X	0	0	1	0 1	X	Lx 0 1 0 1	Color Black Green Black Buff	Green Buff	128 Display elements per row 64 rows of Display elements	 3 3 L_7 L_6 L_5 L_4 L_3 L_2 L_1 L_0	L_7 L_6 L_5 L_4 L_3 L_2 L_1 L_0	The RESOLUTION GRAPHICS ONE mode uses a maximum of 1024 bytes of display RAM in which one bit specifies one picture element
1	1	X	X	0	1	0	0 1	X	Same color as Color Graphics One		Green Buff	128 Display elements per row 64 rows of Display elements		C_1 C_0 C_1 C_0 C_1 C_0 C_1 C_0	The COLOR GRAPHICS TWO mode uses a maximum of 2048 bytes of display RAM in which one pair of bits specifies one picture element
1	1	X	X	0	1	1	0 1	X	Same color as Resolution Graphics One		Green Buff	128 Display elements per row 96 rows of Display elements		L_7 L_6 L_5 L_4 L_3 L_2 L_1 L_0	The RESOLUTION GRAPHICS TWO mode uses a maximum of 1536 bytes of display RAM in which one bit specifies one picture element
1	1	X	X	1	0	0	0 1	X	Same color as Color Graphics One		Green Buff	128 Display elements per row 96 rows of Display elements		C_1 C_0 C_1 C_0 C_1 C_0 C_1 C_0	The COLOR GRAPHICS THREE mode uses a maximum of 3072 bytes of display RAM in which one pair of bytes specifies one picture element.
1	1	X	X	1	0	1	0 1	X	Same color as Resolution Graphics One		Green Buff	128 Display elements per row 192 rows of Display elements		L_7 L_6 L_5 L_4 L_3 L_2 L_1 L_0	The RESOLUTION GRAPHICS THREE mode uses a maximum of 3072 bytes of display RAM in which one bit specifies one picture element.
1	1	X	X	1	1	0	0 1	X	Same color as Color Graphics One		Green Buff	128 Display elements per row 192 rows of Display elements		C_1 C_0 C_1 C_0 C_1 C_0 C_1 C_0	The COLOR GRAPHICS SIX mode uses a maximum of 6144 bytes of display RAM in which one pair of bits specifies one picture element
1	1	X	X	1	1	1	0 1	X	Same color as Resolution Graphics One		Green Buff	256 Display elements per row 192 rows of Display elements		L_7 L_6 L_5 L_4 L_3 L_2 L_1 L_0	The RESOLUTION GRAPHICS SIX mode uses a maximum of 6144 bytes of display RAM in which one bit specifies one picture element

AN799

APPLICATION PROTOTYPE BOARD (APB) FOR MC6801/MC6803/MC68701 MCUs

Prepared By
David Ruhberg
Applications Engineer

INTRODUCTION

Now that cost effective single-chip MCUs are introduced, a similar cost effective design is required for their evaluation. The MC6801 MCU Family Application Prototype Board (APB) is a printed circuit board meeting these requirements and may be fabricated from the artwork provided in Appendix A. Fabrication of the APB will allow evaluation of the MC6801 Family of MCUs and custom programmed MC6801 versions. The wirewrap allows the user to construct and finalize a prototype for PC board fabrication. The existing artwork may then be used as a nucleus to reduce layout time.

GENERAL APPLICATIONS

After assembling the APB, it can be used for evaluation of Motorola's MC6801L1, MC6803, MC6803NR, and MC68701 Microcomputer/Microprocessors, plus custom programmed versions of the MC6801. All of the basic address decoding, logic support, etc., is an integral part of the completed APB; the only changes or modifications required are a result of user expansion. Figure 1 shows the basic components used with the APB.

The printed circuit board is made up of two separate areas; a printed circuit area and a wirewrap area. Once the board is fabricated, the printed circuit area bypasses the prototype "wirewrap stage," and eliminates the associated wirewrap mistakes. This area provides connection and mounting space for the components, shown in Figure 2. The wirewrap area can be used for mounting and/or connecting additional devices such as buffers, memory, decoders, etc.

As an elementary software development system, the completed APB contains the necessary hardware to accomodate

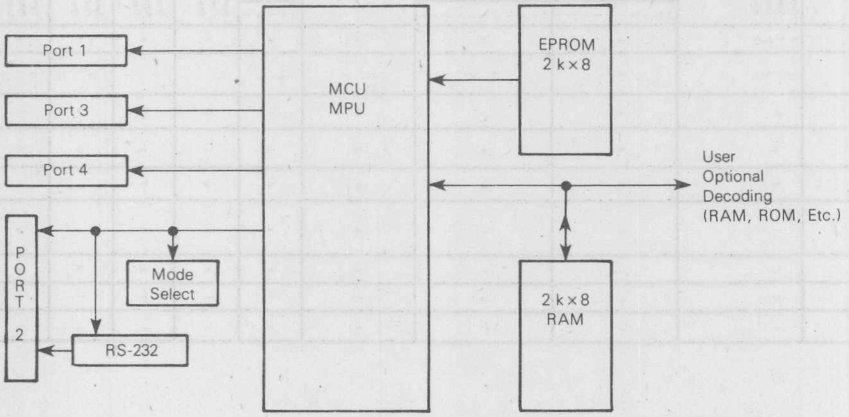


FIGURE 1 — BASIC APB COMPONENTS

the monitor ROM features in LILbug®. Thus, the APB together with LILbug provides features which allow the user to: (1) Develop and edit software programs; (2) Hardware trace through the user's program; (3) Insert, display, and remove breakpoints in the program; and (4) Provide punch, load, and verify commands for software I/O. To utilize the ROM features in LILbug, the MC6801L1 MCU must be used.

The APB provides an RS-232 full-duplex interface to provide the connection between a terminal and the MCU Serial Communication Interface (SCI). The punch, load, and verify are convenient to use when the terminal is equipped with magnetic tape capabilities. The APB circuit board is only 4" × 6.25" (4" × 4.125" if the wirewrap area is eliminated). Therefore, in addition to its use as a software/hardware development tool, its compact size allows it to fit into many application environments. Figure 3 provides a view of the completed APB. Mounting holes are provided at the four corners of the APB board.

SERIAL COMMUNICATION INTERFACE

One of the most useful features of the APB is its adaptability to interface with various serial I/O devices. Two examples of these I/O devices are a terminal and a memory tape system.

To communicate with the APB a user-supplied data terminal is generally required. The APB "powers up" expecting a data terminal set for 300 baud and full-duplex operation. The interface necessary to convert the serial I/O TTL levels to RS-232 levels is provided by the circuit that uses Q1, Q2 and Q3, shown in Figure 2. The APB end of the cable connecting the APB to the data terminal must be an 8-pin male DIP plug. The input to this circuitry is through pin "b" on the Port 2 socket (see Figure 2). The output signals leave through pin "a" on the Port 2 socket.

The APB utilizes an RS-232 interface to provide a convenient method of accessing tape. When the terminal has a tape system (e.g., a Silent 700/w Dual Digital Tape Drivers), the RS-232 interface may be used and no additional hardware is needed. The information format used by the on-board monitor (LILbug) is the S1-S9 which is the same as that used in EXbug®. An example of this is shown in Figure 4. The first two digits provide the identification (ID), S1 the Data Records, and S9 for the Data Trailer. The next two digits in the row contain the length of the data string (L) and the next four digits contain the hex address (ADD). The data string then follows, and the last two digits contain a check sum of the data in the row. When using LILbug, the user has the option of redefining its I/O table such that the input and/or output is user-defined. Refer to "LILbug Monitor for the MC6801L1" (not part of this Application Note) for more information.

For any given input frequency ($4f_O$), the serial I/O port will operate at one of four programmable baud rates. The four baud rates are determined by the input frequency ($4f_O$). Of the four available baud rates, the particular one used is obtained by writing the appropriate bits into the rate and mode control register. Details can be found in the "MC6801 Advance Information Sheet" (not included as part of this Application Note). The MCU will run at one-fourth of the input frequency (f_O or E). The particular baud rate is derived by dividing the E clock by 16, 128, 1024 or 4096. Two examples of input frequency selection are shown below. An external baud rate clock may be supplied as described in the "MC6801 Advance Information Sheet."

Example	Input	E	Baud Rate			
	Freq. MHz	MHz	E/16	E/128	E/1024	E/4096
1	2.4576	0.6144	38.4 k	4.8 k	600	150
2	4.9152*	1.2288	76.8 k	9.6 k	1200	300

*For input frequency between 4 and 5 MHz use MC6801-1 MCU.

TABLE 1 — MODE SELECTION SUMMARY

Mode	Pin 10 S4c	Pin 9 S4b	Pin 8 S4a	ROM	RAM	Interrupt Vectors	Bus Mode	Operating Mode
7	H	H	H	I	I	I	I	Single Chip
6	H	H	L	I	I	I	MUX(6)	Multiplexed/Partial Decode(5)
5	H	L	H	I	I	I	NMUX(6)	Non-Multiplexed/Partial Decode(5)
4	H	L	L	I(2)	I(1)	I	I	Single Chip Test
3	L	H	H	E	E	E	MUX	Multiplexed/No RAM or ROM(4)
2	L	H	L	E	I	E	MUX	Multiplexed/RAM(4)
1	L	L	H	I	I	E	MUX	Multiplexed/RAM and ROM(4)
0	L	L	L	I	I	I(3)	MUX	Multiplexed Test(4)

Legend:

I — Internal	NMUX — Non-Multiplexed
E — External	L — Logic "0" (Switch Closed)
MUX — Multiplexed	H — Logic "1" (Switch Open)

Notes:

(1) Internal RAM is addressed at $XX80

(2) Internal ROM is disabled

(3) $\overline{\text{RESET}}$ vector is external for 2 cycles after $\overline{\text{RESET}}$ goes high

(4) Addresses associated with Ports 3 and 4 are considered external in Modes 0, 1, 2, and 3

(5) Addresses associated with Port 3 are considered external in Modes 5 and 6

(6) Port 4 default is user data input; address output is optional by writing to Port 4 Data Direction Register

® Registered trademark of Motorola Inc.

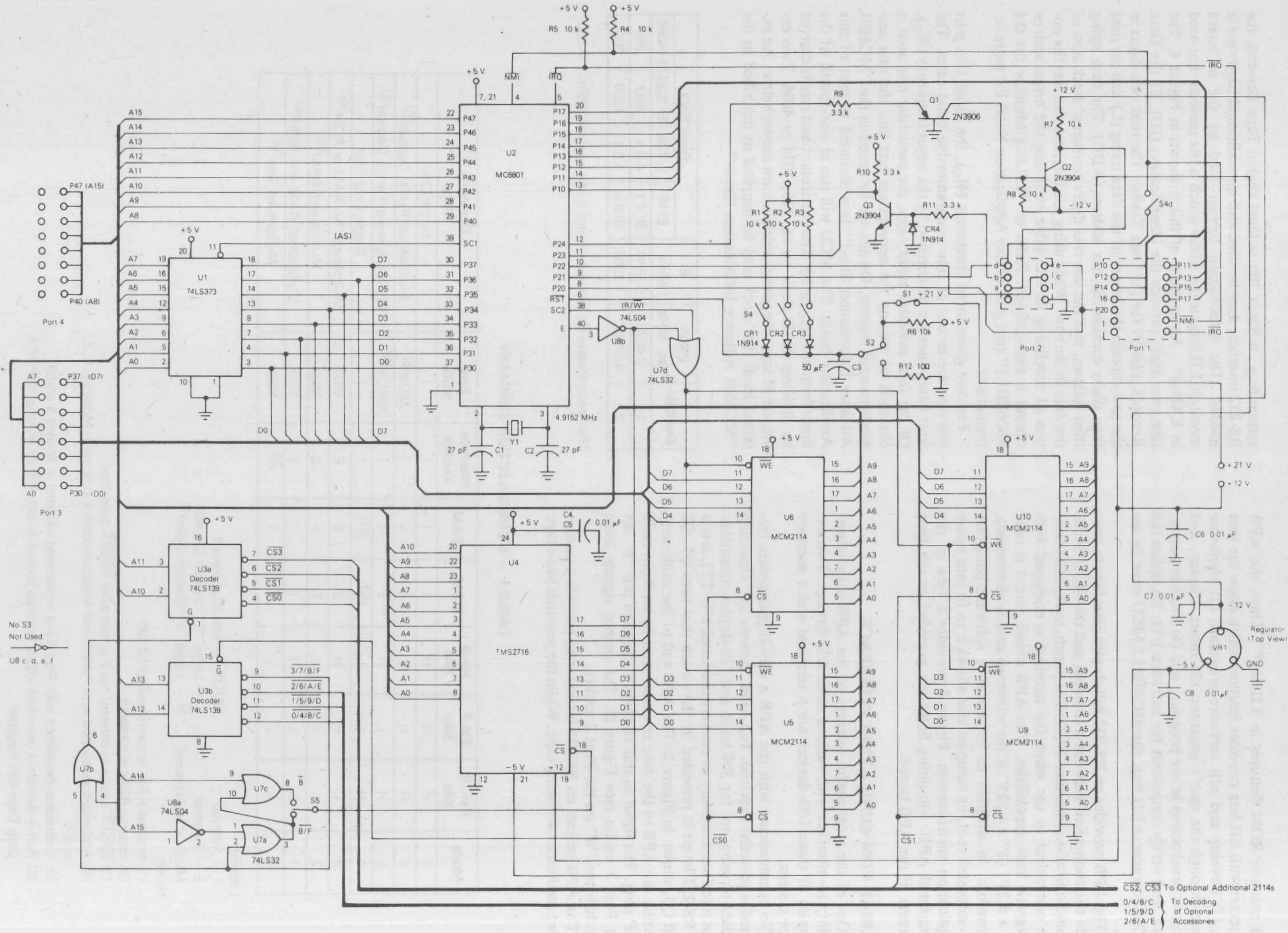
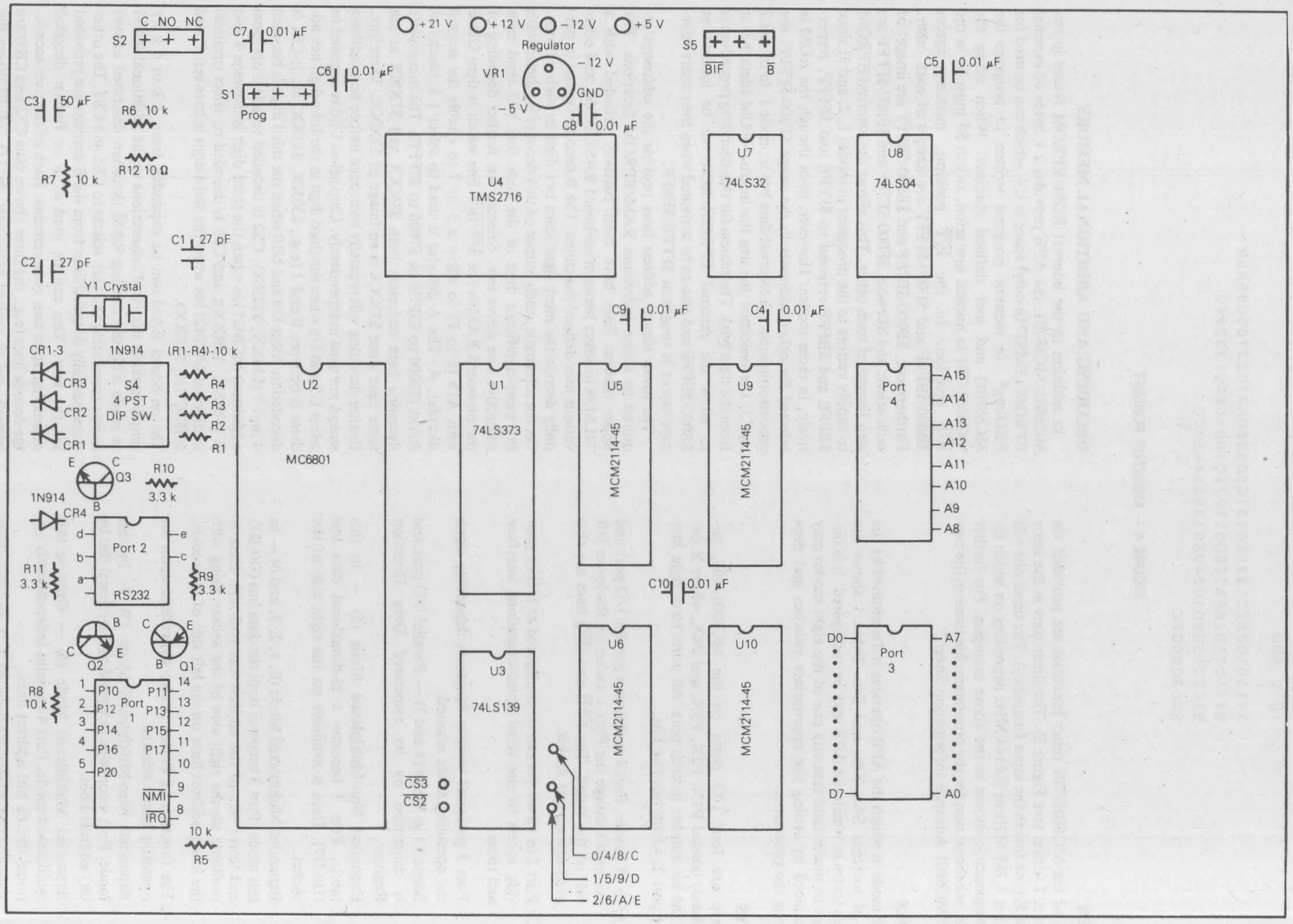


FIGURE 2 — MC6801 APB SCHEMATIC DIAGRAM



Note: No S3

FIGURE 3 — APB PART LOCATION DETAIL

```
ID L ADD

S11301008E0132FE0137C603960AA1022705095A59
S111011026F63EBD01197E010016BA013339F0
S10C0133801004050139534554 00
S9030000FC
```

FIGURE 4 — MONITOR FORMAT

TIMER

All of the MC6801/701 timer functions are provided via the Port 2 socket (see Figure 3). The timer runs at the same rate as E (one fourth the input frequency). The timer rate will be either 1.288 MHz or 0.6144 MHz depending on which input frequency is chosen in the above examples. For further details, see the section for the programmable timer in the current "MC6801 Advance Information Sheet."

MODES

The mode in which the APB operates is determined by the state of switches S4a, S4b, and S4c. Table 1 shows the various states in which the APB may be configured. It is important to understand that any one of the eight modes may be obtained by setting the appropriate switches and then resetting the processor.

PORTS

There are four I/O ports on the MC6801/701, appropriately labeled P1X, P2X, P3X, and P4X, where X denotes the bit number in each port. All ports have eight bits except port 2, which has five bits.

Port 1: In all modes, Port 1 is always a parallel I/O port and accessed through the Port 1 socket on the lower left side of the board. The $\overline{\text{NMI}}$ and $\overline{\text{IRQ}}$ lines are also available on this socket.

Port 2: Port 2 in all modes can be configured as I/O or provide access to the serial communications interface and timer.

Port 3: Port 3 performs various functions depending upon the operating mode selected.

Single Chip Mode (4 and 7) — Parallel I/O port and is controlled by its associated Data Direction Register.

Expanded Non-Multiplexed Mode (5) — In this mode, Port 3 becomes a bi-directional data bus (D0-D7). Data is available on the right side of the socket.

Expanded Multiplexed Mode (0, 1, 2, 3, and 6) — In this mode, Port 3 becomes both the data bus (D0-D7) and lower bits of the address bus (A0-A7). Data is available on the right side of the socket along with the lower address lines on the left side of the socket.

Port 4: The function of Port 4 is also dependent upon the operating mode selected.

Expanded Non-Multiplexed Mode (5) — In this mode, Port 4 contains independently optional (bit by bit) address lines.

Expanded Multiplexed Mode (6) — Only in this multiplexed mode, Port 4 contains independently optional (bit by bit) address lines.

Expanded Multiplexed Modes (0, 1, 2, and 3) — Port 4 functions only as address lines A8-A15.

DECODING AND ADDITIONAL MEMORY

In addition to the internal ROM/EPROM found in the MC6801/MC68701, the APB provides 2 k bytes of external EPROM (TMS2716 only) space at U4, which can be used for PRObug® (a monitor program written to program the MC68701) and user defined routines. When using the MC6801/701 in normal operation, switch S5 must be in the $\overline{\text{B/F}}$ position. In the $\overline{\text{B/F}}$ position, memory spaces $B000-$BFFF and $F000-$FFFF are images of each other. Furthermore, $B000-$B7FF and $B800-$BFFF are images of each other; and likewise, $F000-$F7FF and $F800-$FFFF are also images of each other. This allows the external EPROM to supply vectors to the processor in modes 1, 2, and 3; since $BFFE and $BFFF respond to $FFFE and $FFFF, respectively, in these modes. However, even though the ROM is selected for other addresses in the range $F800-$FFFF, the processor ignores the external data bus in mode 1. In modes 2 and 3, the processor ignores the internal ROM (detaches it from the data bus). The reason for this decoding procedure is to allow the external EPROM/ROM to be located at $B000-$BFFF and also to be activated when the restart interrupt vector is sought at $FFFE-$FFFF.

The lower eleven address lines provide the addresses required to access individual RAM/EPROM locations. The upper address lines have been partially decoded with a 74LS139 to select between on-board RAM, ROM, and other various user defined functions. The B decoder is used to partially decode the most significant hex digit in each address. On the schematic, each output of this decoder is labeled with the most-significant digit of the code that will send each respective line active low. Selected for further decoding of the on-board RAM is the $\overline{\text{3/7/B/F}}$ line which is then ORed with A15 (8 to F) to allow a 3 or 7 to enable the second decoder, A. The A decoder is used to select 1 k blocks of RAM ($3000 to $3FFF or $7000 to $7FFF). The incomplete decoding here addresses both $3XXX and $7XXX at the same time since $3XXX is an image of $7XXX. Therefore, further decoding will typically be needed before both address spaces are used independently. Chip select CS0 is decoded to select U5 and U6 when the third digit in the four digit hex address is between 0 and 3 (e.g., $30XX, $33XX, etc.); CS1 is decoded to select U9 and U10 when the third digit is between 4 and 7 ($34XX, $37XX); CS2 is decoded to select optional additional MCM2114s when the third digit is between 8 and B ($38XX, $3BXX); and CS3 is decoded to select optional additional MCM2114s when the third digit is between C and F ($3CXX, $3FXX).

The on-board RAM can be expanded from 2 k to 4 k by simply making the same connections to the additional RAM as made with the existing RAM (location described above) except connecting the chip selects to CS2 and CS3. The other three partially decoded lines from the B decoder are provided for the user. This can be used as is or further decoded depending upon user requirements. Each of the user accessible decode lines (e.g., the three above, plus CS2 and CS3) are brought out to unused holes near the U3 (74LS139) on the PC board to provide easier access (see Figure 4).

There are three positions marked on the APB which indicate where power is to be supplied for normal operation. The user must supply filtered +5 V, +12 V and −12 V. (Since the negative voltage supply is necessary for the RS-232 interface, no significant advantage is gained by using +5 V only EPROMs.) The worst-case current consumption is 5 V/790 mA, +12 V/20 mA and −12 V/60 mA. The +5 V supply is used by all ICs on the APB. The +12 V and −12 V supplies are used by the TMS2716 EPROM and the RS-232 output driver transistor.

A wirewrap area is provided to add other M6800 family peripherals to the basic APB system. The user is then permitted to construct his target MC6801-based system. To facilitate ease of construction in the wirewrape area, the address lines A0-A15 and data lines D0-D7 are available adjacent to the wirewrap area. These signals are available as pins of two DIP sockets, preferably wirewrap sockets, when external parts are employed in the wirewrap area. In the single chip mode, DIP headers may be used to connect to parallel Ports 3 and 4.

SPECIFIC OPERATION

The APB operates with the standard MC6801, MC6801L1, MC6803, MC6803NR, and the MC68701. The MC6801 contains an enhanced MC6800 processor, 2 k of internal mask-programmed ROM, 128 bytes of RAM, a 16-bit timer and a serial I/O section. The MC6801L1 is similar, except the mask-programmed ROM is programmed with a debug monitor called LILbug. The MC6803 is similar to the MC6801, except there is no internal masked-programmed ROM. The MC6803NR has no RAM. The MC68701 is similar to the MC6801, except the internal ROM is electrically programmable, alterable, and ultra-violet erasable (see Figure 5).

MC6801L1 MICROPROCESSOR

To facilitate debugging and development of applications software, the user may elect to use the MC6801L1 processor which contains the monitor LILbug. All modes listed in Table 1 are available on the MC6801L1.

The APB allows use of the hardware trace function of LILbug by closure of S4d. Switch S4d connects the output level of the internal timer to the non-maskable interrupt of the MC6801. LILbug then provides the programming necessary to implement a trace through specified RAM or ROM memory locations. This capability allows the user to single-step through programs that are ROM-based. The Memory Map of various modes utilizing the MC6801L1 on the APB Board is presented in Figure 6.

MC6803 and EPROM

Since the MC6803 has no on-chip ROM available to the user, the only useable modes are those that access external ROM to pick up its restart vectors. These modes are:

a) The MULTIPLEXED/RAM (mode 2)

b) The MULTIPLEXED/NO RAM or ROM (mode 3)

The Memory Map for the MC6803 is shown in Figure 7.

For the case of the MC6803NR, the mode should be set to the MULTIPLEXED/NO RAM or ROM mode (mode 3). The Memory Map is similar to that shown in Figure 7 except all on-chip RAM address space is replaced with user definable space.

MC68701 and PRObug

When using the MC68701 with the APB, all modes are the same, as shown in Table 1, except mode 0 is used in the on-chip EPROM programming mode.

In order to program the MC68701, the user must supply well-filtered +21 Vdc. However, before the +21 V is actually utilized by the internal MC68701 EPROM, the user must place switch S1 in the closed position.

For illustrative purposes, it is assumed the user will make use of the PRObug software available in external ROM. After inserting the PRObug ROM into the external ROM/EPROM socket (U4), the internal EPROM can be programmed. In order to use PRObug to program the internal EPROM of the MC68701, switch S5 must be in the B position. This position allows the external ROM to respond only to $B000-$BFFF. This is necessary in the programming mode of the MC68701 since the ROM should not be selected during processor output of image address $F000-$FFFF. PRObug may still be used since the MC68701 processor puts out the restart addresses $BFFE and $BFFF after reset when in the programming mode.

After programming is complete, the external ROM/EPROM (PRObug) may be replaced with another ROM/EPROM to utilize the same addresses. The Memory Map for the MC68701 is shown in Figure 8.

More details about the PRObug software are provided in the "PRObug Preliminary Programming Monitor" (not part of this Application Note).

ASSEMBLY

The APB can be assembled using the part location detail of Figure 3. No special instructions are necessary.

CONCLUSION

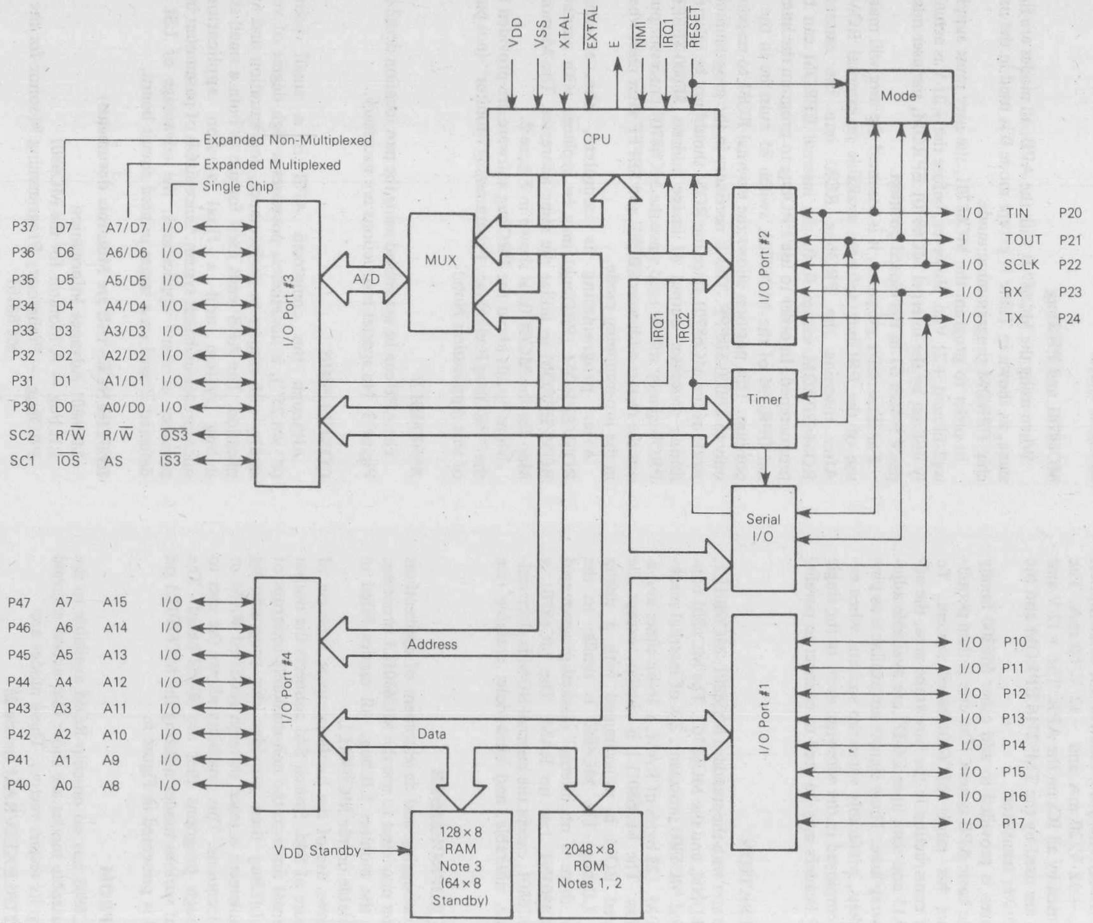
Although the completed APB is a small board (4" × 6.25"), it nonetheless possesses a high degree of versatility with respect to performance, configuration and application. The APB lends itself for use as both a small size debug system and a final version applications microprocessor-based system. Since size is of paramount importance in many applications, the advantage of LSI is defeated if used on a larger printed circuit board.

REFERENCES (All are Motorola documents)

MC6801 Advance Information

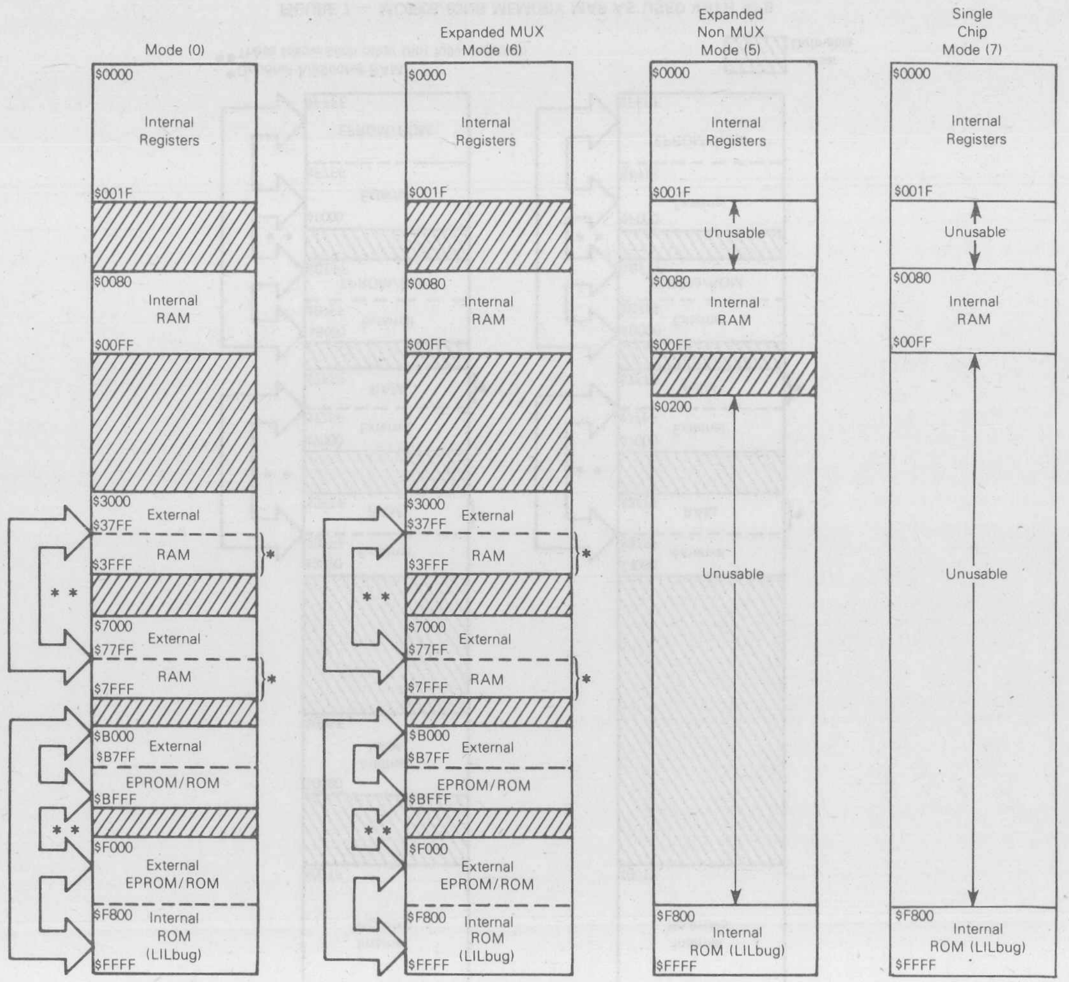
LILbug — A Monitor for the MC6801

PRObug — Preliminary Programming Monitor for the MC68701



Notes:
1. Not on the MC6803 or MC6803NR
2. EPROM on the MC68701
3. Not on MC6803NR

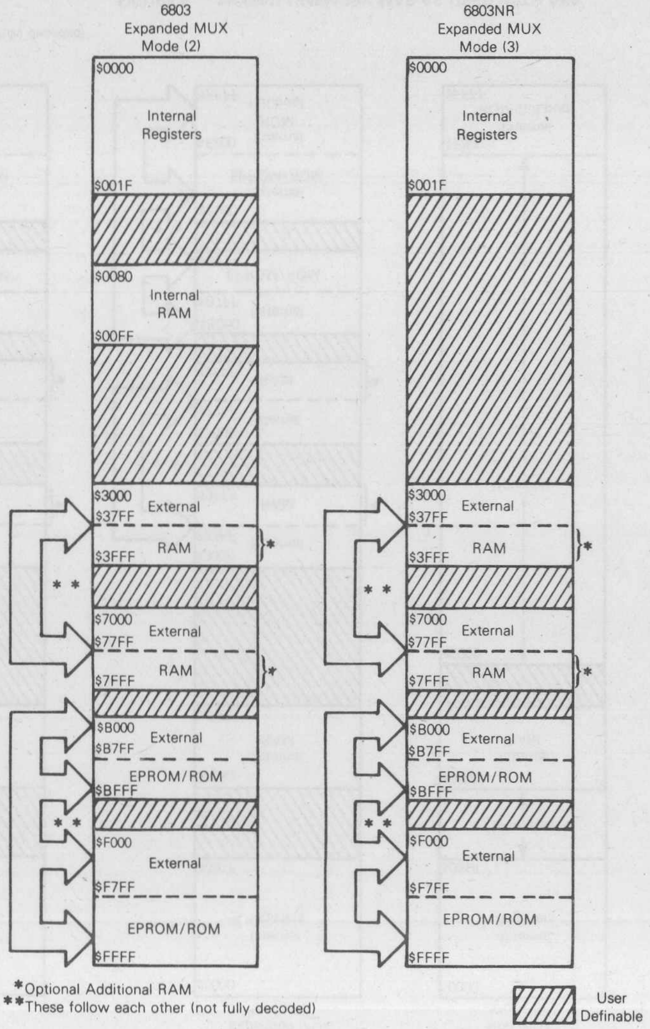
FIGURE 5 — MC6801 SINGLE CHIP MICROCOMPUTER



*Optional Additional RAM
**These follow each other (not fully decoded)

▨ User Definable

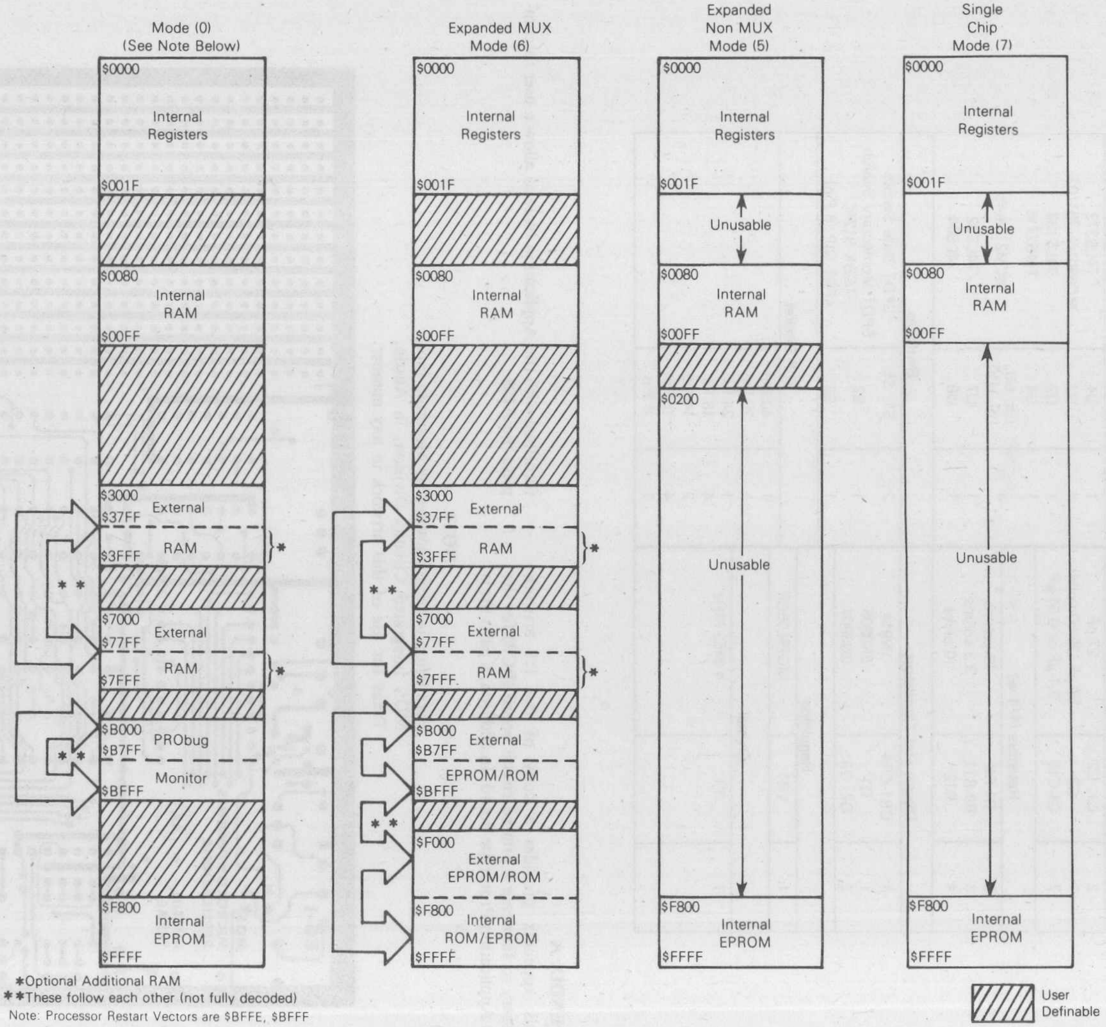
FIGURE 6 — MC6801L1 MEMORY MAP AS USED WITH APB



*Optional Additional RAM

**These follow each other (not fully decoded)

FIGURE 7 — MC6803/03NR MEMORY MAP AS USED WITH APB



*Optional Additional RAM

**These follow each other (not fully decoded)

Note: Processor Restart Vectors are $BFFE, $BFFF

FIGURE 8 — MC68701 MEMORY MAP AS USED WITH APB

APB KIT PARTS LIST

Quantity	Ref. Desig.	Value/Description
	Capacitors	
2	C1, C2	27 pF
1	C3	50 μF (35-50 volts)
7	C4-C10	0.1 μF or 0.01 μF
	Resistors (1/4 w)	
8	R1-R8	10 kohms
3	R9-R11	3.3 kohms
1	R12	10 ohms
	Diodes and Transistors	
4	CR1-CR4	1N914
1	Q1	2N3906
2	Q2, Q3	2N3904
	Regulator	
1	VR1	MC79L05CP
	Crystal	
1	Y1	4.9152 MHz

Quantity	Ref. Desig.	Value/Description
	Motorola ICs	
1	U1	74LS373
1	U2	MC6801-1/701/03
1	U3	74LS139
1	U4	TMS2716
4	U5, U6, U9, U10	MCM2114-45
1	U7	74LS32
1	U8	74LS04
	Switches	
2	S1, S5	SPDT Slide Switch
1	S2	SPDT Momentary Switch (C&K 8121C)
1	S4	4PST DIP (8 Pin)
	DIP Socket	
1	40 Pin	
1	20 Pin	
1	24 Pin	
4	18 Pin	
2	14 Pin	
4	16 Pin	
1	8 Pin	

APPENDIX A

This appendix provides a copy of the 1:1 artwork necessary to fabricate a printed circuit board (PCB) for the APB Application Prototype Board. In addition, a parts list is furnished with the Application Note to allow a user to complete the APB.

NOTE

Permission is hereby granted by Motorola Inc., MOS Integrated Circuits Division, in Austin, Texas for use of this artwork in any manner.

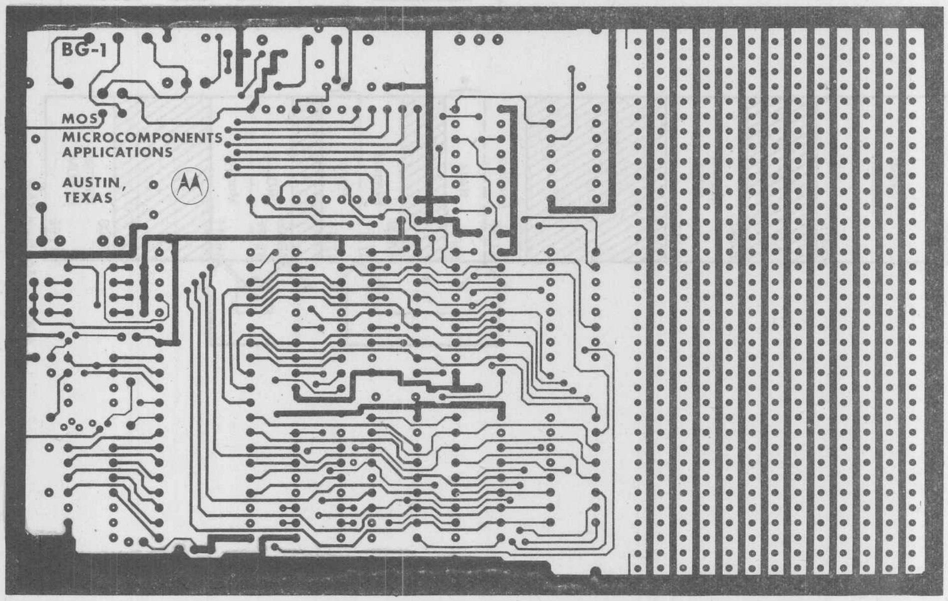


FIGURE A-1 — COMPONENT SIDE ARTWORK

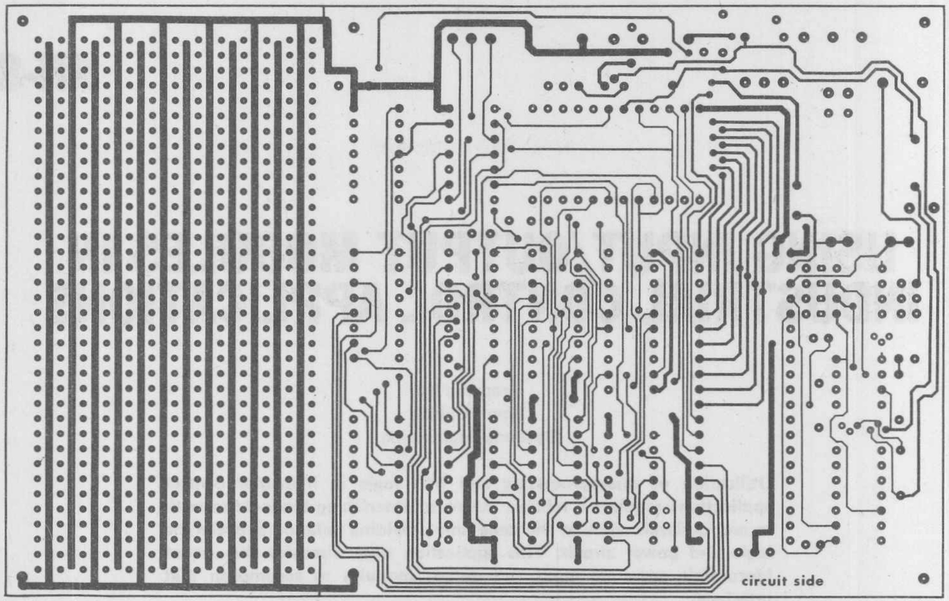


FIGURE A-2 — SOLDER SIDE ARTWORK

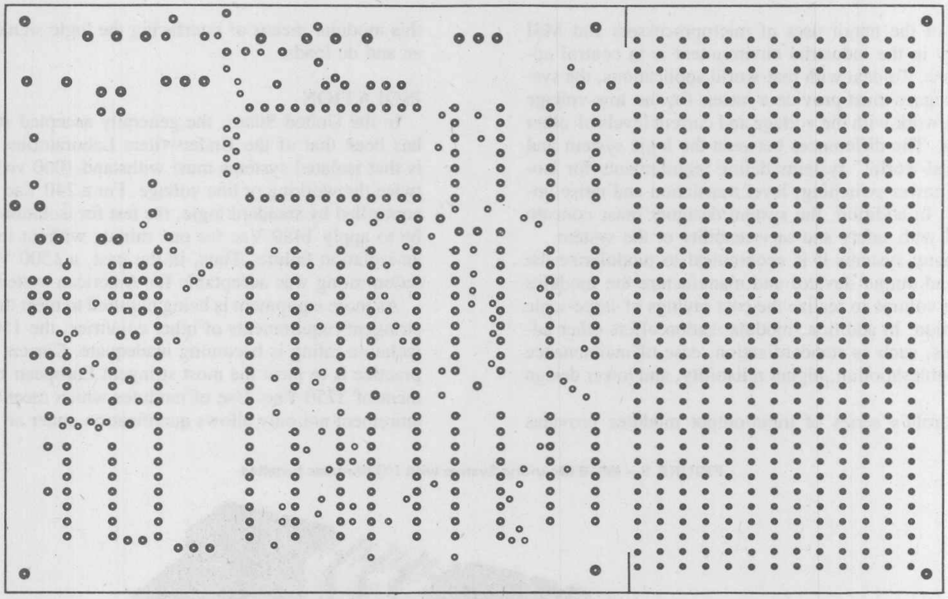


FIGURE A-3 — HOLE PATTERN ARTWORK

AN-801

USING INPUT/OUTPUT MODULES IN INDUSTRIAL CONTROL APPLICATIONS

Prepared by
Tom Hopkins
Systems Engineering

Utilization of microprocessors and MSI logic in industrial control applications requires a reliable means of interfacing the logic to both ac and dc levels while at the same time providing isolation between the logic and power circuit. This application note discusses the use of Motorola's series of input and output modules to accomplish that interface.

One of the major uses of microprocessors and MSI circuitry in the industrial environment is in control applications. To deal with real-world applications, the system designer must provide a means for the low-voltage logic to work with the voltage and current levels of other systems. The differences between the logic system and the "real-world" systems define requirements for isolation, power switching, level translation and noise immunity. In addition, the system designer must concern himself with safety and serviceability of the system.

In many systems it is economical to modularize the input and output devices and manufacture the modules in large volume to realize the cost savings of large-scale production. In addition, modularization offers other advantages, such as standardization, ease of maintenance and troubleshooting, higher reliability, and lower design cost.

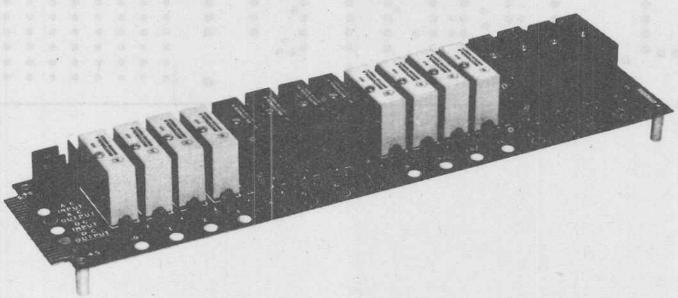
Motorola's series of input/output modules provides this modular means of interfacing the logic signals with ac and dc loads.

ISOLATION

In the United States, the generally accepted standard has been that of the Underwriters Laboratories, which is that isolated systems must withstand 1000 volts plus twice the working or line voltage. For a 240 Vac system controlled by standard logic, the test for isolation would be to apply 1480 Vac for one minute without inducing an isolation failure. Thus, in the past, a 1500 Vac isolation rating was acceptable for American systems.

As more equipment is being required to meet the more stringent requirements of other countries, the 1500 Vac isolation rating is becoming inadequate. Current design practice is to meet the most stringent European requirement of 3750 Vac. Use of modules which meet this requirement not only allows qualification under all known

FIGURE 1 – MS16 Mounting System with I/O Modules Installed



requirements in multinational markets, but establishes a clearly superior and safer product.

POWER SWITCHING

Motorola output modules are not intended to be the final load handling devices in all systems. They do, however, have ratings adequate to handle many small loads such as fractional horsepower motors, small heaters, solenoid valves, and lamps. In addition, the modules are capable of driving final load handling devices such as motor starters.

SAFETY CONSIDERATIONS

One of the important safety considerations is how to connect the output device to the wiring harness. Current practice is evolving toward the elimination of screw terminals on I/O modules. By using the plug-and-socket type of connection, the module can be installed or removed without working with hot wires. This can result in a significant saving of maintenance time, since the electrician no longer must lock out feeder circuits before maintenance. Hence, the current practice of plugging the module into a socketed mounting board and attaching the wiring harness to the board using screw terminals is becoming universally accepted. This type of mounting also has the advantage of making installation more convenient since the wiring can be done before the "electronics" is installed.

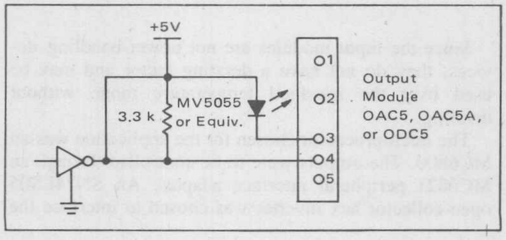
A second topic related to safety is that of fusing. Since most fusing requirements reflect code-writing agencies' concerns to "protect wire," fusing specifications are often outgrowths of safe current levels for wire with regard to heat generation. From the electronics point of view, we are usually more concerned with protecting electronic equipment than wire. Hence, the size, location, and type of fuse is best selected by the system designer. The Motorola mounting boards have provision for a pigtail fuse to be installed in series with each module and the field wiring. The standard boards have a 5-A fuse installed at the factory.

LOGIC INTERFACE

Once the decision to use I/O modules has been made, the only remaining task for the system engineer is to interface the module with the logic and the equipment. For the most part, the interface with the equipment is quite simple, since the module most generally goes in series with the field device.

On the logic side, the interface to be used depends on the type of logic used in the logic system. For TTL logic, the interface is quite simple since a standard TTL output will drive the output modules directly, as shown in Figure 2. This configuration may be used with standard TTL, Schottky (S), low-power Schottky (LS), and high-speed (H) series devices. Low-power TTL (L) may be buffered with an LS device. Although the standard TTL output configuration will drive the module, it may be desirable to use open-collector devices for the output module drivers.

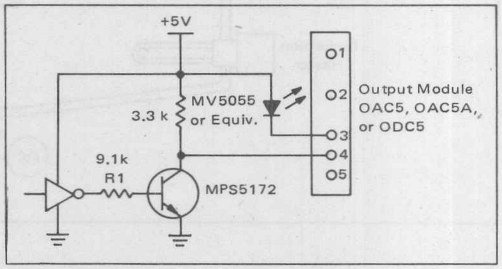
FIGURE 2 – Module Interface for TTL Logic



To interface the modules to MOS logic requires a bit more circuitry. The most obvious interface is to buffer the MOS with a TTL device. For most NMOS devices such as the M6800 family, a standard TTL device may be used. For CMOS operating at 5 volts, a low-power Schottky device may be used as the buffer.

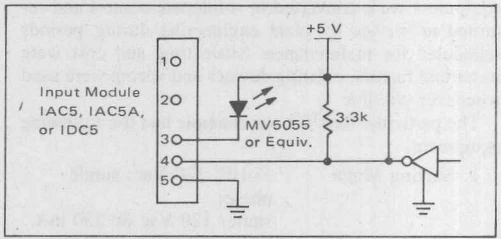
A second method of interfacing MOS devices to the I/O modules is use of a simple saturating transistor, as shown in Figure 3. Here the MOS device drives the base of an NPN transistor, which, in turn, drives the output module. By changing the value of R1 to 39 kΩ, this configuration may be used to interface CMOS operating at 15 volts with 15-volt logic modules OAC15, OAC15A, and ODC15.

FIGURE 3 – 5-Volt Interface for MOS Logic



Interfacing input modules to logic is a simple matter. Since the input modules are open-collector devices, the only additional component necessary is a pullup resistor, as shown in Figure 4.

FIGURE 4 – Interface to Logic Input

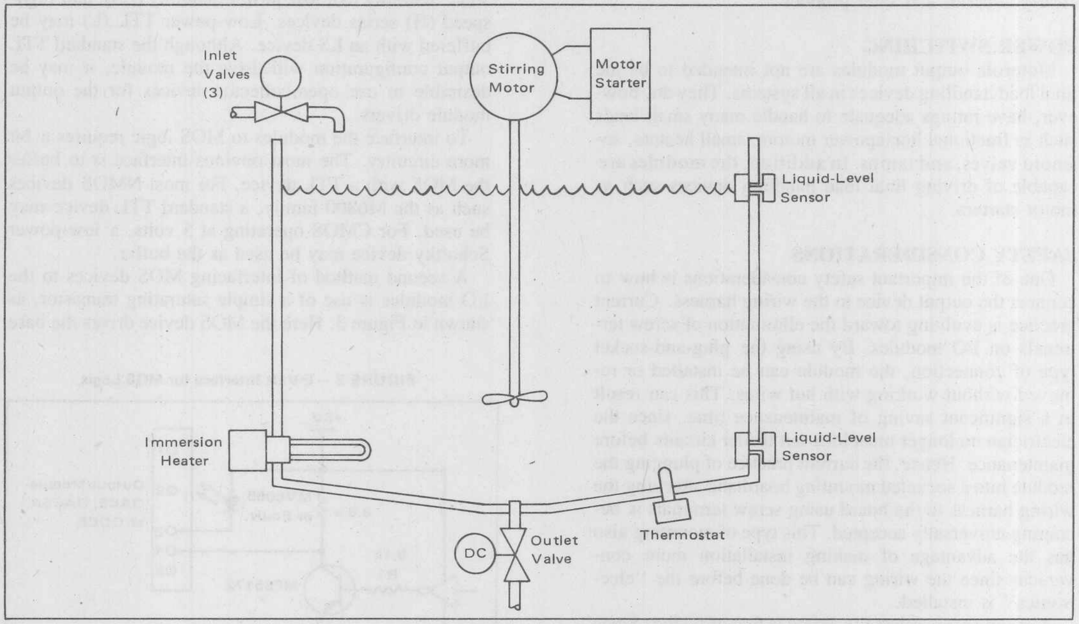


In all three illustrations, an indicator LED is added to indicate the status of the device. If a mounting board such as the MS16 is used, both the indicator LED and the 3.3 kΩ resistor are installed on the board at the factory.

TYPICAL APPLICATION

The application of I/O modules in an industrial environment can best be illustrated by working through a case history. The example problem involves a mixing tank in a batch processing plant. The tank involved, shown in Figure 5, is one of a number of similar tanks

FIGURE 5 – Mixing Tank with Makeup Heater



in a plant that batch-processes liquids through a number of mixing, stirring, and heating cycles. The original plant was controlled by relay logic and had high operating cost due to direct operating labor and maintenance expense. The goal of the conversion of the plant to a distributed processor-based control system was to increase flexibility while increasing reliability and reducing labor.

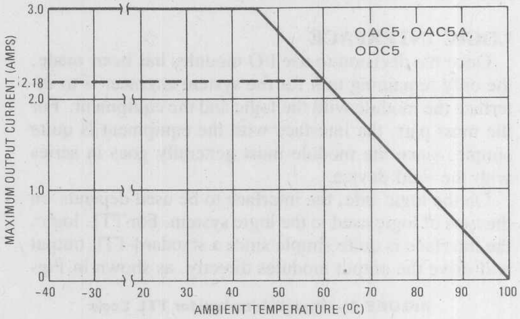
The conversion had to take place piecemeal to avoid shutting down the entire operation. Various pieces of equipment were converted to solid-state control and returned to service by plant engineering during periods scheduled for maintenance. Since time and cost were important factors, existing devices and wiring were used whenever possible.

The particular tank for our example had the following equipment:

1	Stirring Motor	¾-HP, 120 Vac, single phase; starter 120 Vac @ 250 mA
3	Inlet Valves	DC solenoid operated; 24 Vdc @ 1.2 A
1	Outlet Valve	DC solenoid operated; 24 Vdc @ 2.0 A
1	Immersion Heater	200 W resistive-coil type; 120 Vac @ 1.7 A
2	Liquid-Level Sensors	SPST N.C. contacts
1	Thermostat	SPST N.C. contacts

The new electronics housing may have internal temperatures up to 60°C maximum. Since the 60°C ambient temperature is above the maximum temperature for full ratings, the output modules were derated, using the derating curve shown in Figure 6. From the graph it was found that the output modules had a current rating of 2.18 A at 60° C. This rating was sufficient for all of the equipment used in this application.

FIGURE 6 – Maximum Output Current versus Temperature



Since the input modules are not power-handling devices, they do not have a derating factor and may be used over the specified temperature range without derating.

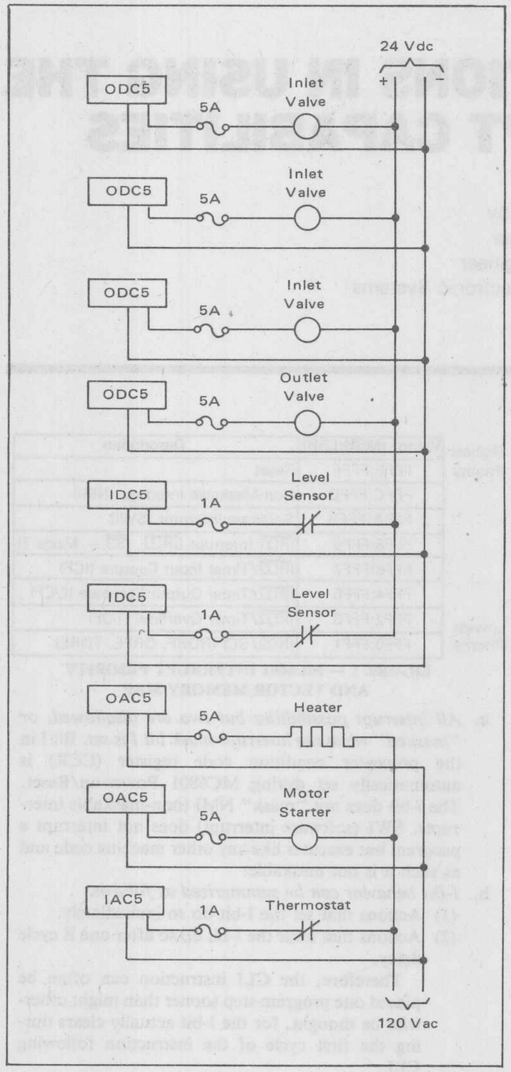
The microprocessor chosen for the application was an MC6800. The outputs were to be controlled through an MC6821 peripheral interface adapter. An SN74LS05 open-collector hex inverter was chosen to interface the

MC6821 and the output modules. The SN74LS05 drives the output modules as shown in Figure 1.

The logic outputs of the three input modules are connected to peripheral input pins of the MC6821 as shown in Figure 3.

The configuration for the wiring harness was fixed as shown in Figure 7. The thermostat could have been used to furnish either a dc signal, an ac signal, or to directly switch the 5-volt supply. Switching of the 5-volt supply would have limited the flexibility of the system and so was rejected. The ac signal was selected for the thermostat because it resulted in the least amount of rewiring, as it was previously used to switch ac relay signals. For similar reasons, the level switches were wired to the existing 24 Vdc "relay" supply, which was retained so that the existing solenoid valves could be utilized.

FIGURE 7 – Wiring Harness



The I/O modules were mounted on an MS16 mounting system. This system provides a complete means of connecting the I/O modules to the field wiring and the logic system, while keeping the I/O modules outside of the card cage which housed the microprocessor.

CONCLUSION

Motorola Input/Output modules can provide an economical means of connecting logic control to the "real-world" in many industrial applications. The use of this modular approach not only provides the necessary isolation but increases the safety, reliability, and serviceability of the system.

AN-807

SPECIAL CONSIDERATIONS IN USING THE MC6801 INTERRUPT CAPABILITIES

Prepared by
Clint Bauer
Systems Engineer
Motorola Automotive Electronic Systems

M6800 Microprocessor family components are used in numerous real-time control applications, many of which use external interrupt, timer and/or ACIA interface devices to increase system capability. The MC6801 microcomputer brings all these capabilities together with ROM and RAM on a single chip, while also providing a dramatically enhanced, yet machine-code compatible version of the MC6800 processor.

The MC6801 interrupt control methods are also enhanced, but still retain the same philosophy of operation used by other M6800 family components. The improvements increase performance, but also make possible several application-dependent constraints which merit consideration in certain systems. It is hoped that the information contained in this application note will aid the reader during his system design, so that a similar education is not required at debug time. It is assumed that the reader is familiar with basic operation of the MC6801 as described in the MC6801 Data Sheet and/or the MC6801 Manual. An optional review of MC6801 interrupt operation is provided first, followed by a discussion of important interrupt design constraints.

The MC6801 interrupt structure is similar to that available in the MC6800. The principle difference is that the MC6801 has four additional interrupt vectors and handshake logic to control them (see Figure 1). MC6800 systems are able to support external circuits that offer this same capability, but normally do so by sharing the single $\overline{\text{IRQ}}$ line and interrupt vector. The additional MC6801 vectors allow more efficient interrupt service by eliminating polling requirements for the triple-function timer/counter (where quick response is especially helpful), and reducing them elsewhere.

Having more vectors, MC6801 systems now offer a greater probability that near simultaneous interrupts will occur. For example, the three vector internal timer will often handle multiple asynchronous events. Therefore, it is important that the MC6801 system designer carefully observe the exact rules concerning interrupt recognition, entry, and service. A review of these rules is provided below.

	Vector (MSB:LSB)	Description
Highest Priority	FFFE:FFFF	Reset
	FFFC:FFFD	Non-Maskable Interrupt (NMI)
	FFFA:FFFB	Software Interrupt (SWI)
	FFF8:FFF9	$\overline{\text{IRQ1}}$ Interrupt ($\overline{\text{IRQ1}}$, $\overline{\text{IS3}}$ – Mode 7)
	FFF6:FFF7	$\overline{\text{IRQ2}}$/Timer Input Capture (ICF)
	FFF4:FFF5	$\overline{\text{IRQ2}}$/Timer Output Compare (OCF)
	FFF2:FFF3	$\overline{\text{IRQ2}}$/Timer Overflow (TOF)
Lowest Priority	FFF0:FFF1	$\overline{\text{IRQ2}}$/SCI (RDRF, ORFE, TDRE)

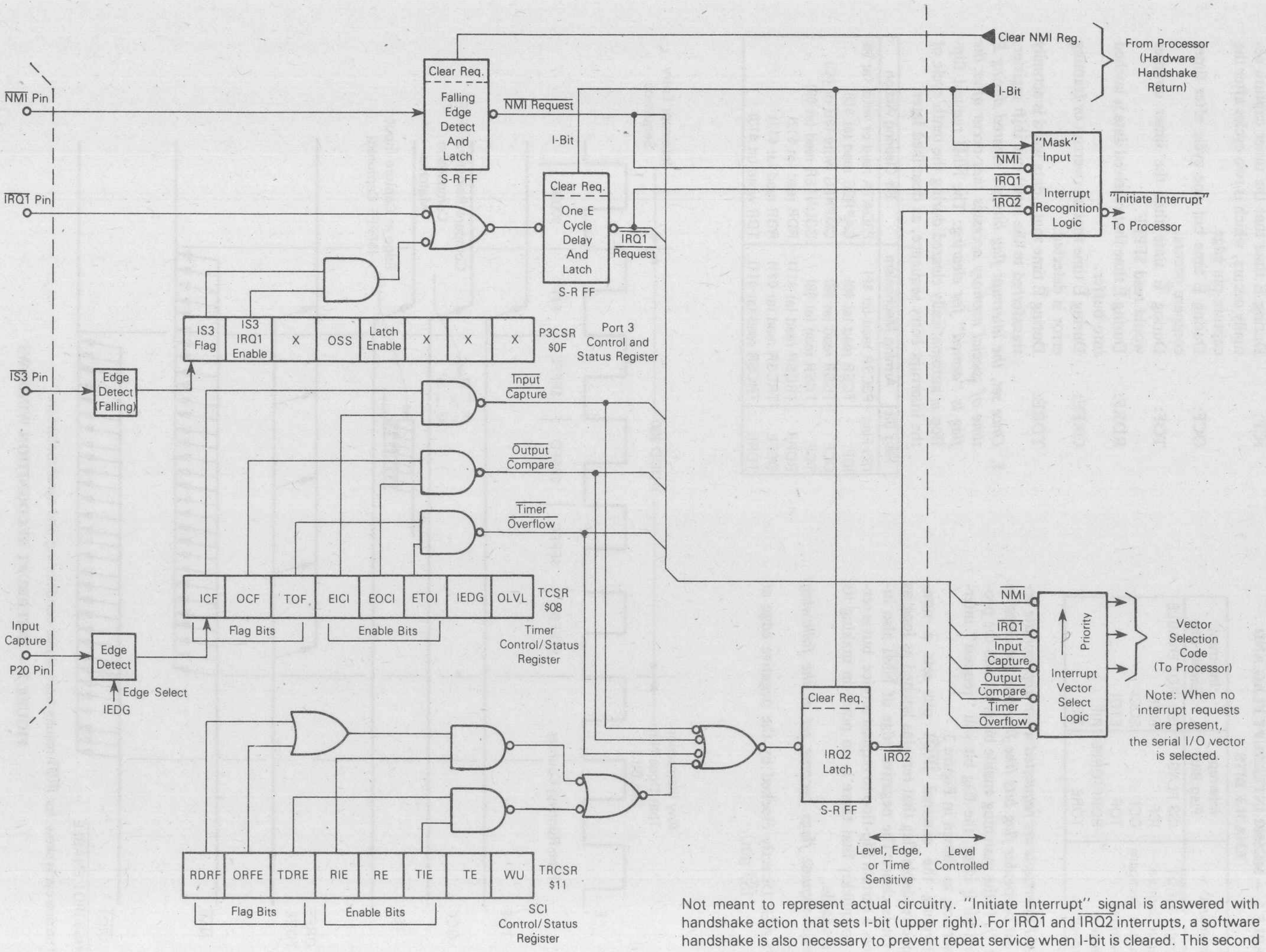
FIGURE 1 — MC6801 INTERRUPT PRIORITY AND VECTOR MEMORY MAP

a. *All interrupt possibilities but two are disallowed, or "masked" when the interrupt-mask bit I is set.* Bit I in the processor condition code register (CCR) is automatically set during MC6801 Power-up/Reset. The I-bit does not "mask" NMI (non-maskable interrupt). SWI (software interrupt) does not interrupt a program but executes like any other machine code and as such it is not maskable.

b. *I-Bit behavior can be summarized as follows:*
   (1) Actions that set the I-bit do so immediately.
   (2) Actions that clear the I-bit do so after one E cycle delay.

   Therefore, the CLI instruction can often be placed one program step sooner than might otherwise be thought, for the I-bit actually clears during the first cycle of the instruction following CLI.

c. *Most MC6801 interrupts can be inhibited at a second level.* Specific control bits in several MC6801 registers (see Figure 2) separately enable or disable the six interrupt possibilities shown in Table 1. All interrupt enable bits are cleared (disabled) during MC6801 Power-up/Reset. User programs can set or clear these bits, the action taking place during E time of an MPU write cycle to the specified register.



Not meant to represent actual circuitry. "Initiate Interrupt" signal is answered with handshake action that sets I-bit (upper right). For $\overline{\text{IRQ1}}$ and $\overline{\text{IRQ2}}$ interrupts, a software handshake is also necessary to prevent repeat service when I-bit is cleared. This second handshake clears the appropriate interrupt request directly or must indirectly cause line $\overline{\text{IRQ1}}$ to return to its inactive state.

FIGURE 2 — CONCEPTUAL REPRESENTATION OF MC6801 INTERRUPT STRUCTURE

TABLE 1 — MC6801 INTERRUPT FLAG AND ENABLE BITS

Interrupt	Interrupt Flag Bits	Interrupt Enable Bits
Input Strobe 3 $\overline{\text{IRQ1}}$	IS3 FLAG	IS3 IRQ1 ENABLE
Timer Input Capture	ICF	EICI
Timer Output Compare	OCF	EOCI
Timer Overflow	TOF	ETOI
Serial Receive	RDRF/ORFE	RIE
Serial Transmit	TDRE	TIE

d. *MC6801 interrupts are requested when appropriate actions set particular flag bits (the flag bits are listed in Table 1).* If the matching enable bit is set and the processor I-bit is clear, the flag bit will "request" interrupt service, as shown in Figure 3.

Activating the external $\overline{\text{IRQ1}}$ pin sets a non-machine-readable flag that remains latched as long as the I-bit is clear. The negative edge of $\overline{\text{NMI}}$ also influences a certain flip-flop to request service, but is serviced so quickly that there is no point in making its state readable.

e. *Interrupt request flags become set at the following times:*

IS3 FLAG: Directly clocked by the negative edge at $\overline{\text{IS3}}$ pin.

ICF: During $\overline{\text{E}}$ time that the timer capture actually occurs, which is two cycles after the capture pin edge.

OCF: During $\overline{\text{E}}$ time but one cycle after timer compare occurs.

TOF: During $\overline{\text{E}}$ time that the timer counter would read $FFFF.

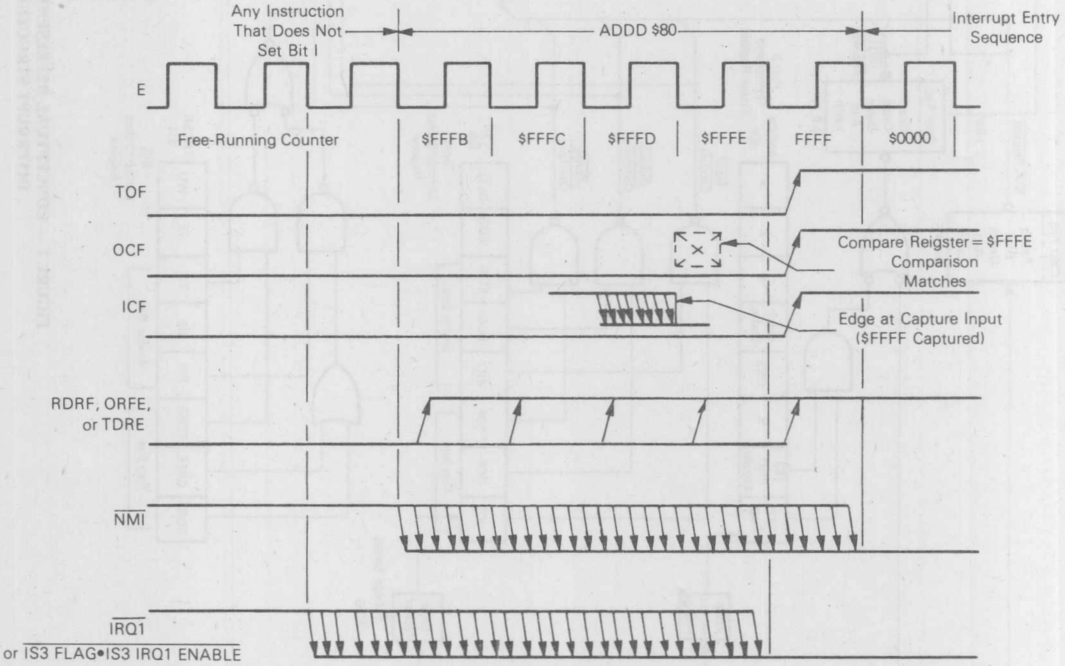
RDRF: During $\overline{\text{E}}$ time that received data is latched into buffer.

ORFE: During $\overline{\text{E}}$ time that an overrun or framing error is detected.

TDRE: During $\overline{\text{E}}$ time that a data word is actually transferred to the serial out shift register.

f. *Once set, the interrupt flag bits are cleared during E time of special memory accesses that occur after the flag is "armed" for clearing:* The $\overline{\text{NMI}}$ request flip-flop is automatically cleared during the tenth cycle of the interrupt entry sequence, as described later.

Flag Bit	Arming Mechanism	Bit Clearing Action
IS3 Flag	P3CSR read (at $F)	P3DATA read or write (at $6)
ICF	TCSR read (at $8)	CAPREG read (at $D)
OCF	TCSR read (at $8)	CMPREG write ($B or $C)
TOF	TCSR read (at $8)	COUNTER read (at $9)
RDRF	TRCSR read (at $11)	RDR read (at $12)
ORFE	TRCSR read (at $11)	RDR read (at $12)
TDRE	TRCSR read (at $11)	TDR write (at $13)



The one cycle skew for $\overline{\text{IRQ1}}$ results from signal conditioning and synchronization.

FIGURE 3 — INTERRUPT RECOGNITION WINDOWS

g. *Regardless of how interrupts are caused, the end interface between each interrupt request and the processor is level controlled, as shown in Figure 2 (as the Interrupt Vector Select Logic block).* This feature gives an MC6801 program more control over interrupt service than is otherwise possible. For example, if the three timer interrupts were enabled and their flags were to simultaneously set, the input capture interrupt (having the highest priority of the three) would be serviced first. This service routine could temporarily inhibit compare interrupt service by clearing bit EOCI, which allows overflow interrupt service (normally the lower priority) to occur when capture service is complete. If the end interrupt request interface was latch rather than level controlled, clearing bit EOCI in the example would not prevent the compare interrupt from being serviced before timer overflow.

Individual flag bits are separately latched, however. In the example just given, bit OCF is temporarily inhibited but will indeed be serviced when the program restores bit EOCI to its enable state.

h. *Interrupt requests trigger interrupt service at times well defined relative to the end of the instruction in progress, as shown in Figure 3.*

i. *After recognition, all interrupts are initiated by a twelve-cycle interrupt entry sequence (see Figure 4).* The particular request that initiates the interrupt entry sequence will normally be, but is not always, the same one immediately serviced. Exceptions can occur where two or more interrupts occur at nearly the same time, because actual selection of which interrupt to service is delayed until near the end of the resulting interrupt entry sequence. At the ninth cycle, a decision is made as to whether $\overline{\text{NMI}}$, $\overline{\text{IRQ1}}$, or $\overline{\text{IRQ2}}$ will be serviced. If $\overline{\text{IRQ2}}$ is selected, the exact selection of which $\overline{\text{IRQ2}}$ to service is made during the tenth cycle. Requests not selected remain pending but are masked (I-bit sets during the tenth cycle), allowing the selected service routine to proceed undisturbed. Some example patterns of near-coincidental interrupt service are shown in Figure 5.

j. *Interrupt service is complete when the processor executes an RTI instruction.* This ten cycle instruction simply returns seven bytes from the stack to the processor registers, restoring the original machine state present when the interrupt was serviced (assuming the interrupt routine does not modify stack contents). In particular, the original I-bit is restored. If it returns to a logic "0", the $\overline{\text{IRQ1}}$ and $\overline{\text{IRQ2}}$ latches of Figure 2 are again enabled so that any pending request can be serviced.

k. *A CLI instruction can be executed during interrupt service to allow prompt processor response to pending $\overline{\text{IRQ1}}$ or $\overline{\text{IRQ2}}$ requests.* The benefits gained by this are sometimes offset by increased program complexity and greater required stack depth.

l. *All interrupt service routines (except $\overline{\text{NMI}}$ and SWI) should take action that removes its interrupt request prior to executing an RTI instruction.*

An $\overline{\text{IRQ2}}$ or $\overline{\text{IS3}}$ or $\overline{\text{IRQ1}}$ request is normally removed by clearing the appropriate flag bit. As an alternative, the matching enable bit can be cleared. External hardware must remove any external $\overline{\text{IRQ1}}$ interrupt requests, as this line is not directly controlled by the processor. This is best handled by providing handshake logic similar to that used internally to control the $\overline{\text{IRQ2}}$ requests. The MC6821 PIA and MC6846 RIOT devices each provide an excellent $\overline{\text{IRQ1}}$ interface, though discrete logic designs will also work.

m. Interrupt service cycle times are well defined:

C_{serv}: Number of cycles taken away from non-interrupt execution by interrupt execution.

C_{entry}: 12 cycles to enter interrupt service.

C_{clrflg}: 4 to 9 cycles to clear interrupt request (zero for NMI or SWI)

C_{task}: number of cycles to perform desired service.

C_{exit}: 10 cycles to execute RTI instruction.

$$C_{\overline{\text{IRQ1}},\ \text{IRQ2}} = C_{task} + 26 \text{ to } 31 \text{ cycles}$$

$$C_{\overline{\text{NMI}},\ \text{SWI}} = C_{task} + 22 \text{ cycles}$$

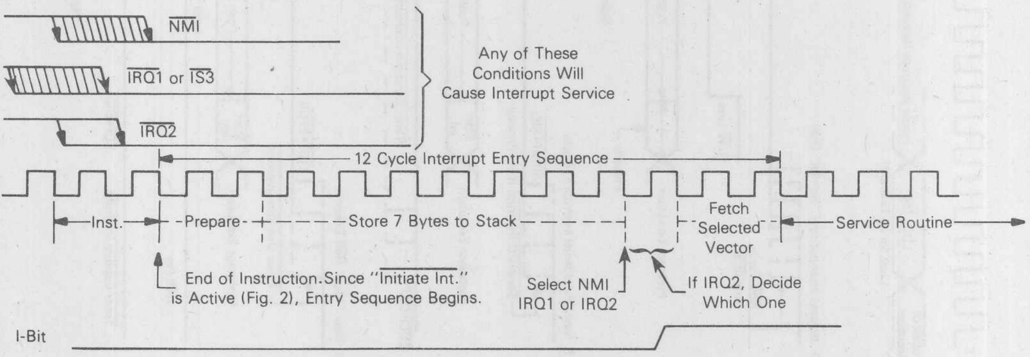
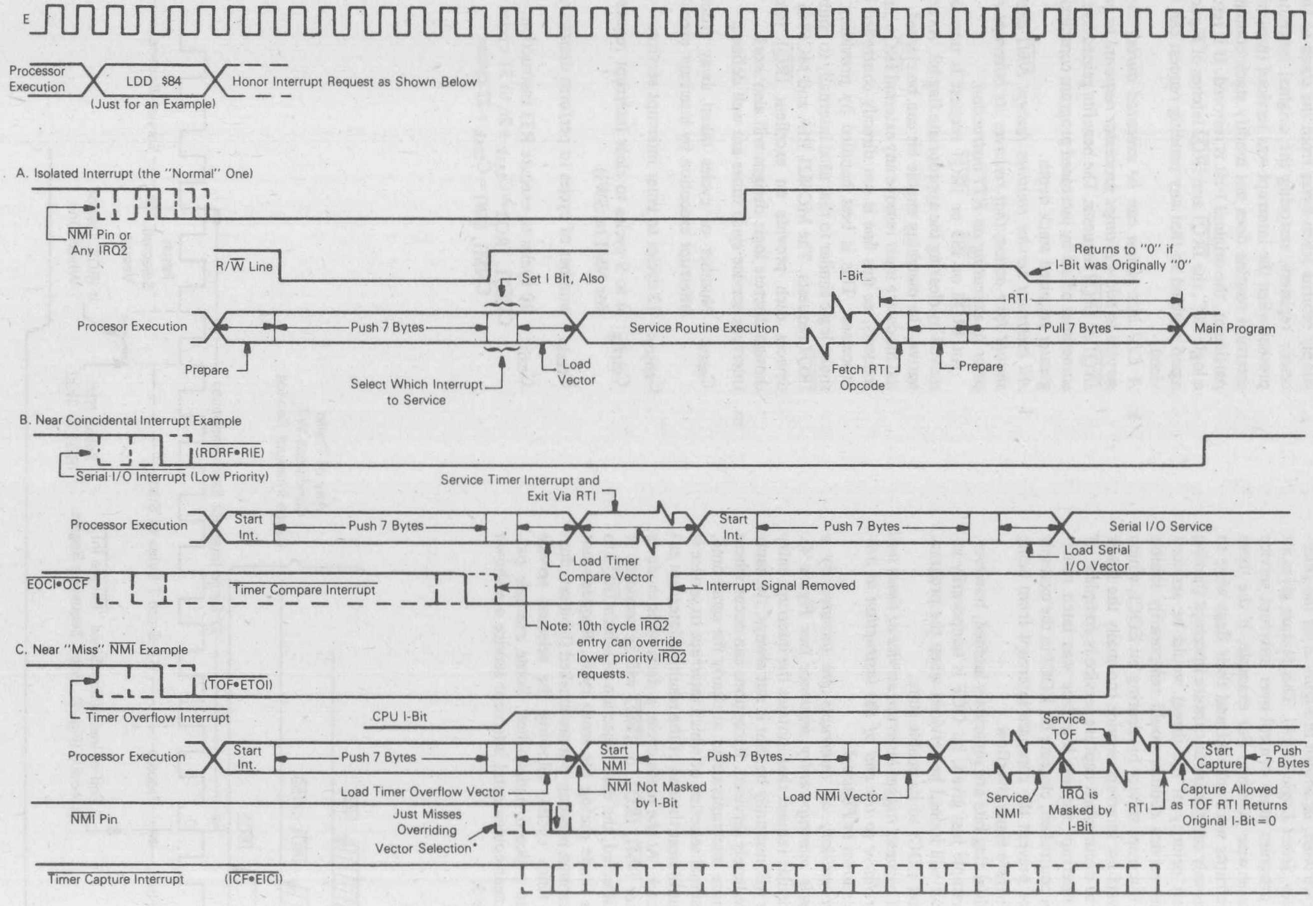

NMI
IRQ1 or IS3
IRQ2
Any of These Conditions Will Cause Interrupt Service
12 Cycle Interrupt Entry Sequence
E
Inst.
Prepare
Store 7 Bytes to Stack
Fetch Selected Vector
Service Routine
End of Instruction. Since "Initiate Int." is Active (Fig. 2), Entry Sequence Begins.
Select NMI IRQ1 or IRQ2
If IRQ2, Decide Which One
I-Bit

FIGURE 4 — MC6801 INTERRUPT ENTRY SEQUENCE



*Note: NMI or IRQ1 requests must be present by end of 9th cycle to participate in vector selection.

FIGURE 5 — MC6801 INTERRUPT EXAMPLES

DESIGN CONSTRAINTS OF THE MC6801 INTERRUPT SYSTEM

The expanded interrupt system of the MC6801 offers important benefits when software is constructed to utilize it properly. However, certain specific software practices should be avoided because unexpected program behavior may result. These practices are now described, along with alternatives that will aid in better achieving the desired results.

AVOID $\overline{\text{IRQ2}}$ HANDSHAKE VIOLATIONS

MC6801 interrupt requests (except for NMI and SWI) are cleared with a software handshake during interrupt service to avoid repetitive service of the same interrupt. The programmer should avoid several improper procedures that can clear these requests at the wrong time, or several difficulties may occur that can cause unexpected system performance.

What happens if an $\overline{\text{IRQ2}}$ request is somehow removed prior to actual service (a handshake violation)? If the request had not yet triggered an interrupt entry sequence, nothing unusual takes place. If indeed triggered, however, the following rule will apply: *An $\overline{\text{IRQ2}}$ interrupt entry sequence that finds no request present during its tenth cycle will always select the serial I/O vector for service.* This may or may not be a problem if the original request was for serial I/O service. On the other hand, programs that allow $\overline{\text{IRQ2}}$ requests to be cleared between interrupt sequence triggering and actual vector selection will service the serial I/O vector in lieu of that desired. Two methods exist that allow this to occur, which are described below and then summarized in Table 2.

1. *Clearing $\overline{\text{IRQ2}}$ Enable Bits While I-bit is Clear* — Programs are often structured such that mask-bit I is clear during background or non-interrupt execution. Some programs will also purposely clear the I-bit during interrupt service routines. At either time, software that clears an $\overline{\text{IRQ2}}$ enable bit should be avoided because the corresponding interrupt flag may have just become set. Figure 6 shows that an $\overline{\text{IRQ2}}$ interrupt only momentarily requested can result in erroneous selection of the serial I/O vector. To prevent this, use instruction SEI to mask all interrupt requests for the short time that it takes to clear the desired enable bit, then clear the I-bit again with instruction CLI. The SEI/CLI combination is unnecessary when the programmer knows that the I-bit is already set, as is usually true within interrupt service routines that do not themselves alter the I-bit.
2. *Clearing Enabled $\overline{\text{IRQ2}}$ Flag Bits while I-bit is Clear* — $\overline{\text{IRQ2}}$ requests can also be removed by clearing the interrupt-flag itself. Doing so just as the interrupt is to be serviced should be avoided to prevent improper serial I/O vector selection, as demonstrated in Figure 7a.

   Two special cases of programming practice can also generate this undesirable result. The double-byte read instructions "LDD TCSR" ($8) and "LDD TRCSR" ($11) are used to arm and clear interrupt flags TOF, RDRF, and ORFE. As such, they are excellent for use as the software handshake needed during service of these flags, but altogether improper any time their interrupts are enabled and the I-bit is clear.

   For example, TOF might set, arm, and clear within the four cycles of "LDD TCSR" execution. Though the request is removed, it is still able to initiate an interrupt entry sequence, resulting in erroneous service of the serial I/O routine (see Figure 7b). Good programming practice would clear interrupt flags only during the appropriate service routine, which is the best solution to this difficulty. "LDD TRCSR" can similarly clear RDRF and/or ORFE while simultaneously initiating an interrupt sequence. Again, the serial I/O vector is selected, which is seemingly proper in this special case. However, the serial interrupt service routine normally polls flags RDRF, ORFE, and TDRE to determine the actual interrupt source. It is possible, then, that RDRF or ORFE service be skipped due to improper flag-clearing.

Table 2 summarizes the several methods by which the serial I/O vector may be improperly selected.

TABLE 2 — METHODS OF GENERATING IMPROPER SERIAL I/O VECTOR SELECTION

The Cause	Control or Flag Bits Affected	The Solution
Clearing IRQ2 enable bit just as interrupt entry sequence begins.	EICI EOCI ETOI TIE RIE	Disable these enable bits only while I-bit is set.
Clearing IRQ2 flags just as interrupt entry sequence begins	All IRQ2 Flags	Do not clear flags directly after CLI instruction.
	TOF RDRF ORFE	Execute these instructions only if I-bit is set. LDD TCSR LDD TRCSR LDX TCSR LDX TRCSR ADDD TCSR ADDD TRCSR SUBD TCSR SUBD TRCSR CPX TCSR CPX TRCSR LDS TCSR LDS TRCSR

AVOID IRQ1 HANDSHAKE VIOLATIONS

IRQ1 requests are latched as long as the I-bit is clear (see Figure 2) and will not cause improper selection of the serial I/O vector. However, it is still wise to observe the precautions described for IRQ2 to prevent any unexpected system performance. For example, handshake violations can clear IRQ1 request flags just as interrupt service is being initiated. As with IRQ2, programmers should avoid clearing IRQ1 flags during an instruction that follows CLI. Any of the "LDD-type" violations described previously should also be avoided any time the I-bit is clear, for IRQ1 flags can also set, arm and clear during a single instruction. These violations allow IRQ1 service to take place, but prevent recognition of the calling flag during IRQ1 polling.

Additionally, the MC6821 and MC6850 offer interrupt request flags that need not be "armed" before clearing — a single memory access does the job. Therefore, limit these accesses to the appropriate service routine so that no request `can be missed.

There is no hardware oriented reason to avoid clearing IRQ1 interrupt enable bits while the I-bit is clear. However, a polling routine cannot reliably test both flag and enable bits when this is the case.

Pulsing the external IRQ1 line by any form of signal generator without a handshake should normally be avoided.* Edge triggered interrupt lines NMI, IS3, and Input Capture are better used for such signals. Or, an MC6821 or MC6846 can transform these into level-sensitive, handshake controlled request signals which are more suitable for IRQ1.

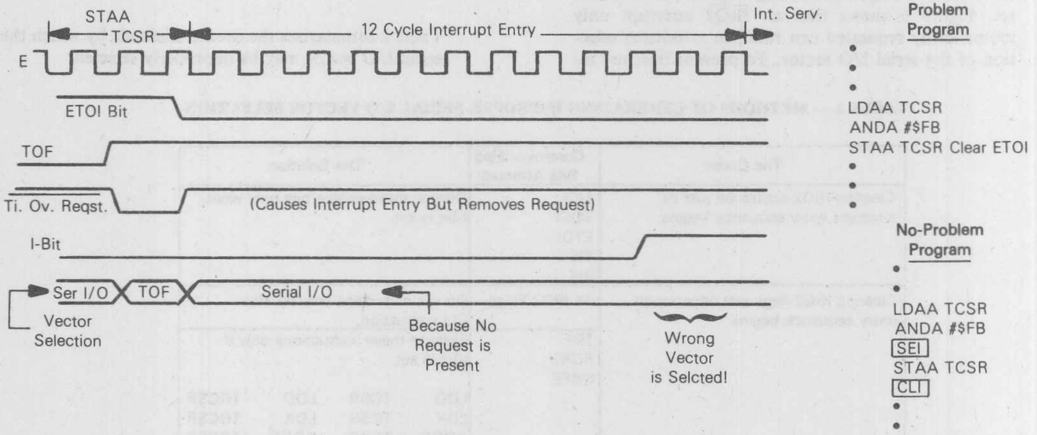
AVOID CLEARING THE I-BIT DURING NMI SERVICE

There is need to be cautious about clearing the I-bit during NMI service because this interrupt can occur at virtually any point in program execution. Some programs that use this technique are likely to service occasional IRQ1 or IRQ2 interrupts twice per request.

Double service occurs whenever an I-bit clearing NMI service routine is executed before the flag-clearing handshake of an already entered IRQ1 or IRQ2 service routine. For example, Figure 8 shows that an NMI occurrence during a particular window of time prevents the quick handshake that clears ICF. When NMI service executes instruction CLI, flag ICF teams with enable bit EICI to again request capture service. As shown, all routines will execute properly and to completion, including *double* service of the twice-called capture routine.

Clearing the I-bit during other service routines will not generate this situation, although doing so before clearing the calling interrupt request is disastrous. The best way to avoid any problem is to leave the I-bit set throughout NMI service. Where this is undesirable, additional software can be added to the NMI routine stack and compare it to all possibilities that lead to double service. Where such is indicated, clearing the I-bit should be skipped. If the I-bit must be cleared every time, additional software should first clear the interrupt flag scheduled for double service. Clearly, the benefits desired when clearing the I-bit during NMI service are potentially offset by the added software required to support this technique. For the same reasons, do not program an NMI interrupt service routine to clear the I-bit record contained on its stack. This would allow all portions of a program to be subjected to I-bit clear execution, resulting in potential double service of interrupts.

*Appendix A offers an application of IRQ1 pulsing that does work, but only under special circumstances.



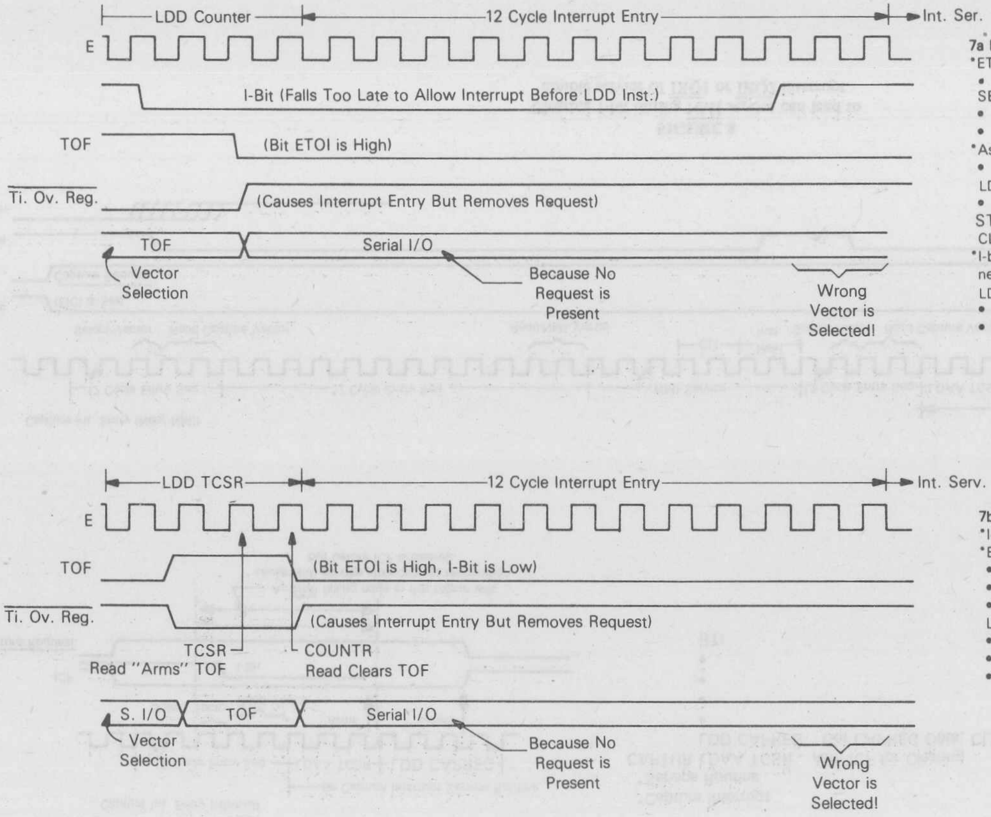
Problem Program

```
•
•
LDAA TCSR
ANDA #$FB
STAA TCSR Clear ETOI
•
•
•
```

No-Problem Program

```
•
•
LDAA TCSR
ANDA #$FB
SEI
STAA TCSR
CLI
•
•
```

FIGURE 6
Programs that clear IRQ2 enable bits while I-bit is clear risk improper vector selection.


LDD Counter
12 Cycle Interrupt Entry
Int. Ser.
E
I-Bit (Falls Too Late to Allow Interrupt Before LDD Inst.)
TOF
(Bit ETOI is High)
Ti. Ov. Reg.
(Causes Interrupt Entry But Removes Request)
TOF
Serial I/O
Vector Selection
Because No Request is Present
Wrong Vector is Selected!
LDD TCSR
12 Cycle Interrupt Entry
Int. Serv.
E
TOF
(Bit ETOI is High, I-Bit is Low)
Ti. Ov. Reg.
(Causes Interrupt Entry But Removes Request)
TCSR Read "Arms" TOF
COUNTR Read Clears TOF
S. I/O
TOF
Serial I/O
Vector Selection
Because No Request is Present
Wrong Vector is Selected!

7a Problem Program

```
*ETOI bit is set
 •
 SEI   Mask Interrupts
 •
 •
*Assume that TOF sets in here
 •
 LDAA TCSR  arms TOF
 •
 STAB $80
 CLI
*I-bit clears in 1st cycle of
 next instruction
 LDD COUNTR  clears TOF
 •
 •
```

No-Problem Program

```
*ETOI bit is set
 •
 SEI
 •
*TOF sets here
 •
*LDAA TCSR
 •
*CLI       } or: STAB $80
 STAB $80  }      CLI
                  NOP
*Timer Overflow Interrupts
 LDD COUNTR
```

7b Problem Program

```
*I-Bit is clear
*ETOI bit is set
 •
 •
 •
 LDD TCSR
 •
 •
 •
```

No-Problem Program

Best Solution: Do not write code that allows flag clearing outside of interrupt service routine!

Poor Alternatives:

```
1. •
   •
   LDAA TCSR
   LDAB COUNTR
   •
   •
2. •
   •
   SEI
   LDD TCSR
   CLI
   •
   •
```

FIGURE 7
Programs that clear "enabled" IRQ2 flag bits while I-bit is clear risk improper vector selection.

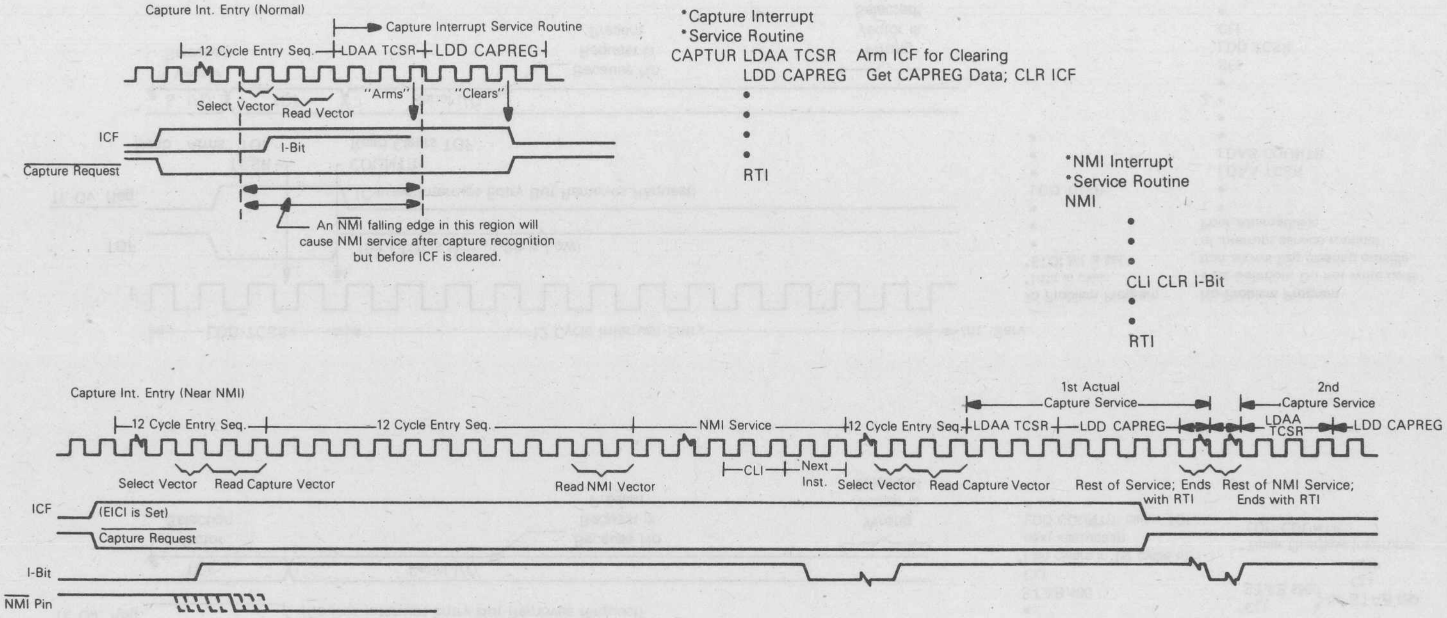


FIGURE 8
Clearing I-bit during NMI service can lead to double service of IRQ1 or IRQ2 interrupt.

APPENDIX A
USING IRQ1 AS AN INPUT PIN

If the circumstances are right, an I/O limited MC6801 system may be able to use the $\overline{\text{IRQ1}}$ pin as an extra input. Where no other interrupts are used, this can be accomplished with the simple program of Figure A-1 to clear RAM byte IRQTST while also clearing the I-bit. The state of the $\overline{\text{IRQ1}}$ pin then determines whether IRQTST will be changed (interrupt occurs) or remain constant (no interrupt occurs). The background program discovers which is the case by simply reading IRQTST. Notice that the processor has no control of input pin $\overline{\text{IRQ1}}$ in this method, but can still perform the necessary interrupt handshake by setting the I-bit record stored on the stack. This prevents repeat service that would otherwise tie up the processor as long as the $\overline{\text{IRQ1}}$ pin is held low.

The same basic method can also be used when other interrupts are to be serviced as well. The $\overline{\text{IRQ1}}$ pin is again tested in the manner just described, but now routine IRQSRV must also poll other interrupt requests in case they need service, as shown in Figure A-2. If it is important that the various interrupts be serviced promptly, the programmer can scatter CLI instructions through his background software. This still allows the $\overline{\text{IRQ1}}$ pin to be used as an input, and also permits normal service of all interrupts while $\overline{\text{IRQ1}}$ is high. Whenever $\overline{\text{IRQ1}}$ is low, IRQSRV becomes an alternate entry path for other maskable interrupt requests.

```
          0080  A IRQTST EQU    $80       RAM BYTE AT $0080
                  *BACKGROUND PROGRAM, I-BIT IS SET.
2000                     ORG    $2000
          2000  A BKGRND EQU    *
                  *      .
                  *FIND OUT IF IRQ1 PIN IS HIGH OR LOW
2000  0E                 CLI              I CLRS DURING NEXT INST.
2001  7F  0080  A        CLR    IRQTST    WILL IT STAY ZERO?
                  *IF IRQ1 IS LOW, SERVICE OCCURS AT THIS MOMENT
2004  96  80    A        LDAA   IRQTST    IS NOW #$FF IF IRQ1 IS LOW
2006  26  00 2008        BNE    IRQLOW    IRQ1 DID OCCUR
                  *IRQ1 PIN WAS HIGH
                  *      .
          2008  A IRQLOW EQU    *         IRQ1 PIN WAS LOW
                  *      .
2008  7E  2000  A        JMP    BKGRND    END OF BACKGROUND LOOP


                  *IRQ1 SERVICE ROUTINE
200B  73  0080  A IRQSRV COM    IRQTST    CHANGE IRQTST!
200E  30                 TSX              X=SP+1
200F  A6  00    A        LDAA   0,X       THE CCR BYTE ON STACK
2011  8A  10    A        ORAA   #$10      SET I-BIT FOR RETURN
2013  A7  00    A        STAA   0,X
2015  3B                 RTI
                  *I-BIT IS SET TO PREVENT IRQ1 RESERVICE
```

FIGURE A-1
Using $\overline{\text{IRQ1}}$ as an input pin.

```
          0080  A TCSR   EQU    $8         TIMER C/S REGISTER
          000D  A CAPREG EQU    $D         CAPTURE REGISTER
          0011  A TRCSR  EQU    $11        TX/RX C/S REGISTER
          0012  A RXBUF  EQU    $12        RECEIVE BUFFER
          0080  A IRQTST EQU    $80        RAM BYTE AT $0080
2000                     ORG    $2000
                  *IRQ1 SERVICE ROUTINE
2000  73  0080  A IRQSRV COM    IRQTST     CHANGE IRQTST!
2003  30                 TSX               X=SP+1
2004  A6  00    A        LDAA   0,X        THE CCR BYTE ON STACK
2006  8A  10    A        ORAA   #$10       SET I-BIT FOR RETURN
2008  A7  00    A        STAA   0,X
                  *BEFORE RTI, SEE IF OTHER INTERRUPTS ARE PENDING
200A  96  08    A        LDAA   TCSR       CHECK INPUT CAPTURE
200C  2B  09 2017        BMI    TIMIC1     TIMER INPUT CAPTURE PENDING
200E  DC  11    A        LDD    TRCSR      CHECK SCI IRQ2 REQUESTS
2010  85  E0    A        BITA   #$E0       CHK RDRF,ORFE,TDRE FLAGS
2012  26  08 201C        BNE    SCIIN2     SERVICE SCI INTERRUPT
2014  3B                 RTI               EXIT:NO INTERRUPTS PENDING
                  *I-BIT IS SET TO PREVENT IRQ1 RESERVICE

2015  96  08    A TIMIC  LDAA   TCSR       ARM ICF FOR CLEARING
2017  DC  0D    A TIMIC1 LDD    CAPREG     CLR ICF, GET CAPTURE DATA
                  *             .
2019  3B                 RTI

201A  DC  11    A SCIINT LDD    TRCS       ACCA=TRCS, ACCB=RXBUF
201C  48          SCIIN2 ASLA              SORT OUT SCI FLAGS
                  *             .
201D  3B                 RTI
```

FIGURE A-2
Routine IRQSRV can also poll other interrupt requests when using $\overline{\text{IRQ1}}$ as an input.

AN-808

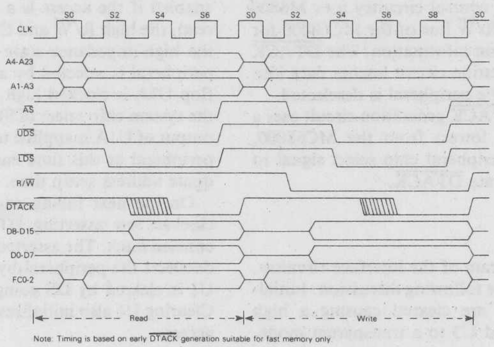
INTERFACING M6800 PERIPHERAL DEVICES TO THE MC68000 ASYNCHRONOUSLY

Prepared by:
Arnold J. Morales
Microprocessor Applications Engineer

This application note describes a technique for interfacing M6800 peripheral devices to a MC68000 microprocessor using a four-chip TTL circuit. Any M6800 peripheral is easily interfaced to the MC68000 using the M6800 peripheral control interface (E, $\overline{\text{VMA}}$, $\overline{\text{VPA}}$) that is designed into the MC68000. However, when using this interface, the peripheral must be driven by the MC68000 enable (E) signal. The frequency of this clock is one-tenth of the MC68000 clock frequency with a 60/40 (6 clocks high, 4 clocks low) duty cycle. Certain applications may require a clock frequency other than the one-tenth sample that is readily available. An application using a MC68B54 Advanced Data Link Controller (ADLC) at a high data transfer rate could require an E clock frequency of up to two megahertz because the data transfer rate of the ADLC depends on the transmit and receive clocks which are limited by the E clock frequency.

TIMING CONSIDERATIONS

Typical read and write timing for the MC68000 is shown in Figure 1. The relationship between the MC68000 timing and the access timing for the interface circuit given in this application is shown in Figure 2. The best case timing has data strobe occurring with the minimum setup time to allow peripheral selection on the next falling edge of the E clock. In the worst case timing, the data strobe did not occur in time to allow peripheral selection on the next falling edge of E. Therefore, a full E cycle has to occur and then the peripheral selection is done on the falling edge of that full cycle. The resulting cycle times for these best and worst cases and a comparison between asynchronous and synchronous interfacing is summarized in Table 1.



Note: Timing is based on early $\overline{\text{DTACK}}$ generation suitable for fast memory only.

Figure 1. MC68000 Read and Write Cycle Timing Diagram

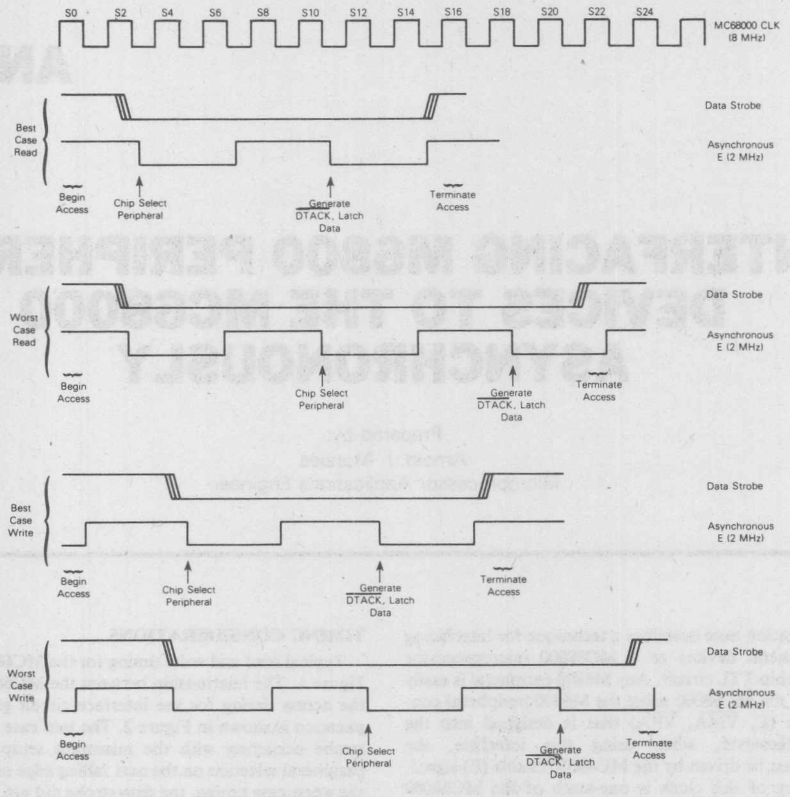


Figure 2. Asynchronous Interface Access Timing Diagrams

BLOCK DIAGRAM

Figure 3 is a block diagram of a circuit that allows M6800 peripherals, operating at any frequency within their operating range, to be asynchronously interfaced to a MC68000 processor. The data bus is driven by a pair of octal transparent latches. The latch control circuitry uses M6800 peripheral chip select and the R/$\overline{\text{W}}$ line of the MC68000 for output enable and data direction information. The $\overline{\text{DTACK}}$ signal from the $\overline{\text{DTACK}}$ generation circuit latches data into the enabled octal latch when the peripheral is deselected.

The peripheral select and $\overline{\text{DTACK}}$ generation circuit uses a data strobe (either upper or lower) from the MC68000, peripheral E, and a M6800 peripheral chip select signal to select the peripheral and generate $\overline{\text{DTACK}}$.

CIRCUIT OPERATION

Figure 4 is a schematic diagram of the interface circuitry. Refer to this diagram during the following discussion. Initially flip flops U1A and U1B are cleared causing a high $\overline{\text{DTACK}}$ output setting U2 and U3 to a transparent mode. Latch U2 is in the high-impedance state due to a high on the output enable ($\overline{\text{OE}}$) input. Latch U3 is enabled due to a low on the $\overline{\text{OE}}$ input.

At the start of a M6800 peripheral access, latch U3 remains enabled if the access is a MC68000 write. If the access is a read, the high R/$\overline{\text{W}}$ and CS inputs to U4A cause U3 to go to the high-impedance state and U2 to become enabled. The peripheral is selected by a low chip select prime ($\overline{\text{CS}}$'). Flip flop U1A is clocked high on the first falling edge of E with the system chip select (CS) and data strobe (DS) high. The Q output of U1A is applied to U4D, asserting $\overline{\text{CS}}$'. Selecting the peripheral at this time ensures that the peripheral has adequate address setup time.

On the next falling edge of E, the $\overline{\text{Q}}$ output of U1B is clocked low asserting $\overline{\text{DTACK}}$ and latching data into the enabled latch. The asserted $\overline{\text{DTACK}}$ signal, inverted by U4D, deselects the peripheral by causing $\overline{\text{CS}}$' to go high. Flip flop U1 is cleared by DS going low when the access terminates. Clearing U1 also initializes the interface circuitry for the next access.

Table 1. Synchronous and Asynchronous Interface Access Time

	Read Access Times (MC68000 Cycles)		Write Access Times (MC68000 Cycles)	
	Best Case	Worst Case	Best Case	Worst Case
Synchronous	9	18	9	18
Asynchronous	8	11	9	12

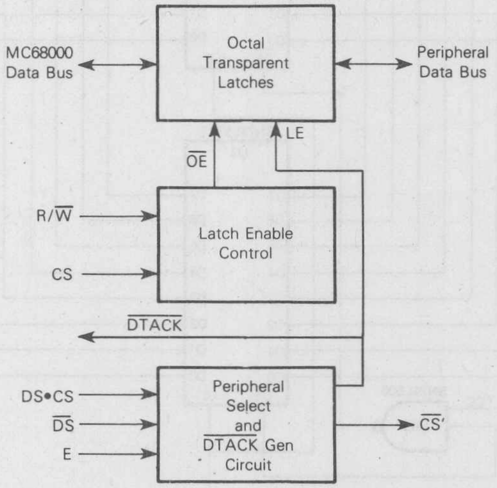


Figure 3. M6800 Peripheral to MC68000 Interface — Block Diagram

SAMPLE CIRCUIT

An example of this interface circuitry is given in the following paragraphs. This example illustrates how the MC68000 can be interfaced to both a MC6854 Advanced Data Link Controller (ADLC) and a MC6840 Programmable Timer Module (PTM) at the same time. The circuit shown in Figure 5 uses the two megahertz "B" version parts connected to a MC68000 driven at eight megahertz.

The base addresses for the peripherals in this example are $18001 for the ADLC and $18801 for the PTM. When the MC68000 transfers bytes it asserts the upper data strobe for even addresses and the lower data strobe for odd addresses. The circuit in this example uses the lower data strobe; therefore only odd MC68000 addresses are used. A memory map of the example system is given in Figure 6.

Device Selection — A SN74LS138 1-of-8 decoder (U5) used as an address decoder is used in conjunction with a chip select signal (CS') developed by U6E to select either the ADLC or the PTM. The PTM requires two chip select inputs, one high and one low, to be selected. The low input is provided by the $\overline{O5}$ output of U5 while the high input is provided by an inverted sample of the $\overline{CS}$', developed by U6E.

To select the ADLC, the $\overline{O5}$ output of U5 must be high and the $\overline{O1}$ output must be low. The ANDing of $\overline{O5}$ with the high CS' developed by U6E generates the low chip select input required by the ADLC.

Test Program — A flow chart of the test program is given in Figure 7 and a listing is provided in Figure 8. Refer to these figures during the following discussions. The first five lines of code initiate operation of timer 3 in the PTM in the continuous mode, resulting in a square wave at the output of timer 3, pin 6. The remaining lines of code are for testing the ADLC. The test program is based on the loopback test program given in Motorola publication MC6854UM (AD).

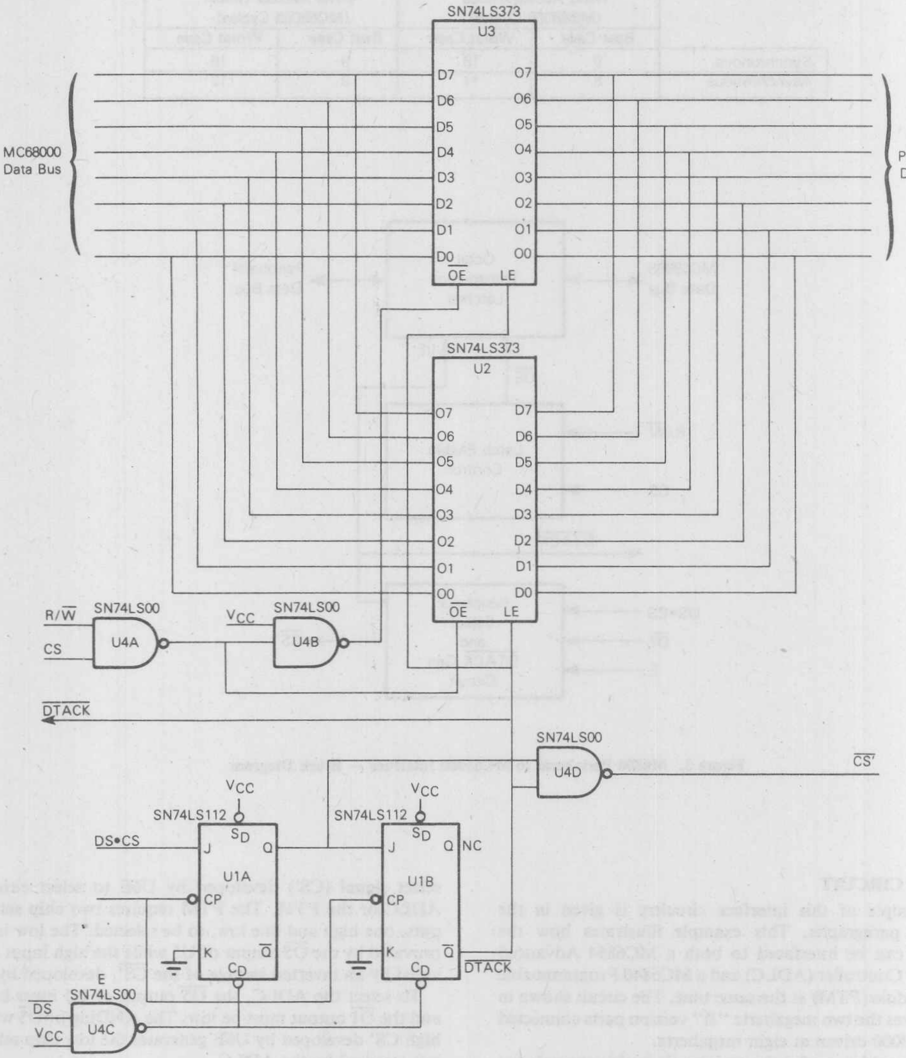


Figure 4. M6800 Peripheral to MC68000 Interface — Schematic Diagram

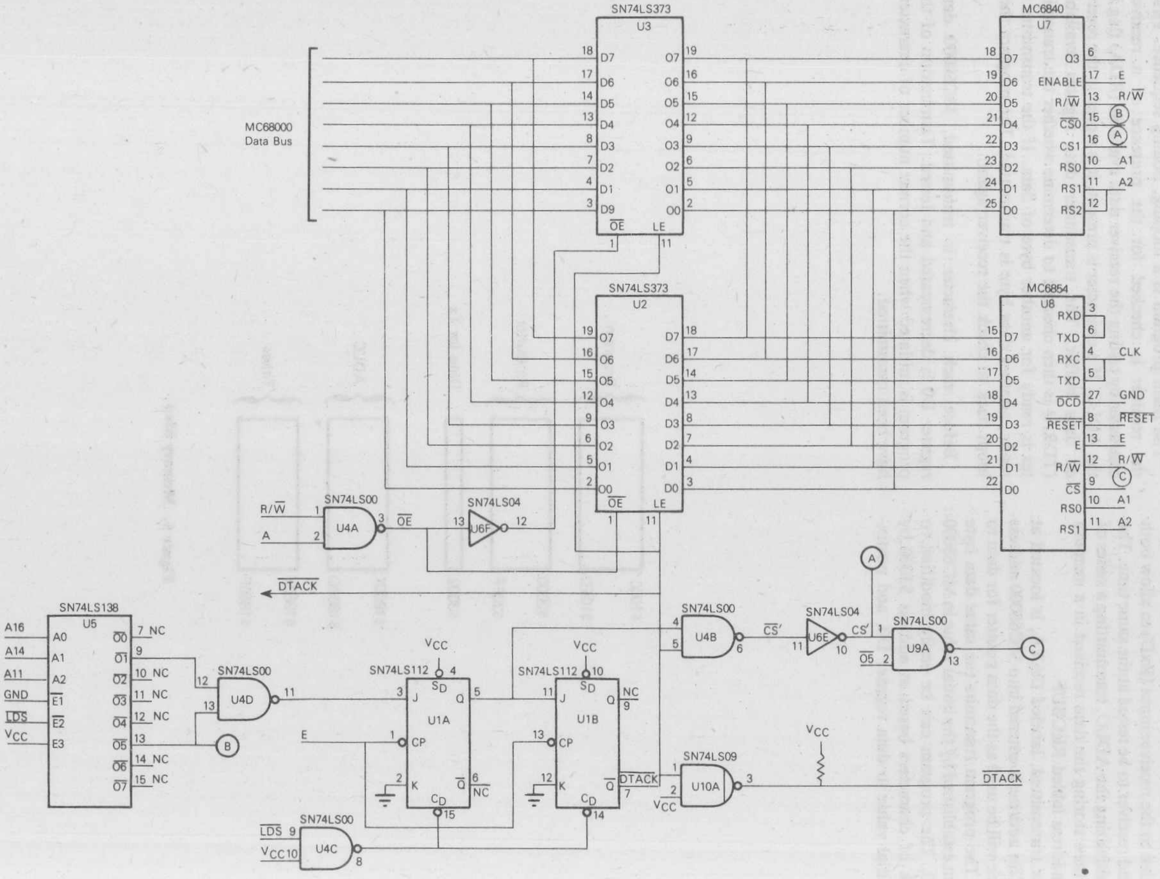


Figure 5. MC68000/MC6840/MC6854 Circuit Example

The ADLC transmitter and receiver clock inputs (TxC, RxC), are tied together and provided with a clock frequency determined by the desired data transfer rate. The transmitter output (TxD) is tied to the receiver input (RxD) to allow both the transmitter and receiver to be tested at the same time. The test consists of initializing the ADLC, transmitting a series of data bytes, and then storing the data received in a memory buffer based at address labled RECBUF.

The byte to be transmitted, labeled DATA, is located at address $3000. This address is entered into MC68000 address register A1, which will be used as the data pointer for data to be transmitted. The program transmits the same data byte 128 times, a count established by the initial value in MC68000 data register D0. The program can be easily modified to transmit a block of characters based at address $3000 by changing the initial value in data register D0, and postincrementing address register A1 after each character is transmitted (line 72).

The main program is a looping, polling sequence. First, the receiver is checked for the presence of a received character by testing the receiver data available (RDA) flag in the ADLC. If a character is present, it is stored in the received data buffer. The transmitter data register available (TDRA) is then checked to determine whether the transmitter is ready for another byte of data. If the transmitter is ready, another data byte is transmitted. The program then loops back to check the receiver again.

Before each character is transmitted, MC68000 data register D0 is decremented and tested. Termination of the program is initiated when the correct number of characters have been transmitted.

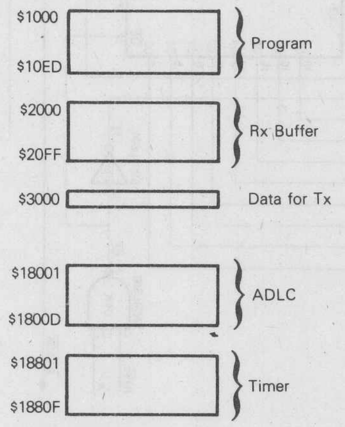


Figure 6. Memory Map

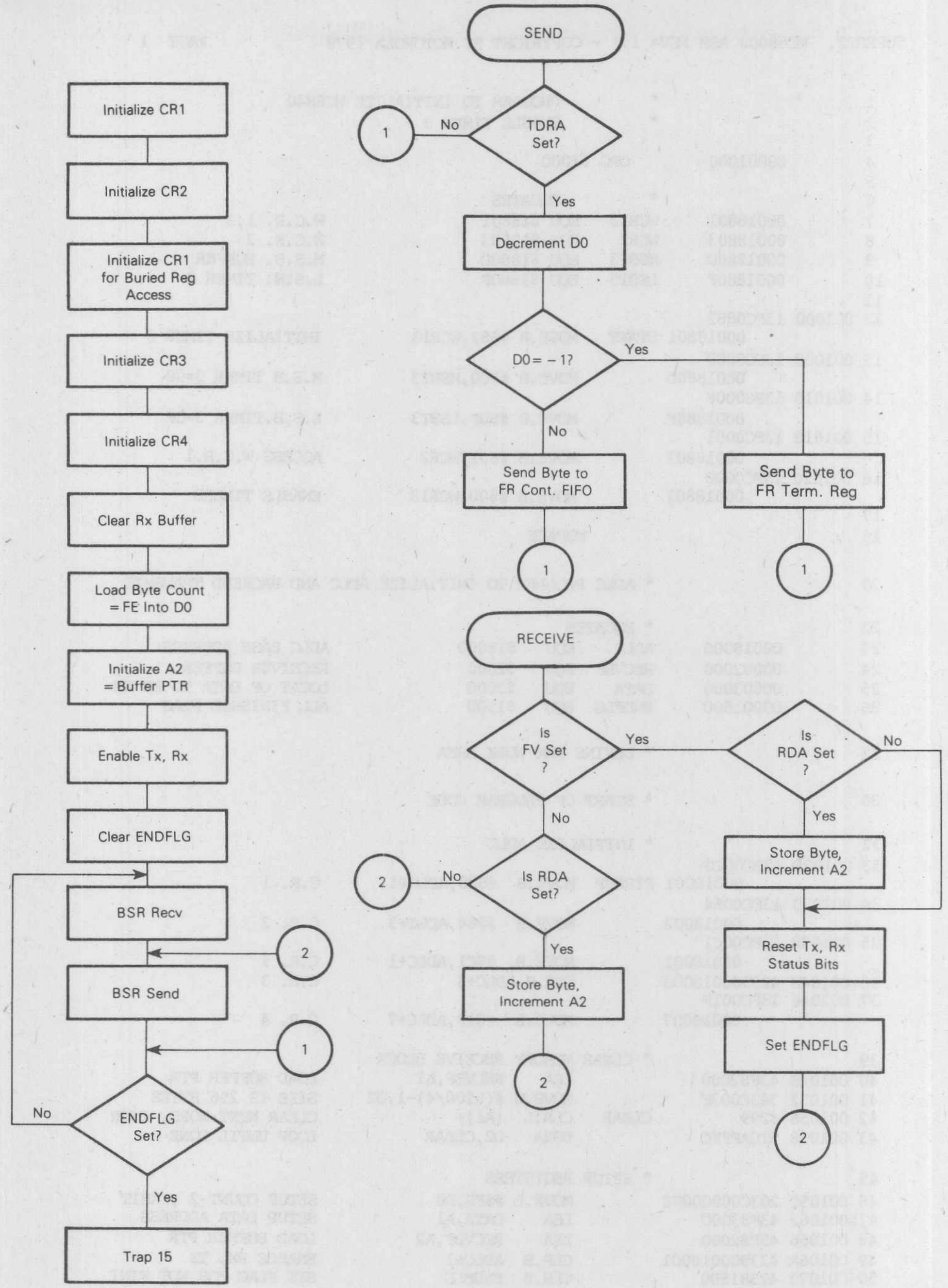


Figure 7. Program Flowchart

```
SUPERT2.  MC68000 ASM REV= 1.1 - COPYRIGHT BY MOTOROLA 1978            PAGE  1

 1                   *        PROGRAM TO INITIALIZE MC6840
 2                   *        ENABLE TIMER 3
 3
 4        00001000       ORG $1000
 5
 6                   *        EQUATES
 7        00018801     WCR13   EQU $18801                   W.C.R. 1,3
 8        00018803     WCR2    EQU $18803                   W.C.R. 2
 9        0001880D     MSBT3   EQU $1880D                   M.S.B. BUFFER
10        0001880F     LSBT3   EQU $1880F                   L.S.B. TIMER 3
11
12 001000 13FC0082
             00018801 START   MOVE.B #$82,WCR13           INITIALIZE TIMER 3
13 001008 13FC0000
             0001880D         MOVE.B #$00,MSBT3           M.S.B TIMER 3=00
14 001010 13FC000F
             0001880F         MOVE.B #$0F,LSBT3           L.S.B.TIMER 3=OF
15 001018 13FC0001
             00018803         MOVE.B #$01,WCR2            ACCESS W.C.R.1
16 001020 13FC0000
             00018801         MOVE.B #$00,WCR13           ENABLE TIMERS
17
18                           NOPAGE

20                   * ADLC PROGRAM TO INITIALIZE ADLC AND BACKEND TRANSMIT

22                   * EQUATES
23        00018000   ADLC    EQU    $18000              ADLC BASE ADDRESS
24        00002000   RECVBF  EQU    $2000               RECEIVER BUFFER
25        00003000   DATA    EQU    $3000               LOCAT OF DATA TO BE XM
26        00001500   ENDFLG  EQU    $1500               ALL FINISHED FLAG

28                   * DEFINE RAM WORK AREA

30                   * START OF PROGRAM CODE

32                   * INITIALIZE ADLC
33 001028 13FC00C0
             00018001 FIREUP  MOVE.B   #$C0,ADLC+1        C.R. 1
34 001030 13FC0064
             00018003         MOVE.B   #$64,ADLC+3        C.R. 2
35 001038 13FC00C1
             00018001         MOVE.B   #$C1,ADLC+1        C.R. 1
36 001040 423900018003        CLR.B ADLC+3                C.R. 3
37 001046 13FC001F
             00018007         MOVE.B   #$1F,ADLC+7        C.R. 4

39                   * CLEAR MEMORY RECEIVE BLOCK
40 00104E 43F82000            LEA      RECVBF,A1          LOAD BUFFER PTR
41 001052 343C003F            MOVE.W #($100/4)-1,D2       SIZE IS 256 BYTES
42 001056 4299       CLEAR    CLR.L    (A1)+              CLEAR NEXT WORD, INCR
43 001058 51CAFFFC            DBRA     D2,CLEAR           LOOP UNTIL DONE

45                   * SETUP REGISTERS
46 00105C 203C000000FE        MOVE.L #$FE,D0              SETUP COUNT-2 , XMIT
47 001062 43F83000            LEA      DATA,A1            SETUP DATA ADDRESS
48 001066 45F82000            LEA      RECVBF,A2          LOAD BUFFER PTR
49 00106A 423900018001        CLR.B    ADLC+1             ENABLE RX, TX
50 001070 42381500            CLR.B    ENDFLG             SET FLAG FOR NOT FINI
```

Figure 8. Program Listing

```
52                     * PROCESS TRANSMIT AND RECEIVE TASK TILL DONE
53 001074 61000038     PROCES  BSR    RECV                ATTEMPT RECEIVE
54 001078 6100000C             BSR    SEND                ATTEMPT TO SEND
55 00107C 4A381500             TST.B  ENDFLG              FINISHED?
56 001080 67F2                 BEQ    PROCES              LOOP IF NOT FINISHED
57 001082 4E4F                 TRAP   15                  BREAKPOINT WHEN FINIS
58 001084 0000                 DC.W   #0                  BREAKPOINT CODE


60                     * ATTEMPT TO TRANSMIT A CHARACTER
61 001086 08390006
              00018001 SEND    BTST.B #06,ADLC+1          TDRA SET? XMIT READY?
62 00108E 67000014             BEQ RETURN
63 001092 0C40FFFF             CMP  #-1,D0                ? LAST BYTE SENT?
64 001096 6700000C             BEQ    RETURN              YES IGNORE SENDING MO
65 00109A 51C8000A             DBRA   D0,MORE             COUNT DOWN, BRANCH NO
66                     * PROCESS LAST BYTE BY TERMINATING TRANSMISSION
67 00109E 13D100018007         MOVE.B (A1),ADLC+7         LAST BYTE INTO FRAME
68 0010A4 4E75         RETURN  RTS                        RETURN TO MAINLINE
69                     * PROCESS NEXT BYTE TO TRANSMIT (NOT THE LAST)
70 0010A6 13D100018005 MORE    MOVE.B (A1),ADLC+5         SEND NEXT BYTE TO FRA
71 0010AC 4E75                 RTS                        RETURN TO CALLER


73                     * ATTEMPT TO RECEIVE A CHARACTER
74 0010AE 08390001
              00018003 RECV    BTST.B  #1,ADLC+3          ? FRAME RECEIVED
75 0010B6 66000008             BNE    GOTFRM              BRANCH IF SO
76 0010BA 61000016             BSR    TRYINP              ATTEMPT INPUT OF NEXT
77 0010BE 4E75                 RTS                        RETURN TO MAINLINE
78                     * END OF FRAME PROCESSING
79 0010C0 61000010     GOTFRM  BSR    TRYINP              INSURE LAST BYTE PROC
80 0010C4 13FC0064
              00018003         MOVE.B #$64,ADLC+3         CLEAR TX,RX STATUS
81 0010CC 52381500             ADD.B  #1,ENDFLG           FLAG RECEIVER DONE
82 0010D0 4E75                 RTS                        RETURN TO MAINLINE


84                     * PROCESS INPUT BYTE IF ANY
85 0010D2 08390000
              00018001 TRYINP  BTST.B #0,ADLC+1           ? INPUT BYTE READY, T
86 0010DA 67C8                 BEQ    RETURN              RETURN IF NO BYTE
87 0010DC 14F900018005         MOVE.B ADLC+5,(A2)+        STORE BYTE
88 0010E2 4E75                 RTS                        RETURN


90                             END
91
92                     *©

****** TOTAL ERRORS  0--   0

SYMBOL TABLE

ADLC     018000 CLEAR    001056 DATA     003000 ENDFLG   001500
FIREUP   001028 GOTFRM   0010C0 LSBT3    01880F MORE     0010A6
MSBT3    01880D PROCES   001074 RECV     0010AE RECVBF   002000
RETURN   0010A4 SEND     001086 START    001000 TRYINP   0010D2
WCR13    018801 WCR2     018803
```

Figure 8. Program Listing (Concluded)

INTERFACING THE MC68000 TO THE MC6846 RIOT

Prepared by
David Ruhberg
Microprocessor Applications Engineer

The MC6846 ROM I/O Timer (RIOT) provides several versatile functions which the MC68000 may use with minimal effort. The RIOT features a 2K by 8 mask-programmable ROM, an 8-bit I/O port, and a 16-bit programmable timer/counter, in one forty-pin package. The MC68000 has the option of addressing the RIOT singly or in pairs, depending on the desired bus width. The 8-bit bus can be used if the upper and lower data strobes are used. Note that if a single RIOT is used, the MC68000 will not be able to obtain executable code from the ROM within the RIOT. This is due to the limitation introduced by the width of the data bus on the RIOT. Therefore, to effectively interface the ROM in the RIOT to the MC68000, a 16-bit data bus is used in this application. This configuration makes three 16-bit capabilities available to the MC68000. They are:

- 2K by 16 bits of mask-programmable ROM
- two parallel, 8-bit I/O ports, or one parallel, 16-bit I/O port
- two 16-bit timers that can be used together or independently

HARDWARE

The basic connections needed for the RIOT to function with the MC68000 are: the lower ten address lines (A1-A10), the sixteen data lines (D0-D15), and the R/$\overline{W}$, $\overline{RESET}$, E, and chip select signals. All of these may be obtained directly from the MC68000 with the exception of the chip select signals. As shown in Figure 1, the eight high-order data lines go to one RIOT and the eight low-order data lines go to the other RIOT. All other connections between the RIOTs are made in parallel. To obtain the chip select signals, some decoding circuitry must be provided. The RIOT may be run synchronously or asynchronously with the MC68000.

Synchronous Operation — To run the RIOT synchronously, some decoding circuitry must be used to provide a low input to the $\overline{VPA}$ pin of the MC68000 when the RIOT is selected. This synchronizes the MC68000 with E and generates the $\overline{VMA}$ signal.

To use the synchronous output of the MC68000, the decoding circuitry must also generate a $\overline{VPA}$ signal, in addition to the chip selects. This signal informs the MC68000 that it is addressing a M6800 peripheral and synchronizes the processor with the E clock. The $\overline{VPA}$ signal also causes $\overline{VMA}$ to be generated which can be used for other M6800 peripherals.

Asynchronous Operation — Operating the RIOTs asynchronously with the MC68000 allows the processor to begin executing the next instruction without waiting to synchronize with the E clock. To operate asynchronously, the decoding circuitry must generate a $\overline{DTACK}$ signal in addition to the chip selects.

SAMPLE CIRCUIT

The sample circuit interfaces parallel RIOTs through the MC68000 Design Module (MEX68KDM) data bus, as shown in Figure 2. Since only sixteen address lines (A1-A16) are brought out on this bus, addressing from the MC68000 is incomplete. Special attention should be given in addressing these devices from the MC68000 since the A0 address line on the RIOT corresponds to the A1 address line on the MC68000.

The RIOT used in this sample circuit (MC6846P3 TVBug) has the following characteristics. Having CS0 high and CS1 low selects the ROM, while having CS0 low and CS1 high selects the I/O Timer. Also, address lines A6 and A10 must be high and lines A3, A4, and A5 must be low for the I/O Timer to be selected. The three least-significant address bits are used to address the various I/O Timer control registers. A memory map is given in Figure 3. Besides power and ground, the E, CS0, CS1, R/$\overline{W}$, $\overline{RESET}$, and the ten address lines are connected in parallel to the RIOTs.

As mentioned earlier, the address lines are obtained from the limited address bus of the Design Module. Buffers are required to reduce noise on the bus lines. Bidirectional, three-stateable buffers are used on the data lines so that data may be transmitted to and received from the Module. The receive enable ($\overline{RE}$) signal will go low when R/$\overline{W}$ is low and the I/O

Timer chip select (CS1) is high. The driver enable (DE) signal will go high when R/$\overline{W}$ is high and either chip select is high.

The decoding circuitry shown in Figure 4 generates the chip select signals for synchronous operation. A high chip select signal (CS0) is generated for the ROM at addresses $10000 through $10FFF. The high chip select signal (CS1) is generated for the I/O Timer registers at addresses $11880 through $1188F. The two chip select signals are NORed together to generate the $\overline{VPA}$ signal. Valid peripheral address should be generated from an open collector gate or passed through a three-stateable buffer to permit a wire-ORed signal.

For asynchronous operation, the U10 NOR gate used to generate $\overline{VPA}$ is replaced with the TTL circuit shown in Figure 5.

The seven-segment displays are used to show the contents on the parallel I/O data registers in the RIOTs.

Address lines A4, A5, and A7 are decoded to allow software manipulation of the timer contained in each RIOT.

SOFTWARE

The software needed for the MC68000 to use the RIOT is straightforward. One point to keep in mind is that the MC68000 addresses every eight bits, even though it executes sixteen-bit instructions. This means that the least-significant byte of an instruction is always located at an odd address, and likewise the most-significant byte is always located at an even address. Since the hardware writes to all sixteen bits at once, the addresses for the control registers are located two addresses apart (i.e., PCR: 11882, DDR: 11884, etc.). In using the ROM, no preliminary software is necessary. However, to use the I/O lines, the peripheral control register must be initialized and the data direction register must be configured before data may be transmitted to or received from the peripheral data register.

The software for the sample circuit is given in Figure 6. The I/O lines are connected to four, seven-segment displays through MC14511 BCD-to-decimal decoders. The software initializes the I/O lines (as outputs) and then outputs the first ten bytes of each ROM. Since the decoders cannot decode hexadecimal numbers greater than nine, the software subtracts eight if the number is greater than nine. Then the code is output to the display for operator inspection. This software is included to give you an idea of the simplicity involved in interfacing to these M6800 peripherals.

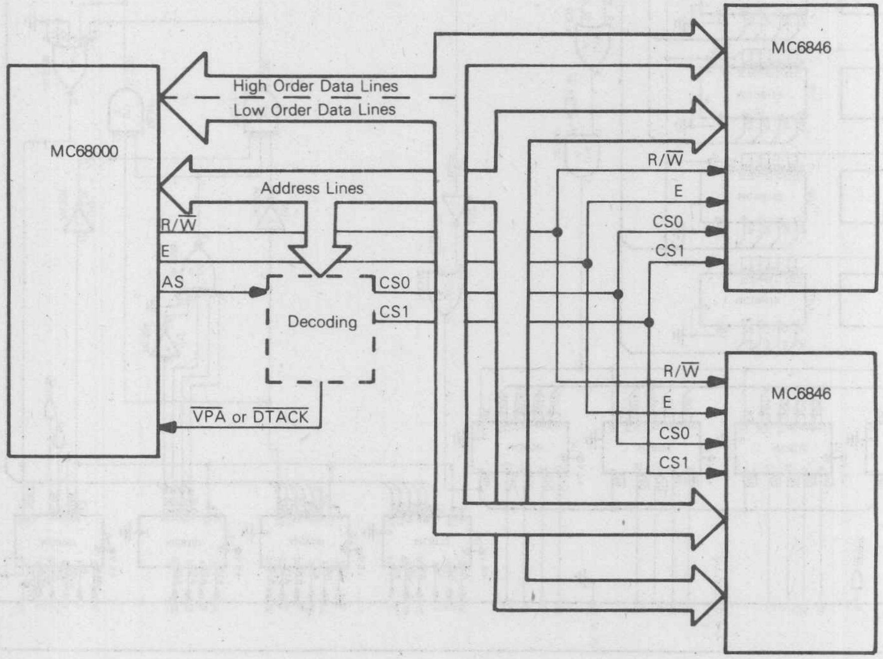


Figure 1. MC6846 to MC68000 Interface — Block Diagram

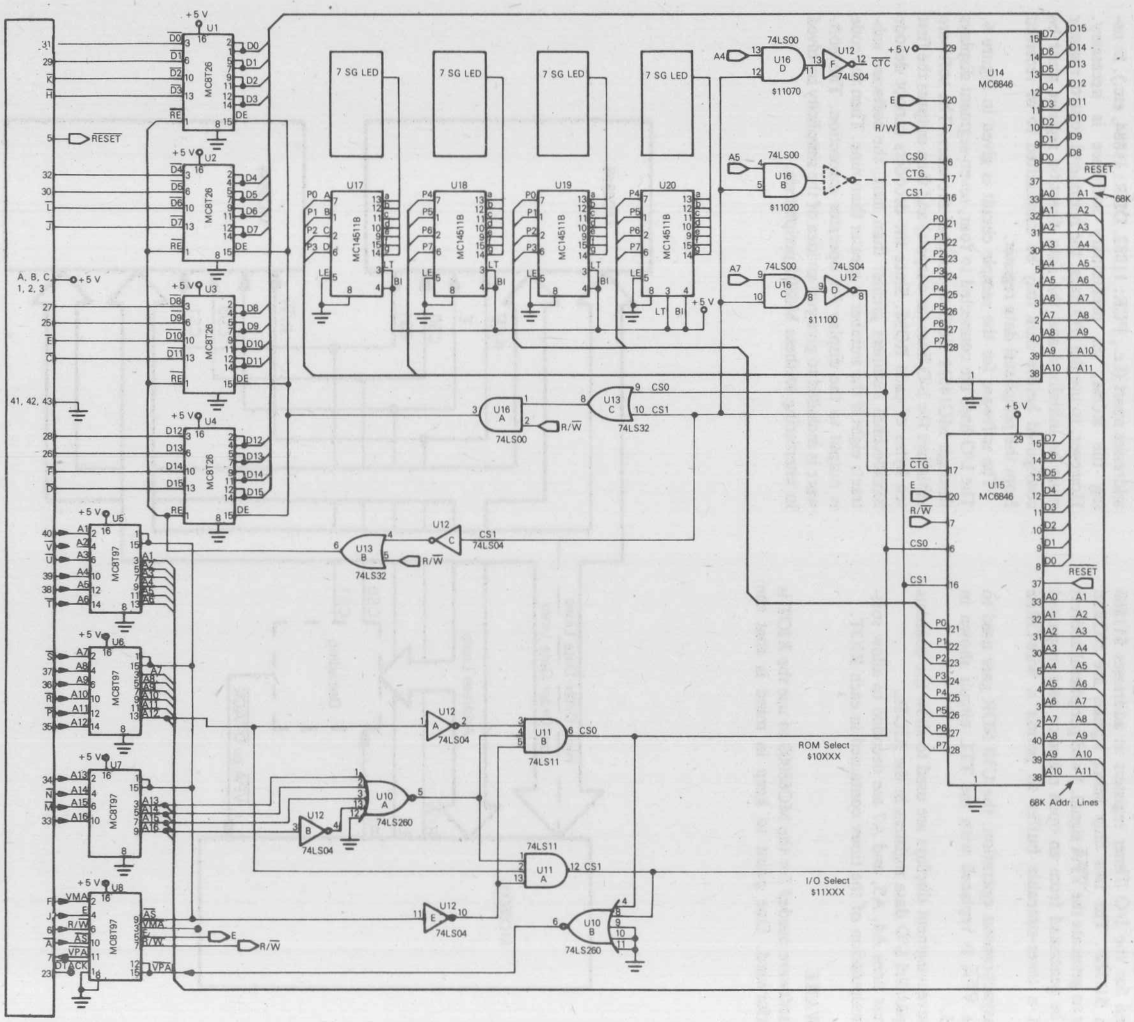


Figure 2. MC6846 to MC68000 Interface — Schematic

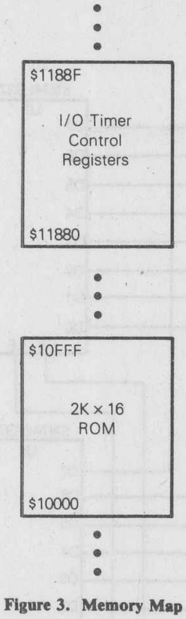


Figure 3. Memory Map

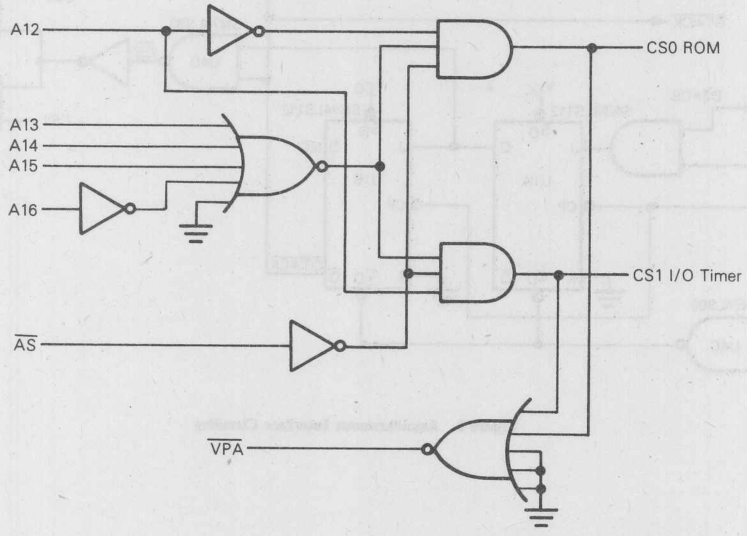


Figure 4. Decoding Circuitry (Synchronous Interface)

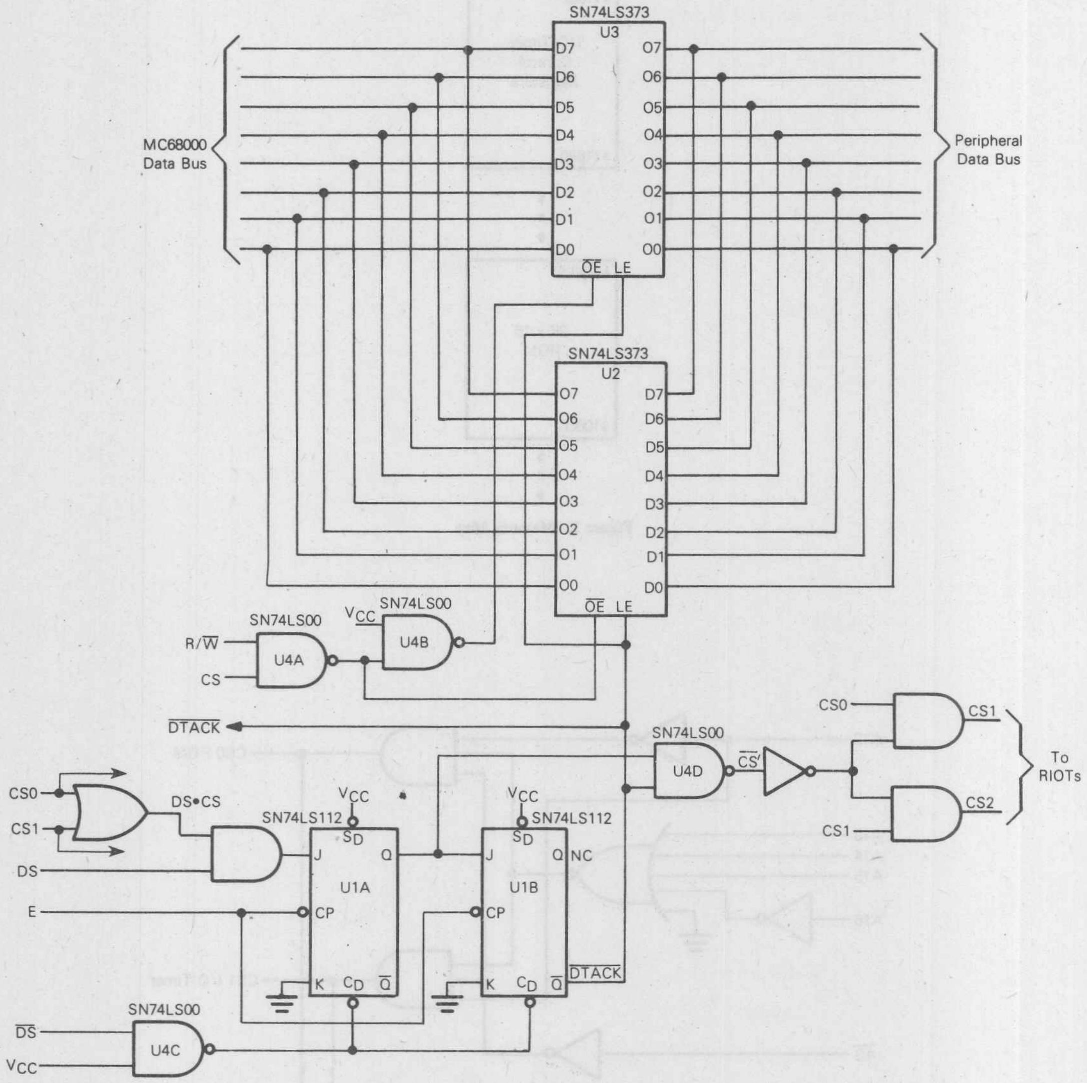
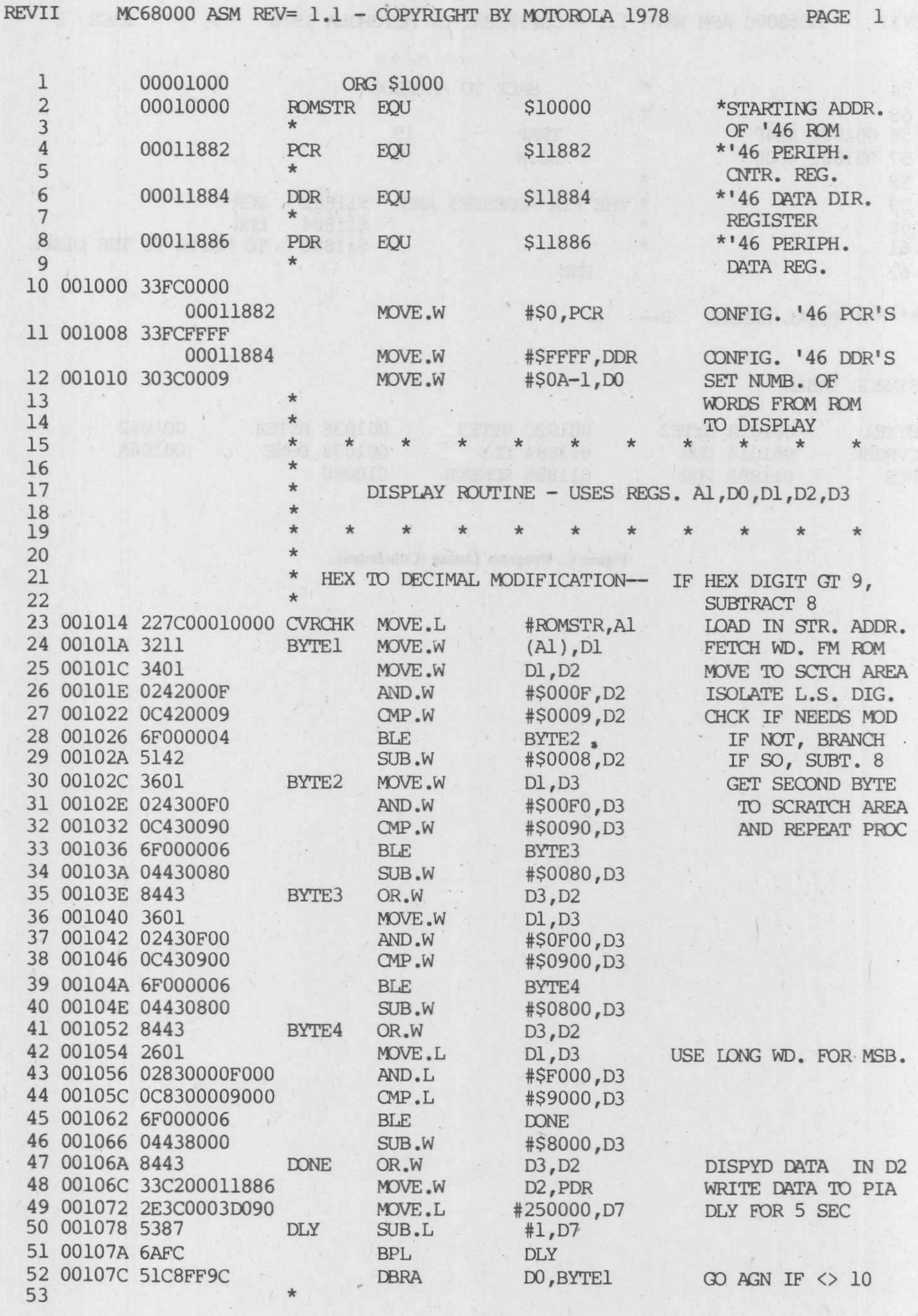


Figure 5. Asynchronous Interface Circuitry



```
REVII    MC68000 ASM REV= 1.1 - COPYRIGHT BY MOTOROLA 1978             PAGE  1

       00001000             ORG $1000
       00010000     ROMSTR  EQU          $10000          *STARTING ADDR.
                    *                                     OF '46 ROM
       00011882     PCR     EQU          $11882          *'46 PERIPH.
                    *                                     CNTR. REG.
       00011884     DDR     EQU          $11884          *'46 DATA DIR.
                    *                                     REGISTER
       00011886     PDR     EQU          $11886          *'46 PERIPH.
                    *                                     DATA REG.
001000 33FC0000
       00011882             MOVE.W       #$0,PCR         CONFIG. '46 PCR'S
001008 33FCFFFF
       00011884             MOVE.W       #$FFFF,DDR      CONFIG. '46 DDR'S
001010 303C0009             MOVE.W       #$0A-1,D0       SET NUMB. OF
                    *                                    WORDS FROM ROM
                    *                                    TO DISPLAY
                    *    *    *    *    *    *    *    *    *    *    *
                    *
                    *    DISPLAY ROUTINE - USES REGS. A1,D0,D1,D2,D3
                    *
                    *    *    *    *    *    *    *    *    *    *    *
                    *
                    *  HEX TO DECIMAL MODIFICATION--  IF HEX DIGIT GT 9,
                    *                                    SUBTRACT 8
001014 227C00010000 CVRCHK  MOVE.L       #ROMSTR,A1      LOAD IN STR. ADDR.
00101A 3211         BYTE1   MOVE.W       (A1),D1         FETCH WD. FM ROM
00101C 3401                 MOVE.W       D1,D2           MOVE TO SCTCH AREA
00101E 0242000F             AND.W        #$000F,D2       ISOLATE L.S. DIG.
001022 0C420009             CMP.W        #$0009,D2       CHCK IF NEEDS MOD
001026 6F000004             BLE          BYTE2             IF NOT, BRANCH
00102A 5142                 SUB.W        #$0008,D2         IF SO, SUBT. 8
00102C 3601         BYTE2   MOVE.W       D1,D3             GET SECOND BYTE
00102E 024300F0             AND.W        #$00F0,D3           TO SCRATCH AREA
001032 0C430090             CMP.W        #$0090,D3          AND REPEAT PROC
001036 6F000006             BLE          BYTE3
00103A 04430080             SUB.W        #$0080,D3
00103E 8443         BYTE3   OR.W         D3,D2
001040 3601                 MOVE.W       D1,D3
001042 02430F00             AND.W        #$0F00,D3
001046 0C430900             CMP.W        #$0900,D3
00104A 6F000006             BLE          BYTE4
00104E 04430800             SUB.W        #$0800,D3
001052 8443         BYTE4   OR.W         D3,D2
001054 2601                 MOVE.L       D1,D3        USE LONG WD. FOR MSB.
001056 02830000F000         AND.L        #$F000,D3
00105C 0C8300009000         CMP.L        #$9000,D3
001062 6F000006             BLE          DONE
001066 04438000             SUB.W        #$8000,D3
00106A 8443         DONE    OR.W         D3,D2           DISPYD DATA  IN D2
00106C 33C200011886         MOVE.W       D2,PDR          WRITE DATA TO PIA
001072 2E3C0003D090         MOVE.L       #250000,D7      DLY FOR 5 SEC
001078 5387         DLY     SUB.L        #1,D7
00107A 6AFC                 BPL          DLY
00107C 51C8FF9C             DBRA         D0,BYTE1        GO AGN IF <> 10
                    *
```

Figure 6. Program Listing

```
REVII    MC68000 ASM REV= 1.1 - COPYRIGHT BY MOTOROLA 1978          PAGE  2

 54                    *         BACK TO MACSBUG
 55                    *
 56 001080  4E4F               TRAP         15
 57 001082  0000               DC.W          0
 58                    *
 59                    * THE PIA ADRESSES ARE   $11882   PCR
 60                    *                        $11884   DDR
 61                    *                        $11886   TO WRITE TO THE DISPL
 62                            END

****** TOTAL ERRORS   0--   0


SYMBOL TABLE

BYTE1     00101A BYTE2     00102C BYTE3     00103E BYTE4     001052
CVRCHK    001014 DDR       011884 DLY       001078 DONE      00106A
PCR       011882 PDR       011886 ROMSTR    010000
```

Figure 6. Program Listing (Concluded)

AN-810

DUAL 16-BIT PORTS FOR THE MC68000 USING TWO MC6821S

Prepared by:
James McKenzie
Microprocessor Applications Engineering

The MC6821 Peripheral Interface Adapter (PIA) is a 40-pin device having two 8-bit ports. Each port has its own control register and may be configured as input or output on a bit-by-bit basis.

Two PIAs may be configured on the MC68000 microprocessor bus to give two 16-bit ports. The ability of the MC68000 to simultaneously access 16 bits of data at an effective rate of up to two megahertz makes it ideal for processing applications using state-of-the-art A/D or D/A converters. The MC68000 is also suited for applications involving advanced peripherals with parallel inputs or outputs and a high data throughput rate.

ASYNCHRONOUS OPERATION

The schematic for the MC68000/MC6821 asynchronous interface appears in Figure 1. Typical timing diagrams appear in Figure 2. Edge connector designations for MC68000 signals correspond to the MEX68KDM Design Module bus pin allocations (EXORciser bus). The asynchronous interface is responsible for three major tasks:

- Detecting when the PIAs are being addressed
- Synchronizing the MC68000 bus cycle to the local E clock
- Controlling data flow to and from the PIAs

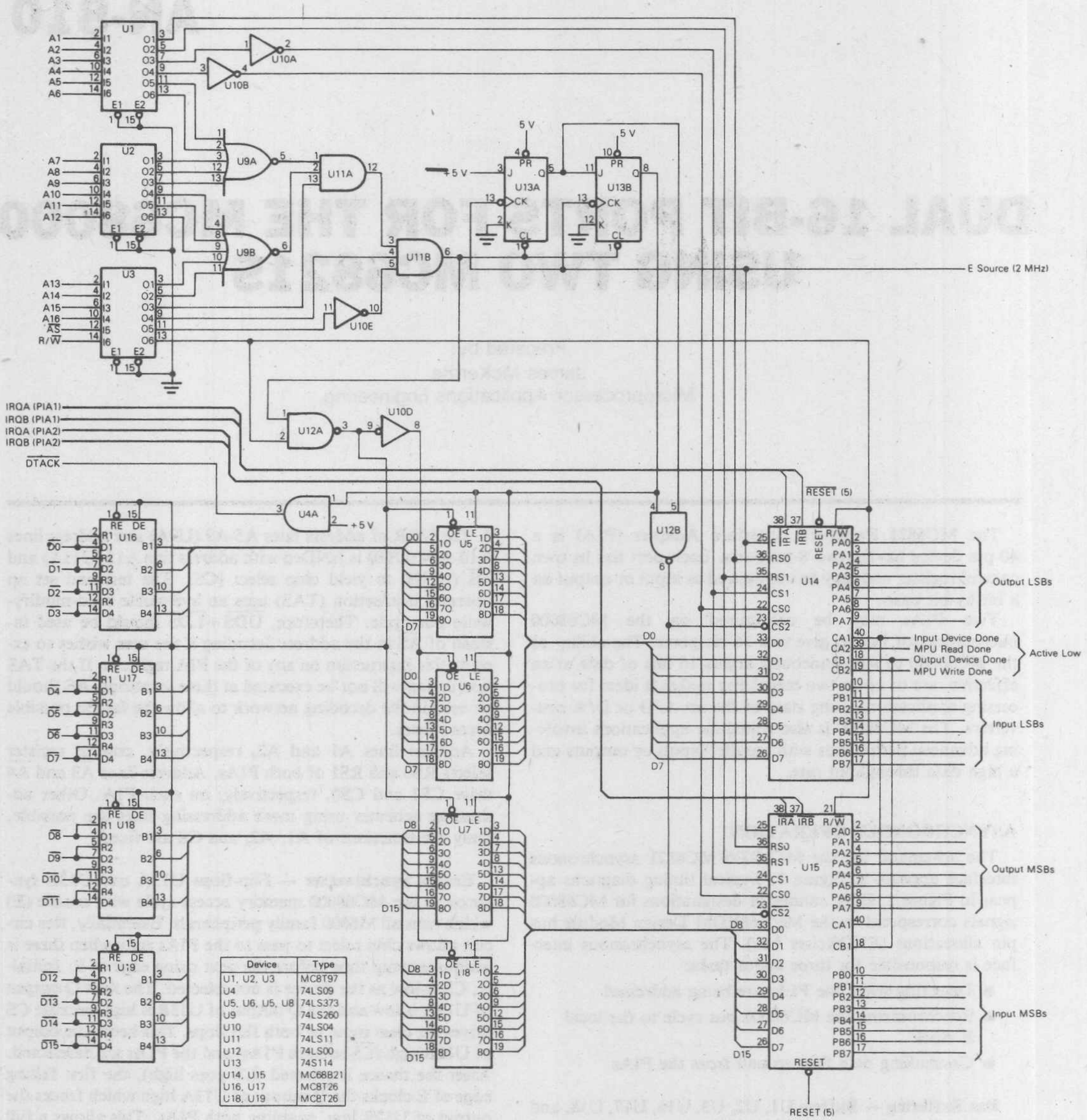
Bus Buffering — Buffers U1, U2, U3, U16, U17, U18, and U19 are all microprocessor bus buffers/drivers which make this design compatible with the MEX68KDM Design Module. Other buffering schemes may be more appropriate in other applications. In particular, buffers U5, U6, U7 and U8 will provide sufficient data bus buffering in a system where the data bus is not inverted.

Address Decoding — The two PIAs are located at \$18000-\$18007, as shown in the memory map given in Figure 3. The NOR of address lines A5-A9 (U9A) and address lines A10-A14 (U9B) is ANDed with address line A15 (U11A) and $\overline{\text{AS}}$ (U11B) to yield chip select (CS). The test and set an operand instruction (TAS) uses an indivisible read-modify-write bus cycle. Therefore, $\overline{\text{UDS}}+\overline{\text{LDS}}$ should be used instead of $\overline{\text{AS}}$ in the address decoding if the user wishes to execute this instruction on any of the PIA registers. If the TAS instruction will not be executed at these locations, AS should be used in the decoding network to allow the fastest possible access times.

Address lines A1 and A2, respectively, control register selects RS0 and RS1 of both PIAs. Address lines $\overline{\text{A3}}$ and $\overline{\text{A4}}$ drive CS1 and CS0, respectively, on each PIA. Other addressing schemes using more addressing lines are possible. Only the functions of A1, A2, and CS are fixed.

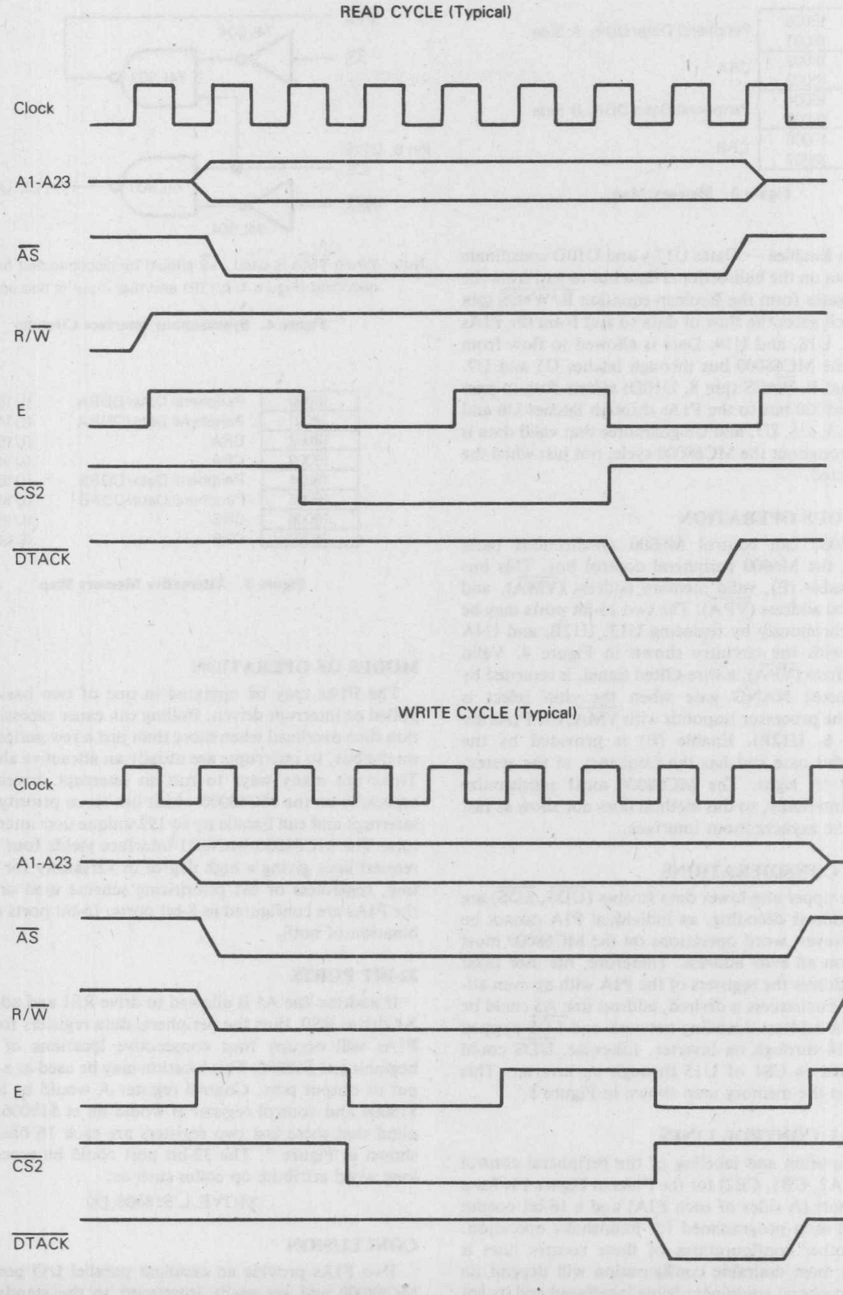
Enable Synchronizer — Flip-flops U13A and U13B synchronize the MC68000 memory access cycle with enable (E) which runs all M6800 family peripherals. Essentially, this circuit allows chip select to pass to the PIAs only when there is adequate setup time before the next rising edge of E. Initially, CS is low as the device is not selected. The next Q output of U13A is low and the $\overline{\text{Q}}$ output of U13B is high because CS drives the clear input of both flip flops. This keeps the output of U12B high ($\overline{\text{CS}}$ to both PIAs) and the PIAs are deselected. After the device is selected (CS goes high), the first falling edge of E clocks the $\overline{\text{Q}}$ output of U13A high which forces the output of U12B low, enabling both PIAs. This allows a full half cycle of E for setup before the rising edge of E. At the next falling edge of E, the Q output of U13A is clocked low. This removes $\overline{\text{CS2}}$ from both PIAs (via U12B), latches the data present on the bus (U5, U6, U7, U8) and returns $\overline{\text{DTACK}}$ to the MC68000 through an open-collector buffer U4A). This terminates the MC68000 memory access cycle.

If a fifty percent duty cycle, two megahertz E signal is used a best case access cycle time of 875 nanoseconds and a worst case access cycle time of 1375 nanoseconds will be obtained.



Device	Type
U1, U2, U3	MC8T97
U4	74LS09
U5, U6, U5, U8	74LS373
U9	74LS260
U10	74LS04
U11	74LS11
U12	74LS00
U13	74S114
U14, U15	MC68B21
U16, U17	MC8T26
U18, U19	MC8T26

Figure 1. Asynchronous MC6821 Interface — Schematic Diagram


READ CYCLE (Typical)

WRITE CYCLE (Typical)

Figure 2. Asynchronous Interface Timing Diagrams

18000 18001	Peripheral Data/DDR, A Side
18002 18003	CRA
18004 18005	Peripheral Data/DDR, B Side
18006 18007	CRB

Figure 3. Memory Map

Bus Buffers Enables — Gates U12A and U10D coordinate the flow of data on the bidirectional data bus to and from the PIAs. These gates form the Boolean equation $\overline{R/W \bullet CS}$ (pin 3, U12A) which gates the flow of data to and from the PIAs on U16, U17, U18, and U19. Data is allowed to flow from the PIAs to the MC68000 bus through latches U5 and U7. Also, the signal R/$\overline{W}$•CS (pin 8, U10D) allows data to pass from the MC68000 bus to the PIAs through latches U6 and U8. Latches U5, U6, U7, and U8 guarantee that valid data is on the bus throughout the MC68000 cycle, not just when the PIAs are selected.

SYNCHRONOUS OPERATION

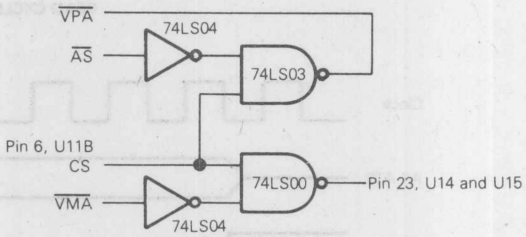
The MC68000 can control M6800 synchronous parts directly using the M6800 peripheral control bus. This bus consists of enable (E), valid memory address ($\overline{VMA}$), and valid peripheral address ($\overline{VPA}$). The two 16-bit ports may be operated synchronously by replacing U13, U12B, and U4A of Figure 1 with the circuitry shown in Figure 4. Valid peripheral address ($\overline{VPA}$), a wire-ORed signal, is returned by an open-collector NAND gate when the chip select is detected. As the processor responds with $\overline{VMA}$, the PIAs are selected (pin 6, U12B). Enable (E) is provided by the MC68000 in this case and has the frequency of the system clock divided by eight. The MC68000 must synchronize $\overline{VMA}$ with E internally, so this method does not allow as fast an access as the asynchronous interface.

SOFTWARE CONSIDERATIONS

Because the upper and lower data strobes ($\overline{UDS}$, $\overline{LDS}$) are not used in address decoding, an individual PIA cannot be accessed. However, word operations on the MC68000 must begin access on an even address. Therefore, the user must take care to address the registers of the PIA with an even address. If individual access is desired, address line A3 could be shifted into the address decoding network and $\overline{LDS}$ applied to CS1 of U14 through an inverter. Likewise, $\overline{UDS}$ could then be applied to CS1 of U15 through an inverter. This would result in the memory map shown in Figure 5.

PERIPHERAL CONTROL LINES

The configuration and labeling of the peripheral control lines (CA1, CA2, CB1, CB2) for the PIAs in Figure 1 is for a 16-bit input port (A sides of each PIA) and a 16-bit output port (B sides) each programmed for handshake operation. Any of the other configurations of these control lines is possible. The most desirable configuration will depend on the type of peripheral equipment being interfaced and its application. A typical initialization routine for the configuration shown in Figure 1 is given in Figure 6.



Note: When $\overline{VMA}$ is used, $\overline{AS}$ should be disconnected from the $\overline{CS}$ decoding (Figure 1, U11B) and that input is tied active.

Figure 4. Synchronous Interface Circuitry

18000	Peripheral Data/DDRA	(U15)
18001	Peripheral Data/DDRA	(U14)
18002	CRA	(U15)
18003	CRA	(U14)
18004	Peripheral Data/DDRB	(U15)
18005	Peripheral Data/DDRB	(U14)
18006	CRB	(U15)
18007	CRB	(U14)

Figure 5. Alternative Memory Map

MODES OF OPERATION

The PIAs may be operated in one of two basic modes, polled or interrupt driven. Polling can cause excessive execution time overhead when more than just a few peripherals are on the bus, so interrupts are usually an attractive alternative. There are many ways to run an interrupt driven system, especially on the MC68000 which has seven priority levels of interrupt and can handle up to 192 unique user interrupt vectors. The MC68000/MC6821 interface yields four interrupt request lines giving a high degree of versatility for interrupting, regardless of the prioritizing scheme used or whether the PIAs are configured as 8-bit ports, 16-bit ports or a combination of both.

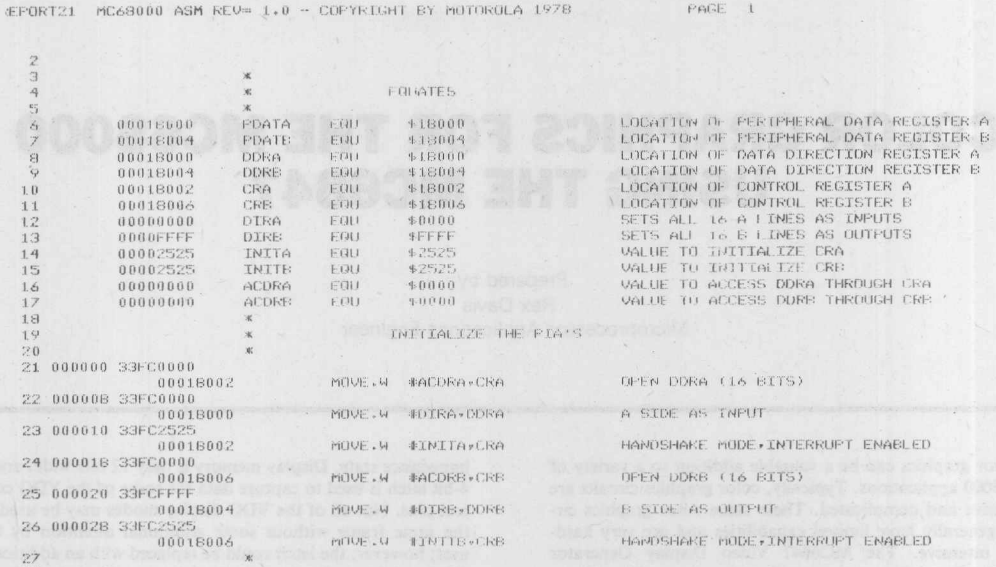
32-BIT PORTS

If address line A1 is allowed to drive RS1 and address line A2 drives RS0, then the peripheral data registers for the two PIAs will occupy four consecutive locations of memory beginning at $18000. This location may be used as a 32-bit input or output port. Control register A would be located at $18004 and control register B would be at $18006. Keep in mind that these last two registers are each 16 bits wide, as shown in Figure 7. The 32-bit port could be accessed with long word attribute op codes such as:

```
MOVE.L $18000,D0
```

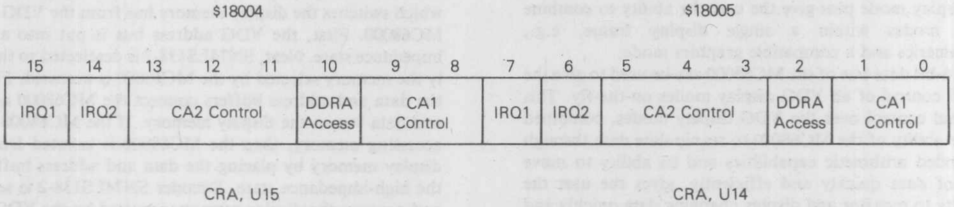
CONCLUSION

Two PIAs provide an excellent parallel I/O port for the MC68000 and are easily interfaced to the standard asynchronous bus. If B series parts are used, the PIAs may be accessed at effective rates of greater than 1.0 megahertz.



```
EPORT21  MC68000 ASM REV= 1.0 - COPYRIGHT BY MOTOROLA 1978          PAGE  1

                                 *
                                 *               EQUATES
                                 *
          00018000     PDATA     EQU      $18000           LOCATION OF PERIPHERAL DATA REGISTER A
          00018004     PDATB     EQU      $18004           LOCATION OF PERIPHERAL DATA REGISTER B
          00018000     DDRA      EQU      $18000           LOCATION OF DATA DIRECTION REGISTER A
          00018004     DDRB      EQU      $18004           LOCATION OF DATA DIRECTION REGISTER B
          00018002     CRA       EQU      $18002           LOCATION OF CONTROL REGISTER A
          00018006     CRB       EQU      $18006           LOCATION OF CONTROL REGISTER B
          00000000     DIRA      EQU      $0000            SETS ALL 16 A LINES AS INPUTS
          0000FFFF     DIRB      EQU      $FFFF            SETS ALL 16 B LINES AS OUTPUTS
          00002525     INITA     EQU      $2525            VALUE TO INITIALIZE CRA
          00002525     INITB     EQU      $2525            VALUE TO INITIALIZE CRB
          00000000     ACDRA     EQU      $0000            VALUE TO ACCESS DDRA THROUGH CRA
          00000000     ACDRB     EQU      $0000            VALUE TO ACCESS DDRB THROUGH CRB
                                 *
                                 *        INITIALIZE THE PIA'S
                                 *
   000000 33FC0000
              00018002                 MOVE.W  #ACDRA,CRA           OPEN DDRA (16 BITS)
   000008 33FC0000
              00018000                 MOVE.W  #DIRA,DDRA           A SIDE AS INPUT
   000010 33FC2525
              00018002                 MOVE.W  #INITA,CRA           HANDSHAKE MODE,INTERRUPT ENABLED
   000018 33FC0000
              00018006                 MOVE.W  #ACDRB,CRB           OPEN DDRB (16 BITS)
   000020 33FCFFFF
              00018004                 MOVE.W  #DIRB,DDRB           B SIDE AS OUTPUT
   000028 33FC2525
              00018006                 MOVE.W  #INITB,CRB           HANDSHAKE MODE,INTERRUPT ENABLED
                                 *
                                 *
```

Figure 6. Initialization Routine



$18004								$18005							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ1	IRQ2	CA2 Control			DDRA Access	CA1 Control		IRQ1	IRQ2	CA2 Control			DDRA Access	CA1 Control	
CRA, U15								CRA, U14							

Figure 7. 16-Bit Control Register

AN-815

COLOR GRAPHICS FOR THE MC68000 USING THE MC6847

Prepared by:
Rex Davis
Microprocessor Applications Engineer

Color graphics can be a valuable addition to a variety of MC68000 applications. Typically, color graphics circuits are expensive and complicated. These same color graphics circuits generally have limited capabilities and are very hardware intensive. The MC6847 Video Display Generator (VDG) offers a low cost, versatile, easy to use alternative.

The VDG creates a composite video signal according to information read from the display memory in the mode defined by the VDG control pins. The composite video signal can be used to modulate the input of any commercially available color or black and white television receiver. The VDG has 12 distinct display modes available and allows certain variations within certain modes. All display modes and their variations are controlled by the state of eight different VDG pins. The eight display mode pins give the user the ability to combine display modes within a single display frame, e.g., alphanumerics and a compatible graphics mode.

The 16-bit data bus of the MC68000 can be used to give the user full control of all VDG display modes on-the-fly. This additional control over the VDG display modes, combined with the ability of the MC68000 to manipulate data through its extended arithmetic capabilities and its ability to move blocks of data quickly and efficiently, gives the user the capability to monitor and display changing data quickly and efficiently as might be necessary in commercial, industrial, or scientific applications. The MC68000/MC6847 interface described in this application note, or a variation of it, could be used in a typical application.

BASIC IMPLEMENTATION

Since the display information for the VDG is stored in memory, the MC68000 can alter the display by changing the contents of that memory. The MC68000/MC6847 interface then becomes a shared-memory interface. The primary problem then becomes a matter of guaranteeing that memory is accessed by only one device at a time. Figure 1 is a block diagram of the MC68000/MC6847 interface.

The MC68000 is buffered from the display memory by three-state buffers and transceivers which are active only during a processor access of display memory. During a processor access, the VDG address bus is forced to a high-impedance state. Display memory is only 12 bits wide, and a 4-bit latch is used to capture data for some of the VDG control pins. Not all of the VDG display modes may be used in the same frame without some additional attention by the user; however, the latch could be replaced with an additional 6K by 4 bits of memory. A data transfer acknowledge ($\overline{\text{DTACK}}$) signal is generated so that either fast or slow memory can be efficiently used.

CIRCUIT DESCRIPTION

A schematic diagram of the MC68000/MC6847 interface is given in Figure 2. The select circuitry is formed around two SN74LS138 1-of-8 decoders. If the MC68000 accesses any of the 6K of memory, pin 8 of the SN74LS21 outputs a signal which switches the display memory bus from the VDG to the MC68000. First, the VDG address bus is put into a high-impedance state. Next, SN74LS138-2 is deselected so that only the memory selected by the MC68000 is accessed. Finally, the data and address buffers connect the MC68000 address and data bus to the display memory. If the MC68000 is not accessing memory, then the MC68000 is isolated from the display memory by placing the data and address buffers in the high-impedance state. Decoder SN74LS138-2 is selected and accesses the display memory required by the VDG. The VDG is then released to scan the memories (MCM2114) for display information.

In order to guarantee that no incompatible display modes are used within the same scan frame, only the last data write to any even memory location can alter the latch; therefore, the VDG will always read the same data from the SN74LS75 latch between MC68000 writes to the display memory. However, the user could still adversely affect the display if the MC68000 writes occurred within the VDG display scan. The frame sync pin of the VDG is low when the VDG is not in the active display window and could be used as an interrupt or polled to determine when the MC68000 can safely write to the display memory.

The $\overline{\text{DTACK}}$ signal is generated by the SN74LS95 shift register and is necessary for the completion of any MC68000 display memory access. The shift register will add two additional wait states for each successive output; i.e., Q1 gives two wait states, Q2 gives four wait states. Output Q0 will give no wait states. Table 1 lists the two critical memory specifications: data setup time and access time and the shift register output necessary to guarantee operation.

Typically, access time is the more critical specification. Output Q1 was chosen to be used with the MCM2114-30 (300 ns access time) memories used in this application although faster or slower memories could have been used. The speed of the display memory determines the rate at which data can be moved into the display memory. Other factors determining the speed of the data transfer are: the method used to determine when the VDG is out of the active display area; and the software routine used to move the data.

CONCLUSION

The MC68000/MC6847 interface described in this application can be expanded or limited, according to the user's application, with relative hardware ease. Any approach of shared memory could be used as long as no bus conflicts result. Even the simple approach taken in this application results in a powerful, low cost color graphics system for the MC68000.

Table 1. Required Shift Register Output

	Q0	Q1	Q2	Q3
Data Setup Time (ns)	230	355	480	605
Access Time (ns)	250	375	500	625

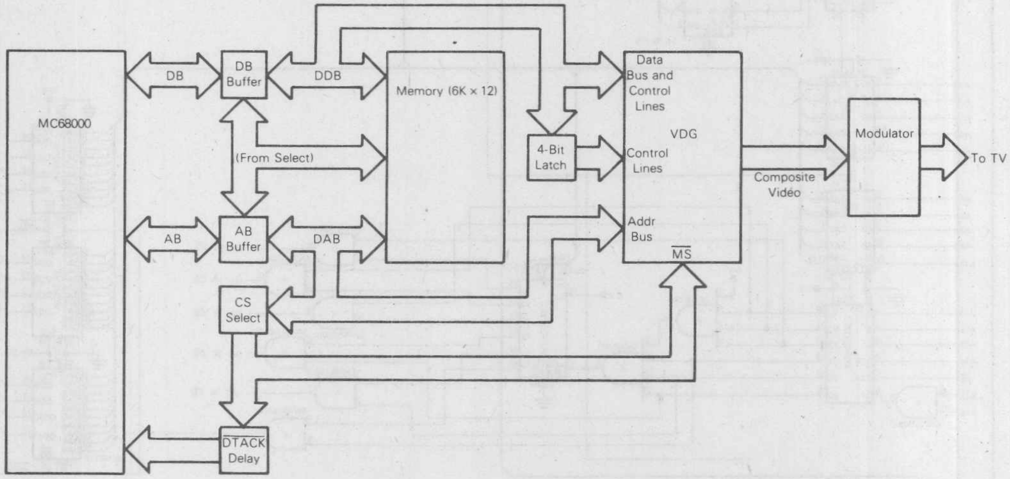


Figure 1. MC68000/MC6847 Interface — Block Diagram

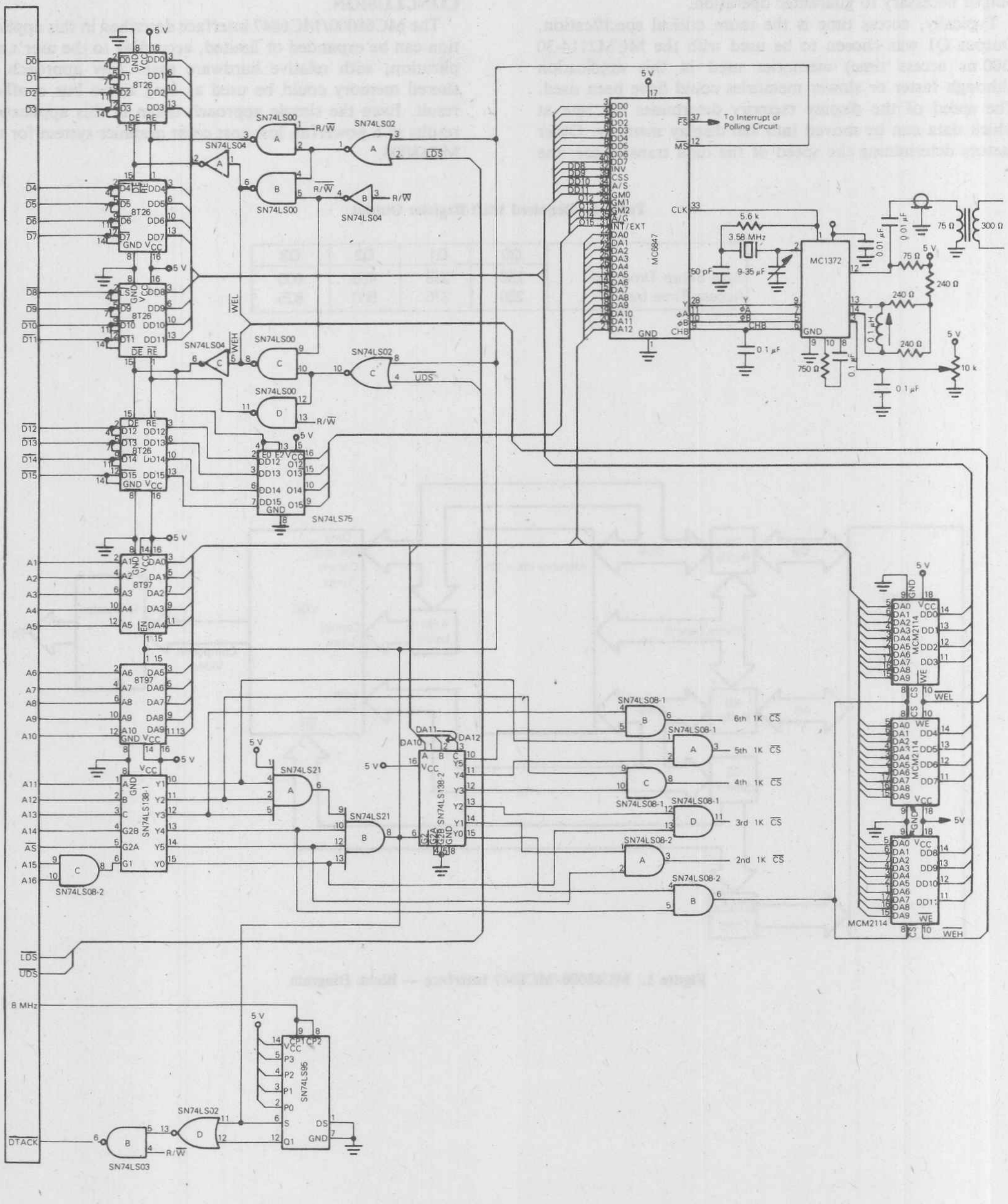


Figure 2. MC68000/MC6847 — Schematic Diagram

SOFTWARE REFRESHED MEMORY CARD FOR THE MC68000

Prepared by:
Duane Graden
Microprocessor Applications Engineer

This application describes the hardware and software to implement a software-refreshed, dynamic memory card for use in an eight megahertz MC68000 system. This refresh approach consumes less than five percent of processor time. The MCM4116 16K RAM was chosen for this design, but the techniques discussed are applicable to the MCM6664 64K RAM as well.

Refresh techniques fall into two categories, hardware and software. Hardware refresh is more component intensive with little or no overhead in program time, while software refresh has less hardware and more program overhead.

Hardware refresh means that the required circuitry must refresh the dynamic RAM cell with little or no impact on execution of instructions by the processor. Normally, this means accessing the address bus during a dead part of the cycle. Another drawback is the complex circuitry, usually requiring the use of expensive delay lines.

Software refresh means that the processor must execute a software routine to refresh dynamic memory. To accomplish this, an interrupt service routine, such as the level seven interrupt service routine on the MC68000, can be dedicated to refresh the memory. Every time the interrupt is recognized, a hardware enable allows the refresh routine to refresh the dynamic RAM.

TIMING SIGNALS

Timing requirements of MCM4116 RAMs and the MC68000 are easy to match because of the asynchronous nature of the MC68000 bus structure. The MC68000 can wait for the slowest RAM through the use of the data transfer acknowledge ($\overline{\text{DTACK}}$) signal. As long as $\overline{\text{DTACK}}$ is asserted a setup time before the falling edge of any clock state (S4 or later), it will be recognized during that state. Termination of the access is 1½ clock periods later. Figure 1 is a timing diagram for a read, write, and refresh operation.

The $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ signals are the row address and column address multiplex control inputs, respectively, for the seven memory address lines A1 through A7. Since no chip select inputs are present with this dynamic memory, $\overline{\text{RAS}}$ is the active low signal that starts a memory access cycle. When $\overline{\text{RAS}}$ falls, the row address of the location to be accessed is latched into memory. Similarly, the falling edge of $\overline{\text{CAS}}$ latches the column address into memory.

The refresh cycle shown in Figure 1, is known as $\overline{\text{RAS}}$-only refresh. Row address select is low, $\overline{\text{CAS}}$ is high, R/$\overline{\text{W}}$ does not matter, and the row address of the row to be refreshed is present on the seven address lines. Each row of memory requires a refresh cycle to be performed every two milliseconds for data to be retained. For the MCM4116 memory, there are 128 rows and, therefore 128 refresh cycles required every two milliseconds.

HARDWARE DESCRIPTION

Figure 2 is the schematic diagram for a dynamic memory card using MCM4116 memories. This card, when used with a MC68000 system, provides 64K bytes of memory from 32K to 96K of the physical address map.

Memory decoding is done with the upper and lower data strobes and address lines A15 and A16. The data strobes divide the memory into even and odd blocks, respectively. The upper data strobe chip selects even bytes from 32K to 96K by activating a row address select upper (RASU) signal. Address lines A15 and A16, through decoder U2 and gate U4, decode whichever of the two banks of even memory (RAS1U or RAS2U) is selected. Similarly, the lower data strobe activates a row address select lower (RASL) signal.

Column address select ($\overline{\text{CAS}}$) is activated on the second falling edge of the eight megahertz clock after $\overline{\text{RAS}}$ is asserted by flip flop U9. Both $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ are turned off when the data strobes are inactive.

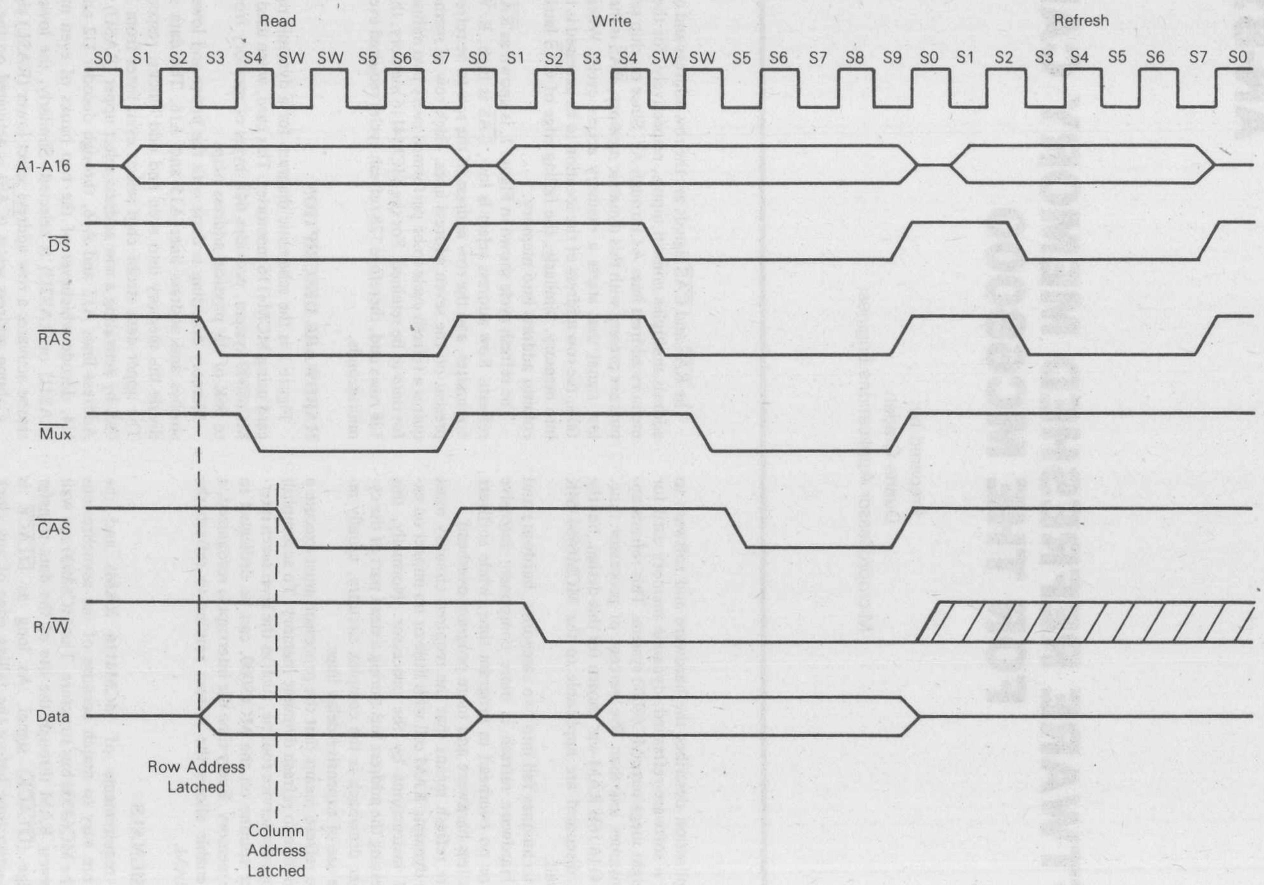

Read
Write
Refresh
S0 S1 S2 S3 S4 SW SW S5 S6 S7 S0 S1 S2 S3 S4 SW SW S5 S6 S7 S8 S9 S0 S1 S2 S3 S4 S5 S6 S7 S0
A1-A16
DS
RAS
Mux
CAS
R/W
Data
Row Address Latched
Column Address Latched

Figure 1. Read, Write, and Refresh Timing Diagrams

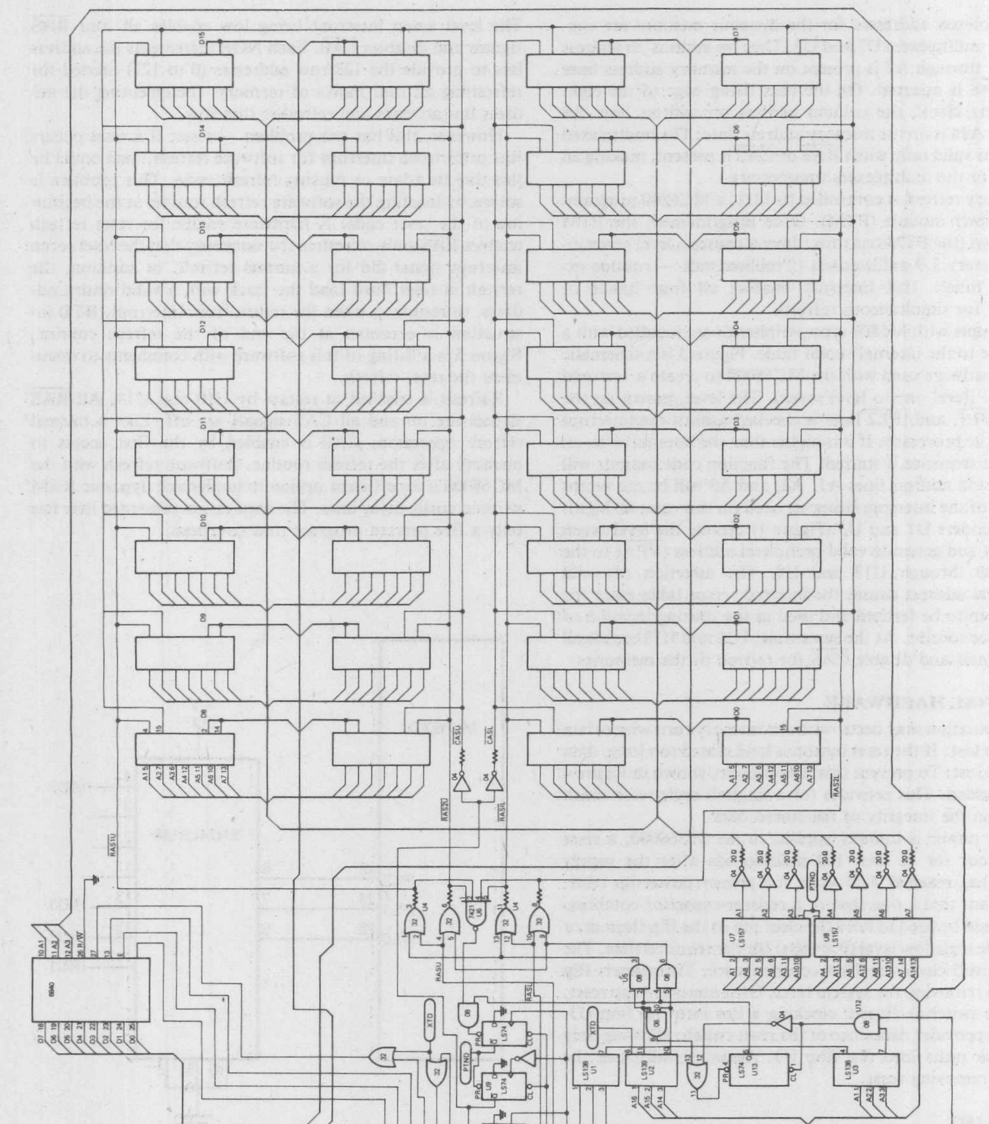


Figure 2. Dynamic Memory Card — Schematic Diagram

Multiplexed addresses for the dynamic memory are supplied by multiplexers U7 and U8. The row address on address lines A1 through A7 is present on the memory address lines until RAS is asserted. On the next rising edge of the eight megahertz clock, the column address on address lines A8 through A14 is on the memory address lines. The multiplexed address is valid only when RAS or CAS is present, making an enable for the multiplexers unnecessary.

Memory refresh is controlled by U11, a MC6840 programmable timer module (PTM). Once programmed, the PTM timer used (the PTM has three timers) causes a level seven interrupt every 1.9 milliseconds (2 milliseconds — routine execution time). This interrupt enables all four banks of memory for simultaneous refresh.

Interrupts with M6800 type peripherals are handled with a reference to the internal vector table. Figure 3 is a schematic of the hardware used with the MC68000 to create a vectored interrupt (level one to level seven). The level present on the IPL0, IPL1, and IPL2 lines is checked against the interrupt level of the processor. If it is higher than the internal level, an interrupt sequence is started. The function code outputs will be high and address lines A1, A2, and A3 will be the vector number of the interrupt being serviced (in this case, all high). Now decoders U1 and U3 (Figure 1) decode the level seven interrupt and generate valid peripheral address (VPA) to the MC68000 through U13 and U9. The assertion of valid peripheral address causes the internal vector table entry for level seven to be fetched and used as the starting location of the service routine. At the same time, U12 and U13 enable all RAS signals and disable CAS for refresh of the memories.

OPTIONAL HARDWARE

One situation may occur with the memory card where data might be lost. If the reset button is held closed too long, data could be lost. To prevent this, the circuitry shown in Figure 4 can be added. This provides for a single E cycle reset which will retain the integrity of the stored data.

When power is initially applied to the MC68000, a reset must occur for at least 100 milliseconds after the supply voltage has reached 4.75 volts for proper power-up reset. This means that a one shot or a resistor-capacitor combination should be used to hold the clear pin of the flip flops at or below the logic low level (0.8 volts) for the required time. The E signal will clock the 2-bit counter twice. This presets flip flop U3, removing the system reset. On a non-powerup reset, the reset switch is closed, clocking a low into flip flop U3. Gate U4 provides debounce of the reset switch, allowing only one clock pulse into flip flop U3. Again, E will clock the counter removing reset.

SOFTWARE

Row address select-only refresh is the refresh method used in this application. It is accomplished by a hardware enable (level seven interrupt) and 128 NOPs for the service routine. The level seven interrupt being low enables all four RAS signals and disables CAS. Each NOP increments the address bus to provide the 128 row addresses (0 to 127) needed for refreshing all four banks of memory. Incrementing the address bus accesses and refreshes that row.

However, this has one problem — reset. If a reset occurs just prior to an interrupt for software refresh, data could be lost due to a late or missing refresh cycle. This problem is solved by locating the software refresh routine at the beginning of the reset code. A hardware enable for reset refresh enables RAS-only refresh in the same way that the level seven interrupt signal did for a normal refresh. In addition, the refresh at reset must load the stack with a valid return address, to return to when the return from interrupt (RTI) instruction is executed at the end of the refresh routine. Figure 5 is a listing of this software with comments to document the reset refresh.

Refresh is enabled at restart by U10 and U13. All RAS signals are on and all CAS signals are off. Like a normal refresh operation, CAS is enabled by the first access to memory after the refresh routine. Software refresh with the MC68000 is an efficient option to implement dynamic RAM without costly delay lines. The application presented here has only a five percent program time overhead.

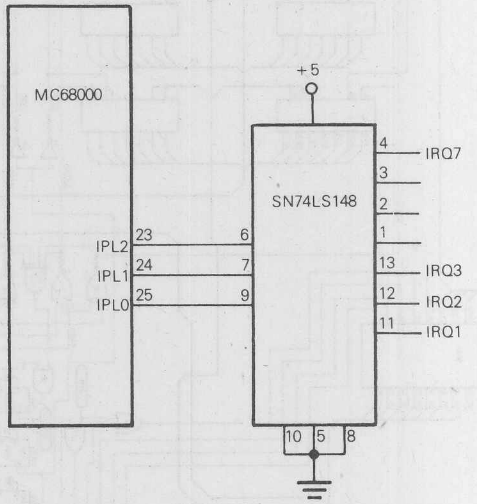
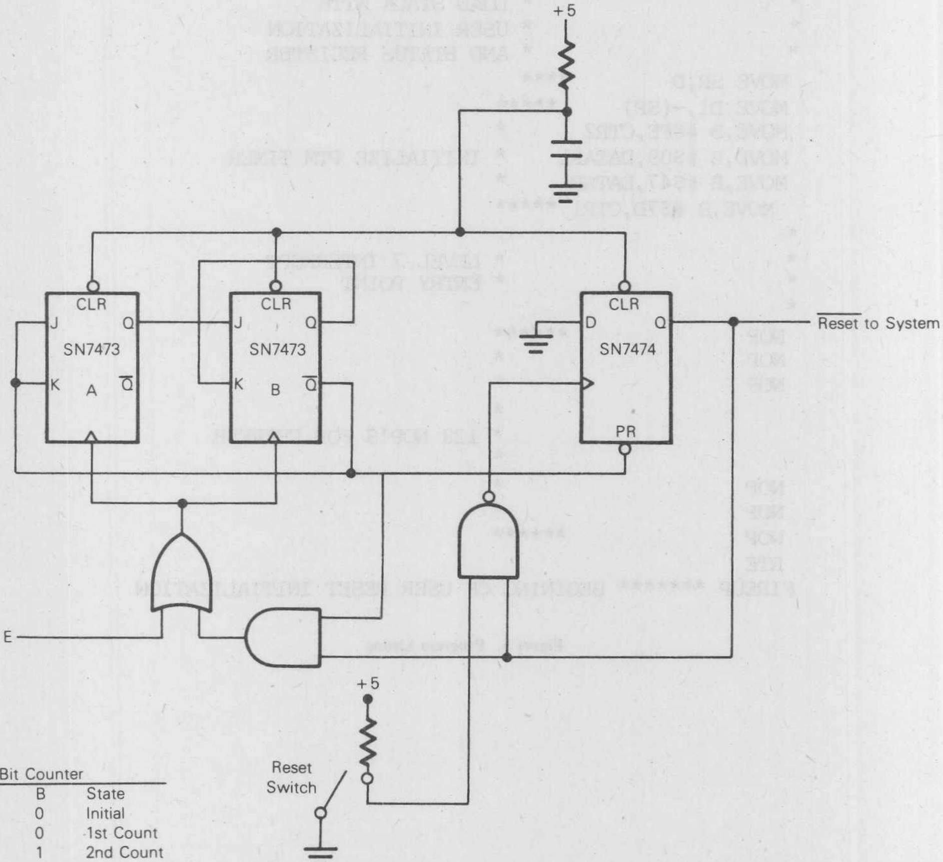


Figure 3. Single-Cycle Reset Circuit — Schematic Diagram



2 Bit Counter

A	B	State
0	0	Initial
1	0	1st Count
0	1	2nd Count
0	0	Reset

Figure 4. Vectored Interrupt Circuit — Schematic Diagram

```
PAGE 001   S68K     .SA:0

DATAR1 EQU $18005
DATAR  EQU $18007
CTR1   EQU $18001
CTR2   EQU $18003
 PEA FIREUP        ****
*                     * LOAD STACK WITH
*                     * USER INITIALIZATION
*                     * AND STATUS REGISTER
 MOVE SR,D         ****
 MOVE D1,-(SP)      *****
 MOVE,B #$FE,CTR2       *
 MOVE,B #$09,DATAR1     * INITIALIZE PTM TIMER
 MOVE,B #$47,DATAR      *
  MOVE,B #$7D,CTR1 *****
*
*                       * LEVEL 7 INTERRUPT
*                       * ENTRY POINT
*
 NOP               ******
 NOP                    *
 NOP                    *
                        *
                        * 128 NOP'S FOR REFRESH
                        *
 NOP                    *
 NOP                    *
 NOP               ******
 RTE
FIREUP ******* BEGINING OF USER RESET INITIALIZATION
```

Figure 5. Program Listing

AN-817

ASYNCHRONOUS COMMUNICATIONS FOR THE MC68000 USING THE MC6850

Prepared by:
Charles Melear
Microprocessor Applications Engineer

Interfacing the MC6850 Asynchronous Communications Interface Adapter (ACIA) to the MC68000 is easy due to the fact that the MC68000 has a special cycle to handle M6800 peripherals. The ACIA data bus can be placed on either the upper or lower eight bits of the MC68000 data bus with equivalent results. Using the upper byte implies an even address and use of the upper data strobe ($\overline{UDS}$), and the lower byte implies an odd address and the use of the lower data strobe ($\overline{LDS}$). In this application, the ACIA is placed on the lower byte of the data bus.

INTERCONNECTIONS

Enable (E) and R/$\overline{W}$ are connected to the corresponding pins of the MC68000. Several signals are generated to form chip selects as shown in Figure 1. Valid memory address ($\overline{VMA}$) from the MC68000 is an active low signal (as opposed to active high for the MC6800) as well as $\overline{LDS}$. The NOR of the two signals is used to develop CS1. The address $F3FFXX is generated by address lines A8 through A23 to enable a SN74LS154 four-to-sixteen line selector. Address lines A4 through A7 are used to generate a low output at 02 of the SN74LS154 to be used for CS2 of the ACIA. Address line A1 is used for the register select (RS) pin of the ACIA. This puts the ACIA status register at address $F3FF21 and the control register at address $F3FF23. If the ACIA has been placed on the upper byte, the addresses would be $F3FF20 and $F3FF22, respectively. To complete the circuit, a signal called valid peripheral address ($\overline{VPA}$) must be generated and returned to the MC68000 to indicate that a M6800 cycle is being executed. The SN74LS154 has two active low chip enable lines which are driven by the gates that form address $F3FFXX from address lines A8 through A23. Since the SN74LS154 always picks M6800 peripherals, the two chip enable lines can be ORed to develop $\overline{VPA}$. Since more than 16 peripherals could exist, it is best to make the device actually driving the $\overline{VPA}$ line an open collector output so that several gates can be wire ORed.

OPERATION

Operating the ACIA is relatively easy as shown in the flow chart given in Figure 2. Once the control register is set up, the status register is monitored for receive data register full (RDRF) and transmit data register empty (TDRE) indications, as well as error signals and handshake lines. The handshake lines such as request to send ($\overline{RTS}$), clear to send (CTS), and data carrier detect ($\overline{DCD}$) indicate which conditions are present so that the MPU can ascertain when transmission can occur. Once all conditions are ready, transmission or reception or both can begin.

A sample program is given in Figure 3 that shows the MC68000 receiving a character from a terminal through the ACIA and then echoing that character back to the terminal. Essentially, the MC68000 checks to see that transmission and reception can occur. The status register is polled until a character is received. The character is read and then written back to the ACIA for transmission to the terminal as soon as the transmit data register is empty. Of course, any number of subroutines or additional code could be executed before looking for the next character from the ACIA.

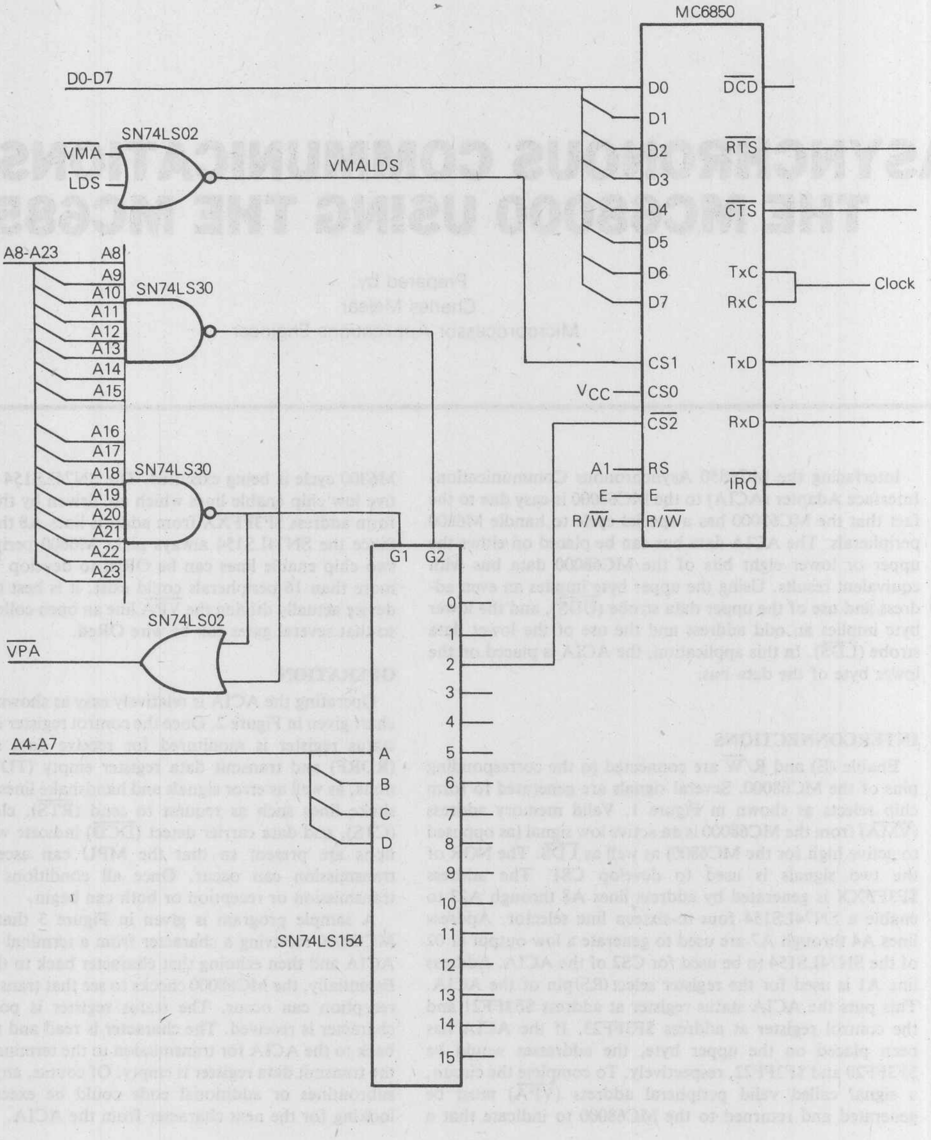


Figure 1. MC6800 to MC6850 Interconnections

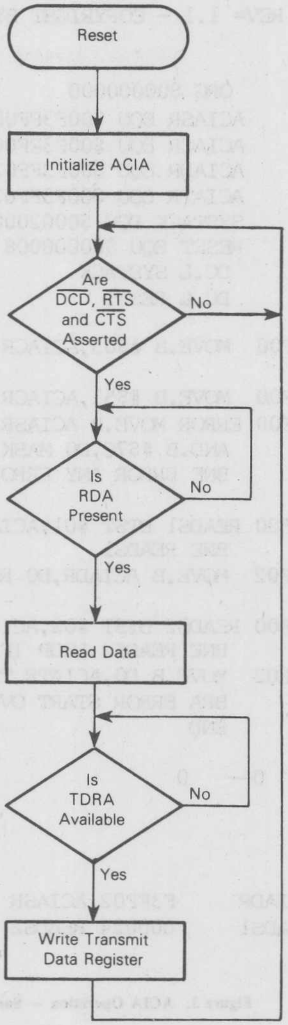
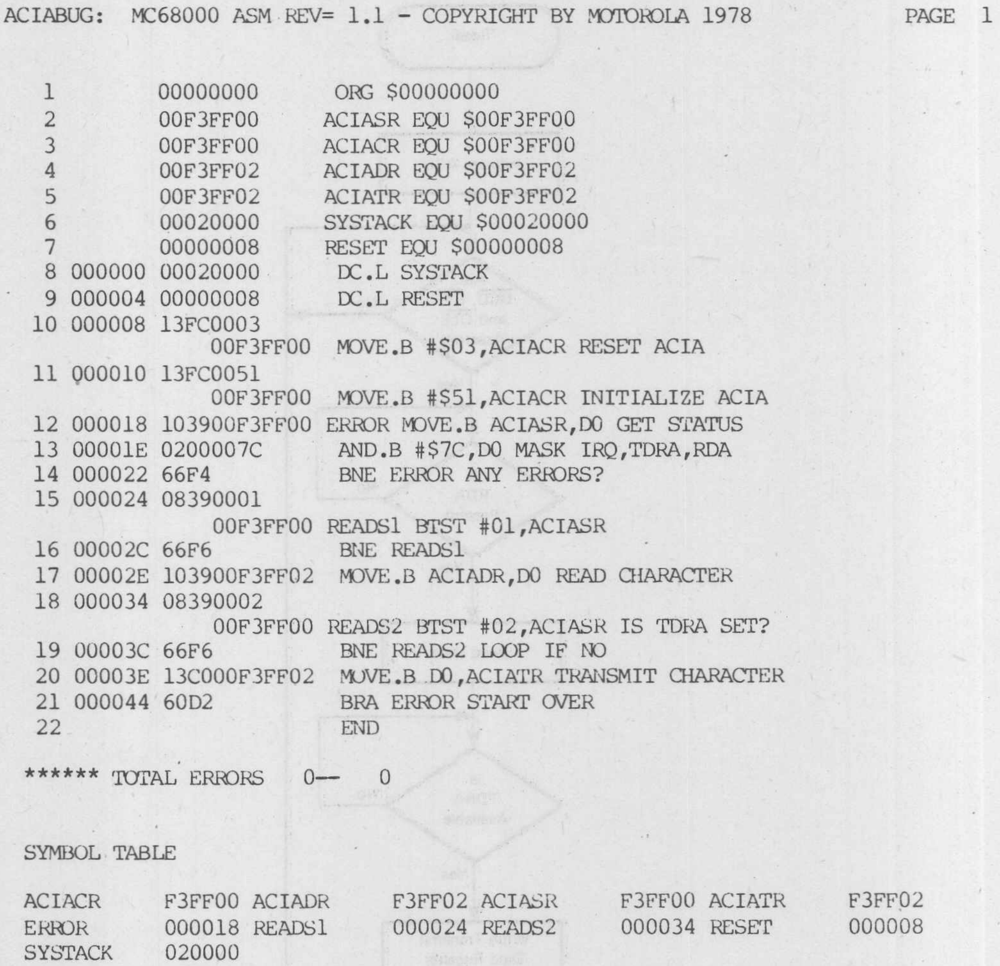

Reset
Initialize ACIA
Are $\overline{DCD}$, $\overline{RTS}$ and $\overline{CTS}$ Asserted
No
Yes
Is RDA Present
No
Yes
Read Data
Is TDRA Available
No
Yes
Write Transmit Data Register

Figure 2. ACIA Operation — Flow Chart



```
ACIABUG:  MC68000 ASM REV= 1.1 - COPYRIGHT BY MOTOROLA 1978            PAGE  1

  1        00000000      ORG $00000000
  2        00F3FF00     ACIASR EQU $00F3FF00
  3        00F3FF00     ACIACR EQU $00F3FF00
  4        00F3FF02     ACIADR EQU $00F3FF02
  5        00F3FF02     ACIATR EQU $00F3FF02
  6        00020000     SYSTACK EQU $00020000
  7        00000008     RESET EQU $00000008
  8 000000 00020000      DC.L SYSTACK
  9 000004 00000008      DC.L RESET
 10 000008 13FC0003
               00F3FF00  MOVE.B #$03,ACIACR RESET ACIA
 11 000010 13FC0051
               00F3FF00  MOVE.B #$51,ACIACR INITIALIZE ACIA
 12 000018 103900F3FF00 ERROR MOVE.B ACIASR,D0 GET STATUS
 13 00001E 0200007C      AND.B #$7C,D0 MASK IRQ,TDRA,RDA
 14 000022 66F4          BNE ERROR ANY ERRORS?
 15 000024 08390001
               00F3FF00 READS1 BTST #01,ACIASR
 16 00002C 66F6          BNE READS1
 17 00002E 103900F3FF02  MOVE.B ACIADR,D0 READ CHARACTER
 18 000034 08390002
               00F3FF00 READS2 BTST #02,ACIASR IS TDRA SET?
 19 00003C 66F6          BNE READS2 LOOP IF NO
 20 00003E 13C000F3FF02  MOVE.B D0,ACIATR TRANSMIT CHARACTER
 21 000044 60D2          BRA ERROR START OVER
 22                      END

****** TOTAL ERRORS   0--   0


SYMBOL TABLE

ACIACR     F3FF00 ACIADR     F3FF02 ACIASR     F3FF00 ACIATR     F3FF02
ERROR      000018 READS1     000024 READS2     000034 RESET      000008
SYSTACK    020000
```

Figure 3. ACIA Operation — Sample Program

AN-818

SYNCHRONOUS I/O FOR THE MC68000 USING THE MC6852

Prepared by:
James McKenzie
Microprocessor Applications Engineering

The MC6852 Synchronous Serial Data Adapter (SSDA) provides both a synchronous serial transmitter and synchronous serial receiver in a single, 24-pin device. Synchronous data communications is inherently more efficient than asynchronous data communications because each character need not be framed for error detection. Hence, synchronous data communications lends itself to higher data rates and applications which are synchronous in nature, such as serial communications between synchronous processors.

The SSDA is particularly well-suited for data communications applications involving byte-oriented protocols such as Bisync. Both the SSDA transmitter and receiver are interfaced to a single 8-bit bidirectional data bus. Data to be transmitted is loaded from the MPU data bus into a 3-byte FIFO on the SSDA. An 8-bit shift register is used to serially transmit data from the last FIFO location; parity may also be appended. Received data enters another 8-bit shift register where parity may be checked. Data from the shift register enters a 3-byte receiver FIFO which presents the data in parallel form to the MPU bus.

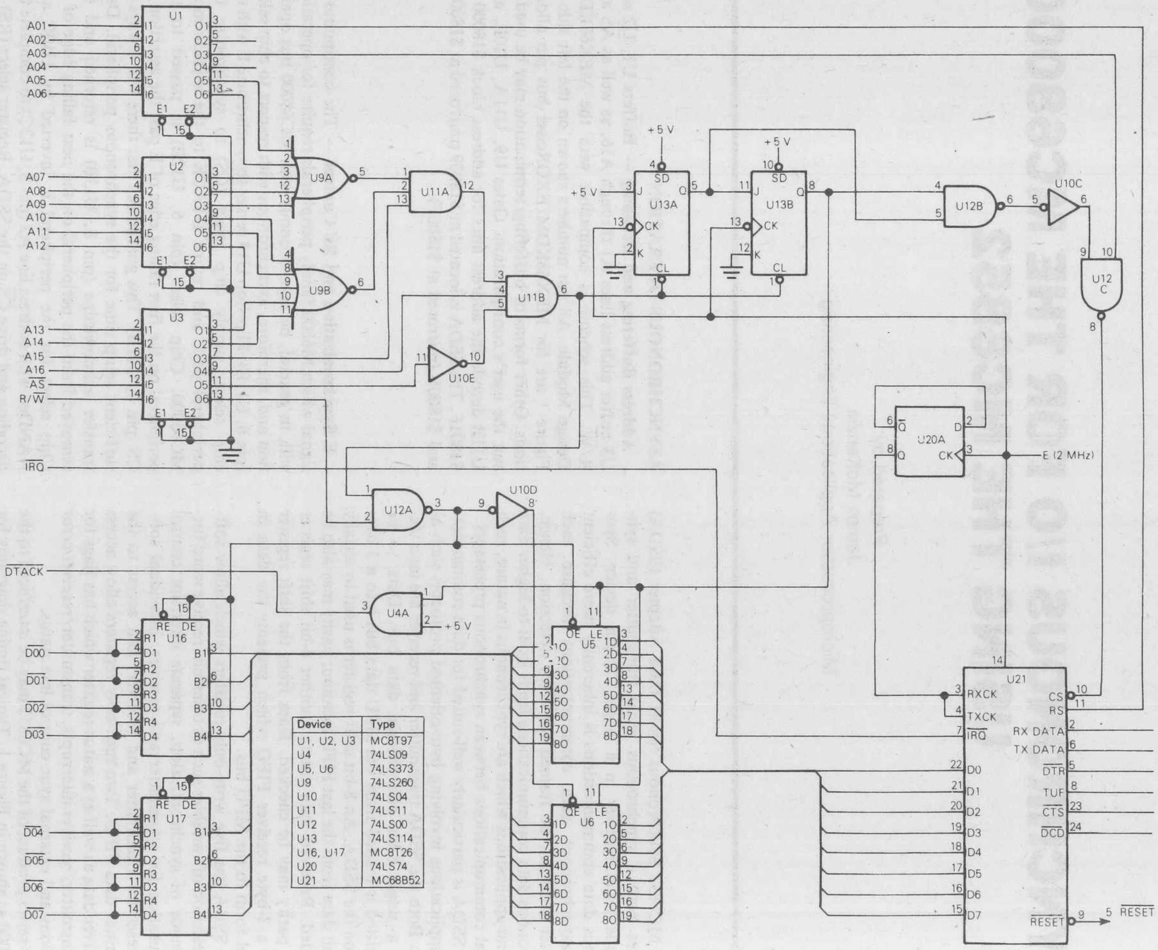
The SSDA has five write-only registers which allow software selection of variables such as transmit/receive word format, mode of synchronization, separate interrupt control configuration for transmitter and receiver, individual software reset for transmitter and receiver, and access to the transmitter data FIFO. Two read-only registers allow access to receiver data as well as a status register which has flags for the transmitter/receiver interrupts, transmitter/receiver error conditions and external sync control line status.

Any series product of the MC6852 may be interfaced to the MC68000 as shown in Figure 1. Typical timing diagrams for this interface (B part only) appear in Figure 2.

ASYNCHRONOUS OPERATION

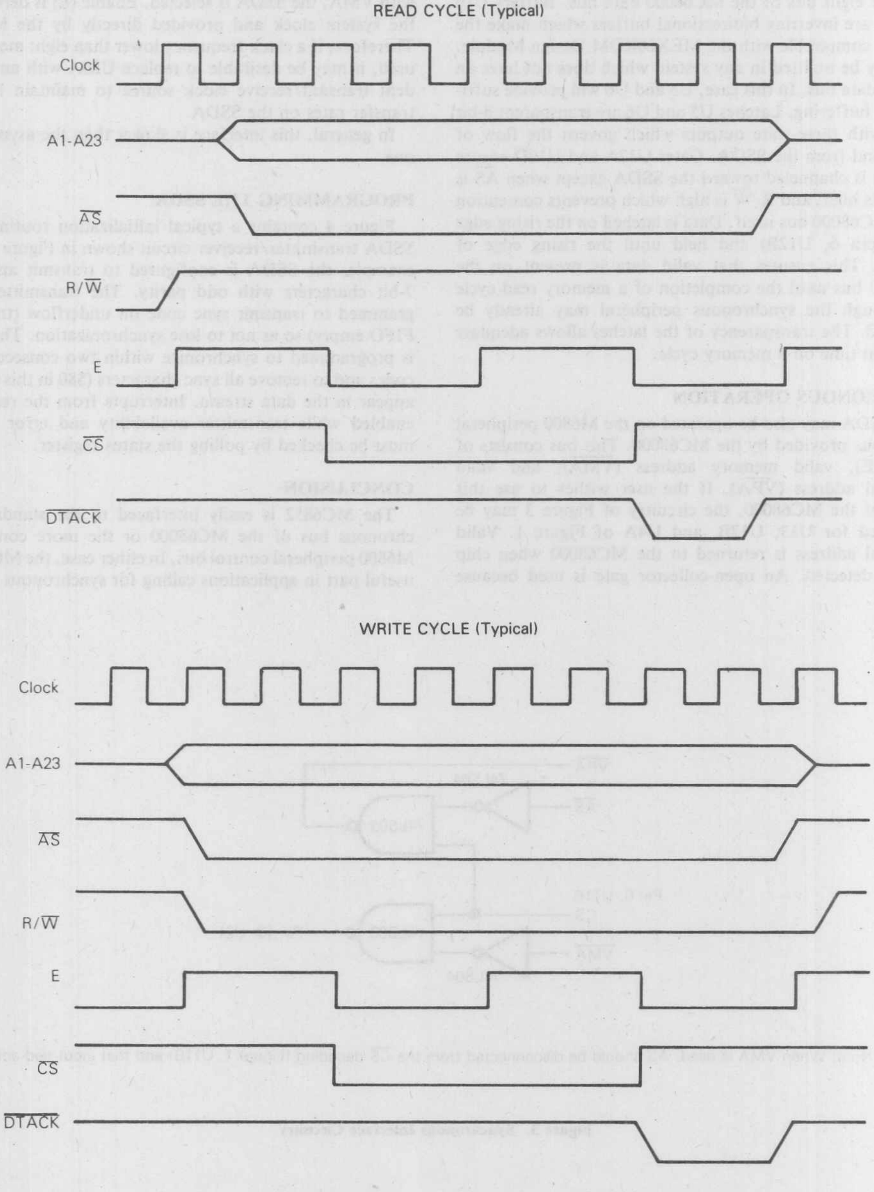
Address Buffering and Decoding — Buffers U1, U2 and U3 buffer address lines A1 through A16, as well as $\overline{\text{AS}}$ and R/$\overline{\text{W}}$. This scheme is compatible with the MEX68KDM Design Module. All pin numbers shown on the left side of Figure 1 are for MEX68KDM/EXORciser bus pin allocations. Other forms of buffering/termination may be used to suit the user's configuration. Gates U9, U11A, U10E, and U11B decode the address lines for address block $18000 to $1801F. The SSDA is located at $18009 (mirrored at $1800D) and $1800B (mirrored at $1800F).

E Synchronization and RS Control — The continuous E signal which M6800 family peripherals require for operation will, in general, be asynchronous with MC68000 bus operation and, therefore, asynchronous with respect to chip select (pin 6, U11B). Flip-flop U13 serves to synchronize E with the chip select, supply chip select ($\overline{\text{CS}}$) to synchronize the peripheral part, and return $\overline{\text{DTACK}}$ to the asynchronous MC68000. Chip select (pin 6, U12B) is passed to the peripheral on the first falling edge of E past the assertion of CS (pin 6, U11B). This guarantees that there will always be sufficient setup time for the synchronous peripheral. Data transfer acknowledge (pin 8, U13B) is returned and $\overline{\text{CS}}$ removed from the peripheral on the next falling edge of E. Chip select for the peripheral is inverted by U10C and NANDed with address line A3 (pin 8, U12C) to complete the decoding and drive $\overline{\text{CS}}$ on the SSDA. Register select (RS) on the SSDA is driven by address line A1.



Device	Type
U1, U2, U3	MC8T97
U4	74LS09
U5, U6	74LS373
U9	74LS260
U10	74LS04
U11	74LS11
U12	74LS00
U13	74LS114
U16, U17	MC8T26
U20	74LS74
U21	MC68B52

Figure 1. MC68000/MC68B52 Interface — Schematic Diagram


READ CYCLE (Typical)

WRITE CYCLE (Typical)

Figure 2. MC68000/MC68B52 Interface — Timing Diagrams

Data Bus — The SSDA shown in Figure 1 is interfaced to the lower eight bits of the MC68000 data bus. Buffers U16 and U17 are inverting bidirectional buffers which make the interface compatible with the MEX68KDM Design Module. They may be omitted in any system which does not have an inverted data bus. In this case, U5 and U6 will provide sufficient bus buffering. Latches U5 and U6 are transparent 8-bit latches with three-state outputs which govern the flow of data to and from the SSDA. Gates U12A and U10D ensure that data is channeled toward the SSDA except when $\overline{\text{AS}}$ is low, CS is high, and R/$\overline{\text{W}}$ is high which prevents contention on the MC68000 bus itself. Data is latched on the rising edge of $\overline{\text{CS}}$ (pin 6, U12B) and held until the rising edge of $\overline{\text{DTACK}}$. This ensures that valid data is present on the MC68000 bus until the completion of a memory read cycle even though the synchronous peripheral may already be deselected. The transparency of the latches allows adequate data setup time on a memory cycle.

SYNCHRONOUS OPERATION

The SSDA may also be operated on the M6800 peripheral control bus provided by the MC68000. This bus consists of enable (E), valid memory address ($\overline{\text{VMA}}$), and valid peripheral address ($\overline{\text{VPA}}$). If the user wishes to use this feature of the MC68000, the circuitry of Figure 3 may be substituted for U13, U12B, and U4A of Figure 1. Valid peripheral address is returned to the MC68000 when chip select is detected. An open-collector gate is used because $\overline{\text{VPA}}$ is a wire ORed signal. When the processor repsonds with $\overline{\text{VMA}}$, the SSDA is selected. Enable (E) is derived from the system clock and provided directly by the MC68000. Therefore, if a clock frequency lower than eight megahertz is used, it may be desirable to replace U20A with an independent transmit/receive clock source to maintain high data transfer rates on the SSDA.

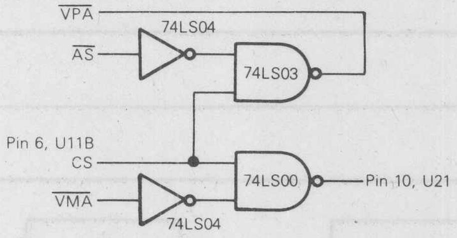
In general, this interface is slower than the asynchronous one.

PROGRAMMING THE SSDA

Figure 4 contains a typical initialization routine for the SSDA transmitter/receiver circuit shown in Figure 1. In this example, the SSDA is configured to transmit and receive 7-bit characters with odd parity. The transmitter is programmed to transmit sync code on underflow (transmitter FIFO empty) so as not to lose synchronization. The receiver is programmed to synchronize within two consecutive sync codes and to remove all sync characters ($80 in this case) that appear in the data stream. Interrupts from the receiver are enabled while transmitter availability and error detection must be checked by polling the status register.

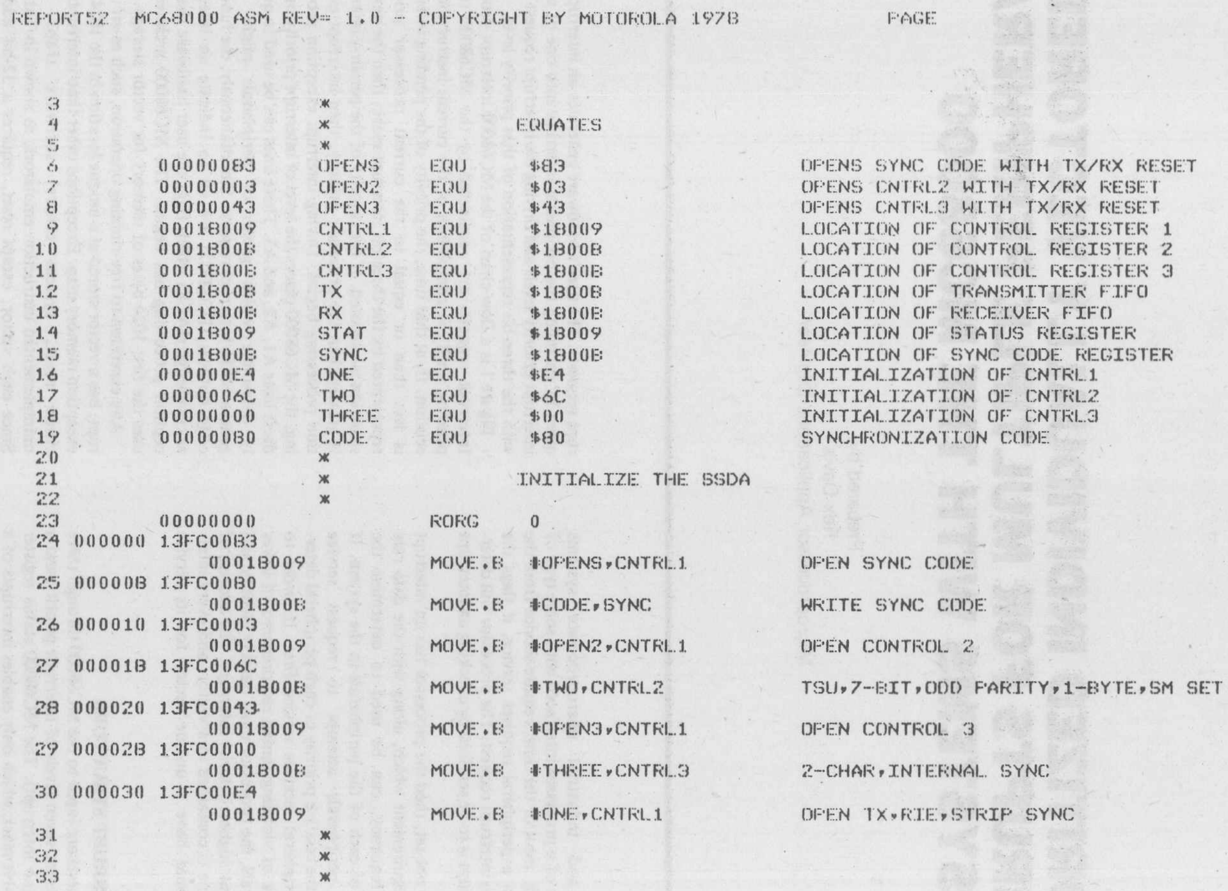
CONCLUSION

The MC6852 is easily interfaced to the standard asynchronous bus of the MC68000 or the more conventional M6800 peripheral control bus. In either case, the MC6852 is a useful part in applications calling for synchronous protocol.



Note: When $\overline{\text{VMA}}$ is used, $\overline{\text{AS}}$ should be disconnected from the $\overline{\text{CS}}$ decoding (Figure 1, U11B) and that input tied active.

Figure 3. Synchronous Interface Circuitry



```
REPORT52  MC68000 ASM REV= 1.0 - COPYRIGHT BY MOTOROLA 1978             PAGE

  3                        *
  4                        *          EQUATES
  5                        *
  6         00000083       OPENS    EQU    $83              OPENS SYNC CODE WITH TX/RX RESET
  7         00000003       OPEN2    EQU    $03              OPENS CNTRL2 WITH TX/RX RESET
  8         00000043       OPEN3    EQU    $43              OPENS CNTRL3 WITH TX/RX RESET
  9         0001B009       CNTRL1   EQU    $1B009           LOCATION OF CONTROL REGISTER 1
 10         0001B00B       CNTRL2   EQU    $1B00B           LOCATION OF CONTROL REGISTER 2
 11         0001B00B       CNTRL3   EQU    $1B00B           LOCATION OF CONTROL REGISTER 3
 12         0001B00B       TX       EQU    $1B00B           LOCATION OF TRANSMITTER FIFO
 13         0001B00B       RX       EQU    $1B00B           LOCATION OF RECEIVER FIFO
 14         0001B009       STAT     EQU    $1B009           LOCATION OF STATUS REGISTER
 15         0001B00B       SYNC     EQU    $1B00B           LOCATION OF SYNC CODE REGISTER
 16         000000E4       ONE      EQU    $E4              INITIALIZATION OF CNTRL1
 17         0000006C       TWO      EQU    $6C              INITIALIZATION OF CNTRL2
 18         00000000       THREE    EQU    $00              INITIALIZATION OF CNTRL3
 19         00000080       CODE     EQU    $80              SYNCHRONIZATION CODE
 20                        *
 21                        *          INITIALIZE THE SSDA
 22                        *
 23         00000000                RORG   0
 24  000000 13FC0083
            0001B009                MOVE.B #OPENS,CNTRL1    OPEN SYNC CODE
 25  00000B 13FC0080
            0001B00B                MOVE.B #CODE,SYNC       WRITE SYNC CODE
 26  000010 13FC0003
            0001B009                MOVE.B #OPEN2,CNTRL1    OPEN CONTROL 2
 27  00001B 13FC006C
            0001B00B                MOVE.B #TWO,CNTRL2      TSU,7-BIT,ODD PARITY,1-BYTE,SM SET
 28  000020 13FC0043
            0001B009                MOVE.B #OPEN3,CNTRL1    OPEN CONTROL 3
 29  00002B 13FC0000
            0001B00B                MOVE.B #THREE,CNTRL3    2-CHAR,INTERNAL SYNC
 30  000030 13FC00E4
            0001B009                MOVE.B #ONE,CNTRL1      OPEN TX,RIE,STRIP SYNC
 31                        *
 32                        *
 33                        *
```

Figure 4. Initialization Routine

AN-819

PRIORITIZED INDIVIDUALLY VECTORED INTERRUPTS FOR MULTIPLE PERIPHERAL SYSTEMS WITH THE MC68000

Prepared by:
Rex Davis
Microprocessor Applications Engineer

Commercial and industrial microprocessor systems typically consists of a processor interfaced with some type of peripherals which, most of the time, require service from the processor. When a peripheral requires service, it flags the processor with an interrupt request. The processor will determine if the interrupt is to be serviced by checking an interrupt mask.

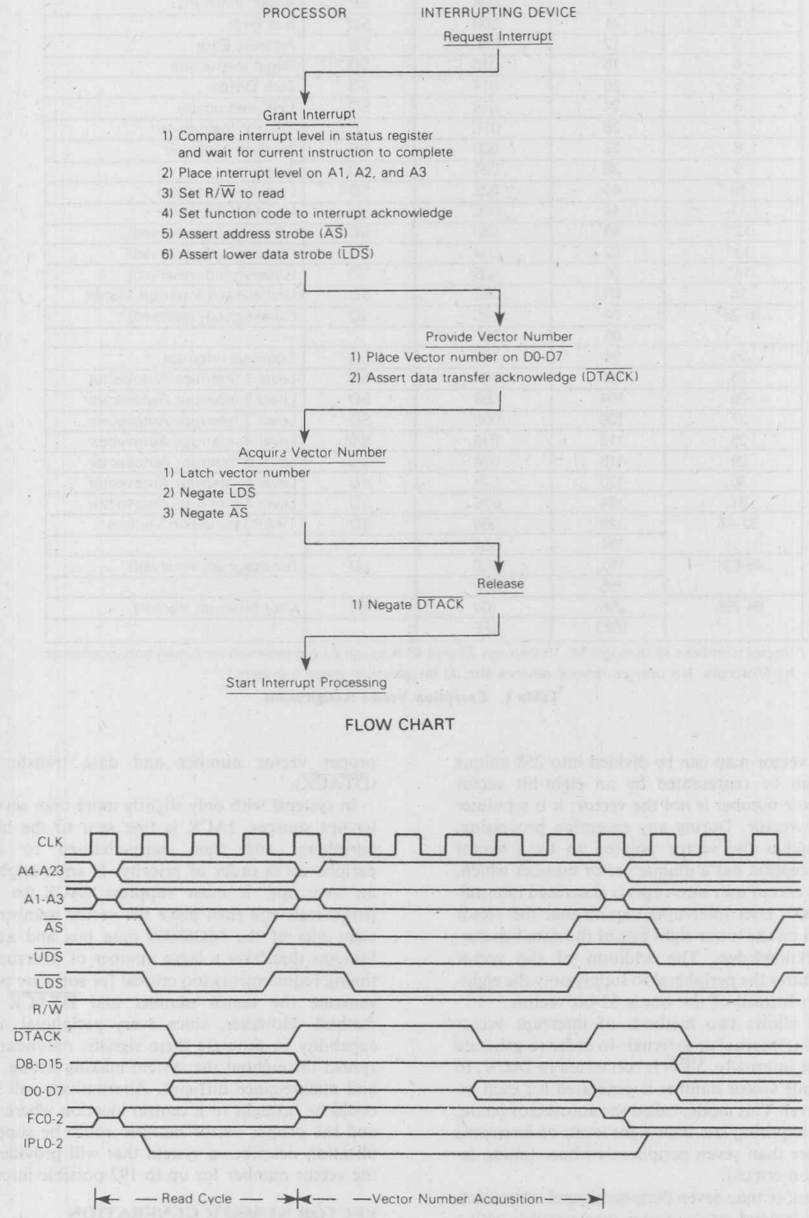
If the mask is not set, then the processor has an interrupt service timing requirement which, along with the data rate and interrupt frequency, can be used to determine the relative priority of each of the peripherals in the system. If two or more peripherals attempt to request service simultaneously, the relative priority of each peripheral determines which peripheral receives service first. In order to minimize the risk of violating timing requirements of lower priority peripherals, the processor must quickly identify and service the current highest priority interrupt. The address of the service routine is contained in a vector; and every interrupt source should have a unique vector for its service routine.

MC68000 INTERRUPT STRUCTURE

Interrupt requests are input to the MC68000 through three pins which represent seven levels of interrupt priority and a quiescent state (no interrupt). The MC68000 status register contains a three-bit mask which only enables interrupts of a higher priority than the level represented in the three-bit mask. The interrupt mask can be modified under software control, thereby increasing user control of peripheral interrupt requests. The user's hardware generates an interrupt request by encoding the peripheral interrupt into one of seven interrupt priority levels and driving the interrupt request lines with the three-bit representation of that priority level.

Figure 1 is a flow chart of the MC68000 interrupt process. Interrupt requests are considered by the MC68000 to be pending until the completion of the current instruction execution. If, at that time, the priority of the pending interrupt is less than or equal to the current processor priority represented by the three-bit interrupt mask, then the next instruction is executed. If the priority of the pending interrupt is greater than the processor priority, then interrupt exception processing begins. During interrupt exception processing, the MC68000 places the level of interrupt priority on address lines A1, A2, and A3. These lines can be used to quickly determine which group of peripherals might have generated the interrupt request. Simultaneously, the function code outputs (FC0-FC2) are set to indicate an interrupt acknowledge (IACK) which flags the user hardware that exception processing has begun. The MC68000 architecture uses the first 1024 bytes of memory for vector storage.

Any exception to free-running operation, such as an interrupt, has a vector stored at a unique location in the 1024 byte exception memory map. Exceptions other than interrupts, include reset, system errors, software traps, and unimplemented instruction emulators, as shown in Table 1. Since each vector, except reset, requires a 32-bit address, four bytes are required to store each vector. Reset is a special case which requires two 32-bit addresses or eight bytes of memory.


PROCESSOR
INTERRUPTING DEVICE
Request Interrupt
Grant Interrupt
1) Compare interrupt level in status register and wait for current instruction to complete
2) Place interrupt level on A1, A2, and A3
3) Set R/W̄ to read
4) Set function code to interrupt acknowledge
5) Assert address strobe (ĀS̄)
6) Assert lower data strobe (L̄D̄S̄)
Provide Vector Number
1) Place Vector number on D0-D7
2) Assert data transfer acknowledge (D̄T̄ĀC̄K̄)
Acquire Vector Number
1) Latch vector number
2) Negate L̄D̄S̄
3) Negate ĀS̄
Release
1) Negate D̄T̄ĀC̄K̄
Start Interrupt Processing
CLK
A4-A23
A1-A3
ĀS̄
ŪD̄S̄
L̄D̄S̄
R/W̄
D̄T̄ĀC̄K̄
D8-D15
D0-D7
FC0-2
IPL0-2
Read Cycle
Vector Number Acquisition

FLOW CHART

TIMING DIAGRAM

Figure 1. MC68000 Interrupt Acknowledge Sequence — Flow Chart and Timing Diagram

Vector Number(s)	Address			Assignment
	Dec	Hex	Space	
0	0	000	SP	Reset: Initial SSP
1	4	004	SP	Reset: Initial PC
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12*	48	030	SD	(Unassigned, reserved)
13*	52	034	SD	(Unassigned, reserved)
14*	56	038	SD	(Unassigned, reserved)
15	60	03C	SD	Uninitialized Interrupt Vector
16-23*	64	04C	SD	(Unassigned, reserved)
	95	05F		—
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32-47	128	080	SD	TRAP Instruction Vectors
	191	0BF		—
48-63*	192	0C0	SD	(Unassigned, reserved)
	255	0FF		—
64-255	256	100	SD	User Interrupt Vectors
	1023	3FF		—

*Vector numbers 12 through 14, 16 through 23 and 48 through 63 are reserved for future enhancements by Motorola. No user peripheral devices should be assigned these numbers.

Table 1. Exception Vector Assignments

The exception vector map can be divided into 255 unique vectors which can be represented by an eight-bit vector number. The vector number is not the vector; it is a pointer to one particular vector. During any exception processing, the MC68000 fetches the vector pointed to by a vector number. Each exception has a unique vector number which, for all exceptions except user interrupts is generated internally by the MC68000. User interrupts require that the vector number be placed on the lower eight bits of the data bus during interrupt acknowledge. The addition of the vector number fetch requires the peripheral to supply only the eight-bit vector number instead of the whole 32-bit vector.

The MC68000 allows two methods of interrupt vector number generation, internal or external. In order to generate the vector number internally, $\overline{\text{VPA}}$ is connected to IACK. In this mode, a unique vector number is generated for each interrupt priority level. This mode, called the autovector mode, is ideal for users requiring less than eight levels of interrupts or users with more than seven peripherals whose timing requirements are non-critical.

For users with more than seven peripherals and whose timing requirements demand service faster than possible with a software polling method, the MC68000 provides an additional 192 interrupts which require external vector number generation. In this case, IACK is not connected to $\overline{\text{VPA}}$; instead, it is used by external hardware to determine that a vector number is needed by the MC68000 and to provide the proper vector number and data transfer acknowledge ($\overline{\text{DTACK}}$).

In systems with only slightly more than seven possible interrupt sources, IACK is first sent to the highest priority peripheral and then daisy-chained to all remaining peripherals in order of priority. If any peripheral generates an interrupt, it must suppress IACK to all remaining peripherals and then place the vector number on the lower eight bits of the MC68000 data bus and assert $\overline{\text{DTACK}}$. Systems that have a large number of interrupt sources and timing requirements too critical for software polling can also generate the vector number and $\overline{\text{DTACK}}$ by the same method. However, since every peripheral must have the capability to generate these signals, redundant hardware is spread throughout the system making debug, modification, and maintenance difficult. Alternatively, all interrupt lines could be brought to a central location where both $\overline{\text{DTACK}}$ and the proper vector number could be supplied. The application describes a system that will provide $\overline{\text{DTACK}}$ and the vector number for up to 192 possible interrupt sources.

VECTOR NUMBER GENERATION

The 192 user interrupt vectors are referenced by sequential vector numbers 64 through 255. Therefore, if fewer than 193 interrupts are required, each interrupt can be assigned a unique vector number which can also be interpreted as a priority. Vector number 64 can be assumed to have the lowest priority and vector number 255 the highest. The 192 levels of externally-generated priority can be represented by eight bits (00000000-10111111). The vector number is the externally-generated priority offset by 64; the vector number can be generated by encoding the interrupt to its priority and then adding 64. This is essentially the same format that is used in the autovectors.

Figure 2 is a block diagram of such a system. All circuitry, except the processor, can be located in one area rather than spread throughout the system. Note that two sets of latches have been inserted to guarantee that no interrupts are lost and that the vector number that is placed on the data bus is the result of only one interrupt. Otherwise, if an interrupt request is generated during interrupt acknowledge, it could cause the vector number to be in a state of transition when the MC68000 is attempting to latch the vector number from the data bus. Latch number one prohibits any new interrupts from being accepted until the vector number has been latched by latch number two. Latch number two isolates the vector number from the data bus until IACK is asserted. After a delay sufficient to allow the vector number to propagate to latch number two, latch number one is released to allow new interrupts to be accepted.

IMPLEMENTATION

The circuitry shown in Figure 3 performs all the tasks necessary to provide vector numbers for up to 192 possible interrupt sources. The 192 interrupt request lines are divided into 24 groups of eight which are input through a SN74LS373 octal latch to a SN74LS148 8-to-3 encoder. The SN74LS148 encoders are daisy-chained so that each stage can disable all succeeding stages which effectively prioritizes the interrupts by group. Within each group, interrupts are prioritized into eight levels which the encoder represents with a three-bit encoded number on lines A0, A1, and A2.

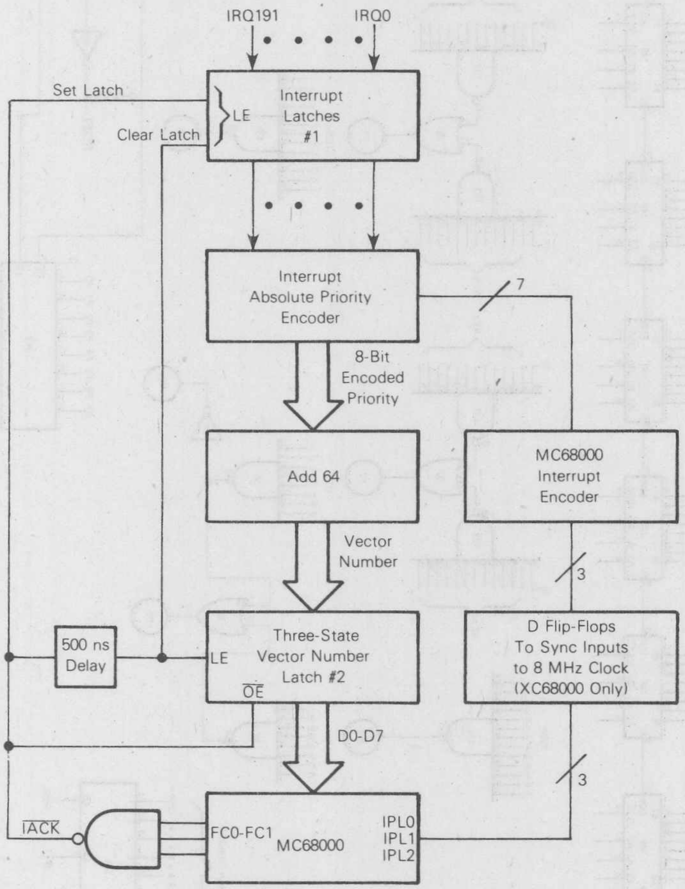

IRQ191
IRQ0
Set Latch
Clear Latch
LE
Interrupt Latches #1
Interrupt Absolute Priority Encoder
7
8-Bit Encoded Priority
Add 64
MC68000 Interrupt Encoder
Vector Number
3
500 ns Delay
LE
Three-State Vector Number Latch #2
OE
D Flip-Flops To Sync Inputs to 8 MHz Clock (XC68000 Only)
D0-D7
3
IACK
FC0-FC1
MC68000
IPL0
IPL1
IPL2

Figure 2. Vector Number Generation Circuit — Block Diagram

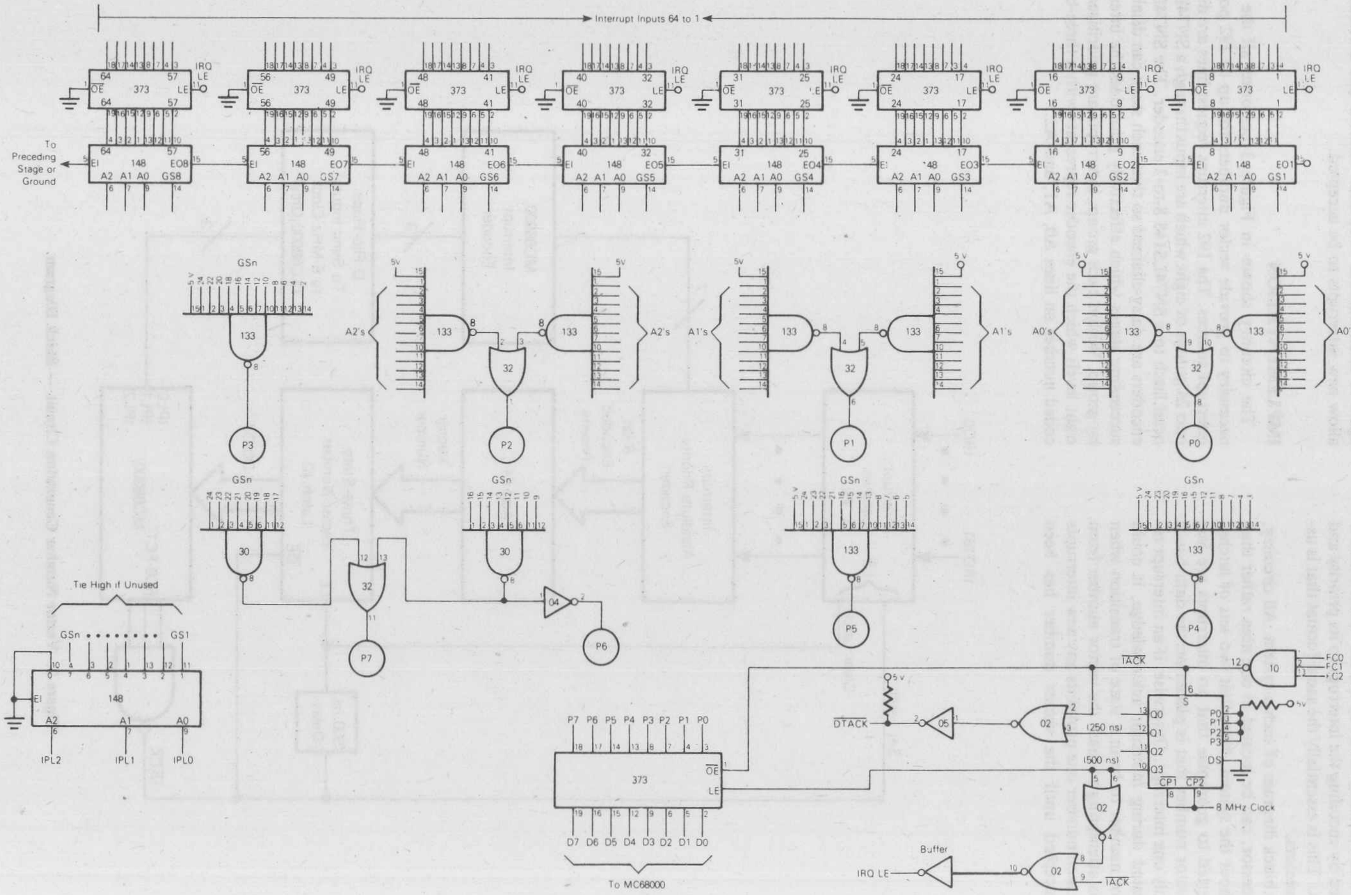


Figure 3. Vector Number Generation Circuit — Schematic Diagram

The A0 line from each of the 24 groups is NANDed to form the A0 of the vector number. The A1 and A2 lines are handled in an identical manner. Bits 3 and 7 of the encoded interrupt are made by NANDing selected GS outputs of the encoders. Bits 6 and 7 of the vector number differ from bits 6 and 7 of the encoded interrupt because of the offset of 64.

After the vector number has had sufficient time to propagate, a second SN74LS373 octal latch is used to capture the number and allow the latches in the 24 interrupt group to be released to accept new interrupt requests. The delay, imposed by a SN74LS95 four-bit shift register to latch the vector number, assumes that all 24 groups are implemented and the MC68000 is running at eight megahertz. Timing requirements can be derived from Figure 4.

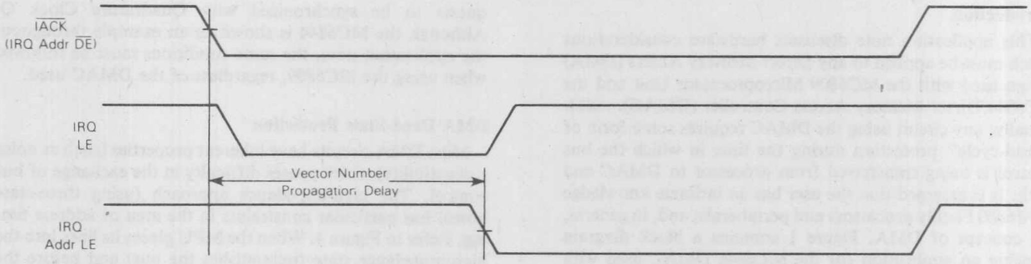
A final SN74LS148 is used to encode the three-bit interrupt requests to the MC68000. The inputs to this encoder are the GS outputs of the SN74LS148 encoders from any group of interrupts. The group providing the GS output and all preceding groups can activate the level to which the GS output is input. However, GS outputs of preceding groups which are input to a higher level in the final SN74LS148 will effectively disable the lower level interrupt request.

If R9M or T6E mask types are used in the system, then the seven levels of interrupts must be latched before encoding on the rising edge of the processor clock. The latch is necessary to synchronize the interrupt request lines.

VARIATIONS

The following variations to the system given in the application can be considered in light of specific system requirements:

1. Flip-flops could be inserted on each group to latch an edge-type interrupt input.
2. The level of interrupt that initiated exception processing could be decoded from address lines A1, A2, and A3 of the MC68000.
3. If vector numbers reserved for other functions, e.g., TRAPS, auto vectors, are unused; then they could be used for user interrupts. However, observe caution when using any reserved vector numbers.



$\overline{\text{DTACK}}$ delay = Vector number propagation delay − 235 ns
(if $\leq \phi$ then no wait states are necessary)

Figure 4. Vector Number Generation Circuit — Timing Diagrams

AN-820

HARDWARE CONSIDERATIONS FOR DIRECT MEMORY ACCESS USING THE MC6809 MICROPROCESSOR UNIT AND MC6844 DMA CONTROLLER

Prepared by:
David Smith
MOS Microprocessor
Systems and Applications

Introduction

This application note discusses hardware considerations which must be applied to any Direct Memory Access (DMA) design used with the MC6809 Microprocessor Unit and the MC6844 Direct Memory Access Controller (DMAC). Additionally, any circuit using the DMAC requires some form of "dead-cycle" protection during the time in which the bus control is being transferred from processor to DMAC and back. It is assumed that the user has an intimate knowledge of M6800 Family processors and peripherals; and, in general, the concept of DMA. Figure 1 contains a block diagram showing an application for the MC6844 DMAC used with the MC6809 MPU. For additional DMA or microprocessor information, refer to the respective data sheets and the MC6809 Users Manual.

MC6809 Requirements

DMA design, using the MC6809, is made easier by the inclusion of the DMA/Bus request ($\overline{\text{DMA/BREQ}}$) feature on-chip. When the $\overline{\text{DMA/BREQ}}$ line is asserted, the MC6809 relinquishes control of the bus by: (1) setting its address, data, and control lines to the high-impedance state; and (2) transferring control to the DMAC by asserting a Bus Grant signal (BA = 1 and BS = 1). This DMA/Bus request feature has timing constraints which must be considered by the designer in order to stay within published specifications.

As shown in Figure 2, the $\overline{\text{DMA/BREQ}}$ signal must occur at least t_{PCSD} before the falling edge of E. This guarantees that the transfer begins on the next falling edge of the E clock. In addition, $\overline{\text{DMA/BREQ}}$ must not be allowed to change during the last 120 ns of this E cycle. If this timing is violated, the MC6809 could enter an undefined state. A simple sync circuit can be made, using a D-type flip-flop as shown in Figure 3. This circuit causes all bus transfer requests to be synchronized with Quadrature Clock Q. Although the MC6844 is shown as an example throughout the application note, the same conditions must be followed when using the MC6809, regardless of the DMAC used.

DMA Dead-State Protection

Most DMA circuits have inherent properties (such as noise susceptibility) which causes difficulty in the exchange of bus control. The high-impedance approach (using three-state buses) has particular constraints in the area of address timing. Refer to Figure 4. When the MPU places its lines into the high-impedance state (relinquishes the bus) and before the DMAC assumes control of the bus, there is a period of time in which the bus lines are not driven. During this "exchange of bus control" time, the bus lines are extremely susceptible to noise. This condition could cause unwanted reads and writes during the bus control transfer. To alleviate this condition, a means of protecting memories and peripherals during transfer must be provided.

One method to provide this protection is to generate a signal called $\overline{\text{DMAVMA}}$, as shown in Figure 5. This signal line (Direct Memory Access Valid Memory Address) is then used as one of the inputs for the memory decoding scheme. Thus, when the $\overline{\text{DMAVMA}}$ line goes high, the memory cannot be enabled. The $\overline{\text{DMAVMA}}$ signal is the result of Bus Grant or DMA Grant (DGRNT, BA = 1 BS = 1) being applied to an Exclusive OR; one direct input and one delayed input. As long as DGRNT does not change, $\overline{\text{DMAVMA}}$ is low and the system decoder is functional. When the MPU relinquishes the bus (places its lines in the high-impedance state) BA and BS both go high, causing DGRNT to change to a high. The DGRNT high causes the $\overline{\text{DMAVMA}}$ line to go high until the next E-clock negative edge clocks the 74LS74 (both Exclusive OR inputs become equal): causing the

$\overline{\text{DMAVMA}}$ line to again go low. Thus, during the time shown as "DEAD" in Figure 4, the system cannot be enabled and the memories and peripherals are protected during the exchange of bus control. The above description pertained to a change from MPU to DMA. However, in Figure 4 the $\overline{\text{DMAVMA}}$ goes high for a corresponding time when the change is from DMA to MPU with one exception: the DGRNT signal change is from high-to-low; but, the effect on the Exclusive OR is the same.

One consideration is that the actual time when $\overline{\text{DMAVMA}}$ is high is not sufficient to provide full protection during the entire bus transition time. This is because the logical function Bus Grant (BA•BS; DGRNT in Figure 4) occurs simultaneously with the high-impedance transition of the address and control bus (see ① of Figure 4). However, a period of 20 to 30 ns could elapse before $\overline{\text{DMAVMA}}$ goes high, resulting from propogation delay through the logic elements. During this 20-30 ns time, the memory is still functional but addresses are invalid; therefore, spurious read/write complications may be possible with some memories. One solution to this problem would be to enable the memory decoding with the E clock. One benefit of this is that most decoding must be synchronized with the falling edge of E anyway, since all data transfers occur then. One negative aspect to this is that the maximum memory access time is shortened to 450 ns (E high time).

Another solution (more oriented toward lengthening access time) would be to protect the memory, starting at the falling edge of E and continuing only throughout the 20-30 ns propogation delay period that the bus is unprotected. Since this unprotected state cannot be detected before the Bus Grant is sent, this period should be protected during every cycle. Then if a request occurs, and that cycle is a high-impedance (dead) state, the $\overline{\text{DMAVMA}}$ signal will cause protection to extend to the end of that cycle. A convenient period of time to use for that additional "pre-transition" protection is the first full quarter-cycle of E. The length of that signal (Figure 4) is long enough to cover the address change time (200 ns maximum) and still be short compared to the remaining 750 ns of cycle time. (Since the addresses are not guaranteed until 200 ns after falling E, and a 30 ns delay in bus transition is inherent, only 20 ns typically is lost in memory access time with this signal.) This first-quarter cycle signal, called FQ (Figures 4 and 6), enters the de-enable path through the OR gate along with $\overline{\text{DMAVMA}}$. It will keep $\overline{\text{DMAVMA}}'$ high during the first quarter of all cycles.

If the designer already uses fast memories, the decoding can be enabled with the E clock. This method does cut access time to 450 ns for all parts, but if this is a more cost effective solution, it will protect the bus during the entire first half of each cycle. This should cover the exchange of control during all DMA dead-states (this method could be used with some peripherals due to address setup times).

These interface procedures should be used with all DMA systems, and in particular with the MC6844 and MC6809. DMA ***does*** require exact timing and bus protection, and these methods will fulfill those requirements.

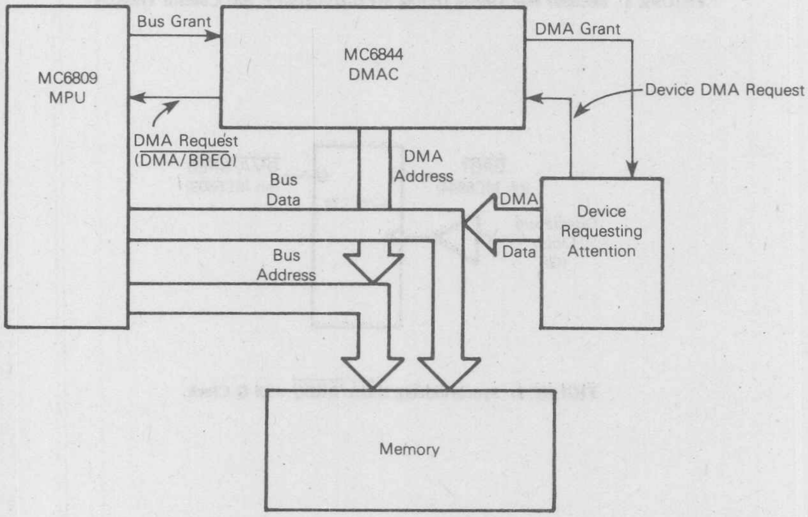


FIGURE 1. Block Diagram Showing DMAC Used in System

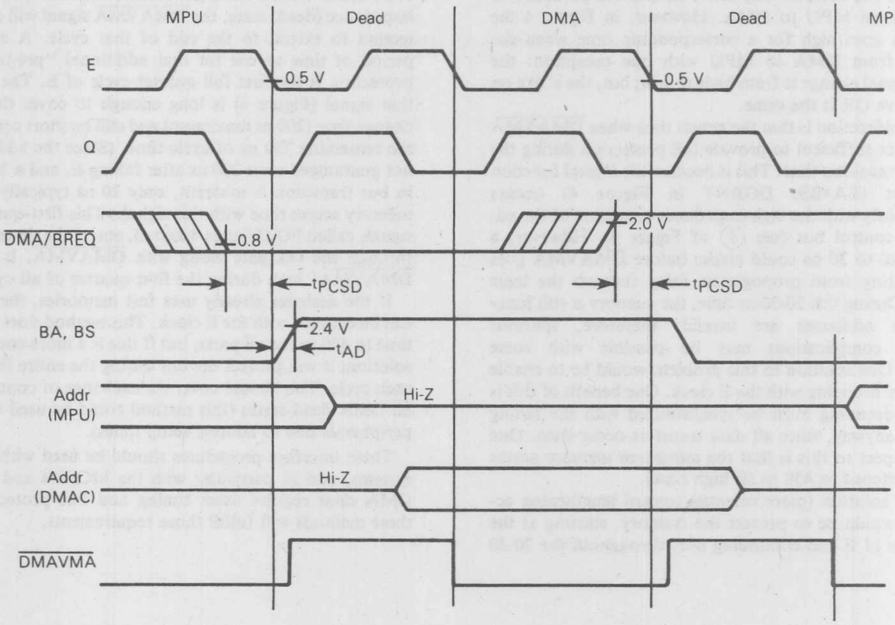


FIGURE 2. MC6809 Bus Timing During MPU-DMA-MPU Bus Control Transfer

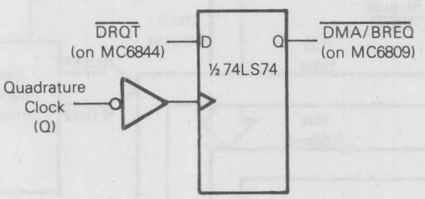


FIGURE 3. Synchronizing DMA/BREQ with Q Clock

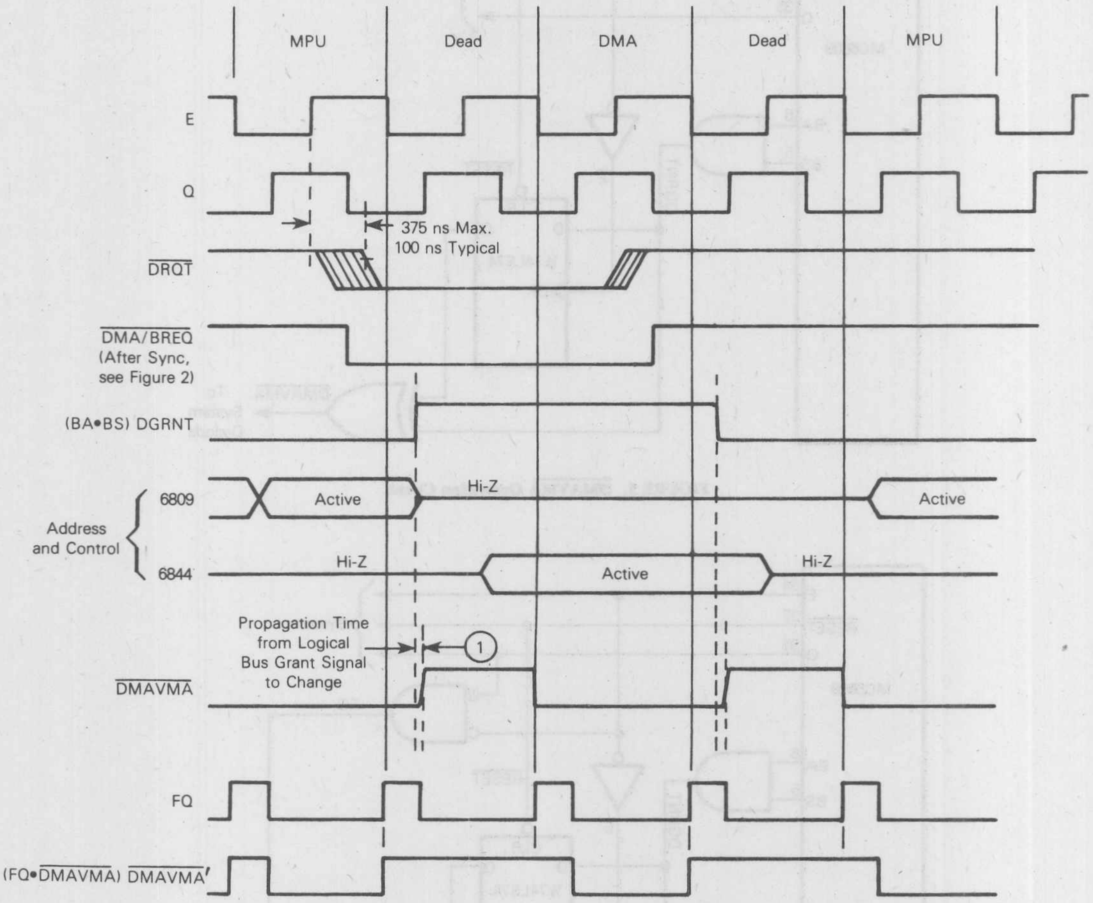


FIGURE 4. System Timing

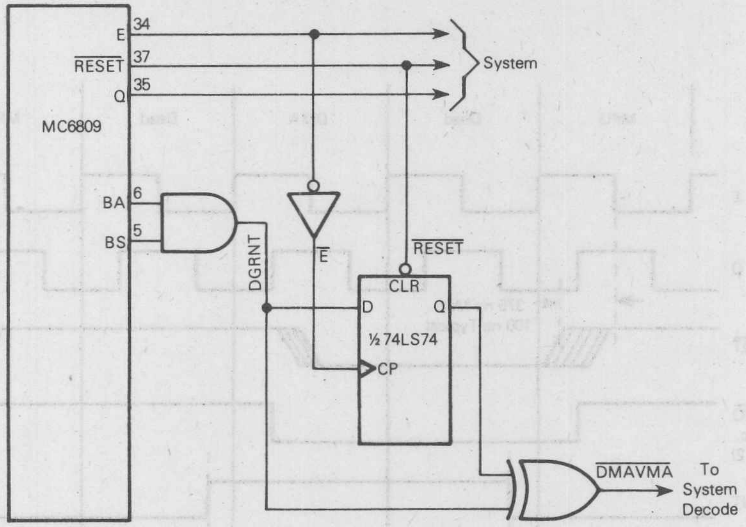


FIGURE 5. $\overline{\text{DMAVMA}}$ Generation Circuit

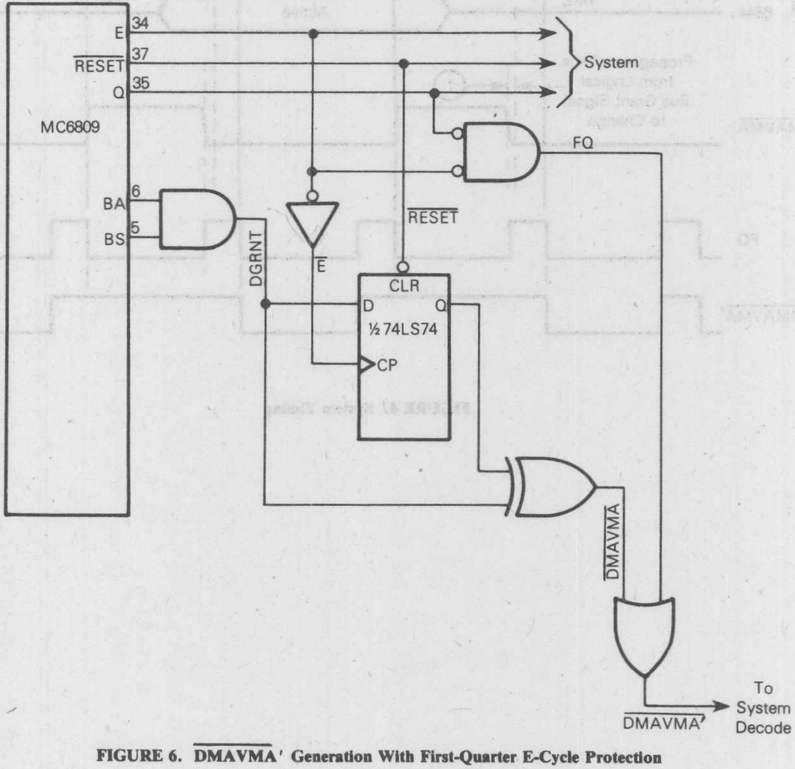

E
34
$\overline{\text{RESET}}$
37
Q
35
System
MC6809
FQ
BA
6
BS
5
DGRNT
$\overline{\text{E}}$
$\overline{\text{RESET}}$
CLR
D
Q
½ 74LS74
CP
$\overline{\text{DMAVMA}}$
$\overline{\text{DMAVMA}}'$
To
System
Decode

FIGURE 6. $\overline{\text{DMAVMA}}'$ Generation With First-Quarter E-Cycle Protection

AN-823

CBUG05 DEBUG MONITOR PROGRAM FOR MC146805E2 MICROPROCESSOR UNIT

Prepared by
Rex Davis
Microprocessor Applications Engineer

1. INTRODUCTION

CBUG05 is a debug monitor program written for the MC146805E2 Microprocessor Unit and contained in the MCM65516 2K×8 CMOS ROM. CBUG05 allows for rapid development and evaluation of hardware and M6805 Family type software, using memory and register examine/change commands as well as breakpoint and single instruction trace commands. CBUG05 also includes software to set and display time, using an optional MC146818 Real-Time Clock (RTC), and routines to punch and load an optional cassette interface. Figure 1 shows a minimum system which only requires the MPU, ROM, keypad inputs and display output interfaces. Port A of the MC146805E2 MPU is required for the I/O; however, Port B and all other MC146805E2 MPU features remain available to the user. A possible expanded system is shown in Figure 2. The memory map is shown in Figure 3. Locations $1700-$173F are available to the user if the optional MC146818 RTC is not used.

FEATURES:

* MC146805E2 Eight-Bit CMOS MPU
- Expandable Multipled Address/Data Bus
- Eight-Bit I/O Port
- Eight-Bit Timer with Prescaler
- Maskable External Interrupt
- 16 Levels of Subroutine Nesting
- Minimum of 38 Bytes of Unused Internal RAM

* MCM65516 2K×8 CMOS with CBUG05
- Memory and Register Examine/Change
- Breakpoints and Single Instruction Trace
- Branch Offset Calculation
- Set/Display Current Time (w/optional MC146818 Real-Time Clock)
- Punch/Load/Verify Cassette Tape (w/optional cassette tape interface)
- Stop Command for Low-power Software Standby
- Software Alterable Interrupt Vectors

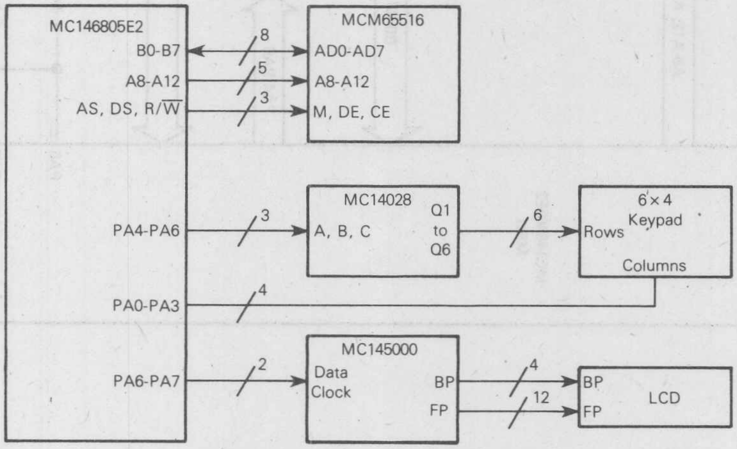


Figure 1. Minimum CBUG05 System

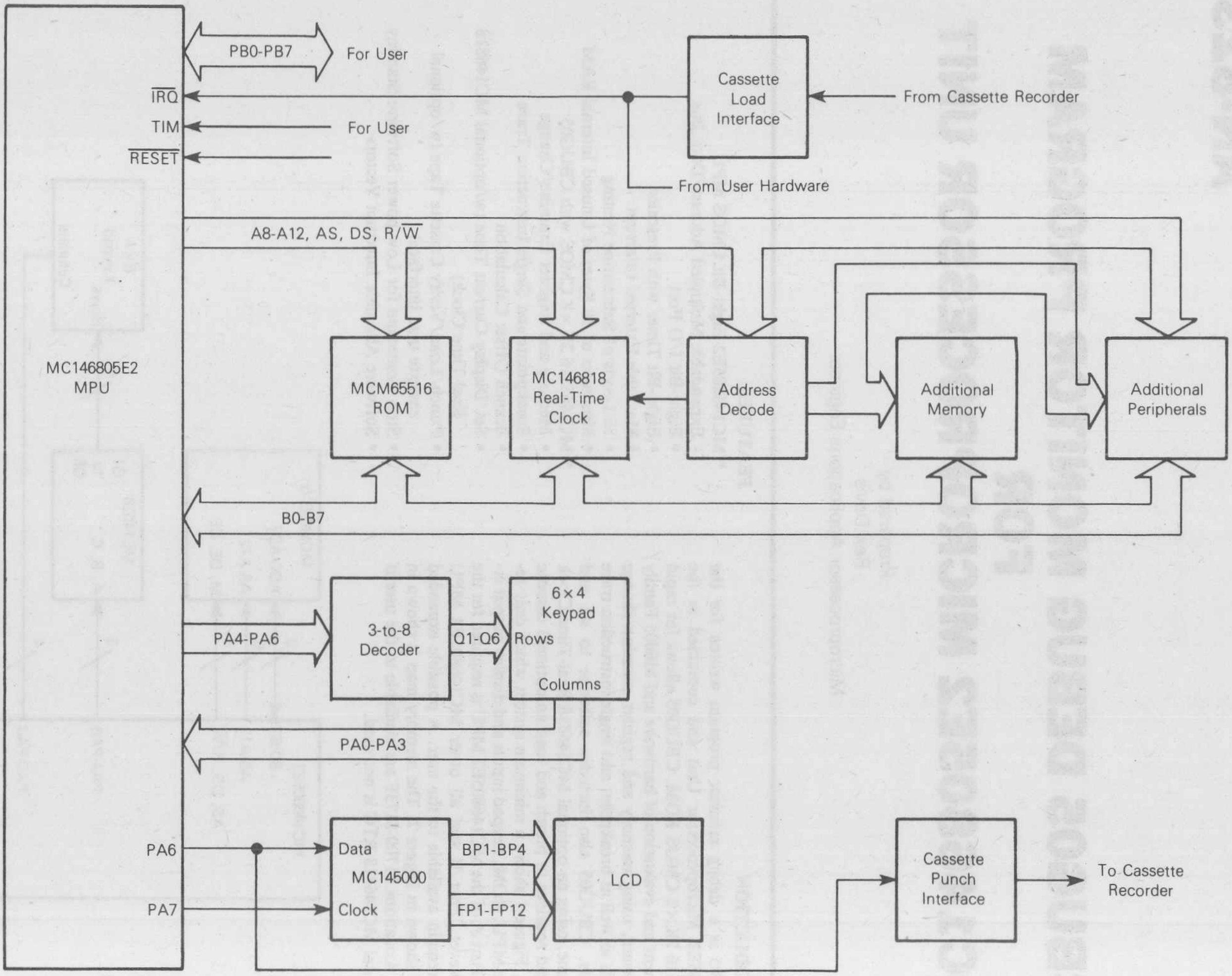


Figure 2. Expanded CBUG05 System

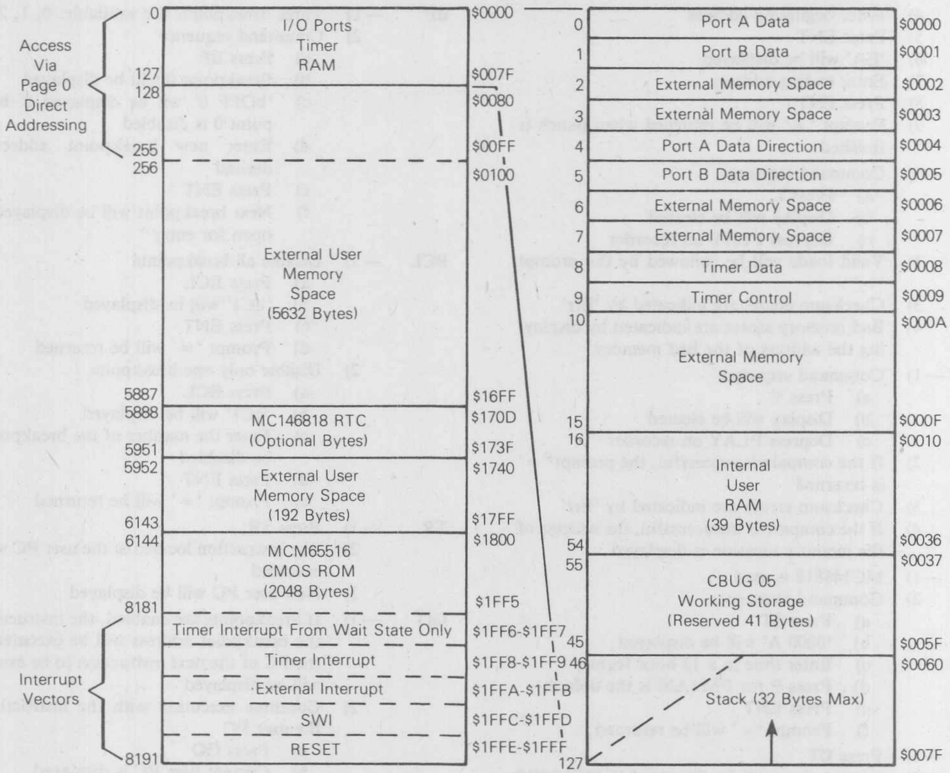


Figure 3. Address Map

2. CBUG05 COMMAND DESCRIPTION

Commands are entered in one of two ways:

(1) If the command requires no additional user input, then only the command key need be depressed; e.g., TR (CBUG05 will execute one instruction), and (2) If the command allows additional user input then the ENT key is used to enter the users input.

ESC will allow exit from all commands except STOP, V, L, & P once the ending address is entered.

1) RS — Reset MC146805E2
2) P — Punch cassette tape
3) L — Load cassette tape
4) V — Verify cassette tape against memory
5) ST — Set current time
6) DT — Display current time
7) OFF — Calculate branch offset
8) BP — Set/display breakpoints
9) BCL — Disable one or all breakpoints
10) TR — Execute one instruction
11) GO — Begin program execution
12) PC — Display user program counter
13) AR — Examine/change user accumulator
14) XR — Examine/change user index register
15) CC — Examine/change user condition code register
16) SP — Display user stack pointer
17) M — Examine/change memory contents
18) STOP — Put the system into a low power standby mode

RS — 1) Automatic on power-up
2) Press RS to:
a) Return from STOP
b) Return to monitor when program control is lost

STOP 1) MC146805E2 oscillator is halted reducing current requirements
2) Command sequence:
a) Press STOP
b) Display will be cleared

P — 1) Place recorder into the record mode
2) Press P
3) 'bA' will be displayed

4) Enter beginning address
5) Press ENT
6) 'EA' will be displayed
7) Enter ending address
8) Press ENT
9) Prompt '=' will be returned when punch is finished

L — 1) Command sequence
a) Press L
b) Display will be cleared
c) Depress PLAY on recorder
2) Valid loads will be followed by the prompt '='
3) Checksum errors are indicated by 'Err'
4) Bad memory stores are indicated by displaying the address of the bad memory.

V — 1) Command sequence
a) Press V
b) Display will be cleared
c) Depress PLAY on recorder
2) If the compare is successful, the prompt '=' is returned
3) Checksum errors are indicated by 'Err'
4) If the compare is unsuccessful, the address of the memory location is displayed

ST — 1) MC146818 is used
2) Command sequence
a) Press ST
b) '0000 A' will be displayed
c) Enter time in a 12 hour format
d) Press P for PM (AM is the default)
e) Press ENT
f) Prompt '=' will be returned

DT — 1) Press DT
2) current time will be displayed if MC146818 has been initialized

OFF — 1) Beginning and ending addresses point to the instruction opcode addresses
2) The opcode for the branch instruction must exist at the beginning address so the monitor can determine whether to do a bit branch or a conditional branch
3) Command sequence
a) Press OFF
b) 'bA' will be displayed
c) Enter beginning address
d) Press ENT
e) 'EA' will be displayed
f) Enter ending address
g) Press ENT
4) If valid:
a) 'USE xx' will be displayed.
b) xx will be loaded into beginning address + 2 for bit branches and address + 1 for conditional branches.
5) If not valid:
a) Offset calculation result is displayed in 2's complement and 'Or' (out of range) is displayed
b) No change is made to instruction at the beginning address.

BP — 1) Three breakpoints are available: 0, 1, 2
2) Command sequence
a) Press BP
b) Breakpoint 0 will be displayed
c) 'bOFF 0' wil be displayed if breakpoint 0 is disabled
d) Enter new breakpoint address if desired
e) Press ENT
f) Next breakpoint will be displayed and open for entry

BCL — 1) Disable all breakpoints
a) Press BCL
b) 'bC1' will be displayed
c) Press ENT
d) Prompt '=' will be returned
2) Disable only one breakpoint
a) Press BCL
b) 'bC1' will be displayed
c) Enter the number of the breakpoint to be disabled
d) Press ENT
e) Prompt '=' will be returned

TR — 1) Press TR
2) The instruction located at the user PC will be executed
3) New user PC will be displayed

GO — 1) If breakpoints are enabled, the instruction at the breakpoint address will be executed and the PC of the next instruction to be executed will be displayed
2) Continue execution with the instruction at the user PC
a) Press GO
b) Current user PC is displayed
c) Press ENT
3) Begin execution at new address
a) Press GO
b) Current user PC is displayed
c) Enter the new PC address
d) Press ENT

M — 1) Press M
2) Last address will be displayed
3) Enter new address if desired
4) Press ENT
5) Address and contents of the address will be displayed in format 'aaaa xx'
6) Enter new contents if desired
7) Save (use one)
a) Press ENT (next address and contents will be displayed)
b) Press M (previous address and contents will be displayed)

PC — 1) Not alterable
2) Command sequence
a) Press PC
b) Current user PC displayed in format 'aaaa PC'

AR — 1) Alterable
2) Command sequence
a) Press AR
b) Current user accumulator contents displayed in format 'ACCA xx'
c) Enter new data if desired
d) Press ENT
e) Prompt '=' will be returned

XR — 1) Alterable
2) Command sequence
a) press XR
b) Current user index register contents displayed in format 'Idr xx'
c) Enter new data if desired
d) Press ENT
e) Prompt '=' will be returned

CC — 1) Alterable
2) Command sequence
a) Press CC
b) Current user condition code will be displayed in format 'COdE xx'
c) Enter new contents if desired
d) Press ENT
e) Prompt '=' will be returned

SP — 1) Not alterable
2) Command sequence
a) Press SP
b) Current user stack pointer will be displayed in format 'aaaa SP'

3. INTERRUPT VECTORS

At reset, CBUG05 sets up an extended JUMP instruction pointing to a default CBUG05 interrupt service routine for each of the three interrupt types. The vectors, of the three interrupt types, point to one of the three JUMP instructions. Since the JMP instructions are located in RAM, the use may alter the two-byte extended address within any of the JMP instructions. The location of the two-byte extended address for each interrupt type is listed in Table 1.

Table 1. Alterable Vector Locations

INTERRUPT TYPE		ADDR
EXTERNAL	:	$41-$42
TIMER	:	$44-$45
TIMER (FROM WAIT)	:	$47-$48

4. MC145000 CMOS MULTIPLEXED LCD DRIVER

The MC145000 LCD Driver is designed to drive LCDs in a multiplexed-by-four configuration. It can drive up to 48 LCD segments or six seven-segment plus decimal point characters. Data for each character is translated into a format that is clocked serially from the MC146805E2 (MPU) to the MC145000 LCD Driver. The MC145000 LCD Driver continuously generates the multiplexed display signals, from the internally stored serial data, without further requirements from the MPU.

The recommended display is a General Electric LXD69D7R09; an 8-digit, 7-segment multiplexed LCD with decimal point. The required connections to the MC145000 LCD Driver are shown in Figure 4.

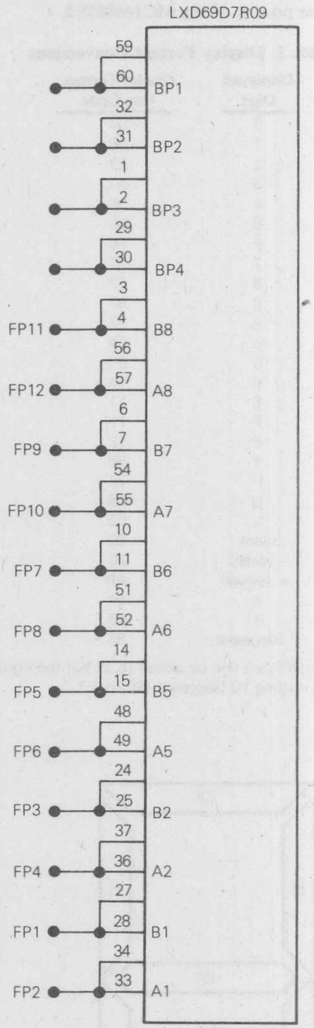


Figure 4. Liquid Crystal Display (LCD) Connections

Each segment of a seven-segment plus decimal point character is represented by one bit of an 8-bit byte. Figure 5 shows the relationship between a character segment and the bit number of the display byte (bit 7 is MSB and bit 0 is LSB). A logical "1" in any bit will activate its corresponding segment. Table 2 lists the hexadecimal code of some common seven-segment characters in display format. For example, the digit 5 is represented by $B5 (10110101) which would activate segments 0, 2, 4, 5, and 7. The decimal point is displayed by setting bit 3 of the display byte to a logical "1" (effectively adding eight to the display byte). Data in BCD or binary format is translated by CBUG05, into the display format, using a lookup table. CBUG05 then left-shifts the character to the MC145000 via port A of the MC146805E2.

Table 2. Display Format Conversions

Displayed Digit	Display Format Hex Code
0	D7
1	06
2	E3
3	A7
4	36
5	B5
6	F5
7	07
8	F7
9	B7
A	77
b	F4
C	D1
d	E6
E	F1
F	71
P	73
Y	B6
H	76
U	D6
L	D0
blank	00
- (dash)	20
= (equal)	A0
n	64
r	60
° (degrees)	33

NOTE: A Decimal Point can be added to all but the right-most display digit by setting b3 [segment (3)] to a 1.

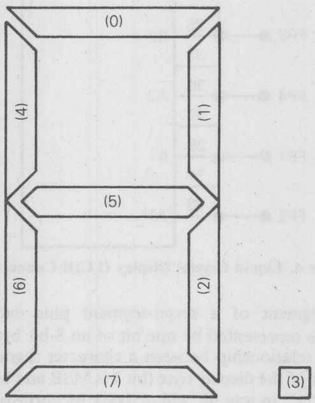


Figure 5. Display Digit Format

Several display routines are available for the user. Figure 6 describes the address, function, and use of these routines. All routines are called using a jump-to-subroutine (JSR) instruction. Most display outputs are initiated by filling a display table with all six characters in the display format to be displayed, then calling a routine (DISTAB) to display the entire table. In other words, the whole display is rewritten every time any character change is made. The display table is called DTABL (locations $49-$4E) and occupies six consecutive bytes where DTABL (location $49) is the left most digit to be displayed.

5. KEYPAD INPUT

CBUG05 requires a 4×6 keypad such as is shown schematically in Figure 7. The six column lines are derived from a three-bit output from port A bits 4-6 driving a 3-to-8 decoder. By using this method port B is saved for the user. Figure 7 shows the required layout of the 4×6 keypad and 3-to-8 decoder. The keypad is continuously scanned for input. If an input is received, a 3075 MPU cycle debounce insures against spurious input. The required debounce time places a lower limit on the MPU clock frequency. At a 1 MHz bus speed (5 MHz oscillator input), the debounce time is about 3 ms. With a 10 kHz bus speed (50 kHz time base input), the debounce time is about 0.3 seconds. Debounce times of approximately 60 milliseconds or more require the keys be held down a longer time than an operator is normally accustomed.

Five routines are listed in Figure 8 of which two (COLUMN and DEBOUNC) are branch routines and one is a look-up table (STABL). One of the other two routines, KEYSCN, checks for a keyboard input and, if valid, returns it to the accumulator in a column-row format. This format can then be converted to a hexadecimal number which corresponds to the one key that was pressed (see STABL routine and Table 3). Note that hexadecimal numbers 0 through F correspond to the keypad keys 0 through F. The last routine of Figure 8, CHARIN, checks for a character and returns a hexadecimal number to the accumulator.

6. CASSETTE TAPE OPTION

The cassette tape option is included to allow for user program storage. Programs are stored in a modified bi-phase format (see Figure 9). The storage format used defines a zero as more than 300 MPU cycles between transitions and less than 300 MPU cycles between transitions. Data is punched with a start bit of one, eight bits of data and a zero stop bit. Tapes are punched with 16K zeros as a leader followed by a BOT and the ending and beginning addresses. The program is then punched followed by the checksum. Tapes are loaded after 256 consecutive zeros are read. The BOT then synchronizes the loading program. The ending and beginning addresses are loaded and data read and stored accordingly. Finally, the checksum is read and compared to the new computed checksum.

Baud rates are determined by the MPU cycle time. The software is set up to provide a default baud rate of 2400 baud if a one microsecond cycle time is used. Cycle times greater than one microsecond will decrease the baud rate proportionally.

```
                   *
                   ****************************************
                   *                                      *
                   *      CLEAR DISPLAY TABLE             *
                   *                                      *
                   *      X REG DESTROYED                 *
                   *                                      *
                   ****************************************
                   *
1DF5 AE Ø5       A CLRTAB LDX    #5
1DF7 6F 49       A CLRLOC CLR    DTABL,X  CLEAR SIX
1DF9 5A                   DECX            LOCATIONS IN
1DFA 2A FB    1DF7        BPL    CLRLOC   DISPLAY TABLE
1DFC 81                   RTS
                   *
                   ****************************************
                   *                                      *
                   *      DISPLAY TABLE CONTENTS          *
                   *                                      *
                   *      A,X REGISTERS DESTROYED         *
                   *                                      *
                   ****************************************
                   *
1DFD AE Ø5       A DISTAB LDX    #5
1DFF E6 49       A DISCHR LDA    DTABL,X  LOAD DISPLAY
1EØ1 AD Ø9    1EØC        BSR    DISPLY   TABLE INTO
1EØ3 5A                   DECX            145ØØØ
1EØ4 2A F9    1DFF        BPL    DISCHR
1EØ6 81                   RTS
                   *
                   ****************************************
                   *                                      *
                   *      BLANK DISPLAY                   *
                   *                                      *
                   *      A,X REGISTERS DESTROYED         *
                   *                                      *
                   ****************************************
                   *
1EØ7 AD EC    1DF5 CLRDIS BSR    CLRTAB   BLANK
1EØ9 AD F2    1DFD        BSR    DISTAB   DISPLAY
1EØB 81                   RTS
                   *
                   ****************************************
                   *                                      *
                   *      SHIFT ONE CHARACTER INTO        *
                   *             DISPLAY                  *
                   *                                      *
                   *      A REGISTER DESTROYED            *
                   *                                      *
                   ****************************************
                   *
1EØC BF 5Ø       A DISPLY STX    WORK1    SAVE INDEX
1EØE 1D ØØ       A        BCLR   6,PORTA  CLEAR DATA
1E1Ø AE Ø8       A        LDX    #8
1E12 48            DIS1   LSLA            SET UP
1E13 24 Ø2    1E17        BCC    DIS2     BIT OF
1E15 1C ØØ       A        BSET   6,PORTA  ACCUMULATOR
1E17 1E ØØ       A DIS2   BSET   7,PORTA  CLOCK
1E19 1F ØØ       A        BCLR   7,PORTA  IT
1E1B 1D ØØ       A        BCLR   6,PORTA  CLEAR DATA
1E1D 5A                   DECX            COMPLETE?
1E1E 26 F2    1E12        BNE    DIS1     NO
1E2Ø BE 5Ø       A        LDX    WORK1    RESTORE INDEX
1E22 81                   RTS
```

Figure 6. Display Routines

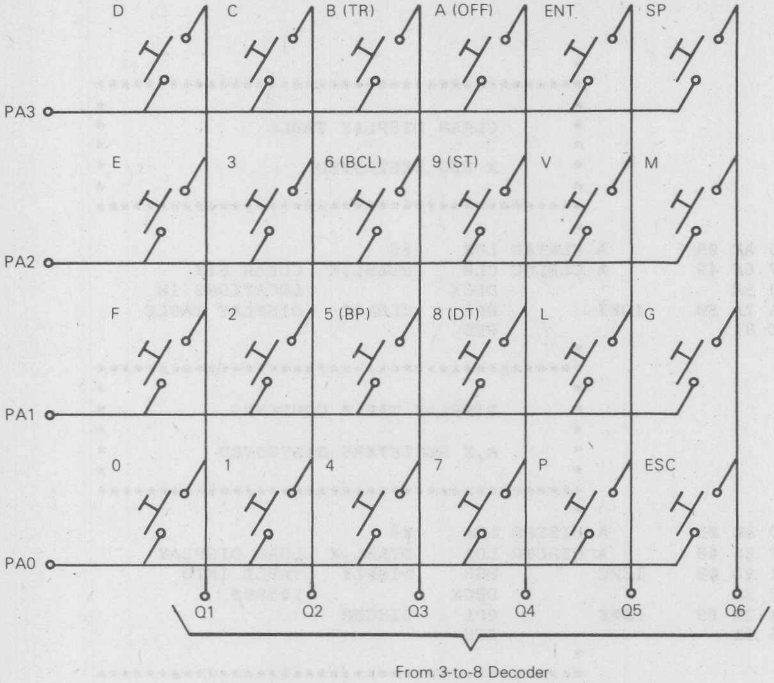


Figure 7. 4×6 Keypad Schematic Diagram

Table 3. Keypad Cross-Reference

KEYPAD CHARACTER	PORT-A DATA	HEXADECIMAL ($) EQUIVALENT
0 (P.C.)	11	0
F (AR)	12	F
E (XR)	14	E
D (CC)	18	D
1 – –	21	1
2 – –	22	2
3 – –	24	3
C – –	28	C
4	31	4
5 (BP)	32	5
6 (B.CL.)	34	6
B (TR)	38	B
7	41	7
8 (DT.)	42	8
9 (ST.)	44	9
A (OFF)	48	A
P	51	17
L	52	16
V	54	15
ENT	58	11
ESC	61	10
G	62	14
M	64	13
SP	68	12

Whatever baud rate is used, the cassette tape and recorder must have an upper frequency response 2-3 times the baud rate and a lower frequency response of 1/2 - 1/3 the baud rate to insure reliability.

7. MC146818 REAL-TIME CLOCK (RTC) OPTION

The RTC can be added to a system to provide time, data, periodic interrupt and many other user functions (see MC146818 ADI-856). The RTC time may be set and displayed using CBUG05 software; however, only the 12-hour mode is available. The displayed time is updated once per second after polling the Update-In-Progress bit (UIP) for a zero. All MC146818 functions are available to the user. The CBUG05 software set and display time routines require that a 4.194304 MHz crystal be used; however, if power consumption is critical then either a 1.04576 MHz or 32.678 KHz oscillator input could be used. The user would be required to set-up the divider chain in the RTC for the particular time base used.

8. INTERNAL AND EXTERNAL MEMORY SPACE

The internal memory space is located in the first 128 bytes of memory and contains the timer registers, I/O port registers, and 112 bytes of RAM. External memory can be mapped at the same addresses as the internal memory space. An MPU write to internal memory space is duplicated externally; however, an MPU read of internal locations will result in only the internal data being recognized. This allows the user to map large memories externally without requiring that accesses to internal memory locations be excluded from the external memory, thus, simplifying external address decoding.

```
                    *****************************************
                    *                                       *
                    *      KEYPAD SCAN                      *
                    *                                       *
                    *      X REGISTER DESTROYED             *
                    *                                       *
                    *      A REGISTER CONTAINS VALUE        *
                    *                                       *
                    *      CARRY SET IF VALID OUTPUT        *
                    *                                       *
                    *****************************************
1E23 98             KEYSCN CLC
1E24 4F                    CLRA
1E25 AE Ø6        A        LDX    #6        SETUP
1E27 AB 1Ø        A KEY1   ADD    #$1Ø      ROW
1E29 B7 ØØ        A        STA    PORTA
1E2B AD Ø6     1E33        BSR    COLUMN    CHECK COLUMNS
1E2D 25 Ø3     1E32        BCS    KEY2      IF VALID GET OUT
1E2F 5A                    DECX             ELSE TRY
1E3Ø 26 F5     1E27        BNE    KEY1      NEXT ROW
1E32 81             KEY2   RTS
                    *
                    *****************************************
                    *                                       *
                    *      CHECK FOR KEY CLOSURE            *
                    *      WITHIN COLUMN AND DEBOUNCE       *
                    *                                       *
                    *      A REGISTER CONTAINS VALUE        *
                    *                                       *
                    *      CARRY SET IF VALID OUTPUT        *
                    *                                       *
                    *****************************************
                    *
1E33 B6 ØØ        A COLUMN LDA    PORTA     READ KEYPAD
1E35 B7 5Ø        A        STA    WORK1     STORE IT
1E37 A5 ØF        A        BIT    #$ØF      KEY CLOSED?
1E39 27 19     1E54        BEQ    COLRET    NO GET OUT
1E3B AD 18     1E55        BSR    DBOUNC    ELSE DEBOUNCE
1E3D B6 ØØ        A        LDA    PORTA     RE-READ KEYPAD
1E3F B1 5Ø        A        CMP    WORK1     SAME KEY CLOSED?
1E41 26 11     1E54        BNE    COLRET    NO GET OUT
1E43 99                    SEC              SET FLAG FOR VALID
1E44 B6 ØØ        A COL1   LDA    PORTA     KEY
1E46 A5 ØF        A        BIT    #$ØF      RELEASED?
1E48 26 FA     1E44        BNE    COL1      NO TRY AGAIN
1E4A AD Ø9     1E55        BSR    DBOUNC    YES DEBOUNCE
1E4C B6 ØØ        A        LDA    PORTA     STILL
1E4E A5 ØF        A        BIT    #$ØF      RELEASED?
1E5Ø 26 F2     1E44        BNE    COL1      NO TRY AGAIN
1E52 B6 5Ø        A        LDA    WORK1     RETURN CHAR IN A-REG
```

Figure 8. KEYSCN, COLUMN, DEBOUNC, CHRIN, and STABL Routines

```
1E54 81            COLRET RTS              YES GO HOME
                   *
                   ******************************************
                   *                                        *
                   *      PAUSE FOR 3Ø75 CYCLES             *
                   *                                        *
                   *      A REGISTER DESTROYED              *
                   *                                        *
                   ******************************************
                   *
1E55 A6 FF       A DBOUNC LDA    #$FF      PAUSE
1E57 21 FE    1E57 DLOOP  BRN    *         256X12
1E59 21 FE    1E59        BRN    *         CYCLES
1E5B 4A                   DECA             OR AT
1E5C 26 F9    1E57        BNE    DLOOP     LEAST
1E5E 81                   RTS              3.Ø MS
                   *
                   *
                   ******************************************
                   *                                        *
                   *      INPUT ONE CHARACTER               *
                   *                                        *
                   *      A REGISTER CONTAINS HEX VALUE     *
                   *                                        *
                   *      X REGISTER CONTAINS HEX VALUE     *
                   *                                        *
                   ******************************************
                   *
        1E5F     A CHRIN  EQU    *
1E5F CD 1E23     A        JSR    KEYSCN    GET KEY
1E62 24 FB    1E5F        BCC    CHRIN     IF NOT VALID RETRY
1E64 5F                   CLRX
1E65 D1 1E6F     A CHRIN1 CMP    STABL,X   CONVERT
1E68 27 Ø3    1E6D        BEQ    CHRIN2    TO HEX
1E6A 5C                   INCX
1E6B 2Ø F8    1E65        BRA    CHRIN1
1E6D 9F            CHRIN2 TXA              IF CANCEL
1E6E 81                   RTS
```

Figure 8. KEYSCN, COLUMN, DEBOUNC, CHRIN, and STABL Routines (Cont'd)

```
                   *
                   ****************************************
                   *                                      *
                   *      CONVERSION TABLE FOR KEYPAD     *
                   *      TO HEX NUMBER                   *
                   *                                      *
                   ****************************************
                   *
1E6F    11       A STABL  FCB    $11      Ø
1E7Ø    21       A        FCB    $21      1
1E71    22       A        FCB    $22      2
1E72    24       A        FCB    $24      3
1E73    31       A        FCB    $31      4
1E74    32       A        FCB    $32      5
1E75    34       A        FCB    $34      6
1E76    41       A        FCB    $41      7
1E77    42       A        FCB    $42      8
1E78    44       A        FCB    $44      9
1E79    48       A        FCB    $48      A
1E7A    38       A        FCB    $38      B
1E7B    28       A        FCB    $28      C
1E7C    18       A        FCB    $18      D
1E7D    14       A        FCB    $14      E
1E7E    12       A        FCB    $12      F
1E7F    61       A        FCB    $61      CANCEL COMMAND
1E8Ø    58       A        FCB    $58      ENTER COMMAND
1E81    68       A        FCB    $68      STACK POINTER
1E82    64       A        FCB    $64      MEMORY
1E83    62       A        FCB    $62      GO
1E84    54       A        FCB    $54      VERIFY TAPE
1E85    52       A        FCB    $52      LOAD TAPE
1E86    51       A        FCB    $51      PUNCH TAPE
                   *
                   ****************************************
                   *                                      *
                   *      HEX TO MUX DISPLAY              *
                   *      CONVERSION TABLE                *
```

Figure 8. KEYSCN, COLUMN, DEBOUNC, CHRIN, and STBL Routines (Cont'd)

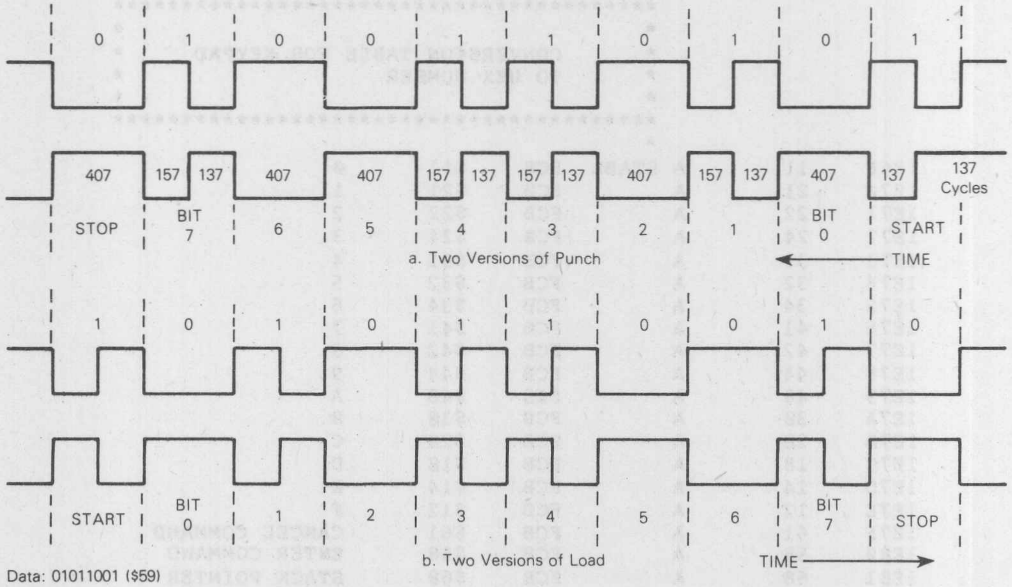


Figure 9. Example of Serial Data Formats for Punch and Load

```
PAGE  ØØ1  CBUGØ5 .SA:1

ØØØØ1                           OPT    CMOS
ØØØØ2                      *
ØØØØ3          ØØØØ     A PORTA  EQU    Ø
ØØØØ4          ØØØ4     A PORTAD EQU    4
ØØØØ5          ØØØ1     A PORTB  EQU    1
ØØØØ6          ØØØ8     A TIMER  EQU    8
ØØØØ7          ØØØ9     A TIMEC  EQU    9
ØØØØ8          17ØA     A CR1    EQU    $17ØA
ØØØØ9          17ØB     A CR2    EQU    $17ØB
ØØØ1Ø          17ØØ     A SEC    EQU    $17ØØ
ØØØ11          17Ø2     A MIN    EQU    $17Ø2
ØØØ12          17Ø4     A HOUR   EQU    $17Ø4
ØØØ13          17Ø7     A DAY    EQU    $17Ø7
ØØØ14          17Ø8     A MONTH  EQU    $17Ø8
ØØØ15          17Ø9     A YEAR   EQU    $17Ø9
ØØØ16          18ØØ     A MONSTR EQU    $18ØØ
ØØØ17          ØØ1F     A PCMASK EQU    $1F
ØØØ18          ØØØ3     A NUMBKP EQU    3
ØØØ19          ØØAØ     A PROMPT EQU    $AØ
ØØØ2Ø          ØØCC     A LJMP   EQU    $CC
ØØØ21          ØØ83     A SWIOP  EQU    $83
ØØØ22                     *
ØØØ23A ØØ4Ø                     ORG    $4Ø
ØØØ24                     *
ØØØ25          ØØ37     A BKPTBL EQU    *-3*NUMBKP
ØØØ26A ØØ4Ø    ØØØ3     A IRQ    RMB    3
ØØØ27A ØØ43    ØØØ3     A TIRQ   RMB    3
ØØØ28A ØØ46    ØØØ3     A TIRQW  RMB    3
ØØØ29A ØØ49    ØØØ6     A DTABL  RMB    6
ØØØ3ØA ØØ4F    ØØØ1     A SWIFLG RMB    1
ØØØ31A ØØ5Ø    ØØØ1     A WORK1  RMB    1
ØØØ32A ØØ51    ØØØ1     A WORK2  RMB    1
ØØØ33A ØØ52    ØØØ1     A ADDRH  RMB    1
ØØØ34A ØØ53    ØØØ1     A ADDRL  RMB    1
ØØØ35A ØØ54    ØØØ1     A WORK3  RMB    1
ØØØ36A ØØ55    ØØØ1     A WORK4  RMB    1
ØØØ37A ØØ56    ØØØ1     A WORK5  RMB    1
ØØØ38A ØØ57    ØØØ1     A WORK6  RMB    1
ØØØ39A ØØ58    ØØØ2     A TEMP   RMB    2
ØØØ4ØA ØØ5A    ØØØ1     A PNCNT  RMB    1
ØØØ41A ØØ5B    ØØØ1     A CHKSUM RMB    1
ØØØ42A ØØ5C    ØØØ1     A SREF   RMB    1
ØØØ43A ØØ5D    ØØØ1     A LCNT   RMB    1
ØØØ44A ØØ5E    ØØØ1     A PCNT1  RMB    1
ØØØ45A ØØ5F    ØØØ1     A PCNTØ  RMB    1
ØØØ46                     *
```

```
PAGE  ØØ2  CBUGØ5  .SA:1

ØØØ48                     *
ØØØ49A 18ØØ                      ORG    $18ØØ
ØØØ5Ø                     *
ØØØ51A 18ØØ A6 FØ       A RESET  LDA    #$FØ     SETUP PORT
ØØØ52A 18Ø2 B7 Ø4       A        STA    PORTAD   FOR KEYPAD
ØØØ53A 18Ø4 3F ØØ       A        CLR    PORTA    AND DISPLAY
ØØØ54A 18Ø6 3F 5C       A        CLR    SREF     INITIALIZE
ØØØ55A 18Ø8 A6 ØF       A        LDA    #$F      TAPE SOFTWARE
ØØØ56A 18ØA B7 5D       A        STA    LCNT     FOR 24ØØ BAUD
ØØØ57A 18ØC A6 12       A        LDA    #$12
ØØØ58A 18ØE B7 5E       A        STA    PCNT1
ØØØ59A 181Ø A6 26       A        LDA    #$26
ØØØ6ØA 1812 B7 5F       A        STA    PCNTØ
ØØØ61                     *
ØØØ62A 1814    1FC5     A VECTOR FDB    IRQV     SET-UP
ØØØ63A 1816    1FC7     A        FDB    TIRQV    INTERRUPT
ØØØ64A 1818    1FC4     A        FDB    TIRQWV   VECTORS
ØØØ65A 181A A6 CC       A        LDA    #LJMP    IN RAM
ØØØ66A 181C B7 4Ø       A        STA    IRQ
ØØØ67A 181E B7 43       A        STA    TIRQ
ØØØ68A 182Ø B7 46       A        STA    TIRQW
ØØØ69A 1822 C6 1814     A        LDA    VECTOR
ØØØ7ØA 1825 B7 41       A        STA    IRQ+1
ØØØ71A 1827 C6 1815     A        LDA    VECTOR+1
ØØØ72A 182A B7 42       A        STA    IRQ+2
ØØØ73A 182C C6 1816     A        LDA    VECTOR+2
ØØØ74A 182F B7 44       A        STA    TIRQ+1
ØØØ75A 1831 C6 1817     A        LDA    VECTOR+3
ØØØ76A 1834 B7 45       A        STA    TIRQ+2
ØØØ77A 1836 C6 1818     A        LDA    VECTOR+4
ØØØ78A 1839 B7 47       A        STA    TIRQW+1
ØØØ79A 183B C6 1819     A        LDA    VECTOR+5
ØØØ8ØA 183E B7 48       A        STA    TIRQW+2
ØØØ81                     *
ØØØ82A 184Ø AE 4F       A        LDX    #SWIFLG
ØØØ83A 1842 7F            INIT   CLR    Ø,X      CLEAR
ØØØ84A 1843 5C                   INCX            WORKING
ØØØ85A 1844 A3 56       A        CPX    #WORK5   STORAGE
ØØØ86A 1846 23 FA    1842        BLS    INIT
ØØØ87A 1848 CD 1DD3     A        JSR    SCNBKP   CLEAR
ØØØ88A 184B A6 FF       A        LDA    #$FF     ALL
ØØØ89A 184D F7            REBCLR STA    Ø,X      BREAKPOINTS
ØØØ9ØA 184E 5C                   INCX
ØØØ91A 184F 5C                   INCX
ØØØ92A 185Ø 5C                   INCX
ØØØ93A 1851 3A 5A       A        DEC    PNCNT
ØØØ94A 1853 26 F8    184D        BNE    REBCLR
ØØØ95A 1855 83                   SWI
ØØØ96                     *
ØØØ97          1856     A SWI    EQU    *
ØØØ98A 1856 ØØ 4F Ø4 185D        BRSET  Ø,SWIFLG,SWICHK FROM RESET?
ØØØ99A 1859 1Ø 4F       A        BSET   Ø,SWIFLG YES
ØØ1ØØA 185B 2Ø 4E    18AB        BRA    GETCMD
ØØ1Ø1A 185D CD 1DD3     A SWICHK JSR    SCNBKP   REMOVE
ØØ1Ø2A 186Ø F6            SWIREP LDA    Ø,X      BREAKPOINTS
ØØ1Ø3A 1861 2B ØB    186E        BMI    SWINOB
ØØ1Ø4A 1863 B7 52       A        STA    ADDRH
ØØ1Ø5A 1865 E6 Ø1       A        LDA    1,X
```

```
ØØ106A 1867 B7 53        A        STA    ADDRL
ØØ107A 1869 E6 Ø2        A        LDA    2,X
ØØ108A 186B CD 1F24      A        JSR    STORE
ØØ109A 186E 5C            SWINOB INCX              GET NEXT B.P.
ØØ11ØA 186F 5C                   INCX
ØØ111A 187Ø 5C                   INCX
ØØ112A 1871 3A 5A        A        DEC    PNCNT
ØØ113A 1873 26 EB     186Ø        BNE    SWIREP
ØØ114                      *
ØØ115A 1875 CD 1916      A        JSR    LOCSTK    FIND STACK
ØØ116A 1878 E6 Ø8        A        LDA    8,X
ØØ117A 187A AØ Ø1        A        SUB    #1        ADJUST
ØØ118A 187C B7 59        A        STA    TEMP+1
ØØ119A 187E E6 Ø7        A        LDA    7,X
ØØ12ØA 188Ø A2 ØØ        A        SBC    #Ø
ØØ121A 1882 B7 58        A        STA    TEMP
ØØ122A 1884 BF 57        A        STX    WORK6     SAVE STACK LOCATION
ØØ123A 1886 CD 1DD3      A        JSR    SCNBKP    SETUP B.P. SCAN
ØØ124A 1889 F6             SWITRY LDA    Ø,X       ADJUSTED P.C.
ØØ125A 188A 2B 15     18A1        BMI    SWICMP    IN B.P. TABLE?
ØØ126A 188C B1 58        A        CMP    TEMP
ØØ127A 188E 26 11     18A1        BNE    SWICMP
ØØ128A 189Ø E6 Ø1        A        LDA    1,X
ØØ129A 1892 B1 59        A        CMP    TEMP+1
ØØ13ØA 1894 26 ØB     18A1        BNE    SWICMP    NO,TRY AGAIN
ØØ131A 1896 BE 57        A        LDX    WORK6     YES,RESTORE S.P.
ØØ132A 1898 E7 Ø8        A        STA    8,X       PUT ADJUSTED P.C.
ØØ133A 189A B6 58        A        LDA    TEMP      INTO STACK
ØØ134A 189C E7 Ø7        A        STA    7,X
ØØ135A 189E CC 1B31      A        JMP    TRACE     EXECUTE 1 INSTRUCTION
ØØ136A 18A1 5C             SWICMP INCX             NEXT B.P.
ØØ137A 18A2 5C                    INCX
ØØ138A 18A3 5C                    INCX
ØØ139A 18A4 3A 5A        A        DEC    PNCNT
ØØ14ØA 18A6 26 E1     1889        BNE    SWITRY    DONE?
ØØ141A 18A8 CC 1928      A        JMP    PCOUNT    YES PRINT P.C.
ØØ142                      *
ØØ143          18AB      A GETCMD EQU    *
ØØ144A 18AB CD 1DF5      A        JSR    CLRTAB
ØØ145A 18AE A6 AØ        A        LDA    #PROMPT   PRINT
ØØ146A 18BØ B7 49        A        STA    DTABL     '='
ØØ147A 18B2 CD 1DFD      A        JSR    DISTAB    PROMPT
ØØ148                      *
ØØ149A 18B5 CD 1E23      A CMDSCN JSR    KEYSCN    CHECK KEYPAD
ØØ15ØA 18B8 24 FB     18B5        BCC    CMDSCN
ØØ151A 18BA 5F                    CLRX
ØØ152A 18BB B7 5Ø        A        STA    WORK1
ØØ153A 18BD D6 18D2      A RJUMP  LDA    PTABL,X   THIS COMMAND?
ØØ154A 18CØ B1 5Ø        A        CMP    WORK1
ØØ155A 18C2 27 ØA     18CE        BEQ    PJUMP     YES
ØØ156A 18C4 A1 68        A        CMP    #$68
ØØ157A 18C6 27 E3     18AB        BEQ    GETCMD
ØØ158A 18C8 5C                    INCX             NO
ØØ159A 18C9 5C                    INCX             GO TO
ØØ16ØA 18CA 5C                    INCX             NEXT
ØØ161A 18CB 5C                    INCX             POSSIBLE
ØØ162A 18CC 2Ø EF     18BD        BRA    RJUMP     TRY AGAIN
ØØ163A 18CE 5C             PJUMP  INCX             GO TO
```

```
00164A 18CF DC 18D2      A       JMP     PTABL,X  COMMAND
00165                   *
```

```
ØØ167                     *
ØØ168A 18D2    11       A PTABL  FCB    $11
ØØ169A 18D3    CC       A        FCB    LJMP
ØØ17ØA 18D4    1928     A        FDB    PCOUNT   PROGRAM COUNTER
ØØ171A 18D6    12       A        FCB    $12
ØØ172A 18D7    CC       A        FCB    LJMP
ØØ173A 18D8    194Ø     A        FDB    AREG     ACCUMULATOR
ØØ174A 18DA    14       A        FCB    $14
ØØ175A 18DB    CC       A        FCB    LJMP
ØØ176A 18DC    195A     A        FDB    XREG     INDEX REGISTER
ØØ177A 18DE    18       A        FCB    $18
ØØ178A 18DF    CC       A        FCB    LJMP
ØØ179A 18EØ    1977     A        FDB    CCODE    CONDITION CODE
ØØ18Ø                     *
ØØ181A 18E2    28       A        FCB    $28
ØØ182A 18E3    CC       A        FCB    LJMP
ØØ183A 18E4    1FD7     A        FDB    PWRDWN   UNUSED
ØØ184                     *
ØØ185A 18E6    32       A        FCB    $32
ØØ186A 18E7    CC       A        FCB    LJMP
ØØ187A 18E8    1A78     A        FDB    BPDIS    DISPLAY/SET BP
ØØ188A 18EA    34       A        FCB    $34
ØØ189A 18EB    CC       A        FCB    LJMP
ØØ19ØA 18EC    1AD6     A        FDB    BPCLR    CLEAR BP
ØØ191A 18EE    38       A        FCB    $38
ØØ192A 18EF    CC       A        FCB    LJMP
ØØ193A 18FØ    1B31     A        FDB    TRACE    TRACE ONE INSTRUCTION
ØØ194                     *
ØØ195A 18F2    42       A        FCB    $42
ØØ196A 18F3    CC       A        FCB    LJMP
ØØ197A 18F4    1CØB     A        FDB    DTIME    DISPLAY TIME
ØØ198A 18F6    44       A        FCB    $44
ØØ199A 18F7    CC       A        FCB    LJMP
ØØ2ØØA 18F8    1B86     A        FDB    STIME    SET TIME
ØØ2Ø1A 18FA    48       A        FCB    $48
ØØ2Ø2A 18FB    CC       A        FCB    LJMP
ØØ2Ø3A 18FC    19E5     A        FDB    OFFSET   OFFSET CALCULATION
ØØ2Ø4                     *
ØØ2Ø5A 18FE    51       A        FCB    $51
ØØ2Ø6A 18FF    CC       A        FCB    LJMP
ØØ2Ø7A 19ØØ    1C35     A        FDB    PUNCH    PUNCH TAPE
ØØ2Ø8A 19Ø2    52       A        FCB    $52
ØØ2Ø9A 19Ø3    CC       A        FCB    LJMP
ØØ21ØA 19Ø4    1CDD     A        FDB    TLOAD    LOAD TAPE
ØØ211A 19Ø6    54       A        FCB    $54
ØØ212A 19Ø7    CC       A        FCB    LJMP
ØØ213A 19Ø8    1D81     A        FDB    VERIFY   VERIFY TAPE
ØØ214                     *
ØØ215A 19ØA    62       A        FCB    $62
ØØ216A 19ØB    CC       A        FCB    LJMP
ØØ217A 19ØC    1D8F     A        FDB    GO       GO
ØØ218A 19ØE    64       A        FCB    $64
ØØ219A 19ØF    CC       A        FCB    LJMP
ØØ22ØA 191Ø    1EAA     A        FDB    MEMEX    MEMORY
ØØ221A 1912    68       A        FCB    $68
ØØ222A 1913    CC       A        FCB    LJMP
ØØ223A 1914    1DDA     A        FDB    STACK    STACK
ØØ224                     *
```

```
ØØ226                         *****************************************
ØØ227                         *                                       *
ØØ228                         *    SEARCH FOR STACK POINTER           *
ØØ229                         *                                       *
ØØ23Ø                         *    X-REG CONTAINS SP-3                *
ØØ231                         *                                       *
ØØ232                         *    A-REG DESTROYED                    *
ØØ233                         *                   *
ØØ234                         *****************************************
ØØ235                         *
ØØ236A 1916 AD Ø1     1919 LOCSTK BSR    LOCST2
ØØ237          ØØ19      A STKHI  EQU    */256
ØØ238          ØØ18      A STKLOW EQU    *-(*/256)*256
ØØ239A 1918 81                    RTS
ØØ24ØA 1919 AE 7F        A LOCST2 LDX    #$7F
ØØ241A 191B A6 19        A LOCLOP LDA    #STKHI
ØØ242A 191D 5A             LOCDWN DECX
ØØ243A 191E F1                    CMP    Ø,X
ØØ244A 191F 26 FC     191D        BNE    LOCDWN
ØØ245A 1921 A6 18                 LDA    #STKLOW
ØØ246A 1923 E1 Ø1        A        CMP    1,X
ØØ247A 1925 26 F4     191B        BNE    LOCLOP
ØØ248A 1927 81                    RTS
ØØ249                         *
ØØ25Ø                         *****************************************
ØØ251                         *                                       *
ØØ252                         *    DISPLAY PROGRAM COUNTER            *
ØØ253                         *                                       *
ØØ254                         *****************************************
ØØ255                         *
ØØ256          1928      A PCOUNT EQU    *
ØØ257A 1928 A6 73        A        LDA    #$73      PRINT
ØØ258A 192A B7 4D        A        STA    DTABL+4   'PC'
ØØ259A 192C A6 D1        A        LDA    #$D1
ØØ26ØA 192E B7 4E        A        STA    DTABL+5
ØØ261A 193Ø AD E4     1916        BSR    LOCSTK    FIND USER PC
ØØ262A 1932 E6 Ø7        A        LDA    7,X       HIGH BYTE
ØØ263A 1934 B7 52        A        STA    ADDRH
ØØ264A 1936 E6 Ø8        A        LDA    8,X       LOW BYTE
ØØ265A 1938 B7 53        A        STA    ADDRL     PRINT IT
ØØ266A 193A CD 1FBØ      A        JSR    PRTADR
ØØ267A 193D CC 18B5      A        JMP    CMDSCN
ØØ268                         *
ØØ269                         *****************************************
ØØ27Ø                         *                                       *
ØØ271                         *    ACCUMULATOR EXAMINE/CHANGE         *
ØØ272                         *                                       *
ØØ273                         *****************************************
ØØ274                         *
ØØ275          194Ø      A AREG   EQU    *
ØØ276A 194Ø A6 77        A        LDA    #$77      PRINT 'ACCA'
ØØ277A 1942 B7 49        A        STA    DTABL
ØØ278A 1944 B7 4C        A        STA    DTABL+3
ØØ279A 1946 A6 D1        A        LDA    #$D1
ØØ28ØA 1948 B7 4A        A        STA    DTABL+1
ØØ281A 194A B7 4B        A        STA    DTABL+2
ØØ282A 194C AD C8     1916        BSR    LOCSTK    FIND ACCUM. VALUE
ØØ283A 194E 9F                    TXA
```

```
ØØ284A 194F AB Ø5       A        ADD    #5
ØØ285A 1951 3F 52       A        CLR    ADDRH     SETUP FOR
ØØ286A 1953 B7 53       A        STA    ADDRL     EXAMINE/CHANGE
ØØ287A 1955 1C 4F       A        BSET   6,SWIFLG
ØØ288A 1957 CC 1EB1     A        JMP    MEMEX3    USING MEMORY ROUTINE
ØØ289                     *
ØØ29Ø                     ***************************************
ØØ291                     *                                     *
ØØ292                     *      INDEX REGISTER EXAMINE/CHANGE  *
ØØ293                     *                                     *
ØØ294                     ***************************************
ØØ295                     *
ØØ296            195A   A XREG   EQU    *
ØØ297A 195A A6 Ø6       A        LDA    #6        PRINT 'ID'
ØØ298A 195C CD 1DF5     A        JSR    CLRTAB
ØØ299A 195F B7 4A       A        STA    DTABL+1
ØØ3ØØA 1961 A6 E6       A        LDA    #$E6
ØØ3Ø1A 1963 B7 4B       A        STA    DTABL+2
ØØ3Ø2A 1965 A6 6Ø       A        LDA    #$6Ø
ØØ3Ø3A 1967 B7 4C       A        STA    DTABL+3
ØØ3Ø4A 1969 AD AB    1916        BSR    LOCSTK    FIND INDEX
ØØ3Ø5A 196B 9F                   TXA              REGISTER VALUE
ØØ3Ø6A 196C AB Ø6       A        ADD    #6
ØØ3Ø7A 196E 3F 52       A        CLR    ADDRH     SETUP FOR
ØØ3Ø8A 197Ø B7 53       A        STA    ADDRL     EXAMINE/CHANGE
ØØ3Ø9A 1972 1C 4F       A        BSET   6,SWIFLG
ØØ31ØA 1974 CC 1EB1     A        JMP    MEMEX3    USING MEMORY ROUTINE
ØØ311                     *
ØØ312                     ***************************************
ØØ313                     *                                     *
ØØ314                     *      CONDITION CODE                 *
ØØ315                     *      EXAMINE/CHANGE                 *
ØØ316                     *                                     *
ØØ317                     ***************************************
ØØ318                     *
ØØ319            1977   A CCODE  EQU    *
ØØ32ØA 1977 CD 1DF5     A        JSR    CLRTAB
ØØ321A 197A A6 D1       A        LDA    #$D1
ØØ322A 197C B7 49       A        STA    DTABL
ØØ323A 197E A6 D7       A        LDA    #$D7
ØØ324A 198Ø B7 4A       A        STA    DTABL+1
ØØ325A 1982 A6 E6       A        LDA    #$E6
ØØ326A 1984 B7 4B       A        STA    DTABL+2
ØØ327A 1986 A6 F1       A        LDA    #$F1
ØØ328A 1988 B7 4C       A        STA    DTABL+3
ØØ329A 198A AD 8A    1916        BSR    LOCSTK    FIND CONDITION
ØØ33ØA 198C 9F                   TXA              CODES
ØØ331A 198D AB Ø4       A        ADD    #4
ØØ332A 198F 3F 52       A        CLR    ADDRH     SETUP FOR
ØØ333A 1991 B7 53       A        STA    ADDRL     EXAMINE/CHANGE
ØØ334A 1993 1C 4F       A        BSET   6,SWIFLG
ØØ335A 1995 CC 1EB1     A        JMP    MEMEX3    USING MEMORY ROUTINE
ØØ336                     *
ØØ337                     ***************************************
ØØ338                     *                                     *
ØØ339                     *             BUILD A BEGINNING       *
ØØ34Ø                     *             AND ENDING              *
ØØ341                     *             ADDRESS RANGE           *
```

```
ØØ342                     *                                      *
ØØ343                     *             TEMP,TEMP+1 : BEGINNING  *
ØØ344                     *             ADDRH,ADDRL : ENDING     *
ØØ345                     *                                      *
ØØ346                     ****************************************
ØØ347                     *
ØØ348A 1998 19 4F       A BLDRNG BCLR   4,SWIFLG
ØØ349A 199A 17 4F       A        BCLR   3,SWIFLG
ØØ35ØA 199C CD 1DF5     A        JSR    CLRTAB   PRINT
ØØ351A 199F A6 F4       A        LDA    #$F4     'BA'
ØØ352A 19A1 B7 4D       A        STA    DTABL+4
ØØ353A 19A3 A6 77       A        LDA    #$77
ØØ354A 19A5 B7 4E       A        STA    DTABL+5
ØØ355A 19A7 CD 1DFD     A        JSR    DISTAB
ØØ356A 19AA CD 1F58     A        JSR    BLDADR   GET SOURCE ADDR.
ØØ357A 19AD 24 2C    19DB        BCC    BLDRN1   VALID?
ØØ358A 19AF B6 52       A        LDA    ADDRH    YES
ØØ359A 19B1 A1 1F       A        CMP    #PCMASK  TOO BIG?
ØØ36ØA 19B3 22 2A    19DF        BHI    BLDRN2   YES
ØØ361A 19B5 B7 58       A        STA    TEMP     NO SAVE IT
ØØ362A 19B7 B6 53       A        LDA    ADDRL
ØØ363A 19B9 B7 59       A        STA    TEMP+1
ØØ364A 19BB CD 1F15     A        JSR    LOAD     FETCH OPCODE OF INSTR.
ØØ365A 19BE B7 57       A        STA    WORK6    SAVE IT
ØØ366A 19CØ CD 1DF5     A        JSR    CLRTAB
ØØ367A 19C3 A6 F1       A        LDA    #$F1     PRINT 'EA'
ØØ368A 19C5 B7 4D       A        STA    DTABL+4
ØØ369A 19C7 A6 77       A        LDA    #$77
ØØ37ØA 19C9 B7 4E       A        STA    DTABL+5
ØØ371A 19CB CD 1DFD     A        JSR    DISTAB
ØØ372A 19CE CD 1F58     A        JSR    BLDADR   GET DESTINATION ADDR
ØØ373A 19D1 24 Ø8    19DB        BCC    BLDRN1   VALID?
ØØ374A 19D3 B6 52       A        LDA    ADDRH    YES
ØØ375A 19D5 A1 1F       A        CMP    #PCMASK  TOO BIG?
ØØ376A 19D7 22 Ø6    19DF        BHI    BLDRN2   YES
ØØ377A 19D9 20 Ø6    19E1        BRA    BLDRET
ØØ378A 19DB 18 4F       A BLDRN1 BSET   4,SWIFLG INVALID
ØØ379A 19DD 20 Ø2    19E1        BRA    BLDRET
ØØ38ØA 19DF 16 4F       A BLDRN2 BSET   3,SWIFLG TOO BIG
ØØ381A 19E1 81            BLDRET RTS
ØØ382                     *
ØØ383                     ****************************************
ØØ384                     *                                      *
ØØ385                     *             CALCULATE BRANCH OFFSET  *
ØØ386                     *             FOR BIT AND CONDITIONAL  *
ØØ387                     *             BRANCHES                 *
ØØ388                     *                                      *
ØØ389                     *             OPCODE MUST BE AT        *
ØØ39Ø                     *             BEGINNING ADDRESS        *
ØØ391                     *                                      *
ØØ392                     *             OFFSET WILL BE INSERTED  *
ØØ393                     *             INTO BRANCH INSTRUCTION  *
ØØ394                     *                                      *
ØØ395                     ****************************************
ØØ396                     *
ØØ397A 19E2 CC 1E97     A OFFERR JMP    ERROR
ØØ398                     *
ØØ399          19E5     A OFFSET EQU    *
```

```
ØØ4ØØA 19E5 AD B1        1998          BSR    BLDRNG
ØØ4Ø1A 19E7 Ø8 4F 2B     1A15          BRSET  4,SWIFLG,ORET
ØØ4Ø2A 19EA Ø6 4F F5     19E2          BRSET  3,SWIFLG,OFFERR
ØØ4Ø3A 19ED B6 53           A          LDA    ADDRL      NO FIND APPARRENT
ØØ4Ø4A 19EF BØ 59           A          SUB    TEMP+1     OFFSET
ØØ4Ø5A 19F1 AØ Ø2           A          SUB    #2
ØØ4Ø6A 19F3 B7 53           A          STA    ADDRL
ØØ4Ø7A 19F5 B6 52           A          LDA    ADDRH
ØØ4Ø8A 19F7 B2 58           A          SBC    TEMP
ØØ4Ø9A 19F9 B7 52           A          STA    ADDRH
ØØ41ØA 19FB B6 57           A          LDA    WORK6      CHECK OPCODE
ØØ411A 19FD A1 1F           A          CMP    #$1F       FOR BIT BRANCH
ØØ412A 19FF 23 41        1A42          BLS    OFFST1
ØØ413A 1AØ1 B6 52           A          LDA    ADDRH
ØØ414A 1AØ3 A1 FF           A          CMP    #$FF       + OR - OFFSET?
ØØ415A 1AØ5 27 Ø3        1AØA          BEQ    OFFST2
ØØ416A 1AØ7 4D                         TSTA              CHECK OFFSET
ØØ417A 1AØ8 26 6Ø        1A6A          BNE    OVRERR     FOR +/- Ø
ØØ418A 1AØA B6 53           A OFFST2   LDA    ADDRL
ØØ419A 1AØC A1 FF           A          CMP    #$FF
ØØ42ØA 1AØE 27 5A        1A6A          BEQ    OVRERR
ØØ421A 1A1Ø AD Ø6        1A18          BSR    USE        PRINT IT IF VALID
ØØ422A 1A12 CC 18B5         A          JMP    CMDSCN
ØØ423A 1A15 CC 18AB         A ORET     JMP    GETCMD
ØØ424                         *
ØØ425A 1A18 CD 1DF5         A USE      JSR    CLRTAB
ØØ426A 1A1B A6 D6           A          LDA    #$D6       PRINT 'USED'
ØØ427A 1A1D B7 49           A          STA    DTABL
ØØ428A 1A1F A6 B5           A          LDA    #$B5
ØØ429A 1A21 B7 4A           A          STA    DTABL+1
ØØ43ØA 1A23 A6 F1           A          LDA    #$F1
ØØ431A 1A25 B7 4B           A          STA    DTABL+2
ØØ432A 1A27 A6 E6           A          LDA    #$E6
ØØ433A 1A29 B7 4C           A          STA    DTABL+3
ØØ434A 1A2B B6 53           A          LDA    ADDRL      PRINT OFFSET
ØØ435A 1A2D CD 1F8C         A          JSR    PRTDAT
ØØ436A 1A3Ø 97                         TAX
ØØ437A 1A31 B6 59           A          LDA    TEMP+1
ØØ438A 1A33 AB Ø1           A          ADD    #1
ØØ439A 1A35 B7 53           A          STA    ADDRL
ØØ44ØA 1A37 B6 58           A          LDA    TEMP
ØØ441A 1A39 A9 ØØ           A          ADC    #Ø         PUT INTO
ØØ442A 1A3B B7 52           A          STA    ADDRH      INSTRUCTION
ØØ443A 1A3D 9F                         TXA
ØØ444A 1A3E CD 1F24         A          JSR    STORE
ØØ445A 1A41 81                         RTS
ØØ446                         *
ØØ447A 1A42 B6 53           A OFFST1   LDA    ADDRL      ADJUST FOR
ØØ448A 1A44 AØ Ø1           A          SUB    #1         BIT BRANCH
ØØ449A 1A46 B7 53           A          STA    ADDRL
ØØ45ØA 1A48 B6 52           A          LDA    ADDRH
ØØ451A 1A4A A2 ØØ           A          SBC    #Ø
ØØ452A 1A4C B7 52           A          STA    ADDRH
ØØ453A 1A4E A1 FF           A          CMP    #$FF       NEG OFFSET?
ØØ454A 1A5Ø 27 Ø3        1A55          BEQ    OFFST3     YES
ØØ455A 1A52 4D                         TSTA              CHECK FOR
ØØ456A 1A53 26 15        1A6A          BNE    OVRERR     +/- Ø AND -1
ØØ457A 1A55 B6 53           A OFFST3   LDA    ADDRL
```

```
ØØ458A 1A57 A1 FF         A        CMP    #$FF
ØØ459A 1A59 27 ØF      1A6A        BEQ    OVRERR
ØØ46ØA 1A5B A1 FE         A        CMP    #$FE
ØØ461A 1A5D 27 ØB      1A6A        BEQ    OVRERR
ØØ462A 1A5F 3C 59         A        INC    TEMP+1
ØØ463A 1A61 26 Ø2      1A65        BNE    OFFITS
ØØ464A 1A63 3C 58         A        INC    TEMP
ØØ465A 1A65 AD B1      1A18 OFFITS BSR    USE      PRINT IF VALID
ØØ466A 1A67 CC 18B5       A        JMP    CMDSCN
ØØ467                      *
ØØ468A 1A6A A6 D7         A OVRERR LDA    #$D7     PRINT 'OR'
ØØ469A 1A6C B7 4D         A        STA    DTABL+4
ØØ47ØA 1A6E A6 6Ø         A        LDA    #$6Ø
ØØ471A 1A7Ø B7 4E         A        STA    DTABL+5
ØØ472A 1A72 CD 1FBØ       A        JSR    PRTADR
ØØ473A 1A75 CC 18B5       A        JMP    CMDSCN
ØØ474                      *
ØØ475                      ****************************************
ØØ476                      *                                      *
ØØ477                      *             DISPLAY/SET BREAKPOINTS  *
ØØ478                      *                                      *
ØØ479                      ****************************************
ØØ48Ø                      *
ØØ481             1A78    A BPDIS  EQU    *
ØØ482A 1A78 3F 57         A        CLR    WORK6
ØØ483A 1A7A 3A 57         A        DEC    WORK6
ØØ484A 1A7C CD 1DD3       A        JSR    SCNBKP   FIND B.P. TABLE
ØØ485A 1A7F BF 51         A        STX    WORK2
ØØ486A 1A81 3F 4D         A BPDIS1 CLR    DTABL+4
ØØ487A 1A83 F6                     LDA    Ø,X      GET B.P.
ØØ488A 1A84 2A 1Ø      1A96        BPL    BPDIS2   VALID?
ØØ489A 1A86 A6 F4         A        LDA    #$F4     NO
ØØ49ØA 1A88 B7 49         A        STA    DTABL    PRINT 'BOFF'
ØØ491A 1A8A A6 D7         A        LDA    #$D7
ØØ492A 1A8C B7 4A         A        STA    DTABL+1
ØØ493A 1A8E A6 71         A        LDA    #$71
ØØ494A 1A9Ø B7 4B         A        STA    DTABL+2
ØØ495A 1A92 B7 4C         A        STA    DTABL+3
ØØ496A 1A94 2Ø Ø9      1A9F        BRA    BPDIS4
ØØ497A 1A96 B7 52         A BPDIS2 STA    ADDRH    PRINT B.P.
ØØ498A 1A98 E6 Ø1         A        LDA    1,X
ØØ499A 1A9A B7 53         A        STA    ADDRL
ØØ5ØØA 1A9C CD 1FBØ       A        JSR    PRTADR
ØØ5Ø1A 1A9F 3C 57         A BPDIS4 INC    WORK6    PRINT B.P. #
ØØ5Ø2A 1AA1 BE 57         A        LDX    WORK6
ØØ5Ø3A 1AA3 D6 1E87       A        LDA    CTABL,X
ØØ5Ø4A 1AA6 B7 4E         A        STA    DTABL+5
ØØ5Ø5A 1AA8 CD 1DFD       A        JSR    DISTAB
ØØ5Ø6A 1AAB CD 1F58       A        JSR    BLDADR   NEW B.P.
ØØ5Ø7A 1AAE BE 51         A        LDX    WORK2
ØØ5Ø8A 1ABØ 25 Ø8      1ABA        BCS    BPDIS7   YES
ØØ5Ø9A 1AB2 A1 1Ø         A        CMP    #$1Ø     NO,ESC?
ØØ51ØA 1AB4 27 1A      1ADØ        BEQ    BPRET    GET OUT
ØØ511A 1AB6 A1 11         A        CMP    #$11     ENTER?
ØØ512A 1AB8 27 ØB      1AC5        BEQ    BPDIS5   GET NEXT B.P.
ØØ513A 1ABA B6 52         A BPDIS7 LDA    ADDRH    TOO BIG?
ØØ514A 1ABC A1 1F         A        CMP    #PCMASK
ØØ515A 1ABE 22 13      1AD3        BHI    BPERR    YES
```

```
ØØ516A 1ACØ F7                 STA    Ø,X       NO,STORE NEW B.P.
ØØ517A 1AC1 B6 53        A     LDA    ADDRL
ØØ518A 1AC3 E7 Ø1              STA    1,X
ØØ519A 1AC5 5C           BPDIS5 INCX            GET NEXT B.P.
ØØ52ØA 1AC6 5C                 INCX
ØØ521A 1AC7 5C                 INCX
ØØ522A 1AC8 BF 51        A     STX    WORK2
ØØ523A 1ACA 3A 5A        A     DEC    PNCNT
ØØ524A 1ACC 26 B3   1A81       BNE    BPDIS1    DONE?
ØØ525A 1ACE 2Ø A8   1A78       BRA    BPDIS     YES START OVER
ØØ526A 1ADØ CC 18AB      A BPRET  JMP    GETCMD
ØØ527                    *
ØØ528A 1AD3 CC 1E97      A BPERR  JMP    ERROR
ØØ529                    *
ØØ53Ø                    ****************************************
ØØ531                    *                                      *
ØØ532                    *              BREAKPOINT CLEAR        *
ØØ533                    *                                      *
ØØ534                    *              TYPE # FOR SINGLE       *
ØØ535                    *              CLEAR AND ENT FOR ALL   *
ØØ536                    *                                      *
ØØ537                    ****************************************
ØØ538                    *
ØØ539       1AD6         A BPCLR  EQU    *
ØØ54ØA 1AD6 CD 1DF5      A     JSR    CLRTAB    PRINT 'BCLR'
ØØ541A 1AD9 A6 F4        A     LDA    #$F4
ØØ542A 1ADB B7 49        A     STA    DTABL
ØØ543A 1ADD A6 D1        A     LDA    #$D1
ØØ544A 1ADF B7 4A        A     STA    DTABL+1
ØØ545A 1AE1 A6 DØ        A     LDA    #$DØ
ØØ546A 1AE3 B7 4B        A     STA    DTABL+2
ØØ547A 1AE5 A6 6Ø        A     LDA    #$6Ø
ØØ548A 1AE7 B7 4C        A     STA    DTABL+3
ØØ549A 1AE9 CD 1DFD      A     JSR    DISTAB
ØØ55ØA 1AEC CD 1DD3      A     JSR    SCNBKP    FIND B.P. TABLE
ØØ551A 1AEF BF 51        A     STX    WORK2
ØØ552A 1AF1 CD 1F49      A     JSR    GETNYB
ØØ553A 1AF4 25 12   1BØ8       BCS    BPCLR1    ENTER?
ØØ554A 1AF6 A1 11        A     CMP    #$11
ØØ555A 1AF8 26 34   1B2E       BNE    BPCRET    NO
ØØ556A 1AFA A6 FF        A     LDA    #$FF      YES,CLEAR ALL
ØØ557A 1AFC BE 51        A     LDX    WORK2
ØØ558A 1AFE F7           BPCLR2 STA    Ø,X
ØØ559A 1AFF 5C                 INCX
ØØ56ØA 1BØØ 5C                 INCX
ØØ561A 1BØ1 5C                 INCX
ØØ562A 1BØ2 3A 5A        A     DEC    PNCNT
ØØ563A 1BØ4 26 F8   1AFE       BNE    BPCLR2
ØØ564A 1BØ6 2Ø 26   1B2E       BRA    BPCRET
ØØ565A 1BØ8 A1 Ø3        A BPCLR1 CMP    #NUMBKP   VALID B.P. #?
ØØ566A 1BØA 24 C7   1AD3       BHS    BPERR     NO
ØØ567A 1BØC 97                 TAX              YES
ØØ568A 1BØD D6 1E87      A     LDA    CTABL,X   PRINT B.P. #
ØØ569A 1B1Ø B7 4E        A     STA    DTABL+5
ØØ57ØA 1B12 4F                 CLRA             FIND IT
ØØ571A 1B13 AØ Ø3        A     SUB    #3
ØØ572A 1B15 AB Ø3        A BPCLR3 ADD    #3
ØØ573A 1B17 5A                 DECX
```

```
ØØ574A 1B18 2A FB      1B15        BPL   BPCLR3
ØØ575A 1B1A B7 57         A        STA   WORK6
ØØ576A 1B1C CD 1DFD       A        JSR   DISTAB   PRINT B.P.
ØØ577A 1B1F CD 1E5F       A        JSR   CHRIN
ØØ578A 1B22 A1 11         A        CMP   #$11     CLEAR IT?
ØØ579A 1B24 26 Ø8      1B2E        BNE   BPCRET   NO
ØØ58ØA 1B26 A6 37         A        LDA   #BKPTBL  YES
ØØ581A 1B28 BB 57         A        ADD   WORK6
ØØ582A 1B2A 97            A        TAX
ØØ583A 1B2B A6 FF         A        LDA   #$FF
ØØ584A 1B2D F7            A        STA   Ø,X
ØØ585A 1B2E CC 18AB       A BPCRET JMP   GETCMD
ØØ586                       *
ØØ587                       ************************************
ØØ588                       *                                  *
ØØ589                       *            TRACE ONE INSTRUCTION *
ØØ59Ø                       *                                  *
ØØ591                       *            TIMER INTERRUPT IS    *
ØØ592                       *            USED                  *
ØØ593                       *                                  *
ØØ594                       ************************************
ØØ595                       *
ØØ596       1B31          A TRACE  EQU   *
ØØ597A 1B31 CD 1916       A        JSR   LOCSTK   FIND S.P.
ØØ598A 1B34 E6 Ø4         A        LDA   4,X
ØØ599A 1B36 A4 Ø8         A        AND   #8
ØØ6ØØA 1B38 B7 57         A        STA   WORK6
ØØ6Ø1A 1B3A E6 Ø7         A        LDA   7,X
ØØ6Ø2A 1B3C B7 52         A        STA   ADDRH
ØØ6Ø3A 1B3E E6 Ø8         A        LDA   8,X
ØØ6Ø4A 1B4Ø B7 53         A        STA   ADDRL
ØØ6Ø5A 1B42 CD 1F15       A        JSR   LOAD     GET OPCODE
ØØ6Ø6A 1B45 A1 83         A        CMP   #$83     SWI?
ØØ6Ø7A 1B47 26 ØF      1B58        BNE   TRACE3
ØØ6Ø8A 1B49 B6 53         A        LDA   ADDRL    YES
ØØ6Ø9A 1B4B AB Ø1         A        ADD   #1       INC PC
ØØ61ØA 1B4D E7 Ø8         A        STA   8,X
ØØ611A 1B4F B6 52         A        LDA   ADDRH
ØØ612A 1B51 A9 ØØ         A        ADC   #Ø
ØØ613A 1B53 E7 Ø7         A        STA   7,X
ØØ614A 1B55 CC 1928       A        JMP   PCOUNT
ØØ615A 1B58 A1 9B         A TRACE3 CMP   #$9B     SEI?
ØØ616A 1B5A 26 15      1B71        BNE   TRACE2
ØØ617A 1B5C E6 Ø4         A        LDA   4,X      YES
ØØ618A 1B5E AA Ø8         A        ORA   #8       SET IT IN
ØØ619A 1B6Ø E7 Ø4         A        STA   4,X      STACK
ØØ62ØA 1B62 B6 53         A        LDA   ADDRL
ØØ621A 1B64 AB Ø1         A        ADD   #1
ØØ622A 1B66 E7 Ø8         A        STA   8,X
ØØ623A 1B68 B6 52         A        LDA   ADDRH
ØØ624A 1B6A A9 ØØ         A        ADC   #Ø
ØØ625A 1B6C E7 Ø7         A        STA   7,X
ØØ626A 1B6E CC 1928       A        JMP   PCOUNT
ØØ627A 1B71 A1 9A         A TRACE2 CMP   #$9A     CLI?
ØØ628A 1B73 26 Ø2      1B77        BNE   TRACE1
ØØ629A 1B75 3F 57         A        CLR   WORK6    YES,CLEAR IT ON STACK
ØØ63ØA 1B77 E6 Ø4         A TRACE1 LDA   4,X      GET COND. CODE
ØØ631A 1B79 A4 F7         A        AND   #$F7     CLEAR IRQ BIT
```

```
ØØ632A 1B7B E7 Ø4       A        STA    4,X        RETURN TO STACK
ØØ633A 1B7D A6 1Ø       A        LDA    #16
ØØ634A 1B7F B7 Ø8       A        STA    TIMER
ØØ635A 1B81 A6 Ø8       A        LDA    #8
ØØ636A 1B83 B7 Ø9       A        STA    TIMEC
ØØ637A 1B85 8Ø                   RTI               EXECUTE
ØØ638                   *
ØØ639                   ****************************************
ØØ64Ø                   *                                      *
ØØ641                   *               SET CURRENT TIME       *
ØØ642                   *               USING MC146818         *
ØØ643                   *                                      *
ØØ644                   *               12-HOUR FORMAT         *
ØØ645                   *                                      *
ØØ646                   ****************************************
ØØ647                   *
ØØ648       1B86        A STIME  EQU    *
ØØ649A 1B86 CD 1DF5     A        JSR    CLRTAB
ØØ65ØA 1B89 A6 77       A        LDA    #$77       AM BY DEFAULT
ØØ651A 1B8B B7 4E       A        STA    DTABL+5
ØØ652A 1B8D 3F 53       A        CLR    ADDRL
ØØ653A 1B8F 3F 52       A        CLR    ADDRH
ØØ654A 1B91 CD 1FBØ     A STIME2 JSR    PRTADR
ØØ655A 1B94 CD 1F49     A        JSR    GETNYB     GET INPUT
ØØ656A 1B97 25 12    1BAB        BCS    STIME1
ØØ657A 1B99 A1 1Ø       A        CMP    #$1Ø       ESC?
ØØ658A 1B9B 27 4F    1BEC        BEQ    STMRET
ØØ659A 1B9D A1 11       A        CMP    #$11       ENT?
ØØ66ØA 1B9F 27 1D    1BBE        BEQ    STIME4
ØØ661A 1BA1 A1 17       A        CMP    #$17       P?
ØØ662A 1BA3 26 EC    1B91        BNE    STIME2
ØØ663A 1BA5 A6 73       A        LDA    #$73       YES,
ØØ664A 1BA7 B7 4E       A        STA    DTABL+5    PRINT P
ØØ665A 1BA9 2Ø E6    1B91        BRA    STIME2
ØØ666A 1BAB A1 Ø9       A STIME1 CMP    #9         GT 9?
ØØ667A 1BAD 22 4Ø    1BEF        BHI    STERR
ØØ668A 1BAF AE Ø4       A        LDX    #4         SHIFT IN NEW
ØØ669A 1BB1 38 53       A STIME3 LSL    ADDRL      INPUT
ØØ67ØA 1BB3 39 52       A        ROL    ADDRH
ØØ671A 1BB5 5A                   DECX
ØØ672A 1BB6 26 F9    1BB1        BNE    STIME3
ØØ673A 1BB8 BA 53       A        ORA    ADDRL
ØØ674A 1BBA B7 53       A        STA    ADDRL
ØØ675A 1BBC 2Ø D3    1B91        BRA    STIME2
ØØ676A 1BBE B6 52       A STIME4 LDA    ADDRH      HOURS GT 12?
ØØ677A 1BCØ A1 12       A        CMP    #$12
ØØ678A 1BC2 22 2B    1BEF        BHI    STERR
ØØ679A 1BC4 4D                   TSTA              HOURS EQ Ø?
ØØ68ØA 1BC5 27 28    1BEF        BEQ    STERR
ØØ681A 1BC7 B6 53       A        LDA    ADDRL      MIN? GT 59?
ØØ682A 1BC9 A1 59       A        CMP    #$59
ØØ683A 1BCB 22 22    1BEF        BHI    STERR
ØØ684A 1BCD A6 8Ø       A        LDA    #$8Ø       PUT IN
ØØ685A 1BCF C7 17ØB     A        STA    CR2        SET TIME MODE
ØØ686A 1BD2 4F                   CLRA
ØØ687A 1BD3 C7 17ØA     A        STA    CR1
ØØ688A 1BD6 Ø4 4E Ø2 1BDB        BRSET  2,DTABL+5,STIME5 PM?
ØØ689A 1BD9 1E 52       A        BSET   7,ADDRH    YES
```

```
ØØ69ØA 1BDB B6 53         A STIME5 LDA    ADDRL    PUT TIME INTO
ØØ691A 1BDD C7 17Ø2       A        STA    MIN      MC146818
ØØ692A 1BEØ B6 52         A        LDA    ADDRH
ØØ693A 1BE2 C7 17Ø4       A        STA    HOUR
ØØ694A 1BE5 4F                     CLRA
ØØ695A 1BE6 C7 17ØB       A        STA    CR2      ALLOW TO RUN
ØØ696A 1BE9 C7 17ØØ       A        STA    SEC      CLR SECONDS
ØØ697A 1BEC CC 18AB       A STMRET JMP    GETCMD
ØØ698                       *
ØØ699A 1BEF CC 1E97       A STERR  JMP    ERROR
ØØ7ØØ                       *
ØØ7Ø1                       ****************************************
ØØ7Ø2                       *                                      *
ØØ7Ø3                       *             WAIT FOR THE END         *
ØØ7Ø4                       *             OF UPDATE CYCLE          *
ØØ7Ø5                       *                                      *
ØØ7Ø6                       ****************************************
ØØ7Ø7                       *
ØØ7Ø8A 1BF2 CD 1E23       A VALID  JSR    KEYSCN
ØØ7Ø9A 1BF5 25 13    1CØA          BCS    VALRET
ØØ71ØA 1BF7 C6 17ØA       A        LDA    CR1      IS UIP LOW?
ØØ711A 1BFA A4 8Ø         A        AND    #$8Ø
ØØ712A 1BFC 27 F4    1BF2          BEQ    VALID    YES,WAIT UNTIL HIGH
ØØ713A 1BFE CD 1E23       A VALID2 JSR    KEYSCN
ØØ714A 1CØ1 25 Ø7    1CØA          BCS    VALRET
ØØ715A 1CØ3 C6 17ØA       A        LDA    CR1      UIP MADE NEG TRANSITION
ØØ716A 1CØ6 A4 8Ø         A        AND    #$8Ø
ØØ717A 1CØ8 26 F4    1BFE          BNE    VALID2
ØØ718A 1CØA 81                VALRET RTS
ØØ719                       *
ØØ72Ø                       ****************************************
ØØ721                       *                                      *
ØØ722                       *             DISPLAY CURRENT TIME     *
ØØ723                       *             FROM MC146818            *
ØØ724                       *                                      *
ØØ725                       *             USES 12-HOUR FORMAT      *
ØØ726                       *                                      *
ØØ727                       ****************************************
ØØ728                       *
ØØ729                1CØB A DTIME  EQU    *
ØØ73ØA 1CØB CD 1DF5       A        JSR    CLRTAB
ØØ731A 1CØE A6 77         A        LDA    #$77
ØØ732A 1C1Ø B7 4E         A        STA    DTABL+5
ØØ733A 1C12 AD DE    1BF2          BSR    VALID    UPDATE OVER
ØØ734A 1C14 24 Ø4    1C1A          BCC    DTIME2
ØØ735A 1C16 5F                     CLRX
ØØ736A 1C17 CC 18BD       A        JMP    RJUMP
ØØ737A 1C1A C6 17Ø4       A DTIME2 LDA    HOUR
ØØ738A 1C1D B7 52         A        STA    ADDRH
ØØ739A 1C1F ØF 52 Ø6 1C28          BRCLR  7,ADDRH,DTIME1 PM?
ØØ74ØA 1C22 1F 52         A        BCLR   7,ADDRH
ØØ741A 1C24 A6 73         A        LDA    #$73     PRINT IT
ØØ742A 1C26 B7 4E         A        STA    DTABL+5
ØØ743A 1C28 C6 17Ø2       A DTIME1 LDA    MIN
ØØ744A 1C2B B7 53         A        STA    ADDRL
ØØ745A 1C2D CD 1FBØ       A        JSR    PRTADR   PRINT TIME
ØØ746A 1C3Ø 2Ø D9    1CØB          BRA    DTIME
ØØ747                       *
```

```
ØØ748                      ****************************************
ØØ749                      *                                      *
ØØ75Ø                      *             PUNCH TAPE               *
ØØ751                      *                                      *
ØØ752                      *             LAST ADDRESS WILL        *
ØØ753                      *             REMAIN UNTIL PUNCH       *
ØØ754                      *             IS COMPLETE              *
ØØ755                      *                                      *
ØØ756                      *             24ØØ BAUD IS DEFAULT     *
ØØ757                      *                                      *
ØØ758                      ****************************************
ØØ759                      *
ØØ76ØA 1C32 CC 1E97      A PUNERR JMP    ERROR
ØØ761                      *
ØØ762          1C35      A PUNCH  EQU    *
ØØ763A 1C35 CD 1998      A        JSR    BLDRNG   BUILD RANGE
ØØ764A 1C38 Ø8 4F 49 1C84         BRSET  4,SWIFLG,PUNRET VALID?
ØØ765A 1C3B Ø6 4F F4 1C32         BRSET  3,SWIFLG,PUNERR VAILD?
ØØ766A 1C3E BE 58        A        LDX    TEMP     SWAP ADDRESSES
ØØ767A 1C4Ø B7 58        A        STA    TEMP
ØØ768A 1C42 BF 52        A        STX    ADDRH
ØØ769A 1C44 B6 53        A        LDA    ADDRL
ØØ77ØA 1C46 BE 59        A        LDX    TEMP+1
ØØ771A 1C48 BF 53        A        STX    ADDRL    ADJUST
ØØ772A 1C4A 4C                    INCA            ENDING
ØØ773A 1C4B 26 Ø2    1C4F         BNE    PUN3     ADDRESS
ØØ774A 1C4D 3C 58        A        INC    TEMP
ØØ775A 1C4F B7 59        A PUN3   STA    TEMP+1
ØØ776A 1C51 AD 3F    1C92         BSR    PUNLDR   PUNCH LEADER
ØØ777A 1C53 A6 B3        A        LDA    #$B3     PUNCH BOT
ØØ778A 1C55 AD 5Ø    1CA7         BSR    PUNBYT
ØØ779A 1C57 3F 5B        A        CLR    CHKSUM   INITIALIZE CHECKSUM
ØØ78ØA 1C59 B6 58        A        LDA    TEMP     PUNCH
ØØ781A 1C5B AD 2A    1C87         BSR    PUNIT    ENDING ADDRESS
ØØ782A 1C5D B6 59        A        LDA    TEMP+1
ØØ783A 1C5F AD 26    1C87         BSR    PUNIT
ØØ784A 1C61 B6 52        A        LDA    ADDRH    PUNCH
ØØ785A 1C63 AD 22    1C87         BSR    PUNIT    BEGINNING ADDRESS
ØØ786A 1C65 B6 53        A        LDA    ADDRL
ØØ787A 1C67 AD 1E    1C87         BSR    PUNIT
ØØ788A 1C69 CD 1F15      A PUN5   JSR    LOAD     GET BYTE FROM MEMORY
ØØ789A 1C6C AD 19    1C87         BSR    PUNIT    PUNCH IT
ØØ79ØA 1C6E 3C 53        A        INC    ADDRL
ØØ791A 1C7Ø 26 Ø2    1C74         BNE    PUN4
ØØ792A 1C72 3C 52        A        INC    ADDRH
ØØ793A 1C74 B6 58        A PUN4   LDA    TEMP     FINISHED?
ØØ794A 1C76 B1 52        A        CMP    ADDRH
ØØ795A 1C78 26 EF    1C69         BNE    PUN5
ØØ796A 1C7A B6 59        A        LDA    TEMP+1
ØØ797A 1C7C B1 53        A        CMP    ADDRL
ØØ798A 1C7E 26 E9    1C69         BNE    PUN5
ØØ799A 1C8Ø B6 5B        A        LDA    CHKSUM   YES, PUNCH
ØØ8ØØA 1C82 AD 23    1CA7         BSR    PUNBYT   CHECKSUM
ØØ8Ø1A 1C84 CC 18AB      A PUNRET JMP    GETCMD
ØØ8Ø2                      *
ØØ8Ø3A 1C87 B7 56        A PUNIT  STA    WORK5
ØØ8Ø4A 1C89 AD 1C    1CA7         BSR    PUNBYT   PUNCH BYTE
ØØ8Ø5A 1C8B B6 56        A        LDA    WORK5    AND UPDATE
```

```
ØØ8Ø6A 1C8D BB 5B             A         ADD      CHKSUM    CHECKSUM
ØØ8Ø7A 1C8F B7 5B             A         STA      CHKSUM
ØØ8Ø8A 1C91 81                          RTS
ØØ8Ø9                           *
ØØ81ØA 1C92 A6 3F             A PUNLDR  LDA      #$3F      PUNCH 16K
ØØ811A 1C94 B7 5Ø             A         STA      WORK1     ZEROS
ØØ812A 1C96 A6 FF             A         LDA      #$FF
ØØ813A 1C98 B7 51             A         STA      WORK2
ØØ814A 1C9A AD 24        1CCØ   PUNLD1  BSR      COMO
ØØ815A 1C9C AD 35        1CD3           BSR      NOCO
ØØ816A 1C9E 3A 51             A         DEC      WORK2
ØØ817A 1CAØ 26 F8        1C9A           BNE      PUNLD1
ØØ818A 1CA2 3A 5Ø             A         DEC      WORK1
ØØ819A 1CA4 26 F4        1C9A           BNE      PUNLD1
ØØ82ØA 1CA6 81                          RTS
ØØ821                           *
ØØ822A 1CA7 AE Ø8             A PUNBYT  LDX      #8        PUNCH
ØØ823A 1CA9 AD 15        1CCØ           BSR      COMO      SYNC
ØØ824A 1CAB AD 13        1CCØ           BSR      COMO      START
ØØ825A 1CAD AD 11        1CCØ   PUNBY1  BSR      COMO      SYNC
ØØ826A 1CAF 46                          RORA
ØØ827A 1CBØ 24 Ø4        1CB6           BCC      PUNBY2    1 OR Ø?
ØØ828A 1CB2 AD ØC        1CCØ           BSR      COMO      1
ØØ829A 1CB4 2Ø Ø2        1CB8           BRA      PUNBY3
ØØ83ØA 1CB6 AD 1B        1CD3   PUNBY2  BSR      NOCO      Ø
ØØ831A 1CB8 5A                  PUNBY3  DECX               ALL
ØØ832A 1CB9 26 F2        1CAD           BNE      PUNBY1    DONE?
ØØ833A 1CBB AD Ø3        1CCØ           BSR      COMO      YES,SYNC
ØØ834A 1CBD AD 14        1CD3           BSR      NOCO      STOP BIT
ØØ835A 1CBF 81                          RTS
ØØ836                           *
ØØ837A 1CCØ BF 54             A COMO    STX      WORK3     MAKE A TRANSITION
ØØ838A 1CC2 ØD ØØ Ø4     1CC9           BRCLR    6,PORTA,COMO1
ØØ839A 1CC5 1D ØØ             A         BCLR     6,PORTA
ØØ84ØA 1CC7 2Ø Ø2        1CCB           BRA      DELAY     PAUSE
ØØ841A 1CC9 1C ØØ             A COMO1   BSET     6,PORTA
ØØ842A 1CCB BE 5E             A DELAY   LDX      PCNT1
ØØ843A 1CCD 5A                  COMO2   DECX
ØØ844A 1CCE 26 FD        1CCD           BNE      COMO2
ØØ845A 1CDØ BE 54             A         LDX      WORK3
ØØ846A 1CD2 81                          RTS
ØØ847                           *
ØØ848A 1CD3 BF 54             A NOCO    STX      WORK3     NO TRANSITION
ØØ849A 1CD5 BE 5F             A         LDX      PCNTØ     DOUBLE DELAY
ØØ85ØA 1CD7 5A                  NOCO1   DECX
ØØ851A 1CD8 26 FD        1CD7           BNE      NOCO1
ØØ852A 1CDA BE 54             A         LDX      WORK3
ØØ853A 1CDC 81                          RTS
ØØ854                           *
ØØ855                           ***********************************************
ØØ856                           *                                             *
ØØ857                           *                LOAD TAPE OR                 *
ØØ858                           *                COMPARE TAPE                 *
ØØ859                           *                *
ØØ86Ø                           ***********************************************
ØØ861                           *
ØØ862              1CDD       A TLOAD   EQU      *
ØØ863A 1CDD 1B 4F             A         BCLR     5,SWIFLG
```

```
PAGE  Ø17  CBUGØ5  .SA:1

ØØ864A 1CDF CD 1EØ7      A         JSR    CLRDIS
ØØ865A 1CE2 A6 FF        A LOADØ   LDA    #$FF      LOAD 256
ØØ866A 1CE4 AD 78     1D5E LOAD1   BSR    EDGE      CONSECUTIVE
ØØ867A 1CE6 25 FA     1CE2         BCS    LOADØ     ZEROS
ØØ868A 1CE8 4A                     DECA
ØØ869A 1CE9 26 F9     1CE4         BNE    LOAD1
ØØ87ØA 1CEB AD 5Ø     1D3D LOAD2   BSR    LOADBY
ØØ871A 1CED A1 B3        A         CMP    #$B3      BOT?
ØØ872A 1CEF 26 FA     1CEB         BNE    LOAD2
ØØ873                      *
ØØ874A 1CF1 3F 5B        A         CLR    CHKSUM    YES, INIT CHECKSUM
ØØ875A 1CF3 CD 1D76      A         JSR    LOADIT    GET ENDING
ØØ876A 1CF6 B7 58        A         STA    TEMP      ADDRESS
ØØ877A 1CF8 AD 7C     1D76         BSR    LOADIT
ØØ878A 1CFA B7 59        A         STA    TEMP+1
ØØ879A 1CFC AD 78     1D76         BSR    LOADIT    GET BEGINNING
ØØ88ØA 1CFE B7 52        A         STA    ADDRH     ADDRESS
ØØ881A 1DØØ AD 74     1D76         BSR    LOADIT
ØØ882A 1DØ2 B7 53        A         STA    ADDRL
ØØ883                      *
ØØ884A 1DØ4 AD 7Ø     1D76 LOAD4   BSR    LOADIT    GET BYTE
ØØ885A 1DØ6 ØB 4F ØB  1D14         BRCLR  5,SWIFLG,LOAD5 COMPARE?
ØØ886A 1DØ9 B7 57        A         STA    WORK6     YES, IS IT
ØØ887A 1DØB CD 1F15      A         JSR    LOAD      SAME?
ØØ888A 1DØE B1 57        A         CMP    WORK6
ØØ889A 1D1Ø 26 25     1D37         BNE    DISADR    NO
ØØ89ØA 1D12 2Ø Ø5     1D19         BRA    LOAD6     YES
ØØ891A 1D14 CD 1F24      A LOAD5   JSR    STORE     NOT COMPARE, SAVE IT
ØØ892A 1D17 25 1E     1D37         BCS    DISADR
ØØ893A 1D19 3C 53        A LOAD6   INC    ADDRL     INC ADDRESS
ØØ894A 1D1B 26 Ø2     1D1F         BNE    LOAD3
ØØ895A 1D1D 3C 52        A         INC    ADDRH
ØØ896A 1D1F B6 58        A LOAD3   LDA    TEMP      FINSHED?
ØØ897A 1D21 B1 52        A         CMP    ADDRH
ØØ898A 1D23 26 DF     1DØ4         BNE    LOAD4
ØØ899A 1D25 B6 59        A         LDA    TEMP+1
ØØ9ØØA 1D27 B1 53        A         CMP    ADDRL
ØØ9Ø1A 1D29 26 D9     1DØ4         BNE    LOAD4
ØØ9Ø2A 1D2B AD 1Ø     1D3D         BSR    LOADBY    YES ,GET
ØØ9Ø3A 1D2D B1 5B        A         CMP    CHKSUM    CHECKSUM
ØØ9Ø4A 1D2F 26 Ø3     1D34         BNE    LDERR     NOT SAME -- ERROR
ØØ9Ø5A 1D31 CC 18AB      A         JMP    GETCMD
ØØ9Ø6                      *
ØØ9Ø7A 1D34 CC 1E97      A LDERR   JMP    ERROR
ØØ9Ø8                      *
ØØ9Ø9A 1D37 CD 1FBØ      A DISADR  JSR    PRTADR    DISPLAY ADDRESS
ØØ91ØA 1D3A CC 18B5      A         JMP    CMDSCN    FOR ERROR
ØØ911                      *
ØØ912A 1D3D BF 5Ø        A LOADBY  STX    WORK1
ØØ913A 1D3F AE Ø8        A         LDX    #8
ØØ914A 1D41 AD 1B     1D5E         BSR    EDGE      SET START
ØØ915A 1D43 AD 19     1D5E LODBY1  BSR    EDGE      BIT
ØØ916A 1D45 24 FC     1D43         BCC    LODBY1
ØØ917A 1D47 AD 15     1D5E         BSR    EDGE      SYNC
ØØ918A 1D49 5A             LODBY2  DECX
ØØ919A 1D4A 2B ØF     1D5B         BMI    LODBYR    FINISHED?
ØØ92ØA 1D4C 44                     LSRA             NO, SHIFT
ØØ921A 1D4D AD ØF     1D5E         BSR    EDGE      GET BIT
```

```
ØØ922A 1D4F 24 Ø6    1D57        BCC    LODBY3   1 OR Ø?
ØØ923A 1D51 AD ØB    1D5E        BSR    EDGE     IF 1 GET CLEAR NEXT
ØØ924A 1D53 AA 8Ø       A        ORA    #$8Ø     TRANSITION
ØØ925A 1D55 2Ø F2    1D49        BRA    LODBY2   SHIFT IN 1
ØØ926A 1D57 AA ØØ       A LODBY3 ORA    #Ø       SHIFT IN Ø
ØØ927A 1D59 2Ø EE    1D49        BRA    LODBY2
ØØ928A 1D5B BE 5Ø       A LODBYR LDX    WORK1
ØØ929A 1D5D 81                   RTS
ØØ93Ø                     *
ØØ931A 1D5E B7 51       A EDGE   STA    WORK2
ØØ932A 1D6Ø BF 54       A        STX    WORK3
ØØ933A 1D62 5F                   CLRX
ØØ934A 1D63 5C           EDGE1  INCX            LOOP TILL
ØØ935A 1D64 4F                   CLRA            TRANSITION
ØØ936A 1D65 2E Ø1    1D68        BIL    EDGE2
ØØ937A 1D67 4C                   INCA
ØØ938A 1D68 B1 5C       A EDGE2  CMP    SREF
ØØ939A 1D6A 27 F7    1D63        BEQ    EDGE1
ØØ94ØA 1D6C B7 5C       A        STA    SREF     UPDATE LEVEL
ØØ941A 1D6E 9F                   TXA             STATUS
ØØ942A 1D6F BØ 5D       A        SUB    LCNT     SET CARRY FOR
ØØ943A 1D71 B6 51       A        LDA    WORK2    1 OR Ø
ØØ944A 1D73 BE 54       A        LDX    WORK3
ØØ945A 1D75 81                   RTS
ØØ946                     *
ØØ947A 1D76 AD C5    1D3D LOADIT BSR    LOADBY   GET BYTE
ØØ948A 1D78 B7 55       A        STA    WORK4    AND UPDATE
ØØ949A 1D7A BB 5B       A        ADD    CHKSUM   CHECKSUM
ØØ95ØA 1D7C B7 5B       A        STA    CHKSUM
ØØ951A 1D7E B6 55       A        LDA    WORK4
ØØ952A 1D8Ø 81                   RTS
ØØ953                     *
ØØ954                     ****************************************
ØØ955                     *                                      *
ØØ956                     *             VERIFY TAPE              *
ØØ957                     *                                      *
ØØ958                     ****************************************
ØØ959                     *
ØØ96Ø          1D81     A VERIFY EQU    *
ØØ961A 1D81 1A 4F       A        BSET   5,SWIFLG
ØØ962A 1D83 CD 1EØ7     A        JSR    CLRDIS
ØØ963A 1D86 CC 1CE2     A        JMP    LOADØ
ØØ964                     *
ØØ965A 1D89 CC 1E97     A GOERR  JMP    ERROR
ØØ966                     *
ØØ967A 1D8C CC 18AB     A GOBACK JMP    GETCMD
ØØ968                     *
ØØ969          1D8F     A GO     EQU    *
ØØ97ØA 1D8F CD 1916     A        JSR    LOCSTK
ØØ971A 1D92 E6 Ø8       A        LDA    8,X
ØØ972A 1D94 B7 53       A        STA    ADDRL
ØØ973A 1D96 E6 Ø7       A        LDA    7,X
ØØ974A 1D98 B7 52       A        STA    ADDRH
ØØ975A 1D9A CD 1F53     A        JSR    GETADR
ØØ976A 1D9D 25 Ø8    1DA7        BCS    GOON     ADDR VALID?
ØØ977A 1D9F A1 1Ø       A        CMP    #$1Ø
ØØ978A 1DA1 27 E9    1D8C        BEQ    GOBACK
ØØ979A 1DA3 A1 11       A        CMP    #$11
```

```
ØØ98ØA 1DA5 26 E2   1D89        BNE   GOERR
ØØ981A 1DA7 CD 1916    A GOON   JSR   LOCSTK  YES PUT IT
ØØ982A 1DAA B6 52      A        LDA   ADDRH   IN STACK
ØØ983A 1DAC A1 1F      A        CMP   #PCMASK TO BIG?
ØØ984A 1DAE 22 D9   1D89        BHI   GOERR   YES
ØØ985A 1DBØ E7 Ø7      A        STA   7,X
ØØ986A 1DB2 B6 53      A        LDA   ADDRL
ØØ987A 1DB4 E7 Ø8      A        STA   8,X
ØØ988A 1DB6 AD 1B   1DD3 CONT   BSR   SCNBKP  FIND B.P. TABLE
ØØ989A 1DB8 F6           GOINSB LDA   Ø,X     INSERTPB.P.'S
ØØ99ØA 1DB9 2B 1Ø   1DCB        BMI   GONOB   VALID?
ØØ991A 1DBB B7 52      A        STA   ADDRH   YES
ØØ992A 1DBD E6 Ø1      A        LDA   1,X
ØØ993A 1DBF B7 53      A        STA   ADDRL
ØØ994A 1DC1 CD 1F15    A        JSR   LOAD    SAVE OPCODE
ØØ995A 1DC4 E7 Ø2      A        STA   2,X
ØØ996A 1DC6 A6 83      A        LDA   #SWIOP
ØØ997A 1DC8 CD 1F24    A        JSR   STORE
ØØ998A 1DCB 5C           GONOB  INCX          GET NEXT B.P.
ØØ999A 1DCC 5C                  INCX
Ø1ØØØA 1DCD 5C                  INCX
Ø1ØØ1A 1DCE 3A 5A      A        DEC   PNCNT
Ø1ØØ2A 1DDØ 26 E6   1DB8        BNE   GOINSB  DONE?
Ø1ØØ3A 1DD2 8Ø                  RTI           YES
Ø1ØØ4                    *
Ø1ØØ5          1DD3    A SCNBKP EQU   *
Ø1ØØ6A 1DD3 A6 Ø3      A        LDA   #NUMBKP
Ø1ØØ7A 1DD5 B7 5A      A        STA   PNCNT
Ø1ØØ8A 1DD7 AE 37      A        LDX   #BKPTBL
Ø1ØØ9A 1DD9 81                  RTS
Ø1Ø1Ø                    *
Ø1Ø11                    **************************************
Ø1Ø12                    *                                    *
Ø1Ø13                    *      DISPLAY STACK POINTER         *
Ø1Ø14                    *                                    *
Ø1Ø15                    **************************************
Ø1Ø16                    *
Ø1Ø17          1DDA    A STACK  EQU   *
Ø1Ø18A 1DDA A6 B5      A        LDA   #$B5    PRINT
Ø1Ø19A 1DDC B7 4D      A        STA   DTABL+4 'SP'
Ø1Ø2ØA 1DDE A6 73      A        LDA   #$73
Ø1Ø21A 1DEØ B7 4E      A        STA   DTABL+5
Ø1Ø22A 1DE2 4F                  CLRA
Ø1Ø23A 1DE3 5F                  CLRX
Ø1Ø24A 1DE4 CD 1F8E    A        JSR   PRTBYT
Ø1Ø25A 1DE7 CD 1916    A        JSR   LOCSTK  FIND USER
Ø1Ø26A 1DEA 9F                  TXA           STACK POINTER
Ø1Ø27A 1DEB AB Ø3      A        ADD   #3
Ø1Ø28A 1DED AE Ø2      A        LDX   #2
Ø1Ø29A 1DEF CD 1F8E    A        JSR   PRTBYT  PRINT IT
Ø1Ø3ØA 1DF2 CC 18B5    A        JMP   CMDSCN
Ø1Ø31                    *
```

```
Ø1Ø33                  *
Ø1Ø34                  **********************************
Ø1Ø35                  *                                *
Ø1Ø36                  *      CLEAR DISPLAY TABLE       *
Ø1Ø37                  *                                *
Ø1Ø38                  *      X REG DESTROYED           *
Ø1Ø39                  *                                *
Ø1Ø4Ø                  **********************************
Ø1Ø41                  *
Ø1Ø42A 1DF5 AE Ø5    A CLRTAB LDX   #5
Ø1Ø43A 1DF7 6F 49    A CLRLOC CLR   DTABL,X  CLEAR SIX
Ø1Ø44A 1DF9 5A                DECX           LOCATIONS IN
Ø1Ø45A 1DFA 2A FB 1DF7        BPL   CLRLOC   DISPLAY TABLE
Ø1Ø46A 1DFC 81                RTS
Ø1Ø47                  *
Ø1Ø48                  **********************************
Ø1Ø49                  *                                *
Ø1Ø5Ø                  *      DISPLAY TABLE CONTENTS    *
Ø1Ø51                  *                                *
Ø1Ø52                  *      A,X REGISTERS DESTROYED   *
Ø1Ø53                  *                                *
Ø1Ø54                  **********************************
Ø1Ø55                  *
Ø1Ø56A 1DFD AE Ø5    A DISTAB LDX   #5
Ø1Ø57A 1DFF E6 49    A DISCHR LDA   DTABL,X  LOAD DISPLAY
Ø1Ø58A 1EØ1 AD Ø9 1EØC        BSR   DISPLY   TABLE INTO
Ø1Ø59A 1EØ3 5A                DECX           145ØØØ
Ø1Ø6ØA 1EØ4 2A F9 1DFF        BPL   DISCHR
Ø1Ø61A 1EØ6 81                RTS
Ø1Ø62                  *
Ø1Ø63                  **********************************
Ø1Ø64                  *                                *
Ø1Ø65                  *      BLANK DISPLAY             *
Ø1Ø66                  *                                *
Ø1Ø67                  *      A,X REGISTERS DESTROYED   *
Ø1Ø68                  *                                *
Ø1Ø69                  **********************************
Ø1Ø7Ø                  *
Ø1Ø71A 1EØ7 AD EC 1DF5 CLRDIS BSR   CLRTAB   BLANK
Ø1Ø72A 1EØ9 AD F2 1DFD        BSR   DISTAB   DISPLAY
Ø1Ø73A 1EØB 81                RTS
Ø1Ø74                  *
Ø1Ø75                  **********************************
Ø1Ø76                  *                                *
Ø1Ø77                  *      SHIFT ONE CHARACTER INTO  *
Ø1Ø78                  *            DISPLAY             *
Ø1Ø79                  *                                *
Ø1Ø8Ø                  *      A REGISTER DESTROYED      *
Ø1Ø81                  *                                *
Ø1Ø82                  **********************************
Ø1Ø83                  *
Ø1Ø84A 1EØC BF 5Ø    A DISPLY STX   WORK1    SAVE INDEX
Ø1Ø85A 1EØE 1D ØØ    A        BCLR  6,PORTA  CLEAR DATA
Ø1Ø86A 1E1Ø AE Ø8    A        LDX   #8
Ø1Ø87A 1E12 48         DIS1   LSLA           SET UP
Ø1Ø88A 1E13 24 Ø2 1E17        BCC   DIS2     BIT OF
Ø1Ø89A 1E15 1C ØØ    A        BSET  6,PORTA  ACCUMULATOR
Ø1Ø9ØA 1E17 1E ØØ    A DIS2   BSET  7,PORTA  CLOCK
```

```
Ø1091A 1E19 1F ØØ       A        BCLR   7,PORTA IT
Ø1092A 1E1B 1D ØØ       A        BCLR   6,PORTA CLEAR DATA
Ø1093A 1E1D 5A                   DECX           COMPLETE?
Ø1094A 1E1E 26 F2    1E12        BNE    DIS1    NO
Ø1095A 1E2Ø BE 5Ø       A        LDX    WORK1   RESTORE INDEX
Ø1096A 1E22 81                   RTS
Ø1097                     *
Ø1098                     ***************************************
Ø1099                     *                                     *
Ø11ØØ                     *      KEYPAD SCAN                    *
Ø11Ø1                     *                                     *
Ø11Ø2                     *      X REGISTER DESTROYED           *
Ø11Ø3                     *                                     *
Ø11Ø4                     *      A REGISTER CONTAINS VALUE      *
Ø11Ø5                     *                                     *
Ø11Ø6                     *      CARRY SET IF VALID OUTPUT      *
Ø11Ø7                     *                                     *
Ø11Ø8                     ***************************************
Ø11Ø9                     *
Ø111ØA 1E23 98            KEYSCN CLC
Ø1111A 1E24 4F                   CLRA
Ø1112A 1E25 AE Ø6       A        LDX    #6      SETUP
Ø1113A 1E27 AB 1Ø       A KEY1   ADD    #$1Ø    ROW
Ø1114A 1E29 B7 ØØ       A        STA    PORTA
Ø1115A 1E2B AD Ø6    1E33        BSR    COLUMN  CHECK COLUMNS
Ø1116A 1E2D 25 Ø3    1E32        BCS    KEY2    IF VALID GET OUT
Ø1117A 1E2F 5A                   DECX           ELSE TRY
Ø1118A 1E3Ø 26 F5    1E27        BNE    KEY1    NEXT ROW
Ø1119A 1E32 81            KEY2   RTS
Ø112Ø                     *
Ø1121                     ***************************************
Ø1122                     *                                     *
Ø1123                     *      CHECK FOR KEY CLOSURE          *
Ø1124                     *      WITHIN COLUMN AND DEBOUNCE     *
Ø1125                     *                                     *
Ø1126                     *      A REGISTER CONTAINS VALUE      *
Ø1127                     *                                     *
Ø1128                     *      CARRY SET IF VALID OUTPUT      *
Ø1129                     *                                     *
Ø113Ø                     ***************************************
Ø1131                     *
Ø1132A 1E33 B6 ØØ       A COLUMN LDA    PORTA   READ KEYPAD
Ø1133A 1E35 B7 5Ø       A        STA    WORK1   STORE IT
Ø1134A 1E37 A5 ØF       A        BIT    #$ØF    KEY CLOSED?
Ø1135A 1E39 27 19    1E54        BEQ    COLRET  NO GET OUT
Ø1136A 1E3B AD 18    1E55        BSR    DBOUNC  ELSE DEBOUNCE
Ø1137A 1E3D B6 ØØ       A        LDA    PORTA   RE-READ KEYPAD
Ø1138A 1E3F B1 5Ø       A        CMP    WORK1   SAME KEY CLOSED?
Ø1139A 1E41 26 11    1E54        BNE    COLRET  NO GET OUT
Ø114ØA 1E43 99                   SEC            SET FLAG FOR VALID
Ø1141A 1E44 B6 ØØ       A COL1   LDA    PORTA   KEY
Ø1142A 1E46 A5 ØF       A        BIT    #$ØF    RELEASED?
Ø1143A 1E48 26 FA    1E44        BNE    COL1    NO TRY AGAIN
Ø1144A 1E4A AD Ø9    1E55        BSR    DBOUNC  YES DEBOUNCE
Ø1145A 1E4C B6 ØØ       A        LDA    PORTA   STILL
Ø1146A 1E4E A5 ØF       A        BIT    #$ØF    RELEASED?
Ø1147A 1E5Ø 26 F2    1E44        BNE    COL1    NO TRY AGAIN
Ø1148A 1E52 B6 5Ø       A        LDA    WORK1   RETURN CHAR IN A-REG
```

```
PAGE  Ø22  CBUGØ5  .SA:1

Ø1149A 1E54 81            COLRET RTS           YES GO HOME
Ø115Ø                     *
Ø1151                     ****************************************
Ø1152                     *                                      *
Ø1153                     *      PAUSE FOR 3Ø75 CYCLES           *
Ø1154                     *                                      *
Ø1155                     *      A REGISTER DESTROYED            *
Ø1156                     *                                      *
Ø1157                     ****************************************
Ø1158                     *
Ø1159A 1E55 A6 FF       A DBOUNC LDA    #$FF     PAUSE
Ø116ØA 1E57 21 FE    1E57 DLOOP  BRN    *        256X12
Ø1161A 1E59 21 FE    1E59        BRN    *        CYCLES
Ø1162A 1E5B 4A                   DECA            OR AT
Ø1163A 1E5C 26 F9    1E57        BNE    DLOOP    LEAST
Ø1164A 1E5E 81                   RTS             3.7 MS
Ø1165                     *
```

```
Ø1167                          *
Ø1168                          ****************************************
Ø1169                          *                                      *
Ø117Ø                          *      INPUT ONE CHARACTER             *
Ø1171                          *                                      *
Ø1172                          *      A REGISTER CONTAINS HEX VALUE   *
Ø1173                          *                                      *
Ø1174                          *      X REGISTER CONTAINS HEX VALUE   *
Ø1175                          *                                      *
Ø1176                          ****************************************
Ø1177                          *
Ø1178          1E5F      A CHRIN  EQU    *
Ø1179A 1E5F CD 1E23      A        JSR    KEYSCN    GET KEY
Ø118ØA 1E62 24 FB     1E5F        BCC    CHRIN     IF NOT VALID RETRY
Ø1181A 1E64 5F                    CLRX
Ø1182A 1E65 D1 1E6F      A CHRIN1 CMP    STABL,X   CONVERT
Ø1183A 1E68 27 Ø3     1E6D        BEQ    CHRIN2    TO HEX
Ø1184A 1E6A 5C                    INCX
Ø1185A 1E6B 2Ø F8     1E65        BRA    CHRIN1
Ø1186A 1E6D 9F             CHRIN2 TXA              IF CANCEL
Ø1187A 1E6E 81                    RTS
Ø1188                          *
Ø1189                          ****************************************
Ø119Ø                          *                                      *
Ø1191                          *      CONVERSION TABLE FOR KEYPAD     *
Ø1192                          *      TO HEX NUMBER                   *
Ø1193                          *                                      *
Ø1194                          ****************************************
Ø1195                          *
Ø1196A 1E6F     11       A STABL  FCB    $11       Ø
Ø1197A 1E7Ø     21       A        FCB    $21       1
Ø1198A 1E71     22       A        FCB    $22       2
Ø1199A 1E72     24       A        FCB    $24       3
Ø12ØØA 1E73     31       A        FCB    $31       4
Ø12Ø1A 1E74     32       A        FCB    $32       5
Ø12Ø2A 1E75     34       A        FCB    $34       6
Ø12Ø3A 1E76     41       A        FCB    $41       7
Ø12Ø4A 1E77     42       A        FCB    $42       8
Ø12Ø5A 1E78     44       A        FCB    $44       9
Ø12Ø6A 1E79     48       A        FCB    $48       A
Ø12Ø7A 1E7A     38       A        FCB    $38       B
Ø12Ø8A 1E7B     28       A        FCB    $28       C
Ø12Ø9A 1E7C     18       A        FCB    $18       D
Ø121ØA 1E7D     14       A        FCB    $14       E
Ø1211A 1E7E     12       A        FCB    $12       F
Ø1212A 1E7F     61       A        FCB    $61       CANCEL COMMAND
Ø1213A 1E8Ø     58       A        FCB    $58       ENTER COMMAND
Ø1214A 1E81     68       A        FCB    $68       STACK POINTER
Ø1215A 1E82     64       A        FCB    $64       MEMORY
Ø1216A 1E83     62       A        FCB    $62       GO
Ø1217A 1E84     54       A        FCB    $54       VERIFY TAPE
Ø1218A 1E85     52       A        FCB    $52       LOAD TAPE
Ø1219A 1E86     51       A        FCB    $51       PUNCH TAPE
Ø122Ø                          *
Ø1221                          ****************************************
Ø1222                          *                                      *
Ø1223                          *      HEX TO MUX DISPLAY              *
Ø1224                          *      CONVERSION TABLE                *
```

```
Ø1225                        *                                      *
Ø1226                        ****************************************
Ø1227                        *
Ø1228A 1E87     D7       A CTABL  FCB    $D7        Ø
Ø1229A 1E88     Ø6       A        FCB    6          1
Ø123ØA 1E89     E3       A        FCB    $E3        2
Ø1231A 1E8A     A7       A        FCB    $A7        3
Ø1232A 1E8B     36       A        FCB    $36        4
Ø1233A 1E8C     B5       A        FCB    $B5        5
Ø1234A 1E8D     F5       A        FCB    $F5        6
Ø1235A 1E8E     Ø7       A        FCB    7          7
Ø1236A 1E8F     F7       A        FCB    $F7        8
Ø1237A 1E9Ø     B7       A        FCB    $B7        9
Ø1238A 1E91     77       A        FCB    $77        A
Ø1239A 1E92     F4       A        FCB    $F4        B
Ø124ØA 1E93     D1       A        FCB    $D1        C
Ø1241A 1E94     E6       A        FCB    $E6        D
Ø1242A 1E95     F1       A        FCB    $F1        E
Ø1243A 1E96     71       A        FCB    $71        F
Ø1244                        *
Ø1245           1E97     A ERROR  EQU    *
Ø1246A 1E97 CD 1DF5      A        JSR    CLRTAB
Ø1247A 1E9A A6 F1        A        LDA    #$F1
Ø1248A 1E9C B7 4A        A        STA    DTABL+1
Ø1249A 1E9E A6 6Ø        A        LDA    #$6Ø
Ø125ØA 1EAØ B7 4B        A        STA    DTABL+2
Ø1251A 1EA2 B7 4C        A        STA    DTABL+3
Ø1252A 1EA4 CD 1DFD      A        JSR    DISTAB
Ø1253A 1EA7 CC 18B5      A        JMP    CMDSCN
```

```
Ø1255                     *
Ø1256                     ****************************************
Ø1257                     *                                      *
Ø1258                     *      MEMORY EXAMINE/CHANGE           *
Ø1259                     *                                      *
Ø126Ø                     ****************************************
Ø1261                     *
Ø1262A 1EAA CD 1F53     A MEMEX  JSR   GETADR   BUILD ADDRESS
Ø1263A 1EAD A1 1Ø       A        CMP   #$1Ø
Ø1264A 1EAF 27 5F    1F1Ø        BEQ   MEMEX4
Ø1265A 1EB1 B7 5Ø       A MEMEX3 STA   WORK1
Ø1266A 1EB3 B6 52       A        LDA   ADDRH
Ø1267A 1EB5 A1 1F       A        CMP   #PCMASK
Ø1268A 1EB7 23 Ø3    1EBC        BLS   MEMOK
Ø1269A 1EB9 CC 1E97     A        JMP   ERROR
Ø127ØA 1EBC B6 5Ø       A MEMOK  LDA   WORK1
Ø1271A 1EBE CD 1F15     A        JSR   LOAD     LOAD DATA
Ø1272A 1EC1 CD 1F8C     A        JSR   PRTDAT   PRINT IT
Ø1273A 1EC4 CD 1F49     A        JSR   GETNYB   GET NEW NIBBLE
Ø1274A 1EC7 A1 1Ø       A        CMP   #$1Ø
Ø1275A 1EC9 27 45    1F1Ø        BEQ   MEMEX4
Ø1276A 1ECB A1 11       A        CMP   #$11
Ø1277A 1ECD 27 19    1EE8        BEQ   ADRINC
Ø1278A 1ECF A1 13       A        CMP   #$13
Ø1279A 1ED1 27 2D    1FØØ        BEQ   ADRDEC
Ø128ØA 1ED3 24 Ø8    1EDD        BCC   CMDMDL   IF VALID
Ø1281A 1ED5 CD 1F8C     A MEMEX1 JSR   PRTDAT   PRINT IT
Ø1282A 1ED8 CD 1F37     A        JSR   GETBY2   SHIFT IN NEXT
Ø1283A 1EDB 25 F8    1ED5        BCS   MEMEX1   IF VALID TRY AGAIN
Ø1284                     *
Ø1285A 1EDD A1 11       A CMDMDL CMP   #$11     ENTER?
Ø1286A 1EDF 26 15    1EF6        BNE   MEMEX2   NO
Ø1287A 1EE1 B6 51       A        LDA   WORK2    RESTORE ACCA
Ø1288A 1EE3 CD 1F24     A        JSR   STORE    YES STORE IT
Ø1289A 1EE6 25 C9    1EB1        BCS   MEMEX3   STORE VALID?
Ø129ØA 1EE8 ØC 4F 25 1F1Ø ADRINC BRSET 6,SWIFLG,MEMEX4
Ø1291A 1EEB 3C 53       A        INC   ADDRL    YES GOTTO
Ø1292A 1EED 26 Ø2    1EF1        BNE   MEMEX5   NEXT
Ø1293A 1EEF 3C 52       A        INC   ADDRH
Ø1294A 1EF1 CD 1FBØ     A MEMEX5 JSR   PRTADR   PRINT IT
Ø1295A 1EF4 2Ø BB    1EB1        BRA   MEMEX3   REPEAT
Ø1296A 1EF6 A1 13       A MEMEX2 CMP   #$13     MEMORY?
Ø1297A 1EF8 26 16    1F1Ø        BNE   MEMEX4   NO
Ø1298A 1EFA B6 51       A        LDA   WORK2
Ø1299A 1EFC AD 26    1F24        BSR   STORE
Ø13ØØA 1EFE 25 B1    1EB1        BCS   MEMEX3
Ø13Ø1A 1FØØ ØC 4F ØD 1F1Ø ADRDEC BRSET 6,SWIFLG,MEMEX4
Ø13Ø2A 1FØ3 3D 53       A        TST   ADDRL    YES THEN
Ø13Ø3A 1FØ5 26 Ø2    1FØ9        BNE   CMDMB2   GET PREVIOUS
Ø13Ø4A 1FØ7 3A 52       A        DEC   ADDRH    ADDRESS
Ø13Ø5A 1FØ9 3A 53       A CMDMB2 DEC   ADDRL
Ø13Ø6A 1FØB CD 1FBØ     A        JSR   PRTADR   PRINT IT
Ø13Ø7A 1FØE 2Ø A1    1EB1        BRA   MEMEX3   REPEAT
Ø13Ø8A 1F1Ø 1D 4F       A MEMEX4 BCLR  6,SWIFLG INVALID CHAR
Ø13Ø9A 1F12 CC 18AB     A        JMP   GETCMD
Ø131Ø                     *
Ø1311                     ****************************************
Ø1312                     *                                      *
```

```
PAGE  Ø26  CBUGØ5  .SA:1

Ø1313                     *        LOAD BYTE AT ADDRH,ADDRL          *
Ø1314                     *        INTO ACCUMULATOR                  *
Ø1315                     *                                          *
Ø1316                     ****************************************
Ø1317                     *
Ø1318A 1F15 BF 5Ø       A LOAD     STX     WORK1      SETUP
Ø1319A 1F17 AE C6       A          LDX     #$C6       ROUTINE
Ø132ØA 1F19 BF 51       A LDSTCM   STX     WORK2      TO DO
Ø1321A 1F1B AE 81       A          LDX     #$81       TWO BYTE
Ø1322A 1F1D BF 54       A          STX     WORK3      LOAD
Ø1323A 1F1F BD 51       A          JSR     WORK2
Ø1324A 1F21 BE 5Ø       A          LDX     WORK1
Ø1325A 1F23 81                     RTS
Ø1326                     *
Ø1327                     ****************************************
Ø1328                     *                                          *
Ø1329                     *        STORE ACCUMULATOR INTO            *
Ø133Ø                     *        BYTE AT ADDRH,ADDRL               *
Ø1331                     *                                          *
Ø1332                     ****************************************
Ø1333                     *
Ø1334A 1F24 BF 5Ø       A STORE    STX     WORK1
Ø1335A 1F26 AE C7       A          LDX     #$C7       SETUP
Ø1336A 1F28 AD EF  1F19            BSR     LDSTCM     ROUTINE
Ø1337A 1F2A B7 55       A          STA     WORK4      TO DO
Ø1338A 1F2C CD 1F15     A          JSR     LOAD       TWO BYTE
Ø1339A 1F2F B1 55       A          CMP     WORK4      STORE
Ø134ØA 1F31 27 Ø1  1F34            BEQ     STRTS
Ø1341A 1F33 99                     SEC
Ø1342A 1F34 BE 5Ø       A STRTS    LDX     WORK1
Ø1343A 1F36 81                     RTS
Ø1344                     *
```

```
Ø1346                     *
Ø1347                     *****************************************
Ø1348                     *                                       *
Ø1349                     *      BUILD A BYTE                     *
Ø135Ø                     *                                       *
Ø1351                     *      A REGISTER CONTAINS BYTE         *
Ø1352                     *                                       *
Ø1353                     *****************************************
Ø1354                     *
Ø1355A 1F37 B7 51       A GETBY2 STA    WORK2
Ø1356A 1F39 AD ØE    1F49        BSR    GETNYB
Ø1357A 1F3B 24 ØB    1F48        BCC    GETBRT
Ø1358A 1F3D 38 51       A        ASL    WORK2
Ø1359A 1F3F 38 51       A        ASL    WORK2
Ø136ØA 1F41 38 51       A        ASL    WORK2
Ø1361A 1F43 38 51       A        ASL    WORK2
Ø1362A 1F45 BA 51       A        ORA    WORK2
Ø1363A 1F47 99                   SEC
Ø1364A 1F48 81             GETBRT RTS
Ø1365                     *
Ø1366                     *****************************************
Ø1367                     *                                       *
Ø1368                     *      GET ONE CHARACTER AND            *
Ø1369                     *      CHECK FOR VALID HEX NUMBER       *
Ø137Ø                     *                                       *
Ø1371                     *      A REGISTER CONTAINS OUTPUT       *
Ø1372                     *                                       *
Ø1373                     *      X REGISTER DESTROYED             *
Ø1374                     *                                       *
Ø1375                     *      CARRY SET IF VALID HEX NUMBER    *
Ø1376                     *                                       *
Ø1377                     *****************************************
Ø1378                     *
Ø1379A 1F49 CD 1E5F     A GETNYB JSR    CHRIN    GET CHARACTER
Ø138ØA 1F4C 98                   CLC
Ø1381A 1F4D A1 ØF       A        CMP    #$ØF     VALID HEX?
Ø1382A 1F4F 22 Ø1    1F52        BHI    GETRET   NO
Ø1383A 1F51 99                   SEC             YES
Ø1384A 1F52 81             GETRET RTS
Ø1385                     *
Ø1386                     *****************************************
Ø1387                     *                                       *
Ø1388                     *      BUILD ADDRESS                    *
Ø1389                     *                                       *
Ø139Ø                     *      A,X REGISTERS DESTROYED          *
Ø1391                     *                                       *
Ø1392                     *      ADDRH,ADDRL CONTAIN ADDRESS      *
Ø1393                     *                                       *
Ø1394                     *      CARRY SET IF NEW ADDRESS         *
Ø1395                     *                                       *
Ø1396                     *****************************************
Ø1397                     *
Ø1398A 1F53 CD 1DF5     A GETADR JSR    CLRTAB   BLANK DISPLAY
Ø1399A 1F56 AD 58    1FBØ        BSR    PRTADR
Ø14ØØA 1F58 AD EF    1F49 BLDADR BSR    GETNYB   GET CHARACTER
Ø14Ø1A 1F5A 25 ØA    1F66        BCS    GETAD1   VALID HEX
Ø14Ø2A 1F5C A1 1Ø       A        CMP    #$1Ø
Ø14Ø3A 1F5E 27 2B    1F8B        BEQ    GETRTS
```

```
Ø14Ø4A 1F6Ø A1 11        A        CMP    #$11      NO ENTER?
Ø14Ø5A 1F62 27 27     1F8B        BEQ    GETRTS    NO TRY AGAIN
Ø14Ø6A 1F64 2Ø ED     1F53        BRA    GETADR
Ø14Ø7A 1F66 3F 52        A GETAD1 CLR    ADDRH     INIT HIGH ADDRESS
Ø14Ø8A 1F68 B7 53        A        STA    ADDRL     PUT CHAR AWAY
Ø14Ø9A 1F6A AD 44     1FBØ        BSR    PRTADR    PRINT NEW ADDRESS
Ø141ØA 1F6C AD DB     1F49 GETALP BSR    GETNYB    GET ANOTHER CHAR
Ø1411A 1F6E 24 12     1F82        BCC    GETARG    VALID?
Ø1412A 1F7Ø 48                    ASLA             YES
Ø1413A 1F71 48                    ASLA             SHIFT IT IN
Ø1414A 1F72 48                    ASLA
Ø1415A 1F73 48                    ASLA
Ø1416A 1F74 AE Ø4        A        LDX    #4
Ø1417A 1F76 48             GETASF ASLA
Ø1418A 1F77 39 53        A        ROL    ADDRL
Ø1419A 1F79 39 52        A        ROL    ADDRH
Ø142ØA 1F7B 5A                    DECX
Ø1421A 1F7C 26 F8     1F76        BNE    GETASF
Ø1422A 1F7E AD 3Ø     1FBØ        BSR    PRTADR    PRINT NEW ADDR
Ø1423A 1F8Ø 2Ø EA     1F6C        BRA    GETALP    GET ANOTHER CHAR
Ø1424A 1F82 A1 1Ø        A GETARG CMP    #$1Ø
Ø1425A 1F84 27 Ø5     1F8B        BEQ    GETRTS
Ø1426A 1F86 A1 11        A        CMP    #$11      IS ENTER?
Ø1427A 1F88 26 E2     1F6C        BNE    GETALP    NO TRY AGAIN
Ø1428A 1F8A 99                    SEC              YES SET FLAG
Ø1429A 1F8B 81             GETRTS RTS
Ø143Ø                      *
```

```
PAGE  Ø29  CBUGØ5  .SA:1

Ø1432                        *
Ø1433                        ****************************************
Ø1434                        *                                      *
Ø1435                        *      PRINT ONE BYTE INTO PAIR        *
Ø1436                        *      OF DISPLAY DIGITS               *
Ø1437                        *                                      *
Ø1438                        *      A REGISTER CONTAINS BYTE        *
Ø1439                        *                                      *
Ø1440                        *      X REGISTER POINTS TO 1ST        *
Ø1441                        *      DIGIT OF PAIR                   *
Ø1442                        *                                      *
Ø1443                        ****************************************
Ø1444                        *
Ø1445A 1F8C AE Ø4          A PRTDAT LDX    #4        PRINT IN LAST TWO DIGIT
Ø1446A 1F8E BF 5Ø          A PRTBYT STX    WORK1
Ø1447A 1F9Ø B7 55          A        STA    WORK4
Ø1448A 1F92 44                      LSRA
Ø1449A 1F93 44                      LSRA
Ø145ØA 1F94 44                      LSRA
Ø1451A 1F95 44                      LSRA
Ø1452A 1F96 97                      TAX
Ø1453A 1F97 D6 1E87        A        LDA    CTABL,X
Ø1454A 1F9A BE 5Ø          A        LDX    WORK1
Ø1455A 1F9C E7 49          A        STA    DTABL,X
Ø1456A 1F9E B6 55          A        LDA    WORK4
Ø1457A 1FAØ A4 ØF          A        AND    #$ØF
Ø1458A 1FA2 97                      TAX
Ø1459A 1FA3 D6 1E87        A        LDA    CTABL,X
Ø146ØA 1FA6 BE 5Ø          A        LDX    WORK1
Ø1461A 1FA8 E7 4A          A        STA    DTABL+1,X
Ø1462A 1FAA CD 1DFD        A        JSR    DISTAB
Ø1463A 1FAD B6 55          A        LDA    WORK4
Ø1464A 1FAF 81                      RTS
Ø1465                        *
Ø1466                        ****************************************
Ø1467                        *                                      *
Ø1468                        *      PRINT ADDRESS ADDRH,ADDRL       *
Ø1469                        *                                      *
Ø147Ø                        *      X REGISTER DESTROYED            *
Ø1471                        *                                      *
Ø1472                        ****************************************
Ø1473                        *
Ø1474A 1FBØ B7 56          A PRTADR STA    WORK5
Ø1475A 1FB2 BF 54          A        STX    WORK3
Ø1476A 1FB4 B6 52          A        LDA    ADDRH
Ø1477A 1FB6 5F                      CLRX
Ø1478A 1FB7 AD D5     1F8E          BSR    PRTBYT
Ø1479A 1FB9 B6 53          A        LDA    ADDRL
Ø148ØA 1FBB AE Ø2          A        LDX    #2
Ø1481A 1FBD AD CF     1F8E          BSR    PRTBYT
Ø1482A 1FBF B6 56          A        LDA    WORK5
Ø1483A 1FC1 BE 54          A        LDX    WORK3
Ø1484A 1FC3 81                      RTS
Ø1485                        *
```

```
01487                     *
01488A 1FC4 80            TIRQWV RTI
01489                     *
01490A 1FC5 80            IRQV   RTI
01491A 1FC6 80                   RTI
01492                     *
01493      1FC7     A TIRQV  EQU    *
01494A 1FC7 A6 40       A        LDA    #$40
01495A 1FC9 B7 09       A        STA    TIMEC
01496A 1FCB CD 1916     A        JSR    LOCSTK
01497A 1FCE E6 04       A        LDA    4,X
01498A 1FD0 BA 57       A        ORA    WORK6
01499A 1FD2 E7 04       A        STA    4,X
01500A 1FD4 CC 1928     A        JMP    PCOUNT
01501                     *
01502A 1FD7 CD 1E07     A PWRDWN JSR    CLRDIS
01503A 1FDA 8E                   STOP
01504                     *
01505A 1FF6                      ORG    $1FF6
01506                     *
01507A 1FF6    0046     A        FDB    TIRQW
01508A 1FF8    0043     A        FDB    TIRQ
01509A 1FFA    0040     A        FDB    IRQ
01510A 1FFC    1856     A        FDB    SWI
01511A 1FFE    1800     A        FDB    RESET
01512                     *
01513                            END
TOTAL ERRORS 00000--00000
```

AN-824

MC68000 DMA USING THE MC6844 DMA CONTROLLER

Prepared by
Arnold Morales
Microprocessor Applications Engineering

The MC6844 DMA Controller (DMAC) can be interfaced to the MC68000 microprocessor to provide flexible, low-cost, relatively high performance DMA control in an MC68000-based system. In designing such a system, three interface requirements must be considered:

1. The DMAC should operate at maximum frequency for efficient data transfer. High performance systems may require the use of the two megahertz device (MC68B44), so the system must allow the MC68000 to access the DMAC asynchronously.
2. Handshake logic must be implemented to arbitrate control of the system bus between the MC68000, the DMA control system, and other possible bus masters.
3. The MC6844 is an 8-bit device intended for use in MC68000 systems, capable of direct memory access through only a 64K memory space, and also lacks certain bus strobes necessary for simple implementation in an MC68000-based system. A bus interface must be designed to allow direct memory access throughout the entire 16 megabyte MC68000 memory map and to provide the required bus strobes needed for successful use in an MC68000-based system.

This application note describes designs to meet each of these requirements. These designs are then combined to form a direct memory access control system for the MC68000. An implementation of the complete system is presented in block diagram form using an MC6854 Advanced Data Link Controller (ADLC) and a static memory buffer.

MC6844 ASYNCHRONOUS INTERFACE OPERATION

The MC6844 can be interfaced asynchronously to the MC68000 using the circuitry presented in Figure 1. This circuit allows the MC68000 to access a DMAC driven by an E clock that is either synchronous or asynchronous to the MC68000 clock. It generates DMAC chip select at the proper time to satisfy DMAC timing requirements, latches data to satisfy data hold time requirements, and asserts data transfer acknowledge at the proper time to ensure valid data transfer between devices. This circuit can be used to interface other MC6800 peripherals, and is used to interface to the ADLC as well as the DMAC in the system implementation presented at the end of this application note.

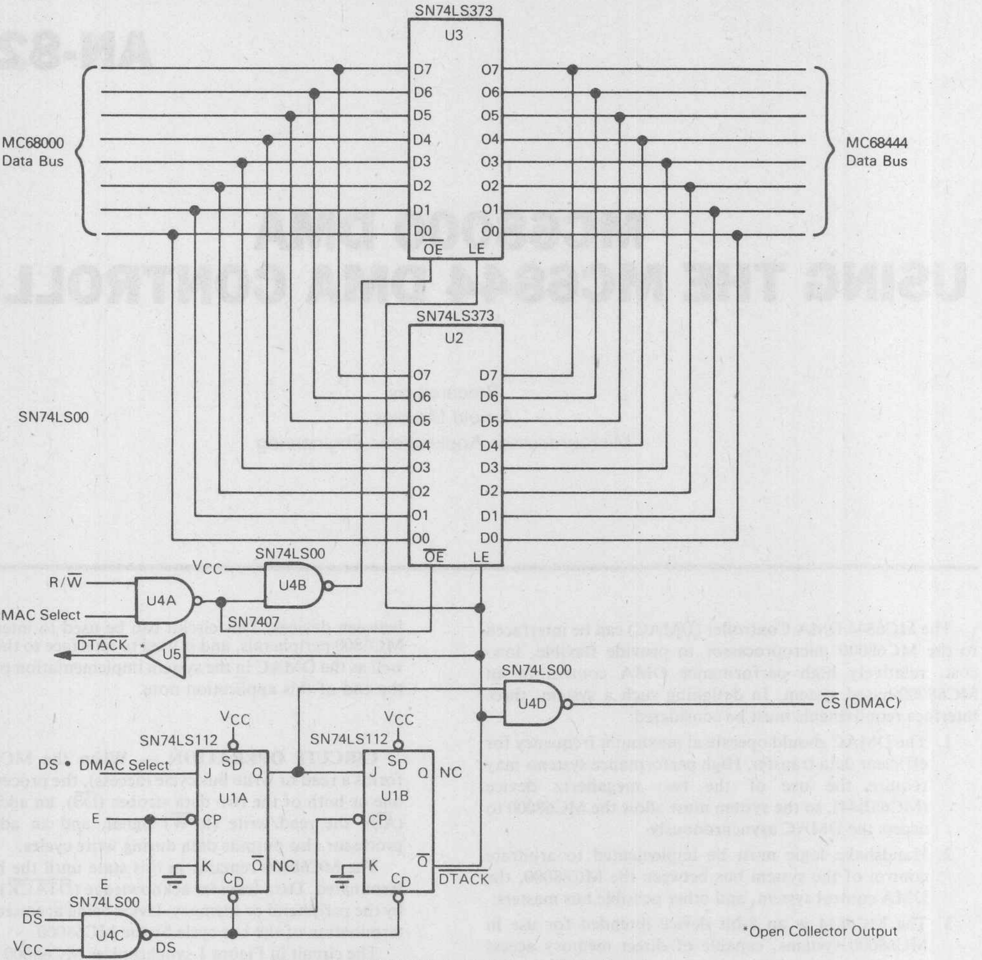
CIRCUIT OPERATION — When the MC68000 performs a read or write bus cycle (access), the processor asserts one or both of the two data strobes ($\overline{\text{DS}}$), an address strobe ($\overline{\text{AS}}$), the read/write (R/$\overline{\text{W}}$) signal, and an address. The processor also outputs data during write cycles.

The MC68000 remains in this state until the bus cycle is terminated. Data transfer acknowledge ($\overline{\text{DTACK}}$) is asserted by the peripheral or memory device being accessed to initiate termination of the bus cycle by the MC68000.

The circuit in Figure 1 synchronizes MC68000 accesses to the DMAC with the E clock. Initially, flip-flops U1A and U1B are cleared causing a high $\overline{\text{DTACK}}$ output setting U2 and U3 into a transparent mode. Latch U2 is in the high-impedance state due to a high on the output enable ($\overline{\text{OE}}$) input. Latch U3 is enabled due to a low on the $\overline{\text{OE}}$ input.

At the start of a DMAC access, latch U3 remains enabled if the access is a write. If the access is a read, the high R/$\overline{\text{W}}$ and DMAC Select inputs to U4A cause U3 to go to the high-impedance state and U2 to become enabled. The DMAC Select signal is asserted when the DMAC is addressed. However, the DMAC is actually selected by the assertion of CS (DMAC). Flip-flop U1A is clocked high on the first falling edge of E after DMAC Select and data strobe (DS) are asserted. The Q output of U1A is applied to U4D, asserting CS (DMAC). Selecting the DMAC at this time ensures that the DMAC has adequate address setup time.

On the next falling edge of E, the $\overline{\text{Q}}$ output of U1B is clocked low asserting $\overline{\text{DTACK}}$ and latching data into the enabled latch. The asserted $\overline{\text{DTACK}}$ signal, inverted by U4D, deselects the DMAC by causing $\overline{\text{CS}}$ (DMAC) to go high. When the access terminates, flip-flop U1 is cleared by the negation of $\overline{\text{DS}}$, and the interface circuitry is initialized for the next access. The $\overline{\text{DTACK}}$ signal is buffered by an open-collector buffer (U5) to allow assertion of $\overline{\text{DTACK}}$ by other devices when the DMAC is not being accessed.



*Open Collector Output

Figure 1. MC6844 Asynchronous Interface

BUS ARBITRATION INTERFACE

The MC6844 is an 8-bit, 4-channel DMA Controller capable of performing direct memory transfers of a user defined number of data bytes (data block) within a 64K byte memory space. Associated with each channel of the controller are:

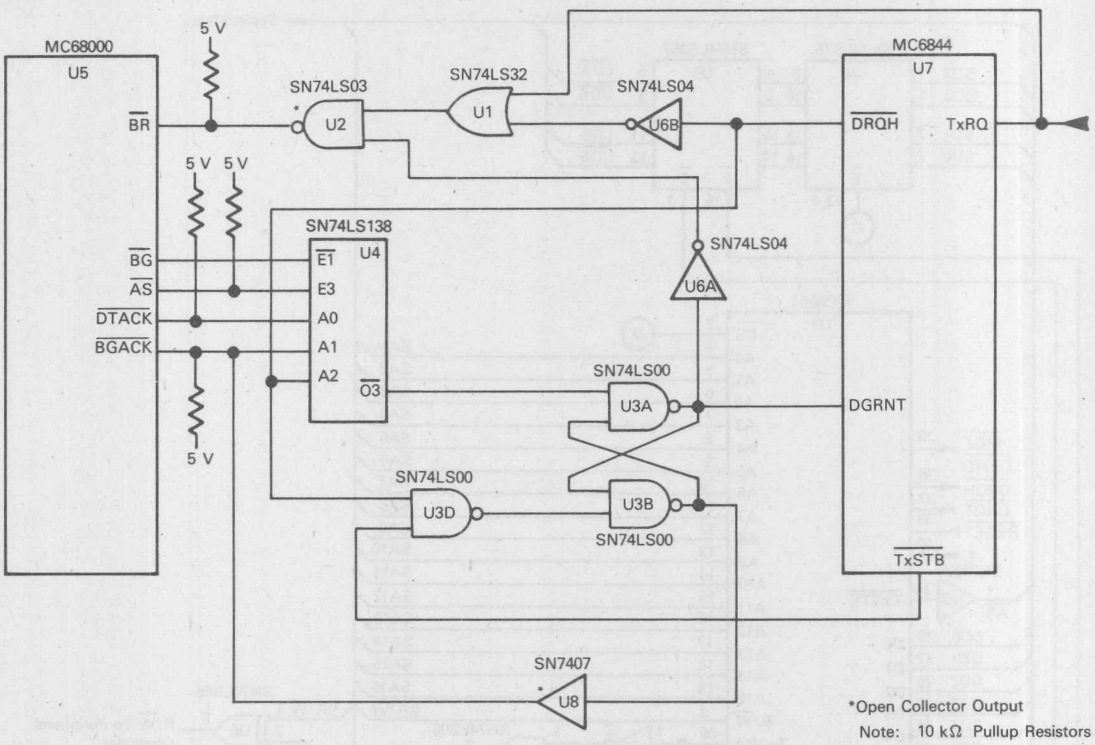
- A transfer request (TxRQ) input which is asserted by a peripheral controller or a processor to request DMA service.
- A 16-bit address register which is initialized with the beginning address of the data block to be transferred.
- A 16-bit byte count register which is initialized with the desired number of data bytes (size of the data block) to be transferred.

Each channel can perform DMA transfers in one of three modes: TSC Steal, Halt Steal, and Halt Burst. Two of these modes, TSC Steal and Halt Steal, are single-byte transfer modes in which the DMAC returns control of the system bus to the processor after each transfer, while the Halt Burst mode is a block transfer mode in which the DMAC retains control of the system bus until the last byte of the data block has been transferred.

The bus arbitration circuit presented in Figure 2 is designed for the Halt Steal and Halt Burst modes of operation. The TSC Steal mode is intended for use with the MC6800 and offers no advantage over the Halt Steal mode in MC68000 applications.

In the Halt Steal mode the DMAC responds to a transfer request by asserting DMA request halt steal ($\overline{\text{DRQH}}$). The DMAC then waits until DMA grant (DGRNT), a DMAC input, is asserted. At this time, one transfer of data is initialized and transfer strobe ($\overline{\text{TxSTB}}$) is asserted, followed by the negation of $\overline{\text{DRQH}}$. This sequence is repeated until all data has been transferred.

The same sequence is followed in the Halt Burst mode with the exception that $\overline{\text{DQRH}}$ is negated only after the last byte of the data block has been transferred. In this mode, bus mastership is arbitrated once, then data transfers occur in succession until all data has been transferred.



*Open Collector Output

Note: 10 kΩ Pullup Resistors

Figure 2. Bus Arbitration Interface

CIRCUIT OPERATION — For either a Halt Steal or Halt Burst DMA transfer by the control systems presented in this application note, three conditions must be met:

1. Transfer request (TxRQ) must be asserted.
2. $\overline{\text{DRQH}}$ must be asserted.
3. All bus masters must have relinquished the bus to ensure that DMA grant (DGRNT) is asserted.

Initially DGRNT is low, bus grant acknowledge ($\overline{\text{BGACK}}$) is not asserted by the interface, and $\overline{\text{TxSTB}}$ is high. The DMAC responds to a transfer request by asserting $\overline{\text{DRQH}}$. Once $\overline{\text{DRQH}}$ is asserted, it remains asserted until the DMAC performs a byte transfer in the Halt Steal mode or until the last byte of a designated memory block is transferred in the Halt Burst mode.

Transfer request (TxRQ) is coupled through U1 and U2 so that MC68000 bus request ($\overline{\text{BR}}$) is asserted when TxRQ is asserted. By requesting a DMA transfer and bus arbitration simultaneously (disregarding gate propagation delay), DMA latency time is minimized. The MC68000 responds to a bus request by asserting bus grant ($\overline{\text{BG}}$) and relinquishing the bus.

When DRQH is asserted and all bus masters are off the system bus, indicated by the negation of $\overline{\text{AS}}$, $\overline{\text{DTACK}}$, and $\overline{\text{BGACK}}$, flip-flop U3A-U3B is set by the assertion of the $\overline{\text{O3}}$ output of U4. The setting of flip-flop U3A-U3B asserts DGRNT to initiate DMA transfer(s), and also asserts $\overline{\text{BGACK}}$ to keep other bus masters off the bus. Bus grant ($\overline{\text{BG}}$) is negated by the MC68000 soon after $\overline{\text{BGACK}}$ is asserted.

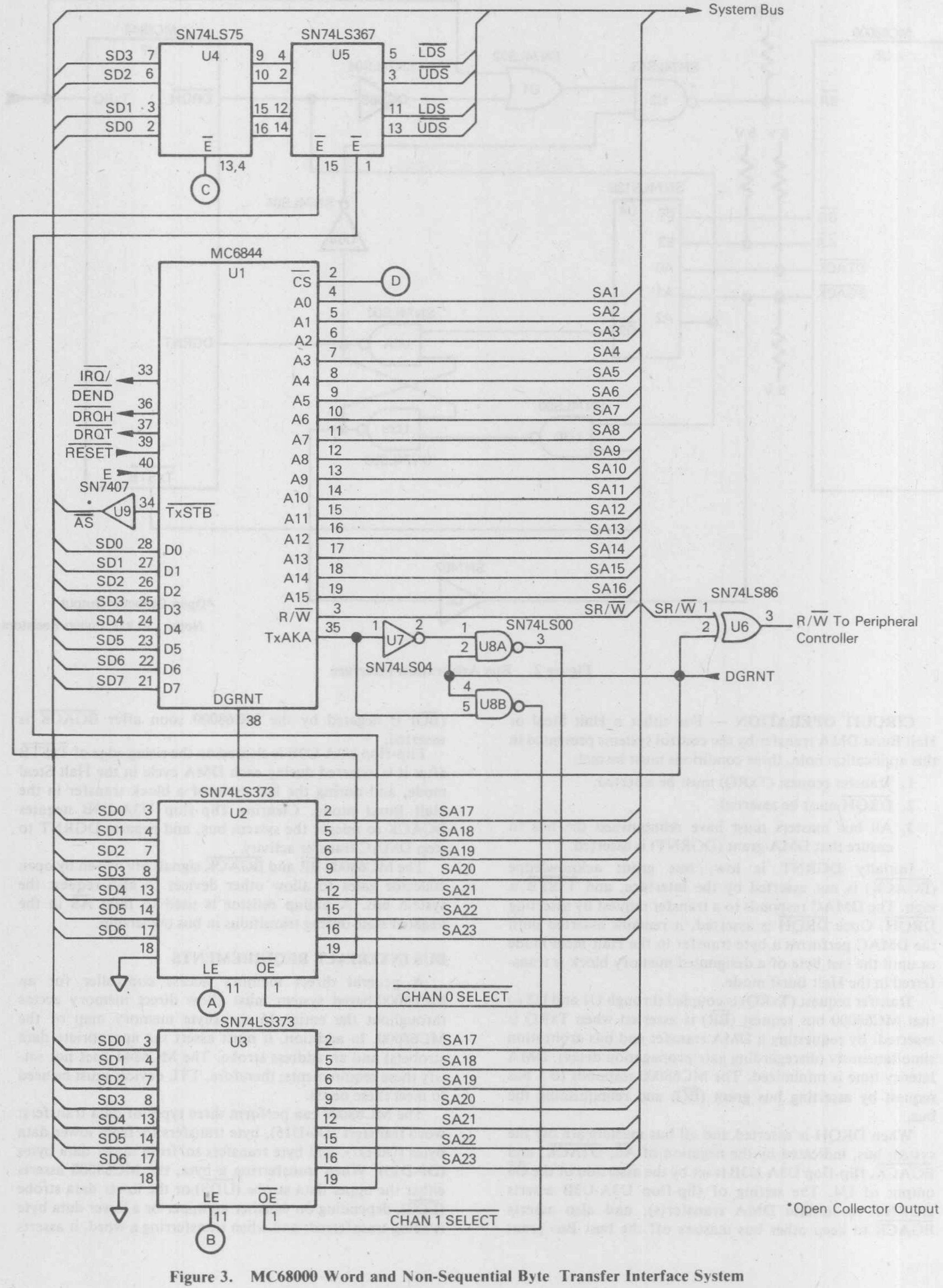
Flip-flop U3A-U3B is cleared on the rising edge of $\overline{\text{TxSTB}}$ after it is asserted during each DMA cycle in the Halt Steal mode, and during the last cycle of a block transfer in the Halt Burst mode. Clearing flip-flop U3A-U3B negates $\overline{\text{BGACK}}$ to release the system bus, and negates DGRNT to stop DMAC transfer activity.

The MC68000 $\overline{\text{BR}}$ and $\overline{\text{BGACK}}$ signals are driven by open collector gates to allow other devices to also request the system bus. A pullup resistor is used to hold $\overline{\text{AS}}$ in the negated state during transitions in bus ownership.

BUS INTERFACE REQUIREMENTS

A general direct memory access controller for an MC68000-based system must allow direct memory access throughout the entire 16 megabyte memory map of the MC68000. In addition, it must assert the appropriate data strobe(s) and an address strobe. The MC6844 does not satisfy these requirements; therefore, TTL devices must be used to meet these needs.

The MC68000 can perform three types of data transfers: word transfers (D0-D15), byte transfers to/from lower data bytes (D0-D7), and byte transfers to/from upper data bytes (D8-D15). When transferring a byte, the MC68000 asserts either the upper data strobe ($\overline{\text{UDS}}$) or the lower data strobe ($\overline{\text{LDS}}$), depending on whether an upper or a lower data byte is being transferred; and when transferring a word, it asserts



*Open Collector Output

Figure 3. MC68000 Word and Non-Sequential Byte Transfer Interface System

both $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$. The MC68000 asserts $\overline{\text{AS}}$ during each type of transfer.

The following are general designs which can be modified to meet individual system requirements. The two designs presented differ in the types of transfer they perform.

Only two of the four DMAC channels are used in each design. However, these interfaces can be easily modified for four-channel operation.

WORD AND NON-SEQUENTIAL BYTE TRANSFER INTERFACE SYSTEM

An MC68000 DMA control system capable of word transfers and byte transfers to/from upper byte or lower byte memory locations is presented in Figure 3.

In this system, address lines A0-A15 from the DMAC are connected to MC68000 system address lines SA1-SA23 and as the DMAC address lines increment or decrement (according to user option), the system address is incremented/decremented by words, rather than bytes; that is, the system address changes in increments of two bytes.

The system upper address lines SA17-SA23, are latched into transparent latches U2 and U3 during initialization, which are enabled during a DMA transfer. Latch U2 is the channel 0 upper address latch, with its chip select labeled A; latch U3 is the channel 1 upper address latch, with its chip select labeled B. During a direct memory access, transfer acknowledge A (TxAKA) from the DMAC is asserted during channel 1 transfers, and negated during channel 0 transfers. This DMAC output is used to enable the proper address latch during a direct memory access.

The type of direct memory access transfer (word or byte) is determined by the state of latch U4 during the access. Latch U4 with its chip select labeled C, is connected to system data bus lines SD0-SD3 and, through three-state buffer U5, to system data strobes $\overline{\text{LDS}}$ and $\overline{\text{UDS}}$. When writing to latch U4 during initialization, the states of SD2 and SD3 determine the states of the data strobes during a channel 1 direct memory access, and the states of SD0 and SD1 determine the states of the data strobes during a channel 0 direct memory access. For word transfer both of the data strobes must be asserted, while for byte transfers either the $\overline{\text{LDS}}$ or $\overline{\text{UDS}}$ is asserted, depending on whether a lower data byte (D0-D7) or an upper data byte (D8-D15) is being transferred.

Note that in memory organized in 16-bit words, byte transfers are to/from either the upper byte or the lower byte of memory during each DMA block transfer.

During a direct memory access the appropriate U4 latch states are gated onto the system bus by U5. The appropriate U5 buffers are enabled by latch U2 during channel 0 access, and by latch U3 during channel 1 access.

When $\overline{\text{DGRNT}}$ is asserted, the R/$\overline{\text{W}}$ signal to the peripheral controller is inverted by exclusive OR gate U6.

Transfer strobe ($\overline{\text{TxSTB}}$) is fed through an open collector buffer to the system $\overline{\text{AS}}$ line. During a direct memory access transfer the $\overline{\text{AS}}$ output of the MC68000 is in the high-impedance state and $\overline{\text{TxSTB}}$ is used as the system address strobe. Transfer strobe is asserted by a DMAC operating at 2 megahertz for at least 370 nanoseconds to indicate a valid address during a direct memory access, and may require conditioning for use as an address strobe during direct memory access in some systems.

SEQUENTIAL MEMORY BYTE TRANSFER INTERFACE SYSTEM

An MC68000 DMA control system capable of byte transfers to/from sequential memory locations in a memory organized in 16-bit words is presented in Figure 4. In this system, address lines A1-A15 from the DMAC are connected to MC68000 system address lines SA1-SA15. Address line A0 of the DMAC is connected to inverting buffer U4. Buffer U4 is enabled by DGRNT to generate the data strobes. Only one data strobe is asserted at a time. Each time the DMAC increments/decrements, the state of $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$ alternate. System address line SA1 changes state only after each data strobe is asserted for one DMA cycle and negated for one DMA cycle. By doing this, data is transferred to/from consecutive byte locations in the word-dimensioned memory map.

Latches U2 and U3 latch upper system address lines SA16-SA23 during initialization and their operation is identical to the circuit presented in Figure 3.

COMPLETE SYSTEM IMPLEMENTATION

A block diagram of a complete MC68000 DMA system using the MC6844 DMAC for controlling DMA between an MC6854 ADLC and a block of memory is presented in Figure 5. Data transfer in this system is between the ADLC and lower memory byte locations (D0-D7).

The ADLC asserts receiver data service request (RDSR) each time the receiver FIFO register requires servicing, and transmitter data service request (TDSR) each time the transmitter is ready for data. These outputs are tied to transfer request channel 0 (TxRQ0) and transfer request channel 1 (TxRQ1) of the DMAC so that DMAC channel 0 services the ADLC receiver, and DMAC channel 1 services the ADLC transmitter.

The block labeled "MC6854 Register Select, R/W Control" is used to address the ADLC transmit or receive register and to invert the read/write signal during a direct memory access. This circuit puts the address bus from the ADLC in the high-impedance state during the direct memory access and forces ADLC register select zero (RS0) to a low state and register select one (RS1) to a high state, so that during a direct memory access either the transmit FIFO register or the receiver FIFO register is selected according to the state of the R/$\overline{\text{W}}$ signal. The circuit uses DMAC $\overline{\text{DEND}}$ to select the frame terminate register of the ADLC during the last byte of a DMA block transfer when servicing the transmitter FIFO. During a direct memory access, the ADLC is selected by assertion of $\overline{\text{TxSTB}}$ to ensure that the ADLC is selected only during valid direct memory access cycles.

System memory is connected directly to the system bus. The "Memory $\overline{\text{DTACK}}$ Gen." consists of a counter driven by the MC68000 clock, and enabled by an asserted address strobe when memory is accessed by the processor. Data transfer acknowledge ($\overline{\text{DTACK}}$) is "picked off" one of the counter pins so that it is asserted at some preset time interval after memory is accessed.

Memory address decoding is the same for both direct memory access and processor data transfers. However, during a direct memory access, memory is deselected by the NOR or $\overline{\text{DGRNT}}$ and E. This ensures that, during a direct memory access, the memory will latch written data at the fall of E, when ADLC data is valid.

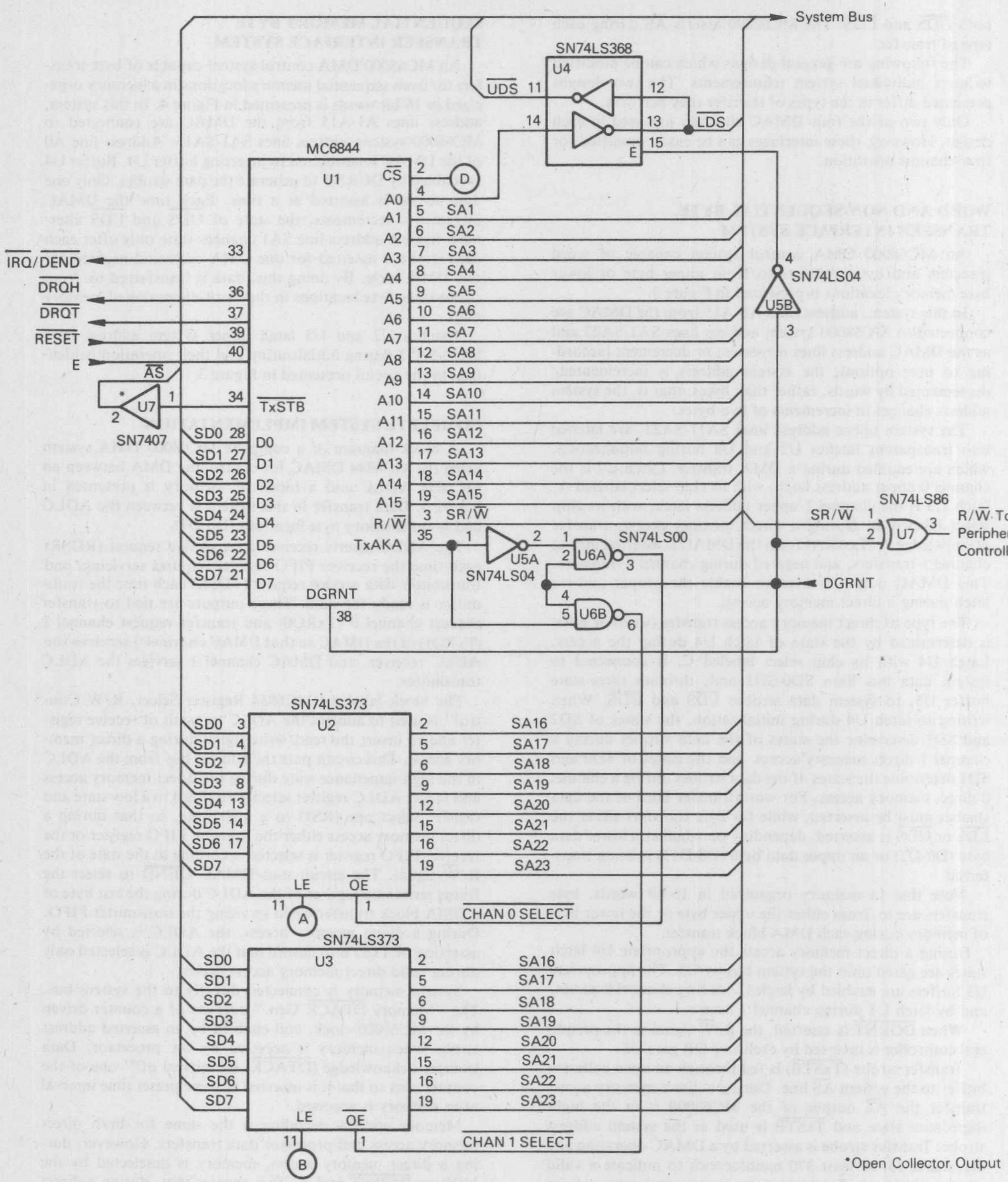


Figure 4. MC68000 Sequential Memory Byte Transfer Interface System

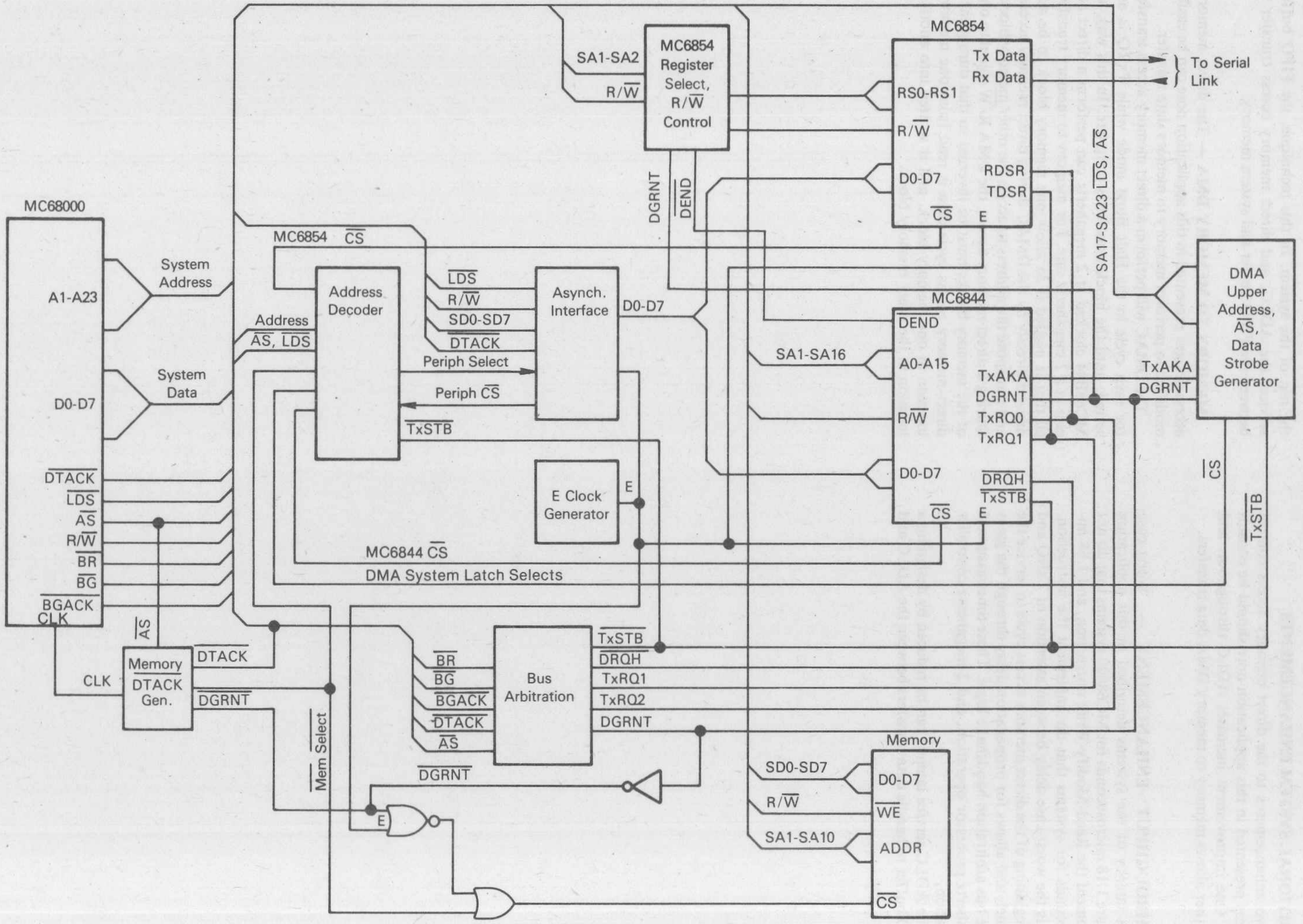


Figure 5. System Implementation

ADDITIONAL SYSTEM ENHANCEMENTS

Two enhancements to the direct memory access control systems presented in this application note should be considered. One improvement increases ADLC throughout, and the other allows memory to memory DMA data transfers.

THROUGHPUT ENHANCEMENT — Worst-case DMA latency of the systems described in this application note are 1.18 microseconds for MC68000 systems that do not implement the Read-Modify-Write instruction, and 1.68 microseconds for systems that do implement the instruction. This is the worst-case delay between assertion of TxRQ and the beginning of the direct memory access cycle to service the channel, and allows for propagation delay through the gates in the bus arbitration handshake logic. These times assume 8 megahertz processor operation, and 2 megahertz controller operation.

The ADLC service latency can be reduced by designing a FIFO buffer to handle data transfers between the ADLC and the rest of the system. In this technique, the FIFO buffer services the ADLC, and direct memory access transfer is between the FIFO buffer and system memory.

MEMORY TO MEMORY DMA — The direct memory access designs presented in this application note can be easily modified to perform memory to memory data transfer.

The DMAC will perform a direct memory access transfer for each cycle in the Halt Burst mode while TxRQ is asserted, until the block transfer is complete. In this way, an MC68B44 clocked at 2 megahertz can perform a direct access at a 2 megahertz rate. For memory to memory transfer, all that is needed is to allow one memory block to be addressed directly by the DMAC during direct memory access, and transpose the address to access the other memory block. During a direct memory access, the DMA R/$\overline{W}$ signal to one of the memory blocks must be inverted so that during each direct memory access cycle data is read from one memory location in one memory block, and is written into another location in the other memory block.

AN-830

AN INTELLIGENT TERMINAL WITH DATA LINK CAPABILITY

By
Charles Melear
Microprocessor Applications Engineer

INTRODUCTION

A small but powerful terminal complete with high speed data link can be constructed with a minimum number of NMOS LSI circuits. Operating systems can be developed to make this terminal act as a word processor, point-of-sale terminal, data input source, etc. The data link capability allows the operator to call in the resources of remote computers at synchronous serial data rates of up to 1.5 megahertz.

Five devices form the core of the terminal as shown in Figure 1. An MC6809 Microprocessor (MPU) was chosen because of its many hardware and software features.

The MC6845 CRT Controller (CRTC) permits the use of a video display monitor. This controller was chosen because it allows complete software control of the video display monitor. Vertical sync delay, horizontal sync width and delay, blanking, number of characters-per-row, and rows-per-screen are all programmable.

Serial keyboard input capability is provided by an MC6850 Asynchronous Communications Interface Adapter (ACIA) which performs the serial/parallel conversions. This application polls the ACIA to check for a data present indication instead of using an interrupt. This polling method provides the highest priority and shortest response time to the high speed data link.

High speed data link capabilities are provided by an MC6854 Advanced Data Link Controller (ADLC). The ADLC detects the start of a message, receives the message, calculates and appends a CRC character, and provides a closing flag. Serial data rates of 1.5 megahertz are possible with this system. To operate at these speeds, direct memory access capability is needed and is provided, in this application, by an MC6844 Direct Memory Access Controller (DMAC). A data transfer can be processed every four bus cycles when an MC6854 ADLC and an MC6844 DMAC are used together.

MC6845 CRT CONTROLLER (CRTC)

The CRTC provides horizontal sync, vertical sync, and blanking to a video display monitor along with the memory address of the data to be displayed. A cursor output is also provided. Once the CRTC is initialized, it performs the function of controlling the video display monitor without intervention by the processor. Initialization is accomplished by writing the appropriate values into the 16 programmable registers. Figure 2 Sheet 1 is a worksheet which can be used to collect the information required to calculate the values needed for the CRTC register worksheet given in Figure 2 Sheet 2. It is assumed that the video display monitor uses a 60 hertz power source and a 15,750 hertz horizontal oscillator frequency. After initialization, the CRTC starts with the address located in the start address register. The ASCII character represented by the hexadecimal value at that location will appear in the upper left-hand corner of the video display monitor. The CRTC advances the memory address lines by one with each character clock. The first row will contain the number of characters specified in the horizontal display register.

Due to synchronization problems between the CRT clock and several other signals, it is possible that the first character could be only partially displayed. Figure 3 shows how this can happen because the time between the CRT clock and display enable (Tx) is an internal function of the CRTC. The first character will be partially displayed because display enable goes high approximately in the middle of the first character. This problem can be resolved by writing an ASCII blank (20) at the first character location and using the second character location to display the first character.

The screen memory must be accessible to the processor for updating. Since the CRTC memory address lines normally drive the screen memory, multiplexers are used to select either the CRTC memory address lines or the processor address lines. A decoding network selects the processor address lines any time an address between $0000 and $1FFF is detected. The data bus for the screen memory is isolated from the processor data bus by SN74LS243 transceivers. These devices are normally in the high-impedance state in both directions except during a processor read or write of the

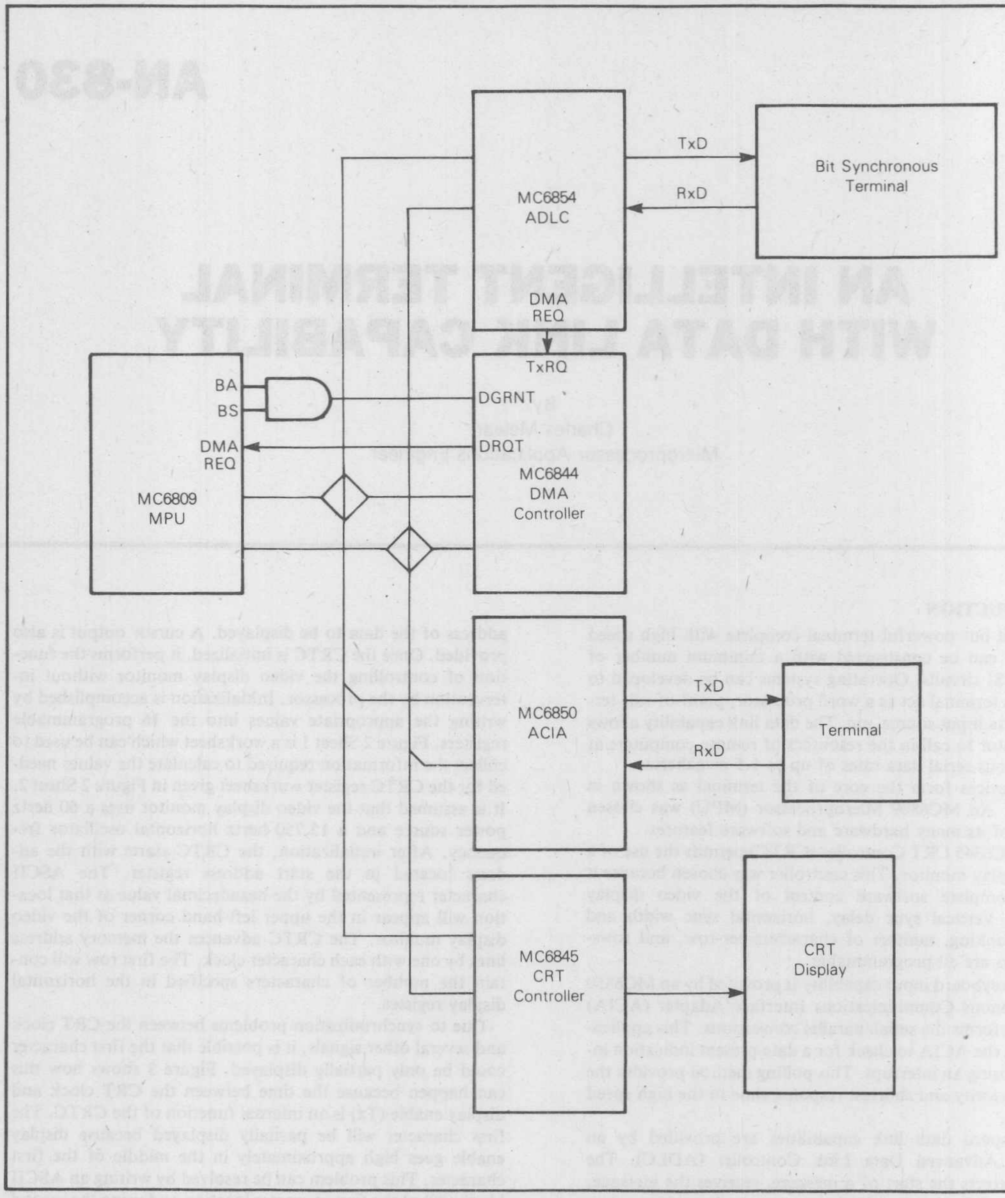


Figure 1. Intelligent Terminal — Block Diagram

screen memory. The direction signals for the transceivers are derived from the select signal to the address line multiplexers and the read/write from the processor.

The output of the screen memory is latched into an SN74LS374 octal latch. The shift/load signal is used as a strobe to the latch. This latch is used to synchronize data flow between the screen memory and the output shift register. The latch holds data for one full CRT clock cycle and is used mainly to remove concern about memory propagation times. The output of the latch drives an MCM66740 character generator which feeds the parallel input of an SN74LS165 shift register. The shift/load signal is used to load the shift register. The dot clock is used to serially shift the data from the shift register.

In order to display a row of characters on a video display screen, the top line of each character must be addressed, then the second line, and so on until each row has been displayed. The CRTC steps through all the addresses to be displayed in

Format Worksheet

1. Displayed Characters Per Row ________ Char.
2. Displayed Character Rows Per Screen ________ Rows
3. Character Matrix a. Columns ________ Columns
   b. Rows ________ Rows
4. Character Block a. Columns ________ Columns
   b. Rows ________ Rows
5. Frame Refresh Rate ________ Hz
6. Horizontal Oscillator Frequency ________ Hz
7. Active Scan Lines (Line 2 × Line 4b) ________ Lines
8. Total Scan Lines (Line 6 ÷ Line 5) ________ Lines
9. Total Rows Per Screen (Line 8 ÷ Line 4b) ________ Rows

9a. Number of Scan Lines Remaining From Line 9 ________ Lines

10. Vertical Sync Delay (Character Rows) ________ Rows
11. Vertical Sync Width (Scan Lines) ____16____ Lines
12. Horizontal Sync Delay (Character Times) ________ Char. Time
13. Horizontal Sync Width (Character Times) ________ Char. Time
14. Horizontal Scan Delay (Character Times) ________ Char. Time
15. Total Character Times (Line 1 + 12 + 13 + 14) ________ Char. Time
16. Character Rate (Line 6 times 15) ________ Hz
17. Dot Clock Rate (Line 4a times 16) ________ Hz

Figure 2. CRTC Programming Worksheet (Sheet 1 of 2)

CRTC Register Worksheet

		Decimal	Hex
R0	Horizontal Total (Line 15 minus 1)		
R1	Horizontal Displayed (Line 1)		
R2	Horizontal Sync Position (Line 1 + Line 12)		
R3	Horizontal Sync Width (Line 13)		
R4	Vertical Total (Line 9 minus 1)		
R5	Vertical Adjust (Line 9a Lines)		
R6	Vertical Displayed (Line 2)		
R7	Vertical Sync Position (Line 2 + Line 10)		
R8	Interlace (00 Normal, 01 Interlace, 03 Interlace and Video)		
R9	Max Scan Line Add (Line 4b minus 1)		
R10	Cursor Start		
R11	Cursor End		
R12	Start Address (H)		
R13	Start Address (L)		
R14	Cursor (H)		
R15	Cursor (L)		
R16	Light Pen (H)		
R17	Light Pen (L)		

Figure 2. CRTC Programming Worksheet (Sheet 2 of 2)

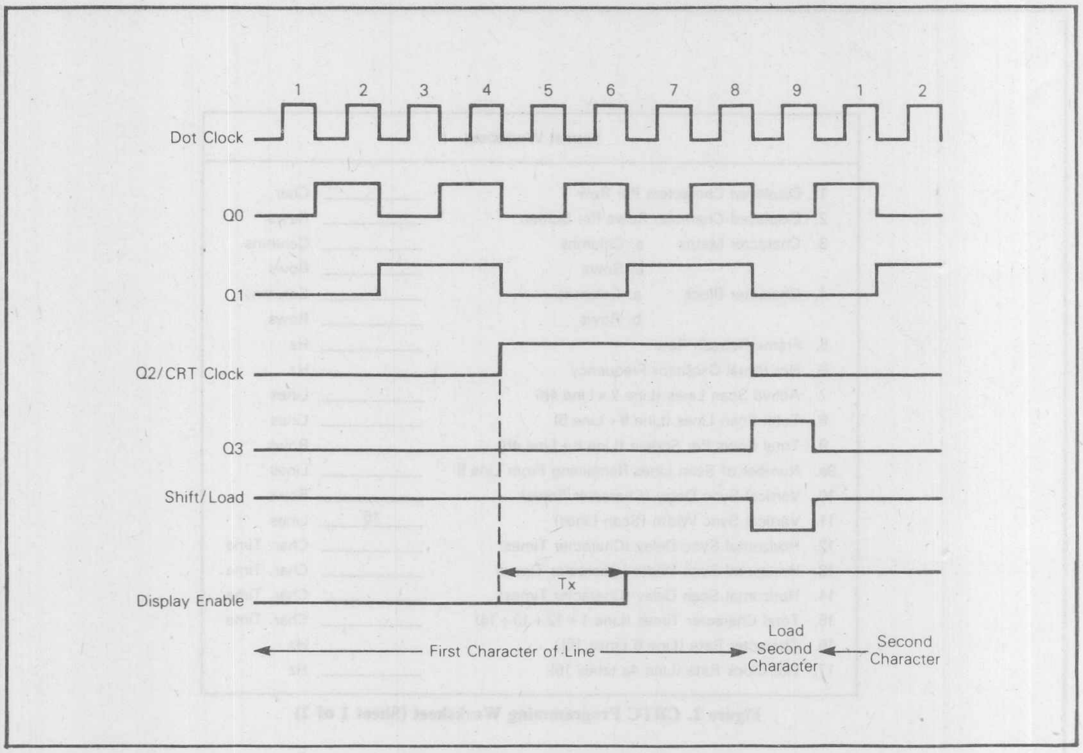


Figure 3. First Character Timing

the first character row, increments the row address by one, and then steps through the same addresses again. This procedure is repeated until the row address is equal to the address contained in the maximum scan line address register. The row address is reset to zero and the second character row is displayed.

A cursor may be programmed to appear at any location within the display memory area. The cursor output signal is logically ORed with the output of the data output shift register to form a new signal called cursor plus data.

Display enable is used for vertical and horizontal blanking. The data to the video display monitor must be enabled only during the time that the beam of the video display monitor is sweeping what has been defined as the display area. Otherwise, random data may appear at the edges of the screen and horizontal and vertical retrace lines may also be visible. Display enable goes high as the first character of a row is displayed and goes low just after the last character of a row is displayed.

Signals which include cursor plus data, display enable, and the select signal to the address line multiplexers are ANDed to form the composite data signal applied to the video display monitor. The select signal to the address line multiplexer is included to suppress any spurious data that may occur when the processor accesses the display memory. Composite data is fed to a D-type flip-flop that is clocked by the dot clock. This ensures that boundaries between dot periods in the composite data signal occur at regular intervals.

MC6844 DIRECT MEMORY ACCESS CONTROLLER (DMAC)

This application has local keyboard interface capability through use of an MC6850 Asynchronous Communications Interface Adapter (ACIA). It also has serial data link capability through the use of an MC6854 Advanced Data Link Controller (ADLC). This is a high speed data link capable of data transfer rates up to 1.5 megabits per second. If used at maximum speed in full duplex, a polling routine would not be able to handle the transmitted and received data. Therefore, direct memory access capability is needed. At one megabit data transfer rates, a data transfer must occur every four microseconds if full duplex operation is used. An MC6844 Direct Memory Access Controller (DMAC) can transfer data at that rate. One transfer is made every eight microseconds on each of two channels or one byte received and one byte transmitted during eight microseconds. The DMAC has four channels, but only two are used in this application. When enabled, two different pins on the ADLC are used to indicate that the transmit data register is empty and that the receiver FIFO buffer is full. These signals are used to make transfer requests to channels zero and one of the DMAC.

When the transfer request line (TxRQ) goes high in response to a service request from the ADLC, the DMAC requests the data bus from the MPU. When the data and address buses are available, the MPU will assert bus available (BA) and bus status (BS). The logical AND of these signals is the DMA grant signal (DGRNT) to the DMAC. When DMA grant is received, the DMAC automatically takes control of the buses in one cycle and performs the data transfer during the next cycle. The bus request from the DMAC is released during the transfer cycle. The MC6809 will not attempt to regain the bus until one full cycle after the release of bus request. The bus available and bus status signals from the MPU are released immediately after the removal of bus requests which causes DMA grant to go low. This allows the DMAC to put its bus drivers in the high-impedance state in the cycle following the transfer without the possibility of bus contention by the MPU.

The DMAC has a number of 8-bit registers to be programmed. Figure 4 is an illustration of these registers. Channel zero is a transmit channel and its address register (registers 0 and 1) is loaded with the first address in memory to be transferred. The channel zero byte count register (registers 2 and 3) is loaded with the number of bytes to be transferred. The address register for channel one (register 4 and 5) is loaded with the first address in memory to serve as a destination for data. The byte count register for channel one (registers 6 and 7) should be loaded with $FFFF since the length of an incoming message is generally not known. This value will allow a message of any length. Registers 8 through F are not used. In this application, channel zero is programmed for the three-state control steal transfer mode and read (from memory to ADLC); and channel one is programmed for three-state control steal transfer mode and write (from ADLC to memory).

The priority control register is used to enable the transmit and receive channels when desired. Only interrupt request/DMA end ($\overline{\mathrm{IRQ}}$/$\overline{\mathrm{DEND}}$) for channel zero is enabled in the interrupt control register. This will cause an interrupt when the channel zero byte count register is decremented to zero indicating that all bytes have been transferred. A DMA end ($\overline{\mathrm{DEND}}$) will occur when the last byte is transferred to the transmit register of the ADLC. DMA end ($\overline{\mathrm{DEND}}$) and interrupt request ($\overline{\mathrm{IRQ}}$) are multiplexed on one pin. By taking the logical OR of DGRNT and $\overline{\mathrm{IRQ}}$/$\overline{\mathrm{DEND}}$, a separate $\overline{\mathrm{IRQ}}$ can be obtained. The separate $\overline{\mathrm{DEND}}$ is obtained by taking the logical OR of the transfer strobe (TxSTB) and interrupt request/DMA end ($\overline{\mathrm{IRQ}}$/$\overline{\mathrm{DEND}}$). In actual use the DMAC is programmed and enabled before the ADLC is enabled. This will ensure that transfers can begin immediately upon initialization of the ADLC.

Programming Model

Register	Address (Hex)	Register Content							
		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Channel Control	1x*	DMA End Flag (DEND)	Busy/Ready Flag	Not Used	Not Used	Address Up/Down	TSC/ Halt	Burst/ Steal	Read/Write ($R/\overline{W}$)
Priority Control	14	Rotate Control	Not Used	Not Used	Not Used	Request Enable #3 (RE3)	Request Enable #2 (RE2)	Request Enable #1 (RE1)	Request Enable #0 (RE0)
Interrupt Control	15	DEND IRQ Flag	Not Used	Not Used	Not Used	DEND IRQ Enable #3 (DIE3)	DEND IRQ Enable #2 (DIE2)	DEND IRQ Enable #1 (DIE1)	DEND IRQ Enable #0 (DIE0)
Data Chain	16	Not Used	Not Used	Not Used	Not Used	Two/Four Channel Select (2/4)	Data Chain Channel Select B	Data Chain Channel Select A	Data Chain Enable

*The x represents the binary equivalent of the channel desired.

Address and Byte Count Registers

Register	Channel	Address (Hex)
Address High	0	0
Address Low	0	1
Byte Count High	0	2
Byte Count Low	0	3
Address High	1	4
Address Low	1	5
Byte Count High	1	6
Byte Count Low	1	7
Address High	2	8
Address Low	2	9
Byte Count High	2	A
Byte Count Low	2	B
Address High	3	C
Address Low	3	D
Byte Count High	3	E
Byte Count Low	3	F

Figure 4. MC6844 DMAC Registers

MC6854 ADVANCED DATA LINK CONTROLLER (ADLC)

The ADLC handles the data link protocol. Basically, the ADLC transmits and receives serial data in full duplex. The data format of message frame is shown in Figure 5. When transmitting, the transmit data output (TxD) will either be high (mark idle) or sending a series of opening flags (flag idle). Upon writing a word to the transmit FIFO register, an opening flag will be sent followed by the data. Details of the ADLC registers are given in Figure 6. Data must be supplied to the transmit FIFO register at a rate sufficient to ensure that the data output shift register never becomes empty. The last byte to be transmitted is written to the transmit last data register. As soon as the last byte is transmitted, the ADLC automatically appends a 16-bit cyclic redundancy character (CRC) in the frame check sequence field and a closing flag. The receiver constantly searches the data stream for an opening flag with which to synchronize. After an opening flag is detected, the first non-flag character and all succeeding bytes are shifted into the receiver FIFO register and CRC calculation is started. The receiver FIFO register must be read fast enough to ensure that a receiver overrun does not occur. When a closing flag is detected, the ADLC takes the prior 16 bits and compares it to the CRC generated by the receiver. The CRC is not shifted into the receiver FIFO register.

The chip select ($\overline{\mathrm{CS}}$) pin of the ADLC must be asserted whenever the DMAC requests data by issuing a transmit strobe or when the address of the ADLC appears on the address bus. The logical ANDing of TxSTB and the address of the ADLC is used to develop a composite chip select ($\overline{\mathrm{CS}}$) signal.

When a DMA transfer occurs between memory and the ADLC, the DMAC controls the R/$\overline{\mathrm{W}}$ line for the system. During the transfer cycle, the R/$\overline{\mathrm{W}}$ line of the ADLC must be inverted with respect to the system R/$\overline{\mathrm{W}}$ line. This is accomplished by exclusive ORing TxSTB and R/$\overline{\mathrm{W}}$. If TxSTB is low (no transfer), the output follows R/$\overline{\mathrm{W}}$. If TxSTB is high (transfer cycle), the output is the complement of R/$\overline{\mathrm{W}}$.

The ADLC requires that the last byte to be transferred be treated differently. The system may set bit four of control register two high and write the last byte into the transmitter (continue) data register or the last byte can be written into the transmitter (last) data register. In this application, the latter method is used by using $\overline{\mathrm{TxSTB}}$, $\overline{\mathrm{IRQ/DEND}}$, and R/$\overline{\mathrm{W}}$ to control a dual, 4-to-1 data selector. The truth table for the data selector is shown in Table 1. When $\overline{\mathrm{DEND}}$ is low, the DMAC is indicating that this is the last byte. $\overline{\mathrm{DEND}}$ occurs coincidentally with $\overline{\mathrm{TxSTB}}$ which forces the register selects (RS0, RS1) of the ADLC high and selects the transmitter (last) data register. If only $\overline{\mathrm{TxSTB}}$ is low, register select zero will be low and register select one will be high and the transmitter (continue) data register will be selected.

MC6809 MICROPROCESSING UNIT (MPU)

The MC6809 must be discussed from two viewpoints — hardware and software.

HARDWARE — The internal clock of the MPU is made to work with the MC6844 DMA Controller. Figure 7 is a timing diagram for a DMA response and three-state steal. The DMA request three-state control steal ($\overline{\mathrm{DRQT}}$) output of DMAC drives the $\overline{\mathrm{DMA/BREQ}}$ input of the MPU. As shown in Figure 7, the first full cycle following $\overline{\mathrm{DRQT}}$ going low (which causes $\overline{\mathrm{DMA/BREQ}}$ to also go low) is a dead cycle. Since the $\overline{\mathrm{DRQT}}$ low output from the DMA results in the $\overline{\mathrm{DMA/BREQ}}$ input to MC6809 going low, it is a dead cycle for both the DMA and the MC6809. Dead time is the time required for the MPU to relinquish control of the bus and the DMAC to gain control of the bus. The next cycle accommodates the DMA transfer. During this cycle, $\overline{\mathrm{DRQT}}$ is released. The MPU automatically inserts one dead cycle after $\overline{\mathrm{DMA/BREQ}}$ is released. This gives the DMAC one cycle to relinquish control of the bus and the MPU to gain control of the bus. After the dead cycle, the MPU assumes normal control.

The MC6809 has no equivalent of the valid memory address (VMA) signal which is available on the MC6800. Normally a VMA is not needed; however, during the dead cycles which precede and follow a DMA transfer, the buses are undefined. This allows the possibility of a spurious write into a random memory location. This possibility can be eliminated by developing a signal called direct memory access valid memory address ($\overline{\mathrm{DMAVMA}}$). The DMA grant DGRNT signal from the DMAC and the E signal are used to

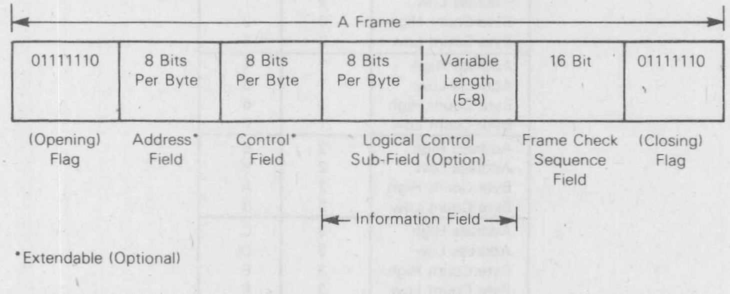


Figure 5. Data Format of a Message Frame

Read Only Registers	0	RDA	Address Present	Bit 0	Same as RS1, RS0 = 10
	1	Status #2 Read Request	Frame Valid	Bit 1	
	2	Loop	Inactive Idle Received	Bit 2	
	3	Flag Detected (When Enabled)	Abort Received	Bit 3	
	4	$\overline{CTS}$	FCS Error	Bit 4	
	5	Tx Underrun	$\overline{DCD}$	Bit 5	
	6	TDRA/Frame Complete	Rx Overrun	Bit 6	
	7	$\overline{IRQ}$ Present	RDA (Receiver Data Available)	Bit 7	

					Transmitter Data	Transmitter Data	
	Bit #	Control Register = 1	Control Register = 2 ($C_1b_0 = 0$)	Control Register = 3 ($C_1b_0 = 1$)	(Continue Data)	(Last Data) ($C_1b_0 = 0$)	Control Register = 4 ($C_1b_0 = 1$)
Write Only Registers	0	Address Control (AC)	Prioritized Status Enable	Logical Control Field Select	Bit 0	Bit 0	Double Flag Single Flag Interframe Control
	1	Receiver Interrupt Enable (RIE)	2 Byte/1 Byte Transfer	Extended Control Field Select	Bit 1	Bit 1	Word Length Select Transmit #1
	2	Transmitter Interrupt Enable (TIE)	Flag/Mark Idle	Auto, Address Extension Mode	Bit 2	Bit 2	Word Length Select Transmit #2
	3	RDSR Mode (DMA)	Frame Complete/ TDRA Select	01/11 Idle	Bit 3	Bit 3	Word Length Select Receive #1
	4	TDSR Mode (DMA)	Transmit Last Data	Flag Detected Status Enable	Bit 4	Bit 4	Word Length Select Receive #2
	5	Rx Frame Discontinue	CLR Rx Status	Loop/Non-Loop Mode	Bit 5	Bit 5	Transmit Abort
	6	Rx RESET	CLR Tx Status	Go Active on Poll/Test	Bit 6	Bit 6	Abort Extend
	7	Tx RESET	RTS Control	Loop On-Line Control DTR	Bit 7	Bit 7	NRZI/NRZ

Figure 6. MC6854 ADLC Internal Register Details

Table 1. Data Selector Truth Table

Operation	$\overline{TxSTB}$	$\overline{DEND}$	R/$\overline{W}$	A	B	1Y	2Y
Normal Operation No DMA Transfer	1	X	X	1	1	A1	A0
DMA Transfer from ADLC to Memory	0	1	0	0	1	1	0
DMA Transfer from to ADLC	0	1	1	0	1	1	0
DMA Transfer of Last Byte from Memory to ADLC	0	0	0	0	0	1	1
DMA Transfer of Last Byte from ALDC to Memory	0	0	1	0	1	1	0

develop the $\overline{\text{DMAVMA}}$ signal as shown in Figure 8. A timing diagram showing the effect of the $\overline{\text{DMAVMA}}$ signal is given in Figure 7.

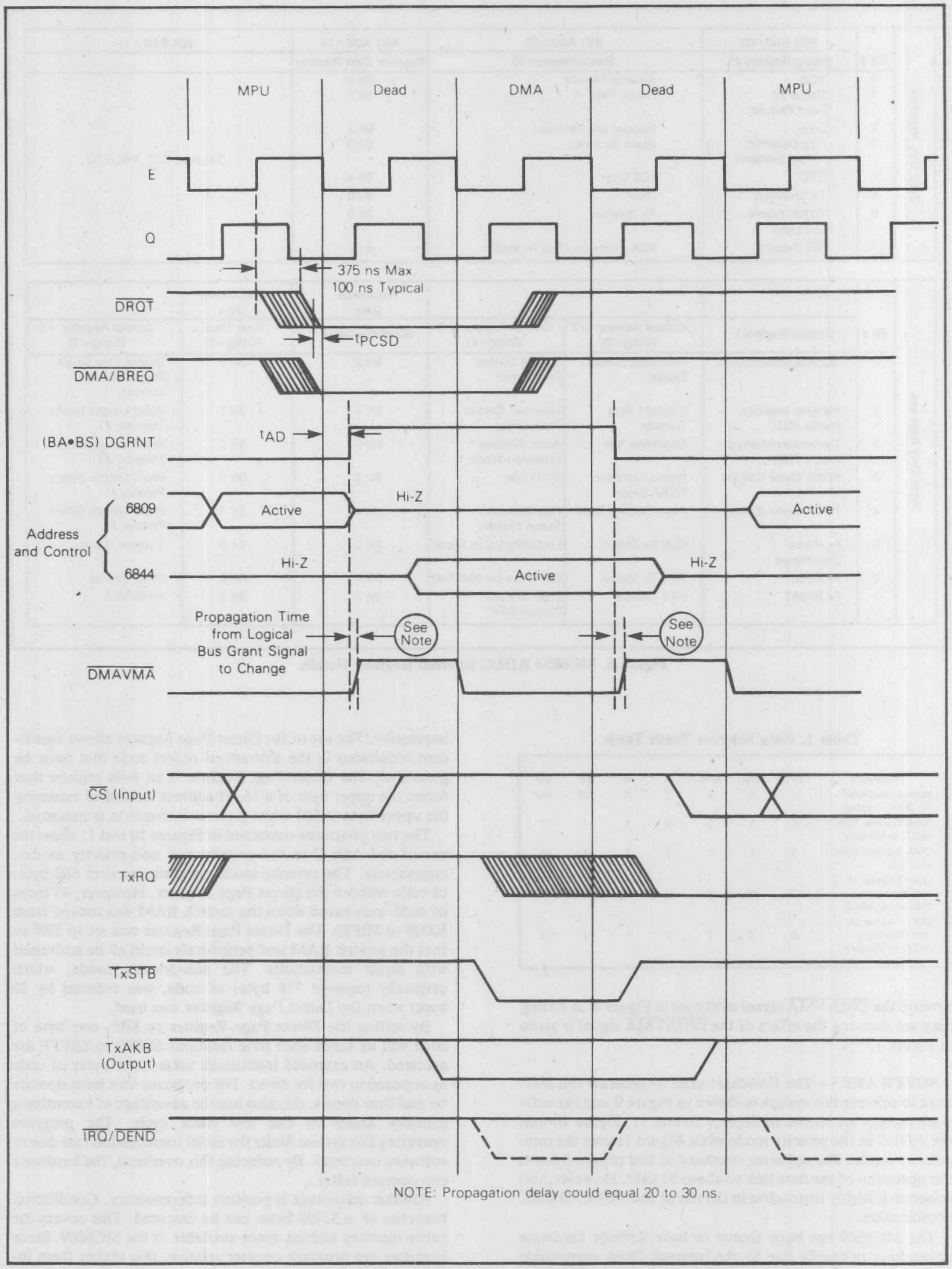
SOFTWARE — The flowchart used to generate the software to operate this system is shown in Figure 9 and the software listings are shown in Figures 10 and 11. Figure 10 uses the ADLC in the priority mode while Figure 11 uses the non-priority mode. The software overhead in this program limits the operation of the data link to about 62 kHz. However, this program is highly instructive in the use of the ADLC/DMAC combination.

The MC6809 has been shown to have definite hardware advantages primarily due to the internal DMA compatible clocks; however, the software advantages are are also quite impressive. The use of the Direct Page Register allows significant reductions in the amount of object code that must be generated. The Direct Page Register is an 8-bit register that forms the upper byte of a 16-bit address instead of assuming the upper byte is $00 when a direct instruction is executed.

The two programs contained in Figures 10 and 11 show the use of the ADLC in the priority and non-priority modes, respectively. The priority mode program requires 660 bytes of code without the Direct Page Register. However, 43 bytes of code were saved when the scratch RAM was moved from $0000 to $BF10. The Direct Page Register was set to $BF so that the scratch RAM and peripherals could all be addressed with direct instructions. The non-priority mode, which originally required 718 bytes of code, was reduced by 36 bytes when the Direct Page Register was used.

By setting the Direct Page Register to $BF, one byte of code will be saved each time locations $BF00 to $BFFF are accessed. An extended instruction takes three bytes of code as opposed to two for direct. For programs that must operate on real time events, this also has the advantage of executing a memory access in one less clock cycle. The program operating this system limits the serial transmission rate due to software overhead. By reducing this overhead, the hardware can operate faster.

Another advantage is position independency. Conditional branches of ± 32768 bytes can be executed. This covers the entire memory address space available to the MC6809. Since branches are program counter relative, this makes them independent of where the program originates. MC6800



NOTE: Propagation delay could equal 20 to 30 ns.

Figure 7. Three-State Steal DMA Timing

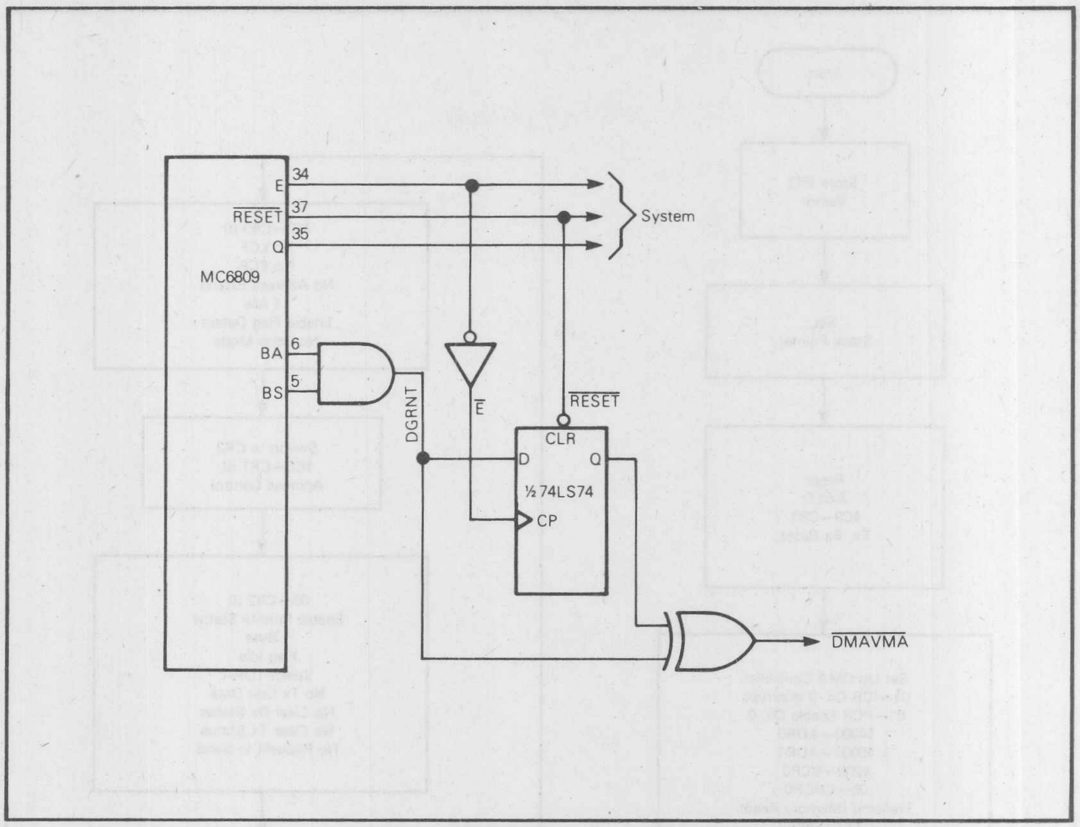


Figure 8. $\overline{\text{DMAVMA}}$ Generation Circuit

branches being ± 128 bytes must be used to branch to jump statements for moves of greater than 128 bytes. A long branch to subroutine instruction also enables ± 32K branches which are also program counter relative. When a program is completely position independent, the code can be placed anywhere in the memory space and work. MC6800 programs cannot be made position independent unless they are written in the first 256 memory locations. Therefore, the MC6809 offers the convenience of position-independent ROMs.

SYSTEM SCHEMATIC

Figure 12 is the schematic for the intelligent terminal.

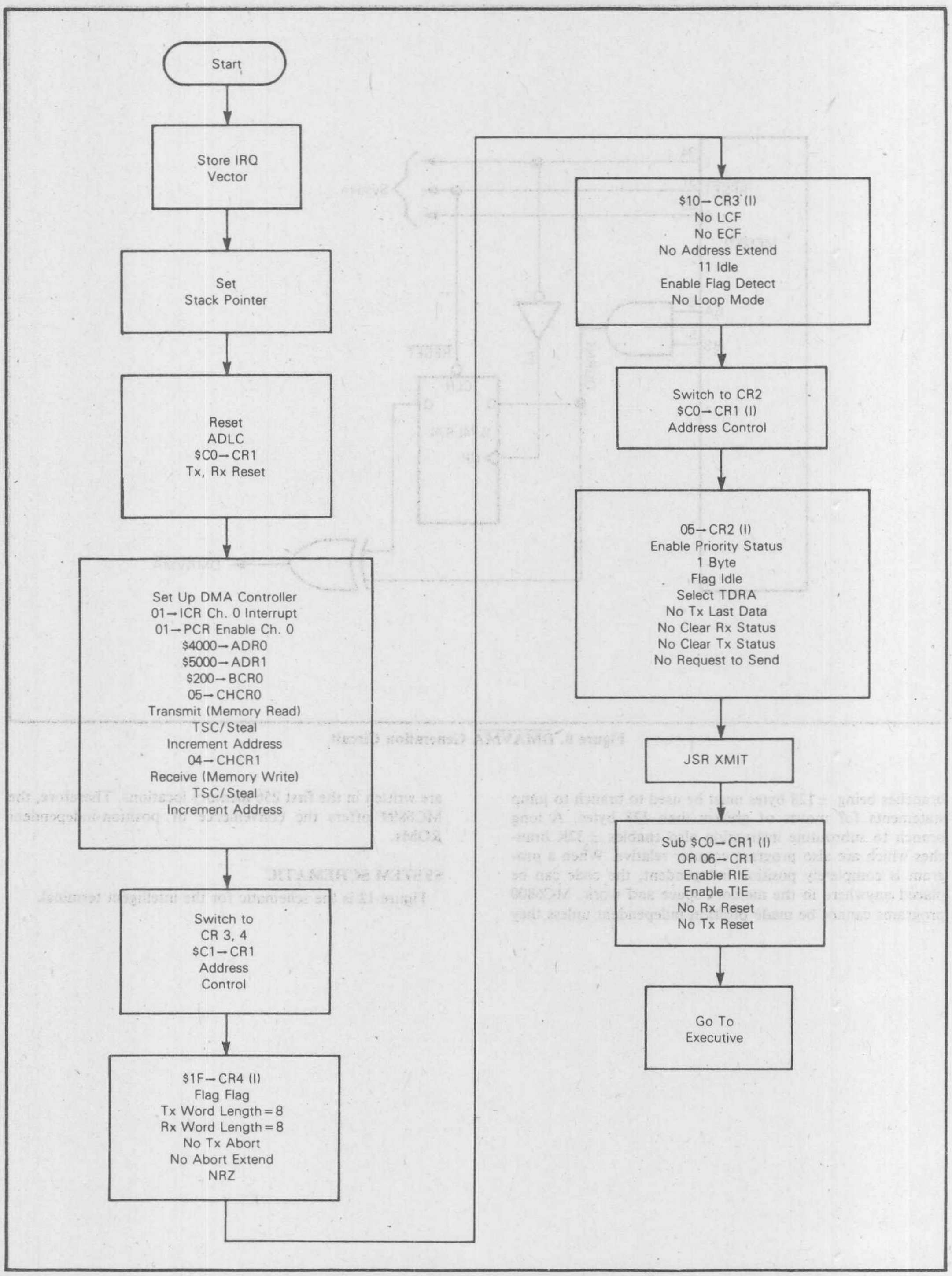


Figure 9. Flowchart of DMA-ADLC Program (Sheet 1 of 9)

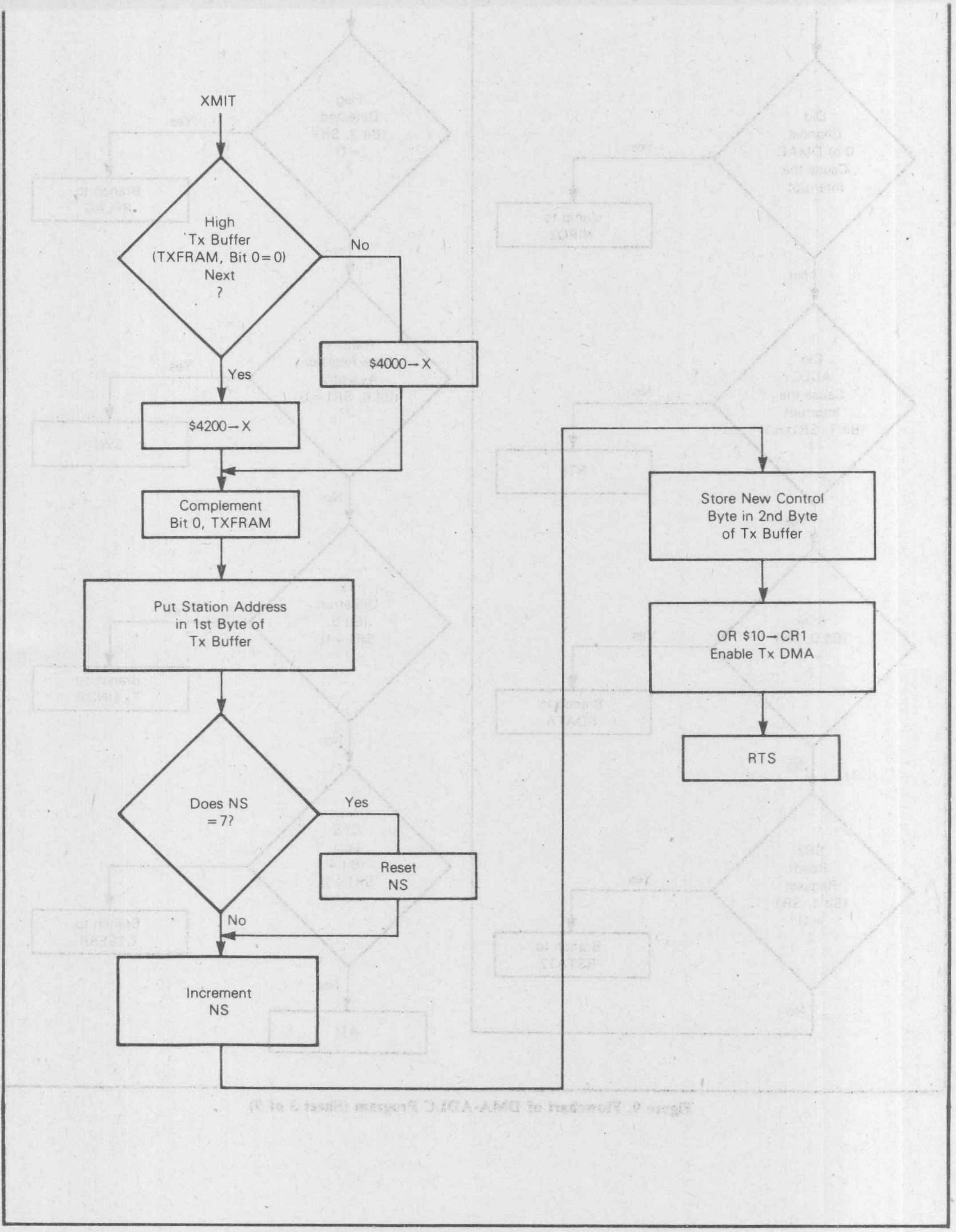

XMIT
High Tx Buffer (TXFRAM, Bit 0=0) Next ?
No
Yes
$4000→X
$4200→X
Complement Bit 0, TXFRAM
Put Station Address in 1st Byte of Tx Buffer
Does NS =7?
Yes
Reset NS
No
Increment NS
Store New Control Byte in 2nd Byte of Tx Buffer
OR $10→CR1 Enable Tx DMA
RTS

Figure 9. Flowchart of DMA-ADLC Program (Sheet 2 of 9)

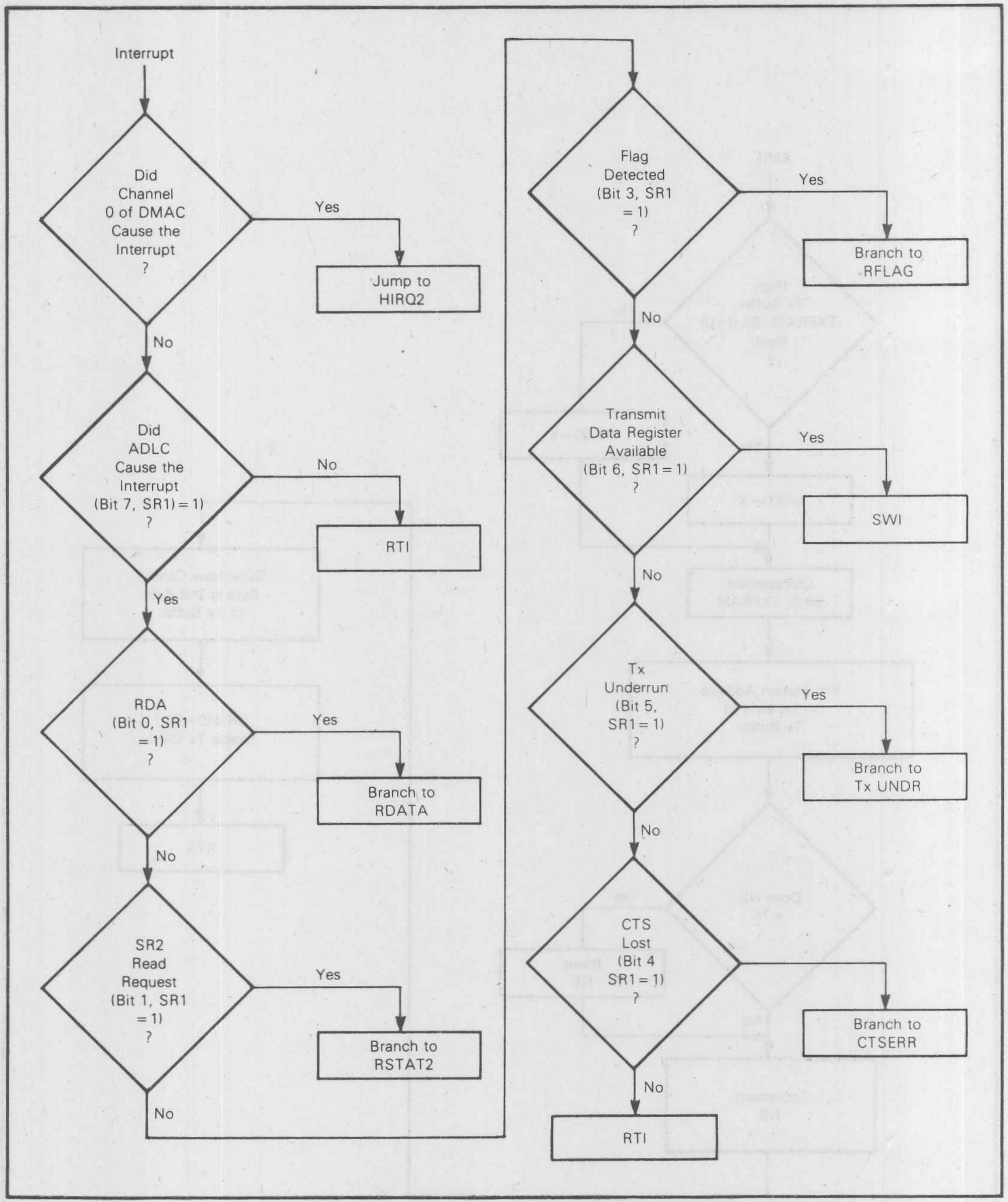

Interrupt
Did Channel 0 of DMAC Cause the Interrupt ?
Yes
Jump to HIRQ2
No
Did ADLC Cause the Interrupt (Bit 7, SR1) = 1) ?
No
RTI
Yes
RDA (Bit 0, SR1 = 1) ?
Yes
Branch to RDATA
No
SR2 Read Request (Bit 1, SR1 = 1) ?
Yes
Branch to RSTAT2
No
Flag Detected (Bit 3, SR1 = 1) ?
Yes
Branch to RFLAG
No
Transmit Data Register Available (Bit 6, SR1 = 1) ?
Yes
SWI
No
Tx Underrun (Bit 5, SR1 = 1) ?
Yes
Branch to Tx UNDR
No
CTS Lost (Bit 4 SR1 = 1) ?
Branch to CTSERR
No
RTI

Figure 9. Flowchart of DMA-ADLC Program (Sheet 3 of 9)

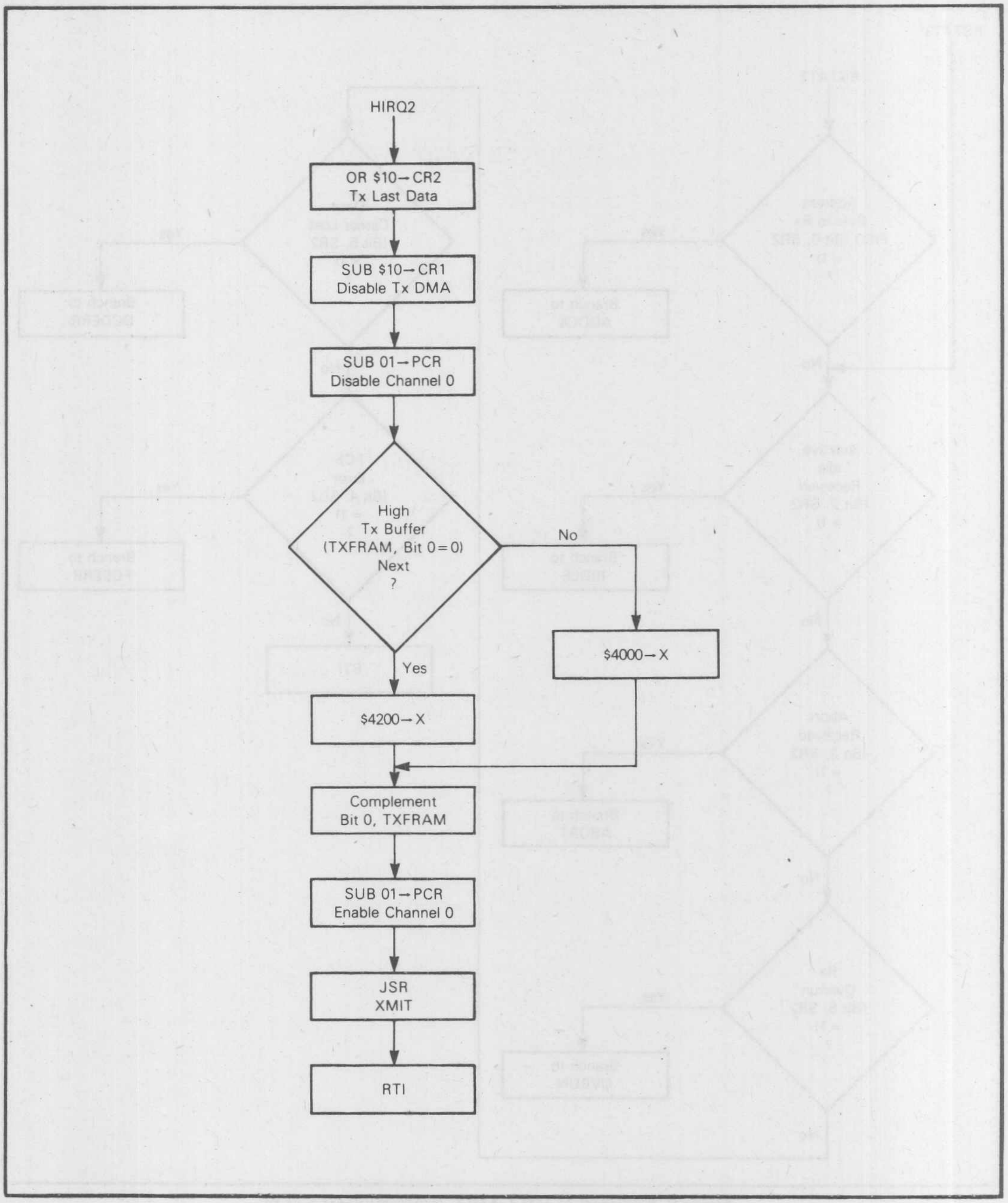


Figure 9. Flowchart of DMA-ADLC Program (Sheet 4 of 9)

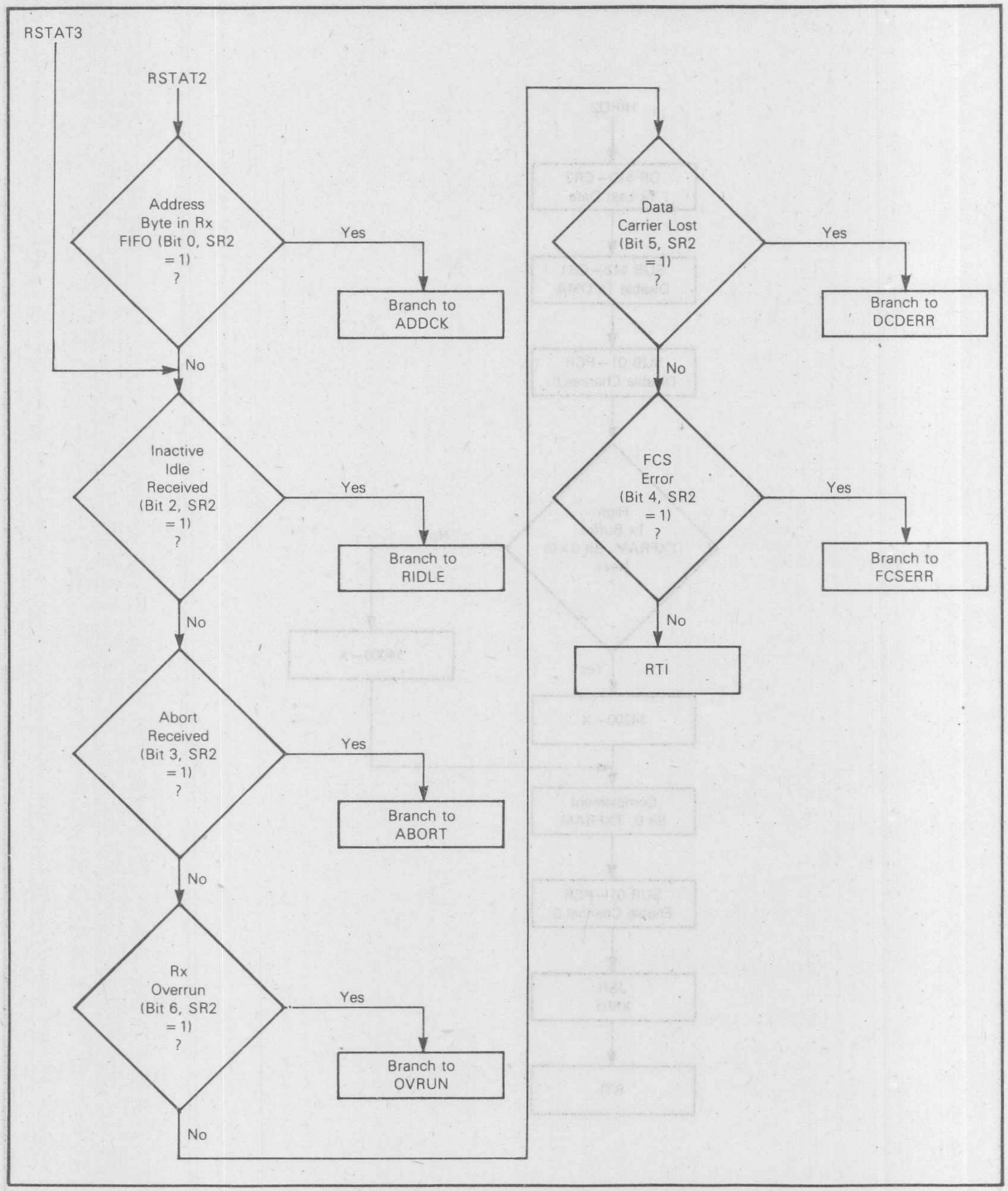

RSTAT3
RSTAT2
Address Byte in Rx FIFO (Bit 0, SR2 = 1) ?
Yes
Branch to ADDCK
No
Inactive Idle Received (Bit 2, SR2 = 1) ?
Yes
Branch to RIDLE
No
Abort Received (Bit 3, SR2 = 1) ?
Yes
Branch to ABORT
No
Rx Overrun (Bit 6, SR2 = 1) ?
Yes
Branch to OVRUN
No
Data Carrier Lost (Bit 5, SR2 = 1) ?
Yes
Branch to DCDERR
No
FCS Error (Bit 4, SR2 = 1) ?
Yes
Branch to FCSERR
No
RTI

Figure 9. Flowchart of DMA-ADLC Program (Sheet 5 of 9)

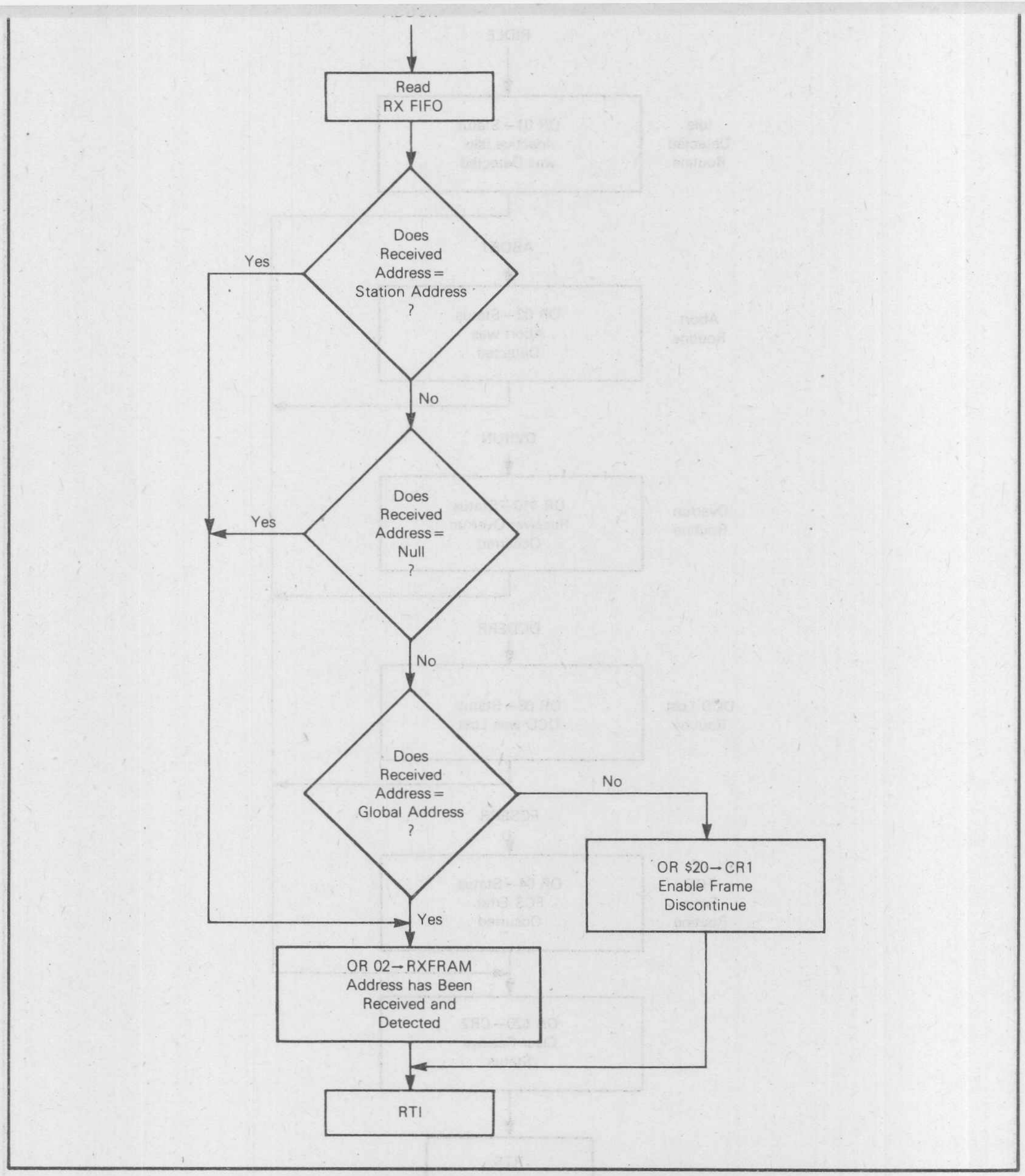

Read RX FIFO
Does Received Address = Station Address ?
Yes
No
Does Received Address = Null ?
Yes
No
Does Received Address = Global Address ?
No
Yes
OR $20→CR1 Enable Frame Discontinue
OR 02→RXFRAM Address has Been Received and Detected
RTI

Figure 9. Flowchart of DMA-ADLC Program (Sheet 6 of 9)

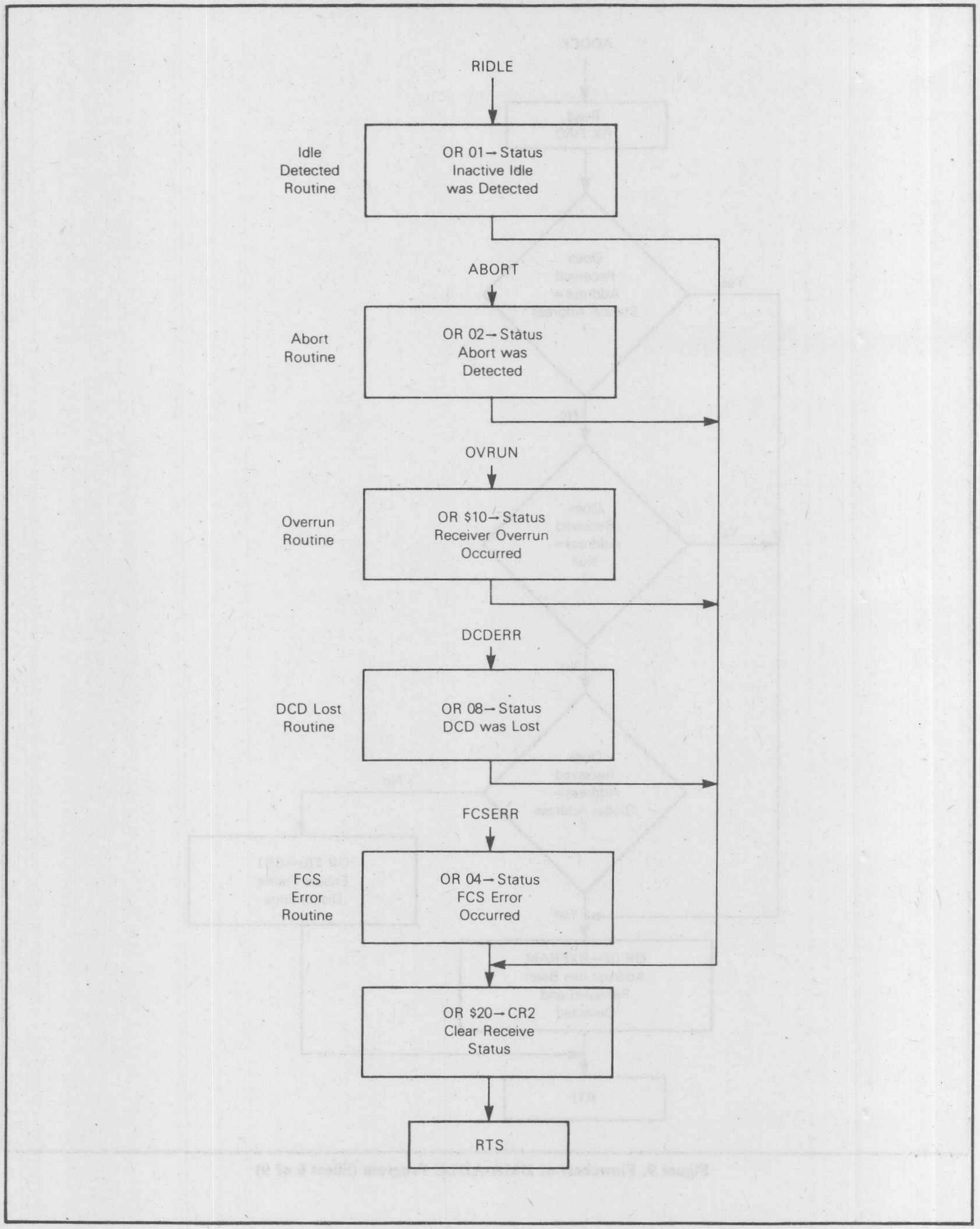

RIDLE
Idle Detected Routine
OR 01→Status Inactive Idle was Detected
ABORT
Abort Routine
OR 02→Status Abort was Detected
OVRUN
Overrun Routine
OR $10→Status Receiver Overrun Occurred
DCDERR
DCD Lost Routine
OR 08→Status DCD was Lost
FCSERR
FCS Error Routine
OR 04→Status FCS Error Occurred
OR $20→CR2 Clear Receive Status
RTS

Figure 9. Flowchart of DMA-ADLC Program (Sheet 7 of 9)

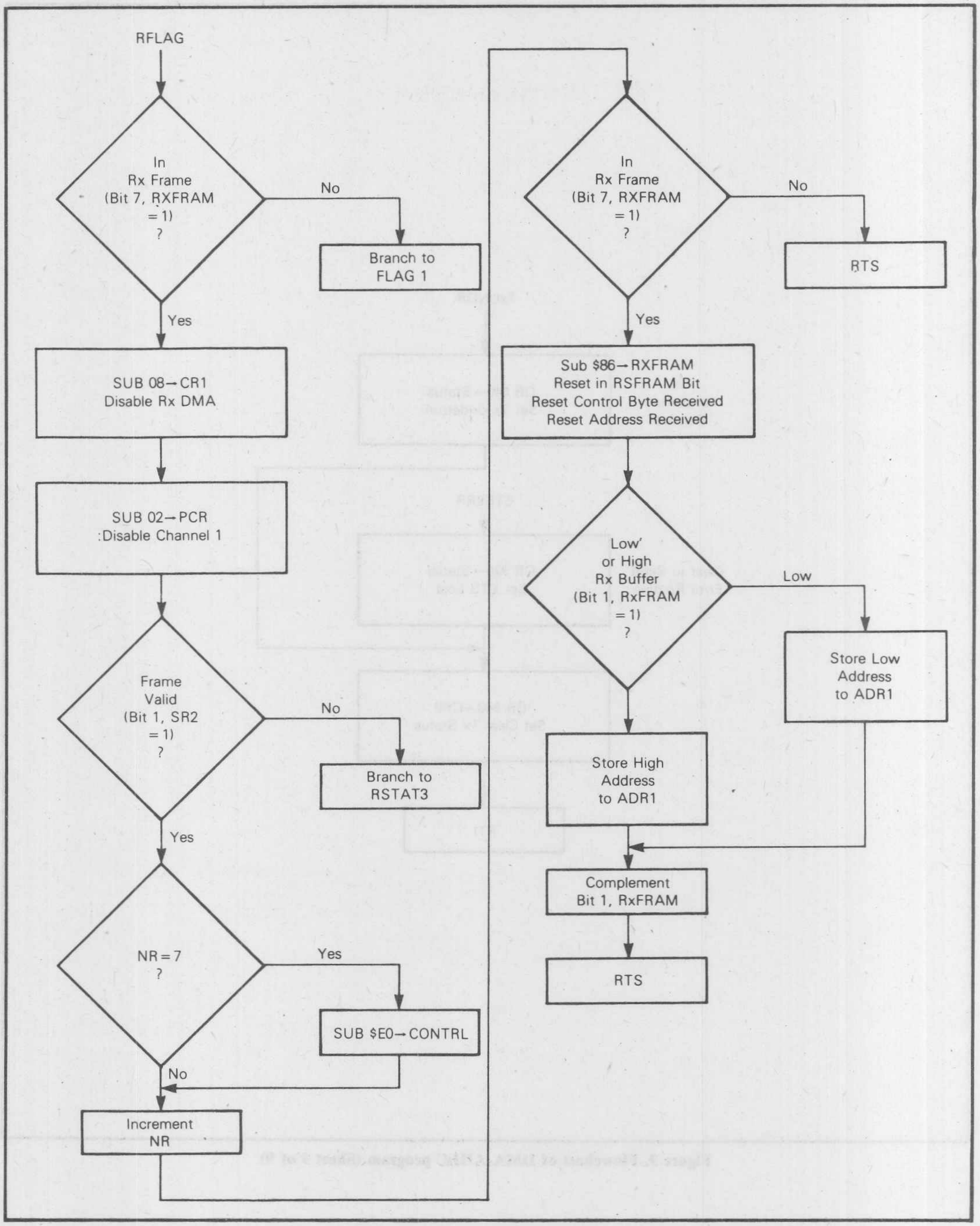

RFLAG
In Rx Frame (Bit 7, RXFRAM = 1) ?
No
Branch to FLAG 1
Yes
SUB 08→CR1 Disable Rx DMA
SUB 02→PCR :Disable Channel 1
Frame Valid (Bit 1, SR2 = 1) ?
No
Branch to RSTAT3
Yes
NR=7 ?
Yes
SUB $E0→CONTRL
No
Increment NR
In Rx Frame (Bit 7, RXFRAM = 1) ?
No
RTS
Yes
Sub $86→RXFRAM Reset in RSFRAM Bit Reset Control Byte Received Reset Address Received
Low' or High Rx Buffer (Bit 1, RxFRAM = 1) ?
Low
Store Low Address to ADR1
Store High Address to ADR1
Complement Bit 1, RxFRAM
RTS

Figure 9. Flowchart of DMA-ADLC Program (Sheet 8 of 9)

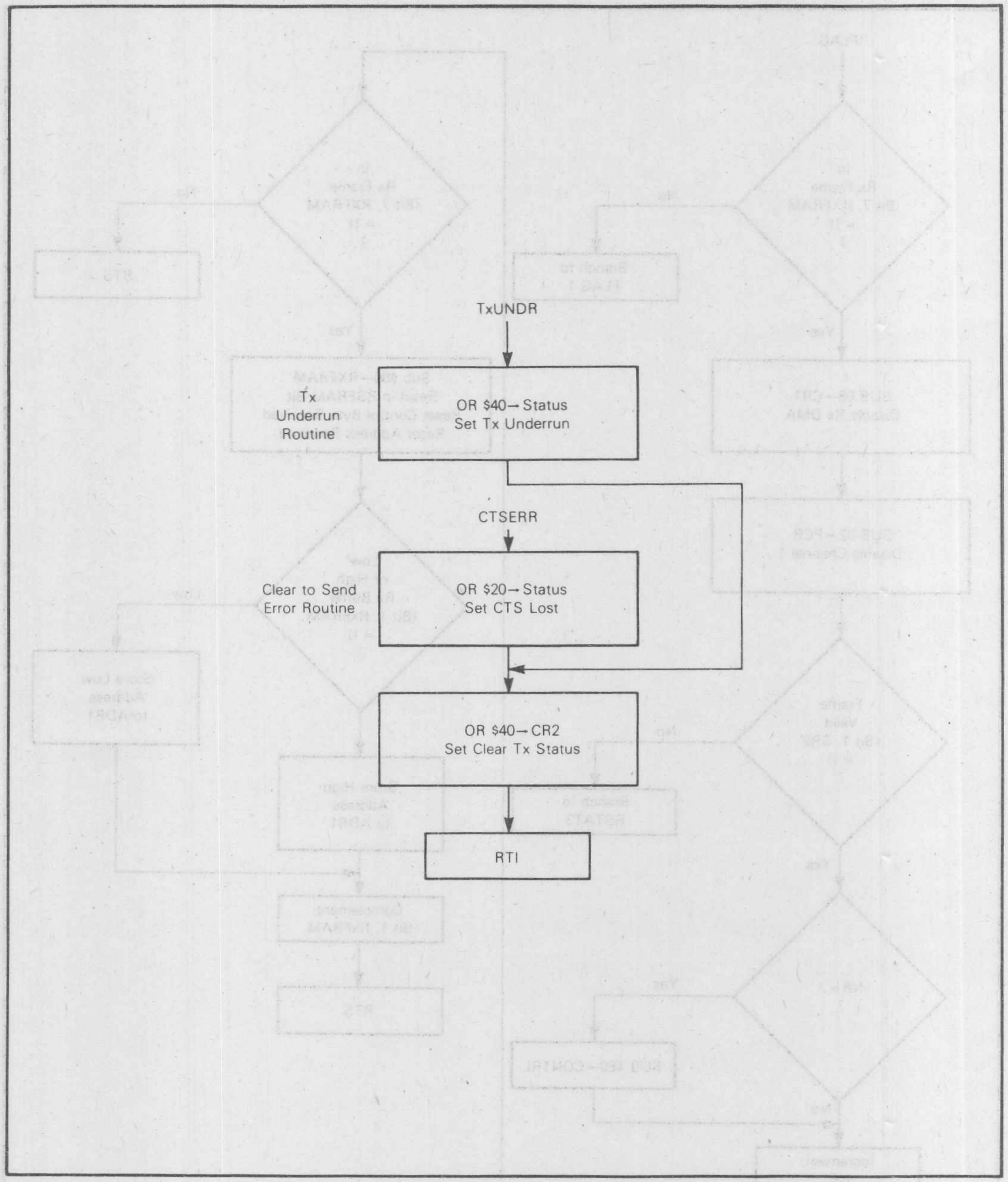
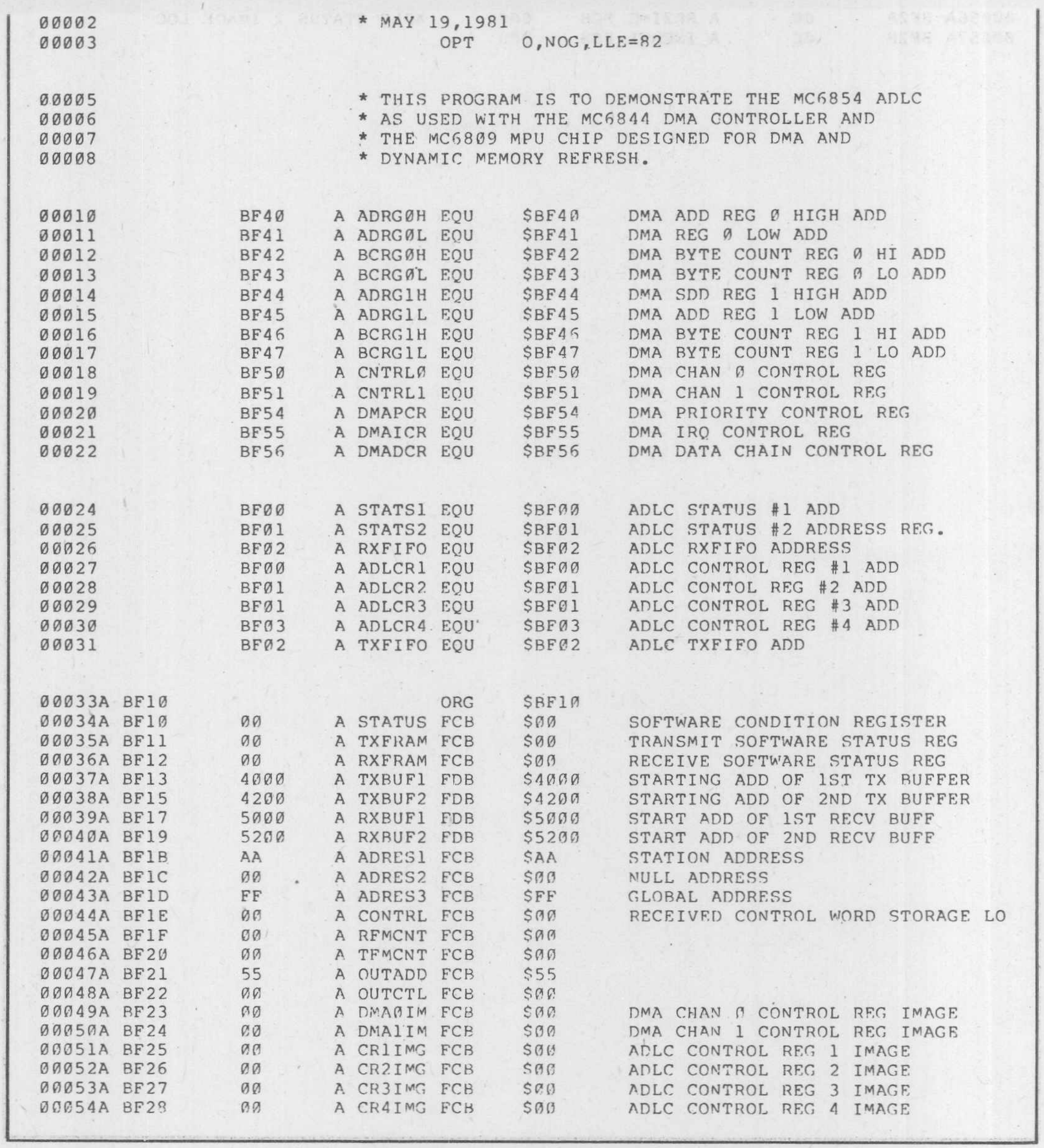
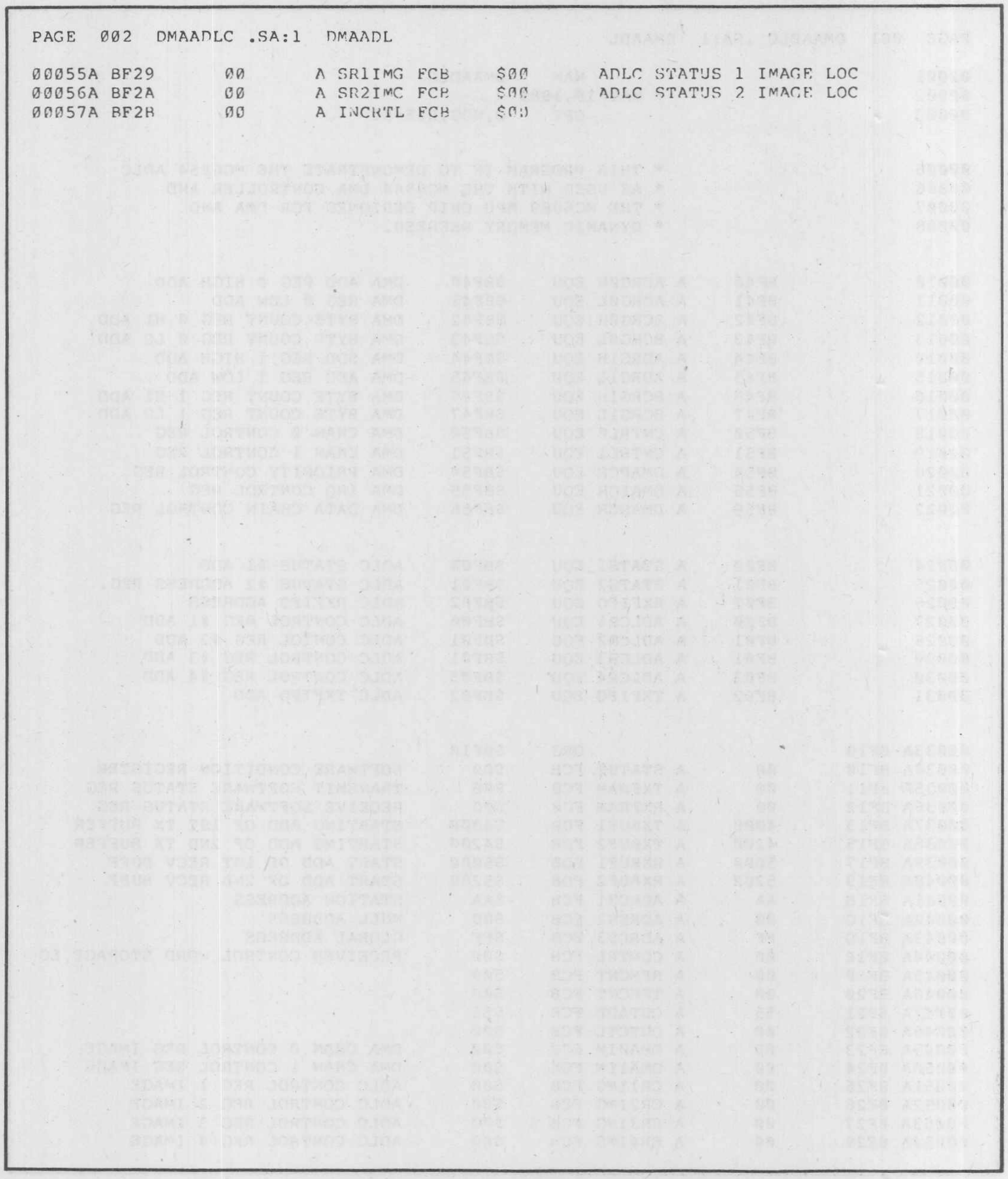


Figure 9. Flowchart of DMA-ADLC program (Sheet 9 of 9)



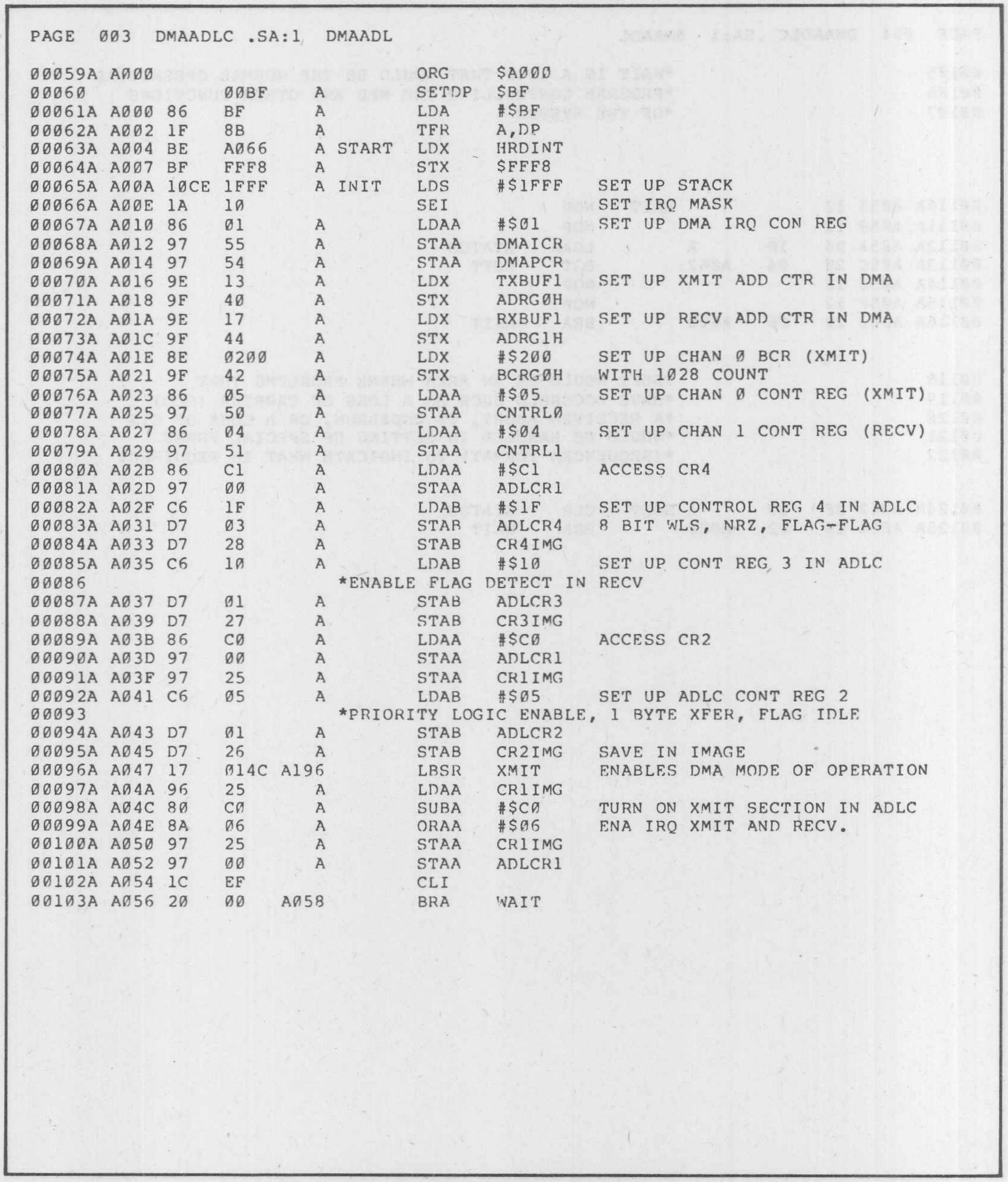
```
ØØØØ2                          * MAY 19,1981
ØØØØ3                                 OPT     O,NOG,LLE=82

ØØØØ5                          * THIS PROGRAM IS TO DEMONSTRATE THE MC6854 ADLC
ØØØØ6                          * AS USED WITH THE MC6844 DMA CONTROLLER AND
ØØØØ7                          * THE MC68Ø9 MPU CHIP DESIGNED FOR DMA AND
ØØØØ8                          * DYNAMIC MEMORY REFRESH.

ØØØ1Ø           BF4Ø    A ADRGØH EQU    $BF4Ø    DMA ADD REG Ø HIGH ADD
ØØØ11           BF41    A ADRGØL EQU    $BF41    DMA REG Ø LOW ADD
ØØØ12           BF42    A BCRGØH EQU    $BF42    DMA BYTE COUNT REG Ø HI ADD
ØØØ13           BF43    A BCRGØL EQU    $BF43    DMA BYTE COUNT REG Ø LO ADD
ØØØ14           BF44    A ADRG1H EQU    $BF44    DMA SDD REG 1 HIGH ADD
ØØØ15           BF45    A ADRG1L EQU    $BF45    DMA ADD REG 1 LOW ADD
ØØØ16           BF46    A BCRG1H EQU    $BF46    DMA BYTE COUNT REG 1 HI ADD
ØØØ17           BF47    A BCRG1L EQU    $BF47    DMA BYTE COUNT REG 1 LO ADD
ØØØ18           BF5Ø    A CNTRLØ EQU    $BF5Ø    DMA CHAN Ø CONTROL REG
ØØØ19           BF51    A CNTRL1 EQU    $BF51    DMA CHAN 1 CONTROL REG
ØØØ2Ø           BF54    A DMAPCR EQU    $BF54    DMA PRIORITY CONTROL REG
ØØØ21           BF55    A DMAICR EQU    $BF55    DMA IRQ CONTROL REG
ØØØ22           BF56    A DMADCR EQU    $BF56    DMA DATA CHAIN CONTROL REG

ØØØ24           BFØØ    A STATS1 EQU    $BFØØ    ADLC STATUS #1 ADD
ØØØ25           BFØ1    A STATS2 EQU    $BFØ1    ADLC STATUS #2 ADDRESS REG.
ØØØ26           BFØ2    A RXFIFO EQU    $BFØ2    ADLC RXFIFO ADDRESS
ØØØ27           BFØØ    A ADLCR1 EQU    $BFØØ    ADLC CONTROL REG #1 ADD
ØØØ28           BFØ1    A ADLCR2 EQU    $BFØ1    ADLC CONTOL REG #2 ADD
ØØØ29           BFØ1    A ADLCR3 EQU    $BFØ1    ADLC CONTROL REG #3 ADD
ØØØ3Ø           BFØ3    A ADLCR4 EQU    $BFØ3    ADLC CONTROL REG #4 ADD
ØØØ31           BFØ2    A TXFIFO EQU    $BFØ2    ADLC TXFIFO ADD

ØØØ33A BF1Ø                      ORG    $BF1Ø
ØØØ34A BF1Ø     ØØ      A STATUS FCB    $ØØ      SOFTWARE CONDITION REGISTER
ØØØ35A BF11     ØØ      A TXFRAM FCB    $ØØ      TRANSMIT SOFTWARE STATUS REG
ØØØ36A BF12     ØØ      A RXFRAM FCB    $ØØ      RECEIVE SOFTWARE STATUS REG
ØØØ37A BF13     4ØØØ    A TXBUF1 FDB    $4ØØØ    STARTING ADD OF 1ST TX BUFFER
ØØØ38A BF15     42ØØ    A TXBUF2 FDB    $42ØØ    STARTING ADD OF 2ND TX BUFFER
ØØØ39A BF17     5ØØØ    A RXBUF1 FDB    $5ØØØ    START ADD OF 1ST RECV BUFF
ØØØ4ØA BF19     52ØØ    A RXBUF2 FDB    $52ØØ    START ADD OF 2ND RECV BUFF
ØØØ41A BF1B     AA      A ADRES1 FCB    $AA      STATION ADDRESS
ØØØ42A BF1C     ØØ      A ADRES2 FCB    $ØØ      NULL ADDRESS
ØØØ43A BF1D     FF      A ADRES3 FCB    $FF      GLOBAL ADDRESS
ØØØ44A BF1E     ØØ      A CONTRL FCB    $ØØ      RECEIVED CONTROL WORD STORAGE LO
ØØØ45A BF1F     ØØ      A RFMCNT FCB    $ØØ
ØØØ46A BF2Ø     ØØ      A TFMCNT FCB    $ØØ
ØØØ47A BF21     55      A OUTADD FCB    $55
ØØØ48A BF22     ØØ      A OUTCTL FCB    $ØØ
ØØØ49A BF23     ØØ      A DMAØIM FCB    $ØØ      DMA CHAN Ø CONTROL REG IMAGE
ØØØ5ØA BF24     ØØ      A DMA1IM FCB    $ØØ      DMA CHAN 1 CONTROL REG IMAGE
ØØØ51A BF25     ØØ      A CR1IMG FCB    $ØØ      ADLC CONTROL REG 1 IMAGE
ØØØ52A BF26     ØØ      A CR2IMG FCB    $ØØ      ADLC CONTROL REG 2 IMAGE
ØØØ53A BF27     ØØ      A CR3IMG FCB    $ØØ      ADLC CONTROL REG 3 IMAGE
ØØØ54A BF28     ØØ      A CR4IMG FCB    $ØØ      ADLC CONTROL REG 4 IMAGE
```

Figure 10. Priority Mode Program Listing (Sheet 1 of 11)



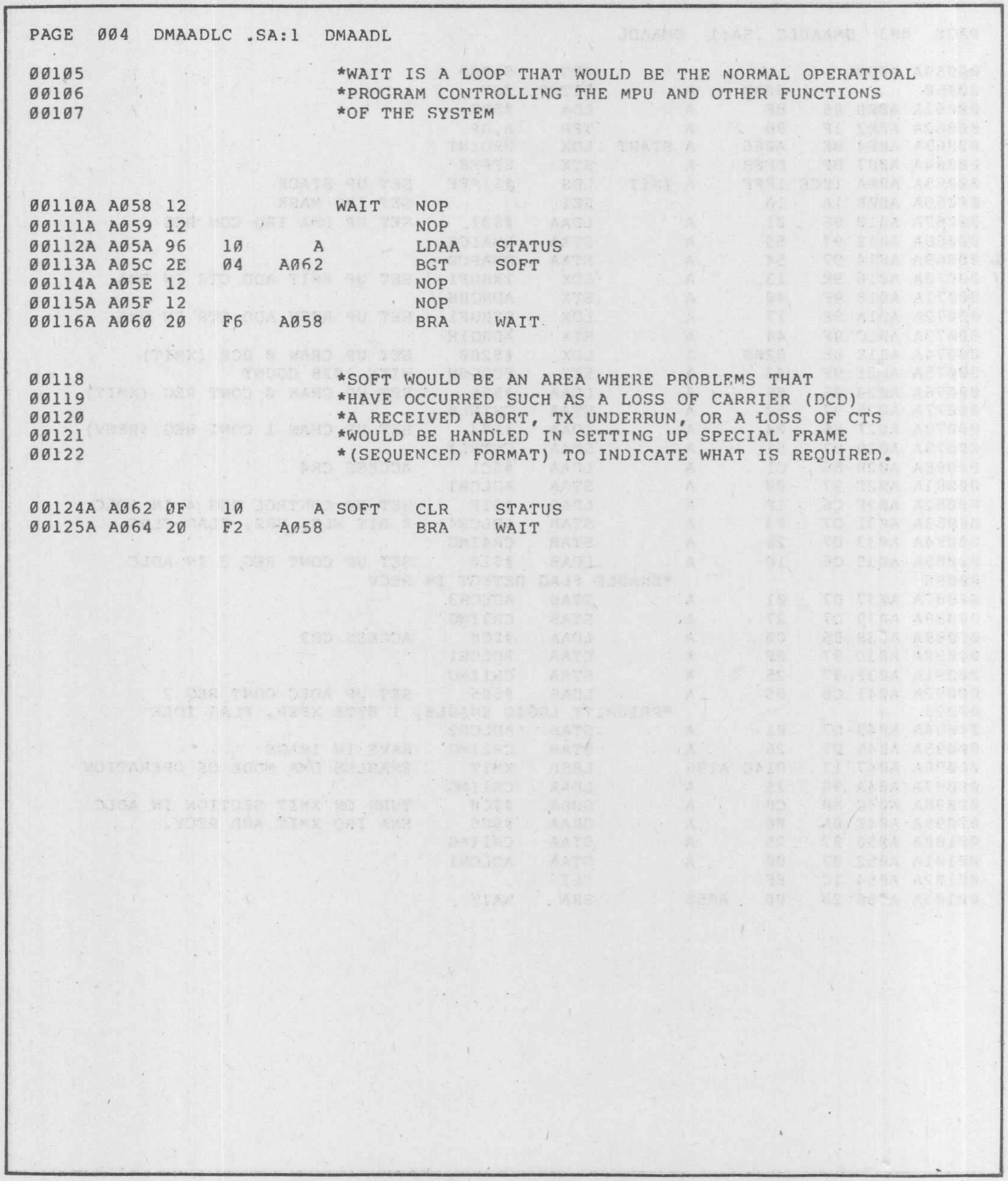
```
PAGE  002  DMAADLC .SA:1   DMAADL

00055A BF29       00      A SR1IMG FCB      $00      ADLC STATUS 1 IMAGE LOC
00056A BF2A       00      A SR2IMC FCB      $00      ADLC STATUS 2 IMAGE LOC
00057A BF2B       00      A INCRTL FCB      $00
```

Figure 10. Priority Mode Program Listing (Sheet 2 of 11)



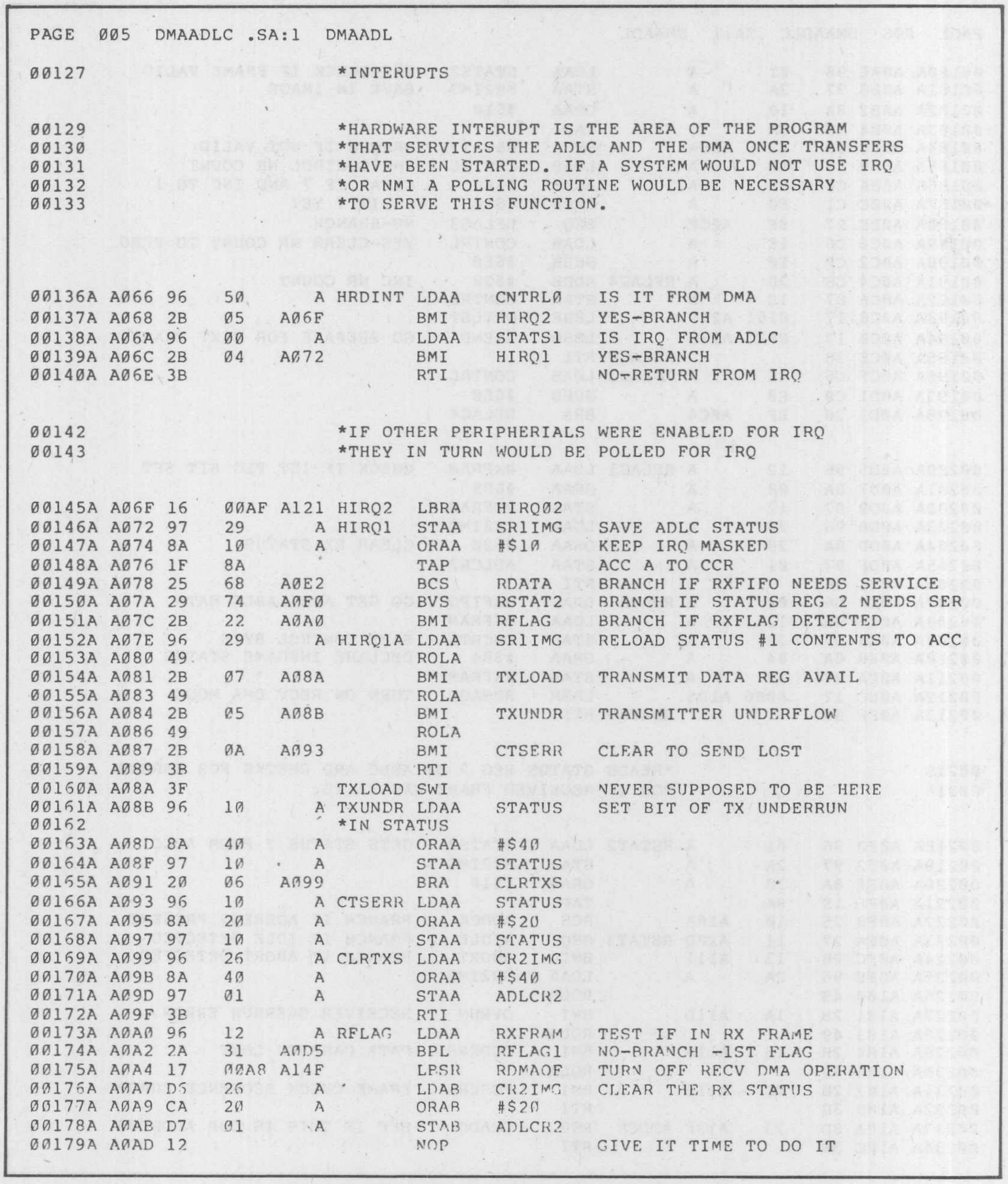
```
PAGE  ØØ3  DMAADLC .SA:1  DMAADL

ØØØ59A AØØØ                    ORG    $AØØØ
ØØØ6Ø          ØØBF      A     SETDP  $BF
ØØØ61A AØØØ 86   BF      A     LDA    #$BF
ØØØ62A AØØ2 1F   8B      A     TFR    A,DP
ØØØ63A AØØ4 BE   AØ66    A START  LDX    HRDINT
ØØØ64A AØØ7 BF   FFF8    A     STX    $FFF8
ØØØ65A AØØA 1ØCE 1FFF    A INIT   LDS    #$1FFF   SET UP STACK
ØØØ66A AØØE 1A   1Ø            SEI             SET IRQ MASK
ØØØ67A AØ1Ø 86   Ø1      A     LDAA   #$Ø1     SET UP DMA IRQ CON REG
ØØØ68A AØ12 97   55      A     STAA   DMAICR
ØØØ69A AØ14 97   54      A     STAA   DMAPCR
ØØØ7ØA AØ16 9E   13      A     LDX    TXBUF1   SET UP XMIT ADD CTR IN DMA
ØØØ71A AØ18 9F   4Ø      A     STX    ADRGØH
ØØØ72A AØ1A 9E   17      A     LDX    RXBUF1   SET UP RECV ADD CTR IN DMA
ØØØ73A AØ1C 9F   44      A     STX    ADRG1H
ØØØ74A AØ1E 8E   Ø2ØØ    A     LDX    #$2ØØ    SET UP CHAN Ø BCR (XMIT)
ØØØ75A AØ21 9F   42      A     STX    BCRGØH   WITH 1Ø28 COUNT
ØØØ76A AØ23 86   Ø5      A     LDAA   #$Ø5     SET UP CHAN Ø CONT REG (XMIT)
ØØØ77A AØ25 97   5Ø      A     STAA   CNTRLØ
ØØØ78A AØ27 86   Ø4      A     LDAA   #$Ø4     SET UP CHAN 1 CONT REG (RECV)
ØØØ79A AØ29 97   51      A     STAA   CNTRL1
ØØØ8ØA AØ2B 86   C1      A     LDAA   #$C1     ACCESS CR4
ØØØ81A AØ2D 97   ØØ      A     STAA   ADLCR1
ØØØ82A AØ2F C6   1F      A     LDAB   #$1F     SET UP CONTROL REG 4 IN ADLC
ØØØ83A AØ31 D7   Ø3      A     STAB   ADLCR4   8 BIT WLS, NRZ, FLAG-FLAG
ØØØ84A AØ33 D7   28      A     STAB   CR4IMG
ØØØ85A AØ35 C6   1Ø      A     LDAB   #$1Ø     SET UP CONT REG 3 IN ADLC
ØØØ86                      *ENABLE FLAG DETECT IN RECV
ØØØ87A AØ37 D7   Ø1      A     STAB   ADLCR3
ØØØ88A AØ39 D7   27      A     STAB   CR3IMG
ØØØ89A AØ3B 86   CØ      A     LDAA   #$CØ     ACCESS CR2
ØØØ9ØA AØ3D 97   ØØ      A     STAA   ADLCR1
ØØØ91A AØ3F 97   25      A     STAA   CR1IMG
ØØØ92A AØ41 C6   Ø5      A     LDAB   #$Ø5     SET UP ADLC CONT REG 2
ØØØ93                      *PRIORITY LOGIC ENABLE, 1 BYTE XFER, FLAG IDLE
ØØØ94A AØ43 D7   Ø1      A     STAB   ADLCR2
ØØØ95A AØ45 D7   26      A     STAB   CR2IMG   SAVE IN IMAGE
ØØØ96A AØ47 17   Ø14C AØ196    LBSR   XMIT     ENABLES DMA MODE OF OPERATION
ØØØ97A AØ4A 96   25      A     LDAA   CR1IMG
ØØØ98A AØ4C 8Ø   CØ      A     SUBA   #$CØ     TURN ON XMIT SECTION IN ADLC
ØØØ99A AØ4E 8A   Ø6      A     ORAA   #$Ø6     ENA IRQ XMIT AND RECV.
ØØ1ØØA AØ5Ø 97   25      A     STAA   CR1IMG
ØØ1Ø1A AØ52 97   ØØ      A     STAA   ADLCR1
ØØ1Ø2A AØ54 1C   EF            CLI
ØØ1Ø3A AØ56 2Ø   ØØ   AØ58     BRA    WAIT
```

Figure 10. Priority Mode Program Listing (Sheet 3 of 11)



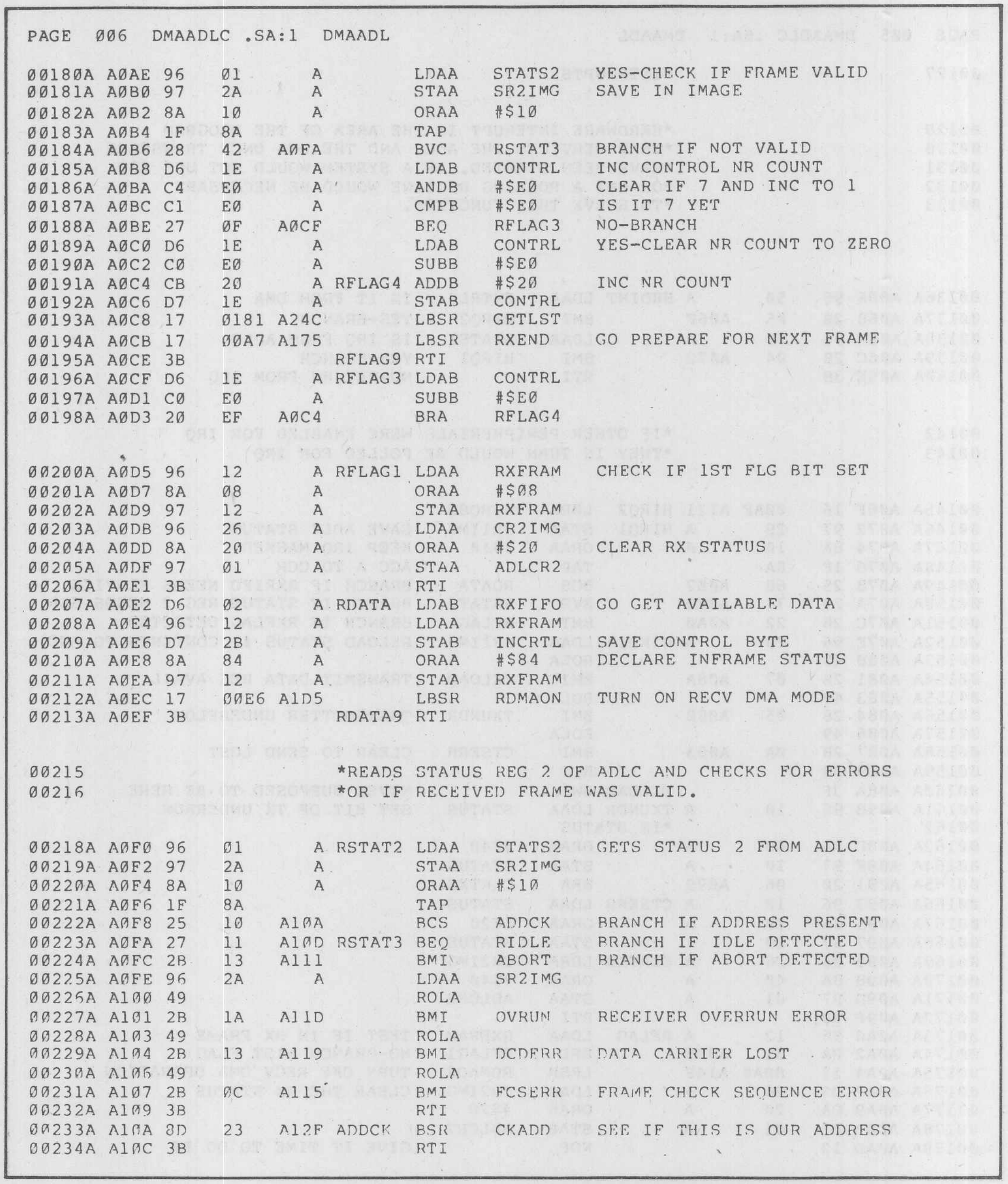
```
PAGE  ØØ4  DMAADLC .SA:1  DMAADL

ØØ1Ø5                        *WAIT IS A LOOP THAT WOULD BE THE NORMAL OPERATIOAL
ØØ1Ø6                        *PROGRAM CONTROLLING THE MPU AND OTHER FUNCTIONS
ØØ1Ø7                        *OF THE SYSTEM



ØØ11ØA AØ58 12               WAIT    NOP
ØØ111A AØ59 12                       NOP
ØØ112A AØ5A 96    1Ø      A          LDAA    STATUS
ØØ113A AØ5C 2E    Ø4   AØ62          BGT     SOFT
ØØ114A AØ5E 12                       NOP
ØØ115A AØ5F 12                       NOP
ØØ116A AØ6Ø 2Ø    F6   AØ58          BRA     WAIT


ØØ118                        *SOFT WOULD BE AN AREA WHERE PROBLEMS THAT
ØØ119                        *HAVE OCCURRED SUCH AS A LOSS OF CARRIER (DCD)
ØØ12Ø                        *A RECEIVED ABORT, TX UNDERRUN, OR A LOSS OF CTS
ØØ121                        *WOULD BE HANDLED IN SETTING UP SPECIAL FRAME
ØØ122                        *(SEQUENCED FORMAT) TO INDICATE WHAT IS REQUIRED.


ØØ124A AØ62 ØF    1Ø      A SOFT     CLR     STATUS
ØØ125A AØ64 2Ø    F2   AØ58          BRA     WAIT
```

Figure 10. Priority Mode Program Listing (Sheet 4 of 11)



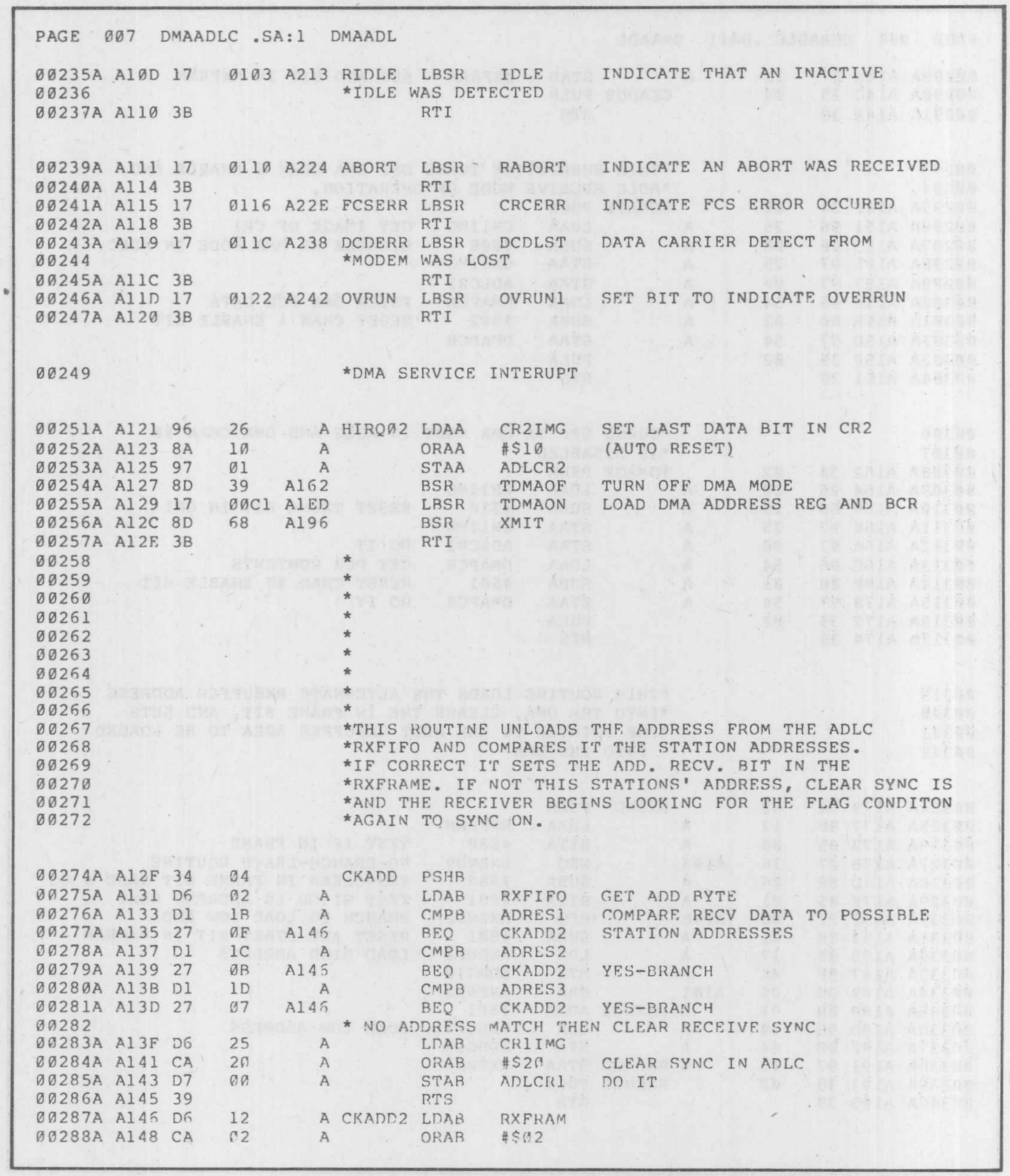
```
PAGE  ØØ5  DMAADLC .SA:1  DMAADL

ØØ127                       *INTERUPTS

ØØ129                       *HARDWARE INTERUPT IS THE AREA OF THE PROGRAM
ØØ13Ø                       *THAT SERVICES THE ADLC AND THE DMA ONCE TRANSFERS
ØØ131                       *HAVE BEEN STARTED. IF A SYSTEM WOULD NOT USE IRQ
ØØ132                       *OR NMI A POLLING ROUTINE WOULD BE NECESSARY
ØØ133                       *TO SERVE THIS FUNCTION.

ØØ136A AØ66 96   5Ø      A HRDINT LDAA   CNTRLØ    IS IT FROM DMA
ØØ137A AØ68 2B   Ø5   AØ6F        BMI    HIRQ2     YES-BRANCH
ØØ138A AØ6A 96   ØØ      A        LDAA   STATS1    IS IRQ FROM ADLC
ØØ139A AØ6C 2B   Ø4   AØ72        BMI    HIRQ1     YES-BRANCH
ØØ14ØA AØ6E 3B                    RTI              NO-RETURN FROM IRQ

ØØ142                       *IF OTHER PERIPHERIALS WERE ENABLED FOR IRQ
ØØ143                       *THEY IN TURN WOULD BE POLLED FOR IRQ

ØØ145A AØ6F 16   ØØAF A121 HIRQ2  LBRA   HIRQØ2
ØØ146A AØ72 97   29      A HIRQ1  STAA   SR1IMG    SAVE ADLC STATUS
ØØ147A AØ74 8A   1Ø      A        ORAA   #$1Ø      KEEP IRQ MASKED
ØØ148A AØ76 1F   8A               TAP              ACC A TO CCR
ØØ149A AØ78 25   68   AØE2        BCS    RDATA     BRANCH IF RXFIFO NEEDS SERVICE
ØØ15ØA AØ7A 29   74   AØFØ        BVS    RSTAT2    BRANCH IF STATUS REG 2 NEEDS SER
ØØ151A AØ7C 2B   22   AØAØ        BMI    RFLAG     BRANCH IF RXFLAG DETECTED
ØØ152A AØ7E 96   29      A HIRQ1A LDAA   SR1IMG    RELOAD STATUS #1 CONTENTS TO ACC
ØØ153A AØ8Ø 49                   ROLA
ØØ154A AØ81 2B   Ø7   AØ8A        BMI    TXLOAD    TRANSMIT DATA REG AVAIL
ØØ155A AØ83 49                   ROLA
ØØ156A AØ84 2B   Ø5   AØ8B        BMI    TXUNDR    TRANSMITTER UNDERFLOW
ØØ157A AØ86 49                   ROLA
ØØ158A AØ87 2B   ØA   AØ93        BMI    CTSERR    CLEAR TO SEND LOST
ØØ159A AØ89 3B                    RTI
ØØ16ØA AØ8A 3F             TXLOAD SWI              NEVER SUPPOSED TO BE HERE
ØØ161A AØ8B 96   1Ø      A TXUNDR LDAA   STATUS    SET BIT OF TX UNDERRUN
ØØ162                       *IN STATUS
ØØ163A AØ8D 8A   4Ø      A        ORAA   #$4Ø
ØØ164A AØ8F 97   1Ø      A        STAA   STATUS
ØØ165A AØ91 2Ø   Ø6   AØ99        BRA    CLRTXS
ØØ166A AØ93 96   1Ø      A CTSERR LDAA   STATUS
ØØ167A AØ95 8A   2Ø      A        ORAA   #$2Ø
ØØ168A AØ97 97   1Ø      A        STAA   STATUS
ØØ169A AØ99 96   26      A CLRTXS LDAA   CR2IMG
ØØ17ØA AØ9B 8A   4Ø      A        ORAA   #$4Ø
ØØ171A AØ9D 97   Ø1      A        STAA   ADLCR2
ØØ172A AØ9F 3B                    RTI
ØØ173A AØAØ 96   12      A RFLAG  LDAA   RXFRAM    TEST IF IN RX FRAME
ØØ174A AØA2 2A   31   AØD5        BPL    RFLAG1    NO-BRANCH -1ST FLAG
ØØ175A AØA4 17   ØØA8 A14F        LBSR   RDMAOF    TURN OFF RECV DMA OPERATION
ØØ176A AØA7 D6   26      A        LDAB   CR2IMG    CLEAR THE RX STATUS
ØØ177A AØA9 CA   2Ø      A        ORAB   #$2Ø
ØØ178A AØAB D7   Ø1      A        STAB   ADLCR2
ØØ179A AØAD 12                    NOP              GIVE IT TIME TO DO IT
```

Figure 10. Priority Mode Program Listing (Sheet 5 of 11)



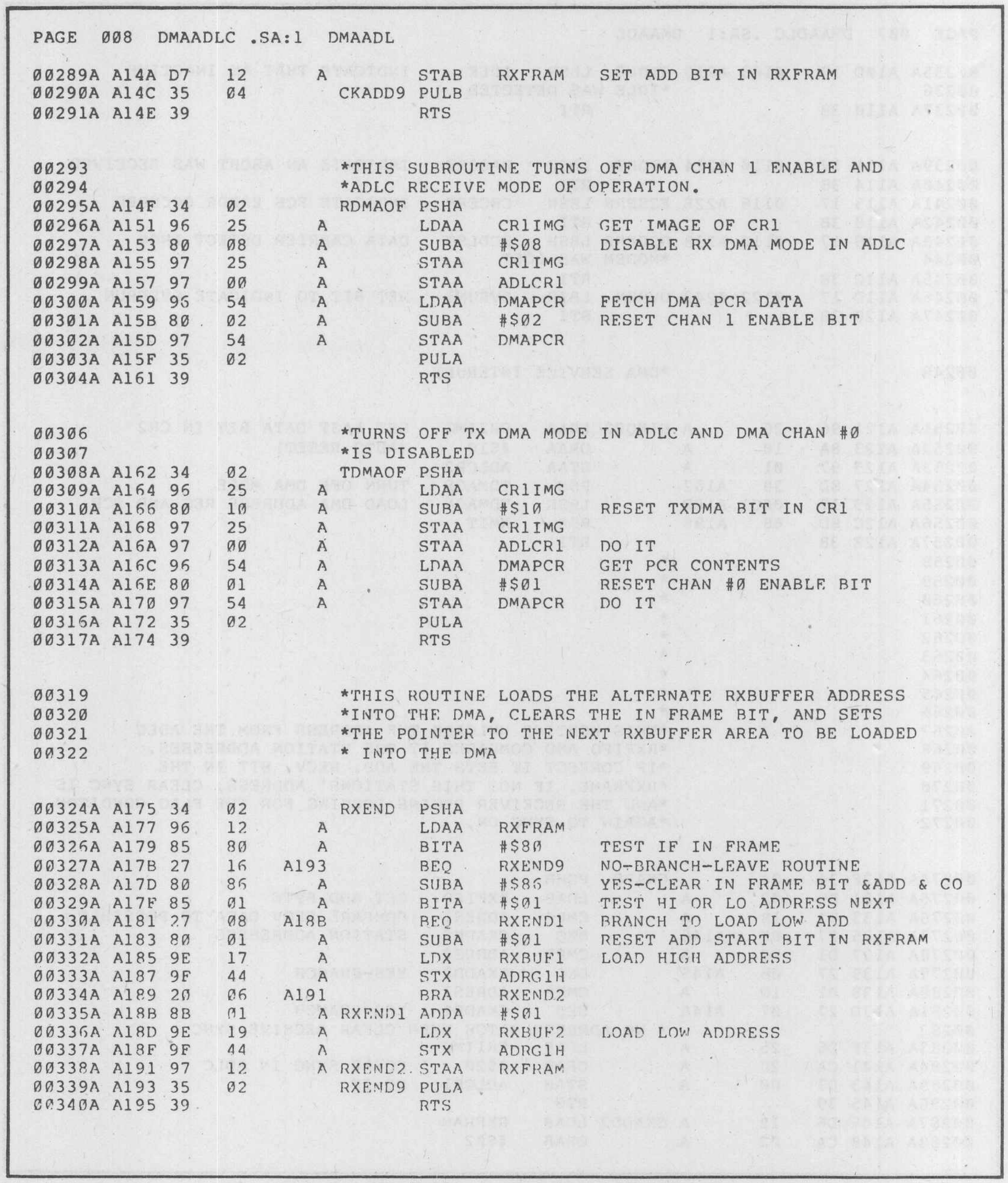
```
PAGE  ØØ6  DMAADLC .SA:1  DMAADL

ØØ18ØA AØAE 96   Ø1      A       LDAA   STATS2    YES-CHECK IF FRAME VALID
ØØ181A AØBØ 97   2A      A       STAA   SR2IMG    SAVE IN IMAGE
ØØ182A AØB2 8A   1Ø      A       ORAA   #$1Ø
ØØ183A AØB4 1F   8A              TAP
ØØ184A AØB6 28   42   AØFA       BVC    RSTAT3    BRANCH IF NOT VALID
ØØ185A AØB8 D6   1E      A       LDAB   CONTRL    INC CONTROL NR COUNT
ØØ186A AØBA C4   EØ      A       ANDB   #$EØ      CLEAR IF 7 AND INC TO 1
ØØ187A AØBC C1   EØ      A       CMPB   #$EØ      IS IT 7 YET
ØØ188A AØBE 27   ØF   AØCF       BEQ    RFLAG3    NO-BRANCH
ØØ189A AØCØ D6   1E      A       LDAB   CONTRL    YES-CLEAR NR COUNT TO ZERO
ØØ19ØA AØC2 CØ   EØ      A       SUBB   #$EØ
ØØ191A AØC4 CB   2Ø      A RFLAG4 ADDB  #$2Ø      INC NR COUNT
ØØ192A AØC6 D7   1E      A       STAB   CONTRL
ØØ193A AØC8 17   Ø181 A24C       LBSR   GETLST
ØØ194A AØCB 17   ØØA7 A175       LBSR   RXEND     GO PREPARE FOR NEXT FRAME
ØØ195A AØCE 3B          RFLAG9   RTI
ØØ196A AØCF D6   1E      A RFLAG3 LDAB  CONTRL
ØØ197A AØD1 CØ   EØ      A       SUBB   #$EØ
ØØ198A AØD3 2Ø   EF   AØC4       BRA    RFLAG4

ØØ2ØØA AØD5 96   12      A RFLAG1 LDAA  RXFRAM    CHECK IF 1ST FLG BIT SET
ØØ2Ø1A AØD7 8A   Ø8      A       ORAA   #$Ø8
ØØ2Ø2A AØD9 97   12      A       STAA   RXFRAM
ØØ2Ø3A AØDB 96   26      A       LDAA   CR2IMG
ØØ2Ø4A AØDD 8A   2Ø      A       ORAA   #$2Ø      CLEAR RX STATUS
ØØ2Ø5A AØDF 97   Ø1      A       STAA   ADLCR2
ØØ2Ø6A AØE1 3B                   RTI
ØØ2Ø7A AØE2 D6   Ø2      A RDATA  LDAB  RXFIFO    GO GET AVAILABLE DATA
ØØ2Ø8A AØE4 96   12      A       LDAA   RXFRAM
ØØ2Ø9A AØE6 D7   2B      A       STAB   INCRTL    SAVE CONTROL BYTE
ØØ21ØA AØE8 8A   84      A       ORAA   #$84      DECLARE INFRAME STATUS
ØØ211A AØEA 97   12      A       STAA   RXFRAM
ØØ212A AØEC 17   ØØE6 A1D5       LBSR   RDMAON    TURN ON RECV DMA MODE
ØØ213A AØEF 3B          RDATA9   RTI

ØØ215                   *READS STATUS REG 2 OF ADLC AND CHECKS FOR ERRORS
ØØ216                   *OR IF RECEIVED FRAME WAS VALID.

ØØ218A AØFØ 96   Ø1      A RSTAT2 LDAA  STATS2    GETS STATUS 2 FROM ADLC
ØØ219A AØF2 97   2A      A       STAA   SR2IMG
ØØ22ØA AØF4 8A   1Ø      A       ORAA   #$1Ø
ØØ221A AØF6 1F   8A              TAP
ØØ222A AØF8 25   1Ø   A1ØA       BCS    ADDCK     BRANCH IF ADDRESS PRESENT
ØØ223A AØFA 27   11   A1ØD RSTAT3 BEQ   RIDLE     BRANCH IF IDLE DETECTED
ØØ224A AØFC 2B   13   A111       BMI    ABORT     BRANCH IF ABORT DETECTED
ØØ225A AØFE 96   2A      A       LDAA   SR2IMG
ØØ226A A1ØØ 49                   ROLA
ØØ227A A1Ø1 2B   1A   A11D       BMI    OVRUN     RECEIVER OVERRUN ERROR
ØØ228A A1Ø3 49                   ROLA
ØØ229A A1Ø4 2B   13   A119       BMI    DCDERR    DATA CARRIER LOST
ØØ23ØA A1Ø6 49                   ROLA
ØØ231A A1Ø7 2B   ØC   A115       BMI    FCSERR    FRAME CHECK SEQUENCE ERROR
ØØ232A A1Ø9 3B                   RTI
ØØ233A A1ØA 8D   23   A12F ADDCK BSR    CKADD     SEE IF THIS IS OUR ADDRESS
ØØ234A A1ØC 3B                   RTI
```

Figure 10. Priority Mode Program Listing (Sheet 6 of 11)



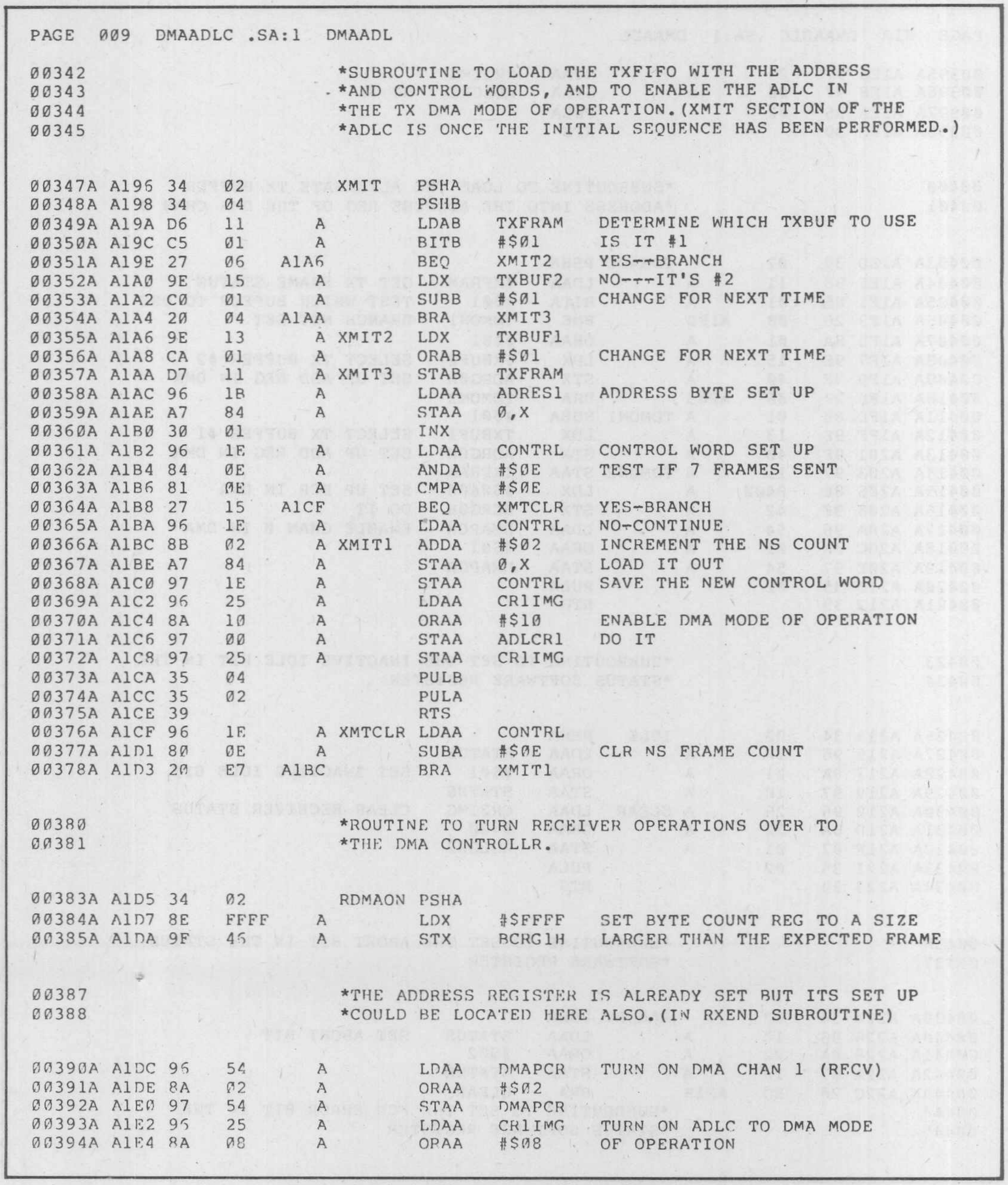
```
PAGE  ØØ7  DMAADLC .SA:1  DMAADL

ØØ235A A1ØD 17   Ø1Ø3 A213 RIDLE  LBSR   IDLE     INDICATE THAT AN INACTIVE
ØØ236                      *IDLE WAS DETECTED
ØØ237A A11Ø 3B                    RTI

ØØ239A A111 17   Ø11Ø A224 ABORT  LBSR   RABORT   INDICATE AN ABORT WAS RECEIVED
ØØ24ØA A114 3B                    RTI
ØØ241A A115 17   Ø116 A22E FCSERR LBSR   CRCERR   INDICATE FCS ERROR OCCURED
ØØ242A A118 3B                    RTI
ØØ243A A119 17   Ø11C A238 DCDERR LBSR   DCDLST   DATA CARRIER DETECT FROM
ØØ244                      *MODEM WAS LOST
ØØ245A A11C 3B                    RTI
ØØ246A A11D 17   Ø122 A242 OVRUN  LBSR   OVRUN1   SET BIT TO INDICATE OVERRUN
ØØ247A A12Ø 3B                    RTI

ØØ249                      *DMA SERVICE INTERUPT

ØØ251A A121 96   26     A  HIRQØ2 LDAA   CR2IMG   SET LAST DATA BIT IN CR2
ØØ252A A123 8A   1Ø     A         ORAA   #$1Ø     (AUTO RESET)
ØØ253A A125 97   Ø1     A         STAA   ADLCR2
ØØ254A A127 8D   39   A162        BSR    TDMAOF   TURN OFF DMA MODE
ØØ255A A129 17   ØØC1 A1ED        LBSR   TDMAON   LOAD DMA ADDRESS REG AND BCR
ØØ256A A12C 8D   68   A196        BSR    XMIT
ØØ257A A12E 3B                    RTI
ØØ258                      *
ØØ259                      *
ØØ26Ø                      *
ØØ261                      *
ØØ262                      *
ØØ263                      *
ØØ264                      *
ØØ265                      *
ØØ266                      *
ØØ267                      *THIS ROUTINE UNLOADS THE ADDRESS FROM THE ADLC
ØØ268                      *RXFIFO AND COMPARES IT THE STATION ADDRESSES.
ØØ269                      *IF CORRECT IT SETS THE ADD. RECV. BIT IN THE
ØØ27Ø                      *RXFRAME. IF NOT THIS STATIONS' ADDRESS, CLEAR SYNC IS
ØØ271                      *AND THE RECEIVER BEGINS LOOKING FOR THE FLAG CONDITON
ØØ272                      *AGAIN TO SYNC ON.

ØØ274A A12F 34   Ø4        CKADD  PSHB
ØØ275A A131 D6   Ø2     A         LDAB   RXFIFO   GET ADD BYTE
ØØ276A A133 D1   1B     A         CMPB   ADRES1   COMPARE RECV DATA TO POSSIBLE
ØØ277A A135 27   ØF   A146        BEQ    CKADD2   STATION ADDRESSES
ØØ278A A137 D1   1C     A         CMPB   ADRES2
ØØ279A A139 27   ØB   A146        BEQ    CKADD2   YES-BRANCH
ØØ28ØA A13B D1   1D     A         CMPB   ADRES3
ØØ281A A13D 27   Ø7   A146        BEQ    CKADD2   YES-BRANCH
ØØ282                      * NO ADDRESS MATCH THEN CLEAR RECEIVE SYNC
ØØ283A A13F D6   25     A         LDAB   CR1IMG
ØØ284A A141 CA   2Ø     A         ORAB   #$2Ø     CLEAR SYNC IN ADLC
ØØ285A A143 D7   ØØ     A         STAB   ADLCR1   DO IT
ØØ286A A145 39                    RTS
ØØ287A A146 D6   12     A  CKADD2 LDAB   RXFRAM
ØØ288A A148 CA   Ø2     A         ORAB   #$Ø2
```

Figure 10. Priority Mode Program Listing (Sheet 7 of 11)



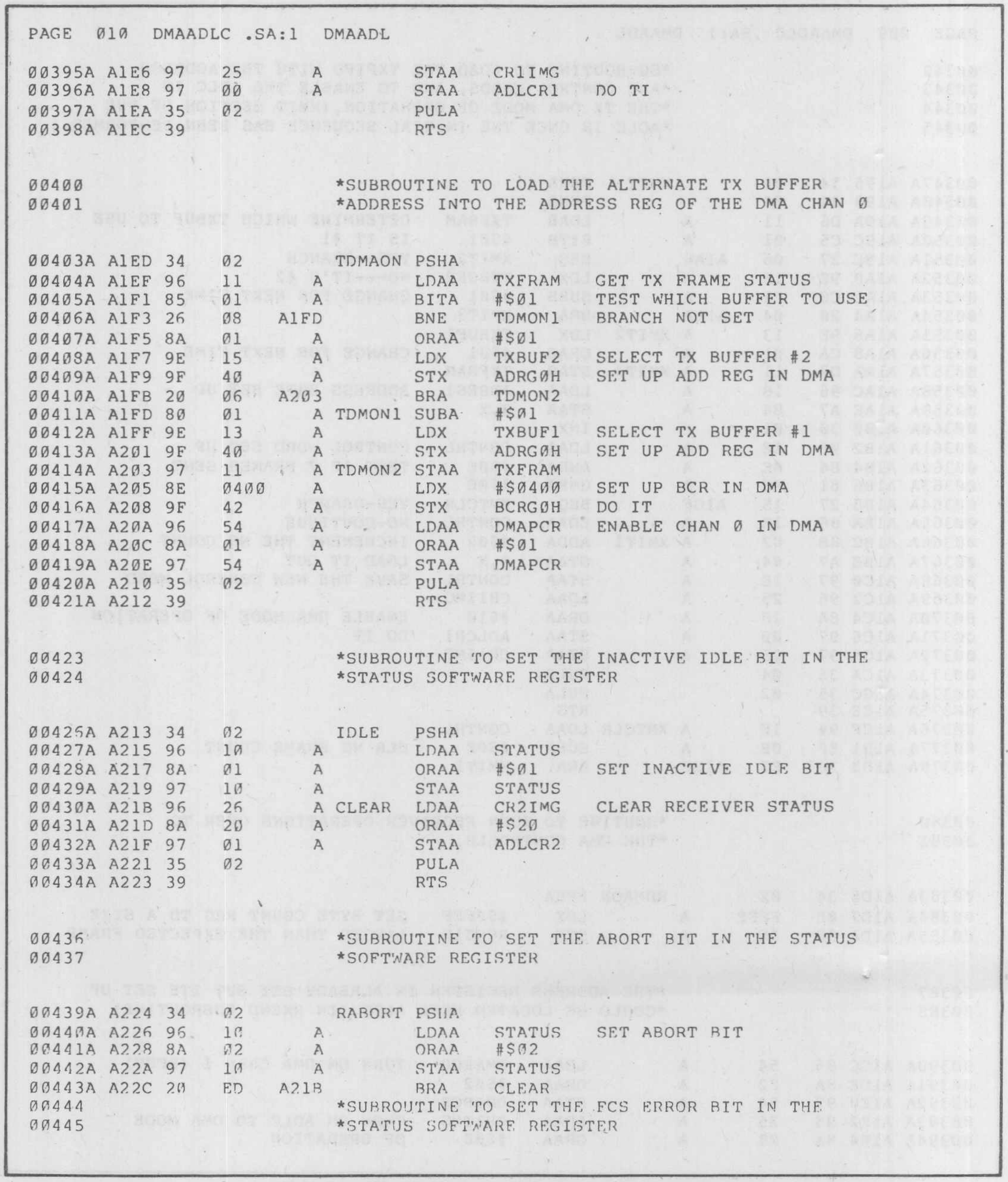
```
PAGE  008  DMAADLC .SA:1  DMAADL

00289A A14A D7   12      A        STAB   RXFRAM    SET ADD BIT IN RXFRAM
00290A A14C 35   04         CKADD9 PULB
00291A A14E 39                    RTS


00293                       *THIS SUBROUTINE TURNS OFF DMA CHAN 1 ENABLE AND
00294                       *ADLC RECEIVE MODE OF OPERATION.
00295A A14F 34   02         RDMAOF PSHA
00296A A151 96   25      A        LDAA   CR1IMG    GET IMAGE OF CR1
00297A A153 80   08      A        SUBA   #$08      DISABLE RX DMA MODE IN ADLC
00298A A155 97   25      A        STAA   CR1IMG
00299A A157 97   00      A        STAA   ADLCR1
00300A A159 96   54      A        LDAA   DMAPCR    FETCH DMA PCR DATA
00301A A15B 80   02      A        SUBA   #$02      RESET CHAN 1 ENABLE BIT
00302A A15D 97   54      A        STAA   DMAPCR
00303A A15F 35   02               PULA
00304A A161 39                    RTS


00306                       *TURNS OFF TX DMA MODE IN ADLC AND DMA CHAN #0
00307                       *IS DISABLED
00308A A162 34   02         TDMAOF PSHA
00309A A164 96   25      A        LDAA   CR1IMG
00310A A166 80   10      A        SUBA   #$10      RESET TXDMA BIT IN CR1
00311A A168 97   25      A        STAA   CR1IMG
00312A A16A 97   00      A        STAA   ADLCR1    DO IT
00313A A16C 96   54      A        LDAA   DMAPCR    GET PCR CONTENTS
00314A A16E 80   01      A        SUBA   #$01      RESET CHAN #0 ENABLE BIT
00315A A170 97   54      A        STAA   DMAPCR    DO IT
00316A A172 35   02               PULA
00317A A174 39                    RTS


00319                       *THIS ROUTINE LOADS THE ALTERNATE RXBUFFER ADDRESS
00320                       *INTO THE DMA, CLEARS THE IN FRAME BIT, AND SETS
00321                       *THE POINTER TO THE NEXT RXBUFFER AREA TO BE LOADED
00322                       * INTO THE DMA


00324A A175 34   02         RXEND  PSHA
00325A A177 96   12      A        LDAA   RXFRAM
00326A A179 85   80      A        BITA   #$80      TEST IF IN FRAME
00327A A17B 27   16   A193        BEQ    RXEND9    NO-BRANCH-LEAVE ROUTINE
00328A A17D 80   86      A        SUBA   #$86      YES-CLEAR IN FRAME BIT &ADD & CO
00329A A17F 85   01      A        BITA   #$01      TEST HI OR LO ADDRESS NEXT
00330A A181 27   08   A18B        BEQ    RXEND1    BRANCH TO LOAD LOW ADD
00331A A183 80   01      A        SUBA   #$01      RESET ADD START BIT IN RXFRAM
00332A A185 9E   17      A        LDX    RXBUF1    LOAD HIGH ADDRESS
00333A A187 9F   44      A        STX    ADRG1H
00334A A189 20   06   A191        BRA    RXEND2
00335A A18B 8B   01      A RXEND1 ADDA   #$01
00336A A18D 9E   19      A        LDX    RXBUF2    LOAD LOW ADDRESS
00337A A18F 9F   44      A        STX    ADRG1H
00338A A191 97   12      A RXEND2 STAA   RXFRAM
00339A A193 35   02         RXEND9 PULA
00340A A195 39                    RTS
```

Figure 10. Priority Mode Program Listing (Sheet 8 of 11)



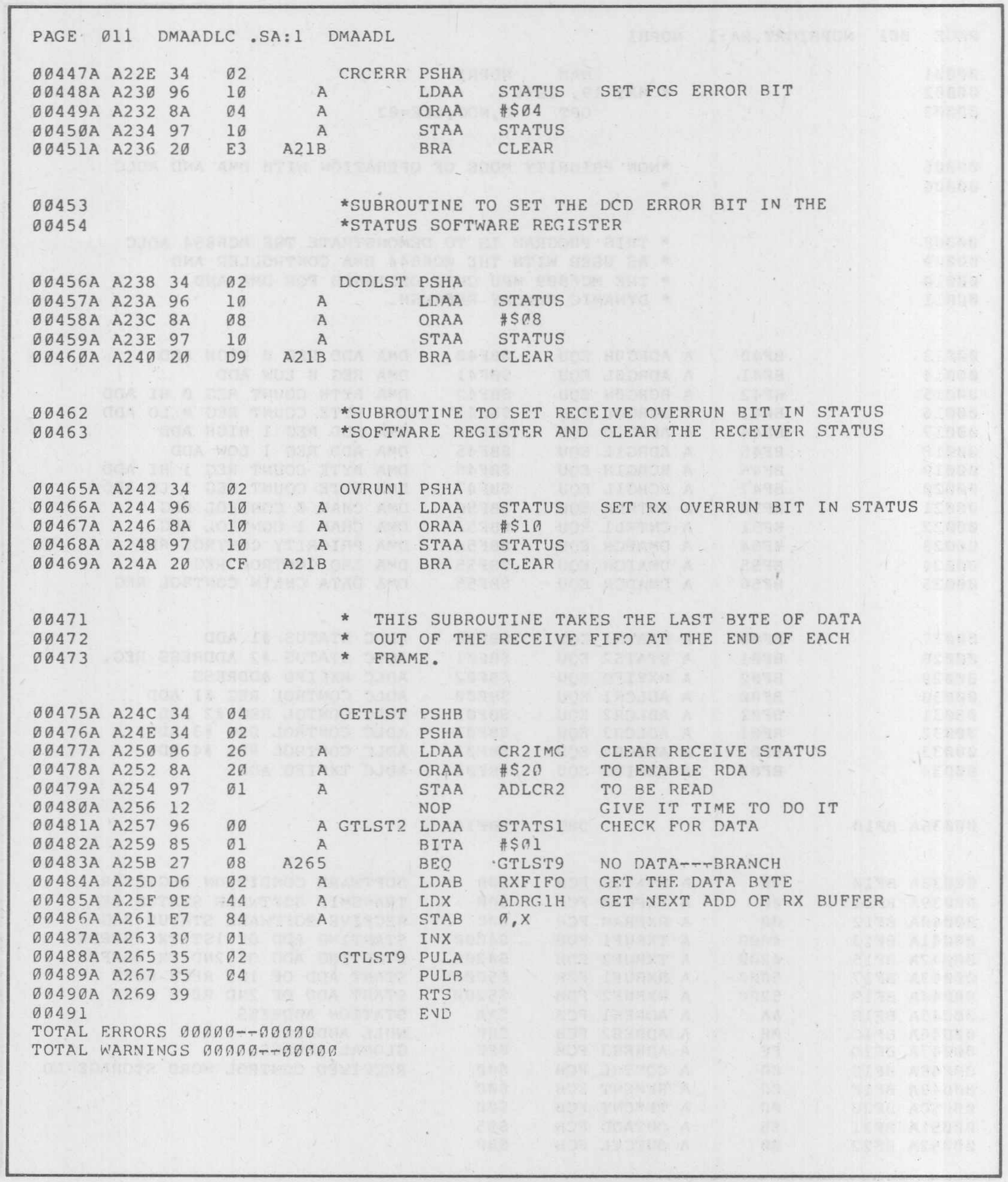
```
PAGE  ØØ9  DMAADLC .SA:1  DMAADL

ØØ342                         *SUBROUTINE TO LOAD THE TXFIFO WITH THE ADDRESS
ØØ343                         *AND CONTROL WORDS, AND TO ENABLE THE ADLC IN
ØØ344                         *THE TX DMA MODE OF OPERATION.(XMIT SECTION OF THE
ØØ345                         *ADLC IS ONCE THE INITIAL SEQUENCE HAS BEEN PERFORMED.)


ØØ347A A196 34   Ø2        XMIT   PSHA
ØØ348A A198 34   Ø4               PSHB
ØØ349A A19A D6   11      A        LDAB   TXFRAM    DETERMINE WHICH TXBUF TO USE
ØØ35ØA A19C C5   Ø1      A        BITB   #$Ø1      IS IT #1
ØØ351A A19E 27   Ø6   A1A6        BEQ    XMIT2     YES--BRANCH
ØØ352A A1AØ 9E   15      A        LDX    TXBUF2    NO---IT'S #2
ØØ353A A1A2 CØ   Ø1      A        SUBB   #$Ø1      CHANGE FOR NEXT TIME
ØØ354A A1A4 2Ø   Ø4   A1AA        BRA    XMIT3
ØØ355A A1A6 9E   13      A XMIT2  LDX    TXBUF1
ØØ356A A1A8 CA   Ø1      A        ORAB   #$Ø1      CHANGE FOR NEXT TIME
ØØ357A A1AA D7   11      A XMIT3  STAB   TXFRAM
ØØ358A A1AC 96   1B      A        LDAA   ADRES1    ADDRESS BYTE SET UP
ØØ359A A1AE A7   84      A        STAA   Ø,X
ØØ36ØA A1BØ 3Ø   Ø1               INX
ØØ361A A1B2 96   1E      A        LDAA   CONTRL    CONTROL WORD SET UP
ØØ362A A1B4 84   ØE      A        ANDA   #$ØE      TEST IF 7 FRAMES SENT
ØØ363A A1B6 81   ØE      A        CMPA   #$ØE
ØØ364A A1B8 27   15   A1CF        BEQ    XMTCLR    YES-BRANCH
ØØ365A A1BA 96   1E      A        LDAA   CONTRL    NO-CONTINUE
ØØ366A A1BC 8B   Ø2      A XMIT1  ADDA   #$Ø2      INCREMENT THE NS COUNT
ØØ367A A1BE A7   84      A        STAA   Ø,X       LOAD IT OUT
ØØ368A A1CØ 97   1E      A        STAA   CONTRL    SAVE THE NEW CONTROL WORD
ØØ369A A1C2 96   25      A        LDAA   CR1IMG
ØØ37ØA A1C4 8A   1Ø      A        ORAA   #$1Ø      ENABLE DMA MODE OF OPERATION
ØØ371A A1C6 97   ØØ      A        STAA   ADLCR1    DO IT
ØØ372A A1C8 97   25      A        STAA   CR1IMG
ØØ373A A1CA 35   Ø4               PULB
ØØ374A A1CC 35   Ø2               PULA
ØØ375A A1CE 39                    RTS
ØØ376A A1CF 96   1E      A XMTCLR LDAA   CONTRL
ØØ377A A1D1 8Ø   ØE      A        SUBA   #$ØE      CLR NS FRAME COUNT
ØØ378A A1D3 2Ø   E7   A1BC        BRA    XMIT1


ØØ38Ø                         *ROUTINE TO TURN RECEIVER OPERATIONS OVER TO
ØØ381                         *THE DMA CONTROLLR.


ØØ383A A1D5 34   Ø2        RDMAON PSHA
ØØ384A A1D7 8E   FFFF    A        LDX    #$FFFF    SET BYTE COUNT REG TO A SIZE
ØØ385A A1DA 9F   46      A        STX    BCRG1H    LARGER THAN THE EXPECTED FRAME


ØØ387                         *THE ADDRESS REGISTER IS ALREADY SET BUT ITS SET UP
ØØ388                         *COULD BE LOCATED HERE ALSO.(IN RXEND SUBROUTINE)


ØØ39ØA A1DC 96   54      A        LDAA   DMAPCR    TURN ON DMA CHAN 1 (RECV)
ØØ391A A1DE 8A   Ø2      A        ORAA   #$Ø2
ØØ392A A1EØ 97   54      A        STAA   DMAPCR
ØØ393A A1E2 96   25      A        LDAA   CR1IMG    TURN ON ADLC TO DMA MODE
ØØ394A A1E4 8A   Ø8      A        ORAA   #$Ø8      OF OPERATION
```

Figure 10. Priority Mode Program Listing (Sheet 9 of 11)



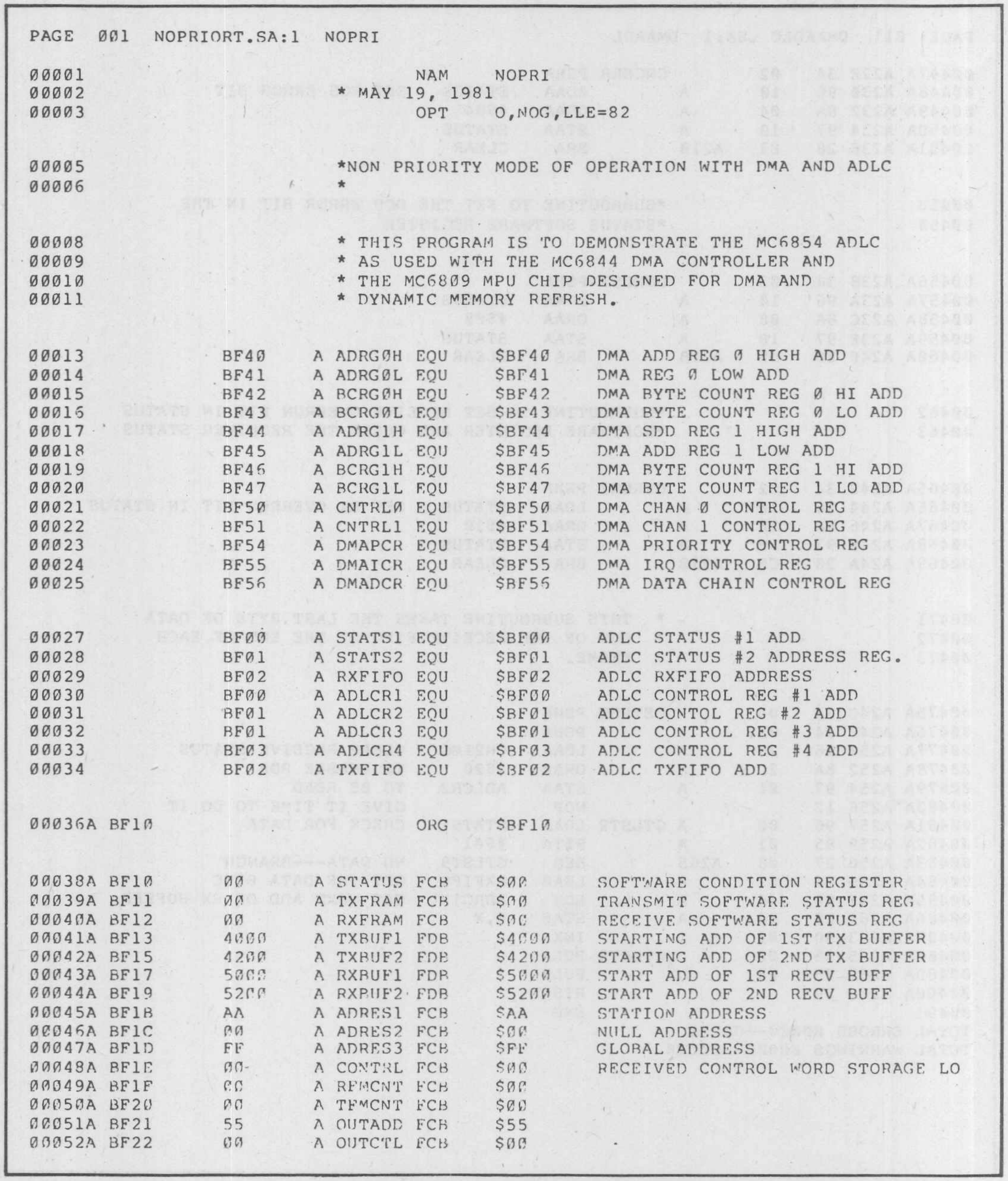
```
PAGE  Ø1Ø  DMAADLC .SA:1  DMAADL

ØØ395A A1E6 97   25      A         STAA  CR1IMG
ØØ396A A1E8 97   ØØ      A         STAA  ADLCR1   DO TI
ØØ397A A1EA 35   Ø2                PULA
ØØ398A A1EC 39                     RTS

ØØ4ØØ                      *SUBROUTINE TO LOAD THE ALTERNATE TX BUFFER
ØØ4Ø1                      *ADDRESS INTO THE ADDRESS REG OF THE DMA CHAN Ø

ØØ4Ø3A A1ED 34   Ø2        TDMAON PSHA
ØØ4Ø4A A1EF 96   11      A         LDAA  TXFRAM   GET TX FRAME STATUS
ØØ4Ø5A A1F1 85   Ø1      A         BITA  #$Ø1     TEST WHICH BUFFER TO USE
ØØ4Ø6A A1F3 26   Ø8   A1FD         BNE   TDMON1   BRANCH NOT SET
ØØ4Ø7A A1F5 8A   Ø1      A         ORAA  #$Ø1
ØØ4Ø8A A1F7 9E   15      A         LDX   TXBUF2   SELECT TX BUFFER #2
ØØ4Ø9A A1F9 9F   4Ø      A         STX   ADRGØH   SET UP ADD REG IN DMA
ØØ41ØA A1FB 2Ø   Ø6   A2Ø3         BRA   TDMON2
ØØ411A A1FD 8Ø   Ø1      A TDMON1  SUBA  #$Ø1
ØØ412A A1FF 9E   13      A         LDX   TXBUF1   SELECT TX BUFFER #1
ØØ413A A2Ø1 9F   4Ø      A         STX   ADRGØH   SET UP ADD REG IN DMA
ØØ414A A2Ø3 97   11      A TDMON2  STAA  TXFRAM
ØØ415A A2Ø5 8E   Ø4ØØ    A         LDX   #$Ø4ØØ   SET UP BCR IN DMA
ØØ416A A2Ø8 9F   42      A         STX   BCRGØH   DO IT
ØØ417A A2ØA 96   54      A         LDAA  DMAPCR   ENABLE CHAN Ø IN DMA
ØØ418A A2ØC 8A   Ø1      A         ORAA  #$Ø1
ØØ419A A2ØE 97   54      A         STAA  DMAPCR
ØØ42ØA A21Ø 35   Ø2                PULA
ØØ421A A212 39                     RTS

ØØ423                      *SUBROUTINE TO SET THE INACTIVE IDLE BIT IN THE
ØØ424                      *STATUS SOFTWARE REGISTER

ØØ426A A213 34   Ø2        IDLE    PSHA
ØØ427A A215 96   1Ø      A         LDAA  STATUS
ØØ428A A217 8A   Ø1      A         ORAA  #$Ø1     SET INACTIVE IDLE BIT
ØØ429A A219 97   1Ø      A         STAA  STATUS
ØØ43ØA A21B 96   26      A CLEAR   LDAA  CR2IMG   CLEAR RECEIVER STATUS
ØØ431A A21D 8A   2Ø      A         ORAA  #$2Ø
ØØ432A A21F 97   Ø1      A         STAA  ADLCR2
ØØ433A A221 35   Ø2                PULA
ØØ434A A223 39                     RTS

ØØ436                      *SUBROUTINE TO SET THE ABORT BIT IN THE STATUS
ØØ437                      *SOFTWARE REGISTER

ØØ439A A224 34   Ø2        RABORT PSHA
ØØ44ØA A226 96   1Ø      A         LDAA  STATUS   SET ABORT BIT
ØØ441A A228 8A   Ø2      A         ORAA  #$Ø2
ØØ442A A22A 97   1Ø      A         STAA  STATUS
ØØ443A A22C 2Ø   ED   A21B         BRA   CLEAR
ØØ444                      *SUBROUTINE TO SET THE FCS ERROR BIT IN THE
ØØ445                      *STATUS SOFTWARE REGISTER
```

Figure 10. Priority Mode Program Listing (Sheet 10 of 11)



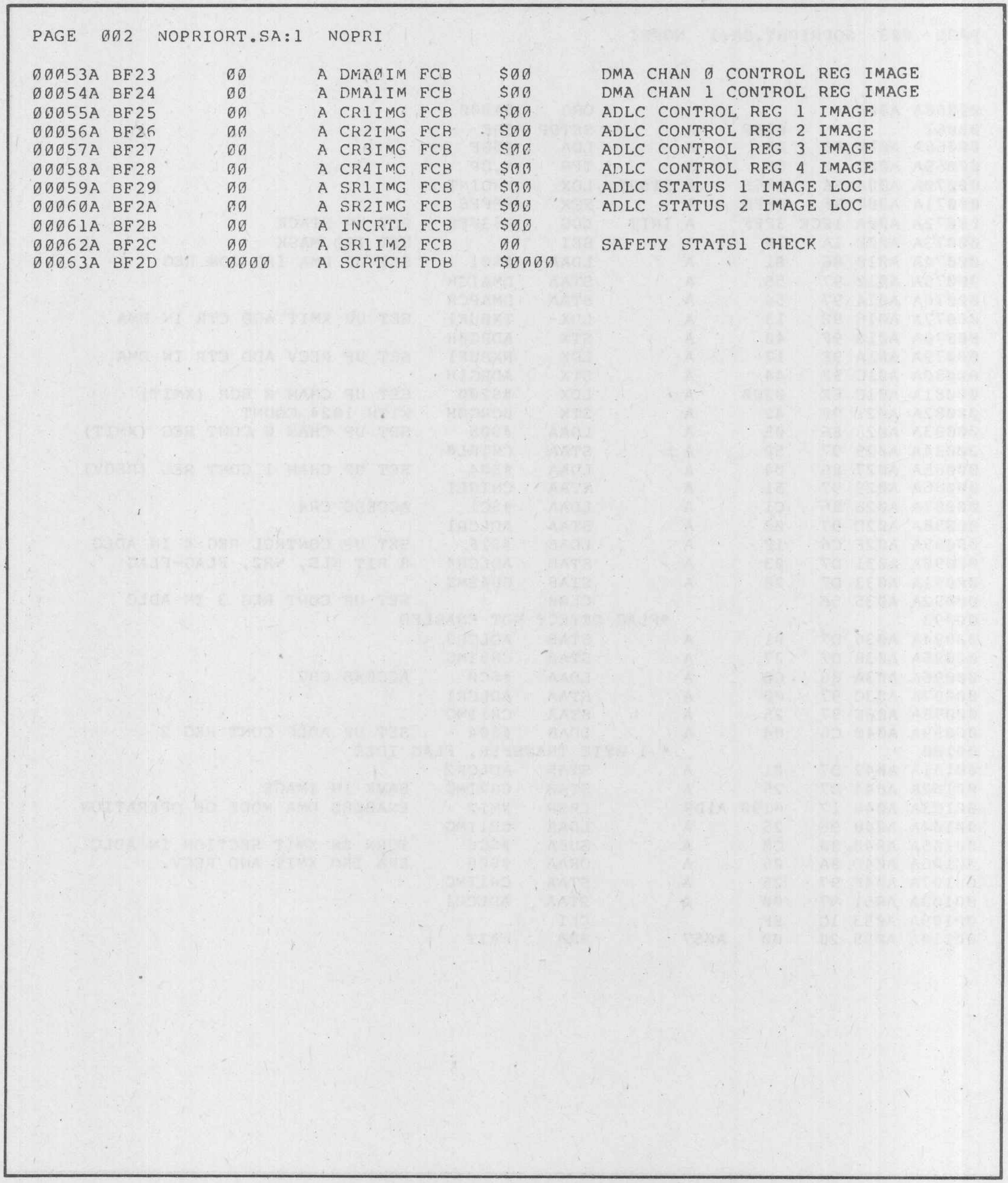
```
PAGE  Ø11   DMAADLC .SA:1   DMAADL

ØØ447A A22E 34      Ø2          CRCERR  PSHA
ØØ448A A23Ø 96      1Ø         A         LDAA     STATUS     SET FCS ERROR BIT
ØØ449A A232 8A      Ø4         A         ORAA     #$Ø4
ØØ45ØA A234 97      1Ø         A         STAA     STATUS
ØØ451A A236 2Ø      E3    A21B           BRA      CLEAR

ØØ453                             *SUBROUTINE TO SET THE DCD ERROR BIT IN THE
ØØ454                             *STATUS SOFTWARE REGISTER

ØØ456A A238 34      Ø2          DCDLST  PSHA
ØØ457A A23A 96      1Ø         A         LDAA     STATUS
ØØ458A A23C 8A      Ø8         A         ORAA     #$Ø8
ØØ459A A23E 97      1Ø         A         STAA     STATUS
ØØ46ØA A24Ø 2Ø      D9    A21B           BRA      CLEAR

ØØ462                             *SUBROUTINE TO SET RECEIVE OVERRUN BIT IN STATUS
ØØ463                             *SOFTWARE REGISTER AND CLEAR THE RECEIVER STATUS

ØØ465A A242 34      Ø2          OVRUN1  PSHA
ØØ466A A244 96      1Ø         A         LDAA     STATUS     SET RX OVERRUN BIT IN STATUS
ØØ467A A246 8A      1Ø         A         ORAA     #$1Ø
ØØ468A A248 97      1Ø         A         STAA     STATUS
ØØ469A A24A 2Ø      CF    A21B           BRA      CLEAR

ØØ471                             *  THIS SUBROUTINE TAKES THE LAST BYTE OF DATA
ØØ472                             *  OUT OF THE RECEIVE FIFO AT THE END OF EACH
ØØ473                             *  FRAME.

ØØ475A A24C 34      Ø4          GETLST  PSHB
ØØ476A A24E 34      Ø2                  PSHA
ØØ477A A25Ø 96      26         A         LDAA     CR2IMG     CLEAR RECEIVE STATUS
ØØ478A A252 8A      2Ø         A         ORAA     #$2Ø       TO ENABLE RDA
ØØ479A A254 97      Ø1         A         STAA     ADLCR2     TO BE READ
ØØ48ØA A256 12                          NOP                 GIVE IT TIME TO DO IT
ØØ481A A257 96      ØØ         A GTLST2  LDAA     STATS1     CHECK FOR DATA
ØØ482A A259 85      Ø1         A         BITA     #$Ø1
ØØ483A A25B 27      Ø8    A265           BEQ      GTLST9     NO DATA---BRANCH
ØØ484A A25D D6      Ø2         A         LDAB     RXFIFO     GET THE DATA BYTE
ØØ485A A25F 9E      44         A         LDX      ADRG1H     GET NEXT ADD OF RX BUFFER
ØØ486A A261 E7      84         A         STAB     Ø,X
ØØ487A A263 3Ø      Ø1                  INX
ØØ488A A265 35      Ø2          GTLST9  PULA
ØØ489A A267 35      Ø4                  PULB
ØØ49ØA A269 39                          RTS
ØØ491                                    END
TOTAL ERRORS ØØØØØ--ØØØØØ
TOTAL WARNINGS ØØØØØ--ØØØØØ
```

Figure 10. Priority Mode Program Listing (Sheet 11 of 11)



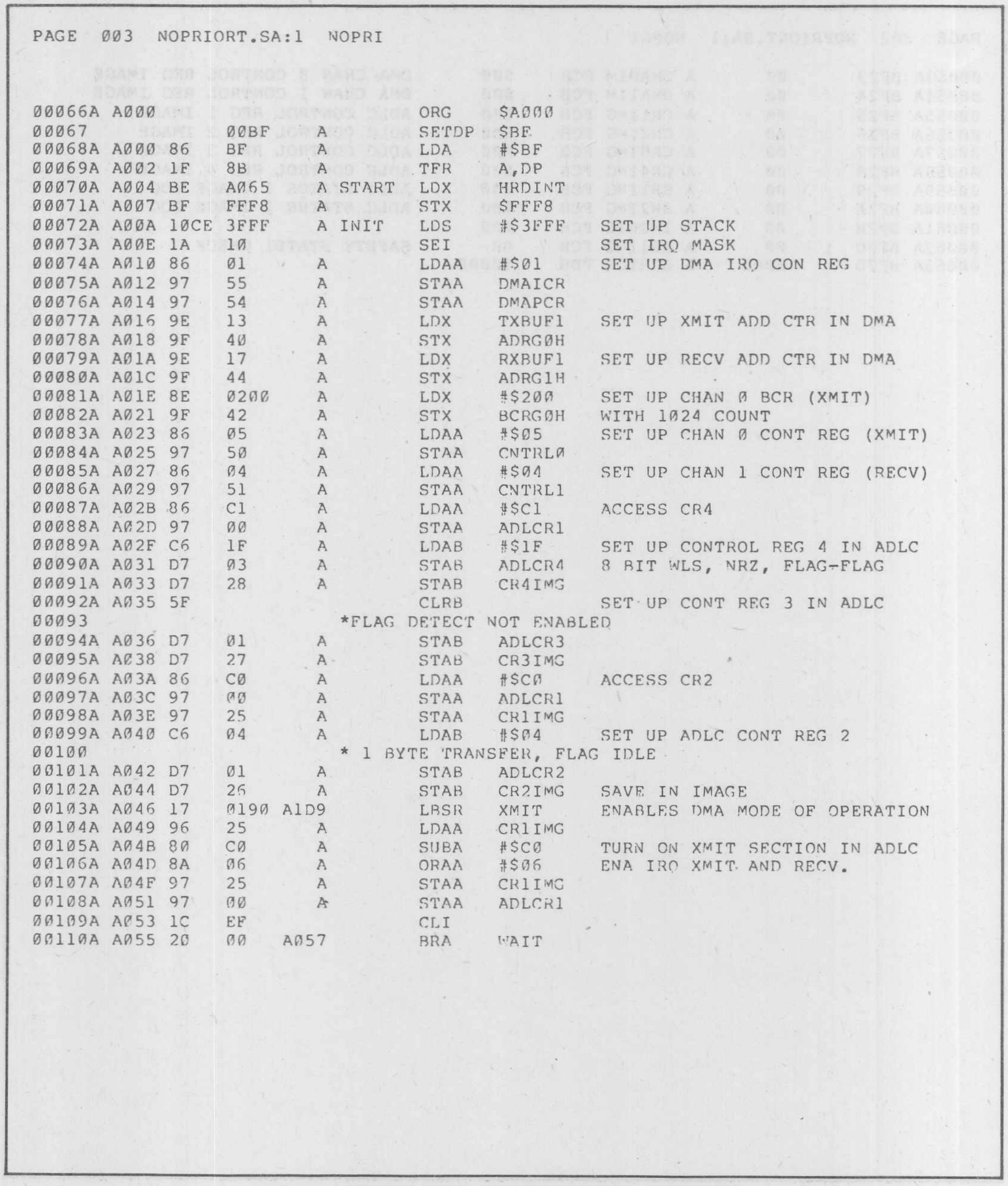
```
PAGE  ØØ1  NOPRIORT.SA:1  NOPRI

ØØØØ1                                   NAM     NOPRI
ØØØØ2                           * MAY 19, 1981
ØØØØ3                                   OPT     O,NOG,LLE=82

ØØØØ5                           *NON PRIORITY MODE OF OPERATION WITH DMA AND ADLC
ØØØØ6                           *

ØØØØ8                           * THIS PROGRAM IS TO DEMONSTRATE THE MC6854 ADLC
ØØØØ9                           * AS USED WITH THE MC6844 DMA CONTROLLER AND
ØØØ1Ø                           * THE MC68Ø9 MPU CHIP DESIGNED FOR DMA AND
ØØØ11                           * DYNAMIC MEMORY REFRESH.

ØØØ13           BF4Ø    A ADRGØH EQU    $BF4Ø    DMA ADD REG Ø HIGH ADD
ØØØ14           BF41    A ADRGØL EQU    $BF41    DMA REG Ø LOW ADD
ØØØ15           BF42    A BCRGØH EQU    $BF42    DMA BYTE COUNT REG Ø HI ADD
ØØØ16           BF43    A BCRGØL EQU    $BF43    DMA BYTE COUNT REG Ø LO ADD
ØØØ17           BF44    A ADRG1H EQU    $BF44    DMA SDD REG 1 HIGH ADD
ØØØ18           BF45    A ADRG1L EQU    $BF45    DMA ADD REG 1 LOW ADD
ØØØ19           BF46    A BCRG1H EQU    $BF46    DMA BYTE COUNT REG 1 HI ADD
ØØØ2Ø           BF47    A BCRG1L EQU    $BF47    DMA BYTE COUNT REG 1 LO ADD
ØØØ21           BF5Ø    A CNTRLØ EQU    $BF5Ø    DMA CHAN Ø CONTROL REG
ØØØ22           BF51    A CNTRL1 EQU    $BF51    DMA CHAN 1 CONTROL REG
ØØØ23           BF54    A DMAPCR EQU    $BF54    DMA PRIORITY CONTROL REG
ØØØ24           BF55    A DMAICR EQU    $BF55    DMA IRQ CONTROL REG
ØØØ25           BF56    A DMADCR EQU    $BF56    DMA DATA CHAIN CONTROL REG

ØØØ27           BFØØ    A STATS1 EQU    $BFØØ    ADLC STATUS #1 ADD
ØØØ28           BFØ1    A STATS2 EQU    $BFØ1    ADLC STATUS #2 ADDRESS REG.
ØØØ29           BFØ2    A RXFIFO EQU    $BFØ2    ADLC RXFIFO ADDRESS
ØØØ3Ø           BFØØ    A ADLCR1 EQU    $BFØØ    ADLC CONTROL REG #1 ADD
ØØØ31           BFØ1    A ADLCR2 EQU    $BFØ1    ADLC CONTOL REG #2 ADD
ØØØ32           BFØ1    A ADLCR3 EQU    $BFØ1    ADLC CONTROL REG #3 ADD
ØØØ33           BFØ3    A ADLCR4 EQU    $BFØ3    ADLC CONTROL REG #4 ADD
ØØØ34           BFØ2    A TXFIFO EQU    $BFØ2    ADLC TXFIFO ADD

ØØØ36A BF1Ø                      ORG    $BF1Ø

ØØØ38A BF1Ø     ØØ      A STATUS FCB    $ØØ      SOFTWARE CONDITION REGISTER
ØØØ39A BF11     ØØ      A TXFRAM FCB    $ØØ      TRANSMIT SOFTWARE STATUS REG
ØØØ4ØA BF12     ØØ      A RXFRAM FCB    $ØØ      RECEIVE SOFTWARE STATUS REG
ØØØ41A BF13     4ØØØ    A TXBUF1 FDB    $4ØØØ    STARTING ADD OF 1ST TX BUFFER
ØØØ42A BF15     42ØØ    A TXBUF2 FDB    $42ØØ    STARTING ADD OF 2ND TX BUFFER
ØØØ43A BF17     5ØØØ    A RXBUF1 FDB    $5ØØØ    START ADD OF 1ST RECV BUFF
ØØØ44A BF19     52ØØ    A RXBUF2 FDB    $52ØØ    START ADD OF 2ND RECV BUFF
ØØØ45A BF1B     AA      A ADRES1 FCB    $AA      STATION ADDRESS
ØØØ46A BF1C     ØØ      A ADRES2 FCB    $ØØ      NULL ADDRESS
ØØØ47A BF1D     FF      A ADRES3 FCB    $FF      GLOBAL ADDRESS
ØØØ48A BF1E     ØØ      A CONTRL FCB    $ØØ      RECEIVED CONTROL WORD STORAGE LO
ØØØ49A BF1F     ØØ      A REMCNT FCB    $ØØ
ØØØ5ØA BF2Ø     ØØ      A TEMCNT FCB    $ØØ
ØØØ51A BF21     55      A OUTADD FCB    $55
ØØØ52A BF22     ØØ      A OUTCTL FCB    $ØØ
```

Figure 11. Non-Priority Mode Program Listing (Sheet 1 of 12)



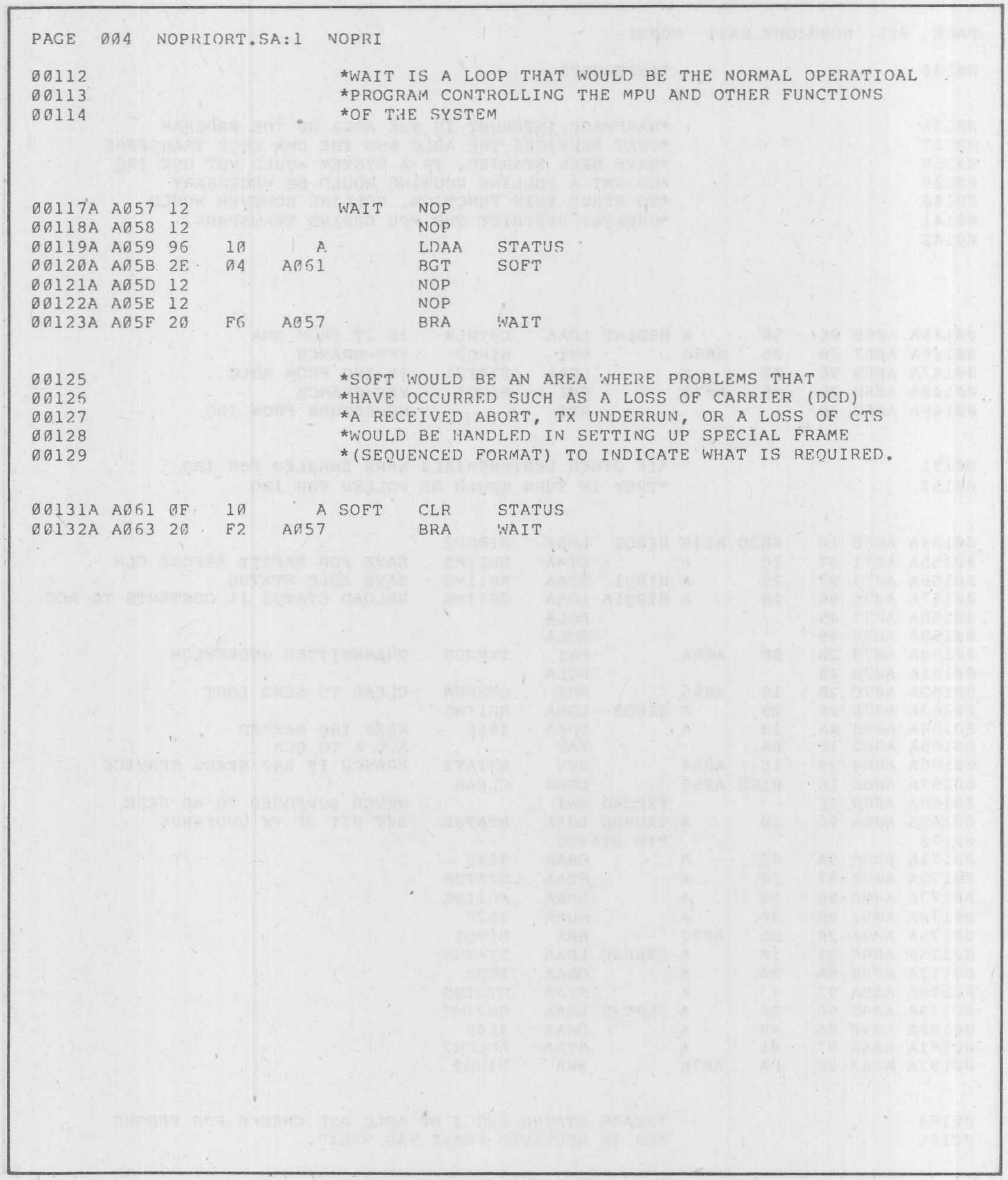
```
PAGE  ØØ2  NOPRIORT.SA:1  NOPRI

ØØØ53A BF23       ØØ        A DMAØIM FCB    $ØØ       DMA CHAN Ø CONTROL REG IMAGE
ØØØ54A BF24       ØØ        A DMA1IM FCB    $ØØ       DMA CHAN 1 CONTROL REG IMAGE
ØØØ55A BF25       ØØ        A CR1IMG FCB    $ØØ       ADLC CONTROL REG 1 IMAGE
ØØØ56A BF26       ØØ        A CR2IMG FCB    $ØØ       ADLC CONTROL REG 2 IMAGE
ØØØ57A BF27       ØØ        A CR3IMG FCB    $ØØ       ADLC CONTROL REG 3 IMAGE
ØØØ58A BF28       ØØ        A CR4IMG FCB    $ØØ       ADLC CONTROL REG 4 IMAGE
ØØØ59A BF29       ØØ        A SR1IMG FCB    $ØØ       ADLC STATUS 1 IMAGE LOC
ØØØ6ØA BF2A       ØØ        A SR2IMG FCB    $ØØ       ADLC STATUS 2 IMAGE LOC
ØØØ61A BF2B       ØØ        A INCRTL FCB    $ØØ
ØØØ62A BF2C       ØØ        A SR1IM2 FCB    ØØ        SAFETY STATS1 CHECK
ØØØ63A BF2D     ØØØØ        A SCRTCH FDB    $ØØØØ
```

Figure 11. Non-Priority Mode Program Listing (Sheet 2 of 12)



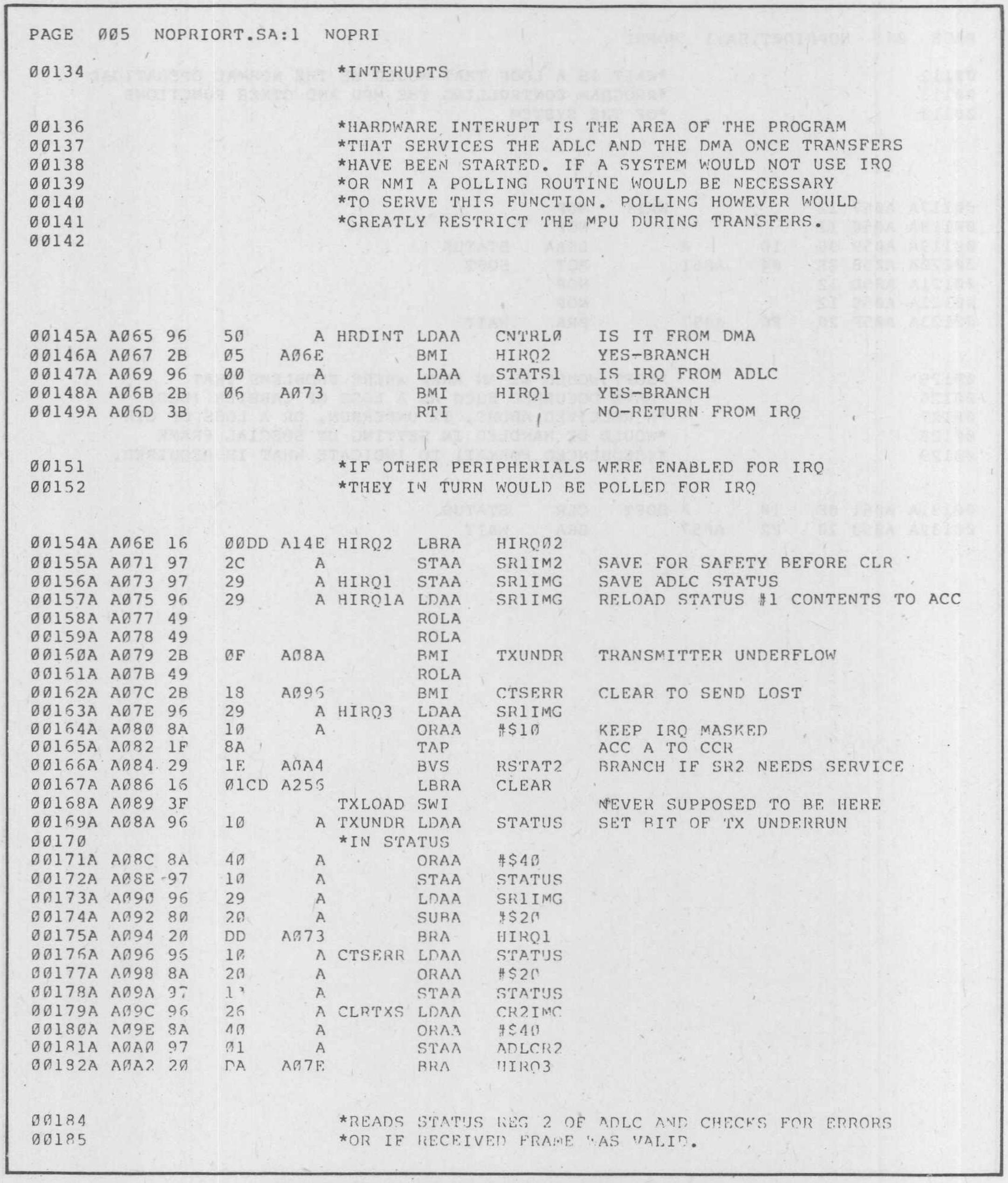
```
PAGE  ØØ3  NOPRIORT.SA:1  NOPRI

ØØØ66A AØØØ                     ORG    $AØØØ
ØØØ67       ØØBF     A          SETDP  $BF
ØØØ68A AØØØ 86   BF       A          LDA    #$BF
ØØØ69A AØØ2 1F   8B       A          TFR    A,DP
ØØØ7ØA AØØ4 BE   AØ65     A START    LDX    HRDINT
ØØØ71A AØØ7 BF   FFF8     A          STX    $FFF8
ØØØ72A AØØA 1ØCE 3FFF     A INIT     LDS    #$3FFF   SET UP STACK
ØØØ73A AØØE 1A   1Ø                  SEI             SET IRQ MASK
ØØØ74A AØ1Ø 86   Ø1       A          LDAA   #$Ø1     SET UP DMA IRQ CON REG
ØØØ75A AØ12 97   55       A          STAA   DMAICR
ØØØ76A AØ14 97   54       A          STAA   DMAPCR
ØØØ77A AØ16 9E   13       A          LDX    TXBUF1   SET UP XMIT ADD CTR IN DMA
ØØØ78A AØ18 9F   4Ø       A          STX    ADRGØH
ØØØ79A AØ1A 9E   17       A          LDX    RXBUF1   SET UP RECV ADD CTR IN DMA
ØØØ8ØA AØ1C 9F   44       A          STX    ADRG1H
ØØØ81A AØ1E 8E   Ø2ØØ     A          LDX    #$2ØØ    SET UP CHAN Ø BCR (XMIT)
ØØØ82A AØ21 9F   42       A          STX    BCRGØH   WITH 1Ø24 COUNT
ØØØ83A AØ23 86   Ø5       A          LDAA   #$Ø5     SET UP CHAN Ø CONT REG (XMIT)
ØØØ84A AØ25 97   5Ø       A          STAA   CNTRLØ
ØØØ85A AØ27 86   Ø4       A          LDAA   #$Ø4     SET UP CHAN 1 CONT REG (RECV)
ØØØ86A AØ29 97   51       A          STAA   CNTRL1
ØØØ87A AØ2B 86   C1       A          LDAA   #$C1     ACCESS CR4
ØØØ88A AØ2D 97   ØØ       A          STAA   ADLCR1
ØØØ89A AØ2F C6   1F       A          LDAB   #$1F     SET UP CONTROL REG 4 IN ADLC
ØØØ9ØA AØ31 D7   Ø3       A          STAB   ADLCR4   8 BIT WLS, NRZ, FLAG-FLAG
ØØØ91A AØ33 D7   28       A          STAB   CR4IMG
ØØØ92A AØ35 5F                       CLRB            SET UP CONT REG 3 IN ADLC
ØØØ93                       *FLAG DETECT NOT ENABLED
ØØØ94A AØ36 D7   Ø1       A          STAB   ADLCR3
ØØØ95A AØ38 D7   27       A          STAB   CR3IMG
ØØØ96A AØ3A 86   CØ       A          LDAA   #$CØ     ACCESS CR2
ØØØ97A AØ3C 97   ØØ       A          STAA   ADLCR1
ØØØ98A AØ3E 97   25       A          STAA   CR1IMG
ØØØ99A AØ4Ø C6   Ø4       A          LDAB   #$Ø4     SET UP ADLC CONT REG 2
ØØ1ØØ                       * 1 BYTE TRANSFER, FLAG IDLE
ØØ1Ø1A AØ42 D7   Ø1       A          STAB   ADLCR2
ØØ1Ø2A AØ44 D7   26       A          STAB   CR2IMG   SAVE IN IMAGE
ØØ1Ø3A AØ46 17   Ø19Ø A1D9           LBSR   XMIT     ENABLES DMA MODE OF OPERATION
ØØ1Ø4A AØ49 96   25       A          LDAA   CR1IMG
ØØ1Ø5A AØ4B 8Ø   CØ       A          SUBA   #$CØ     TURN ON XMIT SECTION IN ADLC
ØØ1Ø6A AØ4D 8A   Ø6       A          ORAA   #$Ø6     ENA IRQ XMIT AND RECV.
ØØ1Ø7A AØ4F 97   25       A          STAA   CR1IMG
ØØ1Ø8A AØ51 97   ØØ       A          STAA   ADLCR1
ØØ1Ø9A AØ53 1C   EF                  CLI
ØØ11ØA AØ55 2Ø   ØØ   AØ57           BRA    WAIT
```

Figure 11. Non-Priority Mode Program Listing (Sheet 3 of 12)



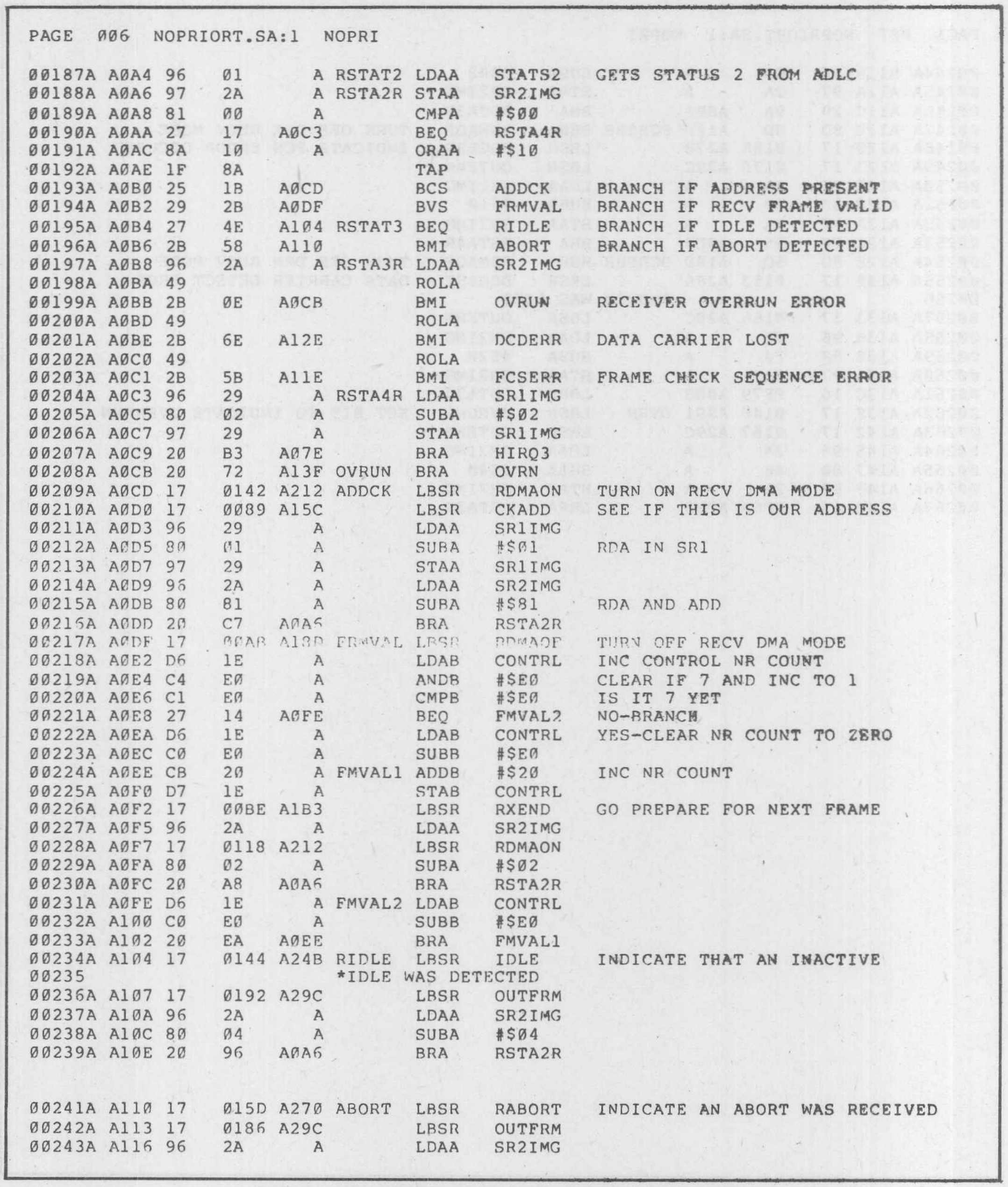
```
PAGE  ØØ4  NOPRIORT.SA:1   NOPRI

ØØ112                       *WAIT IS A LOOP THAT WOULD BE THE NORMAL OPERATIOAL
ØØ113                       *PROGRAM CONTROLLING THE MPU AND OTHER FUNCTIONS
ØØ114                       *OF THE SYSTEM

ØØ117A AØ57 12              WAIT     NOP
ØØ118A AØ58 12                       NOP
ØØ119A AØ59 96    1Ø      A          LDAA   STATUS
ØØ12ØA AØ5B 2E    Ø4   AØ61          BGT    SOFT
ØØ121A AØ5D 12                       NOP
ØØ122A AØ5E 12                       NOP
ØØ123A AØ5F 2Ø    F6   AØ57          BRA    WAIT

ØØ125                       *SOFT WOULD BE AN AREA WHERE PROBLEMS THAT
ØØ126                       *HAVE OCCURRED SUCH AS A LOSS OF CARRIER (DCD)
ØØ127                       *A RECEIVED ABORT, TX UNDERRUN, OR A LOSS OF CTS
ØØ128                       *WOULD BE HANDLED IN SETTING UP SPECIAL FRAME
ØØ129                       *(SEQUENCED FORMAT) TO INDICATE WHAT IS REQUIRED.

ØØ131A AØ61 ØF    1Ø      A SOFT     CLR    STATUS
ØØ132A AØ63 2Ø    F2   AØ57          BRA    WAIT
```

Figure 11. Non-Priority Mode Program Listing (Sheet 4 of 12)



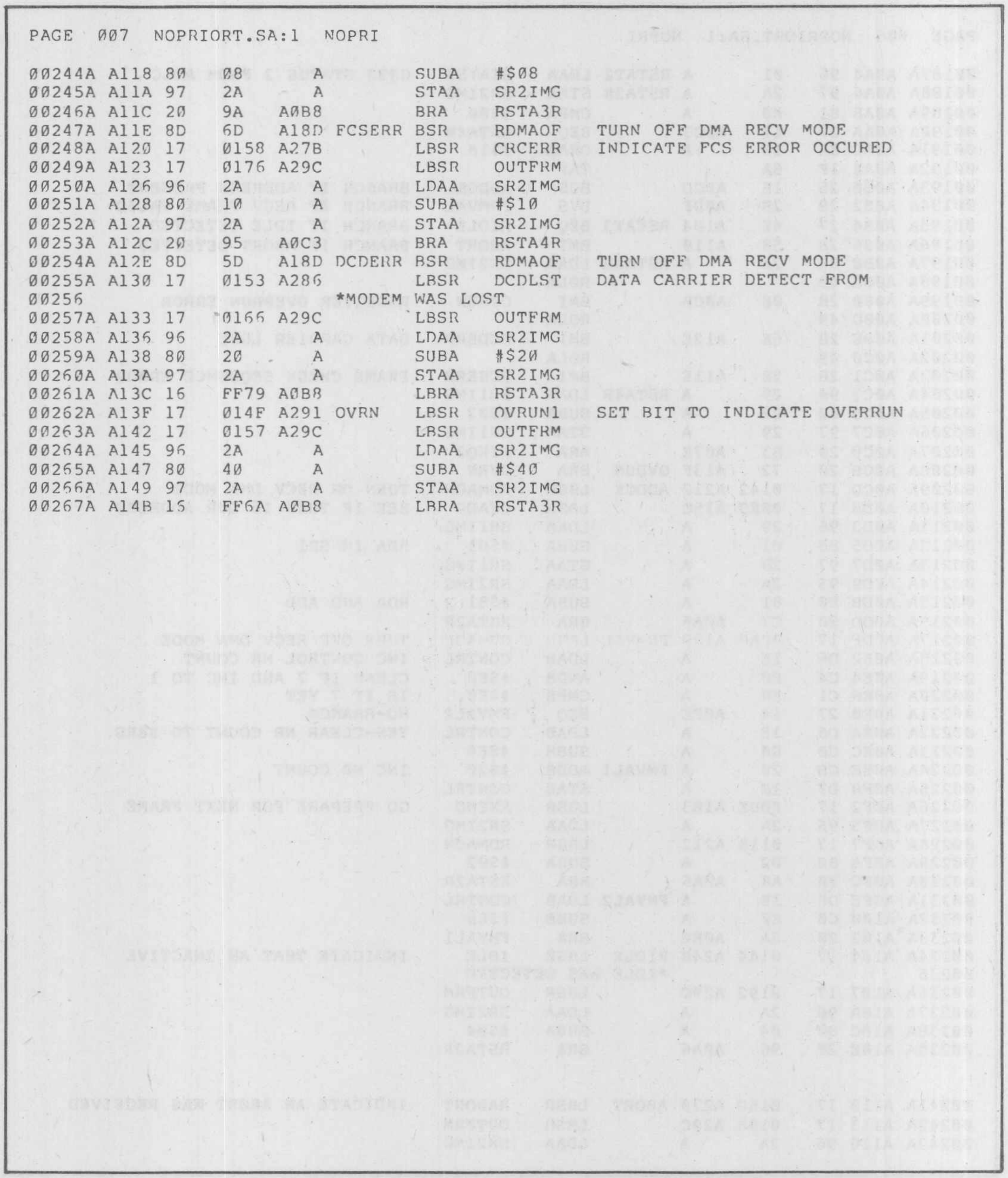
```
PAGE  ØØ5  NOPRIORT.SA:1   NOPRI

ØØ134                          *INTERUPTS

ØØ136                          *HARDWARE INTERUPT IS THE AREA OF THE PROGRAM
ØØ137                          *THAT SERVICES THE ADLC AND THE DMA ONCE TRANSFERS
ØØ138                          *HAVE BEEN STARTED. IF A SYSTEM WOULD NOT USE IRQ
ØØ139                          *OR NMI A POLLING ROUTINE WOULD BE NECESSARY
ØØ14Ø                          *TO SERVE THIS FUNCTION. POLLING HOWEVER WOULD
ØØ141                          *GREATLY RESTRICT THE MPU DURING TRANSFERS.
ØØ142




ØØ145A AØ65 96   5Ø       A HRDINT LDAA   CNTRLØ    IS IT FROM DMA
ØØ146A AØ67 2B   Ø5   AØ6E         BMI    HIRQ2     YES-BRANCH
ØØ147A AØ69 96   ØØ       A        LDAA   STATS1    IS IRQ FROM ADLC
ØØ148A AØ6B 2B   Ø6   AØ73         BMI    HIRQ1     YES-BRANCH
ØØ149A AØ6D 3B                     RTI              NO-RETURN FROM IRQ


ØØ151                          *IF OTHER PERIPHERIALS WERE ENABLED FOR IRQ
ØØ152                          *THEY IN TURN WOULD BE POLLED FOR IRQ


ØØ154A AØ6E 16   ØØDD A14E HIRQ2   LBRA   HIRQØ2
ØØ155A AØ71 97   2C       A        STAA   SR1IM2    SAVE FOR SAFETY BEFORE CLR
ØØ156A AØ73 97   29       A HIRQ1  STAA   SR1IMG    SAVE ADLC STATUS
ØØ157A AØ75 96   29       A HIRQ1A LDAA   SR1IMG    RELOAD STATUS #1 CONTENTS TO ACC
ØØ158A AØ77 49                     ROLA
ØØ159A AØ78 49                     ROLA
ØØ16ØA AØ79 2B   ØF   AØ8A         BMI    TXUNDR    TRANSMITTER UNDERFLOW
ØØ161A AØ7B 49                     ROLA
ØØ162A AØ7C 2B   18   AØ96         BMI    CTSERR    CLEAR TO SEND LOST
ØØ163A AØ7E 96   29       A HIRQ3  LDAA   SR1IMG
ØØ164A AØ8Ø 8A   1Ø       A        ORAA   #$1Ø      KEEP IRQ MASKED
ØØ165A AØ82 1F   8A                TAP              ACC A TO CCR
ØØ166A AØ84 29   1E   AØA4         BVS    RSTAT2    BRANCH IF SR2 NEEDS SERVICE
ØØ167A AØ86 16   Ø1CD A256         LBRA   CLEAR
ØØ168A AØ89 3F              TXLOAD SWI              NEVER SUPPOSED TO BE HERE
ØØ169A AØ8A 96   1Ø       A TXUNDR LDAA   STATUS    SET BIT OF TX UNDERRUN
ØØ17Ø                          *IN STATUS
ØØ171A AØ8C 8A   4Ø       A        ORAA   #$4Ø
ØØ172A AØ8E 97   1Ø       A        STAA   STATUS
ØØ173A AØ9Ø 96   29       A        LDAA   SR1IMG
ØØ174A AØ92 8Ø   2Ø       A        SUBA   #$2Ø
ØØ175A AØ94 2Ø   DD   AØ73         BRA    HIRQ1
ØØ176A AØ96 96   1Ø       A CTSERR LDAA   STATUS
ØØ177A AØ98 8A   2Ø       A        ORAA   #$2Ø
ØØ178A AØ9A 97   1Ø       A        STAA   STATUS
ØØ179A AØ9C 96   26       A CLRTXS LDAA   CR2IMG
ØØ18ØA AØ9E 8A   4Ø       A        ORAA   #$4Ø
ØØ181A AØAØ 97   Ø1       A        STAA   ADLCR2
ØØ182A AØA2 2Ø   DA   AØ7E         BRA    HIRQ3


ØØ184                          *READS STATUS REG 2 OF ADLC AND CHECKS FOR ERRORS
ØØ185                          *OR IF RECEIVED FRAME WAS VALID.
```

Figure 11. Non-Priority Mode Program Listing (Sheet 5 of 12)



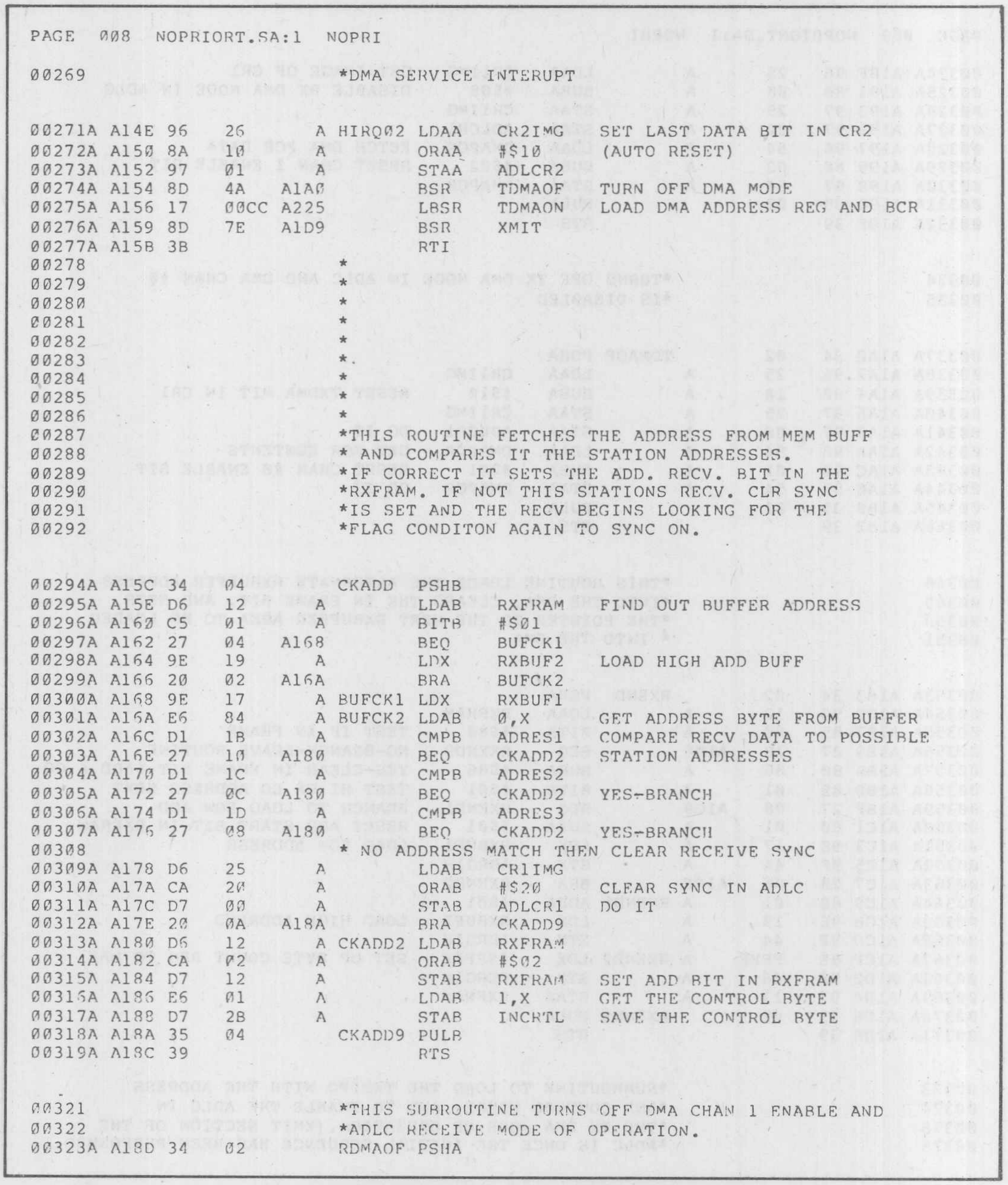
```
PAGE  ØØ6  NOPRIORT.SA:1  NOPRI

ØØ187A AØA4 96   Ø1       A RSTAT2 LDAA   STATS2   GETS STATUS 2 FROM ADLC
ØØ188A AØA6 97   2A       A RSTA2R STAA   SR2IMG
ØØ189A AØA8 81   ØØ       A        CMPA   #$ØØ
ØØ19ØA AØAA 27   17   AØC3         BEQ    RSTA4R
ØØ191A AØAC 8A   1Ø       A        ORAA   #$1Ø
ØØ192A AØAE 1F   8A                TAP
ØØ193A AØBØ 25   1B   AØCD         BCS    ADDCK    BRANCH IF ADDRESS PRESENT
ØØ194A AØB2 29   2B   AØDF         BVS    FRMVAL   BRANCH IF RECV FRAME VALID
ØØ195A AØB4 27   4E   A1Ø4 RSTAT3  BEQ    RIDLE    BRANCH IF IDLE DETECTED
ØØ196A AØB6 2B   58   A11Ø         BMI    ABORT    BRANCH IF ABORT DETECTED
ØØ197A AØB8 96   2A       A RSTA3R LDAA   SR2IMG
ØØ198A AØBA 49                     ROLA
ØØ199A AØBB 2B   ØE   AØCB         BMI    OVRUN    RECEIVER OVERRUN ERROR
ØØ2ØØA AØBD 49                     ROLA
ØØ2Ø1A AØBE 2B   6E   A12E         BMI    DCDERR   DATA CARRIER LOST
ØØ2Ø2A AØCØ 49                     ROLA
ØØ2Ø3A AØC1 2B   5B   A11E         BMI    FCSERR   FRAME CHECK SEQUENCE ERROR
ØØ2Ø4A AØC3 96   29       A RSTA4R LDAA   SR1IMG
ØØ2Ø5A AØC5 8Ø   Ø2       A        SUBA   #$Ø2
ØØ2Ø6A AØC7 97   29       A        STAA   SR1IMG
ØØ2Ø7A AØC9 2Ø   B3   AØ7E         BRA    HIRQ3
ØØ2Ø8A AØCB 2Ø   72   A13F OVRUN   BRA    OVRN
ØØ2Ø9A AØCD 17   Ø142 A212 ADDCK   LBSR   RDMAON   TURN ON RECV DMA MODE
ØØ21ØA AØDØ 17   ØØ89 A15C         LBSR   CKADD    SEE IF THIS IS OUR ADDRESS
ØØ211A AØD3 96   29       A        LDAA   SR1IMG
ØØ212A AØD5 8Ø   Ø1       A        SUBA   #$Ø1     RDA IN SR1
ØØ213A AØD7 97   29       A        STAA   SR1IMG
ØØ214A AØD9 96   2A       A        LDAA   SR2IMG
ØØ215A AØDB 8Ø   81       A        SUBA   #$81     RDA AND ADD
ØØ216A AØDD 2Ø   C7   AØA6         BRA    RSTA2R
ØØ217A AØDF 17   ØØAB A18D FRMVAL  LBSR   RDMAOF   TURN OFF RECV DMA MODE
ØØ218A AØE2 D6   1E       A        LDAB   CONTRL   INC CONTROL NR COUNT
ØØ219A AØE4 C4   EØ       A        ANDB   #$EØ     CLEAR IF 7 AND INC TO 1
ØØ22ØA AØE6 C1   EØ       A        CMPB   #$EØ     IS IT 7 YET
ØØ221A AØE8 27   14   AØFE         BEQ    FMVAL2   NO-BRANCH
ØØ222A AØEA D6   1E       A        LDAB   CONTRL   YES-CLEAR NR COUNT TO ZERO
ØØ223A AØEC CØ   EØ       A        SUBB   #$EØ
ØØ224A AØEE CB   2Ø       A FMVAL1 ADDB   #$2Ø     INC NR COUNT
ØØ225A AØFØ D7   1E       A        STAB   CONTRL
ØØ226A AØF2 17   ØØBE A1B3         LBSR   RXEND    GO PREPARE FOR NEXT FRAME
ØØ227A AØF5 96   2A       A        LDAA   SR2IMG
ØØ228A AØF7 17   Ø118 A212         LBSR   RDMAON
ØØ229A AØFA 8Ø   Ø2       A        SUBA   #$Ø2
ØØ23ØA AØFC 2Ø   A8   AØA6         BRA    RSTA2R
ØØ231A AØFE D6   1E       A FMVAL2 LDAB   CONTRL
ØØ232A A1ØØ CØ   EØ       A        SUBB   #$EØ
ØØ233A A1Ø2 2Ø   EA   AØEE         BRA    FMVAL1
ØØ234A A1Ø4 17   Ø144 A24B RIDLE   LBSR   IDLE     INDICATE THAT AN INACTIVE
ØØ235                      *IDLE WAS DETECTED
ØØ236A A1Ø7 17   Ø192 A29C         LBSR   OUTFRM
ØØ237A A1ØA 96   2A       A        LDAA   SR2IMG
ØØ238A A1ØC 8Ø   Ø4       A        SUBA   #$Ø4
ØØ239A A1ØE 2Ø   96   AØA6         BRA    RSTA2R


ØØ241A A11Ø 17   Ø15D A27Ø ABORT   LBSR   RABORT   INDICATE AN ABORT WAS RECEIVED
ØØ242A A113 17   Ø186 A29C         LBSR   OUTFRM
ØØ243A A116 96   2A       A        LDAA   SR2IMG
```

Figure 11. Non-Priority Mode Program Listing (Sheet 6 of 12)



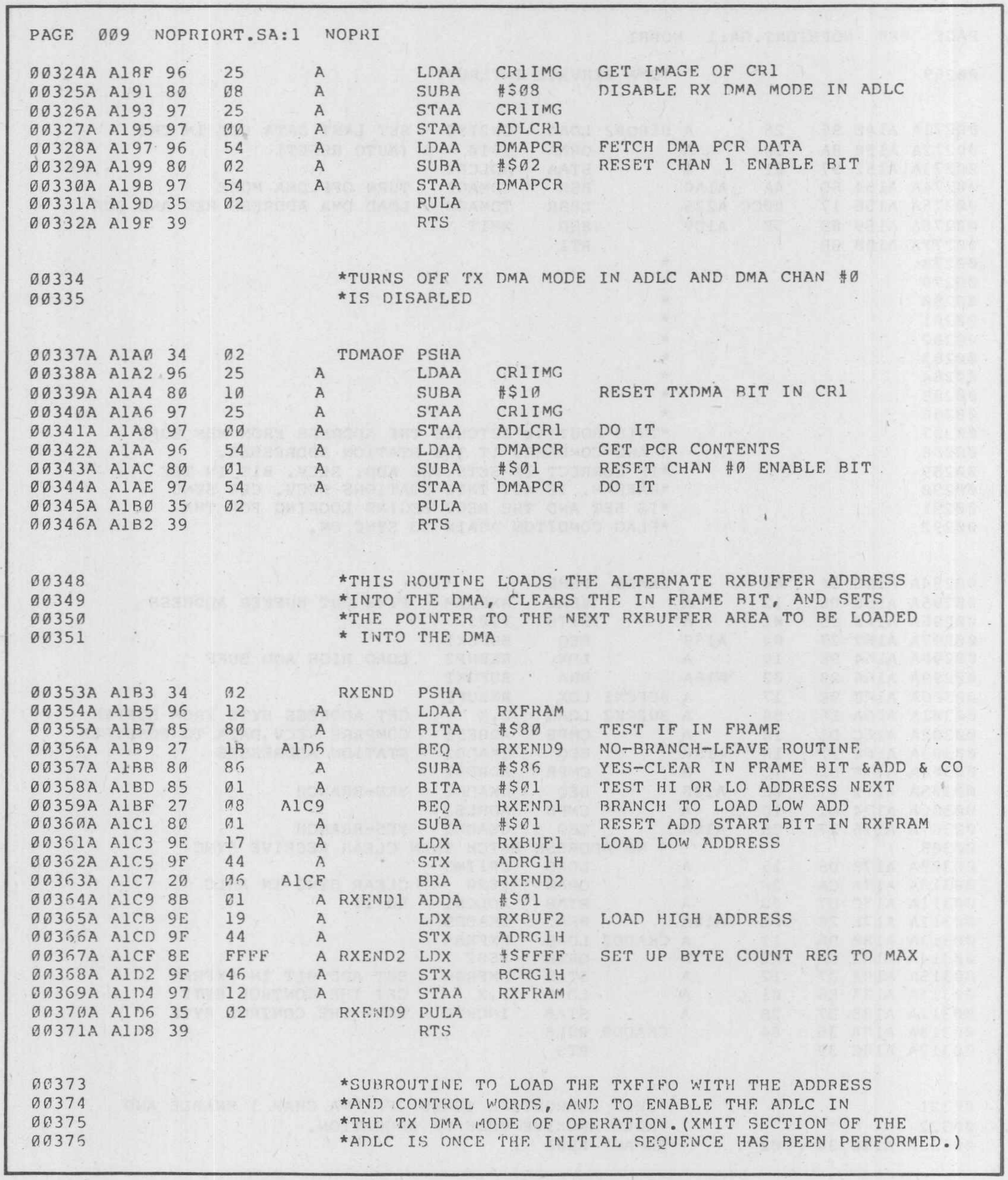
```
PAGE  ØØ7  NOPRIORT.SA:1  NOPRI

ØØ244A A118 8Ø   Ø8       A        SUBA   #$Ø8
ØØ245A A11A 97   2A       A        STAA   SR2IMG
ØØ246A A11C 2Ø   9A   AØB8         BRA    RSTA3R
ØØ247A A11E 8D   6D   A18D FCSERR  BSR    RDMAOF  TURN OFF DMA RECV MODE
ØØ248A A12Ø 17   Ø158 A27B         LBSR   CRCERR  INDICATE FCS ERROR OCCURED
ØØ249A A123 17   Ø176 A29C         LBSR   OUTFRM
ØØ25ØA A126 96   2A       A        LDAA   SR2IMG
ØØ251A A128 8Ø   1Ø       A        SUBA   #$1Ø
ØØ252A A12A 97   2A       A        STAA   SR2IMG
ØØ253A A12C 2Ø   95   AØC3         BRA    RSTA4R
ØØ254A A12E 8D   5D   A18D DCDERR  BSR    RDMAOF  TURN OFF DMA RECV MODE
ØØ255A A13Ø 17   Ø153 A286         LBSR   DCDLST  DATA CARRIER DETECT FROM
ØØ256                      *MODEM  WAS LOST
ØØ257A A133 17   Ø166 A29C         LBSR   OUTFRM
ØØ258A A136 96   2A       A        LDAA   SR2IMG
ØØ259A A138 8Ø   2Ø       A        SUBA   #$2Ø
ØØ26ØA A13A 97   2A       A        STAA   SR2IMG
ØØ261A A13C 16   FF79 AØB8         LBRA   RSTA3R
ØØ262A A13F 17   Ø14F A291 OVRN    LBSR   OVRUN1  SET BIT TO INDICATE OVERRUN
ØØ263A A142 17   Ø157 A29C         LBSR   OUTFRM
ØØ264A A145 96   2A       A        LDAA   SR2IMG
ØØ265A A147 8Ø   4Ø       A        SUBA   #$4Ø
ØØ266A A149 97   2A       A        STAA   SR2IMG
ØØ267A A14B 16   FF6A AØB8         LBRA   RSTA3R
```

Figure 11. Non-Priority Mode Program Listing (Sheet 7 of 12)



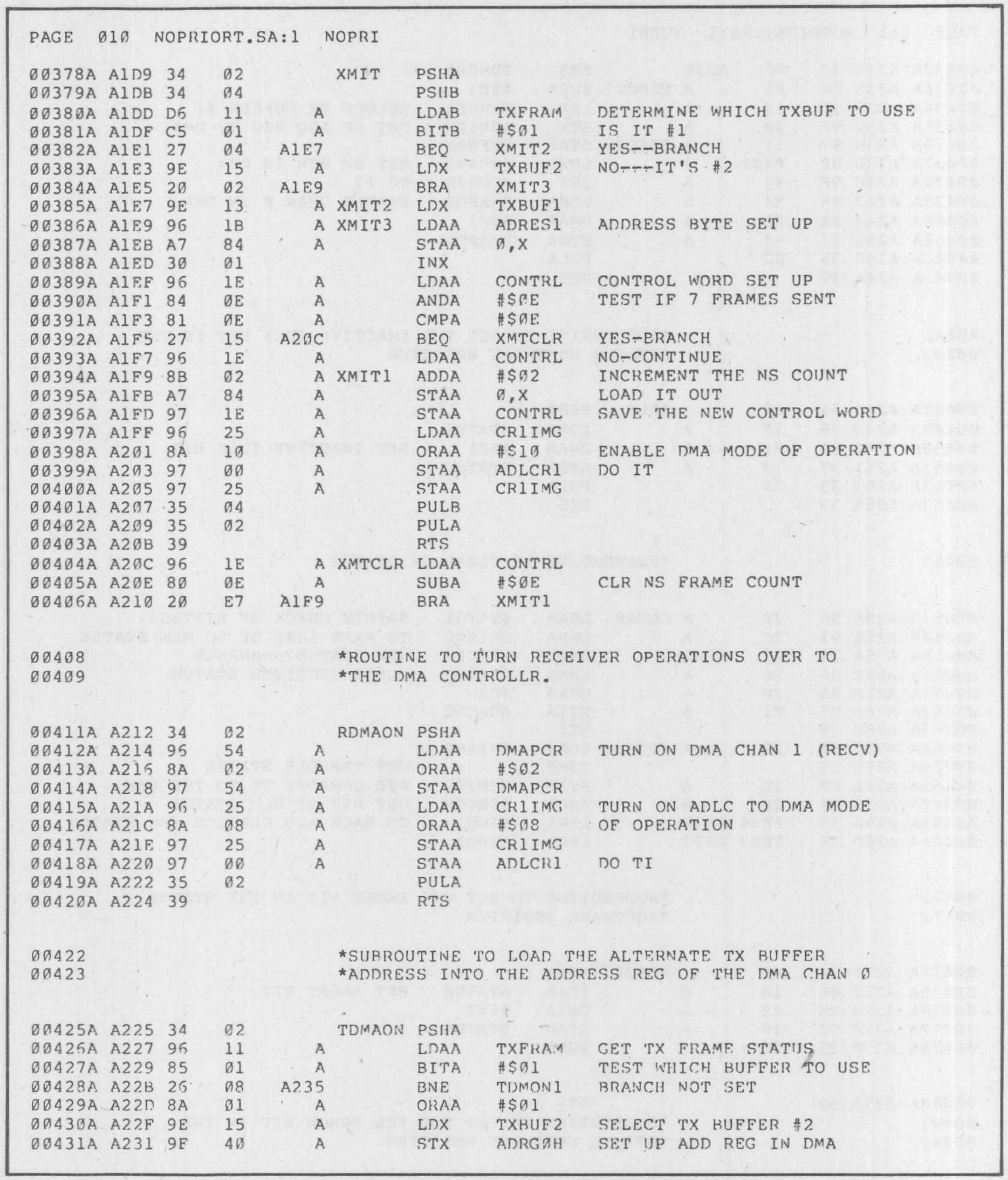
```
PAGE  ØØ8  NOPRIORT.SA:1  NOPRI

ØØ269                        *DMA SERVICE INTERUPT

ØØ271A A14E 96   26        A HIRQØ2 LDAA   CR2IMG   SET LAST DATA BIT IN CR2
ØØ272A A15Ø 8A   1Ø        A        ORAA   #$1Ø     (AUTO RESET)
ØØ273A A152 97   Ø1        A        STAA   ADLCR2
ØØ274A A154 8D   4A   A1AØ          BSR    TDMAOF   TURN OFF DMA MODE
ØØ275A A156 17   ØØCC A225          LBSR   TDMAON   LOAD DMA ADDRESS REG AND BCR
ØØ276A A159 8D   7E   A1D9          BSR    XMIT
ØØ277A A15B 3B                      RTI
ØØ278                        *
ØØ279                        *
ØØ28Ø                        *
ØØ281                        *
ØØ282                        *
ØØ283                        *
ØØ284                        *
ØØ285                        *
ØØ286                        *
ØØ287                        *THIS ROUTINE FETCHES THE ADDRESS FROM MEM BUFF
ØØ288                        * AND COMPARES IT THE STATION ADDRESSES.
ØØ289                        *IF CORRECT IT SETS THE ADD. RECV. BIT IN THE
ØØ29Ø                        *RXFRAM. IF NOT THIS STATIONS RECV. CLR SYNC
ØØ291                        *IS SET AND THE RECV BEGINS LOOKING FOR THE
ØØ292                        *FLAG CONDITON AGAIN TO SYNC ON.


ØØ294A A15C 34   Ø4          CKADD  PSHB
ØØ295A A15E D6   12        A        LDAB   RXFRAM   FIND OUT BUFFER ADDRESS
ØØ296A A16Ø C5   Ø1        A        BITB   #$Ø1
ØØ297A A162 27   Ø4   A168          BEQ    BUFCK1
ØØ298A A164 9E   19        A        LDX    RXBUF2   LOAD HIGH ADD BUFF
ØØ299A A166 2Ø   Ø2   A16A          BRA    BUFCK2
ØØ3ØØA A168 9E   17        A BUFCK1 LDX    RXBUF1
ØØ3Ø1A A16A E6   84        A BUFCK2 LDAB   Ø,X      GET ADDRESS BYTE FROM BUFFER
ØØ3Ø2A A16C D1   1B        A        CMPB   ADRES1   COMPARE RECV DATA TO POSSIBLE
ØØ3Ø3A A16E 27   1Ø   A18Ø          BEQ    CKADD2   STATION ADDRESSES
ØØ3Ø4A A17Ø D1   1C        A        CMPB   ADRES2
ØØ3Ø5A A172 27   ØC   A18Ø          BEQ    CKADD2   YES-BRANCH
ØØ3Ø6A A174 D1   1D        A        CMPB   ADRES3
ØØ3Ø7A A176 27   Ø8   A18Ø          BEQ    CKADD2   YES-BRANCH
ØØ3Ø8                        * NO ADDRESS MATCH THEN CLEAR RECEIVE SYNC
ØØ3Ø9A A178 D6   25        A        LDAB   CR1IMG
ØØ31ØA A17A CA   2Ø        A        ORAB   #$2Ø     CLEAR SYNC IN ADLC
ØØ311A A17C D7   ØØ        A        STAB   ADLCR1   DO IT
ØØ312A A17E 2Ø   ØA   A18A          BRA    CKADD9
ØØ313A A18Ø D6   12        A CKADD2 LDAB   RXFRAM
ØØ314A A182 CA   Ø2        A        ORAB   #$Ø2
ØØ315A A184 D7   12        A        STAB   RXFRAM   SET ADD BIT IN RXFRAM
ØØ316A A186 E6   Ø1        A        LDAB   1,X      GET THE CONTROL BYTE
ØØ317A A188 D7   2B        A        STAB   INCRTL   SAVE THE CONTROL BYTE
ØØ318A A18A 35   Ø4          CKADD9 PULB
ØØ319A A18C 39                      RTS


ØØ321                        *THIS SUBROUTINE TURNS OFF DMA CHAN 1 ENABLE AND
ØØ322                        *ADLC RECEIVE MODE OF OPERATION.
ØØ323A A18D 34   Ø2          RDMAOF PSHA
```

Figure 11. Non-Priority Mode Program Listing (Sheet 8 of 12)



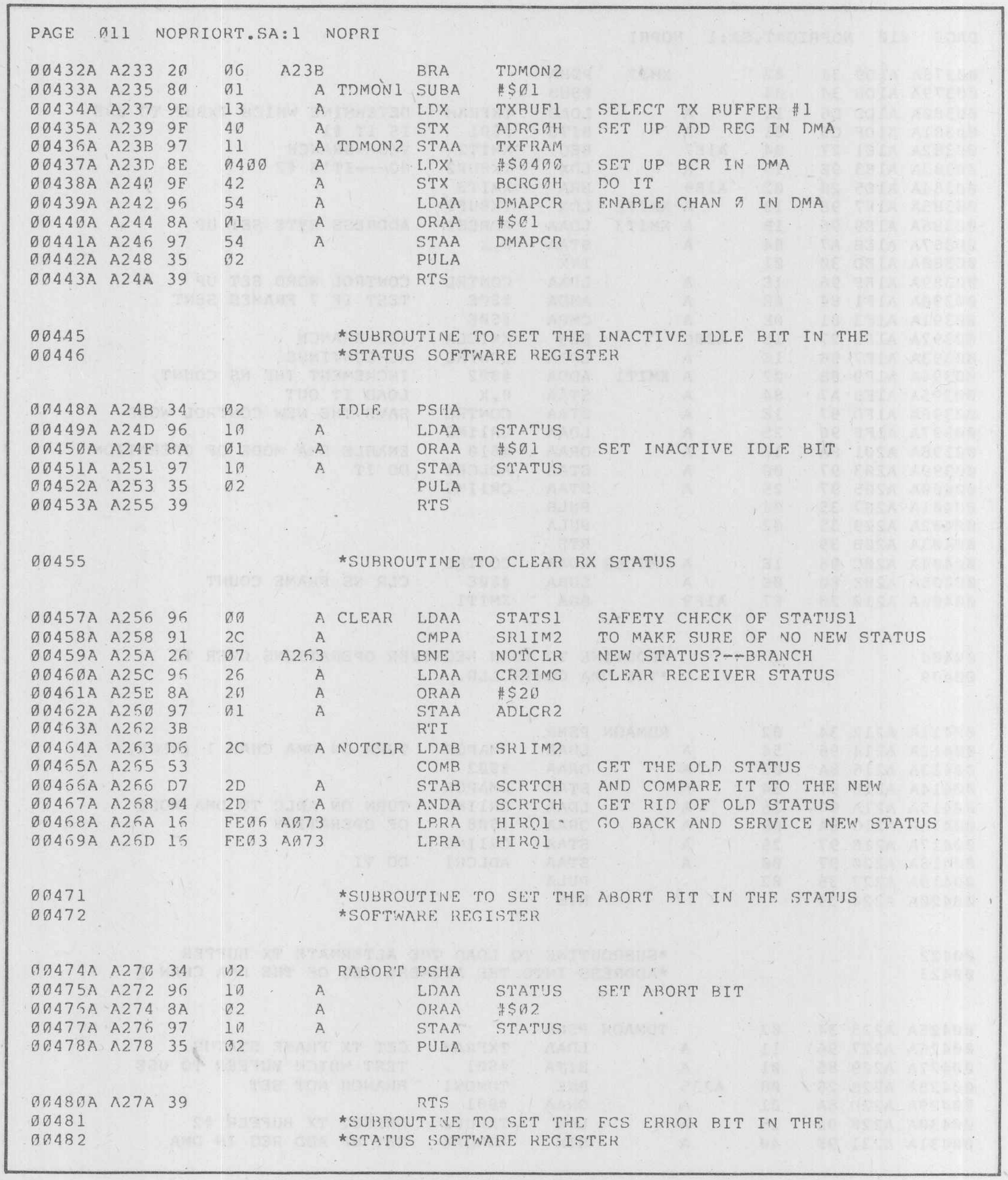
```
PAGE  ØØ9  NOPRIORT.SA:1   NOPRI

ØØ324A A18F 96   25       A            LDAA   CR1IMG    GET IMAGE OF CR1
ØØ325A A191 8Ø   Ø8       A            SUBA   #$Ø8      DISABLE RX DMA MODE IN ADLC
ØØ326A A193 97   25       A            STAA   CR1IMG
ØØ327A A195 97   ØØ       A            STAA   ADLCR1
ØØ328A A197 96   54       A            LDAA   DMAPCR    FETCH DMA PCR DATA
ØØ329A A199 8Ø   Ø2       A            SUBA   #$Ø2      RESET CHAN 1 ENABLE BIT
ØØ33ØA A19B 97   54       A            STAA   DMAPCR
ØØ331A A19D 35   Ø2                    PULA
ØØ332A A19F 39                         RTS

ØØ334                          *TURNS OFF TX DMA MODE IN ADLC AND DMA CHAN #Ø
ØØ335                          *IS DISABLED

ØØ337A A1AØ 34   Ø2            TDMAOF  PSHA
ØØ338A A1A2 96   25       A            LDAA   CR1IMG
ØØ339A A1A4 8Ø   1Ø       A            SUBA   #$1Ø      RESET TXDMA BIT IN CR1
ØØ34ØA A1A6 97   25       A            STAA   CR1IMG
ØØ341A A1A8 97   ØØ       A            STAA   ADLCR1    DO IT
ØØ342A A1AA 96   54       A            LDAA   DMAPCR    GET PCR CONTENTS
ØØ343A A1AC 8Ø   Ø1       A            SUBA   #$Ø1      RESET CHAN #Ø ENABLE BIT
ØØ344A A1AE 97   54       A            STAA   DMAPCR    DO IT
ØØ345A A1BØ 35   Ø2                    PULA
ØØ346A A1B2 39                         RTS

ØØ348                          *THIS ROUTINE LOADS THE ALTERNATE RXBUFFER ADDRESS
ØØ349                          *INTO THE DMA, CLEARS THE IN FRAME BIT, AND SETS
ØØ35Ø                          *THE POINTER TO THE NEXT RXBUFFER AREA TO BE LOADED
ØØ351                          * INTO THE DMA

ØØ353A A1B3 34   Ø2            RXEND   PSHA
ØØ354A A1B5 96   12       A            LDAA   RXFRAM
ØØ355A A1B7 85   8Ø       A            BITA   #$8Ø      TEST IF IN FRAME
ØØ356A A1B9 27   1B    A1D6            BEQ    RXEND9    NO-BRANCH-LEAVE ROUTINE
ØØ357A A1BB 8Ø   86       A            SUBA   #$86      YES-CLEAR IN FRAME BIT &ADD & CO
ØØ358A A1BD 85   Ø1       A            BITA   #$Ø1      TEST HI OR LO ADDRESS NEXT
ØØ359A A1BF 27   Ø8    A1C9            BEQ    RXEND1    BRANCH TO LOAD LOW ADD
ØØ36ØA A1C1 8Ø   Ø1       A            SUBA   #$Ø1      RESET ADD START BIT IN RXFRAM
ØØ361A A1C3 9E   17       A            LDX    RXBUF1    LOAD LOW ADDRESS
ØØ362A A1C5 9F   44       A            STX    ADRG1H
ØØ363A A1C7 2Ø   Ø6    A1CF            BRA    RXEND2
ØØ364A A1C9 8B   Ø1       A    RXEND1  ADDA   #$Ø1
ØØ365A A1CB 9E   19       A            LDX    RXBUF2    LOAD HIGH ADDRESS
ØØ366A A1CD 9F   44       A            STX    ADRG1H
ØØ367A A1CF 8E   FFFF     A    RXEND2  LDX    #$FFFF    SET UP BYTE COUNT REG TO MAX
ØØ368A A1D2 9F   46       A            STX    BCRG1H
ØØ369A A1D4 97   12       A            STAA   RXFRAM
ØØ37ØA A1D6 35   Ø2            RXEND9  PULA
ØØ371A A1D8 39                         RTS

ØØ373                          *SUBROUTINE TO LOAD THE TXFIFO WITH THE ADDRESS
ØØ374                          *AND CONTROL WORDS, AND TO ENABLE THE ADLC IN
ØØ375                          *THE TX DMA MODE OF OPERATION.(XMIT SECTION OF THE
ØØ376                          *ADLC IS ONCE THE INITIAL SEQUENCE HAS BEEN PERFORMED.)
```

Figure 11. Non-Priority Mode Program Listing (Sheet 9 of 12)



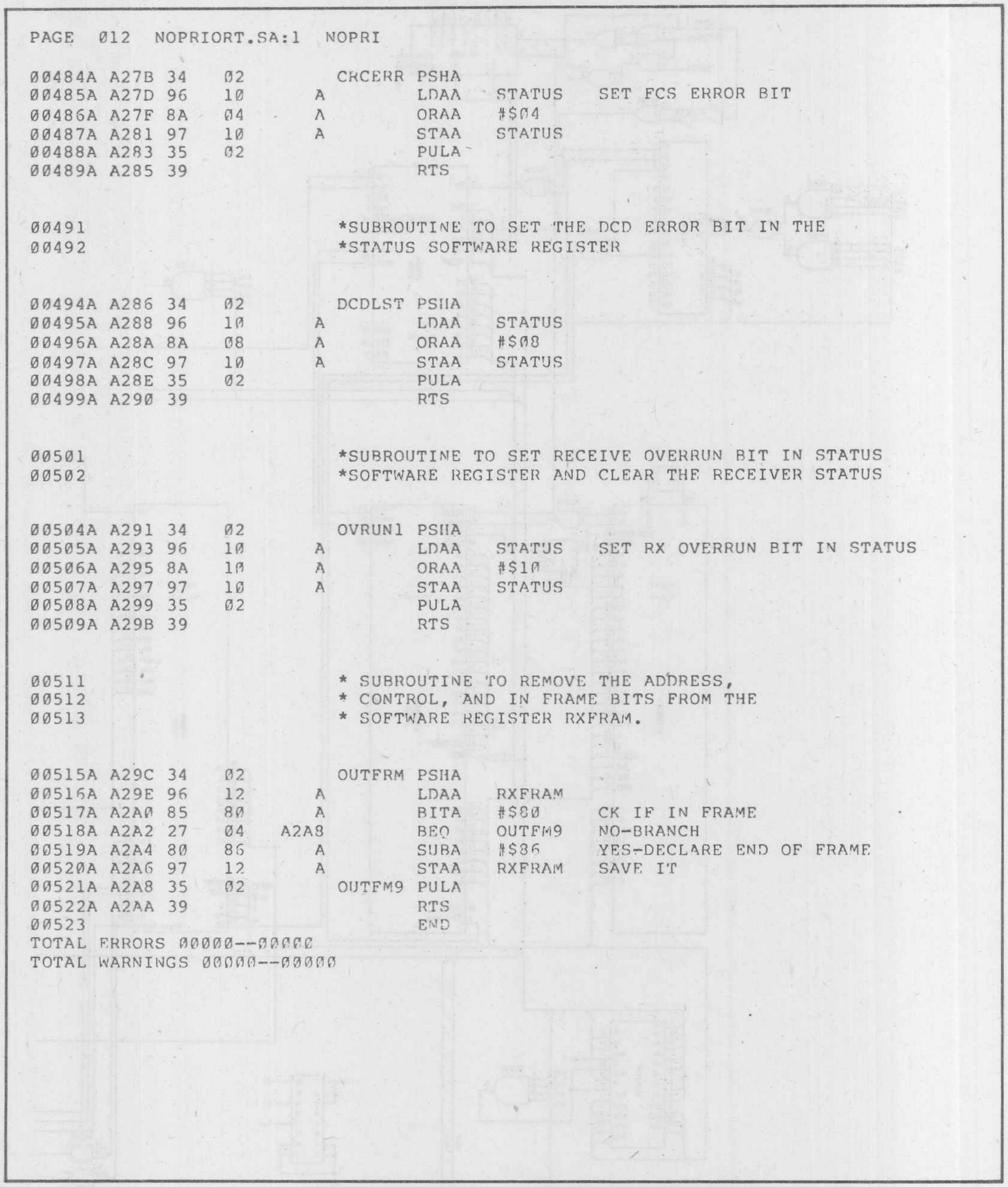
```
PAGE  Ø1Ø  NOPRIORT.SA:1  NOPRI

ØØ378A A1D9 34  Ø2           XMIT   PSHA
ØØ379A A1DB 34  Ø4                  PSHB
ØØ38ØA A1DD D6  11       A          LDAB  TXFRAM  DETERMINE WHICH TXBUF TO USE
ØØ381A A1DF C5  Ø1       A          BITB  #$Ø1    IS IT #1
ØØ382A A1E1 27  Ø4  A1E7            BEQ   XMIT2   YES--BRANCH
ØØ383A A1E3 9E  15       A          LDX   TXBUF2  NO---IT'S #2
ØØ384A A1E5 2Ø  Ø2  A1E9            BRA   XMIT3
ØØ385A A1E7 9E  13       A   XMIT2  LDX   TXBUF1
ØØ386A A1E9 96  1B       A   XMIT3  LDAA  ADRES1  ADDRESS BYTE SET UP
ØØ387A A1EB A7  84       A          STAA  Ø,X
ØØ388A A1ED 3Ø  Ø1                  INX
ØØ389A A1EF 96  1E       A          LDAA  CONTRL  CONTROL WORD SET UP
ØØ39ØA A1F1 84  ØE       A          ANDA  #$ØE    TEST IF 7 FRAMES SENT
ØØ391A A1F3 81  ØE       A          CMPA  #$ØE
ØØ392A A1F5 27  15  A2ØC            BEQ   XMTCLR  YES-BRANCH
ØØ393A A1F7 96  1E       A          LDAA  CONTRL  NO-CONTINUE
ØØ394A A1F9 8B  Ø2       A   XMIT1  ADDA  #$Ø2    INCREMENT THE NS COUNT
ØØ395A A1FB A7  84       A          STAA  Ø,X     LOAD IT OUT
ØØ396A A1FD 97  1E       A          STAA  CONTRL  SAVE THE NEW CONTROL WORD
ØØ397A A1FF 96  25       A          LDAA  CR1IMG
ØØ398A A2Ø1 8A  1Ø       A          ORAA  #$1Ø    ENABLE DMA MODE OF OPERATION
ØØ399A A2Ø3 97  ØØ       A          STAA  ADLCR1  DO IT
ØØ4ØØA A2Ø5 97  25       A          STAA  CR1IMG
ØØ4Ø1A A2Ø7 35  Ø4                  PULB
ØØ4Ø2A A2Ø9 35  Ø2                  PULA
ØØ4Ø3A A2ØB 39                      RTS
ØØ4Ø4A A2ØC 96  1E       A   XMTCLR LDAA  CONTRL
ØØ4Ø5A A2ØE 8Ø  ØE       A          SUBA  #$ØE    CLR NS FRAME COUNT
ØØ4Ø6A A21Ø 2Ø  E7  A1F9            BRA   XMIT1

ØØ4Ø8                        *ROUTINE TO TURN RECEIVER OPERATIONS OVER TO
ØØ4Ø9                        *THE DMA CONTROLLR.

ØØ411A A212 34  Ø2           RDMAON PSHA
ØØ412A A214 96  54       A          LDAA  DMAPCR  TURN ON DMA CHAN 1 (RECV)
ØØ413A A216 8A  Ø2       A          ORAA  #$Ø2
ØØ414A A218 97  54       A          STAA  DMAPCR
ØØ415A A21A 96  25       A          LDAA  CR1IMG  TURN ON ADLC TO DMA MODE
ØØ416A A21C 8A  Ø8       A          ORAA  #$Ø8    OF OPERATION
ØØ417A A21E 97  25       A          STAA  CR1IMG
ØØ418A A22Ø 97  ØØ       A          STAA  ADLCR1  DO TI
ØØ419A A222 35  Ø2                  PULA
ØØ42ØA A224 39                      RTS

ØØ422                        *SUBROUTINE TO LOAD THE ALTERNATE TX BUFFER
ØØ423                        *ADDRESS INTO THE ADDRESS REG OF THE DMA CHAN Ø

ØØ425A A225 34  Ø2           TDMAON PSHA
ØØ426A A227 96  11       A          LDAA  TXFRAM  GET TX FRAME STATUS
ØØ427A A229 85  Ø1       A          BITA  #$Ø1    TEST WHICH BUFFER TO USE
ØØ428A A22B 26  Ø8  A235            BNE   TDMON1  BRANCH NOT SET
ØØ429A A22D 8A  Ø1       A          ORAA  #$Ø1
ØØ43ØA A22F 9E  15       A          LDX   TXBUF2  SELECT TX BUFFER #2
ØØ431A A231 9F  4Ø       A          STX   ADRGØH  SET UP ADD REG IN DMA
```

Figure 11. Non-Priority Mode Program Listing (Sheet 10 of 12)



```
PAGE  Ø11  NOPRIORT.SA:1  NOPRI

ØØ432A A233 2Ø   Ø6   A23B        BRA    TDMON2
ØØ433A A235 8Ø   Ø1      A TDMON1 SUBA   #$Ø1
ØØ434A A237 9E   13      A        LDX    TXBUF1    SELECT TX BUFFER #1
ØØ435A A239 9F   4Ø      A        STX    ADRGØH    SET UP ADD REG IN DMA
ØØ436A A23B 97   11      A TDMON2 STAA   TXFRAM
ØØ437A A23D 8E   Ø4ØØ    A        LDX    #$Ø4ØØ    SET UP BCR IN DMA
ØØ438A A24Ø 9F   42      A        STX    BCRGØH    DO IT
ØØ439A A242 96   54      A        LDAA   DMAPCR    ENABLE CHAN Ø IN DMA
ØØ44ØA A244 8A   Ø1      A        ORAA   #$Ø1
ØØ441A A246 97   54      A        STAA   DMAPCR
ØØ442A A248 35   Ø2               PULA
ØØ443A A24A 39                    RTS

ØØ445                      *SUBROUTINE TO SET THE INACTIVE IDLE BIT IN THE
ØØ446                      *STATUS SOFTWARE REGISTER

ØØ448A A24B 34   Ø2        IDLE   PSHA
ØØ449A A24D 96   1Ø      A        LDAA   STATUS
ØØ45ØA A24F 8A   Ø1      A        ORAA   #$Ø1      SET INACTIVE IDLE BIT
ØØ451A A251 97   1Ø      A        STAA   STATUS
ØØ452A A253 35   Ø2               PULA
ØØ453A A255 39                    RTS

ØØ455                      *SUBROUTINE TO CLEAR RX STATUS

ØØ457A A256 96   ØØ      A CLEAR  LDAA   STATS1    SAFETY CHECK OF STATUS1
ØØ458A A258 91   2C      A        CMPA   SR1IM2    TO MAKE SURE OF NO NEW STATUS
ØØ459A A25A 26   Ø7   A263        BNE    NOTCLR    NEW STATUS?--BRANCH
ØØ46ØA A25C 96   26      A        LDAA   CR2IMG    CLEAR RECEIVER STATUS
ØØ461A A25E 8A   2Ø      A        ORAA   #$2Ø
ØØ462A A26Ø 97   Ø1      A        STAA   ADLCR2
ØØ463A A262 3B                    RTI
ØØ464A A263 D6   2C      A NOTCLR LDAB   SR1IM2
ØØ465A A265 53                    COMB             GET THE OLD STATUS
ØØ466A A266 D7   2D      A        STAB   SCRTCH    AND COMPARE IT TO THE NEW
ØØ467A A268 94   2D      A        ANDA   SCRTCH    GET RID OF OLD STATUS
ØØ468A A26A 16   FEØ6 AØ73        LBRA   HIRQ1     GO BACK AND SERVICE NEW STATUS
ØØ469A A26D 16   FEØ3 AØ73        LBRA   HIRQ1

ØØ471                      *SUBROUTINE TO SET THE ABORT BIT IN THE STATUS
ØØ472                      *SOFTWARE REGISTER

ØØ474A A27Ø 34   Ø2        RABORT PSHA
ØØ475A A272 96   1Ø      A        LDAA   STATUS    SET ABORT BIT
ØØ476A A274 8A   Ø2      A        ORAA   #$Ø2
ØØ477A A276 97   1Ø      A        STAA   STATUS
ØØ478A A278 35   Ø2               PULA

ØØ48ØA A27A 39                    RTS
ØØ481                      *SUBROUTINE TO SET THE FCS ERROR BIT IN THE
ØØ482                      *STATUS SOFTWARE REGISTER
```

Figure 11. Non-Priority Mode Program Listing (Sheet 11 of 12)



```
PAGE  Ø12  NOPRIORT.SA:1  NOPRI

ØØ484A A27B 34   Ø2         CRCERR PSHA
ØØ485A A27D 96   1Ø      A         LDAA   STATUS   SET FCS ERROR BIT
ØØ486A A27F 8A   Ø4      A         ORAA   #$Ø4
ØØ487A A281 97   1Ø      A         STAA   STATUS
ØØ488A A283 35   Ø2                PULA
ØØ489A A285 39                     RTS

ØØ491                       *SUBROUTINE TO SET THE DCD ERROR BIT IN THE
ØØ492                       *STATUS SOFTWARE REGISTER


ØØ494A A286 34   Ø2         DCDLST PSHA
ØØ495A A288 96   1Ø      A         LDAA   STATUS
ØØ496A A28A 8A   Ø8      A         ORAA   #$Ø8
ØØ497A A28C 97   1Ø      A         STAA   STATUS
ØØ498A A28E 35   Ø2                PULA
ØØ499A A29Ø 39                     RTS

ØØ5Ø1                       *SUBROUTINE TO SET RECEIVE OVERRUN BIT IN STATUS
ØØ5Ø2                       *SOFTWARE REGISTER AND CLEAR THE RECEIVER STATUS


ØØ5Ø4A A291 34   Ø2         OVRUN1 PSHA
ØØ5Ø5A A293 96   1Ø      A         LDAA   STATUS   SET RX OVERRUN BIT IN STATUS
ØØ5Ø6A A295 8A   1Ø      A         ORAA   #$1Ø
ØØ5Ø7A A297 97   1Ø      A         STAA   STATUS
ØØ5Ø8A A299 35   Ø2                PULA
ØØ5Ø9A A29B 39                     RTS

ØØ511                       * SUBROUTINE TO REMOVE THE ADDRESS,
ØØ512                       * CONTROL, AND IN FRAME BITS FROM THE
ØØ513                       * SOFTWARE REGISTER RXFRAM.


ØØ515A A29C 34   Ø2         OUTFRM PSHA
ØØ516A A29E 96   12      A         LDAA   RXFRAM
ØØ517A A2AØ 85   8Ø      A         BITA   #$8Ø     CK IF IN FRAME
ØØ518A A2A2 27   Ø4   A2A8         BEQ    OUTFM9   NO-BRANCH
ØØ519A A2A4 8Ø   86      A         SUBA   #$86     YES-DECLARE END OF FRAME
ØØ52ØA A2A6 97   12      A         STAA   RXFRAM   SAVE IT
ØØ521A A2A8 35   Ø2         OUTFM9 PULA
ØØ522A A2AA 39                     RTS
ØØ523                              END
TOTAL ERRORS ØØØØØ--ØØØØØ
TOTAL WARNINGS ØØØØØ--ØØØØØ
```

Figure 11. Non-Priority Mode Program Listing (Sheet 12 of 12)

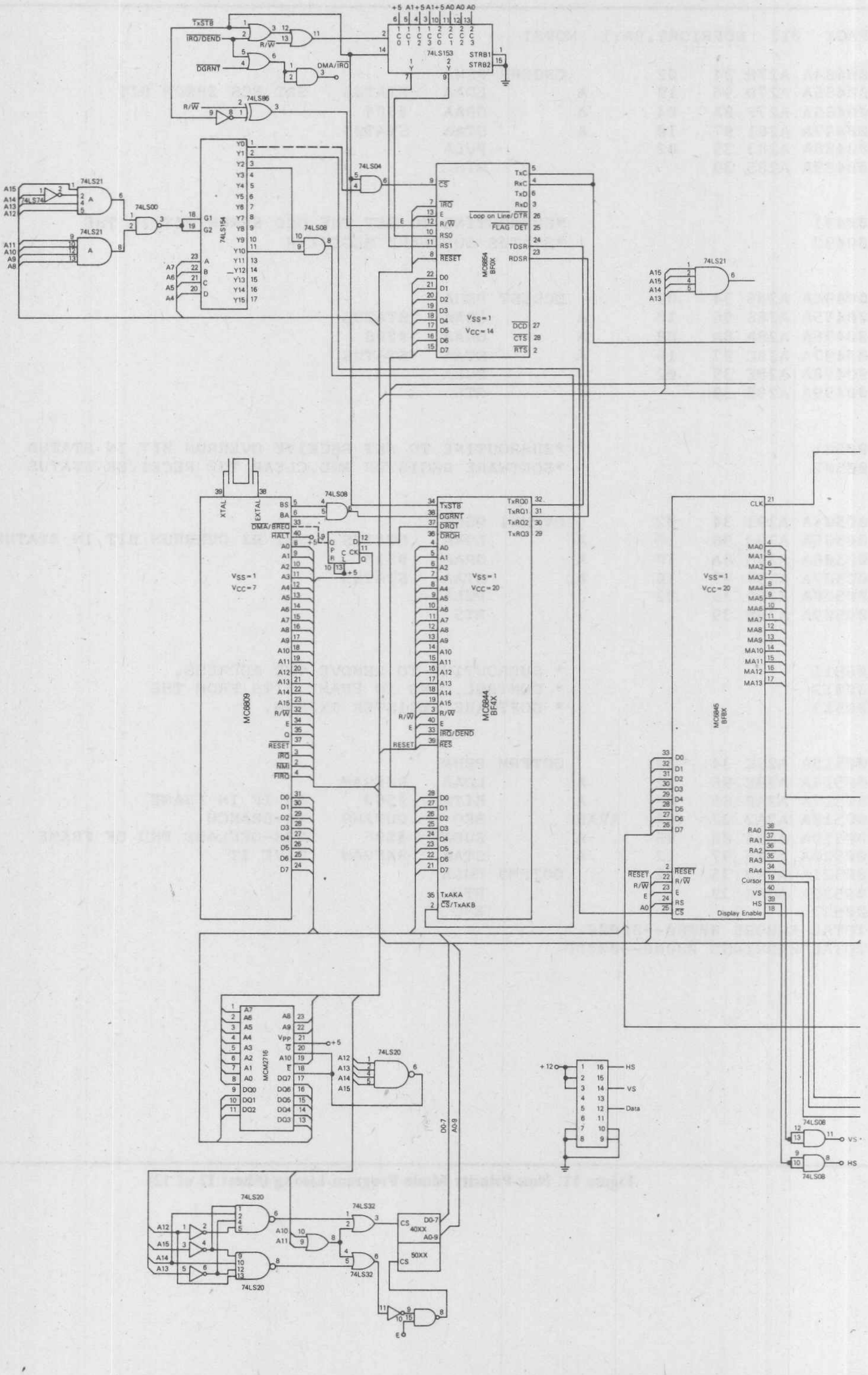
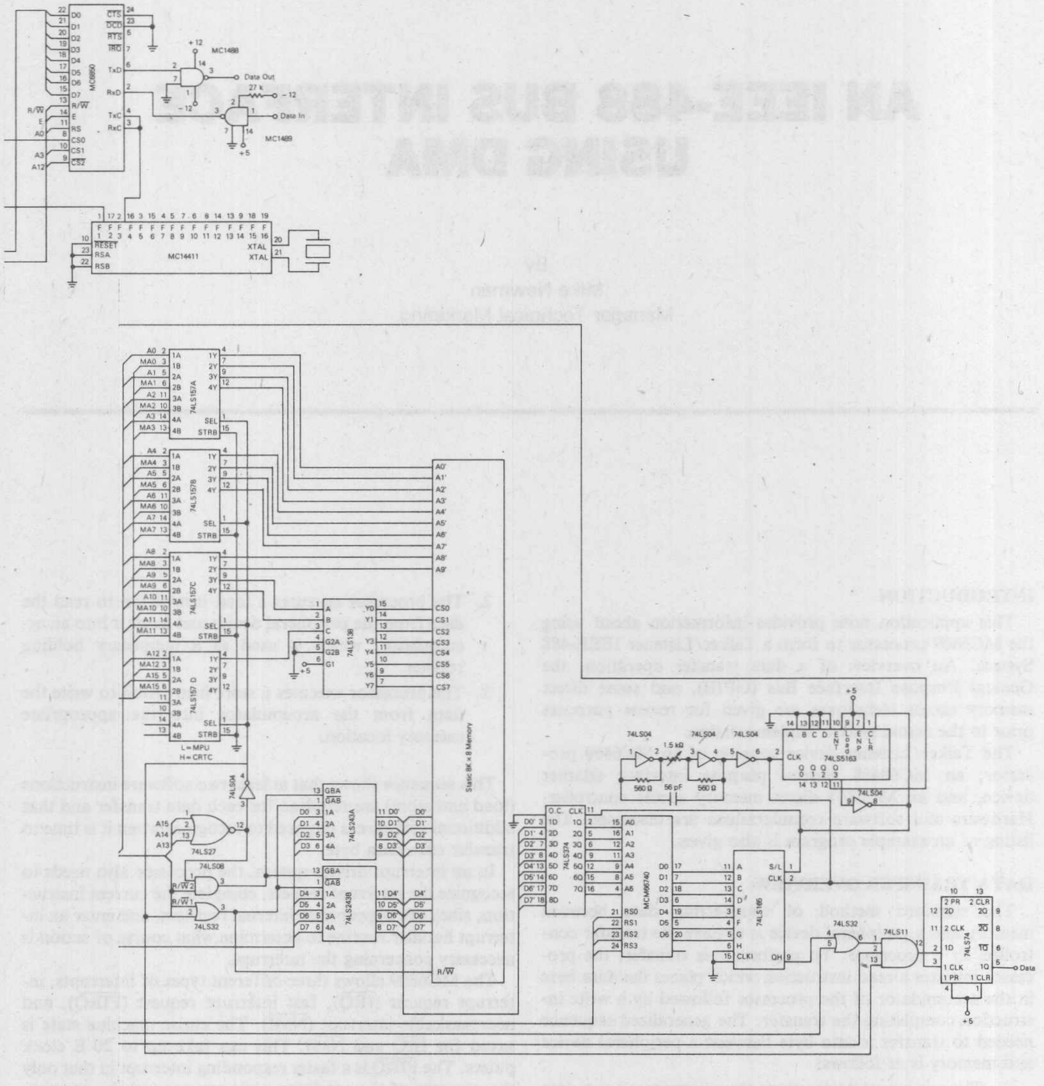


Figure 12. Intelligent Terminal Schematic Diagram


MC6850
CTS
DCD
RTS
IRQ
TxD
RxD
TxC
RxC
D0
D1
D2
D3
D4
D5
D6
D7
R/W
E
A0
A3
A12
RS
CS0
CS1
CS2
+12
MC1488
Data Out
27 k
−12
Data In
MC1489
+5
MC14411
RESET
RSA
RSB
XTAL
74LS157A
74LS157B
74LS157C
74LS157 D
SEL
STRB
L = MPU
H = CRTC
74LS138
Static 8K × 8 Memory
74LS04
74LS27
74LS08
74LS32
74LS243A
74LS243B
GBA
GAB
74LS374
MCM6740
74LS165
74LS163
1.5 kΩ
560 Ω
56 pF
74LS11
74LS74
Data

AN-831

AN IEEE-488 BUS INTERFACE USING DMA

By
Mike Newman
Manager Technical Marketing

INTRODUCTION

This application note provides information about using the MC6809 processor to form a Talker/Listener IEEE-488 System. An overview of a data transfer operation, the General Purpose Interface Bus (GPIB), and some direct memory access techniques are given for review purposes prior to the actual system implementation.

The Talker/Listener device consists of an MC6809 processor, an MC68488 general purpose interface adapter device, and an MC6844 direct memory access controller. Hardware and software considerations are discussed. The listing of an example program is also given.

DATA TRANSFER OVERVIEW

The standard method of transferring data between memory and a peripheral device is to have the transfer controlled by a processor. To perform this transfer, the processor initiates a read instruction which places the data byte in the accumulator of the processor followed by a write instruction completing the transfer. The generalized sequence needed to transfer a data byte between a peripheral device and memory is as follows:

1. The peripheral device alerts the processor when a data byte is to be transferred. The processor recognizes this through either an interrupt sequence or a polling procedure.
2. The processor executes a load instruction to read the data from the peripheral device and loads it into an accumulator, which is used as a temporary holding register.
3. The processor executes a store instruction to write the data from the accumulator into the appropriate memory location.

This sequence shows that at least two software instructions (load and store) are required for each data transfer and that additional software is required to recognize when it is time to transfer each data byte.

In an interrupt driven system, the processor also needs to recognize the interrupt request, complete the current instruction, stack the appropriate internal registers, and enter an interrupt handler routine to determine what course of action is necessary concerning the interrupt.

The MC6809 allows three different types of interrupts, interrupt request ($\overline{\text{IRQ}}$), fast interrupt request ($\overline{\text{FIRQ}}$), and non-maskable interrupt ($\overline{\text{NMI}}$). The entire machine state is saved for $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$. This can take up to 20 E clock pulses. The $\overline{\text{FIRQ}}$ is a faster responding interrupt in that only the contents of the condition code register and the program counter are saved. This can take up to 12 E clock pulses. If any other internal registers need to be saved when using $\overline{\text{FIRQ}}$, they need to be saved via software.

An alternate to using interrupts is to use a polling procedure to recognize when a data byte is to be transferred. In a polling system, the processor monitors the status of the peripheral device using a software polling routine. This routine normally consists of one or more load instructions, each of which is followed by test instructions (e.g., bit test). If the processor is dedicated to continually monitoring the peripheral device, then a polling procedure provides a faster response than the interrupt driven system. Even though the polling procedure can handle data at a faster rate than the interrupt procedure, it still requires a set of software instructions to handle each data byte. Since many systems do not require extremely high data rates, either of these procedures should be more than adequate.

A direct memory access (DMA) method of operation is required when data needs to be transferred at a high rate (approximately 10K bytes per second). For example, when a peripheral device is receiving data from a high-speed disk system. This method of operation does not use processor software to perform the transfer; therefore, the maximum data rate is only limited by either the processor system clock or the peripheral device speed. Within the DMA method of operation there are two modes, halt burst and cycle stealing. The advantages and disadvantages of these modes will be discussed later. To use DMA, a processor with special DMA features and a device called a direct memory access controller (DMAC) is needed. Some peripheral devices may have DMA features included; therefore, a separate DMA device is not required. The MC6844 direct memory access controller is a device that is designed to perform the data transfer between memory and a peripheral in the most efficient manner. It does this by automatically performing the necessary read/write sequence and sending the data byte directly from the peripheral device/memory to memory/peripheral. The processor is free to do other things at this time.

To use the DMA method of operation, the controller must first be initialized (direction of data transfer, starting memory location, etc.) by the processor. Then, once a transfer is requested by the peripheral device, the appropriate handshake sequence needs to occur allowing the processor to give up control of the system to the DMAC, remove itself from the bus, and allow the transfer to take place. Once the transfer is complete, the DMAC returns control to the processor and removes itself from the bus. The timing for the handshake between the processor and DMAC and the actual data transfer must be very precise in order to maximize the transfer rate. The MC6809 provides the necessary handshake signals and timing to allow DMA operations to occur with maximum efficiency.

GENERAL PURPOSE INTERFACE BUS OVERVIEW

The purpose of the IEEE-488 Standard is to allow the interconnection of programmable instruments with a minimum amount of engineering. The intent is to remove the need for adapters and numerous types of patching cables when different types of instruments are connected together in a system. The IEEE-488 Standard allows system configurations using programmable instruments, calculators, and other types of peripheral devices produced by different manufacturers. The IEEE-488 Standard provides a set of rules for establishing an unambiguous communications link which produces a high degree of compatibility, while maintaining flexibility between independently manufactured products. The standard defines a special bus structure known as the general purpose interface bus (GPIB). Any device meeting the specifications described in the standard is directly compatible with the GPIB without the need for an adapter. The GPIB can be thought of as the communications link between two or more instruments, as shown in Figure 1. The devices on the bus are considered to be either listeners, talkers, or controllers. Listeners receive data from talkers or controllers; talkers send data to listeners; controllers control and synchronize the devices on the bus.

This communications link is a parallel bus in contrast to the serial links commonly associated with most other types of data commuications. Bit-parallel, byte-serial format is used for communications on the GPIB. Bit-parallel refers to a set of concurrent data bits being transmitted simultaneously, and the byte-serial refers to consecutive bytes being carried over the data link in a serial fashion. The GPIB consists of 16 transmission lines which are categorized into:

1. eight data bus lines
2. three data byte transfer control or handshake lines
3. five general interface management lines.

The eight data bus lines are used to transfer data from talkers to listeners. They are also used to transfer interface messages from a controller (when used) to various devices. All transfers are asynchronous and occur according to the three-wire handshake. This handshake synchronizes the talker readiness to transmit data with the listeners readiness to receive data.

At any point in time, an individual device on the GPIB is either idle, monitoring the activity on the bus, a talker sending data to listeners, a listener receiving data from the talker, or a controller controlling the activity of the bus.

A minimum system may consist of just one talker and one listener. For example, a dedicated voltmeter could be outputting data to a dedicated printer. In such a system it is necessary for the two devices to have interfacing options that allow local messages to assign them as either a talker or a listener. This assignment is most likely made at power-up and does not change thereafter.

Many devices are both talkers and listeners. A programmable multimeter, for instance, is a listener when receiving its programmed instructions and a talker when sending its data to another device such as a printer or disk. There can be many listeners at one time, but only one talker.

Controllers are used in systems where it is desirable to be able to change the functions of devices that can be both talkers and listeners. The word controller in the context of the IEEE-488 Standard refers to a special device that connects to the GPIB. It is a complete unit in itself and directs the flow of data by assigning devices to be either listeners or talkers. It can also interrupt data flow and command specific

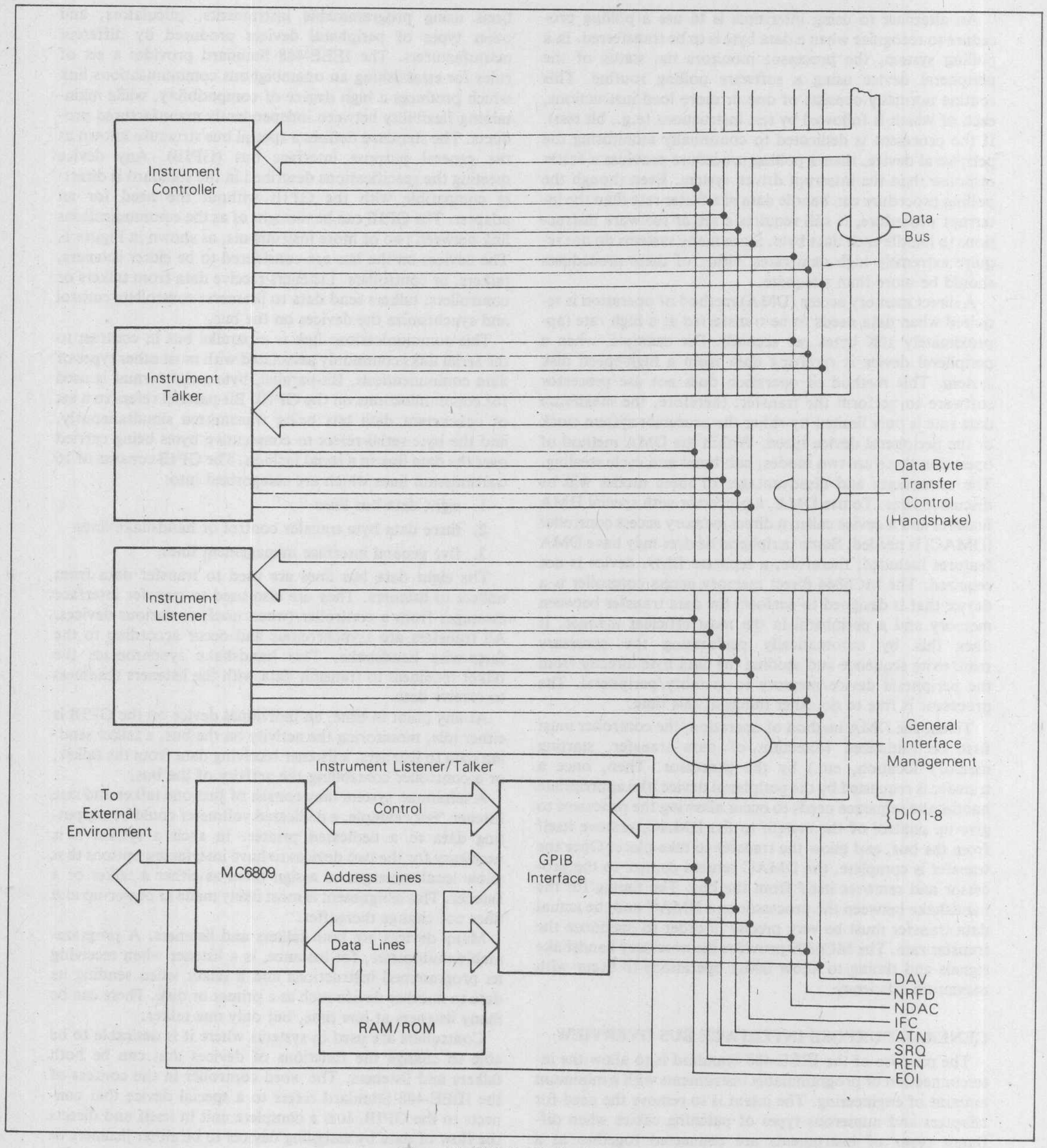


Figure 1. IEEE-488 Bus System

actions to be taken within the devices. The word controller does not refer to a processor on the instrument side of the GPIB.

The controller alters activity on the bus by sending interface messages. The active controller is the only device capable of sending interface messages. It does this in one of two ways:

1. Uniline Messages — The controller can send a message over any one of the five general interface management lines.
2. Multiline Messages — The controller can send a message over the eight data bus lines. It does this by asserting the attention ($\overline{\text{ATN}}$) general interface management line signifying to all devices on the bus that the eight bus lines contain a multiline message rather than data.

These messages are interface commands which do not interact directly with the measurement process of an instrument. They interact only with the interface logic within connected devices. The primary purpose of these messages is to carry out the proper protocol in setting up, maintaining, and terminating an orderly flow of device dependent messages. (Device dependent messages refer to the information being sent by the addressed talker device to the addressed listener devices and not the messages used to control the interface.) The multiline and uniline messages are used to address devices to be talkers or listeners, to tell a device to ignore or not ignore front panel settings, to inquire about any problems the device has, to reset the interface circuitry, to begin making a measurement, etc.

Addresses are assigned to each device so it can respond to addressed commands. Using this address, the controller can pick out a specific device and instruct it to be either a talker or listener. The controller does not assign addresses; this must come from some external means such as a set of switches attached to the device or a subroutine resident in the software controlling the device. The address is placed in the GPIB interface for the device during an initialization sequence. Once resident in the interface circuitry, the device can respond to addressed commands. The address is a 15-bit digital number that allows the controller to talk to a particular device.

A talker sends a data byte over the GPIB to a listener or listeners using an asynchronous three-wire handshake. The transfer begins when the talker asserts data available ($\overline{\text{DAV}}$) and is completed when the slowest listener accepts the data byte by asserting data accepted (DAC). The third handshake line, ready for data (RFD), is used to let the talker know that the listeners are ready for data. There are actually four states in a data transfer.

1. The talker generates a new byte.
2. The states of the data bus signal lines settle.
3. The listeners accept the data.
4. The listeners become ready for the next byte.

Since there can be many listeners (maximum of 14; 14 listeners plus one talker for 15 devices maximum), it is possible to have some that respond very quickly (e.g., a disk) and some that respond slowly (e.g., a teletype) to the same data byte. In this case, the overall speed of transmission over the bus is governed by, and cannot exceed the response rate of the slowest active listener.

The following example is given to demonstrate the command structure of the GPIB bus and how this relates to the internal processor system of a device. In this example, a device assigned a GPIB address of 3 is to send a block of data using DMA to a device assigned a GPIB address of 1. One procedure for establishing this link is as follows:

1. Once connected to the system (other devices may also be connected to this system), the power to each device is turned on. The unique GPIB address for each device is placed in its respective general purpose interface adapter(MC68488) during a power-on initialization sequence by the processor along with other appropriate initialization procedures.
2. The GPIB controller takes control of the bus by asserting $\overline{\text{ATN}}$ and, with the appropriate interface commands, clears all devices on the bus. Remember that the GPIB controller only talks to the general purpose interface adapter (MC68488) and not directly to the device processor. It is up to the MC68488 to alert the processor through either a polling or an interrupt routine when the processor needs to take action.
3. The GPIB controller makes device 3 a listener and sends it information concerning the upcoming DMA block transfer. The MC68488 interprets these bytes as data and flags the processor on a per byte basis. The processor software interprets these data bytes as device dependent messages. These messages provide information such as the precise data to be sent, the format of the data, mode of processor transfer — DMA or non-DMA, etc.
4. The GPIB controller clears device 3 and makes device 1 a listener. Step 3 is repeated to device 1; however, in this case the information pertains to device 1 as the recipient of the block of data.
5. The GPIB controller leaves device 1 in the listen mode and assigns device 3 to be a talker. The GPIB controller now releases control of the GPIB, by negating $\overline{\text{ATN}}$ allowing the data transfer to take place.
6. The talker now sends the data in a byte-per-byte sequence to the listener. Each byte is accepted by the listener according to the asynchronous handshake.
7. When the last byte is sent, the talker alerts both the listeners and the controller that the next byte is the last byte of the data block by asserting the $\overline{\text{EOI}}$ general interface management line. The end of a data string can also be indicated by a special sequence of data characters (e.g., carriage return followed by line feed) which are interpreted in software.
8. The GPIB controller can now reconfigure the bus for the next data transfer.

DIRECT MEMORY ACCESS MODES OF OPERATION

The MC6844 (DMAC) is capable of three modes of DMA transfer, they are: three-state cycle steal, halt cycle steal, and halt burst. Only the halt burst and three-state cycle steal modes were considered for this system controller since the MC6809 can handle these modes efficiently. The characteristics of these modes are:

Halt Burst Mode — In this mode, the processor is halted and removed from the bus (the appropriate output lines placed in the high-impedance state) while a block of data is transferred between memory and the GPIB. The DMAC manages the control lines (e.g., $R/\overline{W}$, address lines, etc.) and keeps track of how many bytes have been transferred, returning control to the processor when the last byte has been sent. Therefore, if the DMAC has been programmed for a 16K byte transfer, the processor is removed from the bus at the beginning of the transfer and is not brought back on the bus until all 16K bytes have been transferred. This mode of operation provides the direct memory access system with the highest data transfer rate capability; however, even though the DMAC can operate at this high data transfer rate, the actual transfer rate cannot exceed the rate at which the GPIA can issue request.

The main advantage of the halt burst mode is the high data transfer capabilities. The main disadvantage is that the processor is halted during the entire transfer.

Three-State Cycle Steal — In this mode, the processor is neither halted nor removed from the bus for any extended length of time. Rather, the operations of the processor are temporarily suspended and the processor removed from the bus (the appropriate output lines are placed in the high-impedance state) while the DMAC transfers one byte of data. At the end of this transfer, control is given back to the processor. If a block of data is being transferred, the processor is placed back on the bus between each transfer for at least one processor clock cycle. This method of direct memory access operation is slower than the halt burst mode, but does not cause the processor to relinquish control of the bus for long periods of time.

The MC68488 GPIA cannot issue direct memory access transfer requests at a high enough rate to take advantage of the high data transfer rate capabilities of the halt burst mode. This is due to the inherent functionality of the GPIA and the IEEE-488 bus. The GPIA must acknowledge each data byte on the bus before it can issue the next transfer request. This can take up to seven processor clock cycles. In addition, the data on the GPIB is transferred in an asynchronous fashion and cannot be transferred at a rate faster than it can be accepted by the slowest listening device. In many applications the data rate on the bus can be very slow; and as a result, the transfer requests being issued to the DMAC for the device in question could be occurring at a rate considerably slower than one every seven processor clock cycles. If the halt burst mode were used, the MC6809 would be inactive during the non-DMA time that the DMAC is waiting for a transfer request from the GPIA. To take advantage of the non-DMA time and allow the MC6809 to do processing during this time, the three-state cycle steal mode of operation was chosen. Now the processor can be brought back on the bus to perform tasks in between DMA transfers.

SYSTEM OVERVIEW

The DMA system given in this application is essentially divided into seven major circuits as shown in Figure 2. The following paragraphs provide a brief description of each of these circuits. A description of how these circuits are interconnected as a working system is also provided.

MC6809 MICROPROCESSOR — The MC6809 is an advanced member of the MC6800 microprocessor family. It has special DMA capabilities that allow highly efficient DMA data transfers. During non-DMA conditions, the MC6809 continues to operate the system. The MC6809 initializes the other circuits in the system (e.g., MC6844, MC68488, and the display). At other times, it can be used to execute special purpose programs.

MC6844 DIRECT MEMORY ACCESS CONTROLLER — The MC6844 requests control of the bus from the MC6809 and issues the appropriate commands (via the $R/\overline{W}$ line, grant line, and address lines) to perform data transfers. The direct memory access controller never actually receives the data, it directs the flow of the data from one place to the other at the correct time and in the required direction. After the transfer is complete, the MC6844 returns control to the MC6809.

MC68488 GENERAL PURPOSE INTERFACE ADAPTER — The MC68488 provides the interface between the IEEE-488 bus and a processor controlled system. After initialization, the GPIB system controller places the MC68488 in either a talk mode when it is to send data or in a listen mode if it is to receive data.

SYNCHRONIZATION CIRCUITRY — The synchronization circuitry performs two functions: 1) It synchronizes the DMA request signal from the DMAC with the quadrature (Q) signal from the MC6809 by ensuring that the DMA request is not presented to the MC6809 $\overline{\text{DMA}}/\overline{\text{BREQ}}$ input during the last quarter cycle of the E signal. 2) The end or identify ($\overline{\text{EOI}}$) line on the general purpose interface byte is used by a talker to indicate to the listeners that the next data byte received is the last byte of a block. In this system, this line is applied to the synchronization circuitry to disable DMA transfer requests to the MC6809. The $\overline{\text{EOI}}$ input to the synchronization circuitry is used only when DMA transfers are being made from the GPIA to memory.

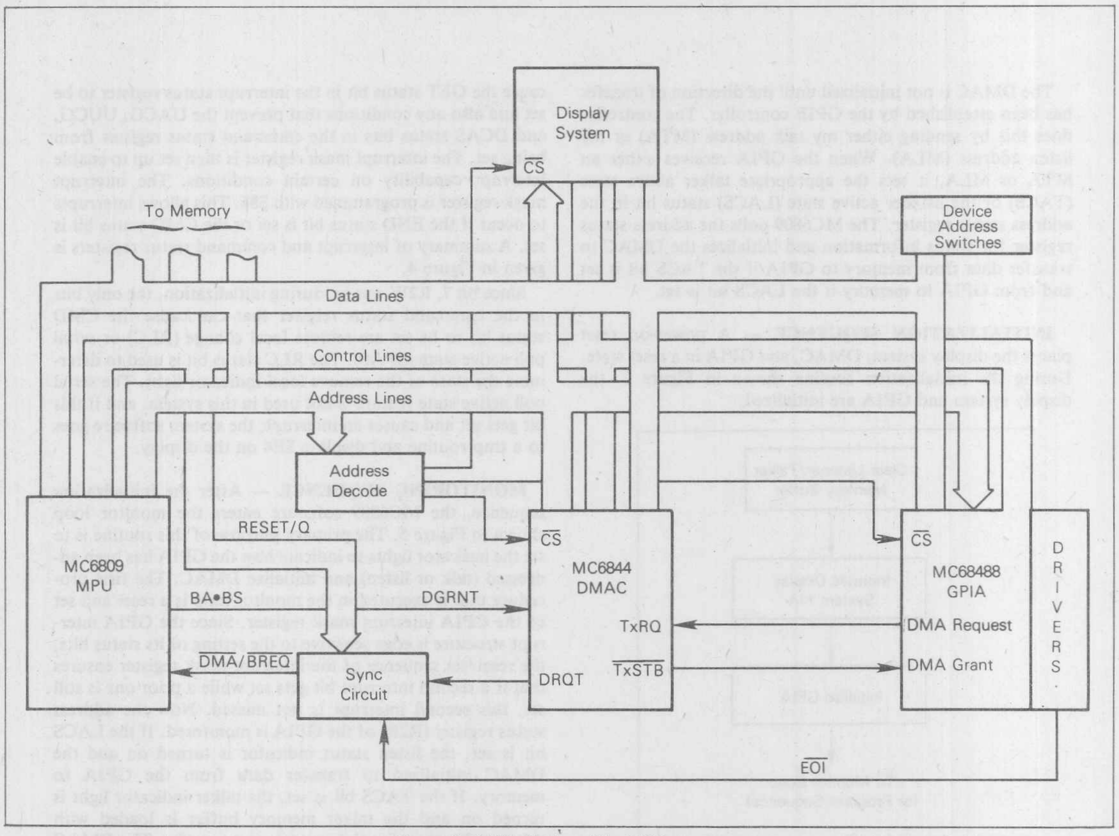


Figure 2. DMA System Block Diagram

DISPLAY SYSTEM — The display system provides a visual indication of: how many blocks of data have been transferred, whether the device is a talker or a listener, and whether the device is in a local or remote state.

DEVICE ADDRESS SWITCHES — This set of toggle switches is isolated from the data bus by buffers. They are used to select the device address for the GPIB, i.e., the address that the GPIB controller uses when sending addressed commands. These switches are manually set to the desired address. The MC6809 initialization program reads the address by enabling the buffers and places it in the MC68488.

OPERATION

This system allows bidirectional data transfers in either a non-DMA mode or a three-state cycle steal DMA mode.

The software is a simplified test program which demonstrates the DMA capability of the system and is not intended as a general purpose application program. The test program only allows data transfers in the DMA mode. After the initialization sequence, the MC6809 simply monitors the GPIA for the direction of data transfer. The DMAC is not initialized during the system initialization sequence. The software initializes the display and GPIA and then enters a monitor loop leaving the DMAC disabled. When the direction of transfer is established, the MC6809 branches to a routine that initializes the DMAC accordingly. For system simplicity, the characteristics of the transfer (e.g., number of bytes to be transferred and beginning memory address) are constants in the DMAC initialization routine. The only variable is direction and this is determined by monitoring the address status register of the GPIA.

The DMAC is not initialized until the direction of transfer has been established by the GPIB controller. The controller does this by sending either my talk address (MTA) or my listen address (MLA). When the GPIA receives either an MTA or MLA, it sets the appropriate talker active state (TACS) or the listener active state (LACS) status bit in the address status register. The MC6809 polls the address status register for status information and initializes the DMAC to transfer data from memory to GPIA if the TACS bit is set and from GPIA to memory if the LACS bit is set.

INITIALIZATION SEQUENCE — A power-on reset places the display system, DMAC, and GPIA in a reset state. During the initialization routine shown in Figure 3, the display system and GPIA are initialized.

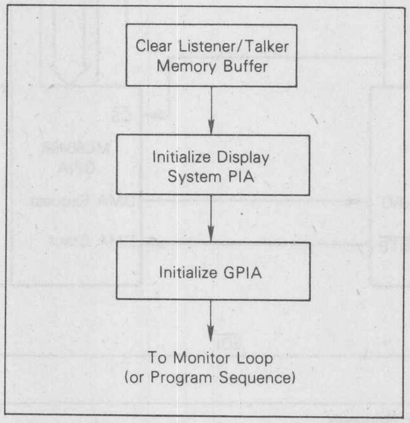

Clear Listener/Talker Memory Buffer
Initialize Display System PIA
Initialize GPIA
To Monitor Loop (or Program Sequence)

Figure 3. Initialization Routine Flow Chart

The display system has an MC6821 peripheral interface adapter (PIA) which drives two seven-segment displays and three indicator lights. During initialization, the PIA lines that control the seven-segment displays are programmed as outputs and set to zero causing the displays to read a $00. In addition, the lines that control the indicator lights are programmed as outputs and set to zero keeping the indicator light off.

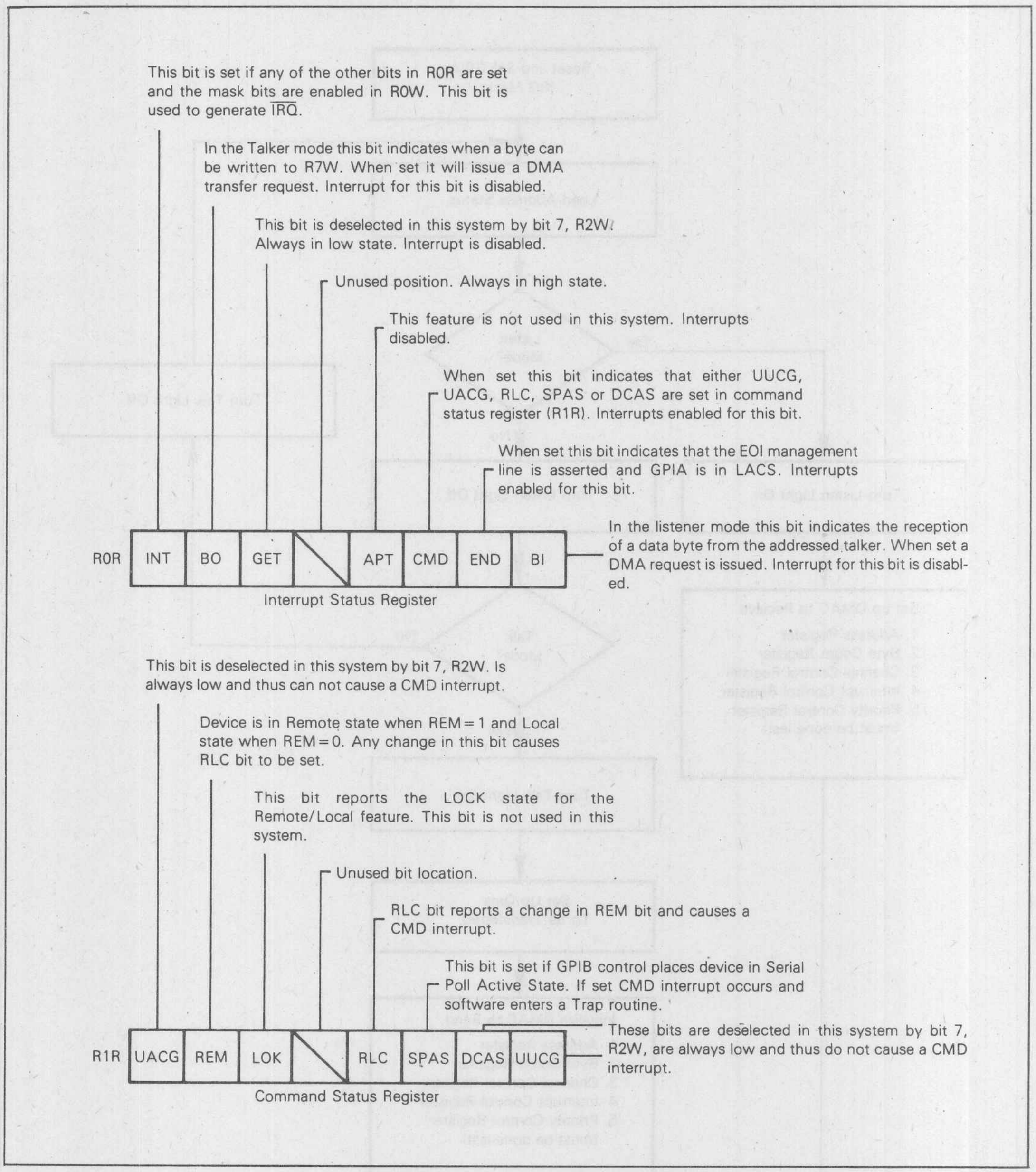
The GPIA is initialized next. The first step is for the MC6809 to read the address selected by the address switches and place this value in the GPIA address register (R4W). This is the value that the GPIB system controller will use to send addressed commands to this device. The next step is to remove the GPIA software reset by writing a $00 to the auxiliary command register (R3W). Until the software reset is removed (bit 7 of R3W written to zero), the only register in the GPIA that can be accessed is the address register. After R3W is written with $00, the MC6809 programs the address mode register (R2W) with a $80. This deselects certain status bits in the interrupt and command status registers from being set. The GPIA ignores any conditions on the GPIB that cause the GET status bit in the interrupt status register to be set and also any conditions that prevent the UACG, UUCG, and DCAS status bits in the command status register from being set. The interrupt mask register is then set up to enable interrupt capability on certain conditions. The interrupt mask register is programmed with $86. This allows interrupts to occur if the END status bit is set or the CMD status bit is set. A summary of interrupt and command status registers is given in Figure 4.

Since bit 7, R2W was set during initialization, the only bits in the command status reigster that can cause the CMD status bit to be set are remote local change (RLC) or serial poll active state (SPAS). The RLC status bit is used to determine the state of the remove local indicator light. The serial poll active state feature is not used in this system, and if this bit gets set and causes an interrupt, the system software goes to a trap routine and displays $E4 on the display.

MONITORING SEQUENCE — After the initialization sequence, the MC6809 software enters the monitor loop shown in Figure 5. The primary purpose of this routine is to set the indicator lights to indicate how the GPIA has been addressed (talk or listen) and initialize DMAC. The first procedure that is executed in the monitor loop is a reset and set of the GPIA interrupt mask register. Since the GPIA interrupt structure is edge sensitive to the setting of its status bits, the reset/set sequence of the interrupt mask register ensures that if a second interrupt bit gets set while a prior one is still set, this second interrupt is not missed. Now the address status register (R2R) of the GPIA is monitored. If the LACS bit is set, the listen status indicator is turned on and the DMAC initialized to transfer data from the GPIA to memory. If the TACS bit is set, the talker indicator light is turned on and the talker memory buffer is loaded with "dummy" values for the example test transfer. The DMAC is now initialized to transfer data from memory to GPIA.

After the direction of DMA transfer is established and the DMA controller initialized, the program enters the wait loop shown in Figure 6. The system enters this loop and waits for a DMA transfer request to be issued by the GPIA. The wait loop is not a necessary part of the system and in many applications can be replaced by the MC6809 performing some task. While in the wait loop, the software checks the address status register for any change in the addressed state. The following conditions result:

1. If there is not a change in address status of the GPIA, no action is taken and the program continually cycles through the wait loop.
2. If the GPIA is unaddressed (e.g., receiving an unlisten or untalk command), the program turns off the DMAC and goes to the monitor loop. This unaddressed condition is detected by monitoring the my address (ma) status bit in the GPIA.
3. If the addressed state changes from talker to listener or from listener to talker during a DMA block transfer, the wait loop branches to a trap routine and $E1 is



This bit is set if any of the other bits in R0R are set and the mask bits are enabled in R0W. This bit is used to generate $\overline{\text{IRQ}}$.

In the Talker mode this bit indicates when a byte can be written to R7W. When set it will issue a DMA transfer request. Interrupt for this bit is disabled.

This bit is deselected in this system by bit 7, R2W. Always in low state. Interrupt is disabled.

Unused position. Always in high state.

This feature is not used in this system. Interrupts disabled.

When set this bit indicates that either UUCG, UACG, RLC, SPAS or DCAS are set in command status register (R1R). Interrupts enabled for this bit.

When set this bit indicates that the EOI management line is asserted and GPIA is in LACS. Interrupts enabled for this bit.

In the listener mode this bit indicates the reception of a data byte from the addressed talker. When set a DMA request is issued. Interrupt for this bit is disabled.

This bit is deselected in this system by bit 7, R2W. Is always low and thus can not cause a CMD interrupt.

Device is in Remote state when REM = 1 and Local state when REM = 0. Any change in this bit causes RLC bit to be set.

This bit reports the LOCK state for the Remote/Local feature. This bit is not used in this system.

Unused bit location.

RLC bit reports a change in REM bit and causes a CMD interrupt.

This bit is set if GPIB control places device in Serial Poll Active State. If set CMD interrupt occurs and software enters a Trap routine.

These bits are deselected in this system by bit 7, R2W, are always low and thus do not cause a CMD interrupt.

Figure 4. GPIA Interrupt and Command Status Register

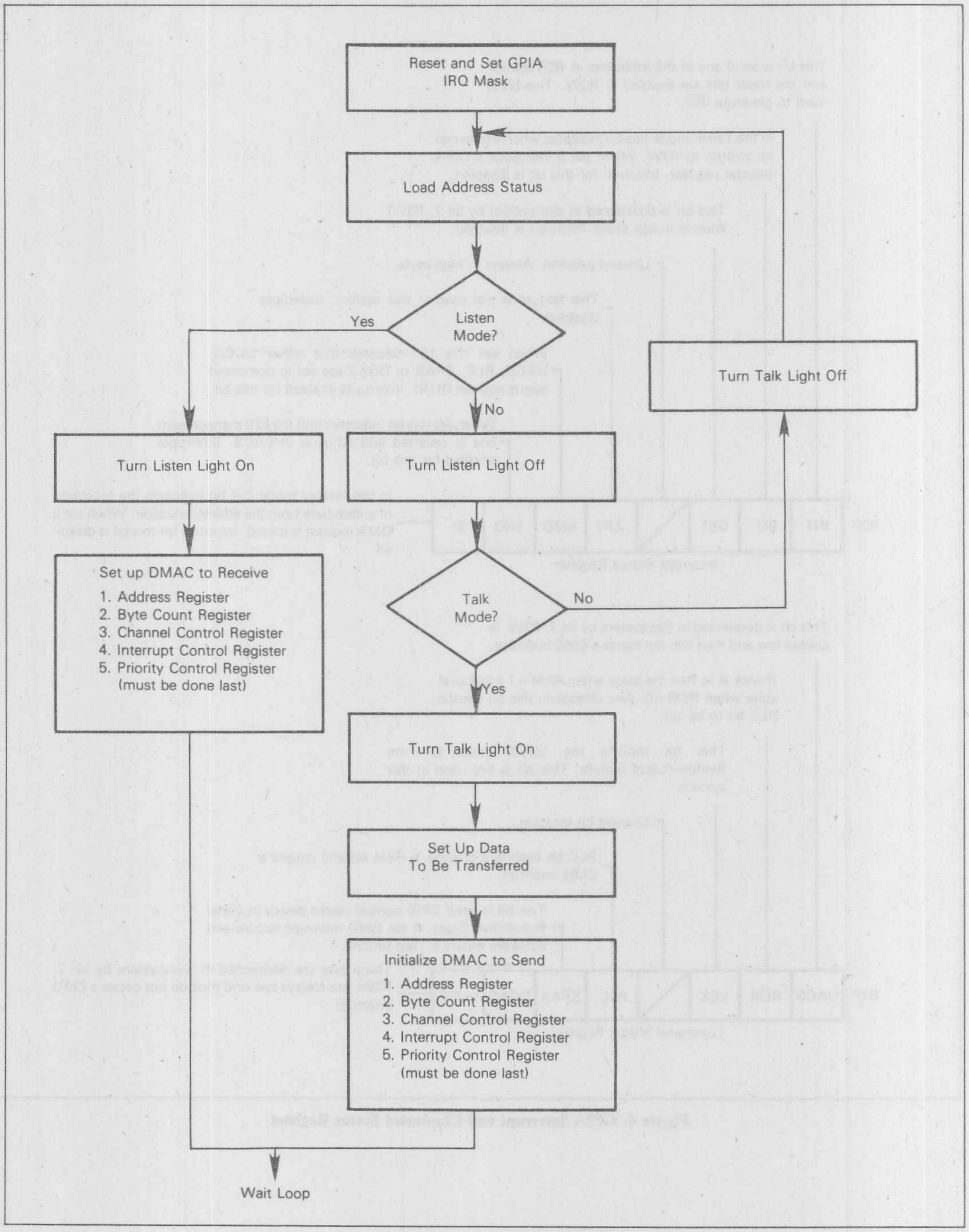


Figure 5. Monitor Loop Flow Chart

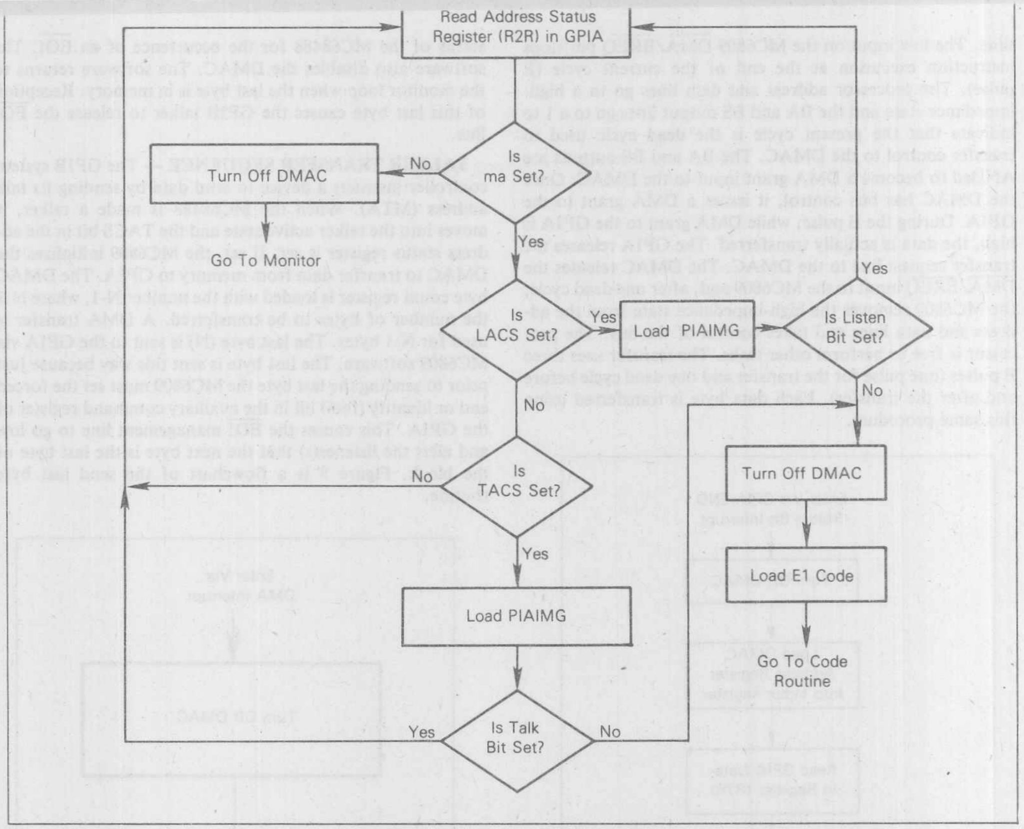


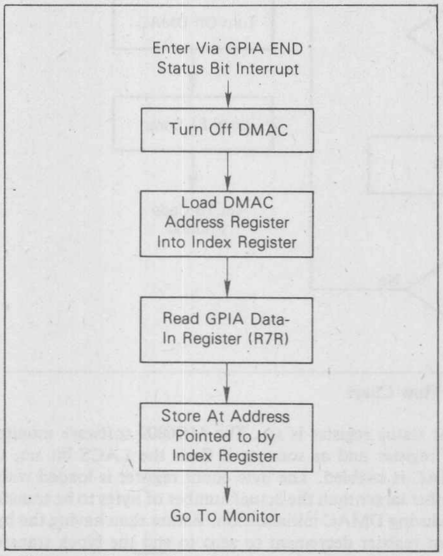
Figure 6. Wait Loop Flow Chart

displayed. Should this type of change occur, an error condition is trapped by the software and no additional block transfers are allowed to occur. The system program must be restarted.

Any change in the address status requires intervention by the GPIB system controller. This does not occur during most block transfers. It is possible, however, for the controller to take over the bus synchronously and untalk/unlisten the devices (condition 2 above). This might occur in response to a service request from some device in the system. Most likely, condition 3 will never occur (changing the talker/listener state immediately to the listener/talker state during a block transfer). If this does occur, the software enters a trap routine and $E1 is displayed.

LISTENER TRANSFER SEQUENCE — When the GPIA enters the listener active state, the LACS bit in the address status register is set. The MC6809 software monitors this register and as soon as it finds the LACS bit set, the DMAC is enabled. The byte count register is loaded with a number larger than the actual number of bytes to be transferred during DMAC initialization. Rather than having the byte count register decrement to zero to end the block transfer, the talker asserts the end or identify ($\overline{\text{EOI}}$) management line to end the transfer. Asserting $\overline{\text{EOI}}$ causes the GPIA to generate an interrupt as an end of block transfer indication and prepare to receive the final byte via software as shown in Figure 7.

After the DMAC is initialized, the software will enter the wait loop. When the GPIA receives a data byte it issues a transfer request to the DMAC. The DMAC, in turn, issues a transfer request ($\overline{\text{DRQT}}$) to the synchronization circuitry. It synchronizes this request with the Q clock from the MC6809 and issues a $\overline{\text{DMA/BREQ}}$ to the MC6809 during the Q high time. The low input on the MC6809 $\overline{\text{DMA}}/\overline{\text{BREQ}}$ pin stops instruction execution at the end of the current cycle (E pulse). The processor address and data lines go to a high-impedance state and the BA and BS output lines go to a 1 to indicate that the present cycle is the dead cycle used to transfer control to the DMAC. The BA and BS outputs are ANDed to become a DMA grant input to the DMAC. Once the DMAC has bus control, it issues a DMA grant to the GPIA. During the E pulse, while DMA grant to the GPIA is high, the data is actually transferred. The GPIA releases the transfer request line to the DMAC. The DMAC releases the $\overline{\text{DMA/BREQ}}$ input to the MC6809 and, after one dead cycle, the MC6809 removes the high-impedance state from the address and data lines and takes control of the bus. The processor is free to perform other tasks. The transfer uses three E pulses (one pulse for the transfer and one dead cycle before and after the transfer). Each data byte is transferred using this same procedure.

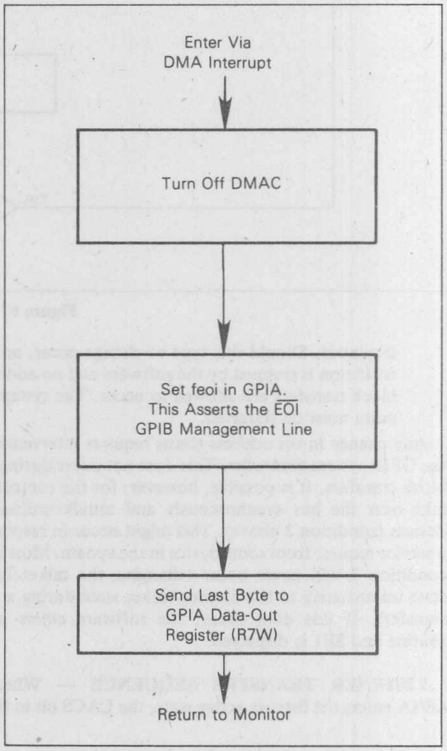


Enter Via GPIA END Status Bit Interrupt
Turn Off DMAC
Load DMAC Address Register Into Index Register
Read GPIA Data-In Register (R7R)
Store At Address Pointed to by Index Register
Go To Monitor

Figure 7. Receive Last Byte Routine Flow Chart

Prior to receiving the last byte of data, the GPIB talker drives the $\overline{\text{EOI}}$ line low. The $\overline{\text{EOI}}$ line is an input to the synchronization circuitry and, when asserted, prevents a DMA request from the DMAC to the MC6809 from being issued. This ensures that the MC6809 does not release control of the bus to the DMAC for the last byte transfer. In addition, the $\overline{\text{EOI}}$ line causes the END status bit in the GPIA to be set which in turn sends an interrupt to the MC6809. When the MC6809 software detects the END status bit set, it branches to a special routine, and the last byte is transferred to memory via processor software. The last byte is transferred by software since the processor must be used to read the status of the MC68488 for the occurrence of an $\overline{\text{EOI}}$. The software also disables the DMAC. The software returns to the monitor loop when the last byte is in memory. Reception of this last byte causes the GPIB talker to release the $\overline{\text{EOI}}$ line.

TALKER TRANSFER SEQUENCE — The GPIB system controller instructs a device to send data by sending its talk address (MTA). When the MC68488 is made a talker, it moves into the talker active state and the TACS bit in the address status register is set. If set, the MC6809 initializes the DMAC to transfer data from memory to GPIA. The DMAC byte count register is loaded with the number N-1, where N is the number of bytes to be transferred. A DMA transfer is used for N-1 bytes. The last byte (N) is sent to the GPIA via MC6809 software. The last byte is sent this way because just prior to sending the last byte the MC6809 must set the forced end or identify (feoi) bit in the auxiliary command register of the GPIA. This causes the $\overline{\text{EOI}}$ management line to go low and alert the listener(s) that the next byte is the last byte of the block. Figure 8 is a flowchart of the send last byte routine.



Enter Via DMA Interrupt
Turn Off DMAC
Set feoi in GPIA. This Asserts the EOI GPIB Management Line
Send Last Byte to GPIA Data-Out Register (R7W)
Return to Monitor

Figure 8. Send Last Byte Routine Flow Chart

As soon as the MC68488 enters the talker active state, a transfer request is issued indicating that the MC68488 is an active talker and the output buffer is empty. Each time the byte written to the GPIA output buffer is accepted by the listener(s) on the bus, another transfer request is issued. The transfer request handshake sequence between the MC68488, MC6844, and MC6809 is the same in the talker mode as it is for the listener mode.

INTERRUPT HANDLING — There are two sources of interrupts, the DMAC and the GPIA. When an interrupt occurs, the software checks to see if the DMAC caused the interrupt, as shown in Figure 9. The DMAC only generates an interrupt when the byte count register decrements to 0. Recall that, in the listener mode, the byte count register is programmed with a hex number larger than the number of bytes to be transferred. In the talker mode, the byte count register is programmed with N-1, where N is the number of bytes to be transferred. Therefore, the only time the DMAC can generate an interrupt in this system is when the GPIA is in the talker mode and is ready to transfer the last data byte from memory to GPIA.

If a DMAC interrupt occurs, the software checks the R/W bit in the DMAC channel control register. If this bit is not set, the DMAC is programmed to transfer data from GPIA to memory indicating that the GPIA is programmed to be a listener. In this instance, the byte count register was initialized with a number too small for the block size being transferred. The system enters a trap routine and $E2 is displayed. If the R/W bit is set, the system is in a talker mode and it is time to send the last byte of the block. The software enters the send last byte routine.

If a DMAC interrupt did not occur, then the GPIA is checked. If the GPIA INT status bit is not set, then one of two conditions has occurred. Either an extraneous interrupt was produced by another device such as a PIA or the GPIA has produced a "ghost interrupt." Ghost interrupts can occur in this system if the GPIB controller performs an illegal sequence of events or if the GPIA is placed in the serial poll active state (SPAS) and then removed from this state before the MC6809 interrupt software can check the GPIA status. Should any of these conditions occur, the software enters the trap routine and $E3 is displayed.

If the GPIA caused the interrupt, the software first checks the CMD bit in the interrupt status register. If the END bit is not set, the GPIA interrupt occurred from some other source in the interrupt status register. This implies that the interrupt mask register was incorrectly initialized and $E3 is displayed and the program trapped. If the END bit is set, then the last byte of the block is to follow. The program turns off the DMAC and then begins monitoring the BI bit in the interrupt register for the occurrence of the last byte.

If the GPIA caused the interrupt and the CMD bit was set, the software checks the command status register. All the bits in the command status register except the RLC and SPAS bits have been deselected in the initialization sequence. Therefore, the software only needs to check the RLC bit and, if it is not set, can assume that the interrupt was caused by SPAS. Since the SPAS feature of the GPIB is not used in this system, this occurrence causes the software to enter a trap routine. If the RLC bit was set, then the software checks the REM bit to see if the device is in local or remote and operates the remote/local indicator light accordingly.

DATA RATE — The data rate in this type of system is a function of the response of the device being communicated with. During the testing of this operation, a Hewlett Packard GPIB Emulator which has a TTL response rate was used (negligible when compared with the 6809/6844/68488 system). Because of this, the data rates for the system in this application are primarily a function of the 6809/6844/68488 system and any increase from combining the response rates for devices on both sides of the communications link can be considered negligible. The data rate differs slightly depending on whether the GPIA is a talker or a listener. This time difference is a result of the GPIA itself. The data rate as a listener is measured from the time the GPIA made the ready for data (RFD) line true for one transfer to the time RFD is made true for the next transfer. This time is 11 E-clock cycles which results in, for a one megabyte E clock, a transfer rate of 99K bytes per second.

The data rate as a talker is measured from the time the GPIA made $\overline{\text{DAV}}$ true for one transfer to the time $\overline{\text{DAV}}$ is made true for the next transfer. This time is eight E-clock cycles and results in a transfer rate of 125K bytes per second.

SYSTEM HARDWARE

The system hardware is designed to maximize the efficiency of DMA transfers and to provide an orderly processor bus control exchange between the processor and the DMAC. As mentioned earlier, there are two handshake sequences for each DMA transfer. The handshake between the peripheral device and the DMAC is to request and grant a DMA transfer. The handshake between the processor and DMAC is to exchange control of the processor bus. This control exchange must occur in an orderly fashion to eliminate bus contention. System clock cycles called "dead cycles" are provided before and after the actual DMA transfer cycle. It is during these dead cycles that the device in control of the processor bus releases control and goes into a high-impedance state and the other device assumes control by coming out of a high-impedance state. As shown in Figure 10, the timing is designed so that each exchange occurs in one cycle to maximize system efficiency and yet prevent both devices from trying to be in control of the processor bus at the same time. There is a time during each dead cycle where both the processor and DMAC are off the bus and the processor bus and control lines are in the high-impedance state. To prevent a spurious write or read during this time, a signal called DMAVMA is generated which disables the chip select of all peripheral devices.

To ensure that the entire post dead cycle has a DMAVMA, a signal called first quarter (FQ) is used to provide DMAVMA for the first quarter of every MC6809 E clock period. Since the first quarter is not used by peripheral devices, this operation does not pose any system problems.

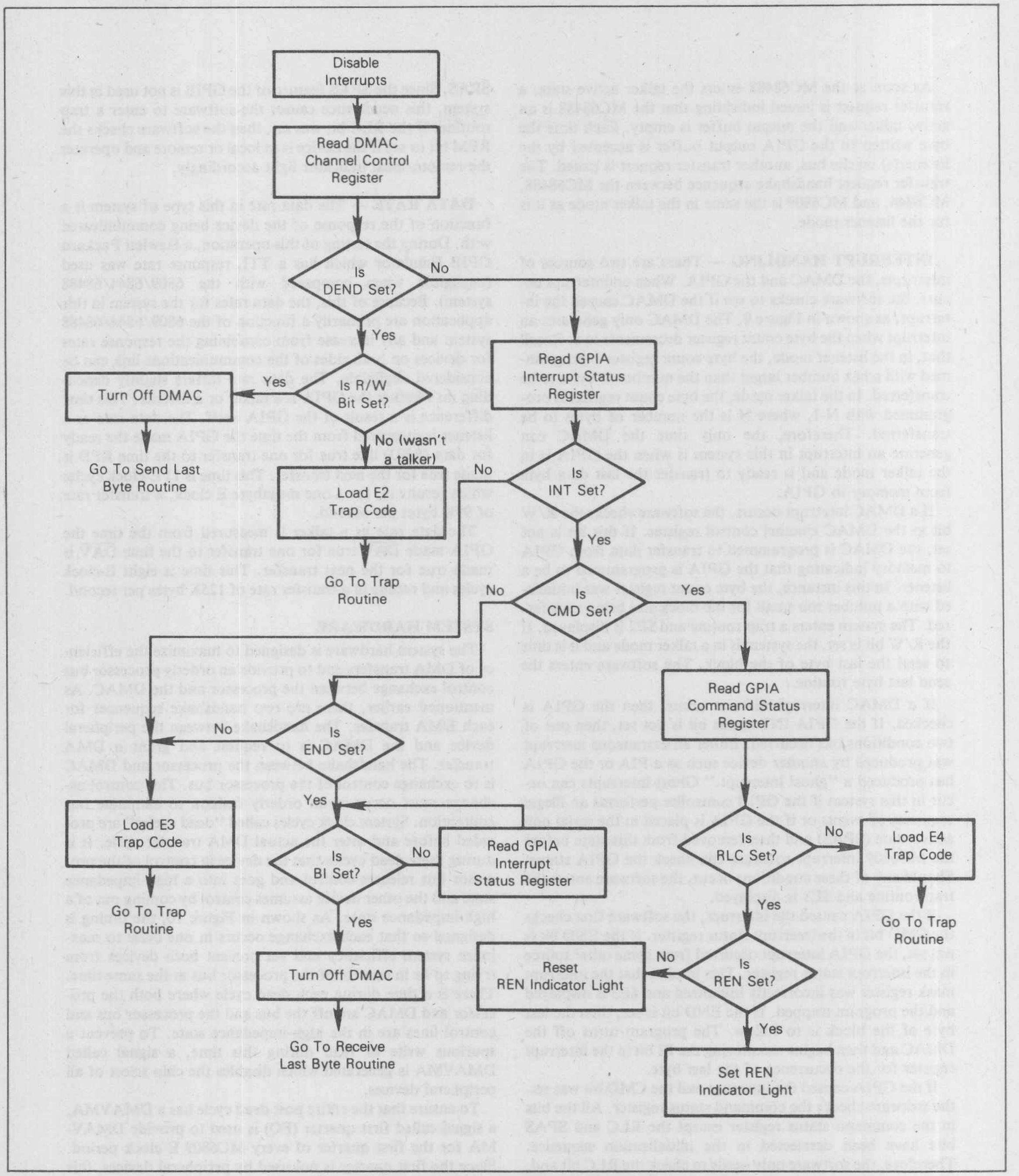


Figure 9. Interrupt Handling Routine Flow Chart

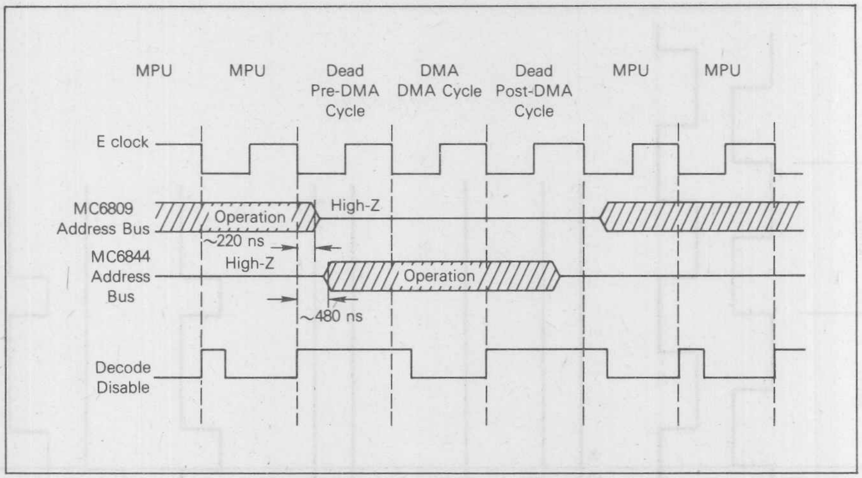


Figure 10. System DMA Cycle Timing

During data chaining operations on the DMAC, an extra post dead cycle occurs during the data chain process itself. The DMAVMA signal is not generated for this extra dead cycle. To prevent spurious read/write operations, the DMA request line from the MC68488 is input to the synchronization circuitry. This allows the MC6809 to take control of the processor during the extra data chaining dead cycle.

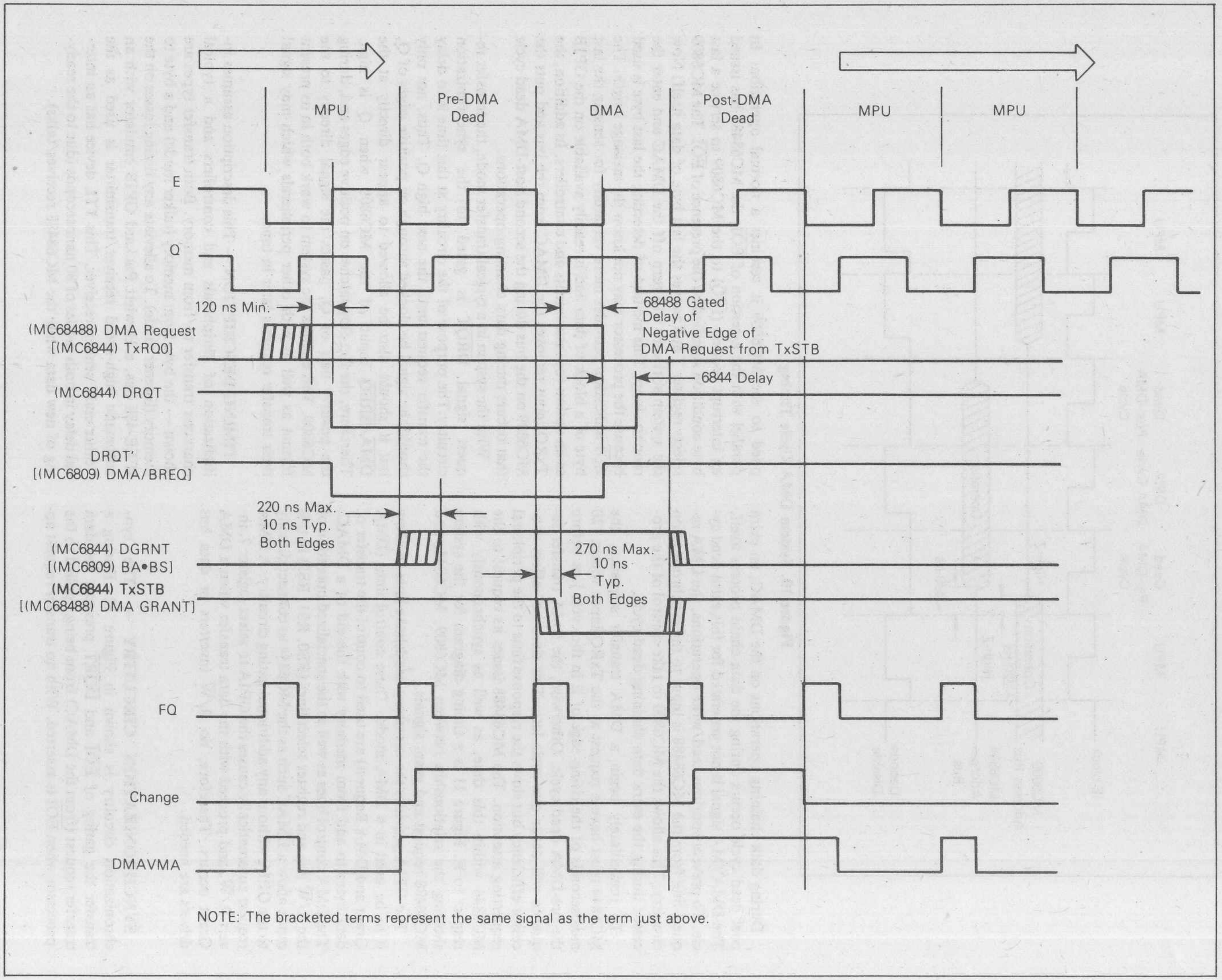
To immediately begin a DMA transfer sequence, the MC6844 must have a request at the TxRQ input within 120 nanoseconds of the rising edge of E in the cycle just before the pre-DMA dead cycle. Otherwise, the DMA transfer sequence will occur one cycle late. This does not affect processor efficiency but slows the response time to the peripheral requesting attention. The MC68488 issues its request to the MC6844 within this time, as well as synchronously with respect to E. Figure 11 is a timing diagram for the system showing the relationship between MC6809, MC6844, and MC68488 request and grant signals.

The GPIA provides the necessary handshake lines to allow it to be used in a DMA mode. These control lines (DMA Grant and DMA Request) are used to control the transfer of data bytes to and from memory with the aid of a DMAC. The DMA control lines as well as the specialized operation of the R/$\overline{\text{W}}$ line and register select lines (RS0, RS1, RS2) in this mode allow a DMAC such as the MC6844 to connect directly to the GPIA without any additional gating circuitry. A DMA request automatically causes the GPIA to select register 7, invert R/$\overline{\text{W}}$, and proceed with the data transfer when a DMA Grant occurs. Therefore, no R/$\overline{\text{W}}$ inverters or data bus drivers are needed.

SYNCHRONIZATION CIRCUITRY — The synchronization circuitry is shown in Figure 12. During a transfer the gating of $\overline{\text{EOI}}$ and $\overline{\text{DQRT}}$ prevents the data transfer request (from the DMAC) from being applied to the processor when $\overline{\text{EOI}}$ is asserted. With no transfer request applied to the MC6809 it resumes a normal operation. In parallel with the assertion of $\overline{\text{EOI}}$, the MC68488 has issued an interrupt request ($\overline{\text{IRQ}}$) to the MC6809 to service a last byte condition signified by the presence of $\overline{\text{EOI}}$. The MC6809 selects register 7 and moves the last byte of data itself. Now the system software will turn off the DMAC and enter the monitor loop. This method of detecting the last byte is used because the processor may not know the message length. The $\overline{\text{EOI}}$ indication provides more versatility for sensing the last byte of a block of data and is readily available on the GPIB as an option for instruments and controllers. In addition, the TxRQ input removes the DMAC from the bus and puts the MC6809 on the bus during the second post-DMA dead cycle that occurs during data chaining operations.

With the system in a typical transfer mode, the transfer request signal $\overline{\text{DRQT}}$ is gated to the synchronization circuitry.The purpose of the circuitry at this time is to delay the transfer request until the next high Q. Thus, not only should the signal be clocked through on positive edges of Q, but it should also be allowed to appear directly at the $\overline{\text{DMA/BREQ}}$ input of the MC6809 when Q is high. Therefore, the flip-flop latches on positive edges and, during the positive half of Q, passes the signal directly to the MC6809. This enables the system to work both in its present format as well as with other peripherals which may signal their transfer requests later in time.

TIMING DESCRIPTION — This description assumes initialization of peripherals and controllers and a typical character transfer to/from memory. Both transfer types are shown — the byte from memory (talker mode) and a byte to memory (listener mode). To alleviate any timing losses on the IEEE-488 bus, a Hewlett Packard GPIB emulator with an automatic high-speed receiver/transmitter is used as the "other end" sender/receiver. This TTL device has an internal delay in both modes of 80 nanoseconds (due to the readying of new data while the MC68488 receives/talks).


MPU
Pre-DMA Dead
DMA
Post-DMA Dead
MPU
MPU
E
Q
120 ns Min.
68488 Gated Delay of Negative Edge of DMA Request from TxSTB
(MC68488) DMA Request [(MC6844) TxRQ0]
6844 Delay
(MC6844) DRQT
DRQT' [(MC6809) DMA/BREQ]
220 ns Max. 10 ns Typ. Both Edges
(MC6844) DGRNT [(MC6809) BA•BS]
270 ns Max. 10 ns Typ. Both Edges
(MC6844) TxSTB [(MC68488) DMA GRANT]
FQ
Change
DMAVMA

NOTE: The bracketed terms represent the same signal as the term just above.

Figure 11. System Timing Diagram

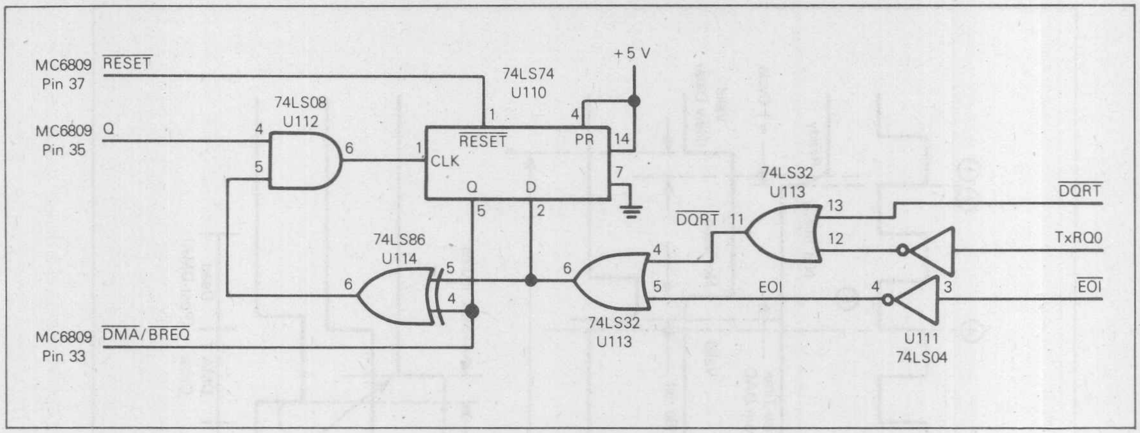


Figure 12. Synchronization Circuitry

LISTENER — Refer to Figure 13. With the GPIA in the listener mode, the ready for data (RFD) handshake line goes high ① as the GPIA is ready for another byte. One emulator box delay (80 ns) later, data is valid on the bus ②. Approximately two clock cycles later, the GPIA has taken the byte and RFD goes low.

Three and a half clock cycles later, the GPIA issues a request to the system using the DMA request line ③. Approximately 300 nanoseconds later, the MC6844 issues $\overline{\text{DRQT}}$. The synchronization circuitry passes the request instantly since Q is high, and the MC6809 receives a $\overline{\text{DMA}}/\overline{\text{BREQ}}$ input. At the beginning of the dead cycle (if the 125 nanosecond lead time on $\overline{\text{DMA}}/\overline{\text{BREQ}}$ was observed), the BA and BS lines both go high to indicate that the bus is in a high-impedance state and is available. With the BA and BS signals ANDed together and sent to the DGRNT input of the MC6844, the DMAC readies the bus for transfer by outputting: the address for the memory store, a write condition on the R/$\overline{\text{W}}$ line, and in the next cycle, a TxSTB to the DMA grant line of the GPIA. As soon as DMA grant is received, the TxRQ is removed from the MC6844 by the GPIA and, 300 nanoseconds later, $\overline{\text{DRQT}}$ is also brought low. By the falling edge of E on the DMA cycle, the GPIA has automatically selected register 7. It has inverted R/$\overline{\text{W}}$ (so that the "write" of the received data to memory means "read" from the GPIA), and on the falling edge of E, the data is latched into memory at the address that the MC6844 has already supplied. Now that a byte has been taken from register 7, the GPIA prepares to receives a new byte from the GPIB. In the post DMA dead cycle, a data accepted (DAC) signal is put on the bus ④. After one (80 ns) emulator box delay the GPIA gets a "Not Valid" indication on the $\overline{\text{DAV}}$ line ⑤. From that time to a new RFD signal ⑥, the internal delay time in the GPIA is required to reset all latches and begin again.

TALKER — The processor bus timing when the GPIA is in the talker mode is the same as for the listener mode. The rate that transfer requests are generated by the GPIA is directly related to how quickly the listener can accept the data. Figure 14 shows the system timing when the GPIA is programmed as a talker.

As soon as the data from the last transfer is accepted at the emulator and a DAC is received ①, the GPIA sends out its DMA request for a new byte from the MC6844. Three cycles later when the DMA occurs ③, the GPIA begins to move that data to the GPIB. One and one-half cycles later ④, the GPIA issues $\overline{\text{DAV}}$, and the emulator issues DAC 80 nanoseconds later. After a response time to "Data Not Valid" (approximately 2 cycles), the emulator is ready for a new byte from the GPIA ⑥.

SYSTEM SOFTWARE

The software shown in this application is not intended to be a general purpose application program. It is an example program showing how the MC68488 can be used with the MC6809 in a DMA system. The memory map for this system is shown in Figure 15.

TRAP ROUTINE — The software has a trap routine which displays a code on the system display. Once the system enters the trap routine, it remains in this routine. If an EXORciser system is used, then the Restart key has to be used to restart the program at the monitor loop location ($D079). A list of the display codes are given below.

Code	Description
E1	The LACS/TACS bit in the GPIA is set, but the listener/talker software flag bit (PIAIMG) is not set. This condition could occur uring a DMA block transfer if the GPIA system controller readdresses

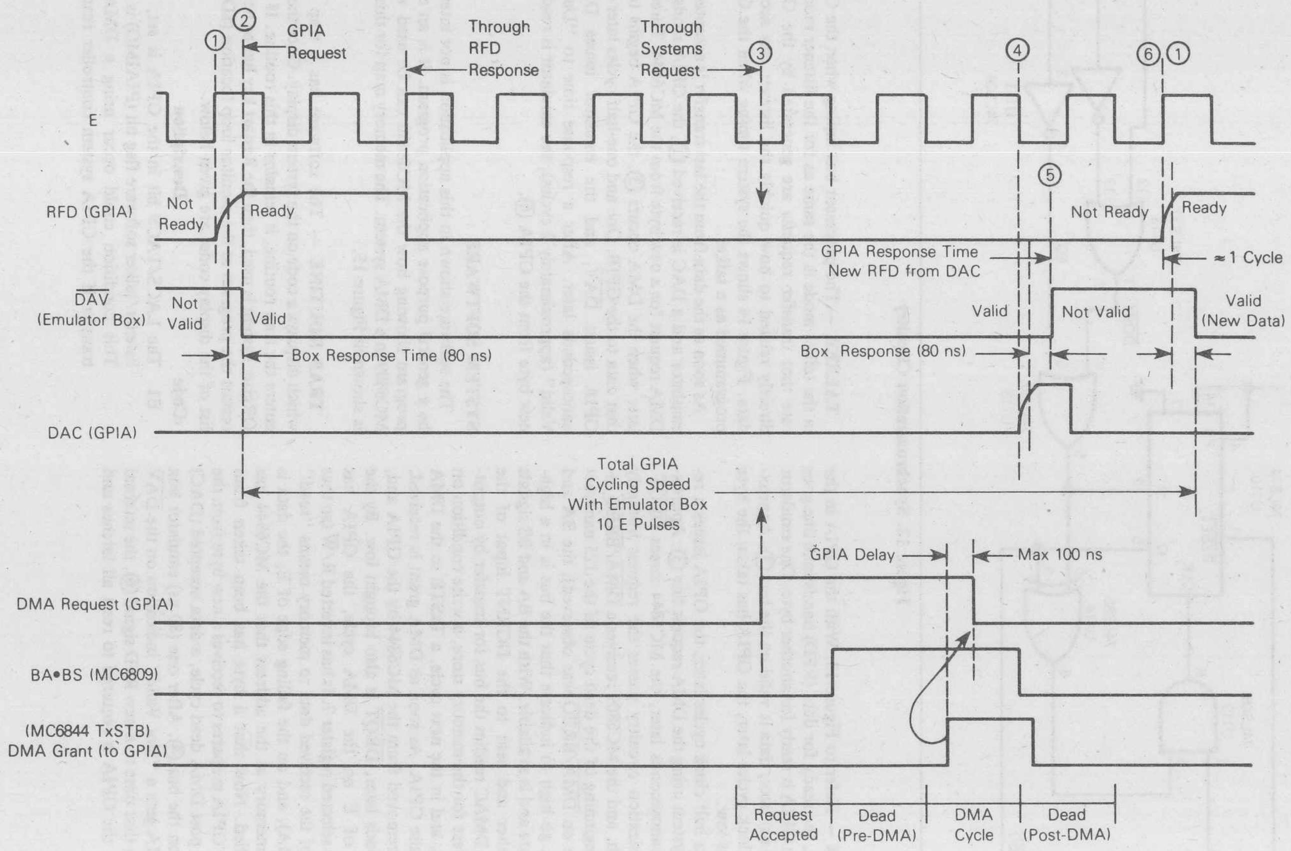


Figure 13. Listener Mode Timing Diagram

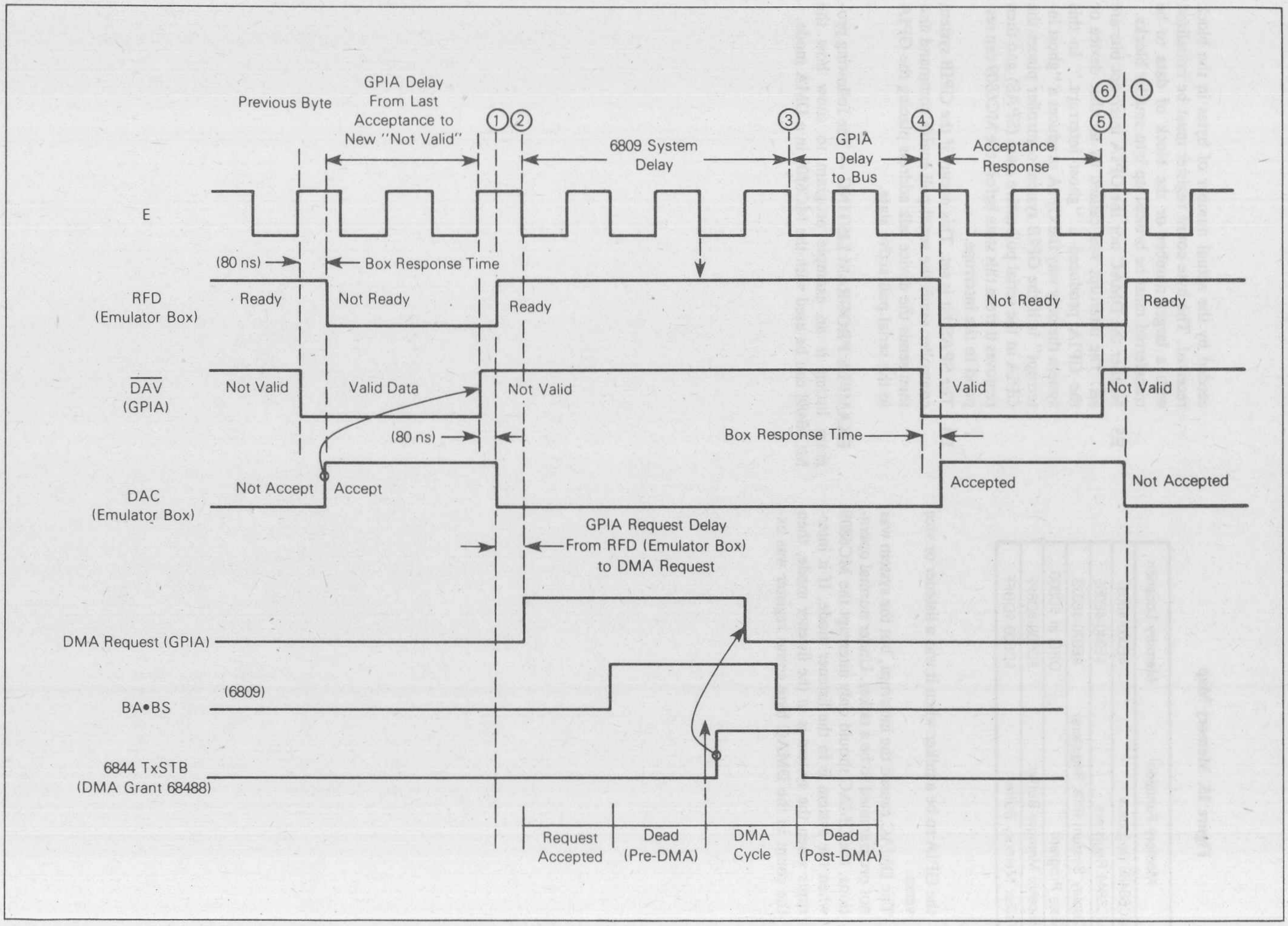


Figure 14. Talker Mode Timing Diagram

Figure 15. Memory Map

Memory Function	Memory Location
MC68488 Registers	$E060-$E067
MC6844 Registers	$E040-$E056
Display System (PIA Registers)	$E070-$E073
Main Program	ORG at $D000
Receive Memory Buffer	$D800-$D8FF
Talker Memory Buffer	$D800-$DBFF

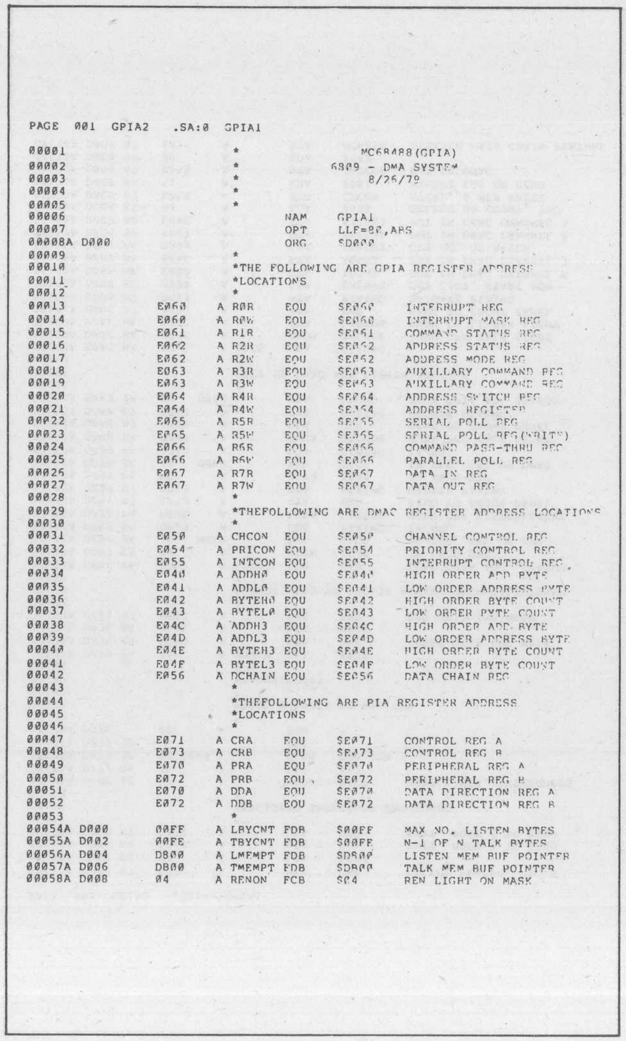
the GPIA to be a talker when it was a listener or vice versa.

E2 The DMAC caused the interrupt, but the system was not programmed to be a talker. Under normal operations, the DMAC should only interrupt the MC6809 when the system is in the listener mode. If it interrupts when the system is in the listener mode, then the count in the DMAC byte count register was exceeded by the actual number of bytes in the block received. The byte count register must be initialized with a larger number or the block of data to be transferred must be broken up into smaller blocks.

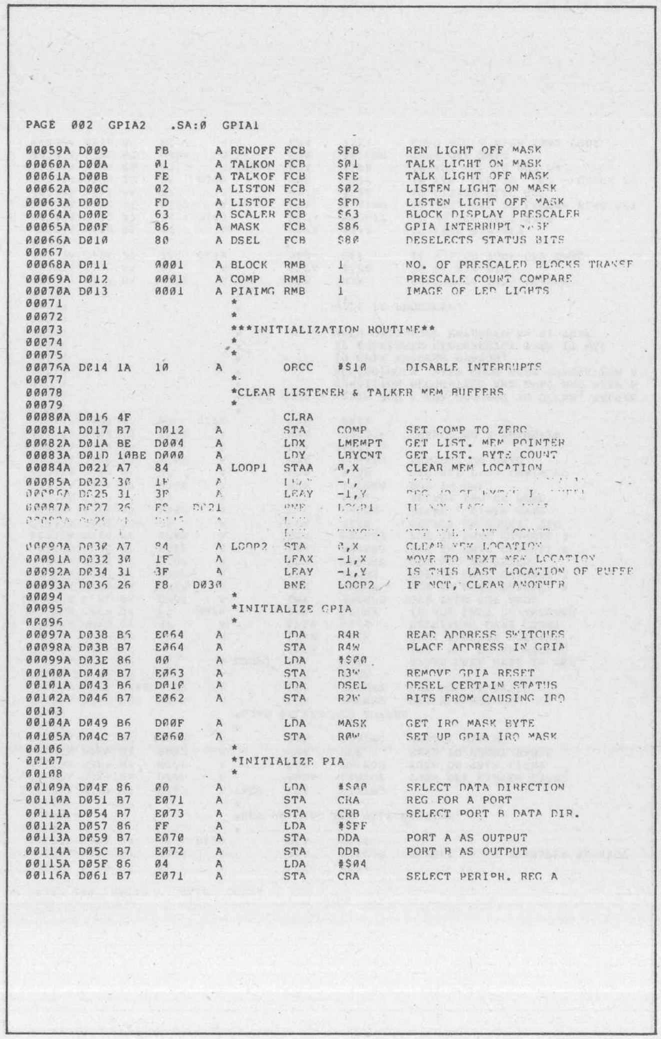
E3 Neither the DMAC nor the GPIA interrupt bits are set. The interrupt was caused by another device or the GPIA produced a "ghost interrupt." In this system the only way the GPIA produces a "ghost interrupt" is if the GPIB system controller places the GPIA in the serial poll active state (SPAS) and then removes it from this state before the MC6809 can respond to the interrupt.

E4 The SPAS bit is set. This occurs if the GPIB system controller sends the serial poll enable command and then sends the device talk address placing the GPIA in the serial poll active state.

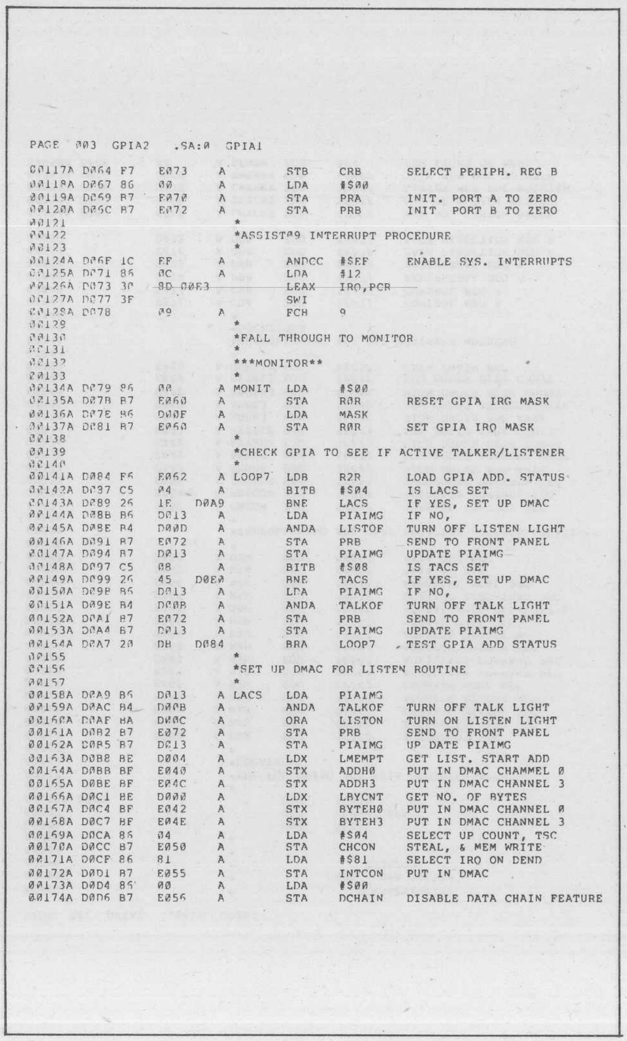
EXAMPLE PROGRAM LISTING — The following program listing is an example program to show how the MC68488 can be used with the MC6809 in a DMA mode.



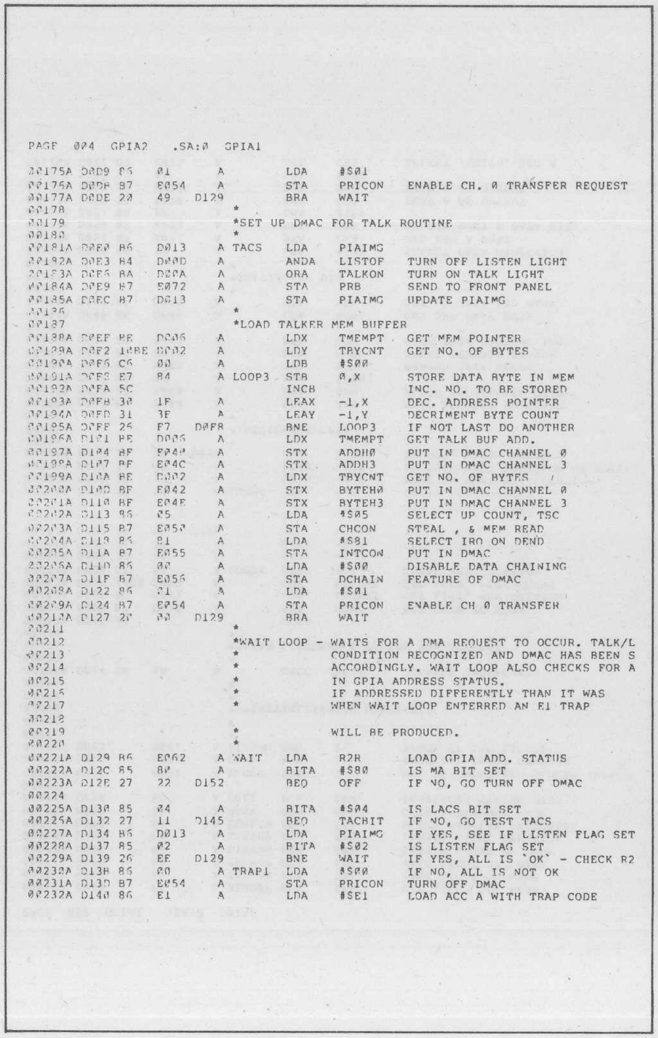
```
PAGE  ØØ1  GPIA2   .SA:Ø  GPIA1

ØØØØ1                    *              MC68488(GPIA)
ØØØØ2                    *            68Ø9 - DMA SYSTEM
ØØØØ3                    *               8/25/79
ØØØØ4                    *
ØØØØ5                    *
ØØØØ6                           NAM    GPIA1
ØØØØ7                           OPT    LLF=8Ø,ABS
ØØØØ8A DØØØ                     ORG    $DØØØ
ØØØØ9                    *
ØØØ1Ø                    *THE FOLLOWING ARE GPIA REGISTER ADDRESS
ØØØ11                    *LOCATIONS
ØØØ12                    *
ØØØ13          EØ6Ø    A RØR    EQU    $EØ6Ø    INTERRUPT REG
ØØØ14          EØ6Ø    A RØW    EQU    $EØ6Ø    INTERRUPT MASK REG
ØØØ15          EØ61    A R1R    EQU    $EØ61    COMMAND STATUS REG
ØØØ16          EØ62    A R2R    EQU    $EØ62    ADDRESS STATUS REG
ØØØ17          EØ62    A R2W    EQU    $EØ62    ADDRESS MODE REG
ØØØ18          EØ63    A R3R    EQU    $EØ63    AUXILLARY COMMAND REG
ØØØ19          EØ63    A R3W    EQU    $EØ63    AUXILLARY COMMAND REG
ØØØ2Ø          EØ64    A R4R    EQU    $EØ64    ADDRESS SWITCH REG
ØØØ21          EØ64    A R4W    EQU    $EØ64    ADDRESS REGISTER
ØØØ22          EØ65    A R5R    EQU    $EØ65    SERIAL POLL REG
ØØØ23          EØ65    A R5W    EQU    $E365    SERIAL POLL REG(WRITE)
ØØØ24          EØ66    A R6R    EQU    $EØ66    COMMAND PASS-THRU REG
ØØØ25          EØ66    A R6W    EQU    $EØ66    PARALLEL POLL REG
ØØØ26          EØ67    A R7R    EQU    $EØ67    DATA IN REG
ØØØ27          EØ67    A R7W    EQU    $EØ67    DATA OUT REG
ØØØ28                    *
ØØØ29                    *THEFOLLOWING ARE DMAC REGISTER ADDRESS LOCATIONS
ØØØ3Ø                    *
ØØØ31          EØ5Ø    A CHCON  EQU    $EØ5Ø    CHANNEL CONTROL REG
ØØØ32          EØ54    A PRICON EQU    $EØ54    PRIORITY CONTROL REG
ØØØ33          EØ55    A INTCON EQU    $EØ55    INTERRUPT CONTROL REG
ØØØ34          EØ4Ø    A ADDHØ  EQU    $EØ4Ø    HIGH ORDER ADD BYTE
ØØØ35          EØ41    A ADDLØ  EQU    $EØ41    LOW ORDER ADDRESS BYTE
ØØØ36          EØ42    A BYTEHØ EQU    $EØ42    HIGH ORDER BYTE COUNT
ØØØ37          EØ43    A BYTELØ EQU    $EØ43    LOW ORDER BYTE COUNT
ØØØ38          EØ4C    A ADDH3  EQU    $EØ4C    HIGH ORDER ADD BYTE
ØØØ39          EØ4D    A ADDL3  EQU    $EØ4D    LOW ORDER ADDRESS BYTE
ØØØ4Ø          EØ4E    A BYTEH3 EQU    $EØ4E    HIGH ORDER BYTE COUNT
ØØØ41          EØ4F    A BYTEL3 EQU    $EØ4F    LOW ORDER BYTE COUNT
ØØØ42          EØ56    A DCHAIN EQU    $EØ56    DATA CHAIN REG
ØØØ43                    *
ØØØ44                    *THEFOLLOWING ARE PIA REGISTER ADDRESS
ØØØ45                    *LOCATIONS
ØØØ46                    *
ØØØ47          EØ71    A CRA    EQU    $EØ71    CONTROL REG A
ØØØ48          EØ73    A CRB    EQU    $EØ73    CONTROL REG B
ØØØ49          EØ7Ø    A PRA    EQU    $EØ7Ø    PERIPHERAL REG A
ØØØ5Ø          EØ72    A PRB    EQU    $EØ72    PERIPHERAL REG B
ØØØ51          EØ7Ø    A DDA    EQU    $EØ7Ø    DATA DIRECTION REG A
ØØØ52          EØ72    A DDB    EQU    $EØ72    DATA DIRECTION REG B
ØØØ53                    *
ØØØ54A DØØØ    ØØFF    A LBYCNT FDB    $ØØFF    MAX NO. LISTEN BYTES
ØØØ55A DØØ2    ØØFE    A TBYCNT FDB    $ØØFF    N-1 OF N TALK BYTES
ØØØ56A DØØ4    D8ØØ    A LMEMPT FDB    $D8ØØ    LISTEN MEM BUF POINTER
ØØØ57A DØØ6    DBØØ    A TMEMPT FDB    $DBØØ    TALK MEM BUF POINTER
ØØØ58A DØØ8    Ø4      A RENON  FCB    $Ø4      REN LIGHT ON MASK
```



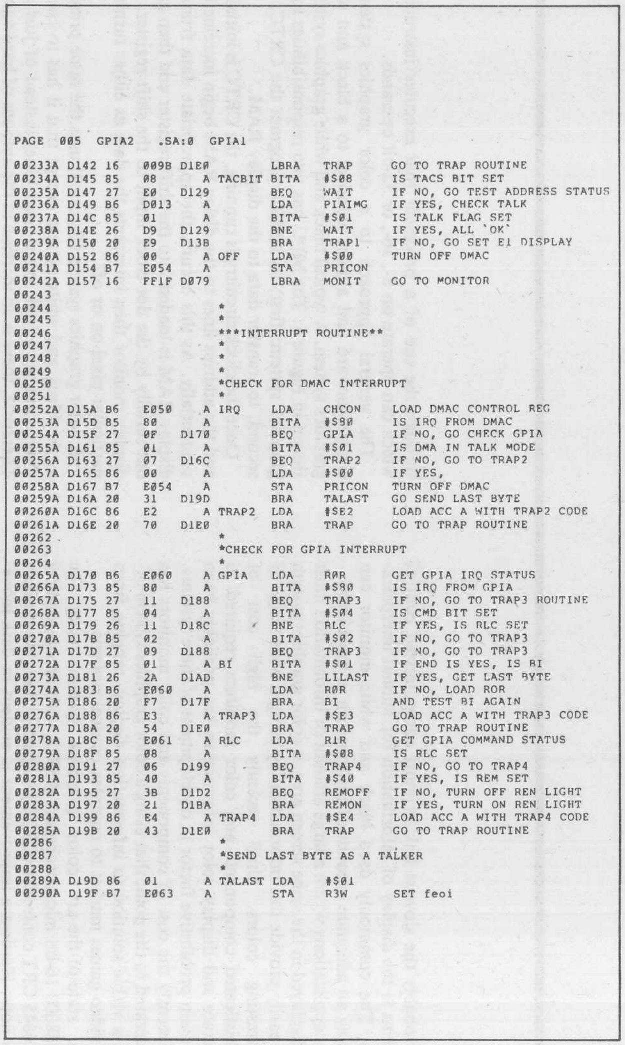
```
PAGE  ØØ2  GPIA2   .SA:Ø  GPIA1

ØØØ59A DØØ9      FB        A RENOFF FCB    $FB      REN LIGHT OFF MASK
ØØØ6ØA DØØA      Ø1        A TALKON FCB    $Ø1      TALK LIGHT ON MASK
ØØØ61A DØØB      FE        A TALKOF FCB    $FE      TALK LIGHT OFF MASK
ØØØ62A DØØC      Ø2        A LISTON FCB    $Ø2      LISTEN LIGHT ON MASK
ØØØ63A DØØD      FD        A LISTOF FCB    $FD      LISTEN LIGHT OFF MASK
ØØØ64A DØØE      63        A SCALER FCB    $63      BLOCK DISPLAY PRESCALER
ØØØ65A DØØF      86        A MASK   FCB    $86      GPIA INTERRUPT MASK
ØØØ66A DØ1Ø      8Ø        A DSEL   FCB    $8Ø      DESELECTS STATUS BITS
ØØØ67
ØØØ68A DØ11      ØØØ1      A BLOCK  RMB    1        NO. OF PRESCALED BLOCKS TRANSF
ØØØ69A DØ12      ØØØ1      A COMP   RMB    1        PRESCALE COUNT COMPARE
ØØØ7ØA DØ13      ØØØ1      A PIAIMG RMB    1        IMAGE OF LED LIGHTS
ØØØ71                        *
ØØØ72                        *
ØØØ73                        ***INITIALIZATION ROUTINE**
ØØØ74                        *
ØØØ75                        *
ØØØ76A DØ14 1A   1Ø        A        ORCC   #$1Ø     DISABLE INTERRUPTS
ØØØ77                        *
ØØØ78                        *CLEAR LISTENER & TALKER MEM BUFFERS
ØØØ79                        *
ØØØ8ØA DØ16 4F                      CLRA
ØØØ81A DØ17 B7   DØ12      A        STA    COMP     SET COMP TO ZERO
ØØØ82A DØ1A BE   DØØ4      A        LDX    LMEMPT   GET LIST. MEM POINTER
ØØØ83A DØ1D 1ØBE DØØØ      A        LDY    LBYCNT   GET LIST. BYTE COUNT
ØØØ84A DØ21 A7   84        A LOOP1  STAA   Ø,X      CLEAR MEM LOCATION
ØØØ85A DØ23 3Ø   1F        A        LEAX   -1,      [illegible]
ØØØ86A DØ25 31   3F        A        LEAY   -1,Y     DEC NO OF BYTES [illegible]
ØØØ87A DØ27 26   F8   DØ21          BNE    LOOP1    IF NOT LAST [illegible]
[illegible]
[illegible]
ØØØ9ØA DØ3Ø A7   84        A LOOP2  STA    Ø,X      CLEAR MEM LOCATION
ØØØ91A DØ32 3Ø   1F        A        LEAX   -1,X     MOVE TO NEXT MEM LOCATION
ØØØ92A DØ34 31   3F        A        LEAY   -1,Y     IS THIS LAST LOCATION OF BUFFE
ØØØ93A DØ36 26   F8   DØ3Ø          BNE    LOOP2    IF NOT, CLEAR ANOTHER
ØØØ94                        *
ØØØ95                        *INITIALIZE GPIA
ØØØ96                        *
ØØØ97A DØ38 B6   EØ64      A        LDA    R4R      READ ADDRESS SWITCHES
ØØØ98A DØ3B B7   EØ64      A        STA    R4W      PLACE ADDRESS IN GPIA
ØØØ99A DØ3E 86   ØØ        A        LDA    #$ØØ
ØØ1ØØA DØ4Ø B7   EØ63      A        STA    R3W      REMOVE GPIA RESET
ØØ1Ø1A DØ43 B6   DØ1Ø      A        LDA    DSEL     DESEL CERTAIN STATUS
ØØ1Ø2A DØ46 B7   EØ62      A        STA    R2W      BITS FROM CAUSING IRQ
ØØ1Ø3
ØØ1Ø4A DØ49 B6   DØØF      A        LDA    MASK     GET IRQ MASK BYTE
ØØ1Ø5A DØ4C B7   EØ6Ø      A        STA    RØW      SET UP GPIA IRQ MASK
ØØ1Ø6                        *
ØØ1Ø7                        *INITIALIZE PIA
ØØ1Ø8                        *
ØØ1Ø9A DØ4F 86   ØØ        A        LDA    #$ØØ     SELECT DATA DIRECTION
ØØ11ØA DØ51 B7   EØ71      A        STA    CRA      REG FOR A PORT
ØØ111A DØ54 B7   EØ73      A        STA    CRB      SELECT PORT B DATA DIR.
ØØ112A DØ57 86   FF        A        LDA    #$FF
ØØ113A DØ59 B7   EØ7Ø      A        STA    DDA      PORT A AS OUTPUT
ØØ114A DØ5C B7   EØ72      A        STA    DDB      PORT B AS OUTPUT
ØØ115A DØ5F 86   Ø4        A        LDA    #$Ø4
ØØ116A DØ61 B7   EØ71      A        STA    CRA      SELECT PERIPH. REG A
```



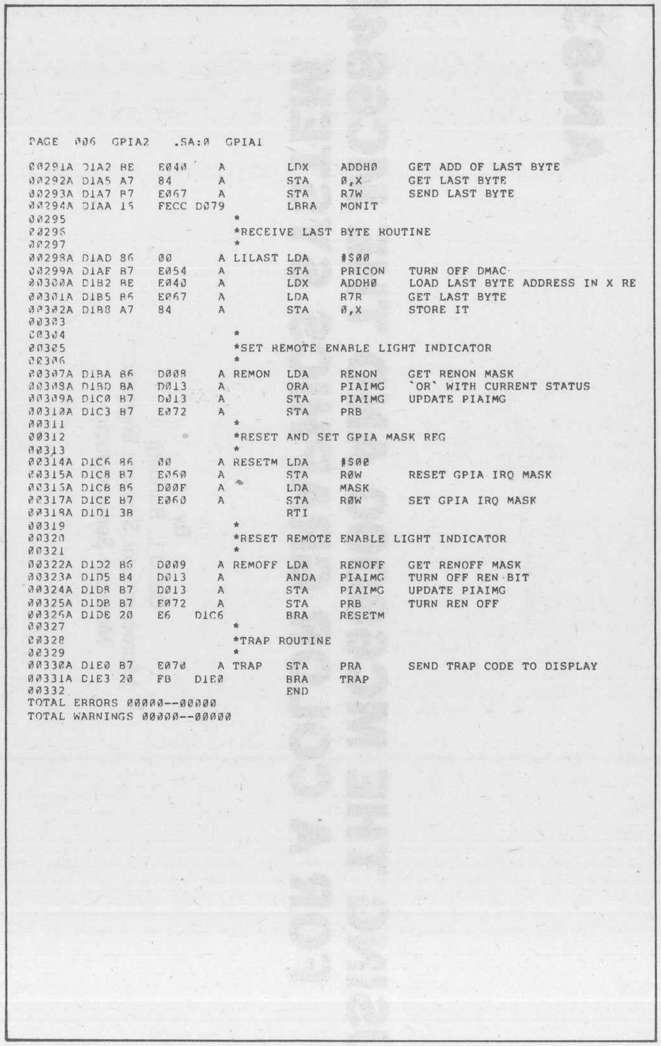
```
PAGE  ØØ3  GPIA2   .SA:Ø  GPIA1

ØØ117A DØ64 F7   EØ73     A          STB    CRB      SELECT PERIPH. REG B
ØØ118A DØ67 86   ØØ       A          LDA    #$ØØ
ØØ119A DØ69 B7   EØ7Ø     A          STA    PRA      INIT. PORT A TO ZERO
ØØ12ØA DØ6C B7   EØ72     A          STA    PRB      INIT  PORT B TO ZERO
ØØ121                       *
ØØ122                       *ASSISTØ9 INTERRUPT PROCEDURE
ØØ123                       *
ØØ124A DØ6F 1C   EF       A          ANDCC  #$EF     ENABLE SYS. INTERRUPTS
ØØ125A DØ71 86   ØC       A          LDA    #12
ØØ126A DØ73 3Ø   ~~8D ØØE3           LEAX   IRQ,PCR~~
ØØ127A DØ77 3F                       SWI
ØØ128A DØ78      Ø9       A          FCB    9
ØØ129                       *
ØØ13Ø                       *FALL THROUGH TO MONITOR
ØØ131                       *
ØØ132                       ***MONITOR**
ØØ133                       *
ØØ134A DØ79 86   ØØ       A MONIT    LDA    #$ØØ
ØØ135A DØ7B B7   EØ6Ø     A          STA    RØR      RESET GPIA IRG MASK
ØØ136A DØ7E 86   DØØF     A          LDA    MASK
ØØ137A DØ81 B7   EØ6Ø     A          STA    RØR      SET GPIA IRQ MASK
ØØ138                       *
ØØ139                       *CHECK GPIA TO SEE IF ACTIVE TALKER/LISTENER
ØØ14Ø                       *
ØØ141A DØ84 F6   EØ62     A LOOP7    LDB    R2R      LOAD GPIA ADD. STATUS
ØØ142A DØ87 C5   Ø4       A          BITB   #$Ø4     IS LACS SET
ØØ143A DØ89 26   1E   DØA9           BNE    LACS     IF YES, SET UP DMAC
ØØ144A DØ8B B6   DØ13     A          LDA    PIAIMG   IF NO,
ØØ145A DØ8E B4   DØØD     A          ANDA   LISTOF   TURN OFF LISTEN LIGHT
ØØ146A DØ91 B7   EØ72     A          STA    PRB      SEND TO FRONT PANEL
ØØ147A DØ94 B7   DØ13     A          STA    PIAIMG   UPDATE PIAIMG
ØØ148A DØ97 C5   Ø8       A          BITB   #$Ø8     IS TACS SET
ØØ149A DØ99 26   45   DØEØ           BNE    TACS     IF YES, SET UP DMAC
ØØ15ØA DØ9B B6   DØ13     A          LDA    PIAIMG   IF NO,
ØØ151A DØ9E B4   DØØB     A          ANDA   TALKOF   TURN OFF TALK LIGHT
ØØ152A DØA1 B7   EØ72     A          STA    PRB      SEND TO FRONT PANEL
ØØ153A DØA4 B7   DØ13     A          STA    PIAIMG   UPDATE PIAIMG
ØØ154A DØA7 2Ø   DB   DØ84           BRA    LOOP7  , TEST GPIA ADD STATUS
ØØ155                       *
ØØ156                       *SET UP DMAC FOR LISTEN ROUTINE
ØØ157                       *
ØØ158A DØA9 B6   DØ13     A LACS     LDA    PIAIMG
ØØ159A DØAC B4   DØØB     A          ANDA   TALKOF   TURN OFF TALK LIGHT
ØØ16ØA DØAF BA   DØØC     A          ORA    LISTON   TURN ON LISTEN LIGHT
ØØ161A DØB2 B7   EØ72     A          STA    PRB      SEND TO FRONT PANEL
ØØ162A DØB5 B7   DØ13     A          STA    PIAIMG   UP DATE PIAIMG
ØØ163A DØB8 BE   DØØ4     A          LDX    LMEMPT   GET LIST. START ADD
ØØ164A DØBB BF   EØ4Ø     A          STX    ADDHØ    PUT IN DMAC CHAMMEL Ø
ØØ165A DØBE BF   EØ4C     A          STX    ADDH3    PUT IN DMAC CHANNEL 3
ØØ166A DØC1 BE   DØØØ     A          LDX    LBYCNT   GET NO. OF BYTES
ØØ167A DØC4 BF   EØ42     A          STX    BYTEHØ   PUT IN DMAC CHANNEL Ø
ØØ168A DØC7 BF   EØ4E     A          STX    BYTEH3   PUT IN DMAC CHANNEL 3
ØØ169A DØCA 86   Ø4       A          LDA    #$Ø4     SELECT UP COUNT, TSC
ØØ17ØA DØCC B7   EØ5Ø     A          STA    CHCON    STEAL, & MEM WRITE
ØØ171A DØCF 86   81       A          LDA    #$81     SELECT IRQ ON DEND
ØØ172A DØD1 B7   EØ55     A          STA    INTCON   PUT IN DMAC
ØØ173A DØD4 86   ØØ       A          LDA    #$ØØ
ØØ174A DØD6 B7   EØ56     A          STA    DCHAIN   DISABLE DATA CHAIN FEATURE
```



```
PAGE  ØØ4  GPIA2   .SA:Ø  GPIA1

ØØ175A DØD9 86   Ø1       A          LDA    #$Ø1
ØØ176A DØDB B7   EØ54     A          STA    PRICON   ENABLE CH. Ø TRANSFER REQUEST
ØØ177A DØDE 2Ø   49   D129           BRA    WAIT
ØØ178                       *
ØØ179                       *SET UP DMAC FOR TALK ROUTINE
ØØ18Ø                       *
ØØ181A DØEØ B6   DØ13     A TACS     LDA    PIAIMG
ØØ182A DØE3 B4   DØØD     A          ANDA   LISTOF   TURN OFF LISTEN LIGHT
ØØ183A DØE6 BA   DØØA     A          ORA    TALKON   TURN ON TALK LIGHT
ØØ184A DØE9 B7   EØ72     A          STA    PRB      SEND TO FRONT PANEL
ØØ185A DØEC B7   DØ13     A          STA    PIAIMG   UPDATE PIAIMG
ØØ186                       *
ØØ187                       *LOAD TALKER MEM BUFFER
ØØ188A DØEF BE   DØØ6     A          LDX    TMEMPT . GET MEM POINTER
ØØ189A DØF2 1ØBE DØØ2     A          LDY    TBYCNT   GET NO. OF BYTES
ØØ19ØA DØF6 C6   ØØ       A          LDB    #$ØØ
ØØ191A DØF8 E7   84       A LOOP3    STB    Ø,X      STORE DATA BYTE IN MEM
ØØ192A DØFA 5C                       INCB            INC. NO. TO BE STORED
ØØ193A DØFB 3Ø   1F       A          LEAX   -1,X     DEC. ADDRESS POINTER
ØØ194A DØFD 31   3F       A          LEAY   -1,Y     DECRIMENT BYTE COUNT
ØØ195A DØFF 26   F7   DØF8           BNE    LOOP3    IF NOT LAST DO ANOTHER
ØØ196A D1Ø1 BE   DØØ6     A          LDX    TMEMPT   GET TALK BUF ADD.
ØØ197A D1Ø4 BF   EØ4Ø     A          STX    ADDHØ    PUT IN DMAC CHANNEL Ø
ØØ198A D1Ø7 BF   EØ4C     A          STX    ADDH3    PUT IN DMAC CHANNEL 3
ØØ199A D1ØA BE   DØØ2     A          LDX    TBYCNT   GET NO. OF BYTES   /
ØØ2ØØA D1ØD BF   EØ42     A          STX    BYTEHØ   PUT IN DMAC CHANNEL Ø
ØØ2Ø1A D11Ø BF   EØ4E     A          STX    BYTEH3   PUT IN DMAC CHANNEL 3
ØØ2Ø2A D113 86   Ø5       A          LDA    #$Ø5     SELECT UP COUNT, TSC
ØØ2Ø3A D115 B7   EØ5Ø     A          STA    CHCON    STEAL , & MEM READ
ØØ2Ø4A D118 86   81       A          LDA    #$81     SELECT IRQ ON DEND
ØØ2Ø5A D11A B7   EØ55     A          STA    INTCON   PUT IN DMAC
ØØ2Ø6A D11D 86   ØØ       A          LDA    #$ØØ     DISABLE DATA CHAINING
ØØ2Ø7A D11F B7   EØ56     A          STA    DCHAIN   FEATURE OF DMAC
ØØ2Ø8A D122 86   Ø1       A          LDA    #$Ø1
ØØ2Ø9A D124 B7   EØ54     A          STA    PRICON   ENABLE CH Ø TRANSFER
ØØ21ØA D127 2Ø   ØØ   D129           BRA    WAIT
ØØ211                       *
ØØ212                       *WAIT LOOP - WAITS FOR A DMA REQUEST TO OCCUR. TALK/L
ØØ213                       *            CONDITION RECOGNIZED AND DMAC HAS BEEN S
ØØ214                       *            ACCORDINGLY. WAIT LOOP ALSO CHECKS FOR A
ØØ215                       *            IN GPIA ADDRESS STATUS.
ØØ216                       *            IF ADDRESSED DIFFERENTLY THAN IT WAS
ØØ217                       *            WHEN WAIT LOOP ENTERRED AN E1 TRAP
ØØ218
ØØ219                       *            WILL BE PRODUCED.
ØØ22Ø                       *
ØØ221A D129 B6   EØ62     A WAIT     LDA    R2R      LOAD GPIA ADD. STATUS
ØØ222A D12C 85   8Ø       A          BITA   #$8Ø     IS MA BIT SET
ØØ223A D12E 27   22   D152           BEQ    OFF      IF NO, GO TURN OFF DMAC
ØØ224
ØØ225A D13Ø 85   Ø4       A          BITA   #$Ø4     IS LACS BIT SET
ØØ226A D132 27   11   D145           BEQ    TACBIT   IF NO, GO TEST TACS
ØØ227A D134 B6   DØ13     A          LDA    PIAIMG   IF YES, SEE IF LISTEN FLAG SET
ØØ228A D137 85   Ø2       A          BITA   #$Ø2     IS LISTEN FLAG SET
ØØ229A D139 26   EE   D129           BNE    WAIT     IF YES, ALL IS `OK` - CHECK R2
ØØ23ØA D13B 86   ØØ       A TRAP1    LDA    #$ØØ     IF NO, ALL IS NOT OK
ØØ231A D13D B7   EØ54     A          STA    PRICON   TURN OFF DMAC
ØØ232A D14Ø 86   E1       A          LDA    #$E1     LOAD ACC A WITH TRAP CODE
```



```
PAGE  ØØ5  GPIA2   .SA:Ø  GPIA1

ØØ233A D142 16   ØØ9B D1EØ        LBRA   TRAP      GO TO TRAP ROUTINE
ØØ234A D145 85   Ø8      A TACBIT BITA   #$Ø8      IS TACS BIT SET
ØØ235A D147 27   EØ   D129        BEQ    WAIT      IF NO, GO TEST ADDRESS STATUS
ØØ236A D149 B6   DØ13    A        LDA    PIAIMG    IF YES, CHECK TALK
ØØ237A D14C 85   Ø1      A        BITA   #$Ø1      IS TALK FLAG SET
ØØ238A D14E 26   D9   D129        BNE    WAIT      IF YES, ALL `OK`
ØØ239A D15Ø 2Ø   E9   D13B        BRA    TRAP1     IF NO, GO SET E1 DISPLAY
ØØ24ØA D152 86   ØØ      A OFF    LDA    #$ØØ      TURN OFF DMAC
ØØ241A D154 B7   EØ54    A        STA    PRICON
ØØ242A D157 16   FF1F DØ79        LBRA   MONIT     GO TO MONITOR
ØØ243
ØØ244                      *
ØØ245                      *
ØØ246                      ***INTERRUPT ROUTINE**
ØØ247                      *
ØØ248                      *
ØØ249                      *
ØØ25Ø                      *CHECK FOR DMAC INTERRUPT
ØØ251                      *
ØØ252A D15A B6   EØ5Ø    A IRQ    LDA    CHCON     LOAD DMAC CONTROL REG
ØØ253A D15D 85   8Ø      A        BITA   #$8Ø      IS IRQ FROM DMAC
ØØ254A D15F 27   ØF   D17Ø        BEQ    GPIA      IF NO, GO CHECK GPIA
ØØ255A D161 85   Ø1      A        BITA   #$Ø1      IS DMA IN TALK MODE
ØØ256A D163 27   Ø7   D16C        BEQ    TRAP2     IF NO, GO TO TRAP2
ØØ257A D165 86   ØØ      A        LDA    #$ØØ      IF YES,
ØØ258A D167 B7   EØ54    A        STA    PRICON    TURN OFF DMAC
ØØ259A D16A 2Ø   31   D19D        BRA    TALAST    GO SEND LAST BYTE
ØØ26ØA D16C 86   E2      A TRAP2  LDA    #$E2      LOAD ACC A WITH TRAP2 CODE
ØØ261A D16E 2Ø   7Ø   D1EØ        BRA    TRAP      GO TO TRAP ROUTINE
ØØ262                      *
ØØ263                      *CHECK FOR GPIA INTERRUPT
ØØ264                      *
ØØ265A D17Ø B6   EØ6Ø    A GPIA   LDA    RØR       GET GPIA IRQ STATUS
ØØ266A D173 85   8Ø      A        BITA   #$8Ø      IS IRQ FROM GPIA
ØØ267A D175 27   11   D188        BEQ    TRAP3     IF NO, GO TO TRAP3 ROUTINE
ØØ268A D177 85   Ø4      A        BITA   #$Ø4      IS CMD BIT SET
ØØ269A D179 26   11   D18C        BNE    RLC       IF YES, IS RLC SET
ØØ27ØA D17B 85   Ø2      A        BITA   #$Ø2      IF NO, GO TO TRAP3
ØØ271A D17D 27   Ø9   D188        BEQ    TRAP3     IF NO, GO TO TRAP3
ØØ272A D17F 85   Ø1      A BI     BITA   #$Ø1      IF END IS YES, IS BI
ØØ273A D181 26   2A   D1AD        BNE    LILAST    IF YES, GET LAST BYTE
ØØ274A D183 B6   EØ6Ø    A        LDA    RØR       IF NO, LOAD ROR
ØØ275A D186 2Ø   F7   D17F        BRA    BI        AND TEST BI AGAIN
ØØ276A D188 86   E3      A TRAP3  LDA    #$E3      LOAD ACC A WITH TRAP3 CODE
ØØ277A D18A 2Ø   54   D1EØ        BRA    TRAP      GO TO TRAP ROUTINE
ØØ278A D18C B6   EØ61    A RLC    LDA    R1R       GET GPIA COMMAND STATUS
ØØ279A D18F 85   Ø8      A        BITA   #$Ø8      IS RLC SET
ØØ28ØA D191 27   Ø6   D199        BEQ    TRAP4     IF NO, GO TO TRAP4
ØØ281A D193 85   4Ø      A        BITA   #$4Ø      IF YES, IS REM SET
ØØ282A D195 27   3B   D1D2        BEQ    REMOFF    IF NO, TURN OFF REN LIGHT
ØØ283A D197 2Ø   21   D1BA        BRA    REMON     IF YES, TURN ON REN LIGHT
ØØ284A D199 86   E4      A TRAP4  LDA    #$E4      LOAD ACC A WITH TRAP4 CODE
ØØ285A D19B 2Ø   43   D1EØ        BRA    TRAP      GO TO TRAP ROUTINE
ØØ286                      *
ØØ287                      *SEND LAST BYTE AS A TALKER
ØØ288                      *
ØØ289A D19D 86   Ø1      A TALAST LDA    #$Ø1
ØØ29ØA D19F B7   EØ63    A        STA    R3W       SET feoi
```



```
PAGE  ØØ6  GPIA2   .SA:Ø  GPIA1

ØØ291A D1A2 BE   EØ4Ø    A        LDX    ADDHØ     GET ADD OF LAST BYTE
ØØ292A D1A5 A7   84      A        STA    Ø,X       GET LAST BYTE
ØØ293A D1A7 B7   EØ67    A        STA    R7W       SEND LAST BYTE
ØØ294A D1AA 16   FECC DØ79        LBRA   MONIT
ØØ295                      *
ØØ296                      *RECEIVE LAST BYTE ROUTINE
ØØ297                      *
ØØ298A D1AD 86   ØØ      A LILAST LDA    #$ØØ
ØØ299A D1AF B7   EØ54    A        STA    PRICON    TURN OFF DMAC
ØØ3ØØA D1B2 BE   EØ4Ø    A        LDX    ADDHØ     LOAD LAST BYTE ADDRESS IN X RE
ØØ3Ø1A D1B5 B6   EØ67    A        LDA    R7R       GET LAST BYTE
ØØ3Ø2A D1B8 A7   84      A        STA    Ø,X       STORE IT
ØØ3Ø3
ØØ3Ø4                      *
ØØ3Ø5                      *SET REMOTE ENABLE LIGHT INDICATOR
ØØ3Ø6                      *
ØØ3Ø7A D1BA 86   DØØ8    A REMON  LDA    RENON     GET RENON MASK
ØØ3Ø8A D1BD BA   DØ13    A        ORA    PIAIMG    `OR` WITH CURRENT STATUS
ØØ3Ø9A D1CØ B7   DØ13    A        STA    PIAIMG    UPDATE PIAIMG
ØØ31ØA D1C3 B7   EØ72    A        STA    PRB
ØØ311                      *
ØØ312                      *RESET AND SET GPIA MASK REG
ØØ313                      *
ØØ314A D1C6 86   ØØ      A RESETM LDA    #$ØØ
ØØ315A D1C8 B7   EØ6Ø    A        STA    RØW       RESET GPIA IRQ MASK
ØØ316A D1CB B6   DØØF    A        LDA    MASK
ØØ317A D1CE B7   EØ6Ø    A        STA    RØW       SET GPIA IRQ MASK
ØØ318A D1D1 3B                    RTI
ØØ319                      *
ØØ32Ø                      *RESET REMOTE ENABLE LIGHT INDICATOR
ØØ321                      *
ØØ322A D1D2 B6   DØØ9    A REMOFF LDA    RENOFF    GET RENOFF MASK
ØØ323A D1D5 B4   DØ13    A        ANDA   PIAIMG    TURN OFF REN BIT
ØØ324A D1D8 B7   DØ13    A        STA    PIAIMG    UPDATE PIAIMG
ØØ325A D1DB B7   EØ72    A        STA    PRB       TURN REN OFF
ØØ326A D1DE 2Ø   E6   D1C6        BRA    RESETM
ØØ327                      *
ØØ328                      *TRAP ROUTINE
ØØ329                      *
ØØ33ØA D1EØ B7   EØ7Ø    A TRAP   STA    PRA       SEND TRAP CODE TO DISPLAY
ØØ331A D1E3 2Ø   FB   D1EØ        BRA    TRAP
ØØ332                             END
TOTAL ERRORS ØØØØØ--ØØØØØ
TOTAL WARNINGS ØØØØØ--ØØØØØ
```

AN-834

USING THE MC68000 AND THE MC6845 FOR A COLOR GRAPHICS SYSTEM

By
David L. Ruhberg
Microcomputer Systems Engineer
Motorola Semiconductor

Probably the slowest link in most computerized control systems is the display of information for human interpretation. The commonly used black and white monitor can display an adequate amount of information in most cases.

In applications where a large amount of information must be displayed in the same screen area, a color graphics system can easily provide this information by using a wide range of contrasting colors. Until recently the high cost of sophisticated components and color monitors required to generate and display color information has probably been the main prohibitive factor in development of these systems.

Recently the cost of components and color monitors has moderated to the point that using a color graphics system offers a viable solution to information display, ranging from the video games market to complex control systems.

A state-of-the-art color graphics system using the MC68000 16-bit microprocessor (MPU) with an economical MC6845 CRT controller (CRTC) is described in this application note. Hardware improvement is evident in data movement occurring in 16-bit words and multiply and divide commands while software compatibilities are greatly enhanced through the use of a processor that executes instructions which can operate on 8-, 16-, or 32-bit operands.

The general approach to a color graphics system is straightforward and almost identical to a black and white graphics system. A typical black and white graphics system is shown in Figure 1. The MPU has two responsibilities to the graphics system: first, to initially program the CRTC, and second, to transfer data to the display RAM.

Once the clock circuitry is running, the CRTC is initialized and the address lines to the dislay RAM begin incrementing sequentially. As this occurs, the appropriate data from the display RAM is loaded into the shift register and then gated out serially by the dot clock input to the shift register. The display monitor then interprets the data as either turning a particular pixel on or off.

A color graphics system (Figure 2) uses the same principle as the black and white system except that it has to control three color guns (red, green, and blue) instead of just one. Therefore, there is an increase in the amount of hardware involved, but not in complexity. The software becomes more involved due to the fact that more information is being handled and displayed. The basic display system works on the principle that three bits (one for each color) controls each pixel instead of just one as in a black and white system. If two guns are on, the resulting color is a combination of the two. If all guns are on, white is the result. With this configuration a total of eight colors, including black and white, are available. Since the three bits needed to control a pixel do not fit into an eight-bit byte evenly, the unused bits could be used to obtain more colors or some other function. In addition, color systems usually require a separate sync input.

The versatility of the internal architecture of the MC68000 (Figure 3) enhances the effectiveness of the color graphics system. Besides containing a 32-bit program counter yielding 16 megabytes of direct addressing range, the MC68000 also contains eight 32-bit data registers (D0-D7) and seven 32-bit address registers (A0-A6). The eight data registers are used for byte (8-bit), word (16-bit), and long word (32-bit) data operations. The seven address registers and the stack pointer may be used for word and long word address operations. In addition, all address and data registers may be used as index registers.

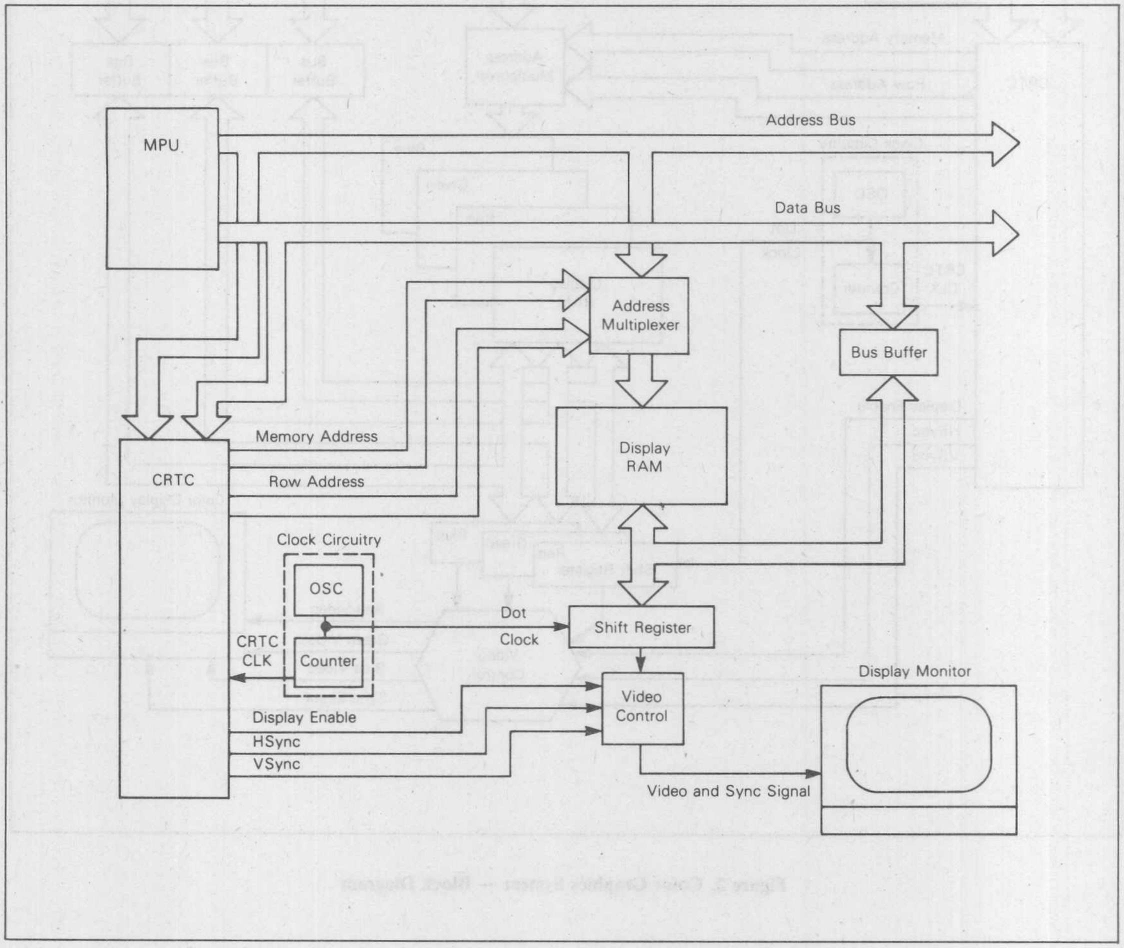


Figure 1. Black and White Graphics System — Block Diagram

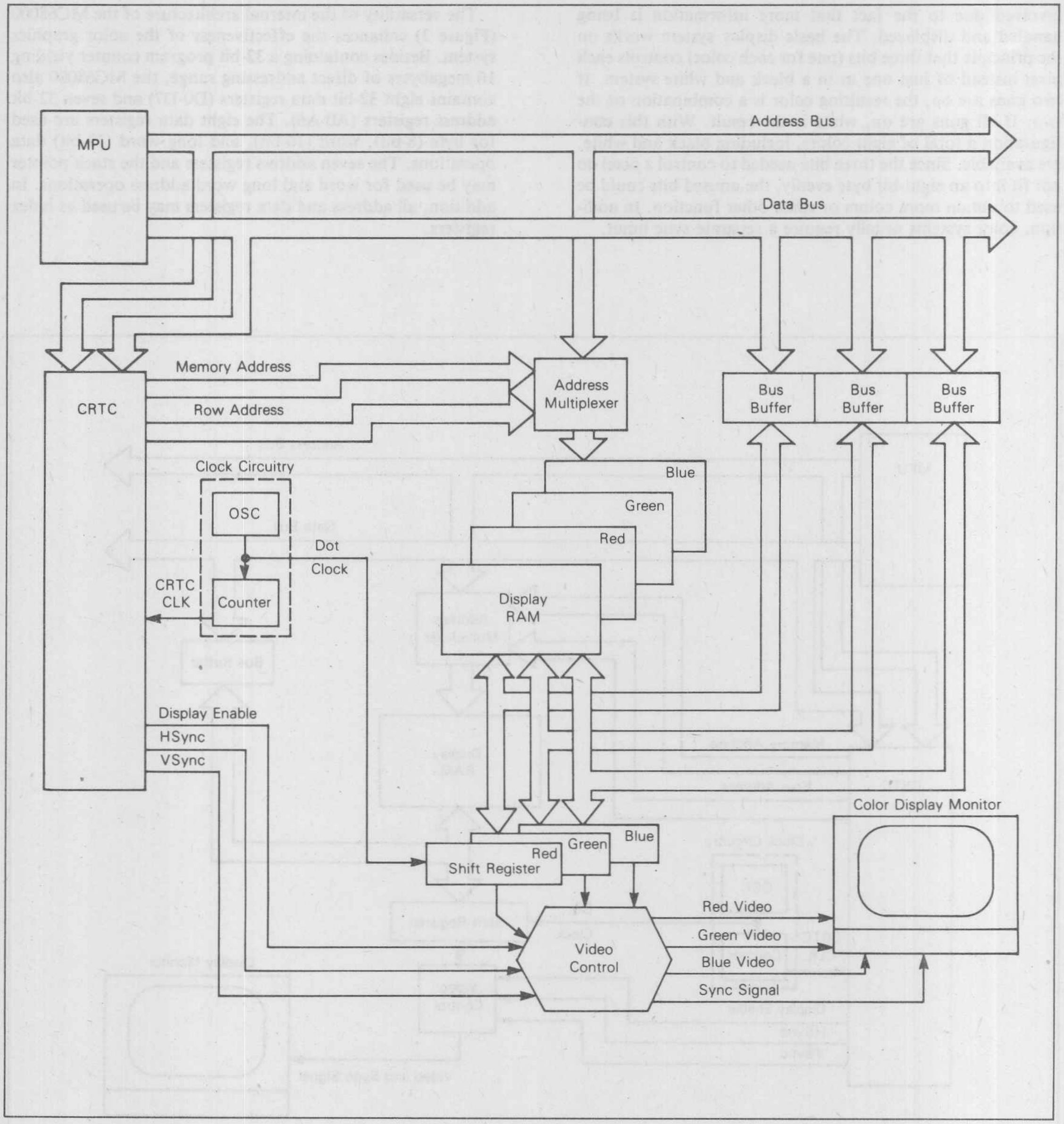


Figure 2. Color Graphics System — Block Diagram

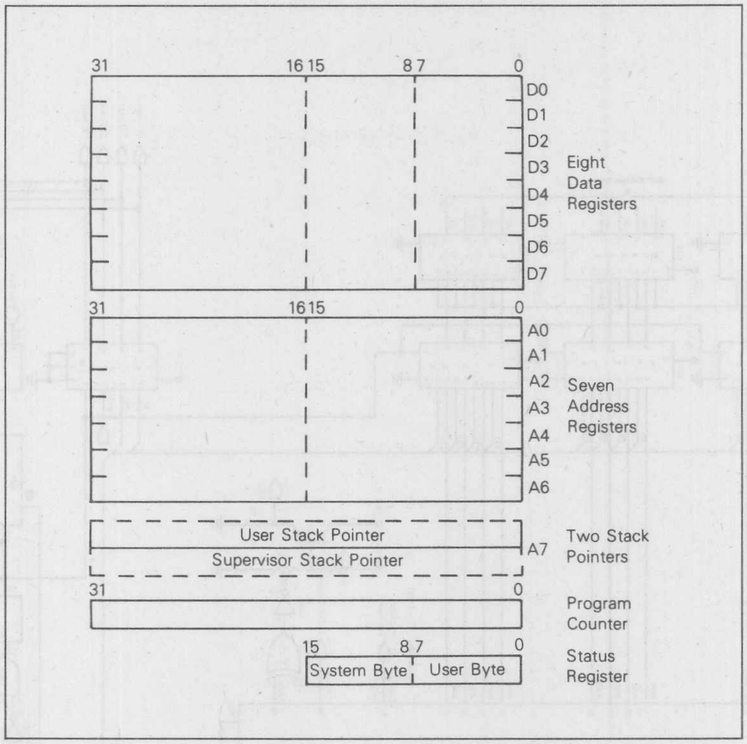


Figure 3. MC68000 Programming Model

SYSTEM HARDWARE DESCRIPTION AND FEATURES

This graphics system consists of two boards: a CPU board and a video board. The CPU board contains the processor, scratch-pad RAM, stack RAM, the program EPROM, and a terminal interface. The video board contains the CRTC, display RAM, multiplexers and buffers, parallel-to-serial shift registers, and the D/A drivers for the color display monitor.

An MC68000 Design Module (MEX68000KDM) is used as the CPU board. The resources available on the MC68000 Design Module allow more design time to be spent on the unique features of the system. The major portions of the system provided by the Design Module are the MPU (MC68000), the address decoding for the EPROM, a terminal interface, and all the software functions provided by the resident monitor (MACSbug). Included in the MACSbug is a transparent down-load feature which allows the system to communicate through the terminal to another system. The other system can provide the access to the floppy disks needed by this color graphics system for saving a full screen of data at a time.

The video board (Figure 4) contains more of the unique hardware features of the color graphics system. The video board can be separated into seven areas: the clock circuit, CRT controller, the DTACK circuit, the bus multiplexers and buffers, the display RAM, the shift registers, and the D/A converter drivers.

The clock circuit generates the five timing signals used throughout the video board; they are: a dot clock, a CRTC clock, a 2X dot clock, a shift register load, and a $\phi 2$ signal. The dot clock is used to drive the serial shift registers. The CRTC clock is used to drive the CRTC. The 2X dot clock and the shift register load are gated together to generate the parallel load (PLOAD) and chip select (PCS) signals for the shift registers and display RAM, respectively. The $\phi 2$ signal is also used to control accesses to the display RAM. A timing diagram of these signals is shown in Figure 5.

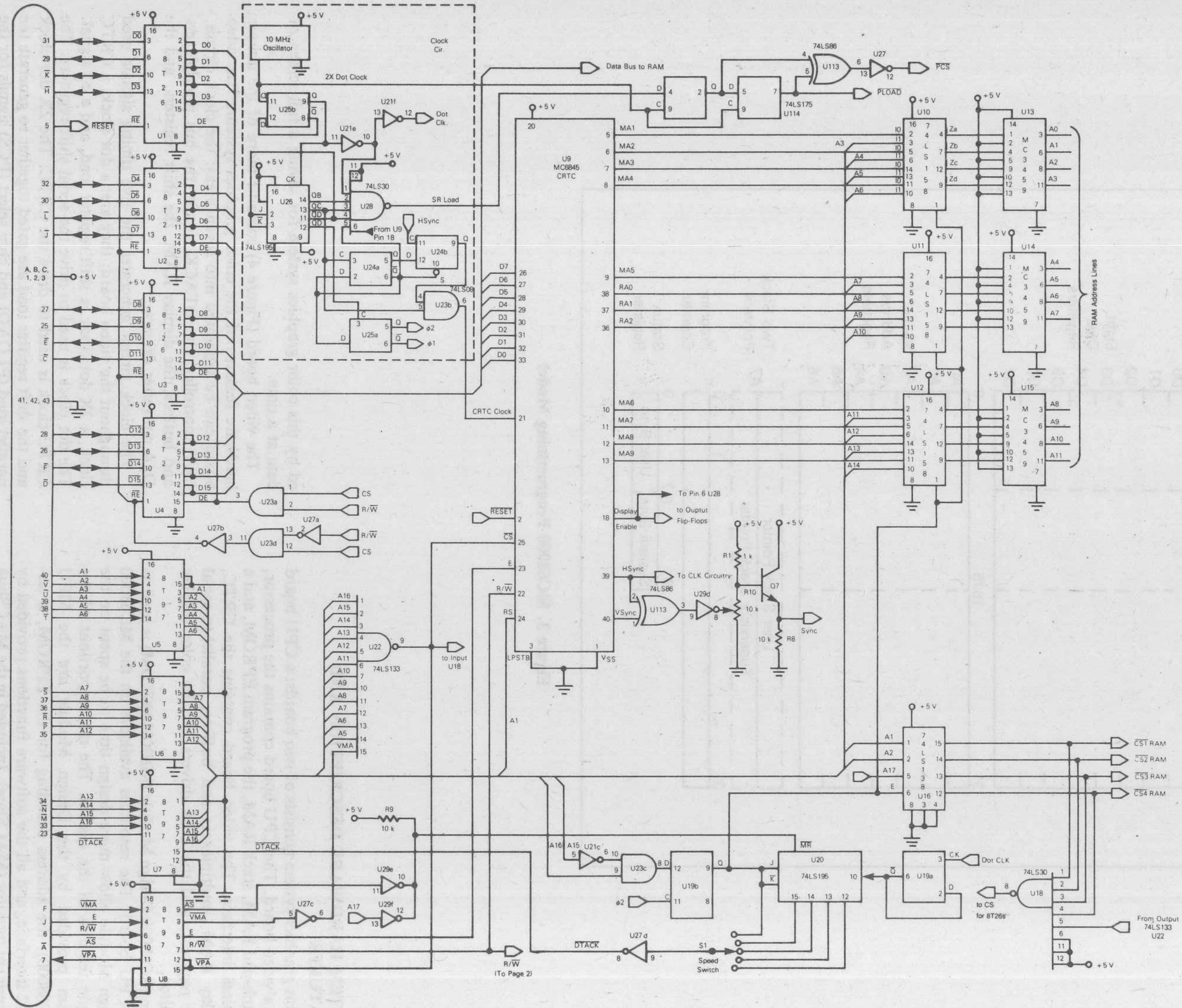


Figure 4. Color Graphics System Schematic (Sheet 1 of 3)

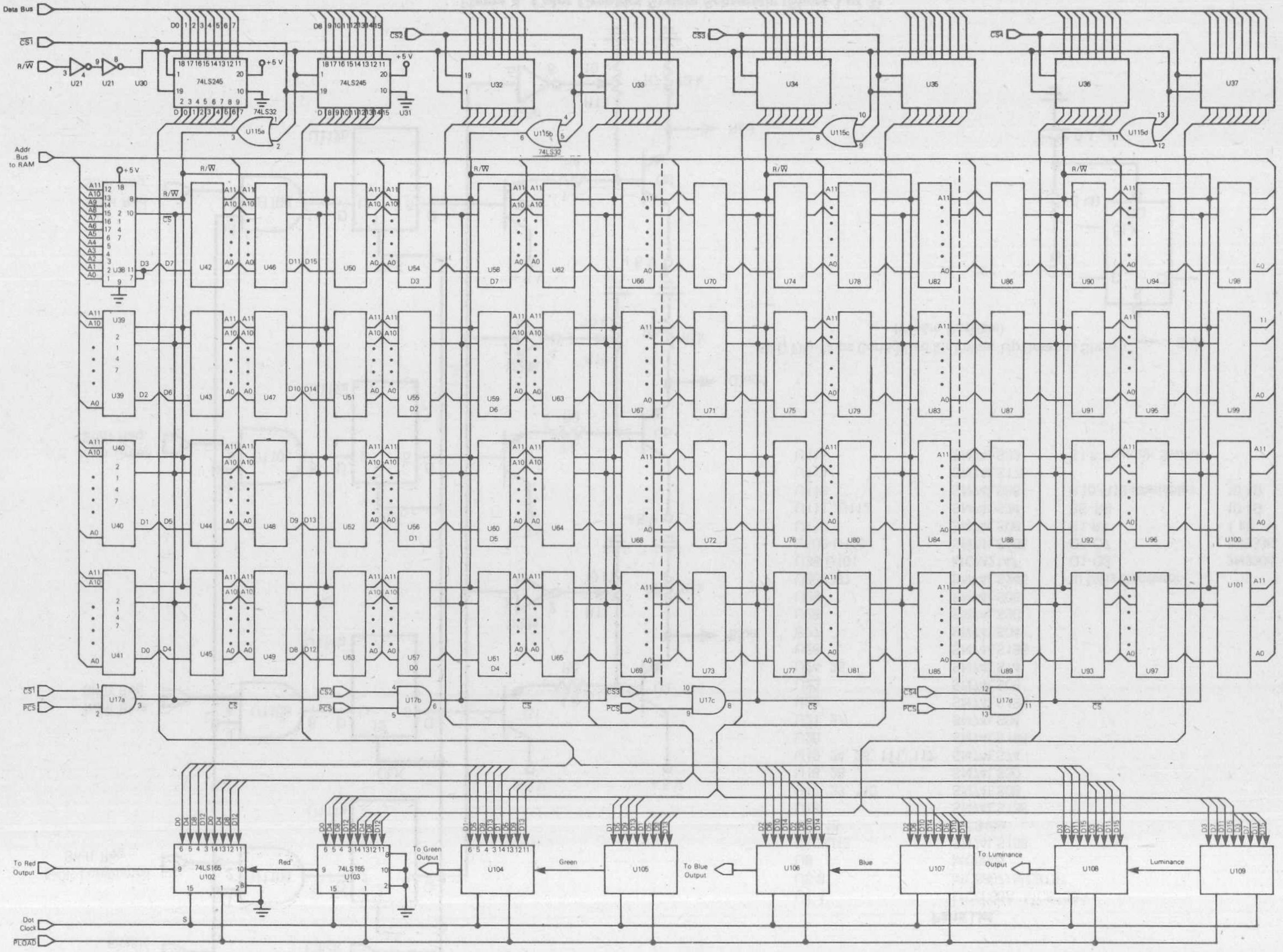
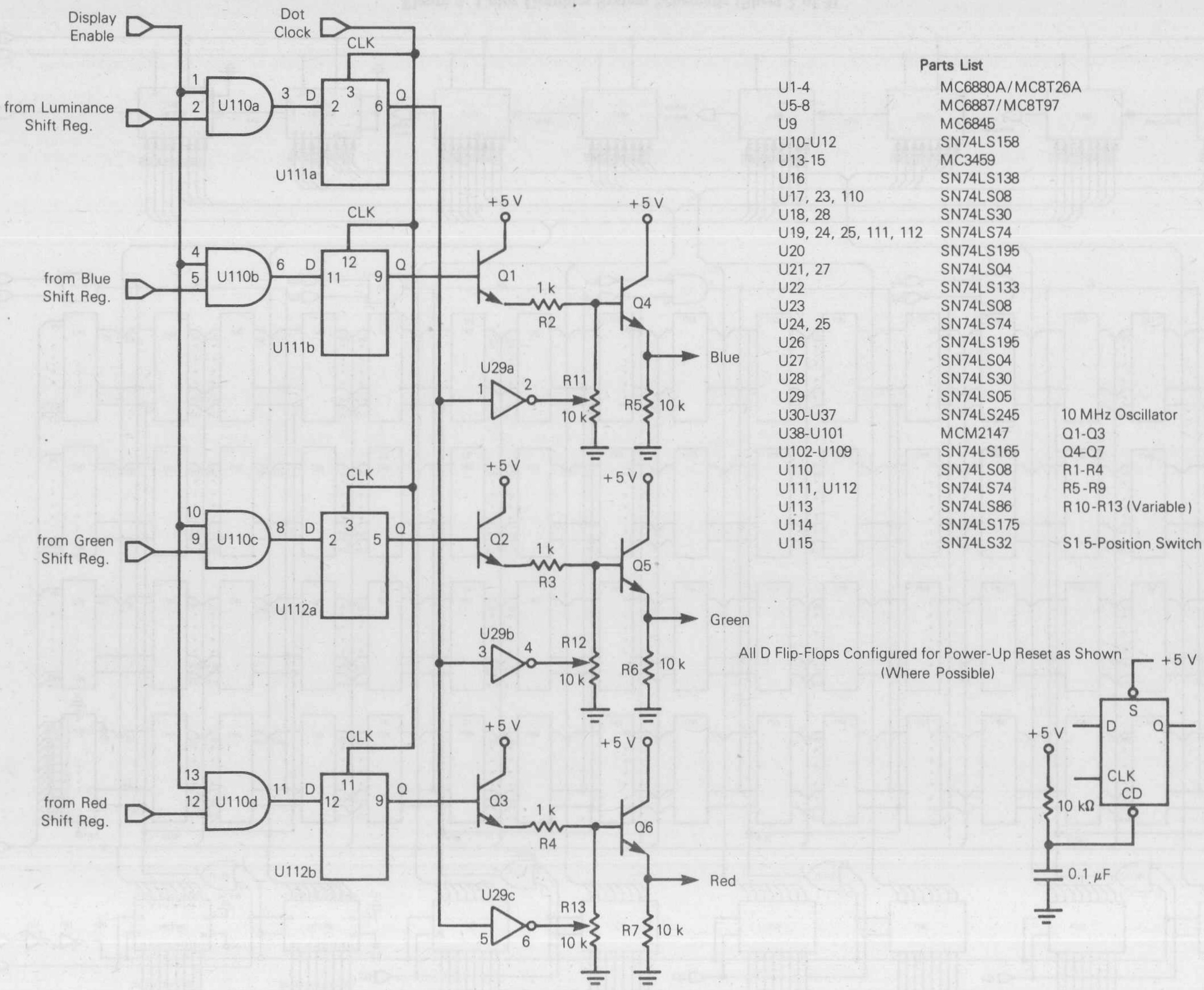


Figure 4. Color Graphics System Schematic (Sheet 2 of 3)



Parts List

U1-4	MC6880A/MC8T26A		
U5-8	MC6887/MC8T97		
U9	MC6845		
U10-U12	SN74LS158		
U13-15	MC3459		
U16	SN74LS138		
U17, 23, 110	SN74LS08		
U18, 28	SN74LS30		
U19, 24, 25, 111, 112	SN74LS74		
U20	SN74LS195		
U21, 27	SN74LS04		
U22	SN74LS133		
U23	SN74LS08		
U24, 25	SN74LS74		
U26	SN74LS195		
U27	SN74LS04		
U28	SN74LS30		
U29	SN74LS05		
U30-U37	SN74LS245	10 MHz Oscillator	
U38-U101	MCM2147	Q1-Q3	2N3904
U102-U109	SN74LS165	Q4-Q7	2N5336
U110	SN74LS08	R1-R4	1 kΩ
U111, U112	SN74LS74	R5-R9	10 kΩ
U113	SN74LS86	R 10-R13 (Variable)	10 kΩ
U114	SN74LS175		
U115	SN74LS32	S1 5-Position Switch	

Figure 4. Color Graphics System Schematic (Sheet 3 of 3)

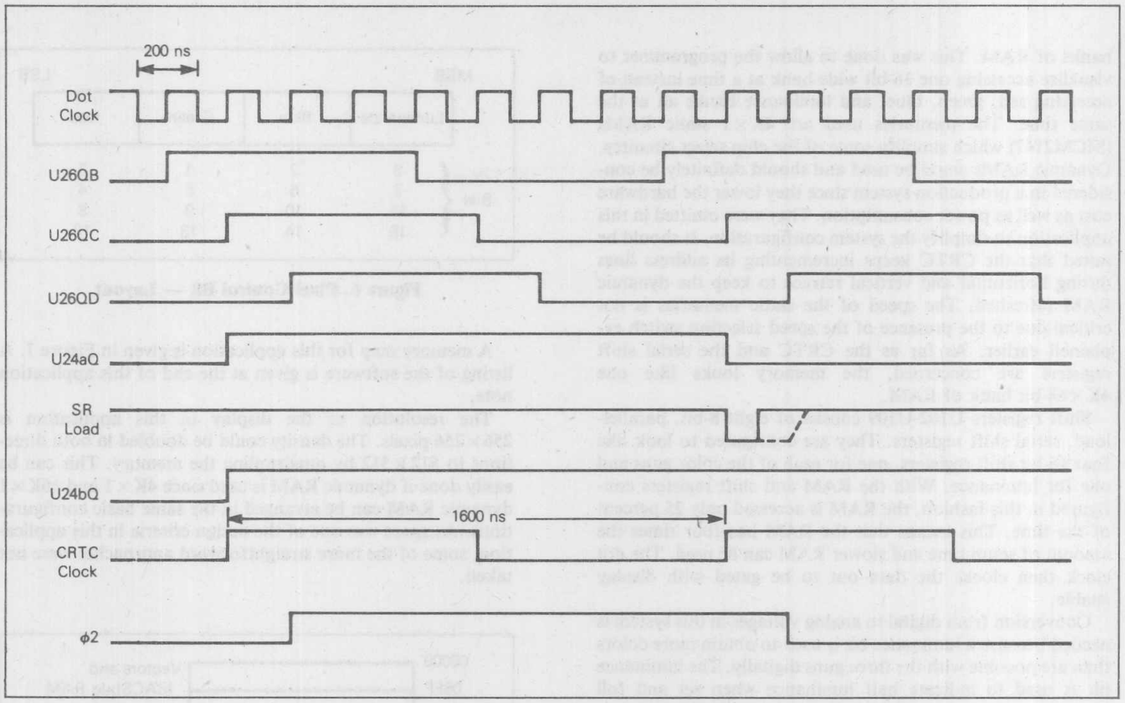


Figure 5. Clock Circuitry Timing Signals

The MC6845 CRT controller (CRTC) is a programmable controller used to prepare the information in the display RAM for use by a video display monitor. The CRTC generates the signals required to provide data at the appropriate times. Since the length and period between these signals varies from system to system, the CRTC is designed to be programmed by an MPU. In this system the internal registers are accessible synchronously through hex ($) address locations $1FFFD and $1FFFF. After programming, the CRTC provides the addresses, horizontal and vertical sync signals, and the display enable signal to the display system. The addresses, output by the CRTC in conjunction with the parallel chip select ($\overline{\text{PCS}}$) signal, are responsible for the correct data getting to the serial shift registers at the correct time. The horizontal and vertical sync signals, after being "exclusively ORed," generate the sync signal required by the color display monitor. The display enable (DE) signal is gated (U28) into either the clock circuitry to inhibit the parallel load and $\overline{\text{PCS}}$ signals or is gated (ANDed at U110, if a low represents black on the screen) with the data stream to keep the guns in the CRT off during vertical and horizontal retrace. In some cases, DE must be delayed due to specific requirements of the CRT being used. A one-shot on the output of the DE pin is usually more than adequate for providing the delay.

The $\overline{\text{DTACK}}$ circuitry is used to return an asynchronous data transfer acknowledge ($\overline{\text{DTACK}}$) signal to the MC68000 from a synchronous device (the display RAM). The ϕ2 signal from the clock circuitry in conjunction with address lines A15 and A16 develop the $\overline{\text{DTACK}}$ response required by the MC68000. When the display RAM address is between $10000-$17FFF, the $\overline{\text{DTACK}}$ signal is returned in 400 nanosecond increments from zero up to 1600 nanoseconds after the enabling signal goes out to the multiplexers. This time is selected by the RAM speed switch, S1. Returning $\overline{\text{DTACK}}$ to the processor is the asynchronous access method by which the MC68000 can access external devices (RAM, ROM, and peripherals). This access method was chosen over the synchronous access method used to address the CRTC because it is faster and, since this is a highly repetitive operation, any time saved here will be significant in the overall speed of the system. The synchronous access method is used to access the CRTC since the CRTC is only initialized once and this method uses fewer components.

The multiplexers and buffers are used to feed the various control signals to the rest of the system. Multiplexers U10, U11, and U12 determine which address bus will access the display RAM. When the control signal is high, the MC68000 has access to the RAM and when low, the CRTC has access. Buffers U13, U14, and U15 are used to drive the large number of devices on the address bus. Data buffers U30-U37 are used to isolate the four banks of RAM from each other. Buffers are also used for almost all the signals coming onto the video board. These board buffers interface with the modified EXORciser bus which the Design Module uses. This bus has only sixteen address lines coming from the Design Module, so address line A17 must be run separately to keep the display RAM from being accessed at the same time MACSbug or the controller program is accessed (addresses $20000 and $22000).

The display RAM is organized into four banks (red, green, blue, and luminance). However, the address lines are configured so that consecutive words are located in consecutive banks of RAM. This was done to allow the programmer to visualize accessing one 16-bit wide bank at a time instead of accessing red, green, blue, and luminance banks all at the same time. The memories used are 4K×1 static RAMs (MCM2147) which simplify some of the chip select circuitry. Dynamic RAMs could be used and should definitely be considered in a production system since they lower the hardware cost as well as power consumption. They were omitted in this application to simplify the system configuration. It should be noted that the CRTC keeps incrementing its address lines during horizontal and vertical retrace to keep the dynamic RAM refreshed. The speed of the static memories is not critical due to the presence of the speed selection switch explained earlier. As far as the CRTC and the serial shift registers are concerned, the memory looks like one 4K×64-bit bank of RAM.

Shift registers U102-U109 consist of eight 8-bit, parallel-load, serial shift registers. They are configured to look like four 16-bit shift registers, one for each of the color guns and one for luminance. With the RAM and shift registers configured in this fashion, the RAM is accessed only 25 percent of the time. This means that the RAM has four times the amount of setup time and slower RAM can be used. The dot clock then clocks the data out to be gated with display enable.

Conversion from digital to analog voltages in this system is needed because a luminance bit is used to obtain more colors than are possible with the three guns digitally. The luminance bit is used to indicate half luminance when set and full luminance when clear. When all guns are off, the screen is black and the state of the luminance bit has no effect. Since the color display monitor uses an analog input on each gun, any number of colors may be obtained if the supporting hardware is provided. The D/A conversion used in this system was done to save space. A cleaner method would be to use special D/A converters and special line drivers for this function.

SOFTWARE DESCRIPTION AND CONSIDERATIONS

The software included to exercise this system consists of five basic commands:

CM — Clear Memory
BX — Box Draw
Q8 — Random Line
ED — Edit
BA} — Provides the capability of saving (BA) a screen on
SH} floppy disk and calling (SH) it back.

The clear memory (CM) command clears the screen. The box drawing (BX) command draws continuously concentric boxes which close in on each other. This gives the effect of running up a hallway. The random line (Q8) drawing command picks random points and connects them together until they form a multisided polygon and then it continues to repeat that shape, all the while collapsing in on itself and changing colors. A scaling function has been implemented to keep the figure occupying a major portion of the screen. The edit (ED) command allows the user to draw figures on the screen using the cursor controls on the terminal and allows a choice of colors. The BA command is used to store a screen full of data on floppy disk while the SH command is used to call it from the floppy disk and display it on the screen.

Each of the routines which write to the display RAM use the basic data layout for every pixel on the screen. Each pixel is controlled by four bits. Each bit corresponds to either luminance, blue, green, or red, as shown in Figure 6.

MSB LSB

	Luminance	Blue	Green	Red
Bits	3	2	1	0
	7	6	5	4
	11	10	9	8
	15	14	13	12

Figure 6. Pixel Control Bit — Layout

A memory map for this application is given in Figure 7. A listing of the software is given at the end of this application note.

The resolution of the display in this application is 256×256 pixels. The density could be doubled in both directions to 512×512 by quadrupling the memory. This can be easily done if dynamic RAM is used since 4K×1 and 16K×1 dynamic RAM can be arranged in the same basic configurations. As space was one of the design criteria in this application, some of the more straightforward approaches were not taken.

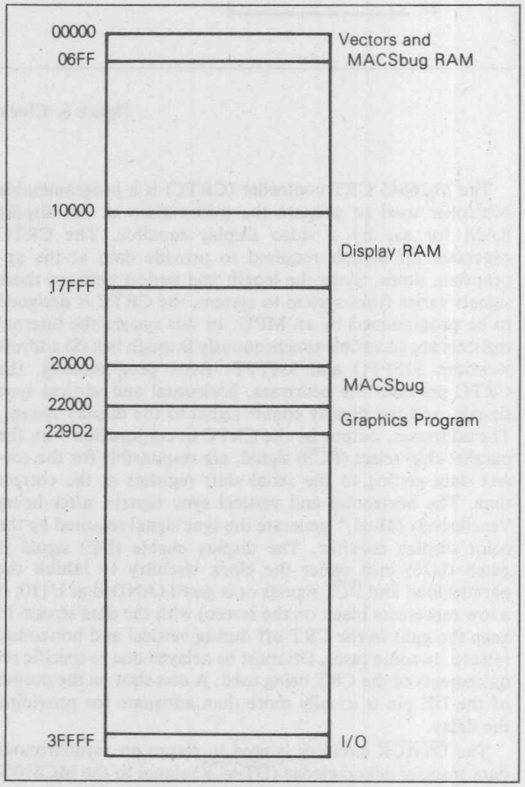


Figure 7. Memory Map

Thanks to Don Voss of Motorola Microsystems for his suggestions on the hardware and his splendid job on the software.

```
SETUP4.S  MC68ØØØ ASM REV= 1.ØF- COPYRIGHT BY MOTOROLA 1978   PAGE  1

  1Ø        ØØØØØØØØ      ORG $ØØØØ
  2Ø                     *
  3Ø                     *
  4Ø                     *
  5Ø        ØØØ2ØØF6     MACSBUG EQU $2ØØF6
  6Ø        ØØØ21BC2     OUTPUT2 EQU $21BC2
  7Ø        ØØØ21F18     FIXBUF EQU $21F18
  8Ø        ØØØ2ØØEE     MSG EQU $2ØØEE
  9Ø        ØØØØ1ØØØ     X1 EQU $1ØØØ
 1ØØ        ØØØØ1ØØ1     Y1 EQU $1ØØ1
 11Ø        ØØØØ1ØØ2     X2 EQU $1ØØ2
 12Ø        ØØØØ1ØØ3     Y2 EQU $1ØØ3
 13Ø        ØØØØ1Ø1Ø     COLOR EQU $1Ø1Ø
 14Ø        ØØØØ1Ø11     NCOLOR EQU $1Ø11
 15Ø        ØØØØ1Ø12     OCOLOR EQU $1Ø12
 16Ø        ØØØØ1Ø14     NUMPT EQU $1Ø14
 17Ø        ØØØØ1Ø16     SCALE EQU $1Ø16
 18Ø        ØØØØ1Ø18     RANADD EQU $1Ø18
 19Ø        ØØØØ1Ø8Ø     ARRAY EQU $1Ø8Ø
 2ØØ        ØØØØ11ØØ     TABLECH EQU $11ØØ
 21Ø        ØØØØ18ØØ     CMDTAB EQU $18ØØ
 22Ø                     *
 23Ø                     *
 24Ø ØØØØØØ 2Ø78Ø578     SETUP MOVE.L $578,AØ
 25Ø ØØØØØ4 227CØØØØ18ØØ  MOVE.L #CMDTAB,A1
 26Ø ØØØØØA 21C9Ø578      MOVE.L A1,$578
 27Ø ØØØØØE 3Ø18         SETUP1 MOVE (AØ)+,DØ
 28Ø ØØØØ1Ø ØC4ØFFFF      CMP #$FFFF,DØ
 29Ø ØØØØ14 67Ø6          BEQ.S SETUP2
 3ØØ ØØØØ16 32CØ          MOVE DØ,(A1)+
 31Ø ØØØØ18 22D8          MOVE.L (AØ)+,(A1)+
 32Ø ØØØØ1A 6ØF2          BRA SETUP1
 33Ø ØØØØ1C B1FCØØØ22ØØØ SETUP2 CMP.L #$22ØØØ,AØ
 34Ø ØØØØ22 6AØ8          BPL.S INIT
 35Ø ØØØØ24 2Ø7CØØØ22Ø82  MOVE.L #$22Ø82,AØ
 36Ø ØØØØ2A 6ØE2          BRA SETUP1
 37Ø ØØØØ2C 328Ø         INIT MOVE DØ,(A1)
 38Ø ØØØØ2E 2Ø7CØØØ22ØD2  MOVE.L #$22ØD2,AØ
 39Ø ØØØØ34 3Ø3CØØØØ      MOVE #$ØØØØ,DØ
 4ØØ ØØØØ38 13CØØØØ1FFFD INIT1 MOVE.B DØ,$1FFFD
 41Ø ØØØØ3E 1218          MOVE.B (AØ)+,D1
 42Ø ØØØØ4Ø 4E71          NOP
 43Ø ØØØØ42 13C1ØØØ1FFFF  MOVE.B D1,$1FFFF
 44Ø ØØØØ48 524Ø          ADD #1,DØ
 45Ø ØØØØ4A ØC4ØØØ1Ø      CMP #$ØØ1Ø,DØ
 46Ø ØØØØ4E 66E8          BNE INIT1
 47Ø ØØØØ5Ø 227CØØØ229F6  MOVE.L #$229F6,A1
 48Ø ØØØØ56 247CØØØØ11ØØ  MOVE.L #TABLECH,A2
 49Ø ØØØØ5C 3Ø3CØ3Ø2      MOVE #77Ø,DØ
 5ØØ ØØØØ6Ø 14D9         SETUP21 MOVE.B (A1)+,(A2)+
   Ø ØØØØ62 534Ø          SUB #1,DØ
   Ø ØØØØ64 66FA          BNE  SETUP21
 52Ø ØØØØ66 6Ø14          BRA.S RETURN
 53Ø ØØØØ68 2Ø7CØØØ1ØØØØ CM MOVE.L #$1ØØØØ,AØ
 54Ø ØØØØ6E 323C2ØØØ      MOVE #$2ØØØ,D1
 55Ø ØØØØ72 428Ø             CLR.L DØ
 56Ø ØØØØ74 2ØCØ            CLRM MOVE.L DØ,(AØ)+
   Ø ØØØØ76 5341             SUB #1,D1
   Ø ØØØØ78 66FA             BNE  CLRM
   Ø ØØØØ7A 4E71             NOP
 58Ø ØØØØ7C 4EF9ØØØ2ØØF6 RETURN JMP MACSBUG
 59Ø ØØØØ82 43              NTABLE DC.W 'CM'
 6ØØ ØØØØ84 ØØØ22Ø68         DC.L $22Ø68
 61Ø ØØØØ88 53               DC.W 'SH'
 62Ø ØØØØ8A ØØØ22ØE2         DC.L $22ØE2
 63Ø ØØØØ8E 42               DC.W 'BX'
 64Ø ØØØØ9Ø ØØØ2218A         DC.L $2218A
 65Ø ØØØØ94 45               DC.W 'ED'
 66Ø ØØØØ96 ØØØ221E8         DC.L $221E8
 67Ø ØØØØ9A 42               DC.W 'BA'
 68Ø ØØØØ9C ØØØ22454         DC.L $22454
 69Ø ØØØØAØ 51               DC.W 'Q1'
 7ØØ ØØØØA2 ØØØ22498         DC.L $22498
 71Ø ØØØØA6 51               DC.W 'Q2'
 72Ø ØØØØA8 ØØØ224A4         DC.L $224A4
 73Ø ØØØØAC 51               DC.W 'Q3'
 74Ø ØØØØAE ØØØ224BØ         DC.L $224BØ
 75Ø ØØØØB2 51               DC.W 'Q4'
 76Ø ØØØØB4 ØØØ224BC         DC.L $224BC
 77Ø ØØØØB8 51               DC.W 'Q5'
 78Ø ØØØØBA ØØØ224C8         DC.L $224C8
 79Ø ØØØØBE 51               DC.W 'Q9'
 8ØØ ØØØØCØ ØØØ226Ø6         DC.L $226Ø6
 81Ø ØØØØC4 48               DC.W 'HP'
 82Ø ØØØØC6 ØØØ226AC         DC.L $226AC
 83Ø ØØØØCA 51               DC.W 'Q8'
 84Ø ØØØØCC ØØØ22818         DC.L $22818
 85Ø ØØØØDØ FFFF             DC.W $FFFF
 86Ø                        *
 87Ø                        *
 88Ø                        *
 89Ø                        *
 9ØØ ØØØØD2 27              CRTC DC.B $27
 91Ø ØØØØD3 2Ø               DC.B $2Ø
 92Ø ØØØØD4 22               DC.B $22
 93Ø ØØØØD5 A3               DC.B $A3
 94Ø ØØØØD6 2Ø               DC.B $2Ø
 95Ø ØØØØD7 Ø6               DC.B $Ø6
 96Ø ØØØØD8 1F               DC.B $1F
 97Ø ØØØØD9 1F               DC.B $1F
 98Ø ØØØØDA 1Ø               DC.B $1Ø
 99Ø ØØØØDB Ø7               DC.B 7
1ØØØ ØØØØDC ØØØØØØØØ         DC.L Ø
1Ø1Ø ØØØØEØ ØØØØ             DC.W Ø
1Ø2Ø                        *
1Ø3Ø                        *
1Ø4Ø ØØØØE2 61ØØØØØ4        SH BSR SHQ
1Ø5Ø ØØØØE6 6Ø94             BRA RETURN
1Ø6Ø ØØØØE8 4EB9ØØØ21BC2 SHQ JSR OUTPUT2
1Ø7Ø ØØØØEE 227CØØØ3FF21  MOVE.L #$3FF21,A1
```

```
1Ø8Ø ØØØØF4 61ØØØØ78      SH1 BSR INPUT
1Ø9Ø ØØØØF8 ØCØØØØØD       CMP.B #$ØD,DØ
11ØØ ØØØØFC 67Ø8           BEQ.S SH2
111Ø ØØØØFE ØCØØØØFF       CMP.B #$FF,DØ
112Ø ØØØ1Ø2 66FØ           BNE SH1
113Ø ØØØ1Ø4 6Ø4Ø           BRA.S SH3
114Ø ØØØ1Ø6 61ØØØØ66      SH2 BSR INPUT
115Ø ØØØ1ØA ØCØØØØØA       CMP.B #$ØA,DØ
116Ø ØØØ1ØE 67F6           BEQ SH2
117Ø ØØØ11Ø ØCØØØØØØ       CMP.B #Ø,DØ
118Ø ØØØ114 67FØ           BEQ SH2
119Ø ØØØ116 ØCØØØØFF       CMP.B #$FF,DØ
12ØØ ØØØ11A 672A           BEQ.S SH3
121Ø ØØØ11C 4EB9ØØØ21F18   JSR FIXBUF
122Ø ØØØ122 2CFC4552524F   MOVE.L #'ERRO',(A6)+
123Ø ØØØ128 2CFC52203B43   MOVE.L #'R ;C',(A6)+
124Ø ØØØ12E 2CFC4845434B   MOVE.L #'HECK',(A6)+
125Ø ØØØ134 2CFC2Ø46494C   MOVE.L #' FIL',(A6)+
126Ø ØØØ13A 2CFC452Ø2Ø2Ø   MOVE.L #'E   ',(A6)+
127Ø ØØØ14Ø 4EF9ØØØ2ØØEE   JMP MSG
128Ø ØØØ146 2Ø7CØØØ1ØØØØ  SH3 MOVE.L #$1ØØØØ,AØ
129Ø ØØØ14C 1Ø3CØØ55       MOVE.B #$55,DØ
13ØØ ØØØ15Ø 61ØØØØ2A       BSR OUTPUT
131Ø ØØØ154 61ØØØØ18      SH4 BSR INPUT
132Ø ØØØ158 12ØØ           MOVE.B DØ,D1
133Ø ØØØ15A 61ØØØØ12       BSR INPUT
134Ø ØØØ15E E14Ø           ASL 8,DØ
135Ø ØØØ16Ø 1ØØ1           MOVE.B D1,DØ
136Ø ØØØ162 3ØCØ           MOVE.W DØ,(AØ)+
137Ø ØØØ164 B1FCØØØ17F8Ø   CMP.L #$17F8Ø,AØ
138Ø ØØØ16A 66E8           BNE SH4
139Ø ØØØ16C 4E75           RTS
14ØØ ØØØ16E 1Ø11          INPUT MOVE.B (A1),DØ
141Ø ØØØ17Ø Ø2ØØØØØ1       AND.B #1,DØ
142Ø ØØØ174 67F8           BEQ INPUT
143Ø ØØØ176 1Ø29ØØØ2       MOVE.B 2(A1),DØ
144Ø ØØØ17A 4E75           RTS
145Ø ØØØ17C 1E11          OUTPUT MOVE.B (A1),D7
146Ø ØØØ17E Ø2Ø7ØØØ2       AND.B #2,D7
147Ø ØØØ182 67F8           BEQ OUTPUT
148Ø ØØØ184 134ØØØØ2       MOVE.B DØ,2(A1)
149Ø ØØØ188 4E75           RTS
15ØØ ØØØ18A 424Ø          BX CLR DØ
151Ø ØØØ18C 32ØØ           MOVE DØ,D1
152Ø ØØØ18E 34ØØ           MOVE DØ,D2
153Ø ØØØ19Ø 363CØØ3F      BX3 MOVE #$3F,D3
154Ø ØØØ194 2Ø7CØØØ1ØØØØ   MOVE.L #$1ØØØØ,AØ
155Ø ØØØ19A 61ØØØØ16      BX1 BSR SHOW
156Ø ØØØ19E 5543           SUB #2,D3
157Ø ØØØ1AØ 6AØ2           BPL.S BX2
158Ø ØØØ1A2 6ØEC           BRA BX3
159Ø ØØØ1A4 524Ø          BX2 ADD #1,DØ
16ØØ ØØØ1A6 5241           ADD #1,D1
161Ø ØØØ1A8 5242           ADD #1,D2
162Ø ØØØ1AA D1FCØØØØØ2Ø2   ADD.L #514,AØ
```

```
1630 0001B0 60E8          BRA BX1
1640 0001B2 3803         SHOW MOVE D3,D4
1650 0001B4 30C0         BX11 MOVE D0,(A0)+
   0 0001B6 5344          SUB #1,D4
   0 0001B8 66FA          BNE  BX11
1670 0001BA 3080          MOVE D0,(A0)
1680 0001BC 3803          MOVE D3,D4
1690 0001BE E544          ASL 2,D4
1700 0001C0 D1FC00000080 BX22 ADD.L #128,A0
1710 0001C6 3081          MOVE D1,(A0)
   0 0001C8 5344          SUB #1,D4
   0 0001CA 66F4          BNE  BX22
1730 0001CC 3803          MOVE D3,D4
1740 0001CE 3080          MOVE D0,(A0)
1750 0001D0 3100         BX33 MOVE D0,-(A0)
   0 0001D2 5344          SUB  #1,D4
   0 0001D4 66FA          BNE  BX33
1770 0001D6 3803          MOVE D3,D4
1780 0001D8 E544          ASL 2,D4
1790 0001DA 91FC00000080 BX44 SUB.L #128,A0
1800 0001E0 3082          MOVE D2,(A0)
   0 0001E2 5344          SUB  #1,D4
   0 0001E4 66F4          BNE  BX44
1820 0001E6 4E75          RTS
1830                      *
1840                      *
1850                      *
1860 0001E8 11FC00801000 ED MOVE.B #$80,X1
1870 0001EE 11FC00801001  MOVE.B #$80,Y1
1880 0001F4 11FC00001011  MOVE.B #0,NCOLOR
1890 0001FA 6100014E     ED1 BSR BLINK
1900 0001FE 61000004      BSR CMD
1910 000202 60F6         BRA ED1
1920 000204 61000230     CMD BSR READK
1930 000208 0C010020      CMP.B #$20,D1
1940 00020C 6A48          BPL.S RTS
1950 00020E 0C01000B      CMP.B #$B,D1
1960 000212 673C          BEQ.S UPARROW
1970 000214 0C01000A      CMP.B #$A,D1
1980 000218 673E          BEQ.S DWARROW
1990 00021A 0C01000C      CMP.B #$C,D1
2000 00021E 673E          BEQ.S RTARROW
2010 000220 0C010008      CMP.B #$8,D1
2020 000224 673E          BEQ.S LTARROW
2030 000226 0C010001      CMP.B #$1,D1
2040 00022A 673E          BEQ.S CMD1  CHARMODE
2050 00022C 0C010003      CMP.B #$3,D1
2060 000230 6756          BEQ.S CMD2  NCOLOR
2070 000232 0C010004      CMP.B #$4,D1
2080 000236 6738          BEQ.S CMD3
2090 000238 0C01000D      CMP.B #$0D,D1
2100 00023C 673E          BEQ.S CR
2110 00023E 0C010005      CMP.B #$5,D1
2120 000242 6732          BEQ.S CMD4
2130 000244 0C010011      CMP.B #$11,D1
```

```
214Ø ØØØ248 66ØA           BNE.S RTS1
215Ø ØØØ24A 588F           ADD.L #4,A7
216Ø ØØØ24C 6ØØØFE2E       BRA RETURN
217Ø ØØØ25Ø 53381ØØ1      UPARROW SUB.B #1,Y1
218Ø ØØØ254 4241          RTS1 CLR D1
219Ø ØØØ256 4E75          RTS RTS
22ØØ ØØØ258 52381ØØ1      DWARROW ADD.B #1,Y1
221Ø ØØØ25C 6ØF6           BRA RTS1
222Ø ØØØ25E 52381ØØØ      RTARROW ADD.B #1,X1
223Ø ØØØ262 6ØFØ           BRA RTS1
224Ø ØØØ264 53381ØØØ      LTARROW SUB.B #1,X1
225Ø ØØØ268 6ØEA           BRA RTS1
226Ø ØØØ26A 588F          CMD1 ADD.L #4,A7
227Ø ØØØ26C 6ØØØØ132       BRA CHARED
228Ø ØØØ27Ø 588F          CMD3 ADD.L #4,A7
229Ø ØØØ272 6ØØØØ1A8       BRA DOT
23ØØ ØØØ276 588F          CMD4 ADD.L #4,A7
231Ø ØØØ278 6ØØØFF8Ø       BRA ED1
232Ø ØØØ27C 5E381ØØ1      CR ADD.B #7,Y1
233Ø ØØØ28Ø 11FCØØØØ1ØØØ   MOVE.B #Ø,X1
234Ø ØØØ286 6ØCC           BRA RTS1
235Ø ØØØ288 61ØØØ1AC      CMD2 BSR READK
236Ø ØØØ28C 267CØØØØ1Ø11   MOVE.L #NCOLOR,A3
237Ø ØØØ292 ØCØ1ØØ52       CMP.B #'R',D1
238Ø ØØØ296 6758           BEQ.S RED
239Ø ØØØ298 ØCØ1ØØ47       CMP.B #'G',D1
24ØØ ØØØ29C 6758           BEQ.S GREEN
241Ø ØØØ29E ØCØ1ØØ42       CMP.B #'B',D1
242Ø ØØØ2A2 6758           BEQ.S BLUE
243Ø ØØØ2A4 ØCØ1ØØ57       CMP.B #'W',D1
244Ø ØØØ2A8 6758           BEQ.S WHITE
245Ø ØØØ2AA ØCØ1ØØ5A       CMP.B #'Z',D1
246Ø ØØØ2AE 6758           BEQ.S BLACK
247Ø ØØØ2BØ ØCØ1ØØ59       CMP.B #'Y',D1
248Ø ØØØ2B4 6758           BEQ.S YELLOW
249Ø ØØØ2B6 ØCØ1ØØ4D       CMP.B #'M',D1
25ØØ ØØØ2BA 6758           BEQ.S MAG
251Ø ØØØ2BC ØCØ1ØØ43       CMP.B #'C',D1
252Ø ØØØ2CØ 6758           BEQ.S CYAN
253Ø ØØØ2C2 ØCØ1ØØ54       CMP.B #'T',D1
254Ø ØØØ2C6 6758           BEQ.S DRED
255Ø ØØØ2C8 ØCØ1ØØ48       CMP.B #'H',D1
256Ø ØØØ2CC 6758           BEQ.S DGR
257Ø ØØØ2CE ØCØ1ØØ4E       CMP.B #'N',D1
258Ø ØØØ2D2 6758           BEQ.S DBLUE
259Ø ØØØ2D4 ØCØ1ØØ45       CMP.B #'E',D1
26ØØ ØØØ2D8 6758           BEQ.S DWH
261Ø ØØØ2DA ØCØ1ØØ55       CMP.B #'U',D1
262Ø ØØØ2DE 6758           BEQ.S DYEL
263Ø ØØØ2EØ ØCØ1ØØ2C       CMP.B #',',D1
264Ø ØØØ2E4 6758           BEQ.S DMAG
265Ø ØØØ2E6 ØCØ1ØØ56       CMP.B #'V',D1
266Ø ØØØ2EA 6758           BEQ.S DCYAN
267Ø ØØØ2EC 4241          RTS2 CLR D1
268Ø ØØØ2EE 4E75          RTS
```

```
269Ø ØØØ2FØ 16BCØØØ9      RED MOVE.B #$9,(A3)
27ØØ ØØØ2F4 6ØF6           BRA RTS2
271Ø ØØØ2F6 16BCØØØA      GREEN MOVE.B #$A,(A3)
272Ø ØØØ2FA 6ØFØ           BRA RTS2
273Ø ØØØ2FC 16BCØØØC      BLUE MOVE.B #$C,(A3)
274Ø ØØØ3ØØ 6ØEA           BRA RTS2
275Ø ØØØ3Ø2 16BCØØØF      WHITE MOVE.B #$F,(A3)
276Ø ØØØ3Ø6 6ØE4           BRA RTS2
277Ø ØØØ3Ø8 16BCØØØØ      BLACK MOVE.B #Ø,(A3)
278Ø ØØØ3ØC 6ØDE           BRA RTS2
279Ø ØØØ3ØE 16BCØØØB      YELLOW MOVE.B #$B,(A3)
28ØØ ØØØ312 6ØD8           BRA RTS2
281Ø ØØØ314 16BCØØØD      MAG MOVE.B #$D,(A3)
282Ø ØØØ318 6ØD2           BRA RTS2
283Ø ØØØ31A 16BCØØØE      CYAN MOVE.B #$E,(A3)
284Ø ØØØ31E 6ØCC           BRA RTS2
285Ø ØØØ32Ø 16BCØØØ1      DRED MOVE.B #1,(A3)
286Ø ØØØ324 6ØC6           BRA RTS2
287Ø ØØØ326 16BCØØØ2      DGR MOVE.B #2,(A3)
288Ø ØØØ32A 6ØCØ           BRA RTS2
289Ø ØØØ32C 16BCØØØ4      DBLUE MOVE.B #4,(A3)
29ØØ ØØØ33Ø 6ØBA           BRA RTS2
291Ø ØØØ332 16BCØØØ7      DWH MOVE.B #7,(A3)
292Ø ØØØ336 6ØB4           BRA RTS2
293Ø ØØØ338 16BCØØØ3      DYEL MOVE.B #3,(A3)
294Ø ØØØ33C 6ØAE           BRA RTS2
295Ø ØØØ33E 16BCØØØ5      DMAG MOVE.B #5,(A3)
296Ø ØØØ342 6ØA8           BRA RTS2
297Ø ØØØ344 16BCØØØ6      DCYAN MOVE.B #6,(A3)
298Ø ØØØ348 6ØA2           BRA RTS2
299Ø                      *
3ØØØ ØØØ34A 1238lØØØ      BLINK MOVE.B X1,D1
3Ø1Ø ØØØ34E 1438lØØ1       MOVE.B Y1,D2
3Ø2Ø ØØØ352 61ØØØ226       BSR GETADD
3Ø3Ø ØØØ356 4643           NOT D3
3Ø4Ø ØØØ358 ØCØ3ØØØF      BL2 CMP.B #$F,D3
3Ø5Ø ØØØ35C 67Ø6           BEQ.S BL1
3Ø6Ø ØØØ35E E84B           LSR 4,D3
3Ø7Ø ØØØ36Ø E849           LSR 4,D1
3Ø8Ø ØØØ362 6ØF4           BRA BL2
3Ø9Ø ØØØ364 11C11Ø12      BL1 MOVE.B D1,OCOLOR
31ØØ ØØØ368 1Ø3CØØØF      BL3 MOVE.B #$F,DØ
311Ø ØØØ36C 1238lØØØ       MOVE.B X1,D1
312Ø ØØØ37Ø 1438lØØ1       MOVE.B Y1,D2
313Ø ØØØ374 61ØØØ1DE       BSR DSP
314Ø ØØØ378 61ØØØØDØ       BSR DLY
315Ø ØØØ37C 42ØØ           CLR.B DØ
316Ø ØØØ37E 61ØØØ1D4       BSR DSP
317Ø ØØØ382 61ØØØØC6       BSR DLY
318Ø ØØØ386 1Ø381Ø12       MOVE.B OCOLOR,DØ
319Ø ØØØ38A 61ØØØ1C8       BSR DSP
32ØØ ØØØ38E 61ØØØØBA       BSR DLY
321Ø ØØØ392 1Ø39ØØØ3FFØ1   MOVE.B $3FFØ1,DØ
322Ø ØØØ398 Ø2ØØØØØ1       AND.B #1,DØ
323Ø ØØØ39C 67CA           BEQ BL3
```

```
324Ø ØØØ39E 4E75            RTS
325Ø                        *
326Ø ØØØ3AØ 31F81ØØØ1ØØ2 CHARED MOVE X1,X2
327Ø ØØØ3A6 61A2            BSR BLINK
328Ø ØØØ3A8 61ØØFE5A        BSR CMD
329Ø ØØØ3AC 4AØ1            TST.B D1
33ØØ ØØØ3AE 67FØ            BEQ CHARED
331Ø ØØØ3BØ 61ØØØØØ4        BSR CHAR
332Ø ØØØ3B4 6ØEA            BRA CHARED
333Ø ØØØ3B6 Ø4Ø1ØØ2Ø     CHAR SUB.B #$2Ø,D1
334Ø ØØØ3BA E741            ASL 3,D1
335Ø ØØØ3BC 267CØØØØ11ØØ    MOVE.L #TABLECH,A3
336Ø ØØØ3C2 Ø281ØØØØØ3FF    AND.L #$3FF,D1
337Ø ØØØ3C8 D7C1            ADD.L D1,A3
338Ø ØØØ3CA 3C3CØØØ4        MOVE #4,D6
339Ø ØØØ3CE 4245         CHARED1 CLR D5
34ØØ ØØØ3DØ ØB13         CHARED2 BTST D5,(A3)
341Ø ØØØ3D2 6636            BNE.S SET
342Ø ØØØ3D4 52381ØØ2     CHARED3 ADD.B #1,X2
343Ø ØØØ3D8 5245            ADD #1,D5
344Ø ØØØ3DA ØC45ØØ1Ø        CMP #16,D5
345Ø ØØØ3DE 6618            BNE.S CHARED4
346Ø ØØØ3EØ 52381ØØ3        ADD.B #1,Y2
347Ø ØØØ3E4 11F81ØØØ1ØØ2    MOVE.B X1,X2
348Ø ØØØ3EA D7F8ØØØ2        ADD.L $2,A3
   Ø ØØØ3EE 5346            SUB  #1,D6
   Ø ØØØ3FØ 66DC            BNE  CHARED1
35ØØ ØØØ3F2 5Ø381ØØØ        ADD.B #8,X1
351Ø ØØØ3F6 4E75            RTS
352Ø ØØØ3F8 ØC45ØØØ8     CHARED4 CMP #8,D5
353Ø ØØØ3FC 66D2            BNE CHARED2
354Ø ØØØ3FE 52381ØØ3        ADD.B #1,Y2
355Ø ØØØ4Ø2 11F81ØØØ1ØØ2    MOVE.B X1,X2
356Ø ØØØ4Ø8 6ØC6            BRA CHARED2
357Ø ØØØ4ØA 1Ø381Ø11     SET MOVE.B NCOLOR,DØ
358Ø ØØØ4ØE 12381ØØ2        MOVE.B X2,D1
359Ø ØØØ412 14381ØØ3        MOVE.B Y2,D2
36ØØ ØØØ416 61ØØØ13C        BSR DSP
361Ø ØØØ41A 6ØB8            BRA CHARED3
362Ø                        *
363Ø ØØØ41C 1Ø381Ø11     DOT MOVE.B NCOLOR,DØ
364Ø ØØØ42Ø 12381ØØØ        MOVE.B X1,D1
365Ø ØØØ424 14381ØØ1        MOVE.B Y1,D2
366Ø ØØØ428 61ØØØ12A        BSR DSP
367Ø ØØØ42C 61ØØFF1C        BSR BLINK
368Ø ØØØ43Ø 61ØØFDD2        BSR CMD
369Ø ØØØ434 6ØE6            BRA DOT
37ØØ                        *
371Ø ØØØ436 1239ØØØ3FFØ1 READK MOVE.B $3FFØ1,D1
372Ø ØØØ43C Ø2Ø1ØØØ1        AND.B #1,D1
373Ø ØØØ44Ø 67F4            BEQ READK
374Ø ØØØ442 1239ØØØ3FFØ3    MOVE.B $3FFØ3,D1
375Ø ØØØ448 4E75            RTS
376Ø ØØØ44A 3C3CØØFF     DLY MOVE #$ØØFF,D6
377Ø ØØØ44E 5346         DLY1 SUB #1,D6
```

```
378Ø ØØØ45Ø 66FC          BNE DLY1
379Ø ØØØ452 4E75          RTS
38ØØ                     *
381Ø                     *
382Ø                     *
383Ø ØØØ454 2Ø7CØØØ1ØØØØ BA MOVE.L #$1ØØØØ,AØ
384Ø ØØØ45A 227CØØØ3FF23  MOVE.L #$3FF23,A1
385Ø ØØØ46Ø 247CØØØ3FF21  MOVE.L #$3FF21,A2
386Ø ØØØ466 1212         L1 MOVE.B (A2),D1
387Ø ØØØ468 Ø2Ø1ØØØ2      AND.B #$2,D1
388Ø ØØØ46C 67F8          BEQ L1
389Ø ØØØ46E 1Ø3CØØ65      MOVE.B #$65,DØ
39ØØ ØØØ472 128Ø          MOVE.B DØ,(A1)
391Ø ØØØ474 1212         LOOP MOVE.B (A2),D1
392Ø ØØØ476 Ø2Ø1ØØØ2      AND.B #$2,D1
393Ø ØØØ47A 67F8          BEQ LOOP
394Ø ØØØ47C 3Ø18          MOVE (AØ)+,DØ
395Ø ØØØ47E 128Ø          MOVE.B DØ,(A1)
396Ø ØØØ48Ø EØ48          LSR 8,DØ
397Ø ØØØ482 1212         L2 MOVE.B (A2),D1
398Ø ØØØ484 Ø2Ø1ØØØ2      AND.B #$2,D1
399Ø ØØØ488 67F8          BEQ L2
4ØØØ ØØØ48A 128Ø          MOVE.B DØ,(A1)
4Ø1Ø ØØØ48C B1FCØØØ18ØØØ  CMP.L #$18ØØØ,AØ
4Ø2Ø ØØØ492 66EØ          BNE LOOP
4Ø3Ø ØØØ494 6ØØØFBE6      BRA RETURN
4Ø4Ø                     *
4Ø5Ø                     *
4Ø6Ø                     *
4Ø7Ø ØØØ498 2C7CØØØ225AC Q1 MOVE.L #$225AC,A6
4Ø8Ø ØØØ49E 3E3CØØ1Ø      MOVE #$1Ø,D7
4Ø9Ø ØØØ4A2 6Ø2E          BRA.S RUN
41ØØ ØØØ4A4 2C7CØØØ225BE Q2 MOVE.L #$225BE,A6
411Ø ØØØ4AA 3E3CØØ1Ø      MOVE #$1Ø,D7
412Ø ØØØ4AE 6Ø22          BRA.S RUN
413Ø ØØØ4BØ 2C7CØØØ225DØ Q3 MOVE.L #$225DØ,A6
414Ø ØØØ4B6 3E3CØØ1Ø      MOVE #$1Ø,D7
415Ø ØØØ4BA 6Ø16          BRA.S RUN
416Ø ØØØ4BC 2C7CØØØ225E2 Q4 MOVE.L #$225E2,A6
417Ø ØØØ4C2 3E3CØØ1Ø      MOVE #$1Ø,D7
418Ø ØØØ4C6 6ØØA          BRA.S RUN
419Ø ØØØ4C8 2C7CØØØ225F4 Q5 MOVE.L #$225F4,A6
42ØØ ØØØ4CE 3E3CØØ1Ø      MOVE #$1Ø,D7
421Ø ØØØ4D2 61ØØØØØ6     RUN BSR RUN1
422Ø ØØØ4D6 6ØØØFBA4      BRA RETURN
423Ø                     *
424Ø                     *
425Ø                     *
426Ø ØØØ4DA 3C3CØØ8Ø     RUN1 MOVE #128,D6
427Ø ØØØ4DE 61ØØØØ34      BSR RAND
428Ø ØØØ4E2 4E96         RUN2 JSR (A6)
429Ø ØØØ4E4 48E76ØØØ      MOVEM.L D1/D2,-(A7)
43ØØ ØØØ4E8 Ø241ØØ7F      AND #$7F,D1
431Ø ØØØ4EC Ø242ØØ7F      AND #$7F,D2
432Ø ØØØ4FØ 61ØØØØ68      BSR DSPLY
```

```
433Ø ØØØ4F4 44Ø1              NEG.B D1
434Ø ØØØ4F6 61ØØØØ62          BSR DSPLY
435Ø ØØØ4FA 44Ø2              NEG.B D2
436Ø ØØØ4FC 61ØØØØ5C          BSR DSPLY
437Ø ØØØ5ØØ 44Ø1              NEG.B D1
438Ø ØØØ5Ø2 61ØØØØ56          BSR DSPLY
439Ø ØØØ5Ø6 4CDFØØØ6          MOVEM.L (A7)+,D1/D2
   Ø ØØØ5ØA 5346              SUB  #1,D6
   Ø ØØØ5ØC 66D4              BNE  RUN2
   Ø ØØØ5ØE 5347              SUB  #1,D7
   Ø ØØØ51Ø 66C8              BNE  RUN1
442Ø ØØØ512 4E75              RTS
443Ø                         *
444Ø                         *
445Ø                         *
446Ø ØØØ514 61ØØØØ1C         RAND BSR RAND1
447Ø ØØØ518 32ØØ              MOVE DØ,D1
448Ø ØØØ51A 61ØØØØ16          BSR RAND1
449Ø ØØØ51E 34ØØ              MOVE DØ,D2
45ØØ ØØØ52Ø 61ØØØØ1Ø         RAND2 BSR RAND1
451Ø ØØØ524 Ø2ØØØØØF          AND.B #$F,DØ
452Ø ØØØ528 67F6              BEQ RAND2
453Ø ØØØ52A ØCØØØØØ8          CMP.B #$Ø8,DØ
454Ø ØØØ52E 67FØ              BEQ RAND2
455Ø ØØØ53Ø 4E75              RTS
456Ø ØØØ532 1Ø381Ø19         RAND1 MOVE.B RANADD+1,DØ
457Ø ØØØ536 E5ØØ              ASL.B 2,DØ
458Ø ØØØ538 DØ381Ø18          ADD.B RANADD,DØ
459Ø ØØØ53C E14Ø              ASL 8,DØ
46ØØ ØØØ53E 1Ø381Ø19          MOVE.B RANADD+1,DØ
461Ø ØØØ542 E54Ø              ASL 2,DØ
462Ø ØØØ544 DØ781Ø18          ADD RANADD,DØ
463Ø ØØØ548 Ø64Ø3619          ADD #$3619,DØ
464Ø ØØØ54C 31CØ1Ø18          MOVE DØ,RANADD
465Ø ØØØ55Ø EØ48              LSR 8,DØ
466Ø ØØØ552 4E75              RTS
467Ø                         *
468Ø                         *
469Ø                         *
47ØØ                         *DSPLY(C,X,Y)
471Ø                         *  DØ=COLOR
472Ø                         *  D1=X 8-BITS
473Ø                         *  D2=Y 8-BITS
474Ø                         *
475Ø ØØØ554 48E7FØ8Ø         DSP MOVEM.L DØ-D3/AØ,-(A7)
476Ø ØØØ558 6ØØC              BRA.S DSP1
477Ø                         *
478Ø ØØØ55A 48E7FØ8Ø         DSPLY MOVEM.L DØ-D3/AØ,-(A7)
479Ø ØØØ55E Ø6Ø1ØØ8Ø          ADD.B #128,D1
48ØØ ØØØ562 Ø6Ø2ØØ8Ø          ADD.B #128,D2
481Ø ØØØ566 Ø24ØØØØF         DSP1 AND #$F,DØ
482Ø ØØØ56A 61ØØØØØE          BSR GETADD
483Ø ØØØ56E C243              AND D3,D1
484Ø ØØØ57Ø 8Ø41              OR D1,DØ
485Ø ØØØ572 3Ø8Ø              MOVE DØ,(AØ)
```

```
486Ø ØØØ574 4CDFØ1ØF         MOVEM.L (A7)+,DØ-D3/AØ
487Ø ØØØ578 4E75             RTS
488Ø ØØØ57A Ø241ØØFF        GETADD AND #$FF,D1
489Ø ØØØ57E 363CFFFØ         MOVE #$FFFØ,D3
49ØØ ØØØ582 E142             ASL 8,D2
491Ø ØØØ584 D242             ADD D2,D1
492Ø ØØØ586 Ø281ØØØØFFFF     AND.L #$FFFF,D1
493Ø ØØØ58C 34Ø1             MOVE D1,D2
494Ø ØØØ58E E449             LSR 2,D1
495Ø ØØØ59Ø E341             ASL 1,D1
496Ø ØØØ592 2Ø7CØØØ1ØØØØ     MOVE.L #$1ØØØØ,AØ
497Ø ØØØ598 D1C1             ADD.L D1,AØ
498Ø ØØØ59A Ø242ØØØ3         AND #3,D2
499Ø ØØØ59E 67Ø8             BEQ.S DSPLY1
5ØØØ ØØØ5AØ E94Ø            DSPLY2 ASL 4,DØ
5Ø1Ø ØØØ5A2 E95B             ROL 4,D3
   Ø ØØØ5A4 5342             SUB  #1,D2
   Ø ØØØ5A6 66F8             BNE  DSPLY2
5Ø3Ø ØØØ5A8 321Ø            DSPLY1 MOVE (AØ),D1
5Ø4Ø ØØØ5AA 4E75             RTS
5Ø5Ø                        *
5Ø6Ø                        *
5Ø7Ø ØØØ5AC 36Ø1            EQU1 MOVE D1,D3
5Ø8Ø ØØØ5AE 38Ø2             MOVE D2,D4
5Ø9Ø ØØØ5BØ 4883             EXT D3
51ØØ ØØØ5B2 4884             EXT D4
511Ø ØØØ5B4 E64B             LSR 3,D3
512Ø ØØØ5B6 E64C             LSR 3,D4
513Ø ØØØ5B8 94Ø3             SUB.B D3,D2
514Ø ØØØ5BA 92Ø4             SUB.B D4,D1
515Ø ØØØ5BC 4E75             RTS
516Ø                        *
517Ø ØØØ5BE 36Ø2            EQU2 MOVE D2,D3
518Ø ØØØ5CØ 4883             EXT D3
519Ø ØØØ5C2 E64B             LSR 3,D3
52ØØ ØØØ5C4 92Ø3             SUB.B D3,D1
521Ø ØØØ5C6 38Ø1             MOVE D1,D4
522Ø ØØØ5C8 4884             EXT D4
523Ø ØØØ5CA E64C             LSR 3,D4
524Ø ØØØ5CC D4Ø4             ADD.B D4,D2
525Ø ØØØ5CE 4E75             RTS
526Ø                        *
527Ø                        *
528Ø ØØØ5DØ 36Ø2            EQU3 MOVE D2,D3
529Ø ØØØ5D2 4883             EXT D3
53ØØ ØØØ5D4 E24B             LSR 1,D3
531Ø ØØØ5D6 D2Ø3             ADD.B D3,D1
532Ø ØØØ5D8 38Ø1             MOVE D1,D4
533Ø ØØØ5DA 4884             EXT D4
534Ø ØØØ5DC E24C             LSR 1,D4
535Ø ØØØ5DE 94Ø4             SUB.B D4,D2
536Ø ØØØ5EØ 4E75             RTS
537Ø                        *
538Ø ØØØ5E2 36Ø2            EQU4 MOVE D2,D3
539Ø ØØØ5E4 4883             EXT D3
```

```
54ØØ ØØØ5E6 E64B          LSR 3,D3
541Ø ØØØ5E8 92Ø3          SUB.B D3,D1
542Ø ØØØ5EA 38Ø1          MOVE D1,D4
543Ø ØØØ5EC 4884          EXT D4
544Ø ØØØ5EE E64C          LSR 3,D4
545Ø ØØØ5FØ 94Ø4          SUB.B D4,D2
546Ø ØØØ5F2 4E75          RTS
547Ø                     *
548Ø ØØØ5F4 36Ø2         EQU5 MOVE D2,D3
549Ø ØØØ5F6 4883          EXT D3
55ØØ ØØØ5F8 E24B          LSR 1,D3
551Ø ØØØ5FA 92Ø3          SUB.B D3,D1
552Ø ØØØ5FC 38Ø1          MOVE D1,D4
553Ø ØØØ5FE 4884          EXT D4
554Ø ØØØ6ØØ E44C          LSR 2,D4
555Ø ØØØ6Ø2 D4Ø4          ADD.B D4,D2
556Ø ØØØ6Ø4 4E75          RTS
557Ø ØØØ6Ø6 2C7CØØØ225AC Q9 MOVE.L #$225AC,A6
558Ø ØØØ6ØC 3A3CØØØ2     Q91 MOVE #2,D5
559Ø ØØØ61Ø 61ØØØØ44     Q92 BSR CMQ
56ØØ ØØØ614 3E3CØØ2Ø      MOVE #$2Ø,D7
561Ø ØØØ618 61ØØFECØ      BSR RUN1
562Ø ØØØ61C 61ØØØØ2C      BSR DLYQ
563Ø ØØØ62Ø 48E7Ø4Ø2      MOVEM.L D5/A6,-(A7)
564Ø ØØØ624 61ØØØØ8E      BSR HP1
565Ø ØØØ628 4CDF4Ø2Ø      MOVEM.L (A7)+,D5/A6
566Ø ØØØ62C 61ØØØØ1C      BSR DLYQ
   Ø ØØØ63Ø 5345          SUB  #1,D5
   Ø ØØØ632 66DC          BNE  Q92
568Ø ØØØ634 61ØØØØ34      BSR LOGO
569Ø ØØØ638 DDFCØØØØØØ12  ADD.L #$12,A6
57ØØ ØØØ63E BDFCØØØ226Ø6  CMP.L #$226Ø6,A6
571Ø ØØØ644 67ØØØ1D2      BEQ Q8
572Ø ØØØ648 6ØC2          BRA Q91
573Ø ØØØ64A 283CØØØAFFFF DLYQ MOVE.L #$ØØØAFFFF,D4
574Ø ØØØ65Ø 5384         DLYQ1 SUB.L #1,D4
575Ø ØØØ652 66FC          BNE DLYQ1
576Ø ØØØ654 4E75          RTS
577Ø ØØØ656 428Ø         CMQ CLR.L DØ
578Ø ØØØ658 323C2ØØØ      MOVE #$2ØØØ,D1
579Ø ØØØ65C 2Ø7CØØØ1ØØØØ  MOVE.L #$1ØØØØ,AØ
58ØØ ØØØ662 2ØCØ         CMQ1 MOVE.L DØ,(AØ)+
   Ø ØØØ664 5341          SUB  #1,D1
   Ø ØØØ666 66FA          BNE  CMQ1
582Ø ØØØ668 4E75          RTS
583Ø ØØØ66A 48E7FFFE     LOGO MOVEM.L DØ-D7/AØ-A6,-(A7)
584Ø ØØØ66E 4EB9ØØØ21F18  JSR FIXBUF
585Ø ØØØ674 2CFC53482Ø53  MOVE.L #'SH S',(A6)+
586Ø ØØØ67A 2CFC4C494445  MOVE.L #'LIDE',(A6)+
587Ø ØØØ68Ø 1CBCØØ2Ø      MOVE.B #' ',(A6)
588Ø ØØØ684 61ØØFA62      BSR SHQ
589Ø ØØØ688 61CØ          BSR DLYQ
59ØØ ØØØ68A 4EB9ØØØ21F18  JSR FIXBUF
591Ø ØØØ69Ø 2CFC5348204D  MOVE.L #'SH M',(A6)+
592Ø ØØØ696 2CFC41534B2Ø  MOVE.L #'ASK ',(A6)+
593Ø ØØØ69C 61ØØFA4A     BSR SHQ
594Ø ØØØ6AØ 4CDF7FFF     MOVEM.L (A7)+,DØ-D7/AØ-A6
595Ø ØØØ6A4 283CØØ1ØFFFF MOVE.L #$ØØ1ØFFFF,D4
596Ø ØØØ6AA 6ØA4         BRA DLYQ1
597Ø                    *
598Ø ØØØ6AC 61ØØØØØ6    HP BSR HP1
599Ø ØØØ6BØ 6ØØØF9CA     BRA RETURN
6ØØØ ØØØ6B4 267CØØØØ1Ø8Ø HP1 MOVE.L #ARRAY,A3
6Ø1Ø ØØØ6BA 619A         BSR CMQ
6Ø2Ø ØØØ6BC 4241         CLR D1
6Ø3Ø ØØØ6BE 4242         CLR D2
6Ø4Ø ØØØ6CØ 363CØØFF     MOVE #$FF,D3
6Ø5Ø ØØØ6C4 38Ø3         MOVE D3,D4
6Ø6Ø ØØØ6C6 61ØØFE6A     BSR RAND1
6Ø7Ø ØØØ6CA Ø2ØØØØØ7     AND.B #7,DØ
6Ø8Ø ØØØ6CE 5AØØ         ADD.B #5,DØ
6Ø9Ø ØØØ6DØ E34Ø         ASL 1,DØ
61ØØ ØØØ6D2 11CØ1Ø14     MOVE.B DØ,NUMPT
611Ø ØØØ6D6 61ØØFE5A     BSR RAND1
612Ø ØØØ6DA Ø2ØØØØ1F     AND.B #$1F,DØ
613Ø ØØØ6DE ØØØØØØØ5     OR.B #$5,DØ
614Ø ØØØ6E2 11CØ1Ø16     MOVE.B DØ,SCALE
615Ø ØØØ6E6 4245         CLR D5
616Ø ØØØ6E8 61ØØFE48    H6 BSR RAND1
617Ø ØØØ6EC Ø24ØØØFF     AND #$FF,DØ
618Ø ØØØ6FØ 178Ø5ØØØ     MOVE.B DØ,Ø(A3,D5)
619Ø ØØØ6F4 B24Ø         CMP DØ,D1
62ØØ ØØØ6F6 6AØ2         BPL.S H1
621Ø ØØØ6F8 12ØØ         MOVE.B DØ,D1
622Ø ØØØ6FA B64Ø        H1 CMP DØ,D3
623Ø ØØØ6FC 6BØ2         BMI.S H2
624Ø ØØØ6FE 16ØØ         MOVE.B DØ,D3
625Ø ØØØ7ØØ 61ØØFE3Ø    H2 BSR RAND1
626Ø ØØØ7Ø4 Ø24ØØØFF     AND #$FF,DØ
627Ø ØØØ7Ø8 178Ø5ØØ1     MOVE.B DØ,1(A3,D5)
628Ø ØØØ7ØC B44Ø         CMP DØ,D2
629Ø ØØØ7ØE 6AØ2         BPL.S H3
63ØØ ØØØ71Ø 14ØØ         MOVE.B DØ,D2
631Ø ØØØ712 B84Ø        H3 CMP DØ,D4
632Ø ØØØ714 6BØ2         BMI.S H4
633Ø ØØØ716 18ØØ         MOVE.B DØ,D4
634Ø ØØØ718 BA381Ø14    H4 CMP.B NUMPT,D5
635Ø ØØØ71C 67Ø4         BEQ.S H5
636Ø ØØØ71E 54Ø5         ADD.B #2,D5
637Ø ØØØ72Ø 6ØC6         BRA H6
638Ø          ØØØØØ722  H5 EQU *
639Ø ØØØ722 92Ø3        H8 SUB.B D3,D1
64ØØ ØØØ724 94Ø4         SUB.B D4,D2
641Ø ØØØ726 4245         CLR D5
642Ø ØØØ728 97335ØØØ    H61 SUB.B D3,Ø(A3,D5)
643Ø ØØØ72C 99335ØØ1    SUB.B D4,1(A3,D5)
644Ø ØØØ73Ø BA381Ø14     CMP.B NUMPT,D5
645Ø ØØØ734 67Ø4         BEQ.S H9
646Ø ØØØ736 54Ø5         ADD.B #2,D5
647Ø ØØØ738 6ØEE         BRA H61
```

```
648Ø ØØØ73A 4243            H9 CLR D3
649Ø ØØØ73C 2Ø3CØØØØFFØØ      MOVE.L #$FFØØ,DØ
65ØØ ØØØ742 Ø241ØØFF          AND #$FF,D1
651Ø ØØØ746 8ØC1              DIVU D1,DØ
652Ø ØØØ748 4245              CLR D5
653Ø ØØØ74A 16335ØØØ         H12 MOVE.B Ø(A3,D5),D3
654Ø ØØØ74E C6CØ              MULU DØ,D3
655Ø ØØØ75Ø EØ4B              LSR 8,D3
656Ø ØØØ752 17835ØØØ          MOVE.B D3,Ø(A3,D5)
657Ø ØØØ756 BA381Ø14          CMP.B NUMPT,D5
658Ø ØØØ75A 67Ø4              BEQ.S H11
659Ø ØØØ75C 54Ø5              ADD.B #2,D5
66ØØ ØØØ75E 6ØEA              BRA H12
661Ø ØØØ76Ø 2Ø3CØØØØFFØØ     H11 MOVE.L #$FFØØ,DØ
662Ø ØØØ766 Ø242ØØFF          AND #$FF,D2
663Ø ØØØ76A 8ØC2              DIVU D2,DØ
664Ø ØØØ76C 4245              CLR D5
665Ø ØØØ76E 16335ØØ1         H14 MOVE.B 1(A3,D5),D3
666Ø ØØØ772 C6CØ              MULU DØ,D3
667Ø ØØØ774 EØ4B              LSR 8,D3
668Ø ØØØ776 17835ØØ1          MOVE.B D3,1(A3,D5)
669Ø ØØØ77A BA381Ø14          CMP.B NUMPT,D5
67ØØ ØØØ77E 67Ø4              BEQ.S H13
671Ø ØØØ78Ø 54Ø5              ADD.B #2,D5
672Ø ØØØ782 6ØEA              BRA H14
673Ø ØØØ784 31D31ØØØ         H13 MOVE (A3),X1
674Ø ØØØ788 3E3CØØ1C         H131 MOVE #$1C,D7
675Ø ØØØ78C 54381Ø14         H132 ADD.B #2,NUMPT
676Ø ØØØ79Ø 1A381Ø14          MOVE.B NUMPT,D5
677Ø ØØØ794 37935ØØØ          MOVE (A3),Ø(A3,D5)
678Ø ØØØ798 3C3CØØØ4         H15 MOVE #4,D6
679Ø ØØØ79C 61ØØFD94          BSR RAND1
68ØØ ØØØ7AØ Ø24ØØØØF          AND #$F,DØ
681Ø ØØØ7A4 67F2              BEQ H15
682Ø ØØØ7A6 ØCØØØØØ8          CMP.B #$8,DØ
683Ø ØØØ7AA 67EC              BEQ H15
684Ø ØØØ7AC ØCØØØØØF          CMP.B #$F,DØ
685Ø ØØØ7BØ 67E6              BEQ H15
686Ø ØØØ7B2 4245             HP6 CLR D5
687Ø ØØØ7B4 12335ØØØ         H17 MOVE.B Ø(A3,D5),D1
688Ø ØØØ7B8 14335ØØ1          MOVE.B 1(A3,D5),D2
689Ø ØØØ7BC 61ØØØØ8A         HP17 BSR LINE
69ØØ ØØØ7CØ BA381Ø14          CMP.B NUMPT,D5
691Ø ØØØ7C4 6748              BEQ.S H16
692Ø ØØØ7C6 12335ØØ2          MOVE.B 2(A3,D5),D1
693Ø ØØØ7CA 14335ØØØ          MOVE.B Ø(A3,D5),D2
694Ø ØØØ7CE Ø241ØØFF          AND #$FF,D1
695Ø ØØØ7D2 Ø242ØØFF          AND #$FF,D2
696Ø ØØØ7D6 9242              SUB D2,D1
697Ø ØØØ7D8 16381Ø16          MOVE.B SCALE,D3
698Ø ØØØ7DC Ø243ØØFF          AND #$FF,D3
699Ø ØØØ7EØ C3C3              MULS D3,D1
7ØØØ ØØØ7E2 EØ49              LSR 8,D1
7Ø1Ø ØØØ7E4 D3335ØØØ          ADD.B D1,Ø(A3,D5)
7Ø2Ø ØØØ7E8 12335ØØ3          MOVE.B 3(A3,D5),D1
```

```
7Ø3Ø ØØØ7EC Ø241ØØFF      AND #$FF,D1
7Ø4Ø ØØØ7FØ 14335ØØ1      MOVE.B 1(A3,D5),D2
7Ø5Ø ØØØ7F4 Ø242ØØFF      AND #$FF,D2
7Ø6Ø ØØØ7F8 9242          SUB D2,D1
7Ø7Ø ØØØ7FA 16381Ø16      MOVE.B SCALE,D3
7Ø8Ø ØØØ7FE Ø243ØØFF      AND #$FF,D3
7Ø9Ø ØØØ8Ø2 C3C3          MULS D3,D1
71ØØ ØØØ8Ø4 EØ49          LSR 8,D1
711Ø ØØØ8Ø6 D3335ØØ1      ADD.B D1,1(A3,D5)
712Ø ØØØ8ØA 5445          ADD #2,D5
713Ø ØØØ8ØC 6ØA6          BRA H17
714Ø ØØØ8ØE 5346         H16 SUB #1,D6
715Ø ØØØ81Ø 66AØ          BNE HP6
716Ø ØØØ812 5347          SUB #1,D7
717Ø ØØØ814 6682          BNE H15
718Ø ØØØ816 4E75          RTS
719Ø ØØØ818 61ØØFE9A     Q8 BSR HP1
72ØØ ØØØ81C 283CØØØAFFFF  MOVE.L #$AFFFF,D4
721Ø ØØØ822 61ØØFE2C      BSR DLYQ1
722Ø ØØØ826 6ØFØ          BRA Q8
723Ø                      *
724Ø                      *
725Ø                      *
726Ø ØØØ828 1229ØØØ2     DXDY MOVE.B 2(A1),D1
727Ø ØØØ82C 9211          SUB.B (A1),D1
728Ø ØØØ82E 65ØA          BCS.S XNEG
729Ø ØØØ83Ø 1341ØØØ4      MOVE.B D1,4(A1)
73ØØ ØØØ834 4229ØØØ6      CLR.B 6(A1)
731Ø ØØØ838 4E75          RTS
732Ø ØØØ83A 137CØØØ1ØØØ6 XNEG MOVE.B #1,6(A1)
733Ø ØØØ84Ø 44Ø1          NEG.B D1
734Ø ØØØ842 1341ØØØ4      MOVE.B D1,4(A1)
735Ø ØØØ846 4E75          RTS
736Ø                      *
737Ø                      *
738Ø        ØØØØØ848     LINE EQU *
739Ø ØØØ848 48E7FFFE     DRAW MOVEM.L DØ-D7/AØ-A6,-(A7)
74ØØ ØØØ84C 227CØØØØ1ØØØ  MOVE.L #X1,A1
741Ø ØØØ852 1341ØØØ2      MOVE.B D1,2(A1)
742Ø ØØØ856 1342ØØØ3      MOVE.B D2,3(A1)
743Ø ØØØ85A 1211          MOVE.B (A1),D1
744Ø ØØØ85C 1429ØØØ1      MOVE.B 1(A1),D2
745Ø ØØØ86Ø 61ØØFCF2      BSR DSP
746Ø ØØØ864 61C2         DRAW1 BSR DXDY
747Ø ØØØ866 5289          ADD.L #1,A1
748Ø ØØØ868 61BE          BSR DXDY
749Ø ØØØ86A 5389          SUB.L #1,A1
75ØØ ØØØ86C 1211          MOVE.B (A1),D1
751Ø ØØØ86E 1429ØØØ1      MOVE.B 1(A1),D2
752Ø ØØØ872 4A29ØØØ4      TST.B 4(A1)
753Ø ØØØ876 6766          BEQ.S DXZ
754Ø ØØØ878 4A29ØØØ5      TST.B 5(A1)
755Ø ØØØ87C 67ØØØØ88      BEQ DYZ
756Ø ØØØ88Ø 1629ØØØ4      MOVE.B 4(A1),D3
757Ø ØØØ884 B629ØØØ5      CMP.B 5(A1),D3
```

```
758Ø ØØØ888 66ØØØØBØ        BNE FULMOV
759Ø ØØØ88C 4A29ØØØ6        TST.B 6(A1)
76ØØ ØØØ89Ø 6626            BNE.S SXN
761Ø ØØØ892 4A29ØØØ7        TST.B 7(A1)
762Ø ØØØ896 6636            BNE.S SYN
763Ø ØØØ898 61ØØFCBA      XPYP1 BSR DSP
764Ø ØØØ89C 52Ø1            ADD.B #1,D1
765Ø ØØØ89E 52Ø2            ADD.B #1,D2
766Ø ØØØ8AØ B229ØØØ2        CMP.B 2(A1),D1
767Ø ØØØ8A4 66F2            BNE XPYP1
768Ø ØØØ8A6 6Ø7E            BRA.S XYDONE
769Ø ØØØ8A8 61ØØFCAA      SXNSYN BSR DSP
77ØØ ØØØ8AC 53Ø1            SUB.B #1,D1
771Ø ØØØ8AE 53Ø2            SUB.B #1,D2
772Ø ØØØ8BØ B229ØØØ2        CMP.B 2(A1),D1
773Ø ØØØ8B4 66F2            BNE SXNSYN
774Ø ØØØ8B6 6Ø6E            BRA.S XYDONE
775Ø ØØØ8B8 4A29ØØØ7      SXN TST.B 7(A1)
776Ø ØØØ8BC 66EA            BNE.S SXNSYN
777Ø ØØØ8BE 61ØØFC94      SNP BSR DSP
778Ø ØØØ8C2 53Ø1            SUB.B #1,D1
779Ø ØØØ8C4 52Ø2            ADD.B #1,D2
78ØØ ØØØ8C6 B229ØØØ2        CMP.B 2(A1),D1
781Ø ØØØ8CA 66F2            BNE SNP
782Ø ØØØ8CC 6Ø58            BRA.S XYDONE
783Ø ØØØ8CE 61ØØFC84      SYN BSR DSP
784Ø ØØØ8D2 52Ø1            ADD.B #1,D1
785Ø ØØØ8D4 53Ø2            SUB.B #1,D2
786Ø ØØØ8D6 B229ØØØ2        CMP.B 2(A1),D1
787Ø ØØØ8DA 66F2            BNE SYN
788Ø ØØØ8DC 6Ø48            BRA.S XYDONE
789Ø ØØØ8DE 4A29ØØØ5      DXZ TST.B 5(A1)
79ØØ ØØØ8E2 6742            BEQ.S XYDONE
791Ø ØØØ8E4 4A29ØØØ7        TST.B 7(A1)
792Ø ØØØ8E8 66ØE            BNE.S DXZYN
793Ø ØØØ8EA 61ØØFC68      DXZ1 BSR DSP
794Ø ØØØ8EE 52Ø2            ADD.B #1,D2
795Ø ØØØ8FØ B429ØØØ3        CMP.B 3(A1),D2
796Ø ØØØ8F4 66F4            BNE DXZ1
797Ø ØØØ8F6 6Ø2E            BRA.S XYDONE
798Ø ØØØ8F8 61ØØFC5A      DXZYN BSR DSP
799Ø ØØØ8FC 53Ø2            SUB.B #1,D2
8ØØØ ØØØ8FE B429ØØØ3        CMP.B 3(A1),D2
8Ø1Ø ØØØ9Ø2 66F4            BNE DXZYN
8Ø2Ø ØØØ9Ø4 6Ø2Ø            BRA.S XYDONE
8Ø3Ø ØØØ9Ø6 4A29ØØØ6      DYZ TST.B 6(A1)
8Ø4Ø ØØØ9ØA 66ØE            BNE.S DYZN
8Ø5Ø ØØØ9ØC 61ØØFC46      DYZ1 BSR DSP
8Ø6Ø ØØØ91Ø 52Ø1            ADD.B #1,D1
8Ø7Ø ØØØ912 B229ØØØ2        CMP.B 2(A1),D1
8Ø8Ø ØØØ916 66F4            BNE DYZ1
8Ø9Ø ØØØ918 6ØØC            BRA.S XYDONE
81ØØ ØØØ91A 61ØØFC38      DYZN BSR DSP
811Ø ØØØ91E 53Ø1            SUB.B #1,D1
812Ø ØØØ92Ø B229ØØØ2        CMP.B 2(A1),D1
```

```
ØØØ924 66F4           BNE DYZN
ØØØ926 32A9ØØØ2      XYDONE MOVE 2(A1),(A1)
ØØØ92A 1211           MOVE.B (A1),D1
ØØØ92C 1429ØØØ1       MOVE.B 1(A1),D2
ØØØ93Ø 61ØØFC22       BSR DSP
ØØØ934 4CDF7FFF       MOVEM.L (A7)+,DØ-D7/AØ-A6
ØØØ938 4E75           RTS
ØØØ93A 3351ØØØ8      FULMOV MOVE (A1),8(A1)
ØØØ93E 1629ØØØ4       MOVE.B 4(A1),D3
ØØØ942 9629ØØØ5       SUB.B 5(A1),D3
ØØØ946 62Ø8           BHI.S FUL1
ØØØ948 337CØØØ1ØØØA   MOVE #$1,1Ø(A1)
ØØØ94E 6Ø46           BRA.S FUL4
ØØØ95Ø 337CØ1ØØØØØA  FUL1 MOVE #$1ØØ,1Ø(A1)
ØØØ956 6Ø3E           BRA.S FUL4
ØØØ958 1629ØØØ8      FUL2 MOVE.B 8(A1),D3
ØØØ95C 9611           SUB.B (A1),D3
ØØØ95E 64Ø2           BCC.S FUL21
ØØØ96Ø 44Ø3           NEG.B D3
ØØØ962 Ø243ØØFF      FUL21 AND #$FF,D3
ØØØ966 1829ØØØ5       MOVE.B 5(A1),D4
ØØØ96A Ø244ØØFF       AND #$FF,D4
ØØØ96E C6C4           MULU D4,D3
ØØØ97Ø 1829ØØØ9       MOVE.B 9(A1),D4
ØØØ974 9829ØØØ1       SUB.B 1(A1),D4
ØØØ978 64Ø2           BCC.S FUL22
ØØØ97A 44Ø4           NEG.B D4
ØØØ97C 1A29ØØØ4      FUL22 MOVE.B 4(A1),D5
ØØØ98Ø Ø244ØØFF       AND #$FF,D4
ØØØ984 Ø245ØØFF       AND #$FF,D5
ØØØ988 C8C5           MULU D5,D4
ØØØ98A 4A29ØØØA       TST.B 1Ø(A1)
ØØØ98E 66ØE           BNE.S FULY
ØØØ99Ø B883           CMP.L D3,D4
ØØØ992 671Ø           BEQ.S GREAT
ØØØ994 62ØE           BHI.S GREAT
ØØØ996 3369ØØØAØØØE  FUL4 MOVE 1Ø(A1),14(A1)
ØØØ99C 6ØØC           BRA.S SAME
ØØØ99E B883          FULY CMP.L D3,D4
ØØØ9AØ 67Ø2           BEQ.S GREAT
ØØØ9A2 62F2           BHI.S FUL4
ØØØ9A4 337CØ1Ø1ØØØE  GREAT MOVE #$Ø1Ø1,14(A1)
ØØØ9AA 1229ØØØ8      SAME MOVE.B 8(A1),D1
ØØØ9AE 1429ØØØ9       MOVE.B 9(A1),D2
ØØØ9B2 4A29ØØØ7       TST.B 7(A1)
ØØØ9B6 66Ø6           BNE.S NEGY
ØØØ9B8 D429ØØØF       ADD.B 15(A1),D2
ØØØ9BC 6ØØ4           BRA.S S2
ØØØ9BE 9429ØØØF      NEGY SUB.B 15(A1),D2
ØØØ9C2 1342ØØØ9      S2 MOVE.B D2,9(A1)
ØØØ9C6 4A29ØØØ6       TST.B 6(A1)
ØØØ9CA 66Ø6           BNE.S NEGX
ØØØ9CC D229ØØØE       ADD.B 14(A1),D1
ØØØ9DØ 6ØØ4           BRA.S S3
ØØØ9D2 9229ØØØE      NEGX SUB.B 14(A1),D1
```

```
868Ø ØØØ9D6 1341ØØØ8       S3 MOVE.B D1,8(A1)
869Ø ØØØ9DA 61ØØFB78      FUL3 BSR DSP
87ØØ ØØØ9DE B229ØØØ2       CMP.B 2(A1),D1
871Ø ØØØ9E2 67ØA           BEQ.S DRAW2
872Ø ØØØ9E4 B429ØØØ3       CMP.B 3(A1),D2
873Ø ØØØ9E8 67Ø4           BEQ.S DRAW2
874Ø ØØØ9EA 6ØØØFF6C       BRA FUL2
875Ø ØØØ9EE 32A9ØØØ8      DRAW2 MOVE 8(A1),(A1)
876Ø ØØØ9F2 6ØØØFE7Ø       BRA DRAW1
877Ø ØØØ9F6 ØØØØ           END

 ****** TOTAL ERRORS   Ø--   Ø


 SYMBOL TABLE

 ARRAY      ØØ1Ø8Ø BA         ØØØ454 BL1        ØØØ364 BL2        ØØØ358
 BL3        ØØØ368 BLACK      ØØØ3Ø8 BLINK      ØØØ34A BLUE       ØØØ2FC
 BX         ØØØ18A BX1        ØØØ19A BX11       ØØØ1B4 BX2        ØØØ1A4
 BX22       ØØØ1CØ BX3        ØØØ19Ø BX33       ØØØ1DØ BX44       ØØØ1DA
 CHAR       ØØØ3B6 CHARED     ØØØ3AØ CHARED1    ØØØ3CE CHARED2    ØØØ3DØ
 CHARED3    ØØØ3D4 CHARED4    ØØØ3F8 CHTAB      ØØØ9F6 CLRM       ØØØØ74
 CM         ØØØØ68 CMD        ØØØ2Ø4 CMD1       ØØØ26A CMD2       ØØØ288
 CMD3       ØØØ27Ø CMD4       ØØØ276 CMDTAB     ØØ18ØØ CMQ        ØØØ656
 CMQ1       ØØØ662 COLOR      ØØ1Ø1Ø CR         ØØØ27C CRTC       ØØØØD2
 CYAN       ØØØ31A DBLUE      ØØØ32C DCYAN      ØØØ344 DGR        ØØØ326
 DLY        ØØØ44A DLY1       ØØØ44E DLYQ       ØØØ64A DLYQ1      ØØØ65Ø
 DMAG       ØØØ33E DOT        ØØØ41C DRAW       ØØØ848 DRAW1      ØØØ864
 DRAW2      ØØØ9EE DRED       ØØØ32Ø DSP        ØØØ554 DSP1       ØØØ566
 DSPLY      ØØØ55A DSPLY1     ØØØ5A8 DSPLY2     ØØØ5AØ DWARROW    ØØØ258
 DWH        ØØØ332 DXDY       ØØØ828 DXZ        ØØØ8DE DXZ1       ØØØ8EA
 DXZYN      ØØØ8F8 DYEL       ØØØ338 DYZ        ØØØ9Ø6 DYZ1       ØØØ9ØC
 DYZN       ØØØ91A ED         ØØØ1E8 ED1        ØØØ1FA EQU1       ØØØ5AC
 EQU2       ØØØ5BE EQU3       ØØØ5DØ EQU4       ØØØ5E2 EQU5       ØØØ5F4
 FIXBUF     Ø21F18 FUL1       ØØØ95Ø FUL2       ØØØ958 FUL21      ØØØ962
 FUL22      ØØØ97C FUL3       ØØØ9DA FUL4       ØØØ996 FULMOV     ØØØ93A
 FULY       ØØØ99E GETADD     ØØØ57A GREAT      ØØØ9A4 GREEN      ØØØ2F6
 H1         ØØØ6FA H11        ØØØ76Ø H12        ØØØ74A H13        ØØØ784
 H131       ØØØ788 H132       ØØØ78C H14        ØØØ76E H15        ØØØ798
 H16        ØØØ8ØE H17        ØØØ7B4 H2         ØØØ7ØØ H3         ØØØ712
 H4         ØØØ718 H5         ØØØ722 H6         ØØØ6E8 H61        ØØØ728
 H8         ØØØ722 H9         ØØØ73A HP         ØØØ6AC HP1        ØØØ6B4
 HP17       ØØØ7BC HP6        ØØØ7B2 INIT       ØØØØ2C INIT1      ØØØØ38
 INPUT      ØØØ16E L1         ØØØ466 L2         ØØØ482 LINE       ØØØ848
 LOGO       ØØØ66A LOOP       ØØØ474 LTARROW    ØØØ264 MACSBUG    Ø2ØØF6
 MAG        ØØØ314 MSG        Ø2ØØEE NCOLOR     ØØ1Ø11 NEGX       ØØØ9D2
 NEGY       ØØØ9BE NTABLE     ØØØØ82 NUMPT      ØØ1Ø14 OCOLOR     ØØ1Ø12
 OUTPUT     ØØØ17C OUTPUT2    Ø21BC2 Q1         ØØØ498 Q2         ØØØ4A4
 Q3         ØØØ4BØ Q4         ØØØ4BC Q5         ØØØ4C8 Q8         ØØØ818
 Q9         ØØØ6Ø6 Q91        ØØØ6ØC Q92        ØØØ61Ø RANADD     ØØ1Ø18
 RAND       ØØØ514 RAND1      ØØØ532 RAND2      ØØØ52Ø READK      ØØØ436
 RED        ØØØ2FØ RETURN     ØØØØ7C RTARROW    ØØØ25E RTS        ØØØ256
 RTS1       ØØØ254 RTS2       ØØØ2EC RUN        ØØØ4D2 RUN1       ØØØ4DA
 RUN2       ØØØ4E2 S2         ØØØ9C2 S3         ØØØ9D6 SAME       ØØØ9AA
 SCALE      ØØ1Ø16 SET        ØØØ4ØA SETUP      ØØØØØØ SETUP1     ØØØØØE
```

```
SETUP4.S  MC68ØØØ ASM REV= 1.ØF- COPYRIGHT BY MOTOROLA 1978    PAGE 18

SETUP2      ØØØØ1C SETUP21     ØØØØ6Ø SH          ØØØØE2 SH1         ØØØØF4
SH2         ØØØ1Ø6 SH3         ØØØ146 SH4         ØØØ154 SHOW        ØØØ1B2
SHQ         ØØØØE8 SNP         ØØØ8BE SXN         ØØØ8B8 SXNSYN      ØØØ8A8
SYN         ØØØ8CE TABLECH     ØØ11ØØ UPARROW     ØØØ25Ø WHITE       ØØØ3Ø2
X1          ØØ1ØØØ X2          ØØ1ØØ2 XNEG        ØØØ83A XPYP1       ØØØ898
XYDONE      ØØØ926 Y1          ØØ1ØØ1 Y2          ØØ1ØØ3 YELLOW      ØØØ3ØE
```

AN-836

USING LOW-COST 1 MHz PERIPHERALS IN A 2 MHz SYSTEM WITH THE MC68B09 AND THE MC68B09E

by

Duane Graden and Hunter Scales
NMOS Microcomputer Applications
Motorola Inc.

INTRODUCTION

With the increasing use of HMOS design techniques in VLSI circuits, the maximum speed of these devices is also on the rise. There are 2 MHz, "B," versions of the popular MC6809 and the new MC6809E with external clock. Both the MC68B09 and the MC68B09E feature a 500 nanosecond cycle time. With a 2 MHz E clock, an add immediate instruction takes just 1 microsecond! These fast, efficient processors offer designers the opportunity to use a microprocessor in applications which have been, until now, too slow.

It would appear that the speed increase necessarily carries with it a cost penalty. That is, by increasing the speed of the bus, faster and therefore more expensive memories and peripherals must be used. However, there are ways to manipulate the 2 MHz MPU access time to accommodate slower peripherals and memories.

MPU ACCESS TIME MANIPULATION

The system clocks on the MC6809 can be delayed (stretched) to allow longer access time for slow memories using the MRDY input pin. Figure 1 shows the timing for this input. The system E and Q clocks are stretched, while E is high and Q is low, in one-quarter bus cycle increments. One quarter cycle of the MC68B09 2 MHz clock is equal to 125 nanoseconds. Since the MC6809E requires an external clock generator, the MRDY signal can be implemented externally for that processor.

A problem arises when stretching the access time for slow memories in that the throughput of the 2 MHz system is reduced markedly because the majority of processor cycles are, in fact, memory accesses. One solution to this problem is a compromise: absorb the cost of fast memories to allow the processor to run all memory cycles at full speed but reduce the speed of the bus for peripheral access. Since many peripherals are accessed only infrequently, this approach incurs minor impact on total throughput.

Unfortunately, slowing the bus cycle to accommodate slow peripherals is not as simple as using slow memories. To begin with, all MC6809 family peripherals require a continuous system clock to function. If the peripherals are specified at 1 MHz, this clock cannot exceed 1 MHz. This requires a separate, 1 MHz peripheral clock. This clock may not be synchronous with the main 2 MHz processor clock. Therefore, the chip enable signals to the peripherals must be delayed until the peripheral clock is low and then meet the chip select ($\overline{CS}$) setup time. In 1 MHz chips, chip select time is 160 nanoseconds before the rising edge of the clock. Some circuits, designed to allow the use of an MC6809 peripheral chip operating at one-half the frequency of the 2 MHz system clock, are described in the following paragraphs.

USING THE MC68B09 WITH 1 MHz PERIPHERALS

The circuit shown in Figure 2 allows 1 MHz peripherals to run with a 2 MHz MC68B09 system by generating an asynchronous peripheral clock (PCLK). When an access of any 1 MHz peripheral takes place, the 2 MHz system clocks, E and Q, are stretched using the MRDY pin. A state machine then waits until PCLK is low and then chip selects the peripheral 250 nanoseconds before the rising edge of PCLK. This provides proper address setup time at the peripherals before chip selecting them. Clocks E and Q are then released and the data is latched.

Refer to the timing diagram in Figure 3 and note the signal relationships during write and read cycles. Initiation of a

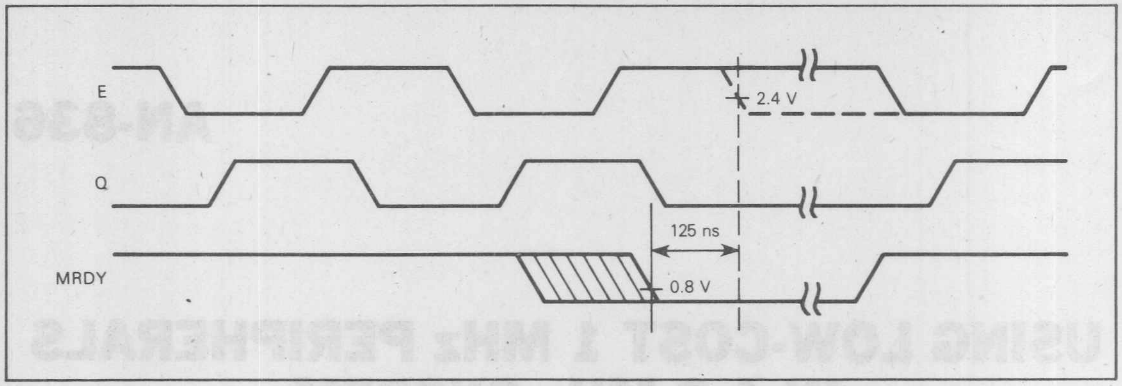


Figure 1. MC6809 MRDY Timing

peripheral access cycle causes the address decoding logic to bring the PERAC signal high ①. Signal MRDY is then brought low ②. While MRDY is low, the high E and low Q clocks are inhibited from switching states. Allowing for the address setup time, the peripheral enable (PE) signal is made true in the center of the PCLK low cycle ③.

Signal MRDY is then raised 375 nanoseconds after the rising edge of PCLK allowing E to fall 125 nanoseconds later ④. In addition, if the access is a read, PCLK is stretched 125 nanoseconds past the fall of E ⑤ to ensure valid data from the peripheral. The delay also allows for some inherent skew in the processor MRDY to E falling time. If the access is a write to the peripheral, then E is allowed to fall after PCLK to ensure that the peripheral clocks in valid data.

Note on Figure 3 that peripheral access (PERAC) may become active high either during the low or high cycle of PCLK. If the access occurs during the first quarter of the PCLK cycle, the E need only be stretched for one 2 MHz bus cycle (250 nanoseconds) until PCLK falls. However, if the access occurs during the last three-quarters of the PCLK cycle, then the stretch has to be continued until the next full cycle of PCLK occurs. The best case example, shown in Figure 3, is a short write, where the write PERAC signal occurs before the end of the first quarter cycle of PCLK. The worst case is a long read, where the read PERAC signal occurs during PCLK high.

MC6809E CLOCK CIRCUIT

Unlike the MC6809, the MC6809E requires an external clock generator to provide the 2 MHz E and Q system clocks. Figure 4 shows a circuit that can generate these clocks. The circuit also generates an internal PCLK signal used to develop the chip select ($\overline{\text{CS}}$) output to the peripherals. This output ensures that an addressed peripheral is accessed at the proper time. Two inputs are required by the circuit. These are the MRDY and the PERAC inputs. Input MRDY is used to stretch the E and Q clocks during slow (1 MHz) peripheral access cycles. Input PERAC is used to signal that a slow peripheral access cycle is active.

An 8 MHz oscillator, formed by crystal Y1, related 74LS04 U4 inverters, and resistors R1 and R2, provides the reference frequency. Two 8 MHz outputs, $\overline{4\text{X}}$ and 4X are obtained from the oscillator. Output $\overline{4\text{X}}$ is used to clock binary counter U7. This counter divides the $\overline{4\text{X}}$ clock by 2, 4, and 8 to derive the 4 MHz 2X clock, the 2 MHz 1X clock, and the 1 MHz PCLK, respectively. Clock 2X is routed to flip-flop U3a, where it is divided by 2 to obtain the 2 MHz Q clock output. Clock 1X is routed via gate U6a to provide the 2 MHz E clock output.

At power-up, flip-flop U3a and U3b are used to establish the correct clock E and clock Q phase relationship. However, this synchronization must be delayed until the oscillator has stabilized. An RC network (C1, R3) provides this delay. At power-up, a momentary low level from the RC network clears U3b to hold the U3b output low. In turn, the low U3b output presets U3a to hold the system Q clock output high. Approximately 50 milliseconds after power-up, the RC network output reaches V_T (1.9 V) of inverter U9 and releases U3b so it can be toggled by clock E. As the D input of U3b is tied to +5 V, the next rising edge of clock E toggles the U3b output from low to high. This action releases the U3a preset input so that U3a can be toggled by clock 2X. The next rising edge of clock 2X, and all subsequent rising edges of 2X, switch the U3a system Q clock output to generate the Q waveform. The +5 V at the D input of U3b ensures that the U3b output remains high and does not interfere with U3a after synchronization. Since the system reset delay is 100 milliseconds, the 50 millisecond clock delay does not interfere in any way with system operation.

MC6809E SLOW MEMORY ACCESS

The MRDY input allows the processor to access slow memories by stretching E and Q. Refer to the timing diagram in Figure 5 and note that when MRDY is pulled low, E is high and Q is low. Refer to Figure 4 and note that these conditions generate a low output at gate U5c. Therefore, a low is present at the input of U2b. The next rising edge of the oscillator 4X clock toggles U2b and causes ENABLE to counter U7 to become inactive, holding E high. The Q clock is also held low by the low clear input to U3a from U2b. Thus, E and Q are stretched as long as MRDY is low. Removing MRDY causes the clocks to be released on the following 4X clock rising edge.

MC6809E SLOW PERIPHERAL ACCESS

Correct access of slow peripherals dictates synchronization between the processor clock, E, and peripheral clock, PCLK. In this case, PCLK is a continuous square wave one-half the frequency of E. Accesses of the slow peripheral are synchronized by stretching E, thus allowing them to operate at their full rated bus speed. Only the processor is slow and only for that cycle.

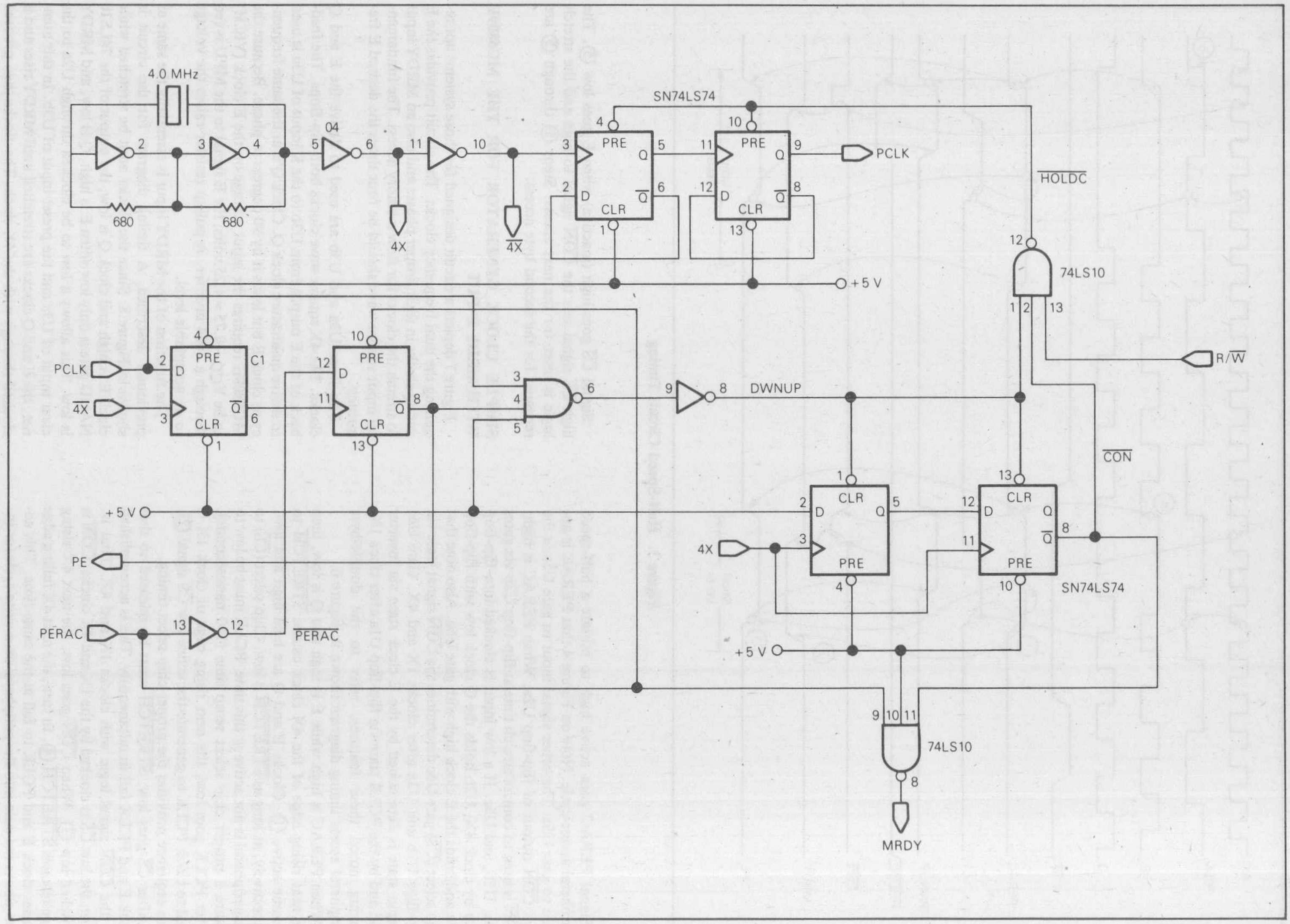


Figure 2. MC6809 Half-Speed Peripheral Clock Circuit

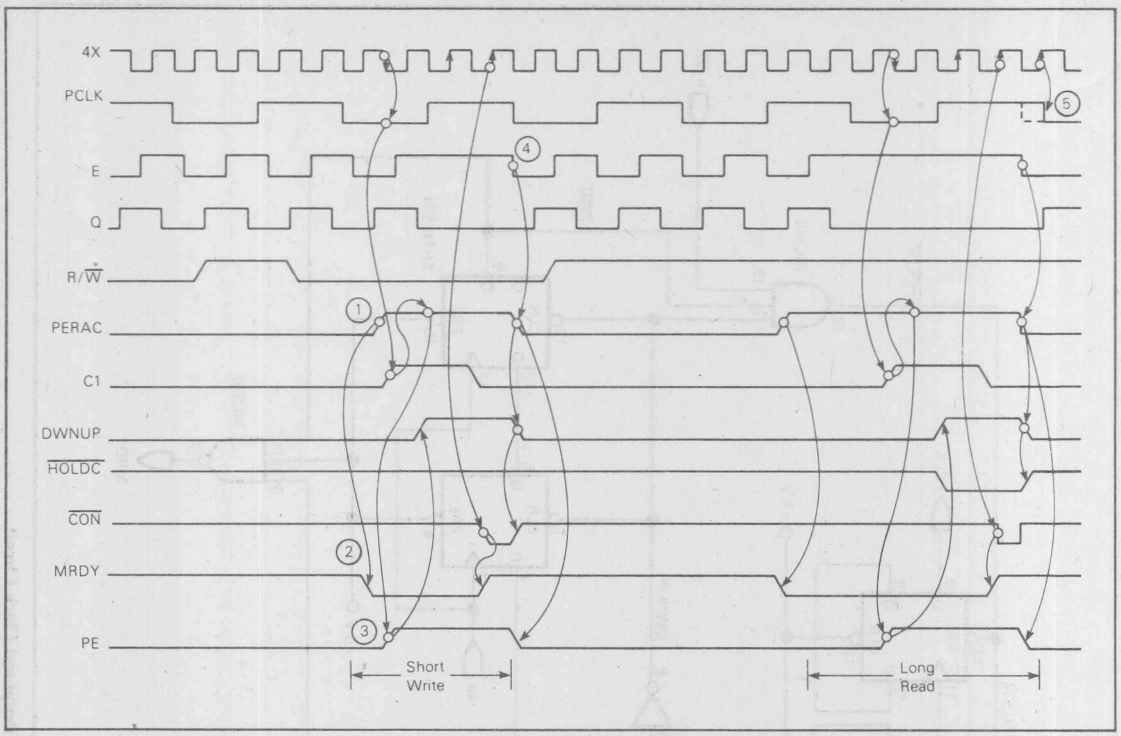


Figure 3. Half-Speed Circuit Timing

Signal PERAC goes active high to initiate a half-speed peripheral access cycle. Note on Figure 4 that PERAC is applied to gate U5a. The other signal input to gate U5a is the high $\overline{\text{CON}}$ output of flip-flop U2a. When PERAC is high, $\overline{\text{CON}}$ is used to control stretch time at flip-flop U2b via gates U5a, U5b, and U5c. If a low input is clocked into flip-flop U2b by clock 4X, U2b holds the Q clock low with flip-flop U3a and holds the E clock high with gate U6a. Also note that chip select ($\overline{\text{CS}}$) gate U6c determines the $\overline{\text{CON}}$ signal state via flip-flips U1b and U2a after clocks 1X and 4X. Gate U6c output state is determined by the E clock state via inverter U4d and by the PCLK state via flip-flop U1a after clock 1X. Having noted these features, refer to the double-byte peripheral access timing diagram shown in Figure 6.

When PERAC is high while E is high and Q is low, then the next rising edge of the 4X clock causes $\overline{\text{STRETCH}}$ to become active ①. Clocks E and Q are held high and low, respectively, as long as $\overline{\text{STRETCH}}$ is low. Chip select ($\overline{\text{CS}}$) to the peripheral is not active at this time. PCLK must be low to ensure a proper chip select setup time (160 nanoseconds). After PCLK goes low, the next rising edge of clock 1X is used to clock PCLK to generate the active low $\overline{\text{CS}}$ signal ②. This sequence provides the proper chip select timing.

After $\overline{\text{CS}}$ goes low, $\overline{\text{STRETCH}}$ must be released so that clock E and PCLK fall simultaneously. This is accomplished by the $\overline{\text{CON}}$ signal logic with clocks 1X and 4X. That is, when the low $\overline{\text{CS}}$ is clocked by the 1X and 4X clocks, $\overline{\text{CON}}$ is switched low ③. When $\overline{\text{CON}}$ goes low, the next 4X rising edge releases $\overline{\text{STRETCH}}$ ④. In turn, the next 4X falling edge causes clock E and PCLK to fall at the same time. This action clocks the data into the peripheral on a write cycle (or into the MPU on a read cycle).

Signal $\overline{\text{CS}}$ goes high (inactive) when E goes low ⑤. The high $\overline{\text{CS}}$ signal sets the $\overline{\text{CON}}$ signal to high and the stretch logic is reset for the next access. Steps ① through ⑤ are repeated for the second byte access.

SIMPLE CLOCK GENERATOR FOR THE MC6809E WITH MRDY INPUT

Figure 7 depicts a circuit designed for those systems not requiring the dual frequency clocks. The circuit provides the E and Q clocks in their proper phases and also an MRDY input to stretch the clocks for slow memory access. The fundamental input square wave should be four times the desired E frequency.

Flip-flops U3a and U3b are used to derive the E and Q clocks. The 4X square wave clocks both flip-flops. The feedback of the E output from U3b to the K input of U3a is used to derive quadrature clock Q. Clock Q is at the same frequency as clock E but leads it by 90 degrees in phase. Because the MC6809E requires the input voltage on the E clock (VICH) to be $V_{CC} - 0.75 = 4.25$ volts, the E signal to the MPU is fed through a 7404 inverter. A pullup resistor raises the voltage to an acceptable level.

The function of the MRDY input is essentially the same as previously described. A timing diagram for the circuit is shown in Figure 8. Since the clocks must be stretched while clock E is high and clock Q is low, the output of the 74LS10 NAND gate is only low when E is high, Q is low, and MRDY is low. This allows a low to be clocked through U2a to the clear input of U3a and the preset input of U3b. In this manner, the E and Q clocks are stretched until MRDY rises and is clocked through with the 4X clock. The clocks then proceed normally from there.

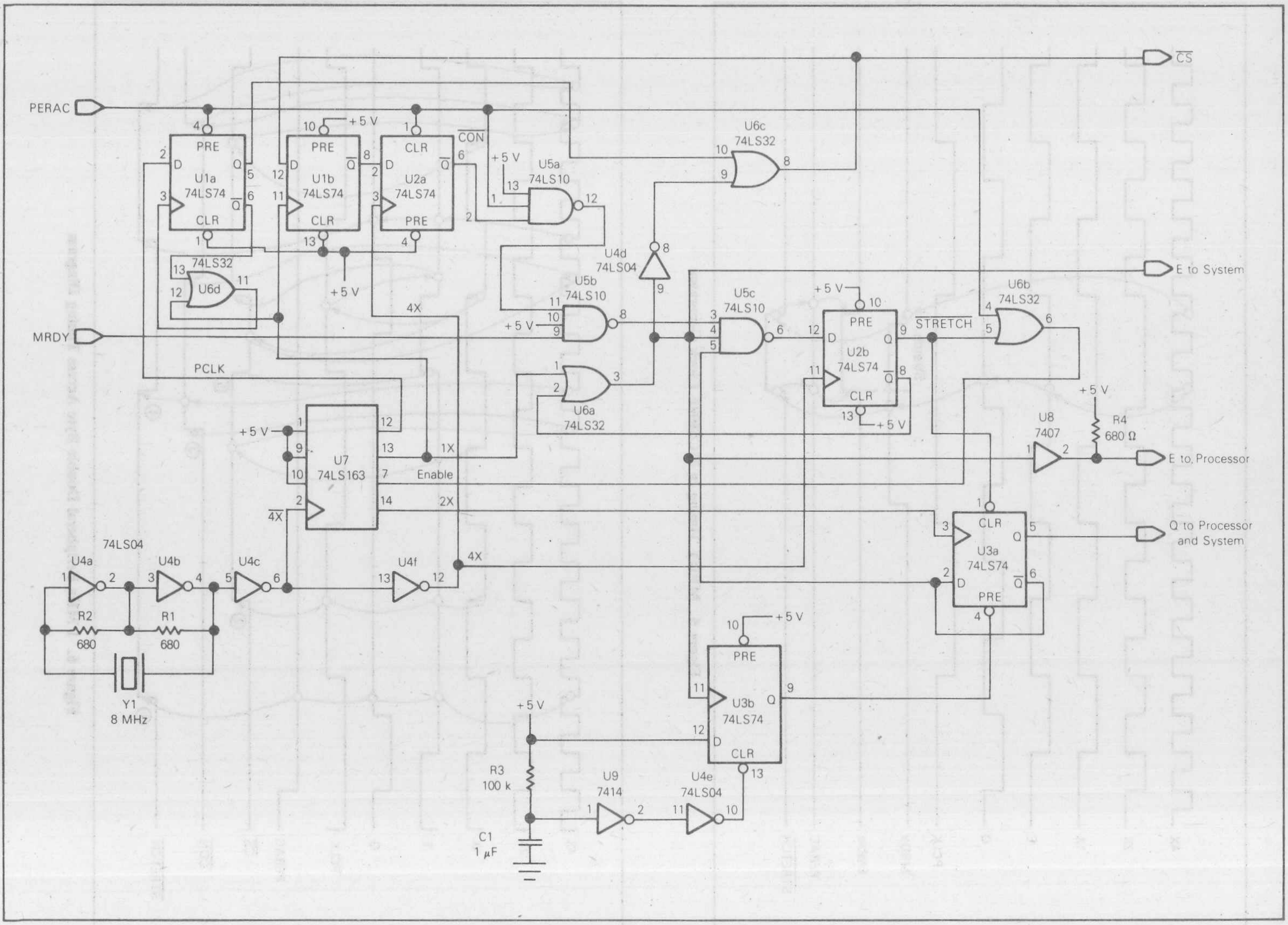


Figure 4. Clock Generator Circuit for MC6809E

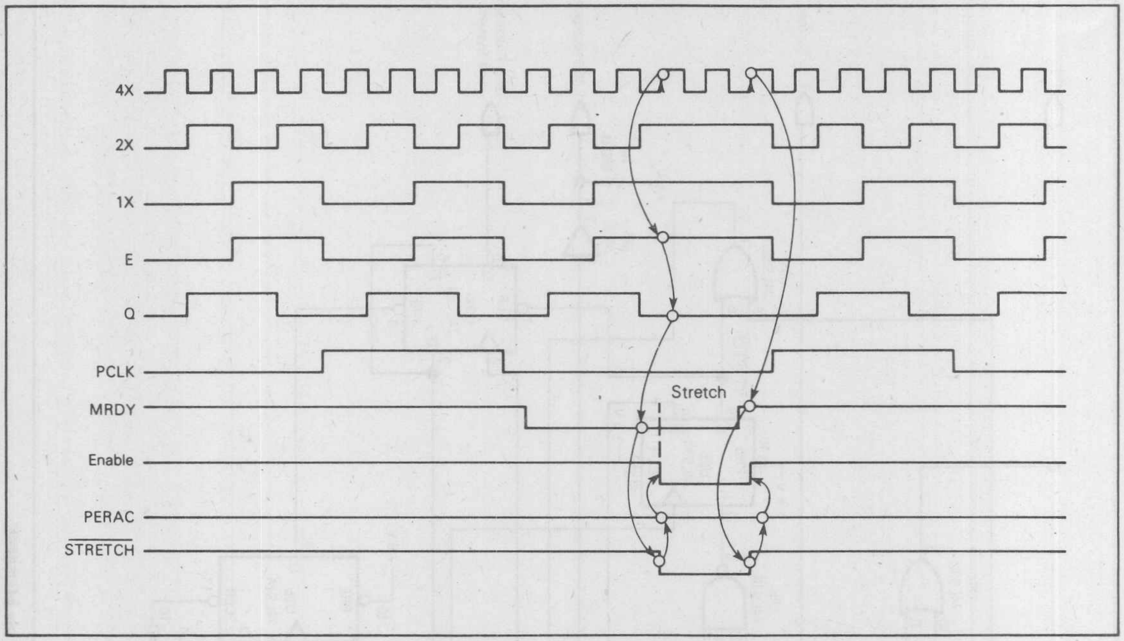


Figure 5. MRDY Timing in MC6809E Clock Generator

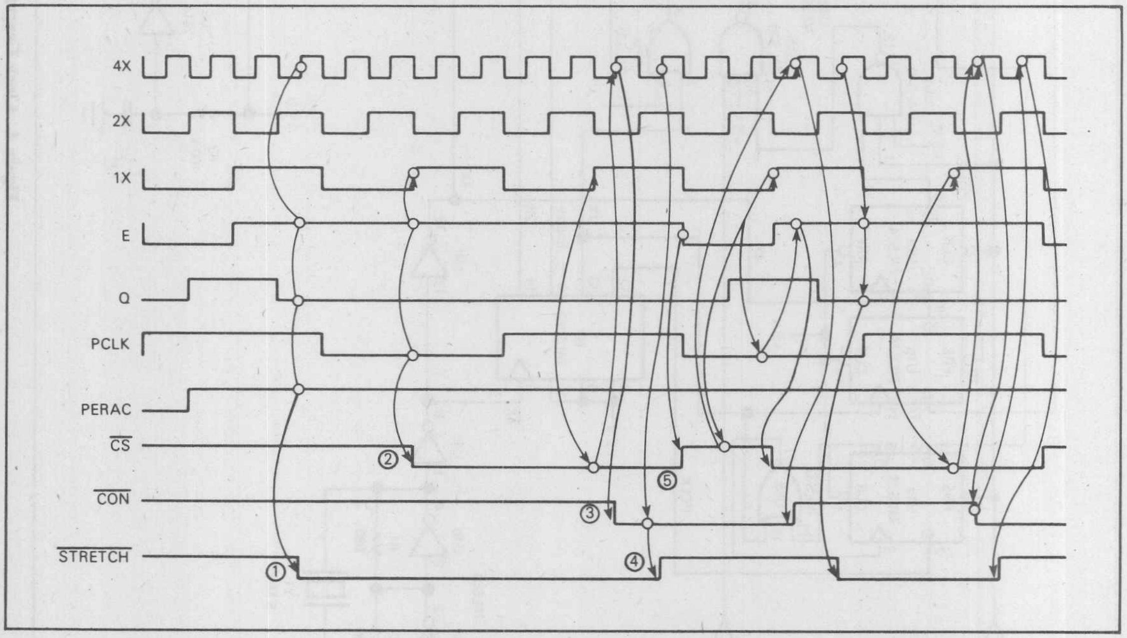


Figure 6. 1 MHz Peripheral Double Byte Access Timing Diagram

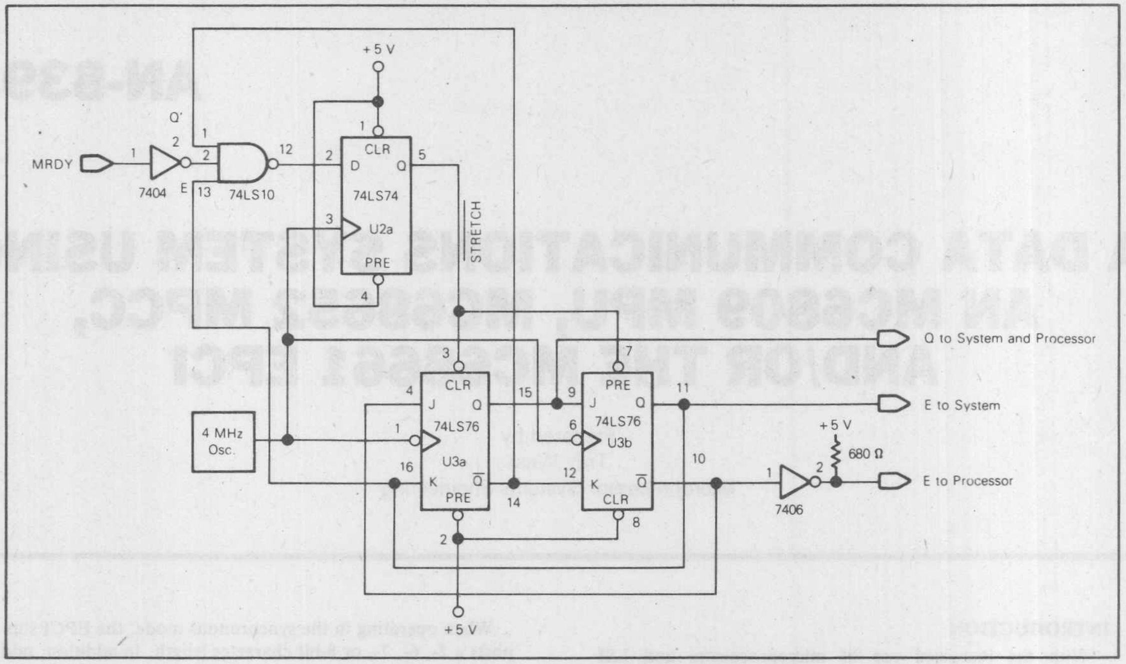


Figure 7. Clock Generator for MC6809E with MRDY Input

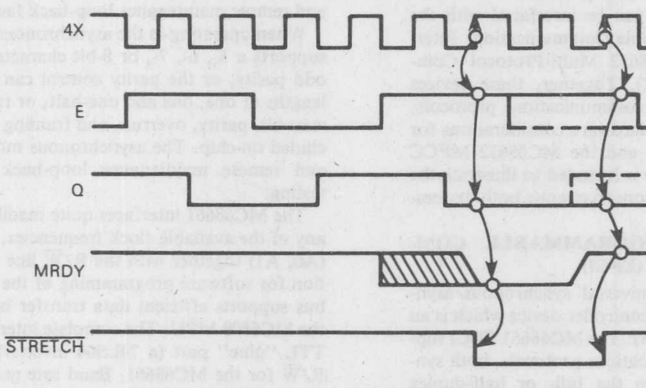


Figure 8. MC6809E Clock Generator Timing Diagram

AN-839

A DATA COMMUNICATIONS SYSTEM USING AN MC6809 MPU, MC68652 MPCC, AND/OR THE MC68661 EPCI

Prepared by
Trey West
Microprocessor Systems Engineering

INTRODUCTION

With the increased use of microprocessors and LSI receiver/transmitter devices in data communication systems, design engineers are constantly searching for new devices which will reduce system complexity and enhance system performance. Motorola's MC6809 (or MC6809E) microprocessor together with one or two LSI devices will provide the design engineer with such products.

The MC6809 microprocessor can be interfaced with the MC68661 Enhanced Programmable Communications Interface (EPCI) and/or the MC68652 Multi-Protocol Communications Controller (MPCC). Together, these devices will support most existing data communications protocols. This application note describes hardware considerations for interfacing the MC68661 EPCI and the MC68652 MPCC with the MC6809. Test software is included to illustrate the effective use of MC6809 instructions to operate both devices.

MC68661 ENHANCED PROGRAMMABLE COMMUNICATIONS INTERFACE (EPCI)

The MC68661 EPCI is a universal synchronous/asynchronous data communications controller device which is an enhanced version of Signetics 2651. The MC68661 EPCI supports many serial data communications protocols, both synchronous and asynchronous, in the full- or half-duplex mode. Programmed instructions can be accepted from the host MPU while supporting these protocols. Special support for BISYNC is provided with the inclusion of an MC68653 Polynomial Generator Checker circuit.

The EPCI contains an internal, software programmable baud rate generator which supports up to 16 commonly used baud rates. Each version of the MC68661 (−A, −B, −C) provides a different set of 16 baud rates (since each operates with a different combination of BRCLK input frequency and rate divisor). However, by providing an external receiver and transmitter clock, any baud rate may be used.

When operating in the synchronous mode, the EPCI supports a 5-, 6-, 7-, or 8-bit character length. In addition, odd or even parity can be used or the parity control can be disabled. The EPCI may be programmed to operate in the transparent mode with DLE stuffing (Tx) and detection (Rx). The EPCI can also operate in the non-transparent (normal) mode. Automatic SYN or DLE-SYN insertion as well as SYN, DLE, and DLE-SYN stripping are provided. Local and remote maintenance loop-back facilitates system testing.

When operating in the asynchronous mode, the EPCI also supports a 5-, 6-, 7-, or 8-bit character length with even or odd parity, or the parity control can be disabled. Stop bit lengths of one, one and one-half, or two may be used. False start bit, parity, overrun, and framing error detection are included on-chip. The asynchronous mode also includes local and remote maintenance loop-back to facilitate system testing.

The MC68661 interfaces quite readily with the MC6809 at any of the available clock frequencies. The two address lines (A0, A1) together with the R/$\overline{W}$ line provide register selection for software programming of the EPCI. The 8-bit data bus supports efficient data transfer between the EPCI and the MC6809 MPU. The complete interface requires only one TTL "glue" part (a 74LS04 inverter) to supply Reset and R/$\overline{W}$ for the MC68661. Baud rate generation is configured internally under software control provided the proper frequency is applied to BRCLK (pin 20). See Figure 1 for an interface schematic diagram.

Software for the asynchronous test is shown in Figure 2. This software provides a full-duplex buffer for a terminal. A character entered at the terminal is simply echoed back to the terminal. This simple routine could be modified to perform the function of a monitor input handler.

NOTE

The first initialization routine listed is for the

ASSIST09 (MC6809) monitor only. This routine simply equates the $\overline{\text{IRQ}}$ vector to a user defined location.

MC68652 MULTI-PROTOCOL COMMUNICATIONS CONTROLLER (MPCC)

The MC68652 MPCC formats, transmits, and receives synchronous serial data while supporting Bit Oriented Protocols (BOP) or Byte Control Protocols (BCP). The data transmission rate is externally controlled and runs from DC to 1 MHz (MC68652) or 2 MHz (MC68652_2). The MPCC supports SDLC, HDLC, and ADCCP in the BOP mode, and DDCMP and BISYNC in the BCP mode. Transmitted character length may be 1 to 8 bits for BOP and 5 to 8 bits for BCP. The MPCC provides for programmable SYNC (BCP) or secondary station address (BOP). Automatic detection and generation of Flag, Abort, and GA sequences are included on-chip. The MPCC also supports zero insertion and deletion for BOP, and SYNC generation, detection, and stripping for BCP. The maintenance mode pin, when asserted, internally connects the transmitter output and TxC to the receiver input and RxC, respectively, to expedite system testing.

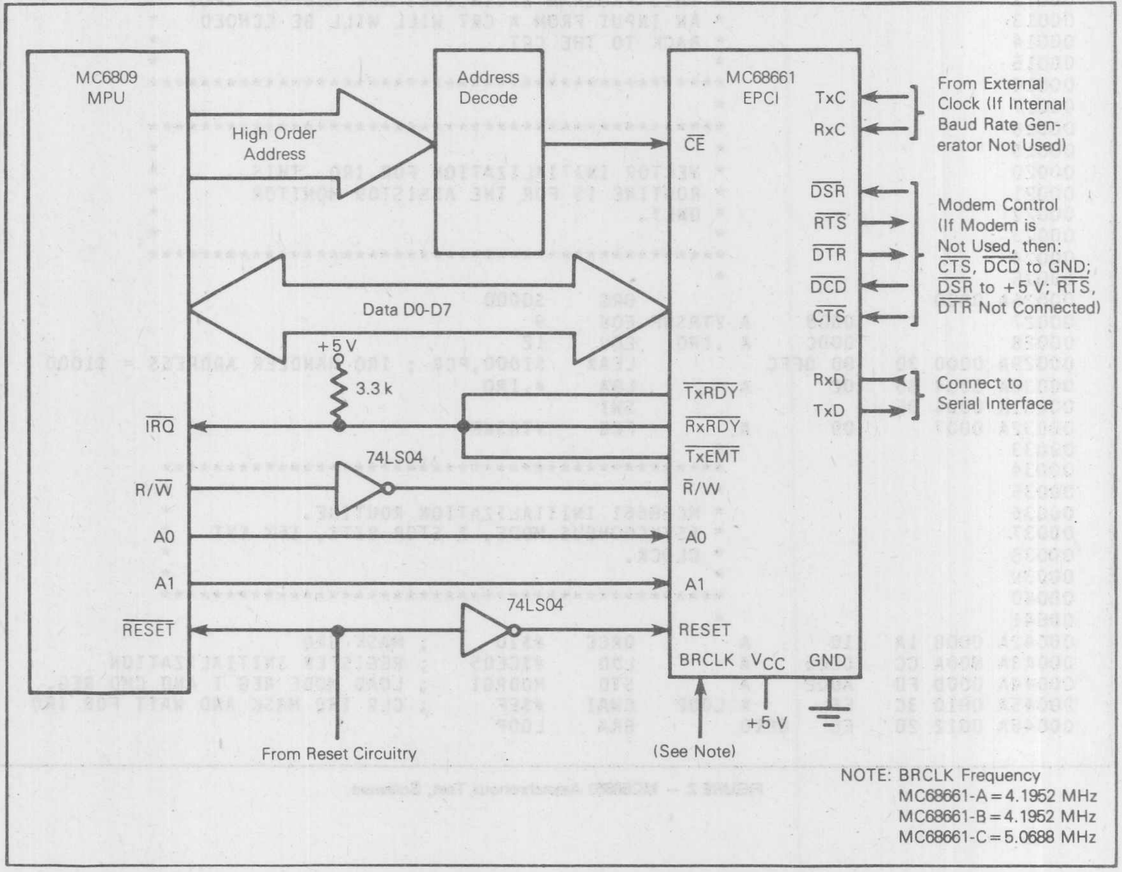
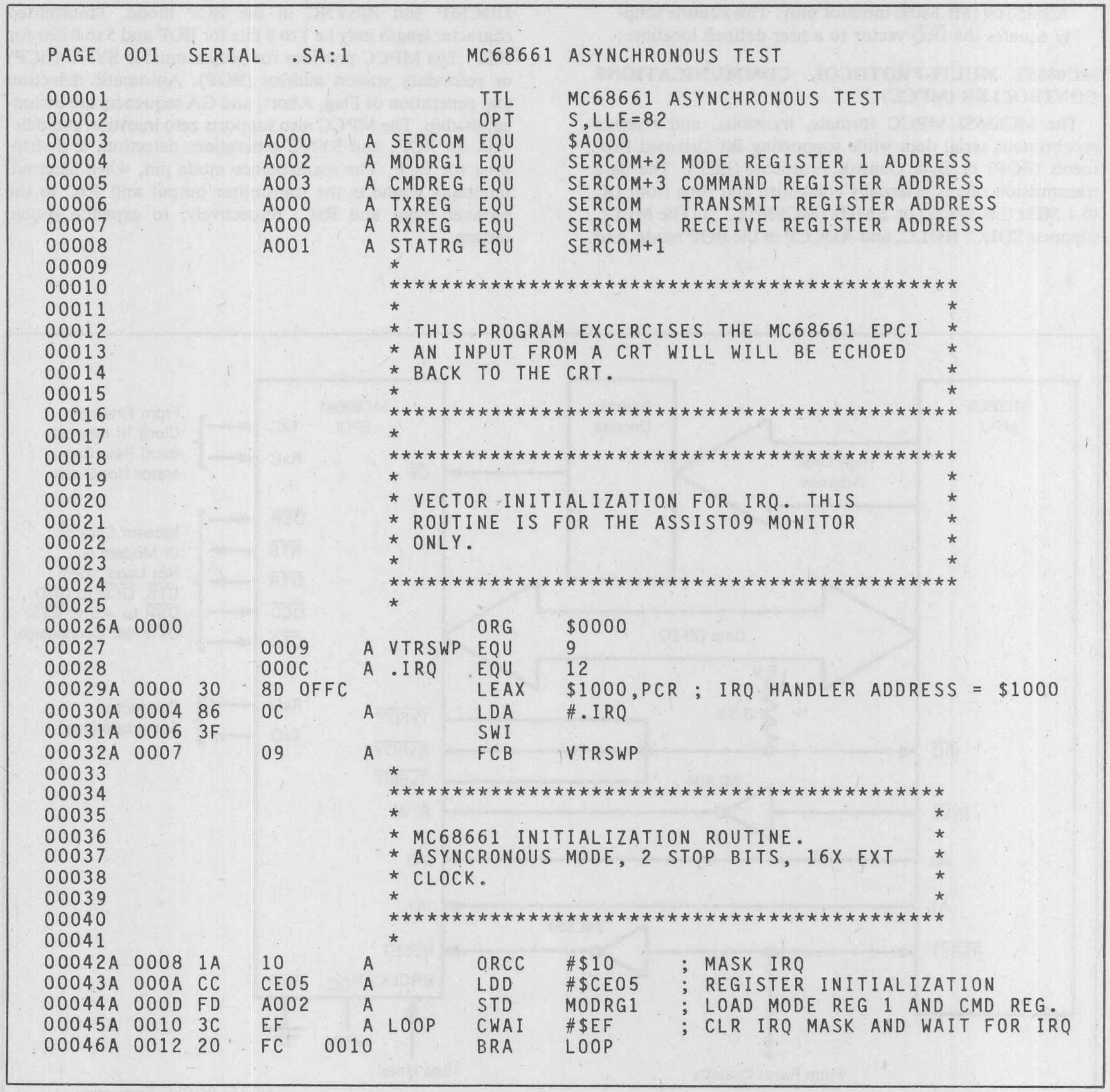


FIGURE 1 — MC6809 to MC68661 Interface Connection, Schematic Diagram



```
PAGE  001   SERIAL   .SA:1        MC68661 ASYNCHRONOUS TEST

00001                              TTL     MC68661 ASYNCHRONOUS TEST
00002                              OPT     S,LLE=82
00003               A000      A SERCOM EQU     $A000
00004               A002      A MODRG1 EQU     SERCOM+2 MODE REGISTER   ADDRESS
00005               A003      A CMDREG EQU     SERCOM+3 COMMAND REGISTER ADDRESS
00006               A000      A TXREG  EQU     SERCOM   TRANSMIT REGISTER ADDRESS
00007               A000      A RXREG  EQU     SERCOM   RECEIVE REGISTER ADDRESS
00008               A001      A STATRG EQU     SERCOM+1
00009                           *
00010                           ********************************************
00011                           *                                          *
00012                           * THIS PROGRAM EXCERCISES THE MC68661 EPCI *
00013                           * AN INPUT FROM A CRT WILL WILL BE ECHOED  *
00014                           * BACK TO THE CRT.                         *
00015                           *                                          *
00016                           ********************************************
00017                           *
00018                           ********************************************
00019                           *                                          *
00020                           * VECTOR INITIALIZATION FOR IRQ. THIS      *
00021                           * ROUTINE IS FOR THE ASSIST09 MONITOR      *
00022                           * ONLY.                                    *
00023                           *                                          *
00024                           ********************************************
00025                           *
00026A 0000                        ORG     $0000
00027               0009      A VTRSWP EQU     9
00028               000C      A .IRQ   EQU     12
00029A 0000 30   8D OFFC           LEAX    $1000,PCR ; IRQ HANDLER ADDRESS = $1000
00030A 0004 86   0C           A    LDA     #.IRQ
00031A 0006 3F                     SWI
00032A 0007      09           A    FCB     VTRSWP
00033                           *
00034                           *******************************************
00035                           *                                         *
00036                           * MC68661 INITIALIZATION ROUTINE.         *
00037                           * ASYNCRONOUS MODE, 2 STOP BITS, 16X EXT  *
00038                           * CLOCK.                                  *
00039                           *                                         *
00040                           *******************************************
00041                           *
00042A 0008 1A   10           A    ORCC    #$10      ; MASK IRQ
00043A 000A CC   CE05         A    LDD     #$CE05    ; REGISTER INITIALIZATION
00044A 000D FD   A002         A    STD     MODRG1    ; LOAD MODE REG 1 AND CMD REG.
00045A 0010 3C   EF           A LOOP   CWAI    #$EF      ; CLR IRQ MASK AND WAIT FOR IRQ
00046A 0012 20   FC    0010        BRA     LOOP
```

FIGURE 2 — MC68661 Asynchronous Test, Software

```
PAGE  002  SERIAL  .SA:1              MC68661 ASYNCHRONOUS TEST

00048                        *
00049                        ****************************************************
00050                        *                                                  *
00051                        * INTERRUPT HANDLER. INPUTS CHARACTERS THEN *
00052                        * ECHOS THEM BACK THROUGH THE TRANSMITTER.  *
00053                        *                                                  *
00054                        ****************************************************
00055                        *
00056A 1000                           ORG     $1000
00057A 1000 1A    10       A          ORCC    #$10        ; MASK IRQ
00058A 1002 B6    A001     A          LDA     STATRG
00059A 1005 84    02       A          ANDA    #$02        ; RECEIVER INTERRUPT?
00060A 1007 26    01    100A          BNE     INPUT
00061A 1009 3B                        RTI                 ; IF NOT RETURN AND WAIT
00062A 100A F6    A000     A INPUT    LDB     RXREG       ; LOAD INPUT CHARACTER
00063A 100D F7    A000     A          STB     TXREG       ; OUTPUT CHARACTER
00064A 1010 3B                        RTI                 ; RETURN AND WAIT FOR NEXT CHAR
00065                                 END
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000


.IRQ    000C  CMDREG A003   INPUT  100A  LOOP   0010  MODRG1 A002  RXREG  A000
SERCOM A000   STATRG A001   TXREG  A000  VTRSWP 0009
```

FIGURE 2 — MC68661 Asynchronous Test, Software
(Concluded)

Interfacing the MPCC with the MC6809 MPU is slightly more involved than the EPCI. The MPCC transfers data on a 16-bit data bus; however, when the BYTE pin is asserted, data transfers for the 16-bit internal registers are accomplished one byte at a time (assuming D0 is connected to D8, D1 to D9, etc.). Refer to Figure 3. Data transfers are accomplished in the following MC6809 sequence: address and R/$\overline{W}$ are asserted immediately after CE is asserted (CE is derived from address decoding, thus the delay is inherent). For a read, DBEN is asserted and an access time later, the proper data appears on the bus specified by the register addressed. For a write, data must be valid 50 nanoseconds prior to the assertion of DBEN. The DBEN signal is generated by ANDing E with inverted Q. With this pulse, data easily meets the 50 nanosecond setup time. Refer to the timing diagram in Figure 4 for further clarification. For the MC68652 1 MHz part, DBEN minimum is specified as 250 nanoseconds; however, the E•$\overline{Q}$ signal for a 1 MHz MC6809 will be somewhat shorter than this. Several 1 MHz parts were tested using the design of Figure 3 and all ran successfully at 1.5 MHz; however, since it is not good design practice to violate a minimum specification, MRDY clock stretching for the

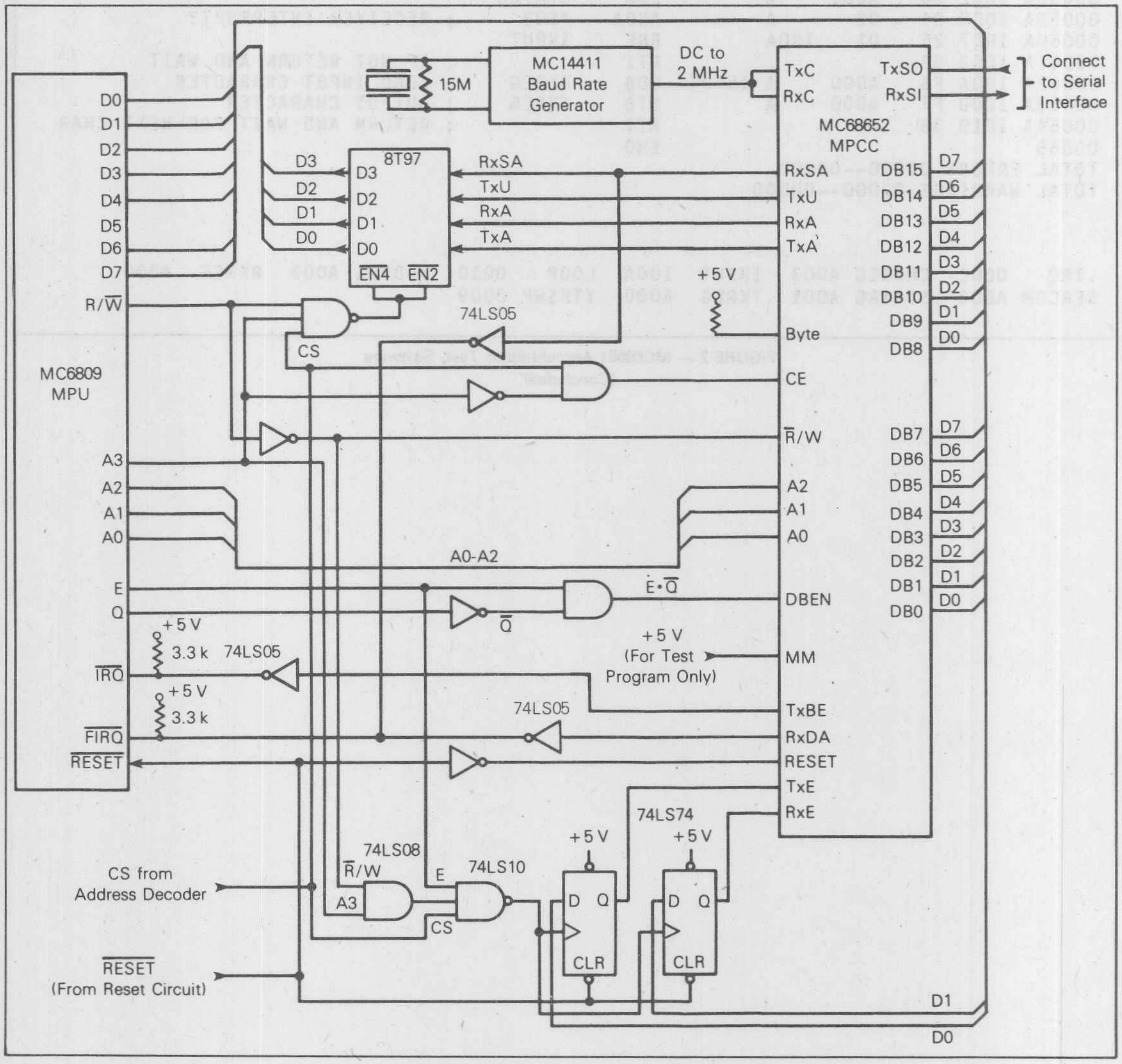


FIGURE 3 — MC6809 to MC68652 Interface Connection, Schematic Diagram

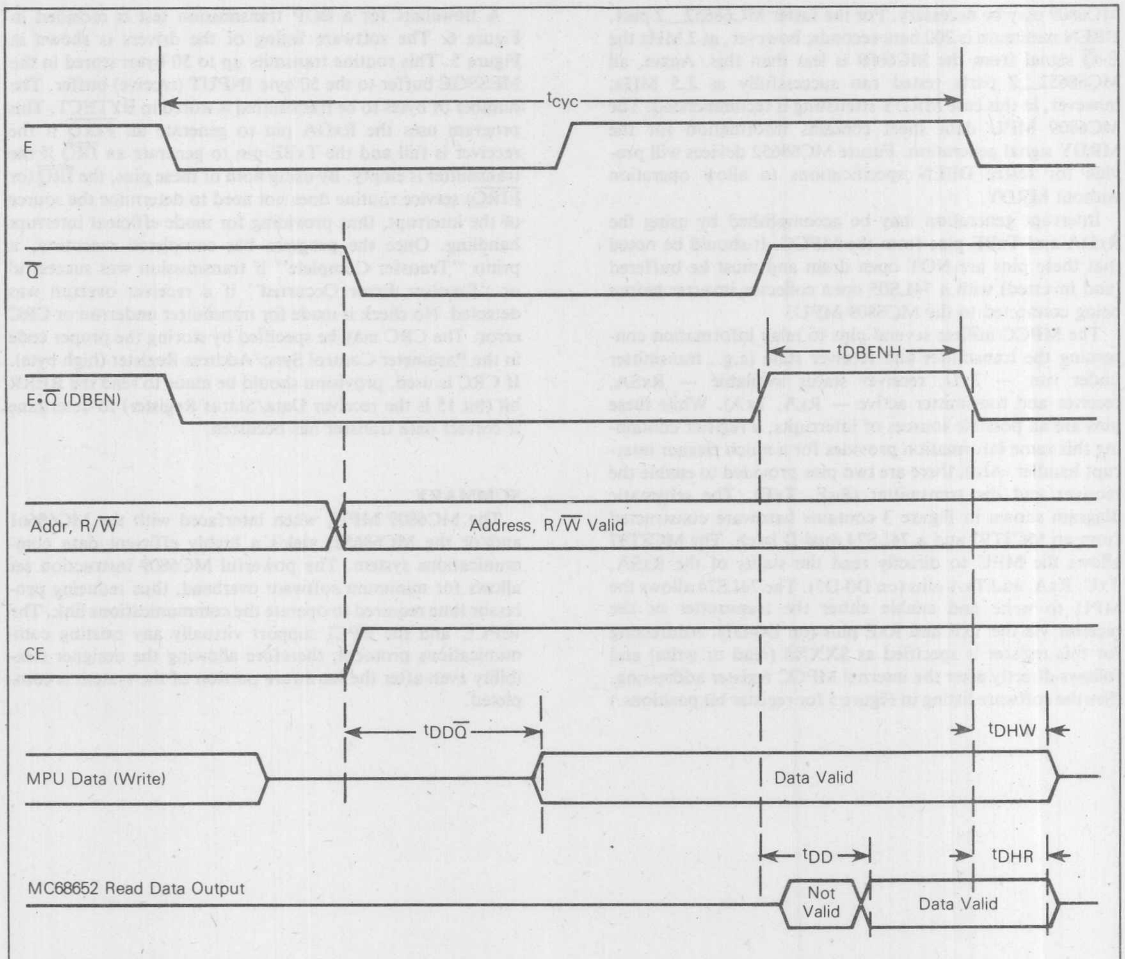


TABLE OF PARAMETER DEFINITIONS

Parameter	Description	MC68652_1 MC6809		MC68652_2 MC68A09		MC68652_2 MC68B09	
		Min	Max	Min	Max	Min	Max
t_{cyc}	MPU E Clock Cycle Time	1 μs		667 ns		500 ns	
t_{DBENH}	Data Bus Enable High Time (E•$\overline{Q}$) MPU	180 ns	230 ns	95 ns	130 ns	120 ns	
t_{DBENH}	DBEN High Time Required by MPCC	250 ns		200 ns		200 ns	
$t_{DD\overline{Q}}$	Data Delay from $\overline{Q}$ MPU (Includes Delay from 74LS04 for $\overline{Q}$ Generation)		180 ns		120 ns		90 ns
t_{DD}	Data Valid Delay from MPCC		200 ns		170 ns		170 ns
t_{DSR}	$t_{DBEN} - t_{DD}$ (MPCC) Data Setup Time		50 ns		30 ns		30 ns
t_{DSR}	Data Setup Time Required by MPU	80 ns		60 ns		40 ns	

NOTES:

1. All required MPCC setup times (Address, R/$\overline{W}$, Data) are met by MC6809 at all clock speeds.
2. MRDY clock stretching is required to meet the t_{DBENH} and t_{DSR} specifications in all three cases.
3. $\overline{Q}$ occurs 20 ns after Q (74LS04 propagation delay).

FIGURE 4 — MC6809 to MC68652 Timing Diagram

MC6809 may be necessary. For the faster MC68652_2 part, DBEN minimum is 200 nanoseconds; however, at 2 MHz the E•$\overline{Q}$ signal from the MC6809 is less than this. Again, all MC68652_2 parts tested ran successfully at 2.5 MHz; however, in this case MRDY stretching is recommended. The MC6809 MPU data sheet contains information for the MRDY signal generation. Future MC68652 devices will provide for faster DBEN specifications to allow operation without MRDY.

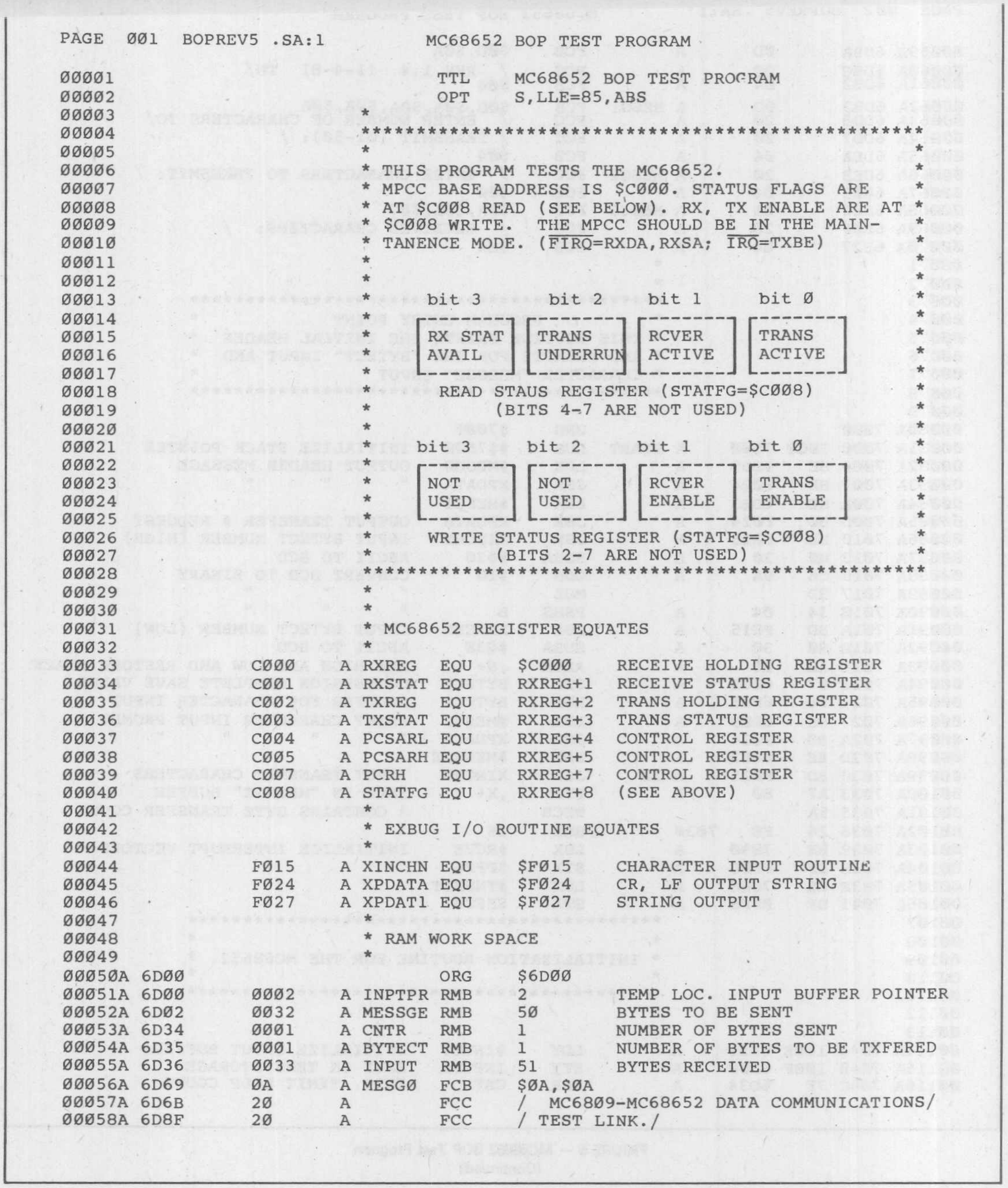
Interrupt generation may be accomplished by using the RxDA and TxBE pins from the MPCC. It should be noted that these pins are NOT open drain and must be buffered (and inverted) with a 74LS05 open collector inverter before being connected to the MC6809 MPU.

The MPCC utilizes several pins to relay information concerning the transmitter and receiver state (e.g., transmitter under run — TxU, receiver status available — RxSA, receiver and transmitter active — RxA, TxA). While these pins are all possible sources of interrupts, a register containing this same information provides for a much cleaner interrupt handler. Also, there are two pins provided to enable the receiver and the transmitter (RxE, TxE). The schematic diagram shown in Figure 3 contains hardware constructed from an MC8T97 and a 74LS74 dual D latch. The MC8T97 allows the MPU to directly read the status of the RxSA, TxU, RxA, and TxA pins (on D0-D3). The 74LS74 allows the MPU to write and enable either the transmitter or the receiver via the TxE and RxE pins (on D0-D1). Addressing for this register is specified as $XXX8 (read or write) and follows directly after the internal MPCC register addressing. (See the software listing in Figure 5 for register bit positions.)

A flowchart for a BOP transmission test is included in Figure 6. The software listing of the drivers is shown in Figure 5. This routine transmits up to 50 bytes stored in the MESSGE buffer to the 50 byte INPUT (receive) buffer. The number of bytes to be transmitted is stored in BYTECT. This program uses the RxDA pin to generate an $\overline{\text{FIRQ}}$ if the receiver is full and the TxBE pin to generate an $\overline{\text{IRQ}}$ if the transmitter is empty. By using both of these pins, the $\overline{\text{IRQ}}$ (or $\overline{\text{FIRQ}}$) service routine does not need to determine the source of the interrupt, thus providing for mode efficient interrupt handling. Once the program has completed execution, it prints "Transfer Complete" if transmission was successful or "Receiver Error Occurred" if a receiver overrun was detected. No check is made for transmitter underrun or CRC error. The CRC may be specified by storing the proper code in the Parameter Control Sync/Address Register (high byte). If CRC is used, provision should be made to read the RERR bit (bit 15 is the receiver Data/Status Register) to determine if correct data transfer has occurred.

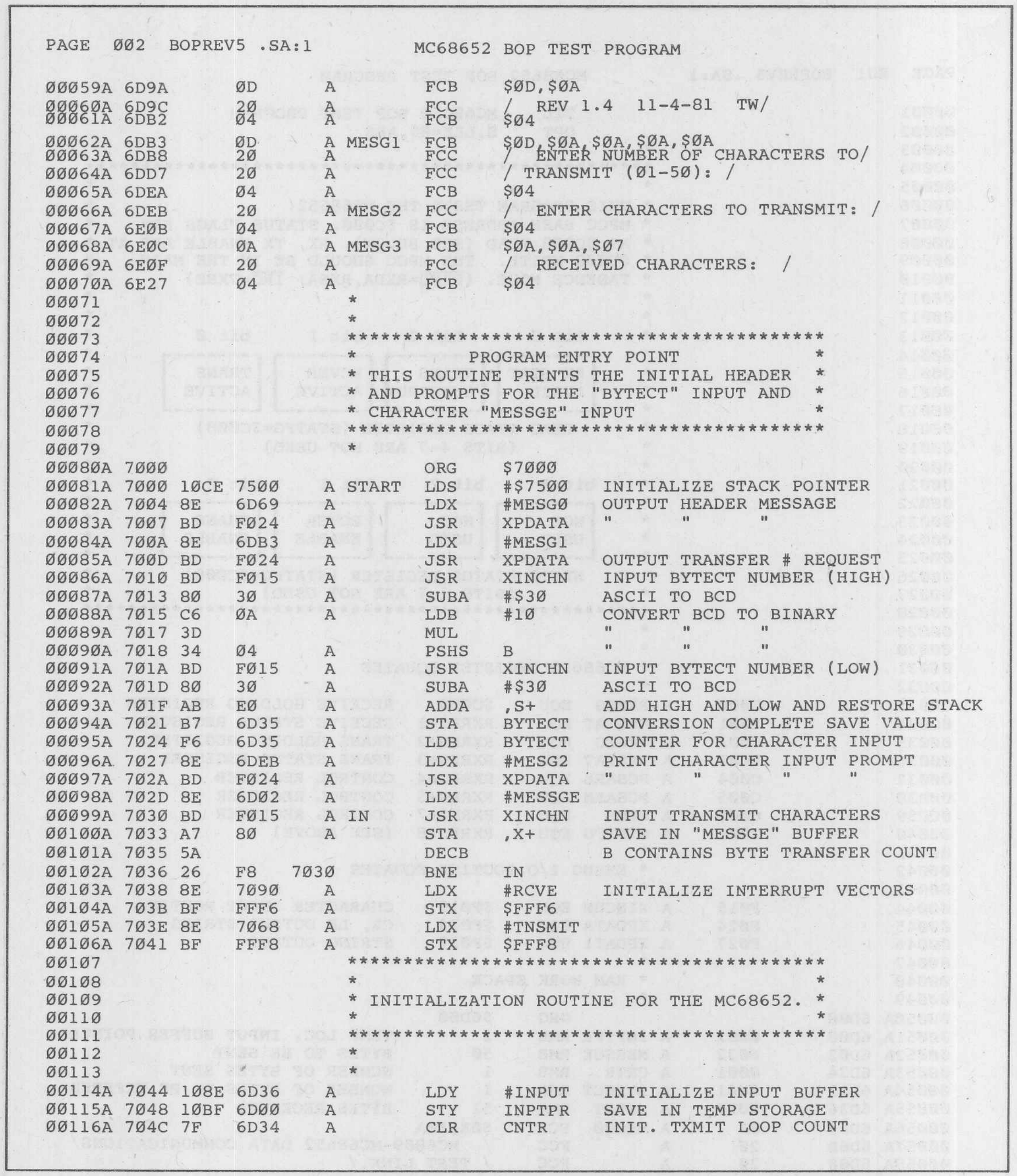
SUMMARY

The MC6809 MPU, when interfaced with the MC68661 and/or the MC68652, yields a highly efficient data communications system. The powerful MC6809 instruction set allows for minimum software overhead, thus reducing processor time required to operate the communications link. The MPCC and the EPCI support virtually any existing communications protocol, therefore allowing the designer flexibility even after the hardware portion of the system is completed.



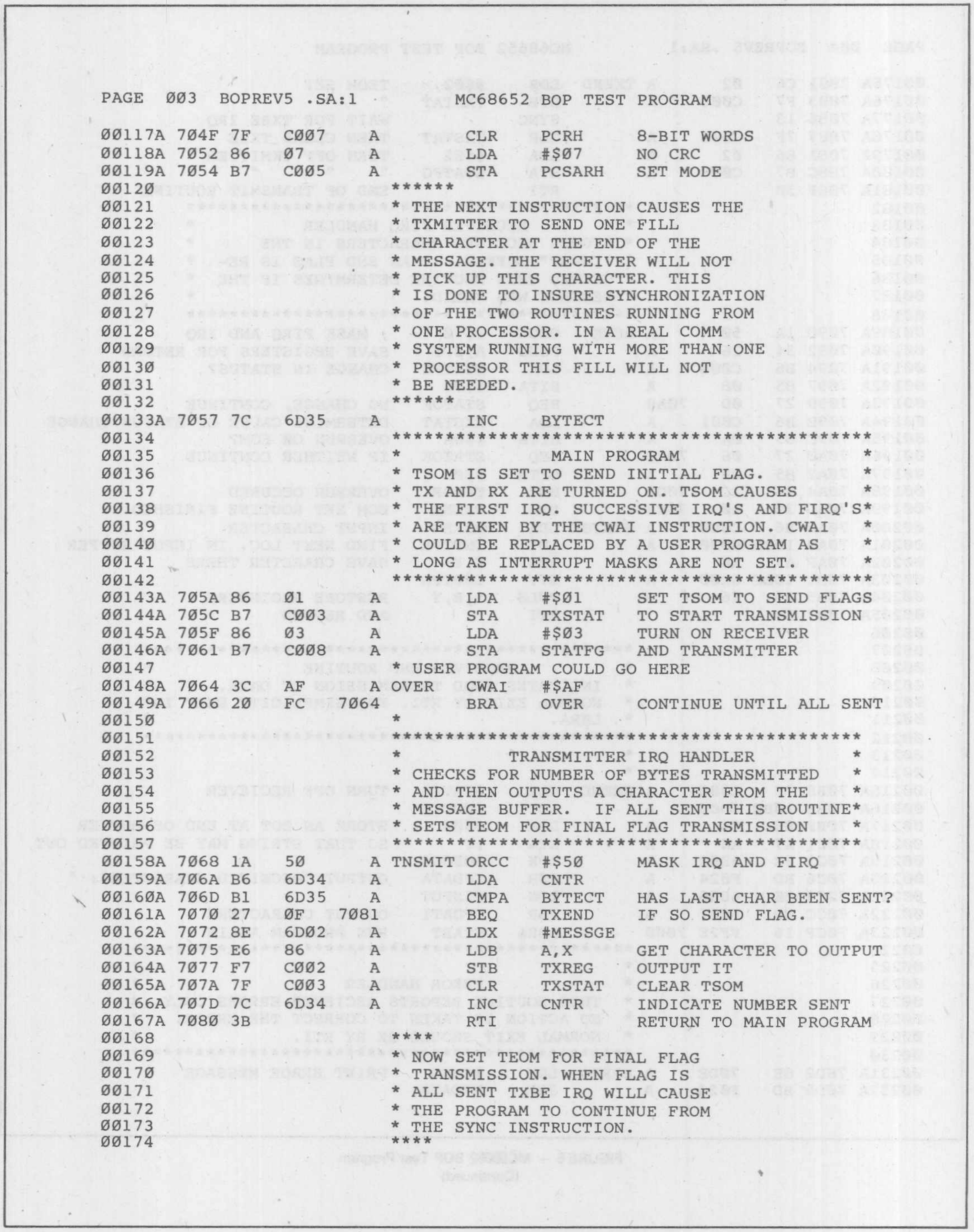
```
PAGE  001  BOPREV5 .SA:1          MC68652 BOP TEST PROGRAM

00001                          TTL    MC68652 BOP TEST PROGRAM
00002                          OPT    S,LLE=85,ABS
00003                      *
00004                      ***********************************************
00005                      *                                             *
00006                      * THIS PROGRAM TESTS THE MC68652.             *
00007                      * MPCC BASE ADDRESS IS $C000. STATUS FLAGS ARE *
00008                      * AT $C008 READ (SEE BELOW). RX, TX ENABLE ARE AT *
00009                      * $C008 WRITE.  THE MPCC SHOULD BE IN THE MAIN- *
00010                      * TANENCE MODE. (FIRQ=RXDA,RXSA; IRQ=TXBE)    *
00011                      *                                             *
00012                      *                                             *
00013                      *   bit 3     bit 2      bit 1      bit 0     *
00014                      *                                             *
00015                      * RX STAT  TRANS      RCVER      TRANS        *
00016                      * AVAIL    UNDERRUN   ACTIVE     ACTIVE       *
00017                      *                                             *
00018                      *   READ STAUS REGISTER (STATFG=$C008)        *
00019                      *        (BITS 4-7 ARE NOT USED)              *
00020                      *                                             *
00021                      *  bit 3     bit 2     bit 1      bit 0       *
00022                      *                                             *
00023                      *  NOT       NOT       RCVER      TRANS       *
00024                      *  USED      USED      ENABLE     ENABLE      *
00025                      *                                             *
00026                      *  WRITE STATUS REGISTER (STATFG=$C008)       *
00027                      *        (BITS 2-7 ARE NOT USED)              *
00028                      ***********************************************
00029                      *
00030                      *
00031                      * MC68652 REGISTER EQUATES
00032                      *
00033           C000     A RXREG   EQU  $C000     RECEIVE HOLDING REGISTER
00034           C001     A RXSTAT  EQU  RXREG+1   RECEIVE STATUS REGISTER
00035           C002     A TXREG   EQU  RXREG+2   TRANS HOLDING REGISTER
00036           C003     A TXSTAT  EQU  RXREG+3   TRANS STATUS REGISTER
00037           C004     A PCSARL  EQU  RXREG+4   CONTROL REGISTER
00038           C005     A PCSARH  EQU  RXREG+5   CONTROL REGISTER
00039           C007     A PCRH    EQU  RXREG+7   CONTROL REGISTER
00040           C008     A STATFG  EQU  RXREG+8   (SEE ABOVE)
00041                      *
00042                      * EXBUG I/O ROUTINE EQUATES
00043                      *
00044           F015     A XINCHN  EQU  $F015     CHARACTER INPUT ROUTINE
00045           F024     A XPDATA  EQU  $F024     CR, LF OUTPUT STRING
00046           F027     A XPDAT1  EQU  $F027     STRING OUTPUT
00047                      *
00048                      * RAM WORK SPACE
00049                      *
00050A 6D00                       ORG  $6D00
00051A 6D00     0002     A INPTPR  RMB  2         TEMP LOC. INPUT BUFFER POINTER
00052A 6D02     0032     A MESSGE  RMB  50        BYTES TO BE SENT
00053A 6D34     0001     A CNTR    RMB  1         NUMBER OF BYTES SENT
00054A 6D35     0001     A BYTECT  RMB  1         NUMBER OF BYTES TO BE TXFERED
00055A 6D36     0033     A INPUT   RMB  51        BYTES RECEIVED
00056A 6D69     0A       A MESG0   FCB  $0A,$0A
00057A 6D6B     20       A         FCC  / MC6809-MC68652 DATA COMMUNICATIONS/
00058A 6D8F     20       A         FCC  / TEST LINK./
```

FIGURE 5 — MC68652 BOP Test Program



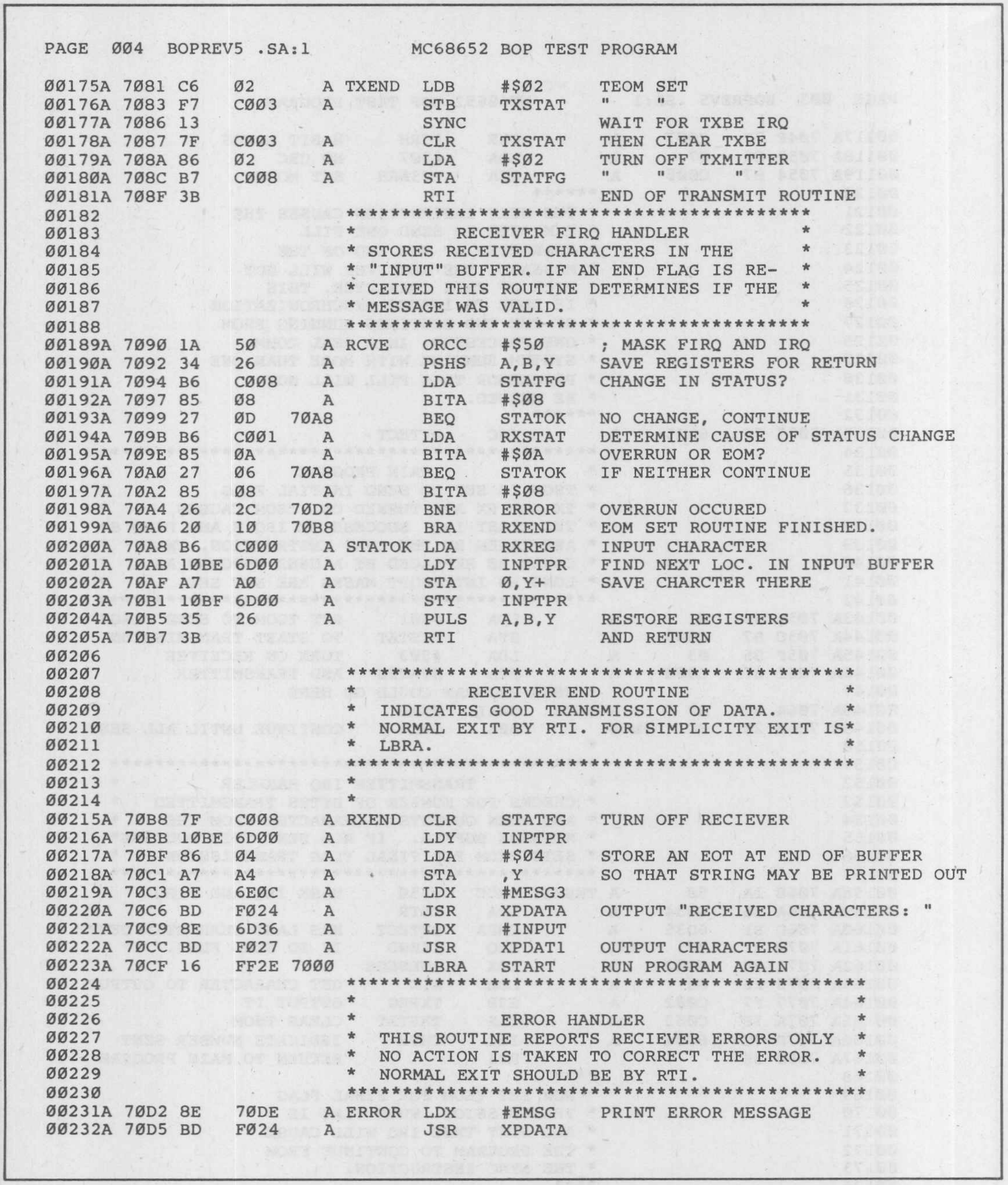
```
PAGE  ØØ2  BOPREV5 .SA:1            MC68652 BOP TEST PROGRAM

ØØØ59A 6D9A      ØD        A        FCB     $ØD,$ØA
ØØØ6ØA 6D9C      2Ø        A        FCC     /  REV 1.4  11-4-81  TW/
ØØØ61A 6DB2      Ø4        A        FCB     $Ø4
ØØØ62A 6DB3      ØD        A MESG1  FCB     $ØD,$ØA,$ØA,$ØA,$ØA
ØØØ63A 6DB8      2Ø        A        FCC     /  ENTER NUMBER OF CHARACTERS TO/
ØØØ64A 6DD7      2Ø        A        FCC     / TRANSMIT (Ø1-5Ø): /
ØØØ65A 6DEA      Ø4        A        FCB     $Ø4
ØØØ66A 6DEB      2Ø        A MESG2  FCC     /  ENTER CHARACTERS TO TRANSMIT: /
ØØØ67A 6EØB      Ø4        A        FCB     $Ø4
ØØØ68A 6EØC      ØA        A MESG3  FCB     $ØA,$ØA,$Ø7
ØØØ69A 6EØF      2Ø        A        FCC     /  RECEIVED CHARACTERS:  /
ØØØ7ØA 6E27      Ø4        A        FCB     $Ø4
ØØØ71                        *
ØØØ72                        *
ØØØ73                        **********************************************
ØØØ74                        *          PROGRAM ENTRY POINT               *
ØØØ75                        * THIS ROUTINE PRINTS THE INITIAL HEADER     *
ØØØ76                        * AND PROMPTS FOR THE "BYTECT" INPUT AND     *
ØØØ77                        * CHARACTER "MESSGE" INPUT                   *
ØØØ78                        **********************************************
ØØØ79                        *
ØØØ8ØA 7ØØØ                         ORG     $7ØØØ
ØØØ81A 7ØØØ 1ØCE 75ØØ      A START  LDS     #$75ØØ    INITIALIZE STACK POINTER
ØØØ82A 7ØØ4 8E   6D69      A        LDX     #MESGØ    OUTPUT HEADER MESSAGE
ØØØ83A 7ØØ7 BD   FØ24      A        JSR     XPDATA    "      "      "
ØØØ84A 7ØØA 8E   6DB3      A        LDX     #MESG1
ØØØ85A 7ØØD BD   FØ24      A        JSR     XPDATA    OUTPUT TRANSFER # REQUEST
ØØØ86A 7Ø1Ø BD   FØ15      A        JSR     XINCHN    INPUT BYTECT NUMBER (HIGH)
ØØØ87A 7Ø13 8Ø   3Ø        A        SUBA    #$3Ø      ASCII TO BCD
ØØØ88A 7Ø15 C6   ØA        A        LDB     #1Ø       CONVERT BCD TO BINARY
ØØØ89A 7Ø17 3D                      MUL               "       "      "
ØØØ9ØA 7Ø18 34   Ø4        A        PSHS    B         "       "      "
ØØØ91A 7Ø1A BD   FØ15      A        JSR     XINCHN    INPUT BYTECT NUMBER (LOW)
ØØØ92A 7Ø1D 8Ø   3Ø        A        SUBA    #$3Ø      ASCII TO BCD
ØØØ93A 7Ø1F AB   EØ        A        ADDA    ,S+       ADD HIGH AND LOW AND RESTORE STACK
ØØØ94A 7Ø21 B7   6D35      A        STA     BYTECT    CONVERSION COMPLETE SAVE VALUE
ØØØ95A 7Ø24 F6   6D35      A        LDB     BYTECT    COUNTER FOR CHARACTER INPUT
ØØØ96A 7Ø27 8E   6DEB      A        LDX     #MESG2    PRINT CHARACTER INPUT PROMPT
ØØØ97A 7Ø2A BD   FØ24      A        JSR     XPDATA    "      "      "      "
ØØØ98A 7Ø2D 8E   6DØ2      A        LDX     #MESSGE
ØØØ99A 7Ø3Ø BD   FØ15      A IN     JSR     XINCHN    INPUT TRANSMIT CHARACTERS
ØØ1ØØA 7Ø33 A7   8Ø        A        STA     ,X+       SAVE IN "MESSGE" BUFFER
ØØ1Ø1A 7Ø35 5A                      DECB              B CONTAINS BYTE TRANSFER COUNT
ØØ1Ø2A 7Ø36 26   F8   7Ø3Ø          BNE     IN
ØØ1Ø3A 7Ø38 8E   7Ø9Ø      A        LDX     #RCVE     INITIALIZE INTERRUPT VECTORS
ØØ1Ø4A 7Ø3B BF   FFF6      A        STX     $FFF6
ØØ1Ø5A 7Ø3E 8E   7Ø68      A        LDX     #TNSMIT
ØØ1Ø6A 7Ø41 BF   FFF8      A        STX     $FFF8
ØØ1Ø7                        **********************************************
ØØ1Ø8                        *                                            *
ØØ1Ø9                        * INITIALIZATION ROUTINE FOR THE MC68652.    *
ØØ11Ø                        *                                            *
ØØ111                        **********************************************
ØØ112                        *
ØØ113                        *
ØØ114A 7Ø44 1Ø8E 6D36      A        LDY     #INPUT    INITIALIZE INPUT BUFFER
ØØ115A 7Ø48 1ØBF 6DØØ      A        STY     INPTPR    SAVE IN TEMP STORAGE
ØØ116A 7Ø4C 7F   6D34      A        CLR     CNTR      INIT. TXMIT LOOP COUNT
```

FIGURE 5 — MC68652 BOP Test Program
(Continued)



```
PAGE  ØØ3  BOPREV5 .SA:1         MC68652 BOP TEST PROGRAM

ØØ117A 7Ø4F 7F  CØØ7     A        CLR    PCRH     8-BIT WORDS
ØØ118A 7Ø52 86  Ø7       A        LDA    #$Ø7     NO CRC
ØØ119A 7Ø54 B7  CØØ5     A        STA    PCSARH   SET MODE
ØØ12Ø                      ******
ØØ121                      * THE NEXT INSTRUCTION CAUSES THE
ØØ122                      * TXMITTER TO SEND ONE FILL
ØØ123                      * CHARACTER AT THE END OF THE
ØØ124                      * MESSAGE. THE RECEIVER WILL NOT
ØØ125                      * PICK UP THIS CHARACTER. THIS
ØØ126                      * IS DONE TO INSURE SYNCHRONIZATION
ØØ127                      * OF THE TWO ROUTINES RUNNING FROM
ØØ128                      * ONE PROCESSOR. IN A REAL COMM
ØØ129                      * SYSTEM RUNNING WITH MORE THAN ONE
ØØ13Ø                      * PROCESSOR THIS FILL WILL NOT
ØØ131                      * BE NEEDED.
ØØ132                      ******
ØØ133A 7Ø57 7C  6D35     A        INC    BYTECT
ØØ134                      *********************************************
ØØ135                      *          MAIN PROGRAM                     *
ØØ136                      * TSOM IS SET TO SEND INITIAL FLAG.         *
ØØ137                      * TX AND RX ARE TURNED ON. TSOM CAUSES      *
ØØ138                      * THE FIRST IRQ. SUCCESSIVE IRQ'S AND FIRQ'S*
ØØ139                      * ARE TAKEN BY THE CWAI INSTRUCTION. CWAI   *
ØØ14Ø                      * COULD BE REPLACED BY A USER PROGRAM AS    *
ØØ141                      * LONG AS INTERRUPT MASKS ARE NOT SET.      *
ØØ142                      *********************************************
ØØ143A 7Ø5A 86  Ø1       A        LDA    #$Ø1     SET TSOM TO SEND FLAGS
ØØ144A 7Ø5C B7  CØØ3     A        STA    TXSTAT   TO START TRANSMISSION
ØØ145A 7Ø5F 86  Ø3       A        LDA    #$Ø3     TURN ON RECEIVER
ØØ146A 7Ø61 B7  CØØ8     A        STA    STATFG   AND TRANSMITTER
ØØ147                      * USER PROGRAM COULD GO HERE
ØØ148A 7Ø64 3C  AF       A OVER   CWAI   #$AF
ØØ149A 7Ø66 2Ø  FC   7Ø64         BRA    OVER     CONTINUE UNTIL ALL SENT
ØØ15Ø                      *
ØØ151                      ********************************************
ØØ152                      *          TRANSMITTER IRQ HANDLER         *
ØØ153                      * CHECKS FOR NUMBER OF BYTES TRANSMITTED   *
ØØ154                      * AND THEN OUTPUTS A CHARACTER FROM THE    *
ØØ155                      * MESSAGE BUFFER.  IF ALL SENT THIS ROUTINE*
ØØ156                      * SETS TEOM FOR FINAL FLAG TRANSMISSION    *
ØØ157                      ********************************************
ØØ158A 7Ø68 1A  5Ø       A TNSMIT ORCC   #$5Ø     MASK IRQ AND FIRQ
ØØ159A 7Ø6A B6  6D34     A        LDA    CNTR
ØØ16ØA 7Ø6D B1  6D35     A        CMPA   BYTECT   HAS LAST CHAR BEEN SENT?
ØØ161A 7Ø7Ø 27  ØF   7Ø81         BEQ    TXEND    IF SO SEND FLAG.
ØØ162A 7Ø72 8E  6DØ2     A        LDX    #MESSGE
ØØ163A 7Ø75 E6  86       A        LDB    A,X      GET CHARACTER TO OUTPUT
ØØ164A 7Ø77 F7  CØØ2     A        STB    TXREG    OUTPUT IT
ØØ165A 7Ø7A 7F  CØØ3     A        CLR    TXSTAT   CLEAR TSOM
ØØ166A 7Ø7D 7C  6D34     A        INC    CNTR     INDICATE NUMBER SENT
ØØ167A 7Ø8Ø 3B                    RTI             RETURN TO MAIN PROGRAM
ØØ168                      ****
ØØ169                      * NOW SET TEOM FOR FINAL FLAG
ØØ17Ø                      * TRANSMISSION. WHEN FLAG IS
ØØ171                      * ALL SENT TXBE IRQ WILL CAUSE
ØØ172                      * THE PROGRAM TO CONTINUE FROM
ØØ173                      * THE SYNC INSTRUCTION.
ØØ174                      ****
```

FIGURE 5 — MC68652 BOP Test Program
(Continued)



```
PAGE  ØØ4  BOPREV5 .SA:1         MC68652 BOP TEST PROGRAM

ØØ175A 7Ø81 C6   Ø2      A TXEND  LDB    #$Ø2     TEOM SET
ØØ176A 7Ø83 F7   CØØ3    A        STB    TXSTAT   "   "
ØØ177A 7Ø86 13                    SYNC            WAIT FOR TXBE IRQ
ØØ178A 7Ø87 7F   CØØ3    A        CLR    TXSTAT   THEN CLEAR TXBE
ØØ179A 7Ø8A 86   Ø2      A        LDA    #$Ø2     TURN OFF TXMITTER
ØØ18ØA 7Ø8C B7   CØØ8    A        STA    STATFG   "    "   "
ØØ181A 7Ø8F 3B                    RTI             END OF TRANSMIT ROUTINE
ØØ182                      ******************************************
ØØ183                      *         RECEIVER FIRQ HANDLER          *
ØØ184                      * STORES RECEIVED CHARACTERS IN THE      *
ØØ185                      * "INPUT" BUFFER. IF AN END FLAG IS RE-  *
ØØ186                      * CEIVED THIS ROUTINE DETERMINES IF THE  *
ØØ187                      * MESSAGE WAS VALID.                     *
ØØ188                      ******************************************
ØØ189A 7Ø9Ø 1A   5Ø      A RCVE   ORCC   #$5Ø       ; MASK FIRQ AND IRQ
ØØ19ØA 7Ø92 34   26      A        PSHS   A,B,Y    SAVE REGISTERS FOR RETURN
ØØ191A 7Ø94 B6   CØØ8    A        LDA    STATFG   CHANGE IN STATUS?
ØØ192A 7Ø97 85   Ø8      A        BITA   #$Ø8
ØØ193A 7Ø99 27   ØD   7ØA8        BEQ    STATOK   NO CHANGE, CONTINUE
ØØ194A 7Ø9B B6   CØØ1    A        LDA    RXSTAT   DETERMINE CAUSE OF STATUS CHANGE
ØØ195A 7Ø9E 85   ØA      A        BITA   #$ØA     OVERRUN OR EOM?
ØØ196A 7ØAØ 27   Ø6   7ØA8        BEQ    STATOK   IF NEITHER CONTINUE
ØØ197A 7ØA2 85   Ø8      A        BITA   #$Ø8
ØØ198A 7ØA4 26   2C   7ØD2        BNE    ERROR    OVERRUN OCCURED
ØØ199A 7ØA6 2Ø   1Ø   7ØB8        BRA    RXEND    EOM SET ROUTINE FINISHED.
ØØ2ØØA 7ØA8 B6   CØØØ    A STATOK LDA    RXREG    INPUT CHARACTER
ØØ2Ø1A 7ØAB 1ØBE 6DØØ    A        LDY    INPTPR   FIND NEXT LOC. IN INPUT BUFFER
ØØ2Ø2A 7ØAF A7   AØ      A        STA    Ø,Y+     SAVE CHARCTER THERE
ØØ2Ø3A 7ØB1 1ØBF 6DØØ    A        STY    INPTPR
ØØ2Ø4A 7ØB5 35   26      A        PULS   A,B,Y    RESTORE REGISTERS
ØØ2Ø5A 7ØB7 3B                    RTI             AND RETURN
ØØ2Ø6
ØØ2Ø7                      **********************************************
ØØ2Ø8                      *          RECEIVER END ROUTINE              *
ØØ2Ø9                      *  INDICATES GOOD TRANSMISSION OF DATA.      *
ØØ21Ø                      *  NORMAL EXIT BY RTI. FOR SIMPLICITY EXIT IS*
ØØ211                      *  LBRA.                                     *
ØØ212                      **********************************************
ØØ213                      *
ØØ214                      *
ØØ215A 7ØB8 7F   CØØ8    A RXEND  CLR    STATFG   TURN OFF RECIEVER
ØØ216A 7ØBB 1ØBE 6DØØ    A        LDY    INPTPR
ØØ217A 7ØBF 86   Ø4      A        LDA    #$Ø4     STORE AN EOT AT END OF BUFFER
ØØ218A 7ØC1 A7   A4      A        STA    ,Y       SO THAT STRING MAY BE PRINTED OUT
ØØ219A 7ØC3 8E   6EØC    A        LDX    #MESG3
ØØ22ØA 7ØC6 BD   FØ24    A        JSR    XPDATA   OUTPUT "RECEIVED CHARACTERS: "
ØØ221A 7ØC9 8E   6D36    A        LDX    #INPUT
ØØ222A 7ØCC BD   FØ27    A        JSR    XPDAT1   OUTPUT CHARACTERS
ØØ223A 7ØCF 16   FF2E 7ØØØ        LBRA   START    RUN PROGRAM AGAIN
ØØ224                      **********************************************
ØØ225                      *                                            *
ØØ226                      *             ERROR HANDLER                  *
ØØ227                      *  THIS ROUTINE REPORTS RECIEVER ERRORS ONLY *
ØØ228                      *  NO ACTION IS TAKEN TO CORRECT THE ERROR.  *
ØØ229                      *  NORMAL EXIT SHOULD BE BY RTI.             *
ØØ23Ø                      **********************************************
ØØ231A 7ØD2 8E   7ØDE    A ERROR  LDX    #EMSG    PRINT ERROR MESSAGE
ØØ232A 7ØD5 BD   FØ24    A        JSR    XPDATA
```

FIGURE 5 — MC68652 BOP Test Program
(Continued)

```
PAGE  ØØ5   BOPREV5 .SA:1          MC68652 BOP TEST PROGRAM

ØØ233A 7ØD8 7F    CØØ8     A        CLR     STATFG    TURN OFF RCVER
ØØ234A 7ØDB 16    FF22 7ØØØ         LBRA    START     RUN PROGRAM AGAIN
ØØ235A 7ØDE       52       A EMSG   FCC     /RECEIVER OVERRUN ERROR OCCURED./
ØØ236A 7ØFD       ØA       A        FCB     $ØA,$ØD,$Ø4
ØØ237                               END
TOTAL ERRORS ØØØØØ--ØØØØØ
TOTAL WARNINGS ØØØØØ--ØØØØØ


BYTECT 6D35   CNTR    6D34   EMSG    7ØDE   ERROR  7ØD2   IN      7Ø3Ø   INPTPR 6DØØ
INPUT  6D36   MESGØ   6D69   MESG1   6DB3   MESG2  6DEB   MESG3   6EØC   MESSGE 6DØ2
OVER   7Ø64   PCRH    CØØ7   PCSARH CØØ5   PCSARL CØØ4   RCVE    7Ø9Ø   RXEND  7ØB8
RXREG  CØØØ   RXSTAT CØØ1   START   7ØØØ   STATFG CØØ8   STATOK 7ØA8   TNSMIT 7
```

FIGURE 5 — MC68652 BOP Test Program
(Concluded)

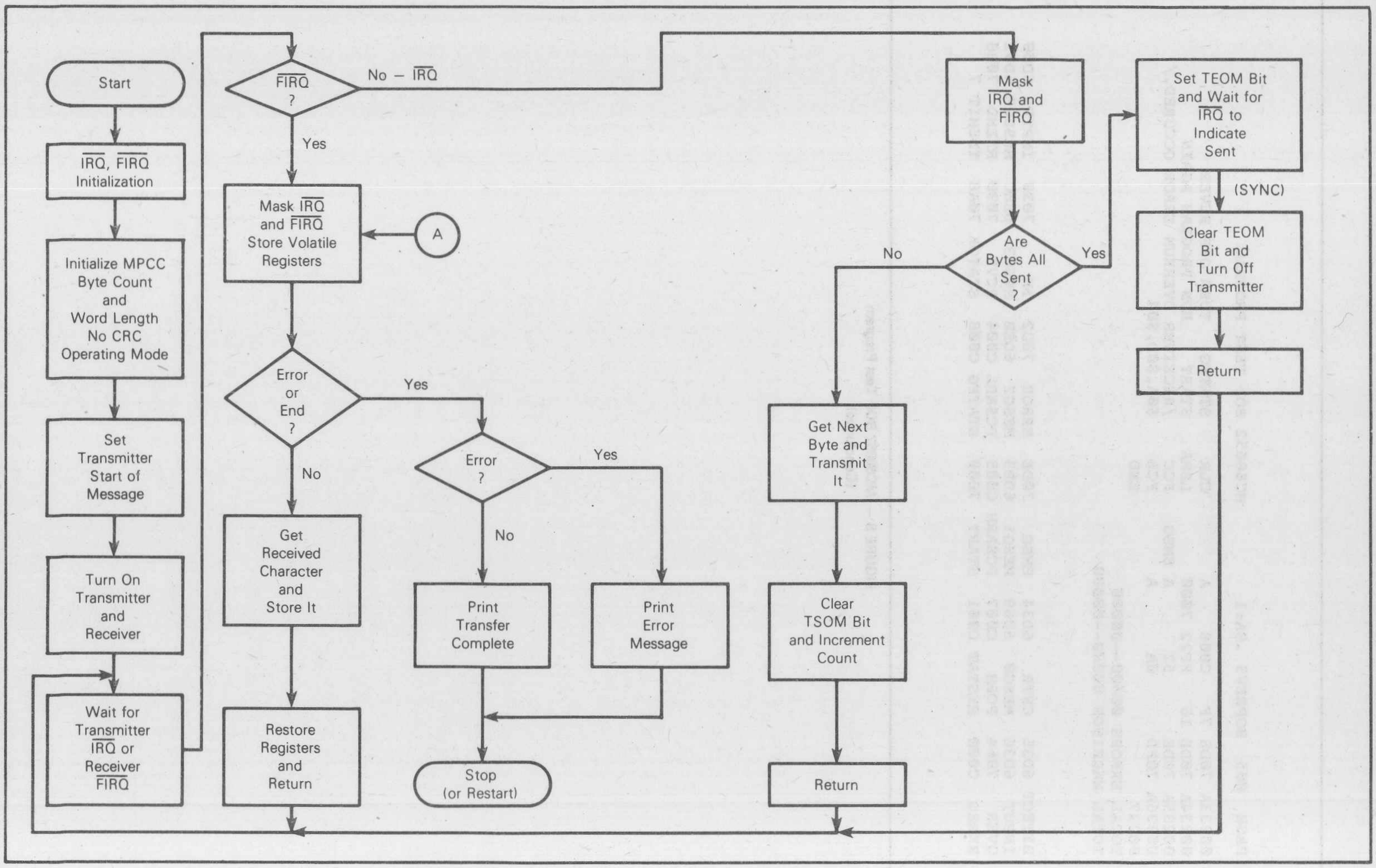

Start
$\overline{\text{IRQ}}$, $\overline{\text{FIRQ}}$ Initialization
Initialize MPCC Byte Count and Word Length No CRC Operating Mode
Set Transmitter Start of Message
Turn On Transmitter and Receiver
Wait for Transmitter $\overline{\text{IRQ}}$ or Receiver $\overline{\text{FIRQ}}$
$\overline{\text{FIRQ}}$?
No – $\overline{\text{IRQ}}$
Yes
Mask $\overline{\text{IRQ}}$ and $\overline{\text{FIRQ}}$ Store Volatile Registers
A
Error or End ?
Yes
No
Get Received Character and Store It
Restore Registers and Return
Error ?
Yes
No
Print Transfer Complete
Print Error Message
Stop (or Restart)
Mask $\overline{\text{IRQ}}$ and $\overline{\text{FIRQ}}$
Are Bytes All Sent ?
No
Yes
Get Next Byte and Transmit It
Clear TSOM Bit and Increment Count
Return
Set TEOM Bit and Wait for $\overline{\text{IRQ}}$ to Indicate Sent
(SYNC)
Clear TEOM Bit and Turn Off Transmitter
Return

FIGURE 6 — BOP Transmit and Receive Test Program Flowchart

AN-850

MULTI-PROCESSOR CONTROLLER USING THE MC6809E AND THE MC68120

Prepared by
David L. Ruhberg
and
Michael C. Wood
Microprocessor Applications Engineering

As the demand for system performance increases, the design engineer is faced with the task of providing additional throughput. To obtain the increased performance, system flexibility should provide for additional expansion without the need for total redesign of the existing system. Two alternatives are available to the designer in developing any microprocessor system: single processor and multi-processor. This application note investigates both alternatives and describes a basic multi-processor system using Motorola's MC6809E and MC68120.

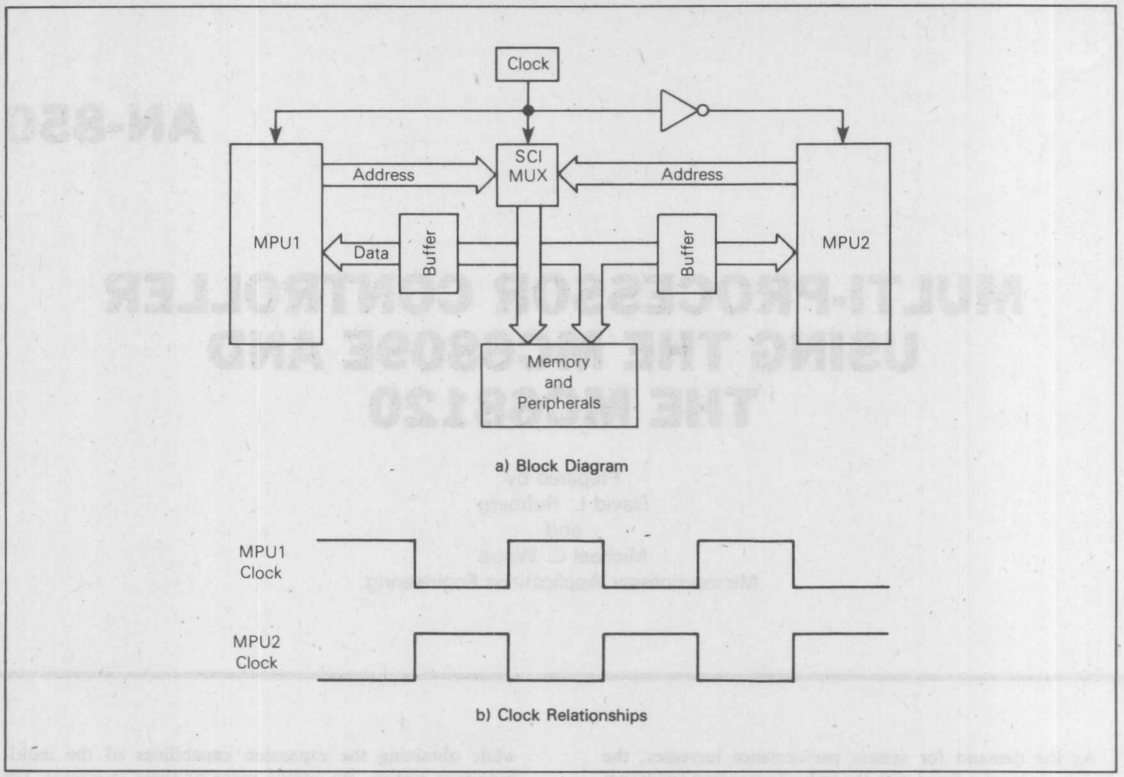
The single processor system is the more common approach in use, since one microprocessing unit (MPU) typically has been able to handle the system performance requirements. Hardware and software are both simpler with only one MPU on the bus; however, as system performance requirements continue to increase, the design engineer is faced with the job of either upgrading the system or redesigning a complete system. The characteristics of a single processor system should be reviewed before jumping into another single processor system redesign. Basically, the total growth of the single processor system is limited to the throughput rate of the MPU, so all future tasks and expansions must be taken into account at design time to avoid another complete system redesign. An MPU capable of handling all of the anticipated expansion must be selected. Thus, the MPU will not perform anywhere near its rated peak efficiency until the system is expanded. In any area where rapid system expansion is anticipated, the single processor system is a temporary solution at best.

The multi-processor configuration can eliminate the expansion problems which are present in a single MPU design. An interface containing a bus arbitrator and data transfer area common to both MPU buses could keep the buses separate and also allow the two systems to communicate. Thus, the simplicity of single bus systems can be maintained while obtaining the expansion capabilities of the multiprocessor system. By adding more of these interfaces, the system expansion occurs by simply adding peripherals to an MPU bus. Two features utilized by the Motorola MC68120 Intelligent Peripheral Controller (IPC) provide the bus arbitrator and data transfer area for a multiple MPU system just described. These features are six semaphore registers and 128 bytes of dual-ported RAM. With the MC6809E MPU operating the system bus (master) and the MC68120 containing the system bus interface, as well as the CPU controlling the local bus (slave), the system now has the best features of both the single and multi-processor approaches.

TRADITIONAL MPU MULTI-PROCESSING

One of the most common multi-processor schemes has been a bi-phase technique in which both processors operate from opposite phases of a system clock (see Figure 1). The memory and peripherals are accessed during each MPU clock high time. This scheme has the benefit of lower costs due to the presence of only one bus; however, some of the cost savings may be consumed in circuitry required to synchronize the clocks and in buffers required to prevent bus contention. In order to debug the bi-phase system, most of the hardware and software in both of the MPU systems must be working. Also, care must be taken when all resources are available to both processors, as in this bi-phase configuration, to avoid inadvertently clearing status flags or making changes in RAM. The major drawback to this system is that the system is limited to two MPUs.

The multiple bus configuration can simplify or eliminate most of the constraints and limitations of the bi-phase approach (see Figure 2) provided a simple bus arbitration scheme is available. The debugging of this type of system is simplified since one bus can operate independent of the other, except when the buses need to communicate with each



a) Block Diagram

b) Clock Relationships

FIGURE 1 — Multi-Processor Bi-Phase Technique

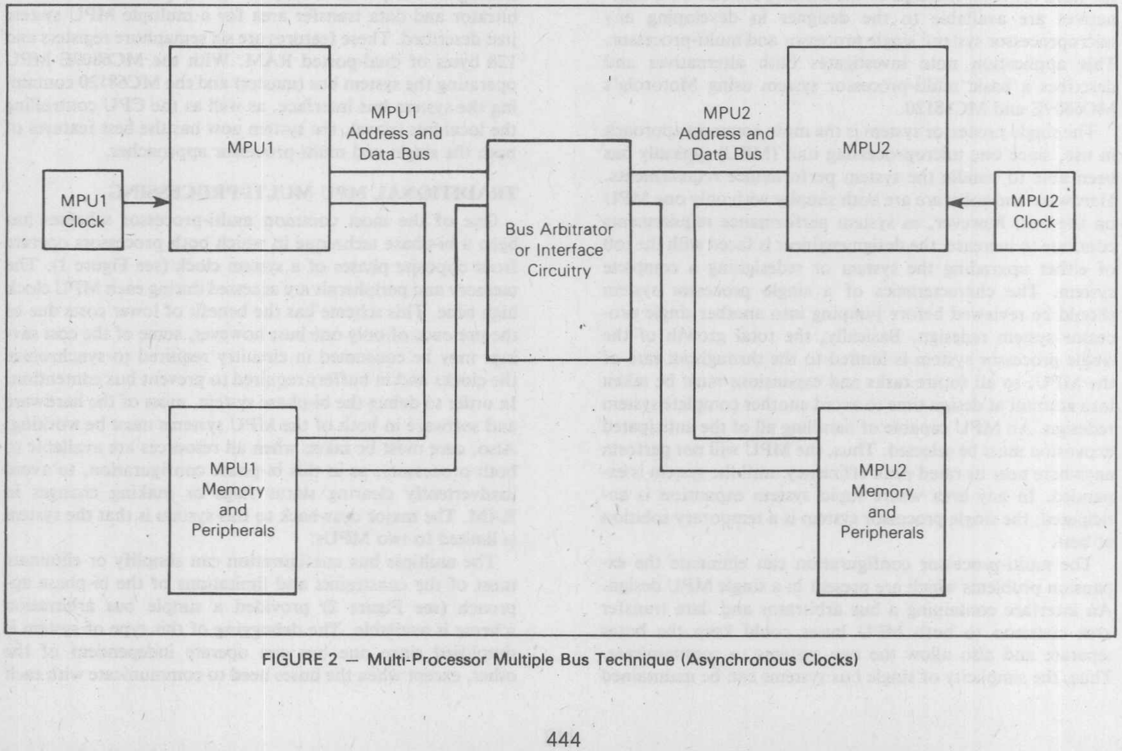


FIGURE 2 — Multi-Processor Multiple Bus Technique (Asynchronous Clocks)

other. This configuration also physically eliminates any chance of one processor accidently clearing any flags in the nonshared resources of the other system. There is no need to determine if the other processor is using the bus for more than one cycle (read-modify-write) since each processor has its own bus, thus eliminating any chance of bus contention. The bi-phase approach is limited to two processors, whereas this system is limited only by the throughput of the system (master) processor.

DESCRIPTION OF THE BASIC SYSTEM

Using the multiple bus scheme, the MC6809E-MC68120 multi-processor pair can be used in many different applications. One particular application could be a system in which the multi-processor pair is responsible for holding the pressure and temperature in a given system within certain limits (see Figure 3). To simplify matters, the application discussed here concentrates only on the MC6809E and MC68120 interface.

HARDWARE

The MC6809E MPU is one of the most advanced 8-bit microprocessor units on the market today. The M6809E (see Figure 4) contains two 16-bit index registers, two 16-bit indexable stack pointers, two 8-bit accumulators (which can be concatenated to form one 16-bit accumulator), and a direct page register that allows the direct addressing mode to be used throughout memory.

The basic instructions of any computer are greatly enhanced by the presence of powerful addressing modes. The M6809E has one of the most complete sets of addressing modes available on any microprocessor today. For example, the M6809E contains 59 basic instructions; however, due to these addressing modes, the M6809E will recognize 1464 different variations of the basic instructions. It features an external clock input which facilitates synchronizing the processor to an overall multi-processor system. Other hardware features include three-state control (TSC) inputs for control of internal bus buffers and the advanced valid memory address (AVMA) allows efficient use of common resources in a multi-processor system. Two outputs which facilitate multiprocessor configurations are the last instruction cycle (LIC) output and the BUSY output. The LIC output indicates when an opcode fetch will occur. The BUSY output is a status line that indicates the need to hold off the bus transfer for the next bus cycle. The M6809E also contains three prioritized interrupts ($\overline{NMI}$, $\overline{IRQ}$, $\overline{FIRQ}$) and a SYNC acknowledge output which allows synchronization to an external event. These features make the MC6809E an easy MPU to incorporate into a multi-processor system.

The MC68120 Intelligent Peripheral Controller (IPC) is a general purpose mask-programmable peripheral controller

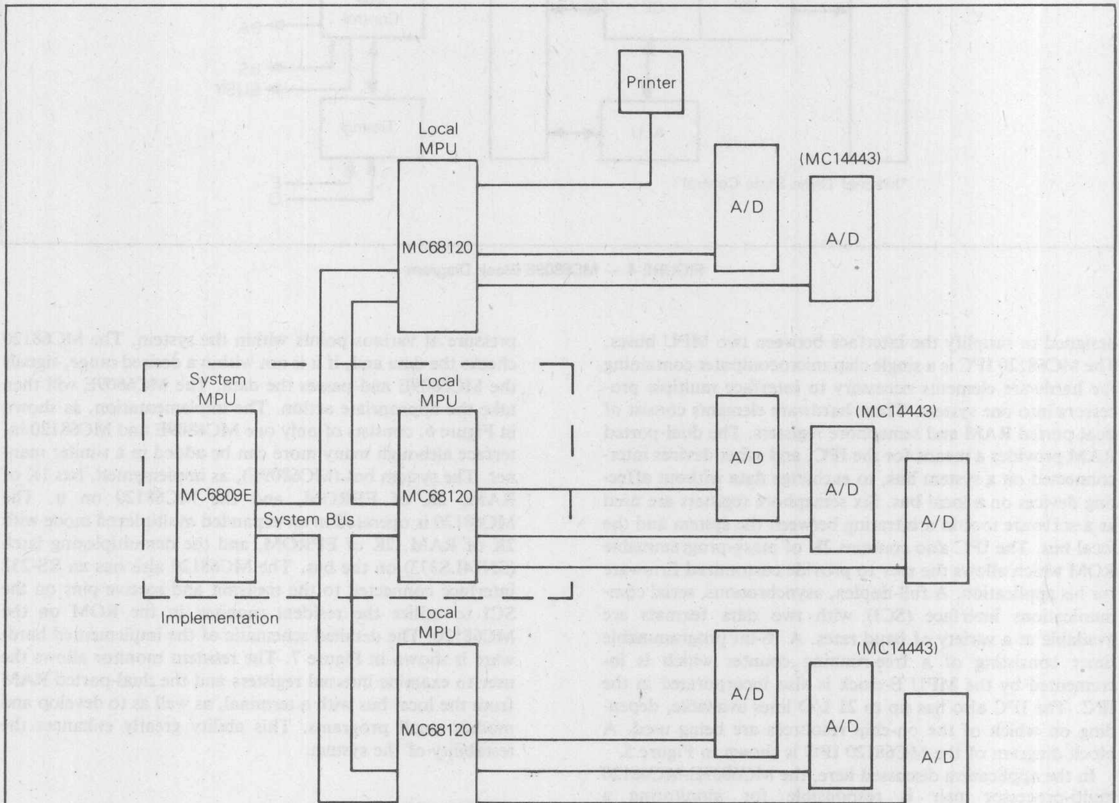


FIGURE 3 — Typical System Configuration

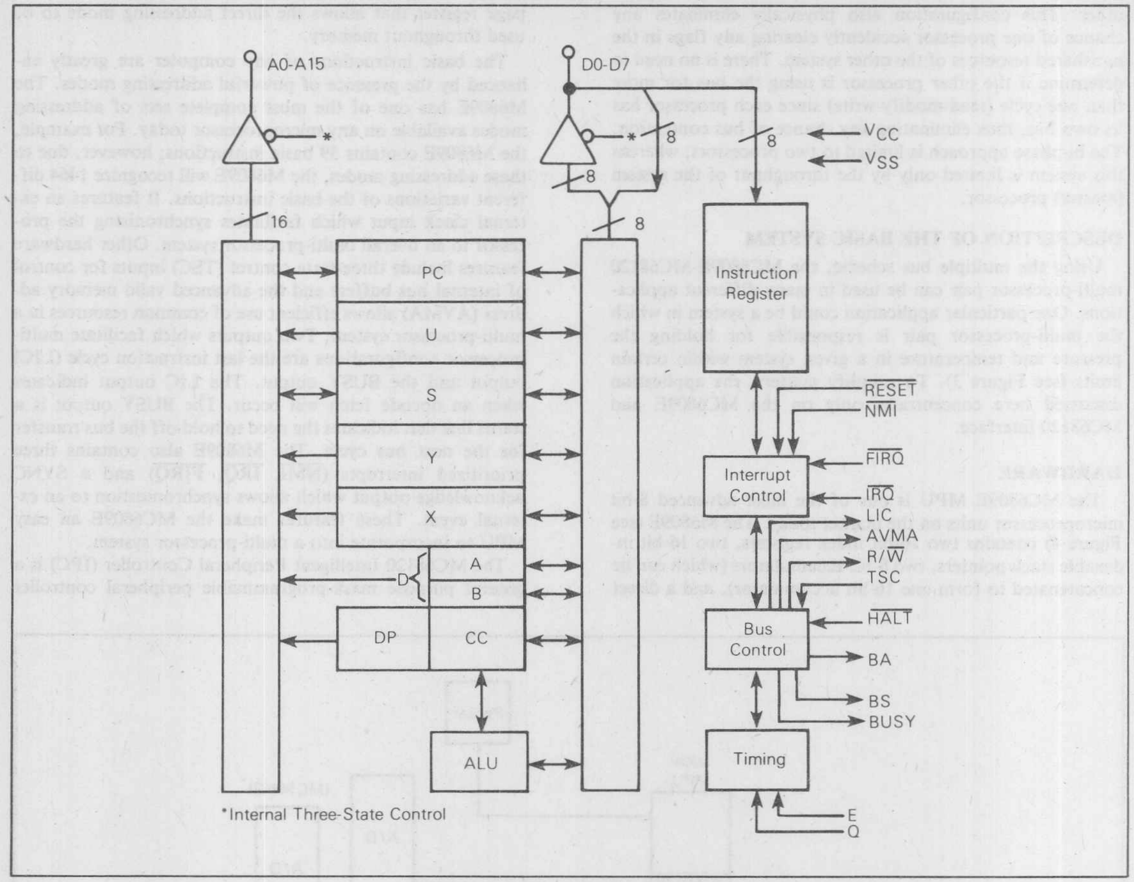


FIGURE 4 — MC6809E Block Diagram

designed to simplify the interface between two MPU buses. The MC68120 IPC is a single chip microcomputer containing the hardware elements necessary to interface multiple processors into one system. These hardware elements consist of dual-ported RAM and semaphore registers. The dual-ported RAM provides a means for the IPC, and other devices interconnected on a system bus, to exchange data without affecting devices on a local bus. Six semaphore registers are used as a software tool in arbitrating between the system and the local bus. The IPC also contains 2K of mask-programmable ROM which allows the user to provide customized firmware for his application. A full-duplex, asynchronous, serial communications interface (SCI) with two data formats are available at a variety of baud rates. A 16-bit programmable timer consisting of a free-running counter which is incremented by the MPU E-clock is also incorporated in the IPC. The IPC also has up to 21 I/O lines available, depending on which of the on-chip resources are being used. A block diagram of the MC68120 IPC is shown in Figure 5.

In the application discussed here, the MC6809E-MC68120 multi-processor pair is responsible for monitoring a temperature and pressure sensitive system and holding it within safe operating limits. The MC68120 is responsible for monitoring the analog-to-digital (A/D) converters (such as Motorola's MC14443) which reflect the temperature and pressure at various points within the system. The MC68120 checks the data and, if it is not within a desired range, signals the MC6809E and passes the data. The MC6809E will then take the appropriate action. The implementation, as shown in Figure 6, consists of only one MC6809E and MC68120 interface although many more can be added in a similar manner. The system bus (MC6809E), as implemented, has 1K of RAM, 2K of EPROM, and the MC68120 on it. The MC68120 is operated in an expanded multiplexed mode with 2K of RAM, 2K of EPROM, and the demultiplexing latch (SN74LS373) on the bus. The MC68120 also has an RS-232 interface connected to the transmit and receive pins on the SCI to utilize the resident monitor in the ROM on the MC68120. The detailed schematic of the implemented hardware is shown in Figure 7. The resident monitor allows the user to examine internal registers and the dual-ported RAM from the local bus with a terminal, as well as to develop and modify small programs. This ability greatly enhances the testability of the system.

SOFTWARE

The software needed for transfer of information in the multi-processor system is made much easier with the use of the semaphore registers and dual-ported RAM, located in the

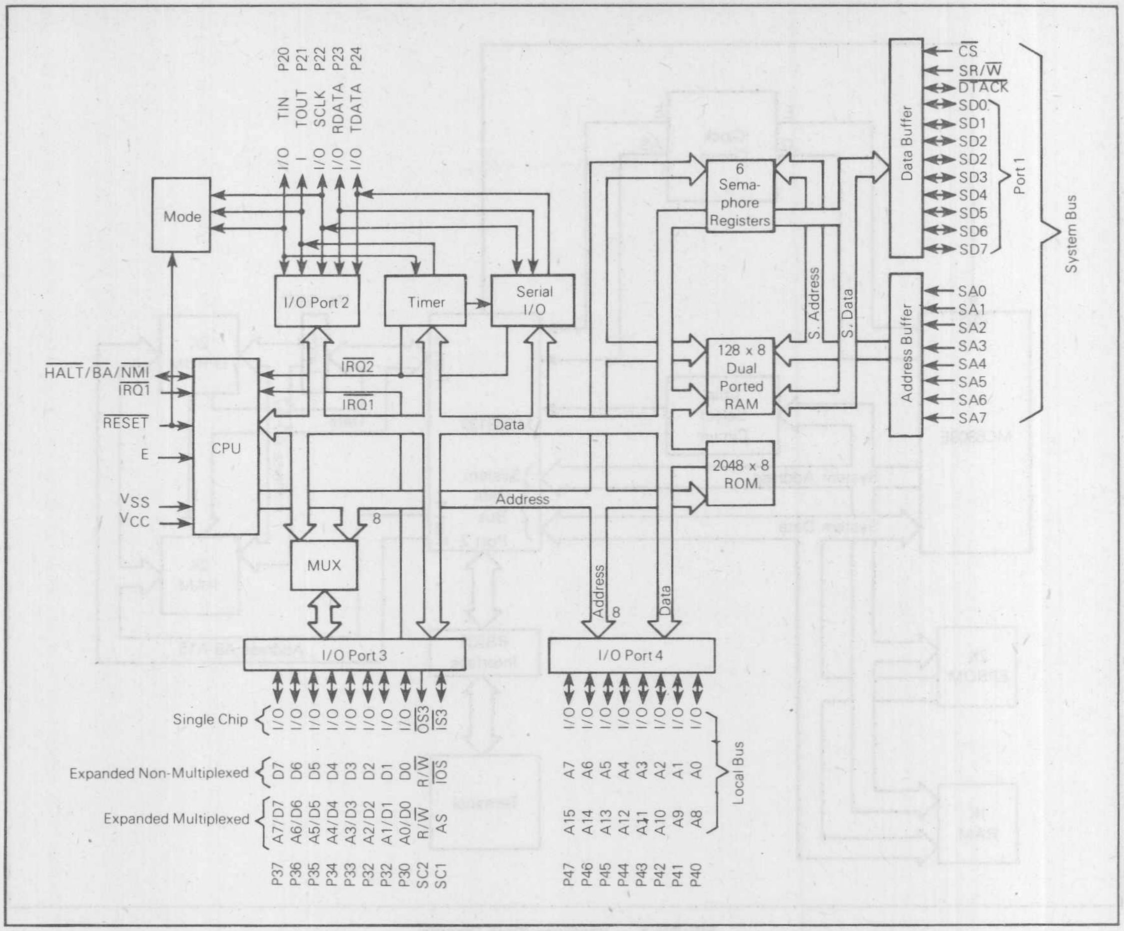


FIGURE 5 — MC68120 Block Diagram

IPC. The dual-ported RAM provides a vehicle for transferring data between a system and local bus while keeping each bus isolated. Semaphore registers are provided as a software tool to arbitrate between shared resources such as the dual-ported RAM or peripheral devices. The semaphore registers may also be used to indicate that a task is in process or has been completed.

Each semaphore register (as shown below) consists of a semaphore bit (SEM, bit 7) and an ownership bit (OWN, bit 6). The remaining six bits (b0-b5) are not used and when read, will read zeroes. The semaphore bits are test and set bits with hardware arbitration during simultaneous accesses. Basically, the semaphore bit is cleared when written and set when read during a single processor access.

b7	b6	b5	b4	b3	b2	b1	b0
SEM	OWN	0	0	0	0	0	0

Semaphore Register

A single processor semaphore bit truth table is shown below. During a write to a semaphore register, the data is disregarded and the semaphore bit is cleared. However, during a read, the data read from the semaphore bit can be interpreted as: 0 — resources are available, 1 — resources are not available. Thus a write to any semaphore register clears the semaphore bit and makes the associated resources "available."

Org. Sem Bit	R/W	Data Read	Resulting Sem Bit
0	R	0	1
1	R	1	1
0	W	–	0
1	W	–	0

Single Processor Semaphore Bit Truth Table

In passing data from the IPC to a system processor through the dual-ported RAM, the semaphore registers can be used to indicate to the system processor that data is ready. The system processor can poll, for example, on semaphore 1 and when data is ready, the IPC CPU will write to semaphore 1, thus clearing the semaphore bit. A simple polling routine for the system processor is shown below. The system processor will always read a 1 in the semaphore bit of semaphore register 1 until semaphore register 1 is written to by the IPC CPU. This will clear the semaphore bit and

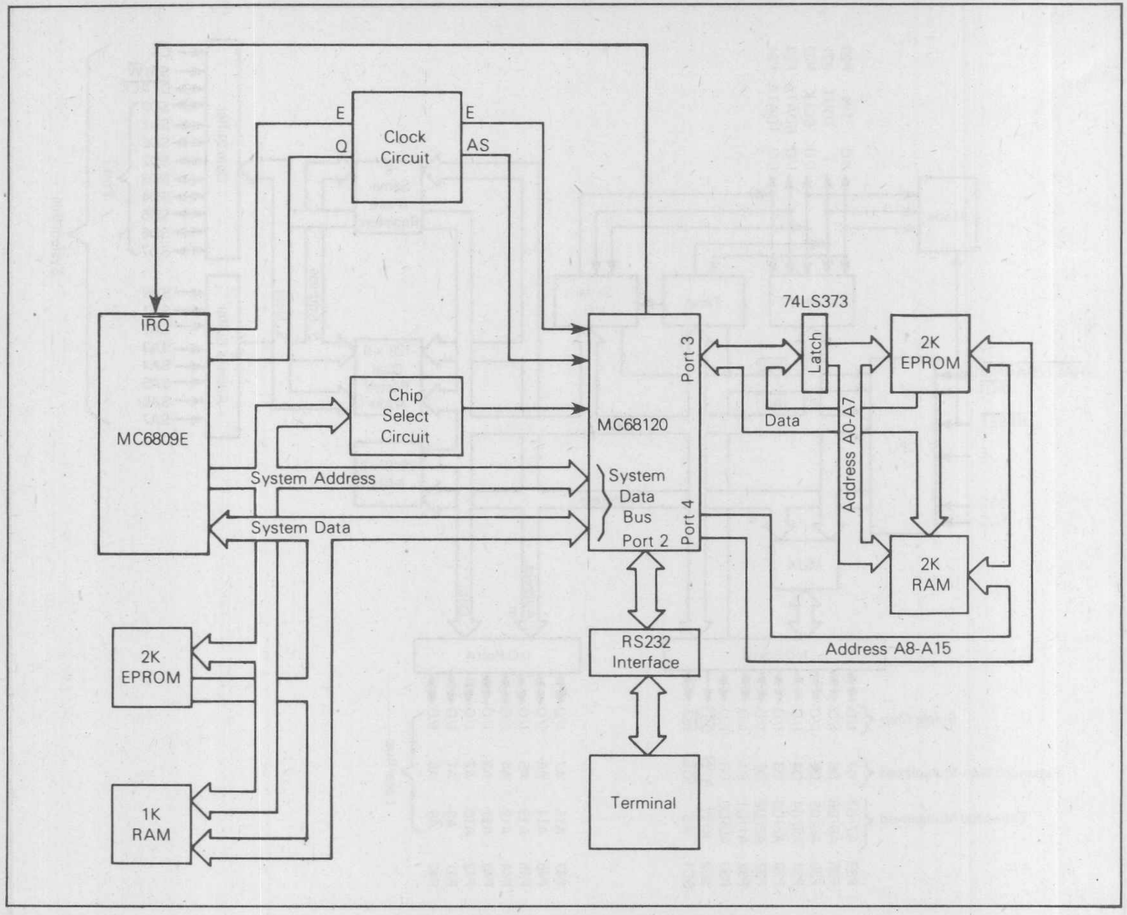


FIGURE 6 — Hardware Block Diagram

cause the system processor to jump to a program and get data.

```
LOOP        LDA         SEMPH1
            ANDA        #$80
            BNE         LOOP
            BSR         GETDATA
```

Polling Routine

It may now be necessary for the IPC CPU to determine if the system processor reads the data from the dual-ported RAM in case more data needs to be sent. Another semaphore register could be dedicated for this purpose or the same semaphore register could be used again. Timing complications could arise when reads and writes of the same semaphore register are occurring from both buses. For example, if the IPC CPU wrote to semaphore 1 to clear the semaphore bit and then polls on semaphore 1, the IPC could set the semaphore bit before the system processor detected it as clear. Therefore, to avoid an inadvertent set, a delay must be incorporated in the program between the read and write of the semaphore to guarantee that the semaphore bit was detected clear by the system processor.

In token-passing applications, the ownership bits can be used to simplify the procedure. The ownership bit is a read-only bit that indicates which processor set the semaphore bit. When the semaphore bit is set, the ownership bit indicates which processor set it. When the semaphore bit is not set, the ownership bit indicates which processor last set the semaphore bit: OWN = 0, the other processor set it; OWN = 1, this processor set SEM. After reset, all semaphores are set and the IPC owns all of them except semaphore 2 which the system processor owns.

As mentioned earlier in the hardware section, this MC6809E-MC68120 system monitors the temperature and pressure in a typical system. Basically, the MC68120 accumulates and monitors the data. The data is transferred to the MC6809E either when the MC6809E requests it, at the end of 12 hours, or if the data is out of the desired range. The CPU on the local bus is responsible for reading the data from the A/D converters every 15 seconds. In this software, it is assumed that the data is formatted in such a way that both the temperature and pressure are available in one byte of data as shown below.

MSB	LSB
TEMP	PRES

One Byte of Data from A/D Converter

The 15 seconds are measured by using the internal timer of the MC68120. The timer sets a flag every 50 ms and after the

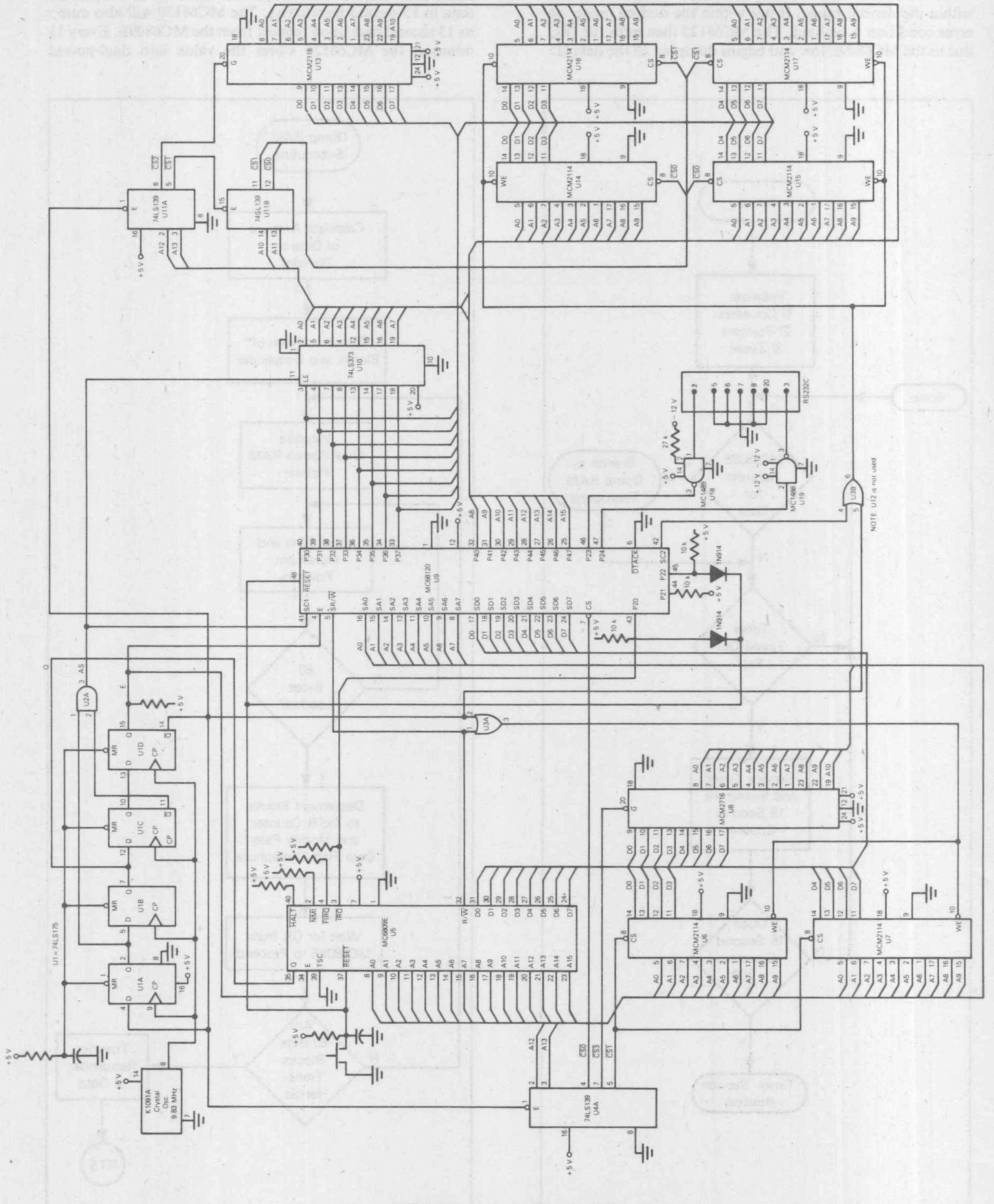


FIGURE 7 — Detailed System Schematic

flag has been set 300 times, the data is read. After the data is read, it is stored in RAM and checked to determine if it is within the desired range. If not within the desired range, an error condition is realized. The MC68120 then pulls the $\overline{IRQ}$ line to the MC6809E low and begins dumping all the data (15 second increments) to the MC6809E through the dual-ported RAM ($B0-EB). The MC68120 can hold up to 8 hours of data in 15 second increments. The MC68120 will also dump its 15 second data upon request from the MC6809E. Every 15 minutes, the MC68120 stores the value into dual-ported

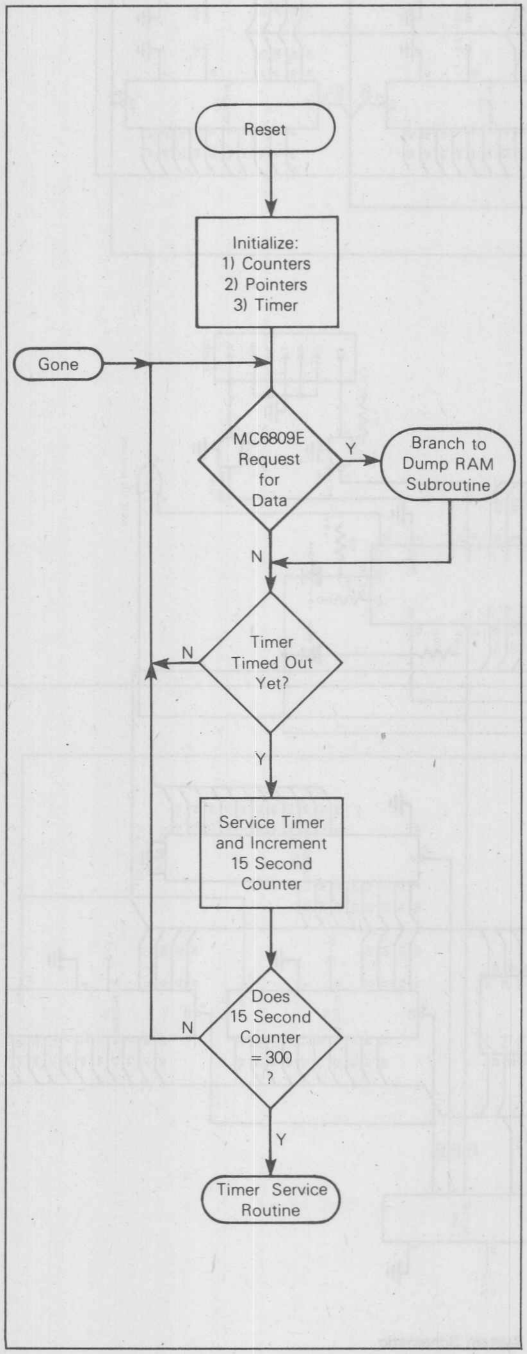


FIGURE 8 — MC68120 Flowchart

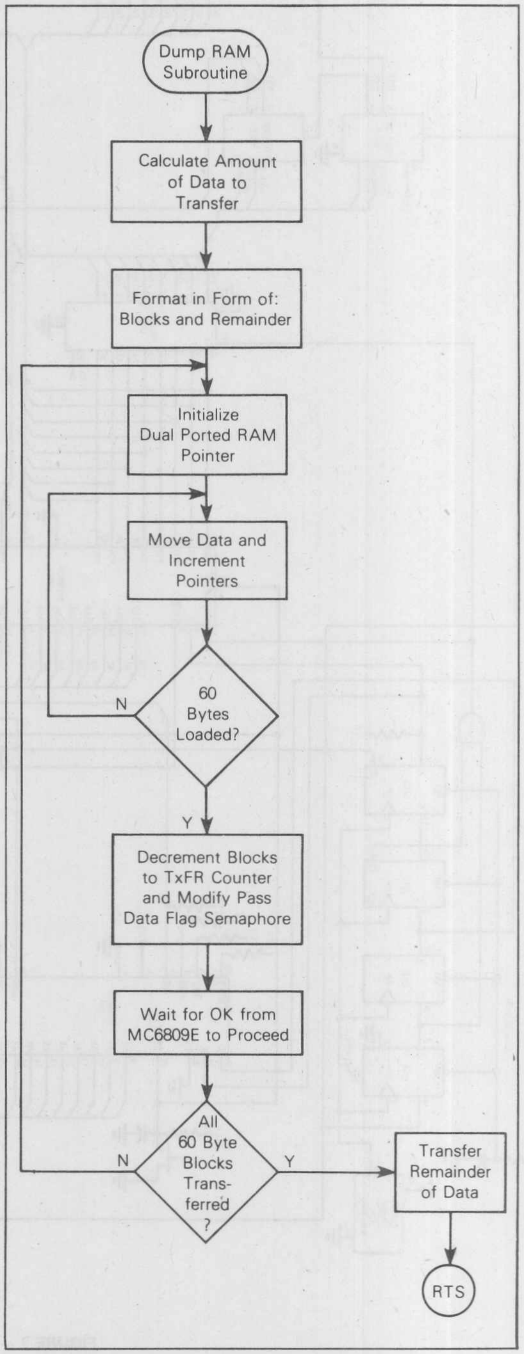

Dump RAM Subroutine
Calculate Amount of Data to Transfer
Format in Form of: Blocks and Remainder
Initialize Dual Ported RAM Pointer
Move Data and Increment Pointers
60 Bytes Loaded?
N
Y
Decrement Blocks to TxFR Counter and Modify Pass Data Flag Semaphore
Wait for OK from MC6809E to Proceed
All 60 Byte Blocks Trans-ferred ?
N
Y
Transfer Remainder of Data
RTS

FIGURE 8 — MC68120 Flowchart (Continued)

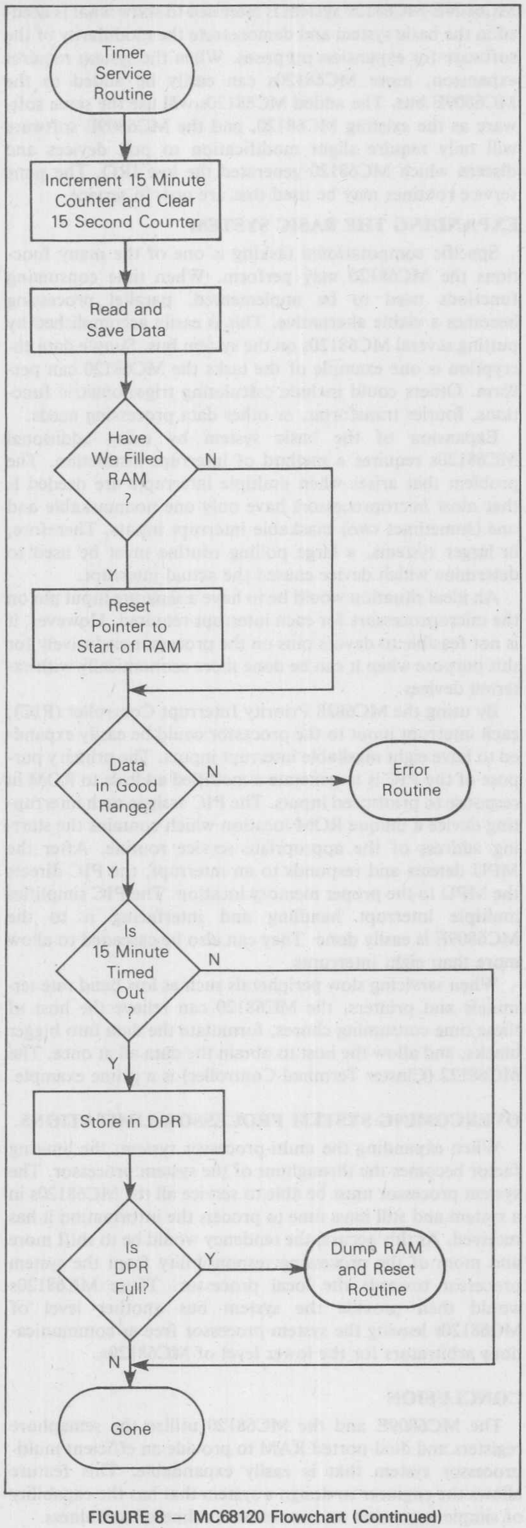


FIGURE 8 — MC68120 Flowchart (Continued)

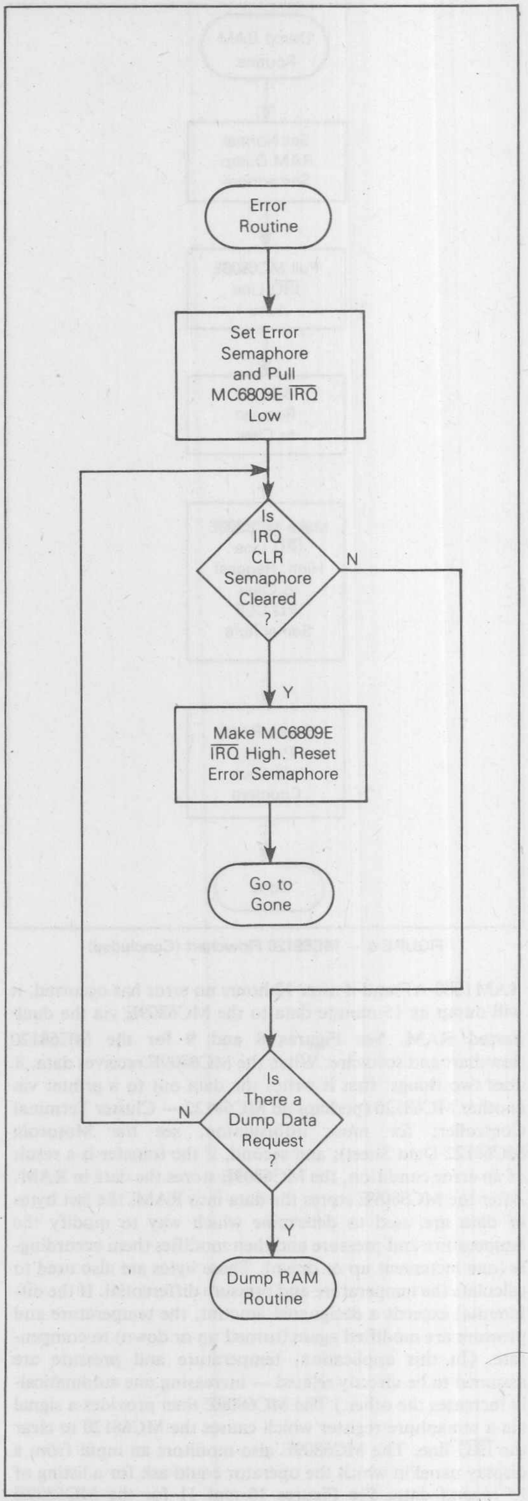
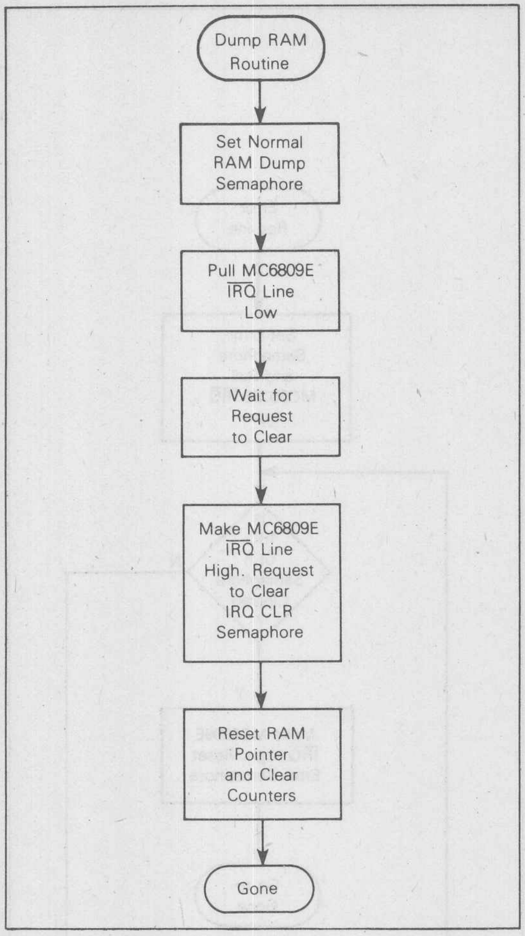

Error Routine
Set Error Semaphore and Pull MC6809E IRQ Low
Is IRQ CLR Semaphore Cleared ?
N
Y
Make MC6809E IRQ High. Reset Error Semaphore
Go to Gone
Is There a Dump Data Request ?
N
Y
Dump RAM Routine

FIGURE 8 — MC68120 Flowchart (Continued)



Dump RAM Routine
Set Normal RAM Dump Semaphore
Pull MC6809E $\overline{IRQ}$ Line Low
Wait for Request to Clear
Make MC6809E $\overline{IRQ}$ Line High. Request to Clear IRQ CLR Semaphore
Reset RAM Pointer and Clear Counters
Gone

FIGURE 8 — MC68120 Flowchart (Concluded)

RAM ($80-AF)and if after 12 hours no error has occurred, it will dump its 15 minute data to the MC6809E via the dual-ported RAM. See Figures 8 and 9 for the MC68120 flowchart and software. When the MC6809E receives data, it does two things: first it writes the data out to a printer via another MC68120 (perhaps an MC68122 — Cluster Terminal Controller; for more information, see the Motorola MC68122 Data Sheet); and second, if the transfer is a result of an error condition, the MC6809E stores the data in RAM. After the MC6809E stores the data into RAM, the last bytes of data are used to determine which way to modify the temperature and pressure and then modifies them accordingly (one increment up or down). These bytes are also used to calculate the temperature and pressure differential. If the differential exceeds a designated amount, the temperature and pressure are modified again (turned up or down) to compensate. (In this application, temperature and pressure are assumed to be directly related — increasing one automatically increases the other.) The MC6809E then provides a signal via a semaphore register which causes the MC68120 to clear the $\overline{IRQ}$ line. The MC6809E also monitors an input from a display panel in which the operator could ask for a listing of 15 second data. See Figures 10 and 11 for the MC6809E flowchart and software. The implemented portion of the MC6809E-MC68120 system is intended to show what is needed in the basic system and demonstrate the modularity of the software for expansion purposes. When the system requires expansion, more MC68120s can easily be added to the MC6809E bus. The added MC68120s will use the same software as the existing MC68120, and the MC6809E software will only require slight modification to poll devices and discern which MC68120 generated the low $\overline{IRQ}$. The same service routines may be used that are now in service.

EXPANDING THE BASIC SYSTEM

Specific computational tasking is one of the many functions the MC68120 may perform. When time consuming functions need to be implemented, parallel processing becomes a viable alternative. This is easily accomplished by putting several MC68120s on the system bus. Simple data encryption is one example of the tasks the MC68120 can perform. Others could include calculating trigonometric functions, fourier transforms, or other data processing needs.

Expansion of the basic system by using additional MC68120s requires a method of interrupt distinction. The problem that arises when multiple interrupts are needed is that most microprocessors have only one nonmaskable and one (sometimes two) maskable interrupt inputs. Therefore, in larger systems, a large polling routine must be used to determine which device caused the actual interrupt.

An ideal situation would be to have a separate input pin on the microprocessors for each interrupt required. However, it is not feasible to devote pins on the processor exclusively for this purpose when it can be done more economically with external devices.

By using the MC6828 Priority Interrupt Controller (PIC), each interrupt input to the processor could be easily expanded to have eight maskable interrupt inputs. The primary purpose of the PIC is to generate a modified address to ROM in response to prioritized inputs. The PIC assigns each interrupting device a unique ROM location which contains the starting address of the appropriate service routine. After the MPU detects and responds to an interrupt, the PIC directs the MPU to the proper memory location. The PIC simplifies multiple interrupt handling and interfacing it to the MC6809E is easily done. They can also be cascaded to allow more than eight interrupts.

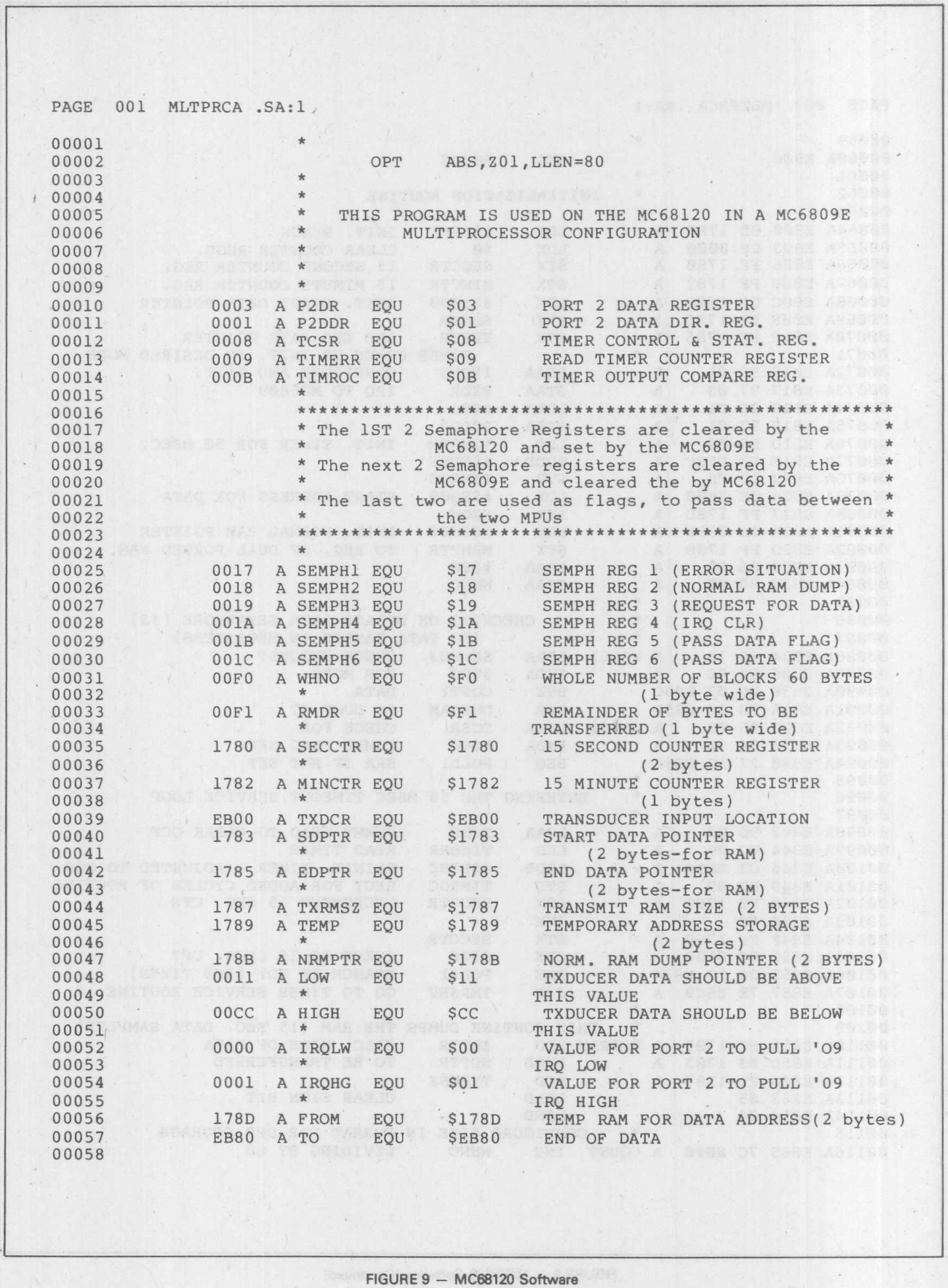
When servicing slow peripherals such as low baud rate terminals and printers, the MC68120 can relieve the host of these time consuming chores, formulate the data into bigger blocks, and allow the host to obtain the data all at once. The MC68122 (Cluster Terminal Controller) is a prime example.

OVERCOMING SYSTEM PROCESSOR LIMITATIONS

When expanding the multi-processor system, the limiting factor becomes the throughput of the system processor. The system processor must be able to service all the MC68120s in a system and still have time to process the information it has received. As this occurs, the tendency would be to shift more and more of the processing responsibility from the system processor towards the local processor. These MC68120s would then provide the system bus another level of MC68120s leaving the system processor free as communications arbitrators for the lower level of MC68120s.

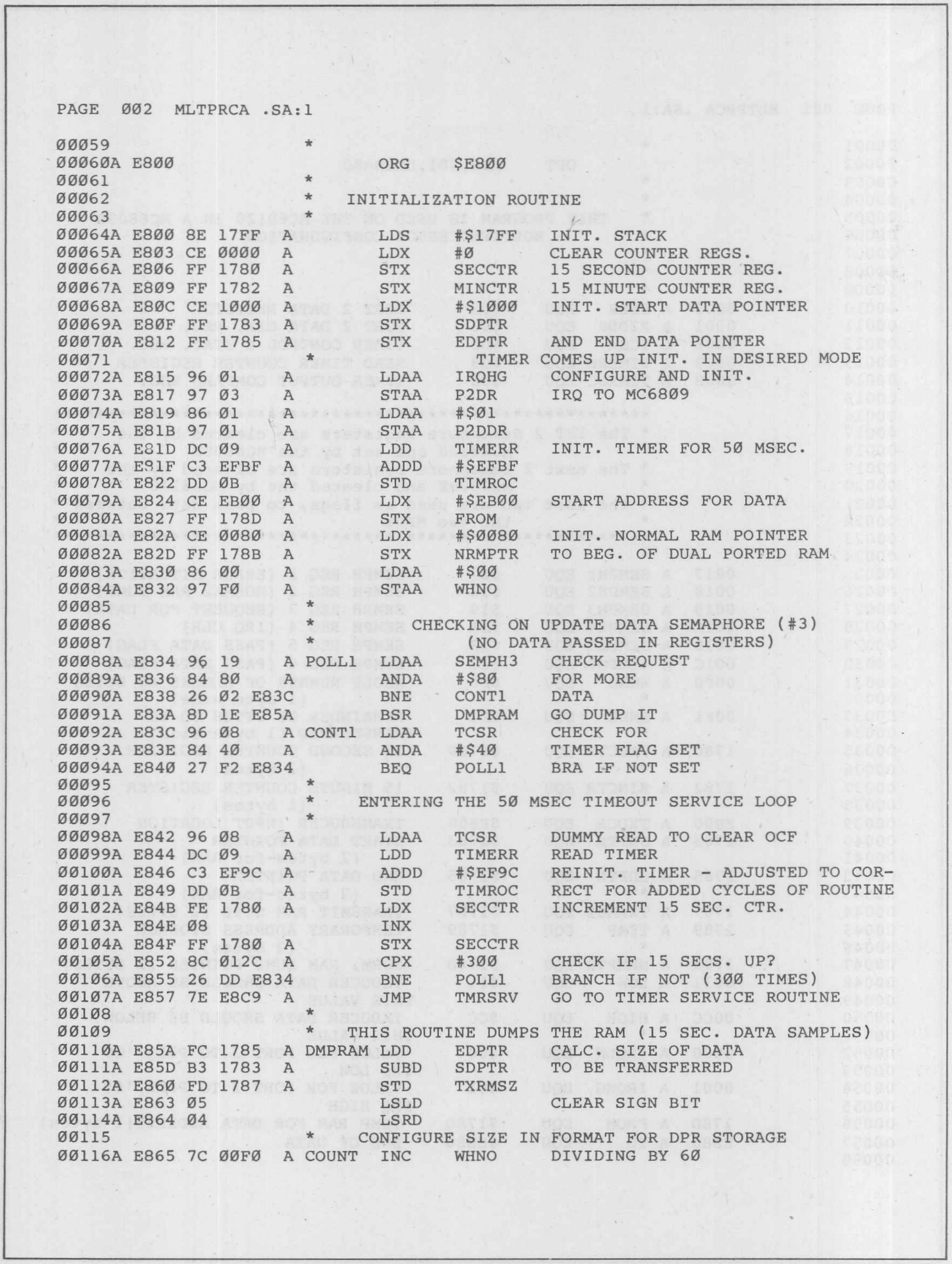
CONCLUSION

The MC6809E and the MC68120 utilize the semaphore registers and dual-ported RAM to provide an efficient multiprocessor system that is easily expandable. This feature allows the engineer to design a system that has the capability of simple expansion and increases its time of usefulness.



```
PAGE  001  MLTPRCA .SA:1

        *
               OPT    ABS,Z01,LLEN=80
        *
        *
        *   THIS PROGRAM IS USED ON THE MC68120 IN A MC6809E
        *         MULTIPROCESSOR CONFIGURATION
        *
        *
0003  A P2DR   EQU    $03      PORT 2 DATA REGISTER
0001  A P2DDR  EQU    $01      PORT 2 DATA DIR. REG.
0008  A TCSR   EQU    $08      TIMER CONTROL & STAT. REG.
0009  A TIMERR EQU    $09      READ TIMER COUNTER REGISTER
000B  A TIMROC EQU    $0B      TIMER OUTPUT COMPARE REG.
        *
        *******************************************************
        * The 1ST 2 Semaphore Registers are cleared by the  *
        *           MC68120 and set by the MC6809E            *
        * The next 2 Semaphore registers are cleared by the *
        *           MC6809E and cleared the by MC68120        *
        * The last two are used as flags, to pass data between *
        *            the two MPUs                             *
        *******************************************************
        *
0017  A SEMPH1 EQU    $17      SEMPH REG 1 (ERROR SITUATION)
0018  A SEMPH2 EQU    $18      SEMPH REG 2 (NORMAL RAM DUMP)
0019  A SEMPH3 EQU    $19      SEMPH REG 3 (REQUEST FOR DATA)
001A  A SEMPH4 EQU    $1A      SEMPH REG 4 (IRQ CLR)
001B  A SEMPH5 EQU    $1B      SEMPH REG 5 (PASS DATA FLAG)
001C  A SEMPH6 EQU    $1C      SEMPH REG 6 (PASS DATA FLAG)
00F0  A WHNO   EQU    $F0      WHOLE NUMBER OF BLOCKS 60 BYTES
        *                               (1 byte wide)
00F1  A RMDR   EQU    $F1      REMAINDER OF BYTES TO BE
        *                     TRANSFERRED (1 byte wide)
1780  A SECCTR EQU    $1780    15 SECOND COUNTER REGISTER
        *                               (2 bytes)
1782  A MINCTR EQU    $1782    15 MINUTE COUNTER REGISTER
        *                               (1 bytes)
EB00  A TXDCR  EQU    $EB00    TRANSDUCER INPUT LOCATION
1783  A SDPTR  EQU    $1783    START DATA POINTER
        *                          (2 bytes-for RAM)
1785  A EDPTR  EQU    $1785    END DATA POINTER
        *                          (2 bytes-for RAM)
1787  A TXRMSZ EQU    $1787    TRANSMIT RAM SIZE (2 BYTES)
1789  A TEMP   EQU    $1789    TEMPORARY ADDRESS STORAGE
        *                               (2 bytes)
178B  A NRMPTR EQU    $178B    NORM. RAM DUMP POINTER (2 BYTES)
0011  A LOW    EQU    $11      TXDUCER DATA SHOULD BE ABOVE
        *                     THIS VALUE
00CC  A HIGH   EQU    $CC      TXDUCER DATA SHOULD BE BELOW
        *                     THIS VALUE
0000  A IRQLW  EQU    $00      VALUE FOR PORT 2 TO PULL '09
        *                     IRQ LOW
0001  A IRQHG  EQU    $01      VALUE FOR PORT 2 TO PULL '09
        *                     IRQ HIGH
178D  A FROM   EQU    $178D    TEMP RAM FOR DATA ADDRESS(2 bytes)
EB80  A TO     EQU    $EB80    END OF DATA

```

FIGURE 9 — MC68120 Software

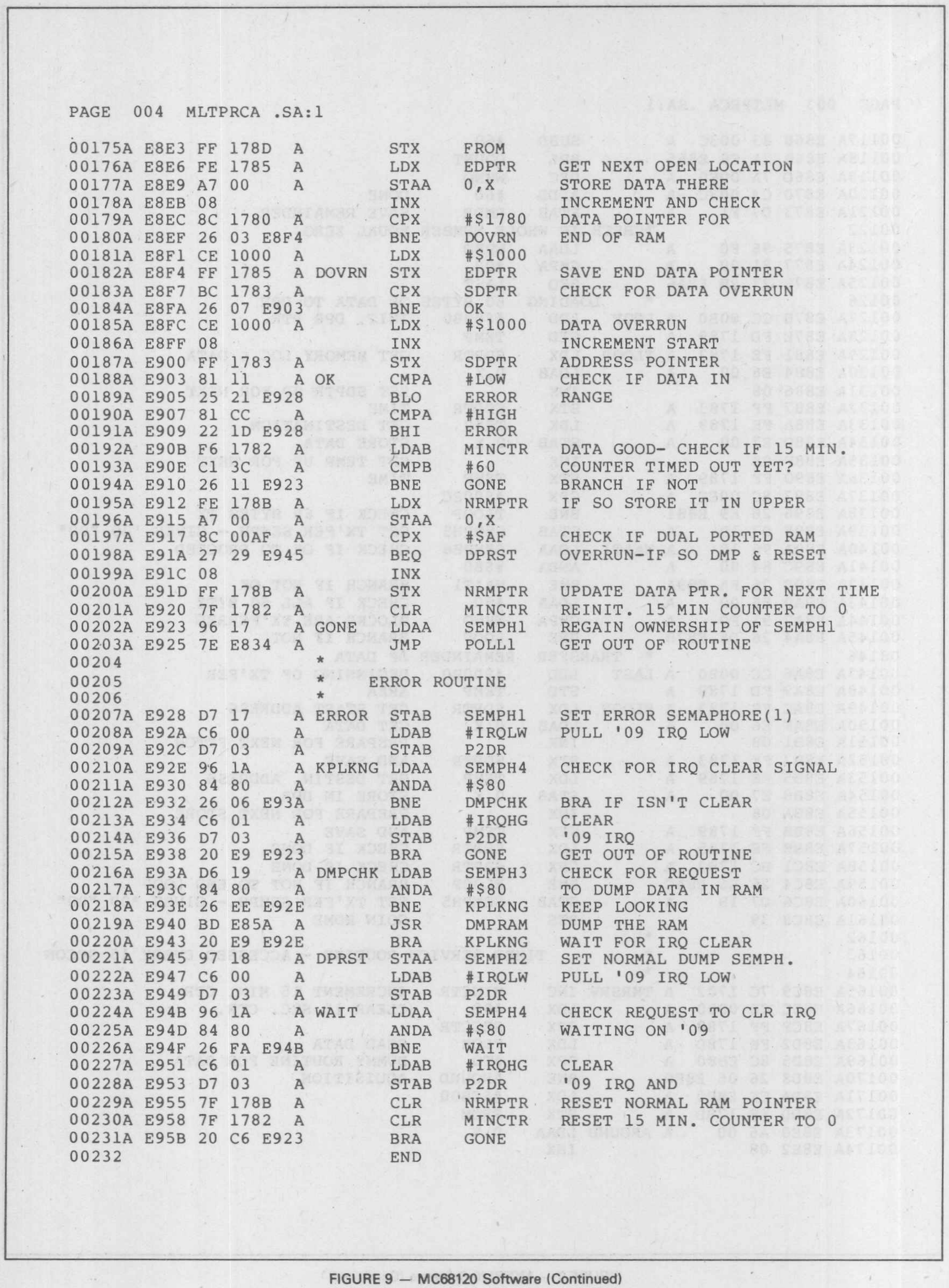


```
PAGE  ØØ2  MLTPRCA .SA:1

ØØØ59                          *
ØØØ6ØA E8ØØ                          ORG    $E8ØØ
ØØØ61                          *
ØØØ62                          *   INITIALIZATION ROUTINE
ØØØ63                          *
ØØØ64A E8ØØ 8E 17FF  A               LDS    #$17FF    INIT. STACK
ØØØ65A E8Ø3 CE ØØØØ  A               LDX    #Ø        CLEAR COUNTER REGS.
ØØØ66A E8Ø6 FF 178Ø  A               STX    SECCTR    15 SECOND COUNTER REG.
ØØØ67A E8Ø9 FF 1782  A               STX    MINCTR    15 MINUTE COUNTER REG.
ØØØ68A E8ØC CE 1ØØØ  A               LDX    #$1ØØØ    INIT. START DATA POINTER
ØØØ69A E8ØF FF 1783  A               STX    SDPTR
ØØØ7ØA E812 FF 1785  A               STX    EDPTR     AND END DATA POINTER
ØØØ71                          *           TIMER COMES UP INIT. IN DESIRED MODE
ØØØ72A E815 96 Ø1    A               LDAA   IRQHG     CONFIGURE AND INIT.
ØØØ73A E817 97 Ø3    A               STAA   P2DR      IRQ TO MC68Ø9
ØØØ74A E819 86 Ø1    A               LDAA   #$Ø1
ØØØ75A E81B 97 Ø1    A               STAA   P2DDR
ØØØ76A E81D DC Ø9    A               LDD    TIMERR    INIT. TIMER FOR 5Ø MSEC.
ØØØ77A E81F C3 EFBF  A               ADDD   #$EFBF
ØØØ78A E822 DD ØB    A               STD    TIMROC
ØØØ79A E824 CE EBØØ  A               LDX    #$EBØØ    START ADDRESS FOR DATA
ØØØ8ØA E827 FF 178D  A               STX    FROM
ØØØ81A E82A CE ØØ8Ø  A               LDX    #$ØØ8Ø    INIT. NORMAL RAM POINTER
ØØØ82A E82D FF 178B  A               STX    NRMPTR    TO BEG. OF DUAL PORTED RAM
ØØØ83A E83Ø 86 ØØ    A               LDAA   #$ØØ
ØØØ84A E832 97 FØ    A               STAA   WHNO
ØØØ85                          *
ØØØ86                          *      CHECKING ON UPDATE DATA SEMAPHORE (#3)
ØØØ87                          *          (NO DATA PASSED IN REGISTERS)
ØØØ88A E834 96 19    A POLL1   LDAA   SEMPH3    CHECK REQUEST
ØØØ89A E836 84 8Ø    A               ANDA   #$8Ø      FOR MORE
ØØØ9ØA E838 26 Ø2 E83C               BNE    CONT1     DATA
ØØØ91A E83A 8D 1E E85A               BSR    DMPRAM    GO DUMP IT
ØØØ92A E83C 96 Ø8    A CONT1   LDAA   TCSR      CHECK FOR
ØØØ93A E83E 84 4Ø    A               ANDA   #$4Ø      TIMER FLAG SET
ØØØ94A E84Ø 27 F2 E834               BEQ    POLL1     BRA IF NOT SET
ØØØ95                          *
ØØØ96                          *   ENTERING THE 5Ø MSEC TIMEOUT SERVICE LOOP
ØØØ97                          *
ØØØ98A E842 96 Ø8    A               LDAA   TCSR      DUMMY READ TO CLEAR OCF
ØØØ99A E844 DC Ø9    A               LDD    TIMERR    READ TIMER
ØØ1ØØA E846 C3 EF9C  A               ADDD   #$EF9C    REINIT. TIMER - ADJUSTED TO COR-
ØØ1Ø1A E849 DD ØB    A               STD    TIMROC    RECT FOR ADDED CYCLES OF ROUTINE
ØØ1Ø2A E84B FE 178Ø  A               LDX    SECCTR    INCREMENT 15 SEC. CTR.
ØØ1Ø3A E84E Ø8                       INX
ØØ1Ø4A E84F FF 178Ø  A               STX    SECCTR
ØØ1Ø5A E852 8C Ø12C  A               CPX    #3ØØ      CHECK IF 15 SECS. UP?
ØØ1Ø6A E855 26 DD E834               BNE    POLL1     BRANCH IF NOT (3ØØ TIMES)
ØØ1Ø7A E857 7E E8C9  A               JMP    TMRSRV    GO TO TIMER SERVICE ROUTINE
ØØ1Ø8                          *
ØØ1Ø9                          *   THIS ROUTINE DUMPS THE RAM (15 SEC. DATA SAMPLES)
ØØ11ØA E85A FC 1785  A DMPRAM  LDD    EDPTR     CALC. SIZE OF DATA
ØØ111A E85D B3 1783  A               SUBD   SDPTR     TO BE TRANSFERRED
ØØ112A E86Ø FD 1787  A               STD    TXRMSZ
ØØ113A E863 Ø5                       LSLD             CLEAR SIGN BIT
ØØ114A E864 Ø4                       LSRD
ØØ115                          *   CONFIGURE SIZE IN FORMAT FOR DPR STORAGE
ØØ116A E865 7C ØØFØ  A COUNT   INC    WHNO      DIVIDING BY 6Ø
```

FIGURE 9 — MC68120 Software (Continued)

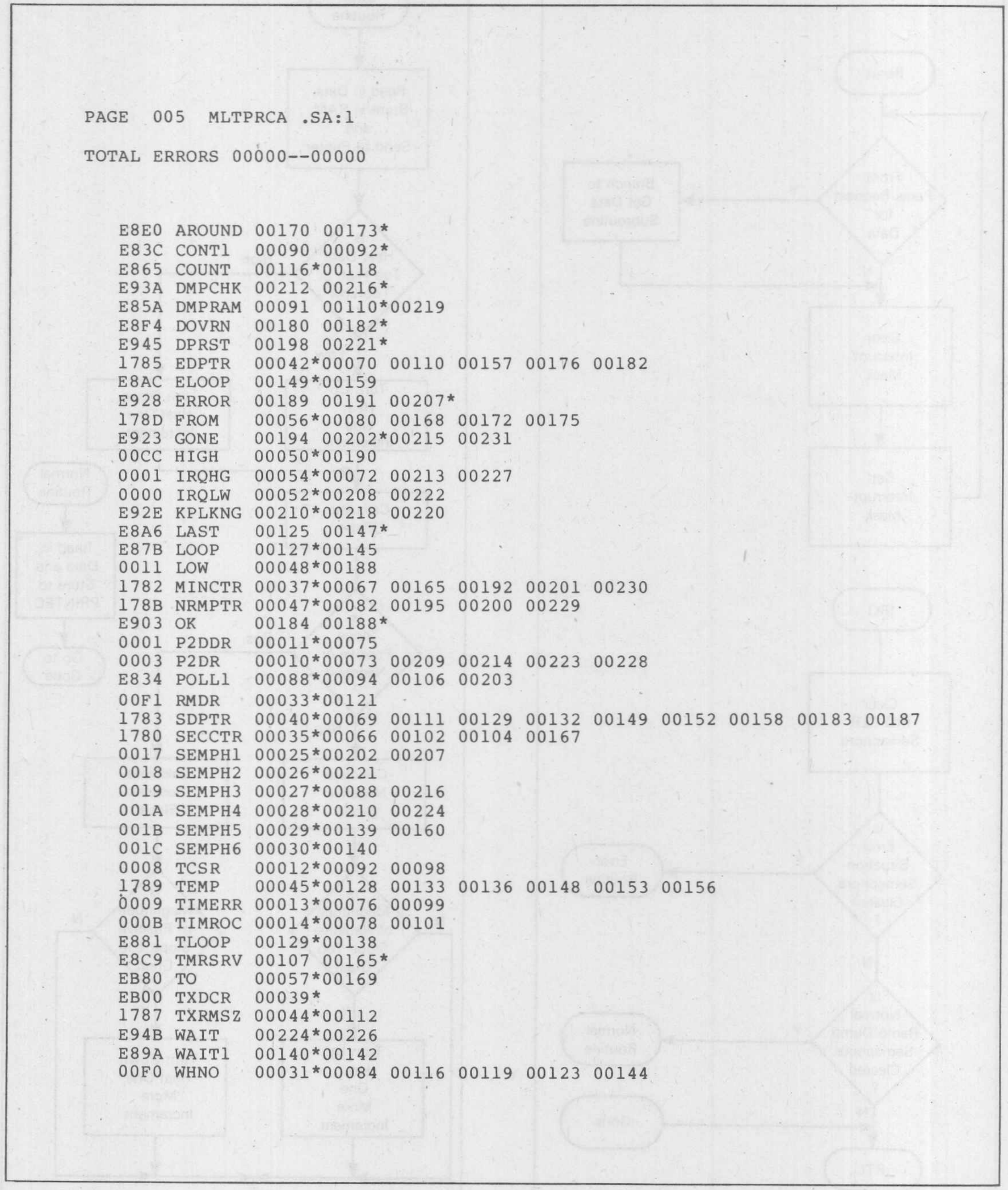
```
PAGE  003  MLTPRCA .SA:1

00117A E868 83 003C  A        SUBD   #60
00118A E86B 2A F8 E865        BPL    COUNT
00119A E86D 7A 00F0  A        DEC    WHNO
00120A E870 C3 003C  A        ADDD   #60     DONE
00121A E873 D7 F1    A        STAB   RMDR    SAVE REMAINDER
00122                 *CHECK IF WHOLE NUMBER EQUAL ZERO
00123A E875 96 F0    A        LDAA   WHNO
00124A E877 81 00    A        CMPA   #00
00125A E879 27 2B E8A6        BEQ    LAST
00126                    *  LOADING 60 BYTES OF DATA TO DPR
00127A E87B CC 00B0  A LOOP   LDD    #$00B0  INIT. DPR PTR.
00128A E87E FD 1789  A        STD    TEMP
00129A E881 FE 1783  A TLOOP  LDX    SDPTR   GET MEMORY LOC & DATA
00130A E884 E6 00    A        LDAB   0,X
00131A E886 08                INX            SET SDPTR UP FOR NEXT
00132A E887 FF 1783  A        STX    SDPTR   TIME
00133A E88A FE 1789  A        LDX    TEMP    GET DESTINATION
00134A E88D E7 00    A        STAB   0,X     STORE DATA
00135A E88F 08                INX            SET TEMP UP FOR NEXT
00136A E890 FF 1789  A        STX    TEMP    TIME
00137A E893 8C 00EC  A        CPX    #$00EC
00138A E896 26 E9 E881        BNE    TLOOP   CHECK IF 60 BYTES TX
00139A E898 D7 1B    A        STAB   SEMPH5  SET TX'FER SEMPH.- GIVES '09 "00"
00140A E89A 96 1C    A WAIT1  LDAA   SEMPH6  CHECK IF OK TO PROCEED
00141A E89C 84 80    A        ANDA   #$80
00142A E89E 26 FA E89A        BNE    WAIT1   BRANCH IF NOT OK
00143A E8A0 86 00    A        LDAA   #00     CHECK IF ALL 60 BYTE
00144A E8A2 91 F0    A        CMPA   WHNO    BLOCKS ARE TX'FERRED
00145A E8A4 26 D5 E87B        BNE    LOOP    BRANCH IF NOT
00146                    *  TRANSFER REMAINDER OF DATA
00147A E8A6 CC 00B0  A LAST   LDD    #$00B0  BEGINNING OF TX'FER
00148A E8A9 FD 1789  A        STD    TEMP    AREA
00149A E8AC FE 1783  A ELOOP  LDX    SDPTR   GET START ADDRESS
00150A E8AF E6 00    A        LDAB   0,X     GET DATA
00151A E8B1 08                INX            PREPARE FOR NEXT FETCH
00152A E8B2 FF 1783  A        STX    SDPTR   AND SAVE
00153A E8B5 FE 1789  A        LDX    TEMP    GET DESTIN. ADDRESS
00154A E8B8 E7 00    A        STAB   0,X     STORE IN DPR
00155A E8BA 08                INX            PREPARE FOR NEXT STORE
00156A E8BB FF 1789  A        STX    TEMP    AND SAVE
00157A E8BE FE 1785  A        LDX    EDPTR   CHECK IF DONE
00158A E8C1 BC 1783  A        CPX    SDPTR   CHECK IF DONE
00159A E8C4 26 E6 E8AC        BNE    ELOOP   BRANCH IF NOT TO END LOOP
00160A E8C6 D7 1B    A        STAB   SEMPH5  SET TX'FER SEMPH.- GIVES °09 "00"
00161A E8C8 39                RTS            GOIN HOME
00162                    *
00163                    *        TIMER SERVICE ROUTINE - ACCESSED EVERY 15 SECON
00164                    *
00165A E8C9 7C 1782  A TMRSRV INC    MINCTR  INCREMENT 15 MIN. CTR.
00166A E8CC CE 0000  A        LDX    #00     CLEAR 15 SEC. CTR.
00167A E8CF FF 1780  A        STX    SECCTR
00168A E8D2 FE 178D  A        LDX    FROM    READ DATA
00169A E8D5 8C EB80  A        CPX    #TO     DUMMY ROUTINE FOR DATA
00170A E8D8 26 06 E8E0        BNE    AROUND  AQUISITION
00171A E8DA CE EB00  A        LDX    #$EB00
00172A E8DD FF 178D  A        STX    FROM
00173A E8E0 A6 00    A AROUND LDAA   0,X
00174A E8E2 08                INX
```

FIGURE 9 — MC68120 Software (Continued)



```
PAGE  004  MLTPRCA .SA:1

00175A E8E3 FF 178D  A        STX    FROM
00176A E8E6 FE 1785  A        LDX    EDPTR     GET NEXT OPEN LOCATION
00177A E8E9 A7 00    A        STAA   0,X       STORE DATA THERE
00178A E8EB 08                INX              INCREMENT AND CHECK
00179A E8EC 8C 1780  A        CPX    #$1780    DATA POINTER FOR
00180A E8EF 26 03 E8F4        BNE    DOVRN     END OF RAM
00181A E8F1 CE 1000  A        LDX    #$1000
00182A E8F4 FF 1785  A DOVRN  STX    EDPTR     SAVE END DATA POINTER
00183A E8F7 BC 1783  A        CPX    SDPTR     CHECK FOR DATA OVERRUN
00184A E8FA 26 07 E903        BNE    OK
00185A E8FC CE 1000  A        LDX    #$1000    DATA OVERRUN
00186A E8FF 08                INX              INCREMENT START
00187A E900 FF 1783  A        STX    SDPTR     ADDRESS POINTER
00188A E903 81 11    A OK     CMPA   #LOW      CHECK IF DATA IN
00189A E905 25 21 E928        BLO    ERROR     RANGE
00190A E907 81 CC    A        CMPA   #HIGH
00191A E909 22 1D E928        BHI    ERROR
00192A E90B F6 1782  A        LDAB   MINCTR    DATA GOOD- CHECK IF 15 MIN.
00193A E90E C1 3C    A        CMPB   #60       COUNTER TIMED OUT YET?
00194A E910 26 11 E923        BNE    GONE      BRANCH IF NOT
00195A E912 FE 178B  A        LDX    NRMPTR    IF SO STORE IT IN UPPER
00196A E915 A7 00    A        STAA   0,X
00197A E917 8C 00AF  A        CPX    #$AF      CHECK IF DUAL PORTED RAM
00198A E91A 27 29 E945        BEQ    DPRST     OVERRUN-IF SO DMP & RESET
00199A E91C 08                INX
00200A E91D FF 178B  A        STX    NRMPTR    UPDATE DATA PTR. FOR NEXT TIME
00201A E920 7F 1782  A        CLR    MINCTR    REINIT. 15 MIN COUNTER TO 0
00202A E923 96 17    A GONE   LDAA   SEMPH1    REGAIN OWNERSHIP OF SEMPH1
00203A E925 7E E834  A        JMP    POLL1     GET OUT OF ROUTINE
00204                  *
00205                  *        ERROR ROUTINE
00206                  *
00207A E928 D7 17    A ERROR  STAB   SEMPH1    SET ERROR SEMAPHORE(1)
00208A E92A C6 00    A        LDAB   #IRQLW    PULL '09 IRQ LOW
00209A E92C D7 03    A        STAB   P2DR
00210A E92E 96 1A    A KPLKNG LDAA   SEMPH4    CHECK FOR IRQ CLEAR SIGNAL
00211A E930 84 80    A        ANDA   #$80
00212A E932 26 06 E93A        BNE    DMPCHK    BRA IF ISN'T CLEAR
00213A E934 C6 01    A        LDAB   #IRQHG    CLEAR
00214A E936 D7 03    A        STAB   P2DR      '09 IRQ
00215A E938 20 E9 E923        BRA    GONE      GET OUT OF ROUTINE
00216A E93A D6 19    A DMPCHK LDAB   SEMPH3    CHECK FOR REQUEST
00217A E93C 84 80    A        ANDA   #$80      TO DUMP DATA IN RAM
00218A E93E 26 EE E92E        BNE    KPLKNG    KEEP LOOKING
00219A E940 BD E85A  A        JSR    DMPRAM    DUMP THE RAM
00220A E943 20 E9 E92E        BRA    KPLKNG    WAIT FOR IRQ CLEAR
00221A E945 97 18    A DPRST  STAA   SEMPH2    SET NORMAL DUMP SEMPH.
00222A E947 C6 00    A        LDAB   #IRQLW    PULL '09 IRQ LOW
00223A E949 D7 03    A        STAB   P2DR
00224A E94B 96 1A    A WAIT   LDAA   SEMPH4    CHECK REQUEST TO CLR IRQ
00225A E94D 84 80    A        ANDA   #$80      WAITING ON '09
00226A E94F 26 FA E94B        BNE    WAIT
00227A E951 C6 01    A        LDAB   #IRQHG    CLEAR
00228A E953 D7 03    A        STAB   P2DR      '09 IRQ AND
00229A E955 7F 178B  A        CLR    NRMPTR    RESET NORMAL RAM POINTER
00230A E958 7F 1782  A        CLR    MINCTR    RESET 15 MIN. COUNTER TO 0
00231A E95B 20 C6 E923        BRA    GONE
00232                         END
```

FIGURE 9 — MC68120 Software (Continued)



```
PAGE  005  MLTPRCA .SA:1

TOTAL ERRORS 00000--00000


   E8E0 AROUND 00170 00173*
   E83C CONT1  00090 00092*
   E865 COUNT  00116*00118
   E93A DMPCHK 00212 00216*
   E85A DMPRAM 00091 00110*00219
   E8F4 DOVRN  00180 00182*
   E945 DPRST  00198 00221*
   1785 EDPTR  00042*00070 00110 00157 00176 00182
   E8AC ELOOP  00149*00159
   E928 ERROR  00189 00191 00207*
   178D FROM   00056*00080 00168 00172 00175
   E923 GONE   00194 00202*00215 00231
   00CC HIGH   00050*00190
   0001 IRQHG  00054*00072 00213 00227
   0000 IRQLW  00052*00208 00222
   E92E KPLKNG 00210*00218 00220
   E8A6 LAST   00125 00147*
   E87B LOOP   00127*00145
   0011 LOW    00048*00188
   1782 MINCTR 00037*00067 00165 00192 00201 00230
   178B NRMPTR 00047*00082 00195 00200 00229
   E903 OK     00184 00188*
   0001 P2DDR  00011*00075
   0003 P2DR   00010*00073 00209 00214 00223 00228
   E834 POLL1  00088*00094 00106 00203
   00F1 RMDR   00033*00121
   1783 SDPTR  00040*00069 00111 00129 00132 00149 00152 00158 00183 00187
   1780 SECCTR 00035*00066 00102 00104 00167
   0017 SEMPH1 00025*00202 00207
   0018 SEMPH2 00026*00221
   0019 SEMPH3 00027*00088 00216
   001A SEMPH4 00028*00210 00224
   001B SEMPH5 00029*00139 00160
   001C SEMPH6 00030*00140
   0008 TCSR   00012*00092 00098
   1789 TEMP   00045*00128 00133 00136 00148 00153 00156
   0009 TIMERR 00013*00076 00099
   000B TIMROC 00014*00078 00101
   E881 TLOOP  00129*00138
   E8C9 TMRSRV 00107 00165*
   EB80 TO     00057*00169
   EB00 TXDCR  00039*
   1787 TXRMSZ 00044*00112
   E94B WAIT   00224*00226
   E89A WAIT1  00140*00142
   00F0 WHNO   00031*00084 00116 00119 00123 00144
```

FIGURE 9 — MC68120 Software (Concluded)

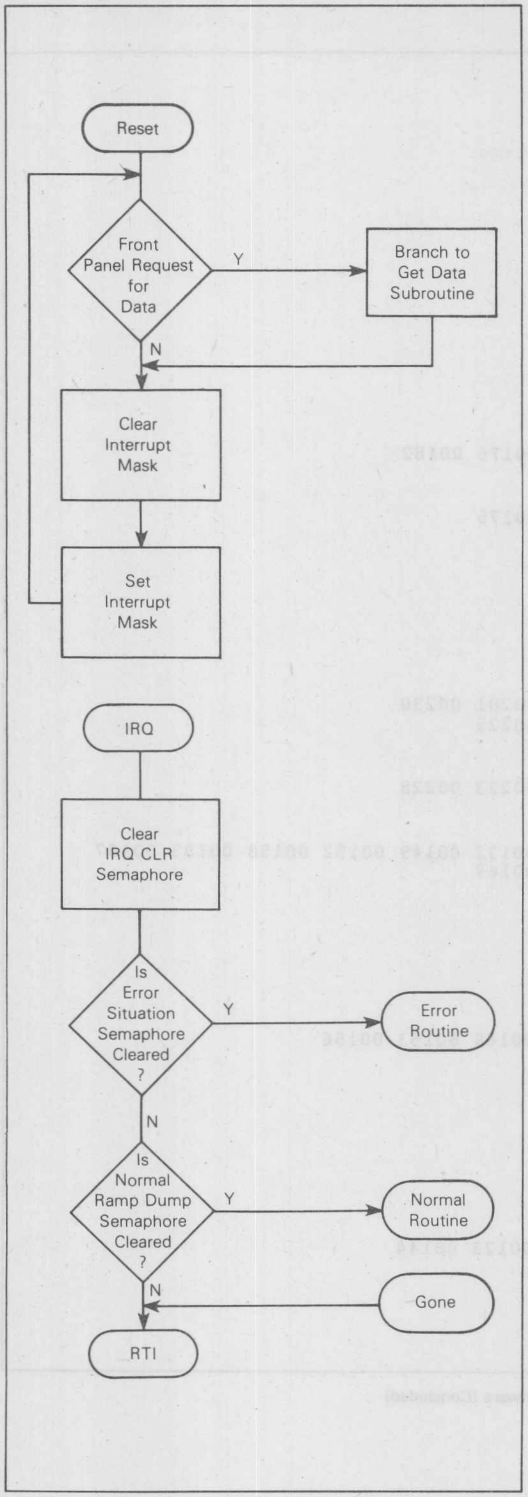

Reset
Front Panel Request for Data
Y
Branch to Get Data Subroutine
N
Clear Interrupt Mask
Set Interrupt Mask
IRQ
Clear IRQ CLR Semaphore
Is Error Situation Semaphore Cleared ?
Y
Error Routine
N
Is Normal Ramp Dump Semaphore Cleared ?
Y
Normal Routine
N
Gone
RTI

FIGURE 10 — MC6809E Flowchart

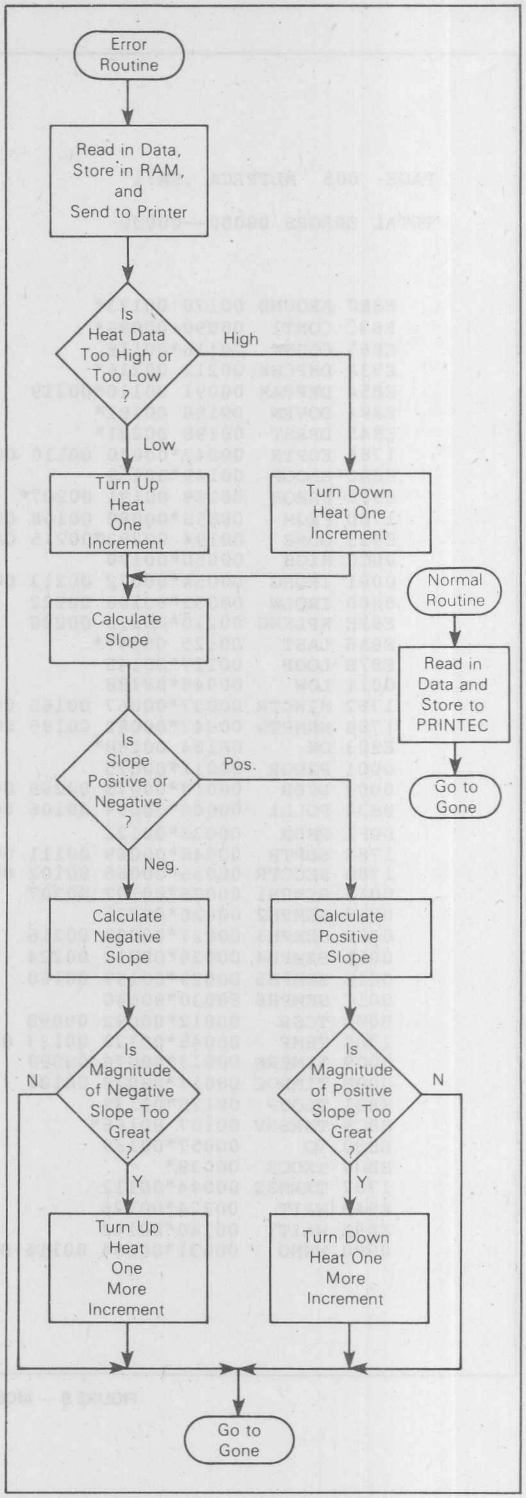
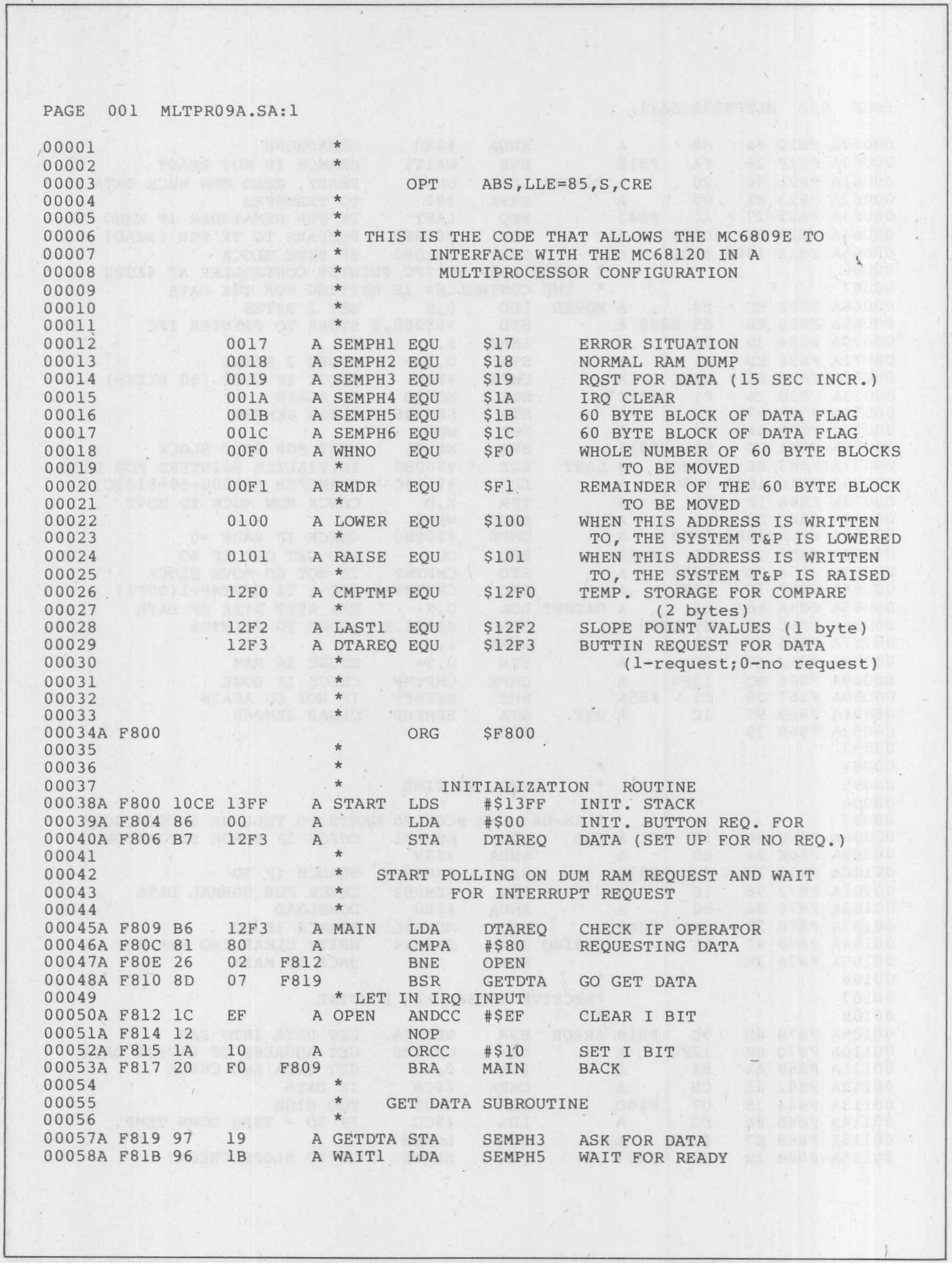
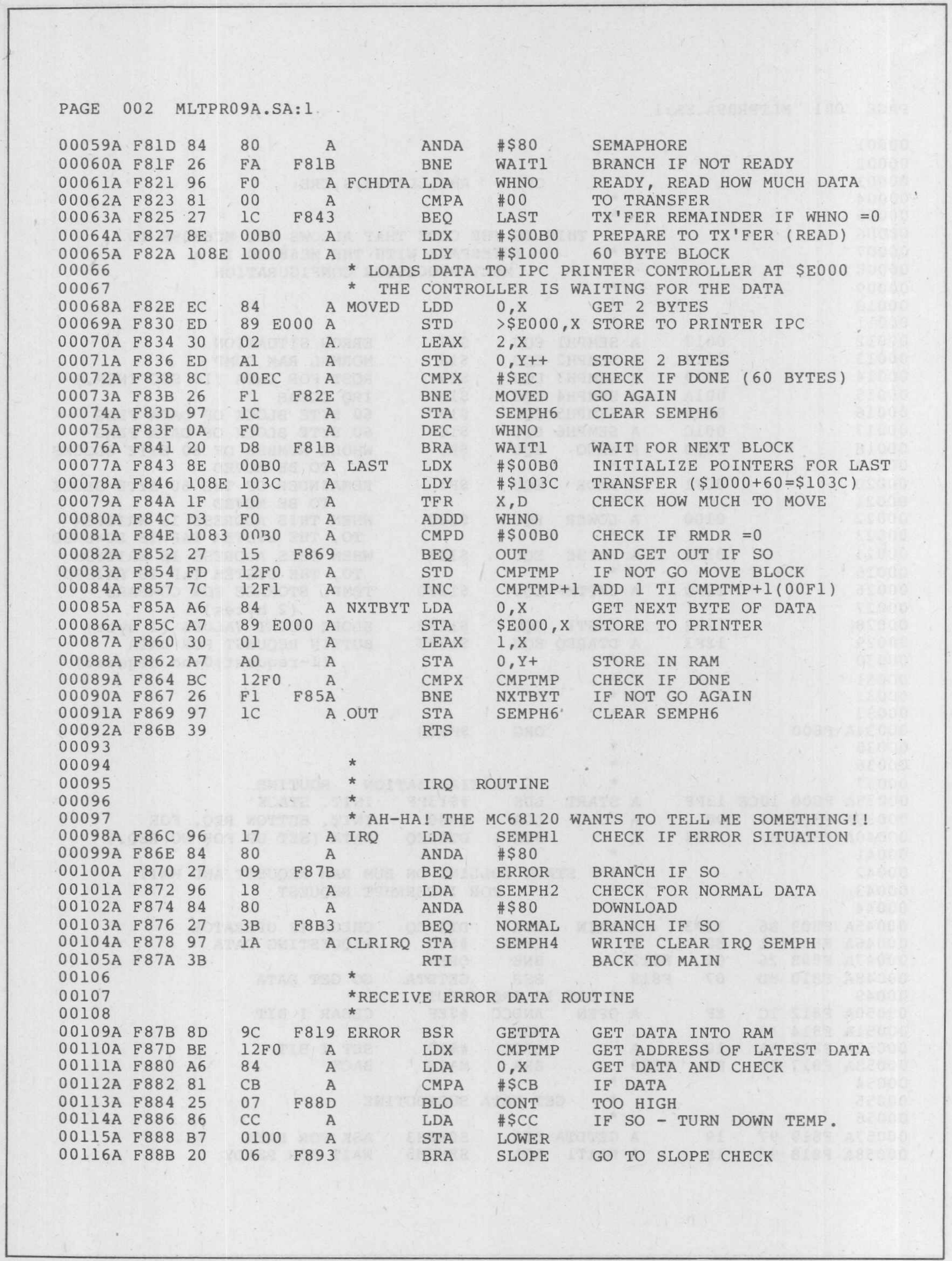

Error Routine
Read in Data, Store in RAM, and Send to Printer
Is Heat Data Too High or Too Low ?
High
Low
Turn Up Heat One Increment
Turn Down Heat One Increment
Normal Routine
Read in Data and Store to PRINTEC
Go to Gone
Calculate Slope
Is Slope Positive or Negative ?
Pos.
Neg.
Calculate Negative Slope
Calculate Positive Slope
Is Magnitude of Negative Slope Too Great ?
N
Y
Is Magnitude of Positive Slope Too Great ?
N
Y
Turn Up Heat One More Increment
Turn Down Heat One More Increment
Go to Gone

FIGURE 10 — MC6809E Flowchart (Concluded)



```
PAGE  001  MLTPR09A.SA:1

00001                      *
00002                      *
00003                      *      OPT    ABS,LLE=85,S,CRE
00004                      *
00005                      *
00006                      *  THIS IS THE CODE THAT ALLOWS THE MC6809E TO
00007                      *        INTERFACE WITH THE MC68120 IN A
00008                      *         MULTIPROCESSOR CONFIGURATION
00009                      *
00010                      *
00011                      *
00012            0017    A SEMPH1 EQU    $17      ERROR SITUATION
00013            0018    A SEMPH2 EQU    $18      NORMAL RAM DUMP
00014            0019    A SEMPH3 EQU    $19      RQST FOR DATA (15 SEC INCR.)
00015            001A    A SEMPH4 EQU    $1A      IRQ CLEAR
00016            001B    A SEMPH5 EQU    $1B      60 BYTE BLOCK OF DATA FLAG
00017            001C    A SEMPH6 EQU    $1C      60 BYTE BLOCK OF DATA FLAG
00018            00F0    A WHNO   EQU    $F0      WHOLE NUMBER OF 60 BYTE BLOCKS
00019                      *                          TO BE MOVED
00020            00F1    A RMDR   EQU    $F1      REMAINDER OF THE 60 BYTE BLOCK
00021                      *                          TO BE MOVED
00022            0100    A LOWER  EQU    $100     WHEN THIS ADDRESS IS WRITTEN
00023                      *                       TO, THE SYSTEM T&P IS LOWERED
00024            0101    A RAISE  EQU    $101     WHEN THIS ADDRESS IS WRITTEN
00025                      *                       TO, THE SYSTEM T&P IS RAISED
00026            12F0    A CMPTMP EQU    $12F0    TEMP. STORAGE FOR COMPARE
00027                      *                             (2 bytes)
00028            12F2    A LAST1  EQU    $12F2    SLOPE POINT VALUES (1 byte)
00029            12F3    A DTAREQ EQU    $12F3    BUTTIN REQUEST FOR DATA
00030                      *                          (1-request;0-no request)
00031                      *
00032                      *
00033                      *
00034A F800                       ORG    $F800
00035                      *
00036                      *
00037                      *          INITIALIZATION    ROUTINE
00038A F800 10CE 13FF    A START  LDS    #$13FF   INIT. STACK
00039A F804 86   00      A        LDA    #$00     INIT. BUTTON REQ. FOR
00040A F806 B7   12F3    A        STA    DTAREQ   DATA (SET UP FOR NO REQ.)
00041                      *
00042                      *   START POLLING ON DUM RAM REQUEST AND WAIT
00043                      *          FOR INTERRUPT REQUEST
00044                      *
00045A F809 B6   12F3    A MAIN   LDA    DTAREQ   CHECK IF OPERATOR
00046A F80C 81   80      A        CMPA   #$80     REQUESTING DATA
00047A F80E 26   02   F812        BNE    OPEN
00048A F810 8D   07   F819        BSR    GETDTA   GO GET DATA
00049                      * LET IN IRQ INPUT
00050A F812 1C   EF      A OPEN   ANDCC  #$EF     CLEAR I BIT
00051A F814 12           A        NOP
00052A F815 1A   10      A        ORCC   #$10     SET I BIT
00053A F817 20   F0   F809        BRA    MAIN     BACK
00054                      *
00055                      *    GET DATA SUBROUTINE
00056                      *
00057A F819 97   19      A GETDTA STA    SEMPH3   ASK FOR DATA
00058A F81B 96   1B      A WAIT1  LDA    SEMPH5   WAIT FOR READY
```

FIGURE 11 — MC6809E Software



```
PAGE  002  MLTPR09A.SA:1

00059A F81D 84    80       A        ANDA   #$80      SEMAPHORE
00060A F81F 26    FA    F81B        BNE    WAIT1     BRANCH IF NOT READY
00061A F821 96    F0       A FCHDTA LDA    WHNO      READY, READ HOW MUCH DATA
00062A F823 81    00       A        CMPA   #00       TO TRANSFER
00063A F825 27    1C    F843        BEQ    LAST      TX'FER REMAINDER IF WHNO =0
00064A F827 8E    00B0     A        LDX    #$00B0    PREPARE TO TX'FER (READ)
00065A F82A 108E  1000     A        LDY    #$1000    60 BYTE BLOCK
00066                        * LOADS DATA TO IPC PRINTER CONTROLLER AT $E000
00067                        *  THE CONTROLLER IS WAITING FOR THE DATA
00068A F82E EC    84       A MOVED  LDD    0,X       GET 2 BYTES
00069A F830 ED    89 E000  A        STD    >$E000,X  STORE TO PRINTER IPC
00070A F834 30    02       A        LEAX   2,X
00071A F836 ED    A1       A        STD    0,Y++     STORE 2 BYTES
00072A F838 8C    00EC     A        CMPX   #$EC      CHECK IF DONE (60 BYTES)
00073A F83B 26    F1    F82E        BNE    MOVED     GO AGAIN
00074A F83D 97    1C       A        STA    SEMPH6    CLEAR SEMPH6
00075A F83F 0A    F0       A        DEC    WHNO
00076A F841 20    D8    F81B        BRA    WAIT1     WAIT FOR NEXT BLOCK
00077A F843 8E    00B0     A LAST   LDX    #$00B0    INITIALIZE POINTERS FOR LAST
00078A F846 108E  103C     A        LDY    #$103C    TRANSFER ($1000+60=$103C)
00079A F84A 1F    10       A        TFR    X,D       CHECK HOW MUCH TO MOVE
00080A F84C D3    F0       A        ADDD   WHNO
00081A F84E 1083  00B0     A        CMPD   #$00B0    CHECK IF RMDR =0
00082A F852 27    15    F869        BEQ    OUT       AND GET OUT IF SO
00083A F854 FD    12F0     A        STD    CMPTMP    IF NOT GO MOVE BLOCK
00084A F857 7C    12F1     A        INC    CMPTMP+1  ADD 1 TI CMPTMP+1(00F1)
00085A F85A A6    84       A NXTBYT LDA    0,X       GET NEXT BYTE OF DATA
00086A F85C A7    89 E000  A        STA    $E000,X   STORE TO PRINTER
00087A F860 30    01       A        LEAX   1,X
00088A F862 A7    A0       A        STA    0,Y+      STORE IN RAM
00089A F864 BC    12F0     A        CMPX   CMPTMP    CHECK IF DONE
00090A F867 26    F1    F85A        BNE    NXTBYT    IF NOT GO AGAIN
00091A F869 97    1C       A OUT    STA    SEMPH6    CLEAR SEMPH6
00092A F86B 39                      RTS
00093
00094                        *
00095                        * IRQ ROUTINE
00096                        *
00097                        * AH-HA! THE MC68120 WANTS TO TELL ME SOMETHING!!
00098A F86C 96    17       A IRQ    LDA    SEMPH1    CHECK IF ERROR SITUATION
00099A F86E 84    80       A        ANDA   #$80
00100A F870 27    09    F87B        BEQ    ERROR     BRANCH IF SO
00101A F872 96    18       A        LDA    SEMPH2    CHECK FOR NORMAL DATA
00102A F874 84    80       A        ANDA   #$80      DOWNLOAD
00103A F876 27    3B    F8B3        BEQ    NORMAL    BRANCH IF SO
00104A F878 97    1A       A CLRIRQ STA    SEMPH4    WRITE CLEAR IRQ SEMPH
00105A F87A 3B                      RTI              BACK TO MAIN
00106                        *
00107                        *RECEIVE ERROR DATA ROUTINE
00108                        *
00109A F87B 8D    9C    F819 ERROR  BSR    GETDTA    GET DATA INTO RAM
00110A F87D BE    12F0     A        LDX    CMPTMP    GET ADDRESS OF LATEST DATA
00111A F880 A6    84       A        LDA    0,X       GET DATA AND CHECK
00112A F882 81    CB       A        CMPA   #$CB      IF DATA
00113A F884 25    07    F88D        BLO    CONT      TOO HIGH
00114A F886 86    CC       A        LDA    #$CC      IF SO - TURN DOWN TEMP.
00115A F888 B7    0100     A        STA    LOWER
00116A F88B 20    06    F893        BRA    SLOPE     GO TO SLOPE CHECK
```

FIGURE 11 — MC6809E Software (Continued)

```
PAGE  003  MLTPR09A.SA:1

00117A F88D 12                CONT    NOP
00118A F88E 86    11       A          LDA     #$11
00119A F890 B7    0101     A          STA     RAISE      INCREASE TEMP.
00120                         *
00121                         *  IF SLOPE NEG. - GETTING HOTTER
00122                         *  IF SLOPE POS. - GETTING COLDER
00123                         *
00124A F893 A6    82       A  SLOPE   LDA     0,-X       GET LAST
00125A F895 B7    12F2     A          STA     LAST1      DATA AND NEXT
00126A F898 A6    84       A          LDA     0,X        TO LAST DATA
00127A F89A B0    12F2     A          SUBA    LAST1
00128A F89D 2A    0B   F8AA           BPL     MAGNC      BRANCH IF COLDER
00129                         *                             NEG. SLOPE- HOTTER
00130A F89F 84    7F       A          ANDA    #$7F       DELETE NEG SIGN
00131A F8A1 81    10       A  MAGNH   CMPA    #$10       COMPARE MAG OF SLOPE
00132A F8A3 25    D3   F878           BLO     CLRIRQ     TO CRIT. SLOPE VALUE
00133A F8A5 B7    0100     A          STA     LOWER      LOWER TEMP. 1 INCR.
00134A F8A8 20    CE   F878           BRA     CLRIRQ
00135A F8AA 81    10       A  MAGNC   CMPA    #$10       COMPARE MAG OF SLOPE TO
00136A F8AC 25    CA   F878           BLO     CLRIRQ     CRIT SLOPE VALUE
00137A F8AE B7    0101     A          STA     RAISE      RAISE TEMP. 1 INCR.
00138A F8B1 20    C5   F878           BRA     CLRIRQ
00139A F8B3 8E    0080     A  NORMAL  LDX     #$0080     PREPARE TO GET DATA
00140A F8B6 EC    84       A          LDD     0,X        GET DATA
00141A F8B8 ED    89 E000  A  MOVIT   STD     $E000,X    MOVE TO PRINTER
00142A F8BC 30    02       A          LEAX    2,X
00143A F8BE 8C    00B0     A          CMPX    #$B0       CHECK IF DONE
00144A F8C1 26    F5   F8B8           BNE     MOVIT      KEEPING GOING
00145A F8C3 20    B3   F878           BRA     CLRIRQ     CLEAR IRQ AND OUT
00146                         *
00147A FFF0                           ORG     $FFF0
00148A FFF0       F800     A          FDB     START
00149A FFF2       F800     A          FDB     START
00150A FFF4       F800     A          FDB     START
00151A FFF6       F800     A          FDB     START
00152A FFF8       F86C     A          FDB     IRQ
00153A FFFA       F800     A          FDB     START
00154A FFFC       F800     A          FDB     START
00155A FFFE       F800     A          FDB     START
00156                                 END
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

   F878 CLRIRQ 00104*00132 00134             F85A NXTBYT 00085*00090
               00136 00138 00145             F812 OPEN   00047 00050*
   12F0 CMPTMP 00026*00083 00084             F869 OUT    00082 00091*
               00089 00110                   0101 RAISE  0Q024*00119 00137
   F88D CONT   00113 00117*                  00F1 RMDR   00020*
   12F3 DTAREQ 00029*00040 00045             0017 SEMPH1 00012*00098
   F87B ERROR  00100 00109*                  0018 SEMPH2 00013*00101
   F821 FCHDTA 00061*                        0019 SEMPH3 00014*00057
   F819 GETDTA 00048 00057*00109             001A SEMPH4 00015*00104
   F86C IRQ    00098*00152                   001B SEMPH5 00016*00058
   F843 LAST   00063 00077*                  001C SEMPH6 00017*00074 00091
   12F2 LAST1  00028*00125 00127             F893 SLOPE  00116 00124*
   0100 LOWER  00022*00115 00133             F800 START  00038*00148 00149
   F8AA MAGNC  00128 00135*                              00150 00151 00153
   F8A1 MAGNH  00131*                                    00154 00155
   F809 MAIN   00045*00053                   F81B WAIT1  00058*00060 00076
   F82E MOVED  00068*00073                   00F0 WHNO   00018*00061 00075
   F8B8 MOVIT  00141*00144                               00080
   F8B3 NORMAL 00103 00139*
```

FIGURE 11 — MC6809E Software (Concluded)

AN-851

MOTOROLA MC6845 CRTC SIMPLIFIES VIDEO DISPLAY CONTROLLERS

Prepared by
Charles Melear and Jack Browne
Microprocessor Applications Engineering

The need for displaying visual information by the general business community has found widespread applications. Banks, airports, department stores, and other businesses need rapid display of visual information at points of sale and points of use. Much of this information is generated by people who have only a limited knowledge of the electronics involved. Therefore, they must rely on the equipment used to automatically receive data, digest it, and display it on a video monitor. Systems could range in complexity from those which display only a few lines of data to complicated word processors. Historically, character printers gave way to line printers. However, obtaining hard copy is cumbersome and slow, and a considerable amount of paper is used. Much of this information is used only momentarily and then discarded, such as inventory checks or airport flight schedules. The efficiency of low cost, high performance video monitors have

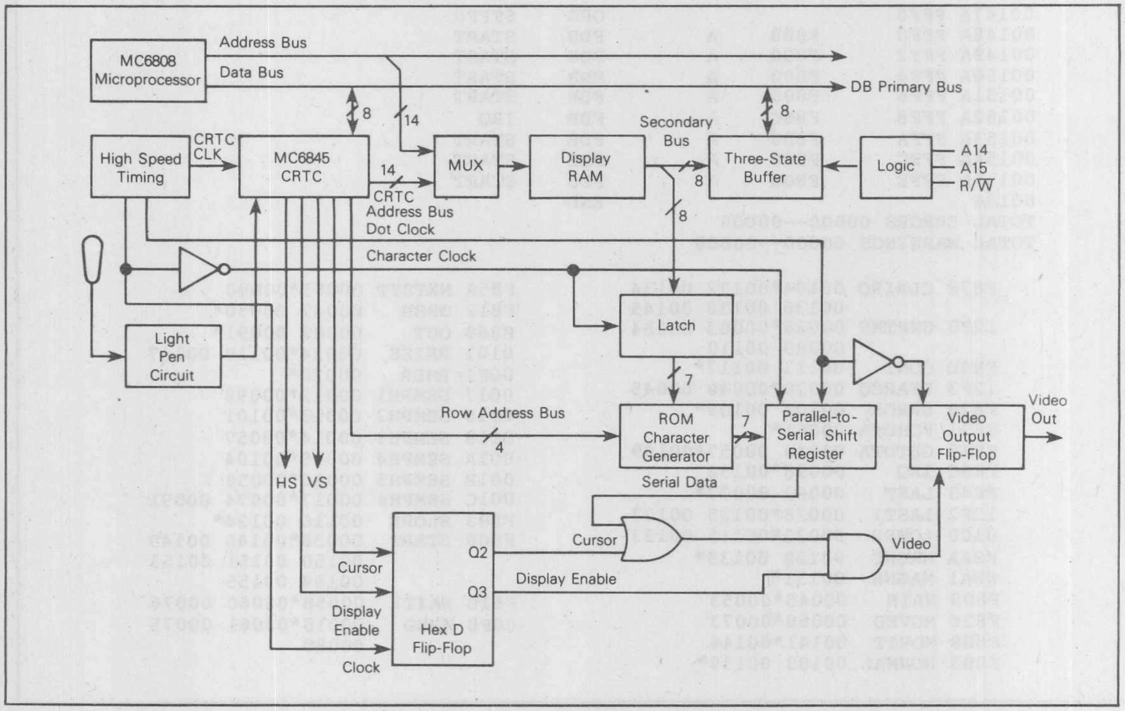


FIGURE 1 — CRT Controller Application

made the transition from hard copy to visual display even more advantageous. As video monitors have come into general use, the requirement for cost savings in the controller has intensified. LSI circuits have been appearing which meet that need.

The Motorola MC6845 CRT controller (CRTC) can economically solve many of the problems encountered with video monitor displays. This is acomplished by using an innovative design aimed at complete control of the monitor with intervention by the MPU only when new information is put into the display memory. The problems to be solved by the MC6845 in a raster scan video display controller are: cost, number of required components, amount of intervention by MPU, timing and synchronization of signals, and software, among others.

Today, CRT controllers can be built using an MC6845 which require approximately 25 ICs plus the extra chips required for memory. This number represents only a fraction of the parts required just a few years ago when SSI and MSI logic devices were used. CRT controllers were built using SSI and MSI logic devices which required well over one hundred ICs. With the MC6845 approach, the number of ICs can be reduced to approximately 25 plus those required for memory.

To illustrate the capabilities of an MC6845 based terminal, the software and "rough" hardware considerations used in its design are discussed. The terminal, as shown in Figure 1, has the following features:

Blinking Cursor
Carriage Return
Backspace
Line Feed
Automatic Scrolling
Move Cursor Up One Line
Paging
Home Cursor
Clear Screen

The CRTC has an address register that can point to any one of eighteen buried registers as shown in Figure 2. These can be programmed for up to 256 characters per row and 128 rows per screen with the only limitation being the bandwidth of the monitor. For this terminal, an 80 by 24 format of 7 by 9 dot matrix characters is used. Horizontal and vertical sync positions are programmable allowing the CRTC to generate the horizontal and vertical retrace pulses. A blanking signal (display enable) is generated during both horizontal and vertical retrace. Two sets of address lines are used. The first set of fourteen lines cycles in a binary fashion through the display memory and is incremented with each CRTC clock pulse. The second set of four lines can be used to address the row address select lines of a character generator. These also cycle in a binary fashion and are incremented with each horizontal sync pulse. A cursor, which may be programmed to blink, is also generated by the CRTC. It will be displayed at the address held in the CRTC cursor address register.

SYSTEM IMPLEMENTATION

Figure 3 represents a complete MC6808-MC6845 based system capable of receiving a digital input, processing it, and displaying alphanumeric data on a video monitor. The timing for the system is derived from a dot clock oscillator. Its frequency determines the rate at which information is shifted to the monitor. The dot clock oscillator output is divided by a counter to obtain the character rate clock. For a 9 column by 12 row character block which accommodates a 7 by 9 character, binary 8 is detected at Q3 on the counter and the resulting inverted output is fed into the synchronous clear input of the counter. For a 7 by 9 block, a logic gate could detect binary 6 on Q0, Q1, and Q2. It is important to use a counter with a synchronous clear so the clear pulse will be one dot clock period wide. The character clock (generated by the rising edge of Q3) serves as a shift/load signal for the output shift register and a clock to latch data from the display memory. The CRTC clock (generated by the trailing edge of Q2) is used to clock the MC6845 CRTC. Each character rate clock increments the address lines (MA0-MA13) of the MC6845. The display memory must be capable of being controlled by either the MPU or the CRTC. Therefore, the address lines for both devices (A0-A13 and MA0-MA13) are routed through multiplexers such as the SN74LS157. The MPU takes control of the display memory only when a new character is to be written. The output of the multiplexer addresses the memory.

As shown in Figure 3, the 8K × 8 static display memory requires 10 address lines for the address bus of the memory elements and 3 address lines for the 3-to-8 line decoder which drives the chip selects of the memory elements. The output of the display memory is fed into an 8-bit latch (74LS374) and is clocked into the latch on the next character clock. This latch helps to prevent address line jitter which could present spurious data to the character generator ROM. The character clock is used to latch data into the SN74LS374. This creates a one character clock delay from the time that an address becomes valid to the memory until data is presented to the character generator ROM. The character clock is also used to load the parallel word from the character generator ROM into the shift register, producing a second character clock delay. Once the shift register is loaded the dot clock is used to serially shift data from the shift register to the video driver.

In order to synchronize both the display enable and cursor output with the shift register output, a two CRTC clock delay must be imposed. Both signals are synchronous with the CRTC address lines. To implement this delay, the two signals (cursor and display enable) are clocked through two latches by the noninverted character clock and fed into the video driver. The video signal is the combination of the shifted data ORed with cursor and then ANDed with Display Enable. This is fed into a "D" flip-flop and clocked out by the dot clock.

The CRTC generates row addresses for the character generator ROM. Cycling is synchronized within the CRTC by the horizontal sync pulse (HSYNC) so that the address lines are incremented by each HSYNC.

When the MPU is required to read or write to the display memory, the address line multiplexer must be switched to the MPU address lines. Since the display memory is located from $0000 to $3FFF, address lines A15 and A14 will both be logic "0" if and only if the display memory is being addressed. Therefore, only "00" needs to be decoded on these two lines as an MPU address select line. In normal operation where the CRT controller is controlling the display memory, the secondary data bus is being driven by the display memory. Also, the MPU data bus is being driven by the MPU or some other peripheral. This requires that the two data buses be isolated from each other except during an MPU read or write of the display memory. This requires bus transceivers that can be set to the high-impedance state in both directions. These are shown in Figure 2 as three MC6885 Hex Buffer-Inverters. (If octal buffers are used, only two are required.)

To complete the entire system, RAM, ROM, and I/O interface circuitry is placed on the data buses. The RAM is used primarily for a scratch pad memory and the locations accessed by the stack pointer register. The ROM contains the operating program to service the I/O interface. The I/O interface can be a keyboard outputting parallel ASCII code or row/column information. As long as some method can be programmed to receive digital data and transfer it onto the data bus, the CRT controller, using an MC6845, can display that information on a video display.

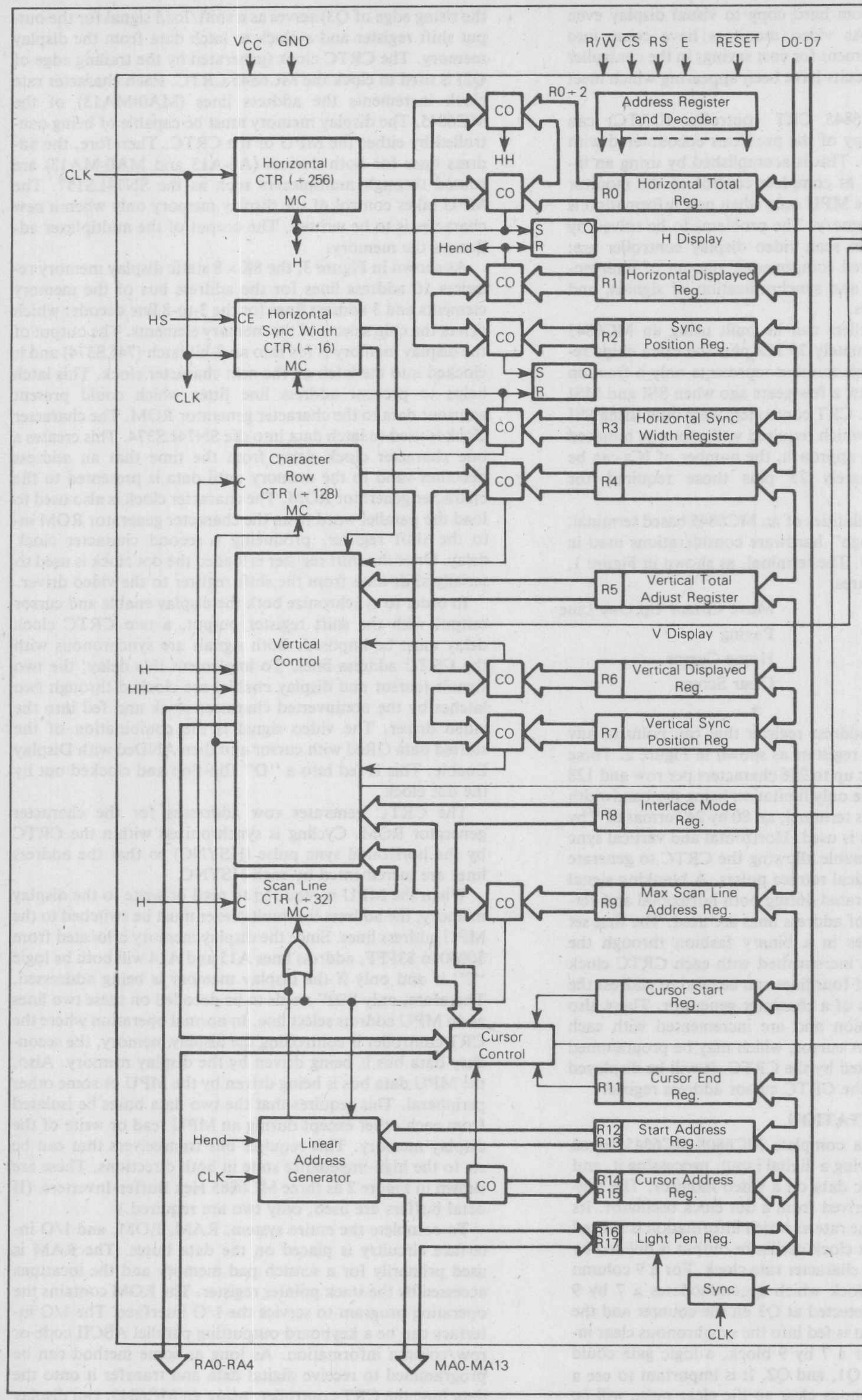


FIGURE 2 — MC6845 CRT Controller Block Diagram

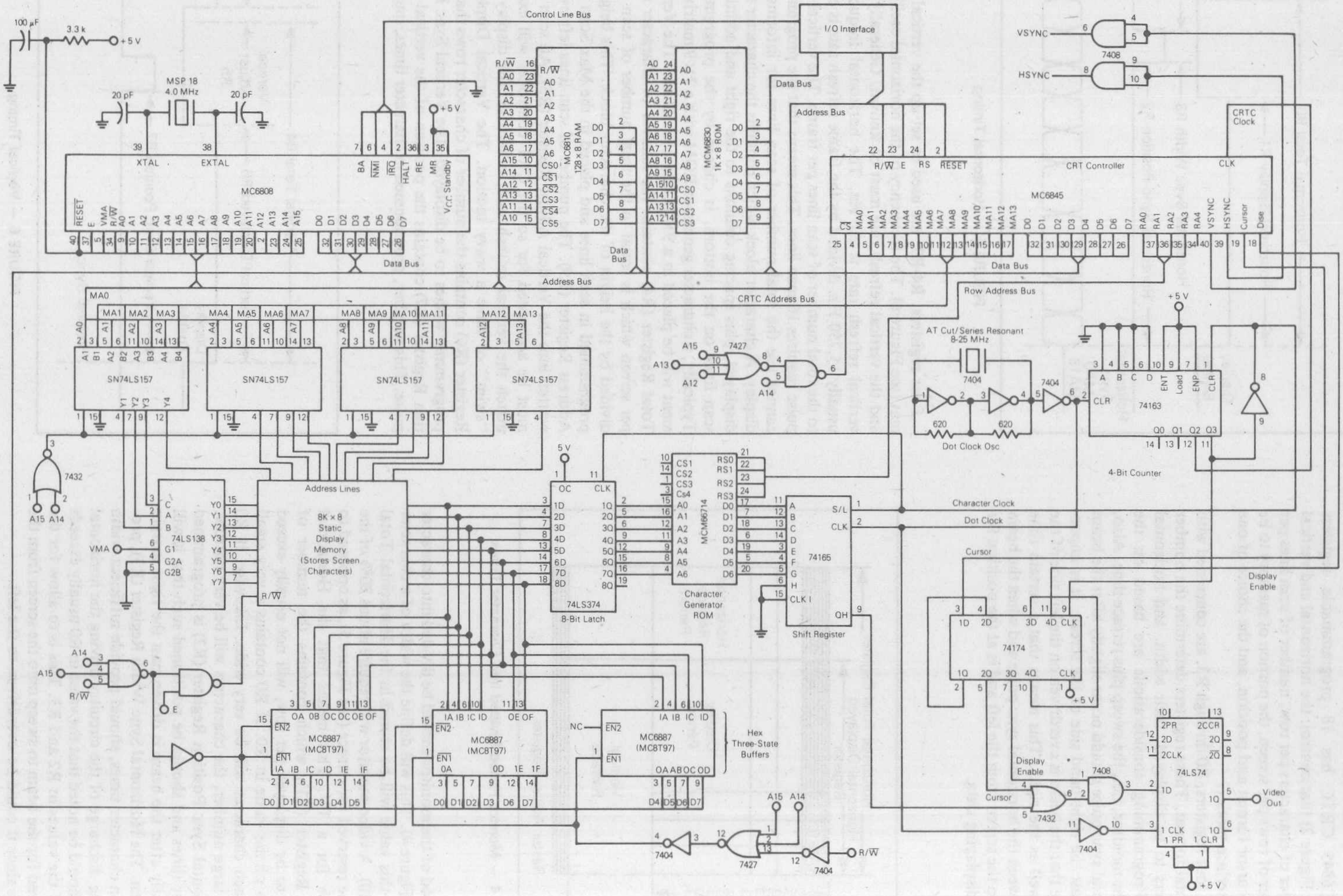


FIGURE 3 — MC6808-MC6845 Display Terminal, Schematic Diagram

DEVICE IMPLEMENTATION

The MC6845 CRTC has 18 programmable registers (R0-R17 in Figure 2) that control: the horizontal and vertical sync, number of characters per row, number of scan lines per row, number of rows per screen, the portion of memory to be displayed, cursor format and position, and the choice of one of three interlace modes.

The first four registers, R0 through R3, are concerned with the horizontal format. These registers determine the number of characters to be displayed, their width, and horizontal position. Programming considerations are based on the period of the monitor, i.e., the sweep plus retrace time. Also, the horizontal sync pulse should occur slightly after the beam is driven past the right-hand side of the screen. It is important to note that the beam is overdriven on the left side of the screen as well as the right. This means that a certain time elapses between the horizontal sync pulse and when the beam sweeps onto the screen from the left and is at the position for it to start displaying data.

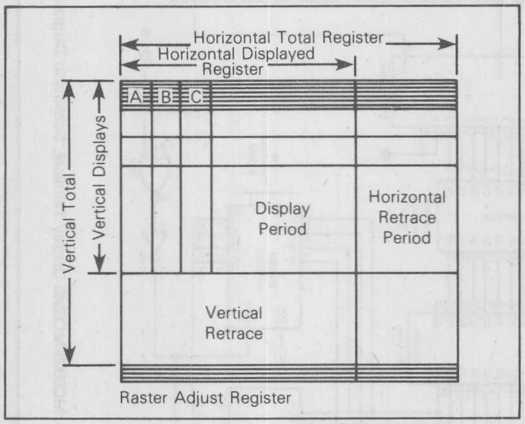


FIGURE 4 — Monitor Period Divided Into Character Times

The period of the monitor should be divided into character times (see Figure 4). This will define the width of a character block and this value will be stored in the Horizontal Total Register (R0). A video monitor will require about 20% of the period to be reserved for retrace (see Figure 5), as opposed to about 35% for a TV. This means that the Horizontal Displayed Register (R1), which contains the number of characters to be displayed per row, will not usually exceed about 80% of the value in R0. If R0 contains a very small number, each character will be very wide. Likewise, if R0 contains a large number, the characters will be very narrow. The Horizontal Sync Position Register (R2) is programmed in character times and should be positioned such that it will occur slightly after the beam is driven past the right margin of the screen. The Horizontal Sync Width Register (R3), programmed in character times, should provide sufficient width to allow the discharge of the circuitry driving the horizontal sweep. It should be noted that the value in R0 usually exceeds the sum of the values in R2 and R3. This is to allow for the time required for the beam to sweep onto the screen from the left margin since it could be overdriven to the left.

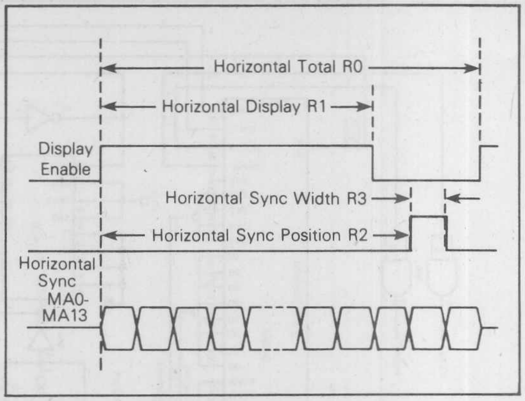


FIGURE 5 — Horizontal Timing

Four registers, R4-R7, are used to set up the vertical format (see Figure 6). The frequency of the horizontal oscillator and the vertical refresh rate must be known. Generally, the vertical refresh rate is 60 Hz. The horizontal frequency, usually 15,750 Hz, divided by the frame refresh rate is equal to the total number of scan lines per frame. The vertical sync pulse requires 16 scan lines. This means that the programmer cannot use the total number of scan lines for information display. A character block which contains the character to be displayed, plus spacing columns to the right and additional scan lines on the bottom, is chosen by the programmer. Typically, a character generator ROM with a 7 × 9 matrix element will be placed in a 9 × 12 character block. The Vertical Total Register (R4) contains the number of character rows per screen which is equal to the total number of scan lines divided by the height of the character block. This height is programmed in scan lines and placed in the Max Scan Line Address Register (R9). The number of scan lines left over is written into the Vertical Adjust Register (R5). All scan lines must be accounted for so the CRT controller will exactly match the vertical refresh rate; otherwise, the display will "swim" or have a wavy motion. The Vertical Displayed Register (R6) contains the number of character rows that the programmer wishes to be displayed. The Vertical Sync Position Register (R7) contains the position of the vertical sync pulse. This number, programmed in character times, must be

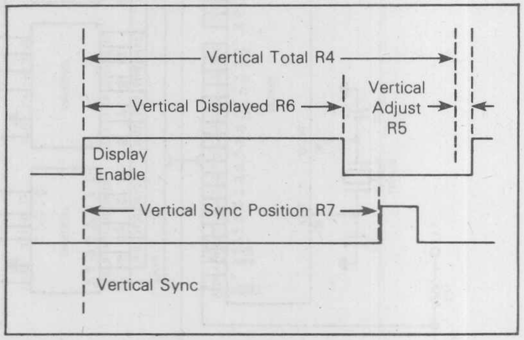


FIGURE 6 — Vertical Timing

greater than or equal to the Vertical Displayed Register (R6), but not so much greater that it shifts the last rows of the displayed text off the bottom of the screen. Once these registers are set up, the raster is completely defined.

Three operating modes are available with the MC6845 which have to do with which field (odd or even) that information is written into. The first mode, Normal Sync, writes information into one field only (see Figure 7). Remember, one frame requires two vertical sweeps of the screen. The first sweep (even field) starts at the upper left corner of the screen and the second sweep (odd field) starts at the top center. When writing into one field, each dot will be updated 60 times per second.

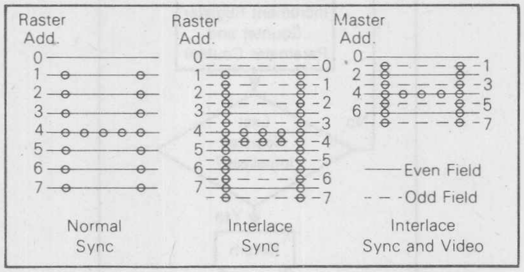


FIGURE 7 — Interlace Mode (R8)

The second mode, Interlace Sync, writes in both fields. The odd field is an exact duplicate of the even field. Essentially, the same information is written twice. This has the advantage of making the letters appear to have solid vertical lines thus improving resolution. However, each dot is now refreshed only 30 times per second which may cause an objectionable flicker on the screen. This flicker cannot be perceived by all people due to variances in eye sight. Also, the persistance of the phosphor will moderate the effect of the flicker.

The third mode, Interlace Sync and Video, also writes in both fields. However, one half the character is written in each field. This means an eight row character block in this mode will have four scan lines in the even field and four in the odd field making a character only half the height of the other two modes. This allows the highest screen density for the MC6845. These modes are programmed in the Interlace Mode Register (R8).

The MC6845 also controls the cursor format and blink rate (see Figure 8). Each character row has a certain number of scan lines as defined by the Max Scan Line Address Register (R9). The Cursor Start Register (R10) specifies on which scan line the cursor begins. Also, this register contains a bit that specifies whether the cursor will blink or not blink. Another bit specifies the blink rate which is either one sixteenth or one thirty second of the field rate.

The Cursor End Register (R11) specifies the scan line at the bottom of the cursor. If the same number is specified for both the starting and ending scan line, the cursor will be one line tall.

There are six remaining registers. The Start Address Registers High (R12) and Low (R13), contain the address of the first byte of memory to be displayed after vertical retrace. The Cursor Registers High (R14) and Low (R15) contain the address where the cursor will appear. The Light Pen Registers High (R16) and Low (R17) will receive the current address appearing on the CRT control address lines following the recognition of the low-to-high transition of the light pen strobe (LPSTB) input. Once the LPSTB low-to-high transition is recognized, the next low-to-high CRTC clock transition latches the address information and loads it into the Light Pen Register. These registers are used primarily by the programmer who writes the software for the terminal. The method in which they are used is discussed in the software considerations portion of this application note.

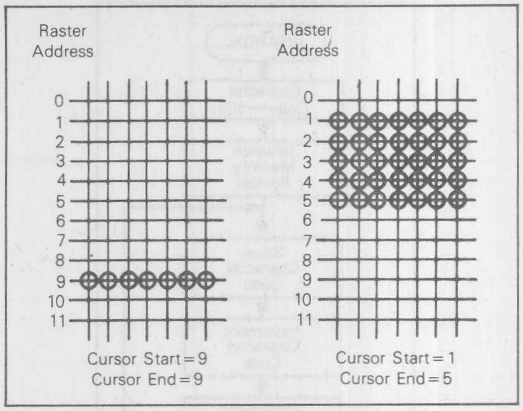


FIGURE 8 — Cursor Start and End Register

In order to complete the hardware discussion, the dot clock and character clocks must be defined. The character clock rate will be the product of the horizontal oscillator frequency and the total horizontal character times described in calculating the value for R0. The dot clock will be the product of the character rate clock and the width of the character block in columns. This requires a different dot clock for each screen format.

SOFTWARE IMPLEMENTATION

Once the system has been defined, software development may begin. The firmware residing in ROM will initialize and support the terminal. When power is applied to the system, the MPU automatically jumps to the reset address stored in location $FFFE and $FFFF. This address is the beginning of the initialization sequence.

After a power-on-reset, the display memory is initialized (to avoid a flash of false data), the eighteen buried registers of the CRT controller are initialized, and characters are accepted from the keyboard. Some control characters will be decoded to implement the following features:

Carriage Return
Backspace
Line Feed
Move Cursor Up One Line
Paging
Home Cursor
Clear Screen

Scrolling up or down will be done automatically as required.

The software was developed using the concepts of structured programming. The first two routines which were written support the hardware development and debugging. The first routine is named CHARGN and its flowchart is shown in Figure 9. This routine initializes the display memory with successive ASCII character codes which help identify addressing problems. The second routine is named CRTINT and initializes the CRT controller (see flowchart in Figure 10). The register values to implement an 80 by 24 display are read from the ROM and stored into the buried registers of the CRT controller. Again, it is important to initialize the display memory prior to initializing the MC6845, to avoid a flash of false data. After the system has been initialized by running this program (as listed in Figure 11), waveforms, timing, and data may be checked, thus speeding the design phase.

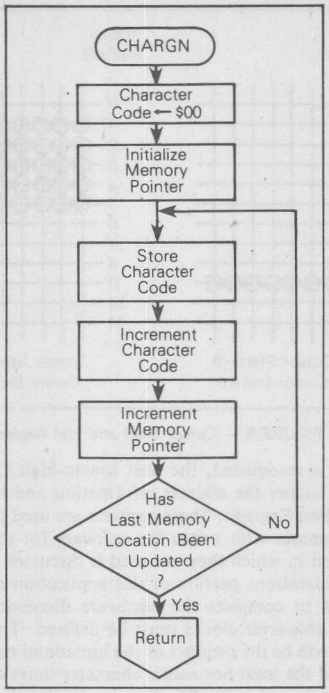


FIGURE 9 — CHARGN Subroutine Flowchart
Loads ASCII character codes into display memory.

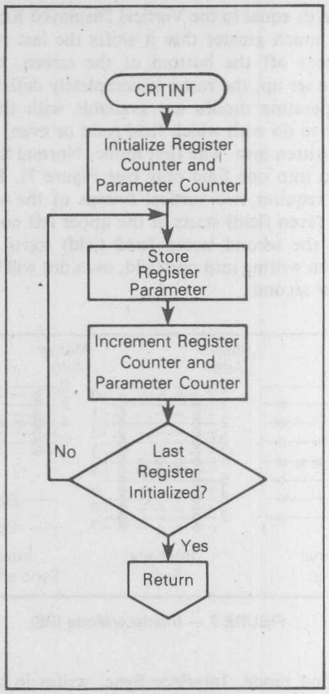

CRTINT
Initialize Register Counter and Parameter Counter
Store Register Parameter
Increment Register Counter and Parameter Counter
Last Register Initialized?
No
Yes
Return

FIGURE 10 — CRTINT Subroutine Flowchart
Initializes the CRTC registers with the previously calculated values stored in the ROM.

```
PAGE   001   BOOT       .SA:1

00001       4000    A CRTCAD EQU  $4000
00002       4001    A CRTCRG EQU  $4001
00003A E3FE                  ORG  $E3FE
00004A E3FE E0      A        FCB  $E0,0
00005A E000                  ORG  $E000
00006A E000 4F        CHARGN CLRA                FILL SCREEN WITH CHARACTER
00007A E001 CE 0000 A        LDX  #$0000
00008A E004 A7 00   A CHAR   STAA 0,X            STORE CHARACTER
00009A E006 4C               INCA                GET NEXT CHARACTER
00010A E007 08               INX                 MOVE TO NEXT LOCATION
00011A E008 8C 1000 A        CPX  #$1000         FINISHED
00012A E00B 26 F7 E004       BNE  CHAR           FINISHED?
00013A E00D 5F        CRTINT CLRB                INITIALIZE CRTC
00014A E00E CE E022 A        LDX  #TABLE
00015A E011 F7 4000 A CRTIN1 STAB CRTCAD         SELECT CRTC REGISTER
00016A E014 A6 00   A        LDAA 0,X
00017A E016 B7 4001 A        STAA CRTCRG
00018A E019 08               INX
00019A E01A 5C               INCB
00020A E01B C1 10   A        CMPB #$10
00021A E01D 26 F2 E011       BNE  CRTIN1
00022A E01F 01        LOOPER NOP
00023A E020 20 FD E01F       BRA  LOOPER
00024A E022 30      A TABLE  FCB  $30,$26,$2B,$02,$14,$01
00025A E028 12      A        FCB  $12,$13,$00,$0B,$40,$08,$00,$00,$00
00026                        END
TOTAL ERRORS 00000--00000
```

FIGURE 11 — CRT DEM Listing
This program, resident in PROM, will initialize the display memory with successive ASCII characters. This will allow initial checkout of the hardware.

These routines must be modified and additional routines written to implement all of the features of the terminal. A MONITOR program (see flowchart in Figure 12) is called by the reset vector stored in the ROM. Under control of the monitor program, the stack pointer is initialized at the end of the RAM (address $A07F), the self-modifying sections of code are dumped to the RAM, and all variables are initialized.

The BLANKR subroutine is then called. It is a revision of the CHARGN subroutine (see flowchart in Figure 9 and listing in Figure 11). Instead of stepping through the entire ASCII character code, the ASCII blank code ($20) is stored in the display memory. After the display memory has been filled with blanks, the CRTINT subroutine, discussed previously, is called.

The monitor program calls CHARRC, a subroutine which accepts and processes a character. Control is returned to the monitor program which in turn loops on the CHARRC subroutine call. The result is that the terminal continuously accepts characters. A flowchart of CHARRC appears in Figure 13. The CHARRC subroutine calls the input character subroutine INCH (see flowchart in Figure 14), which receives one keyboard entry.

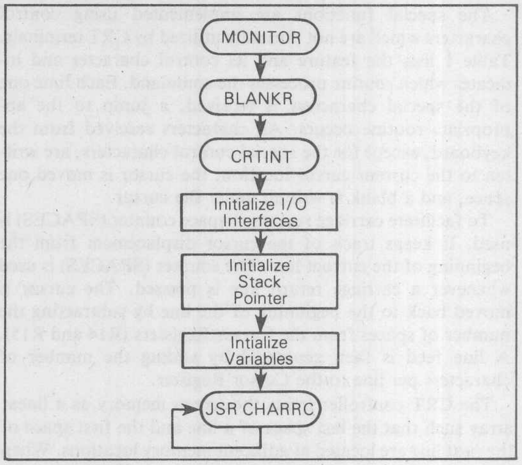


FIGURE 12 — MONITOR Program Flowchart
Calls all routines required to implement the terminal.

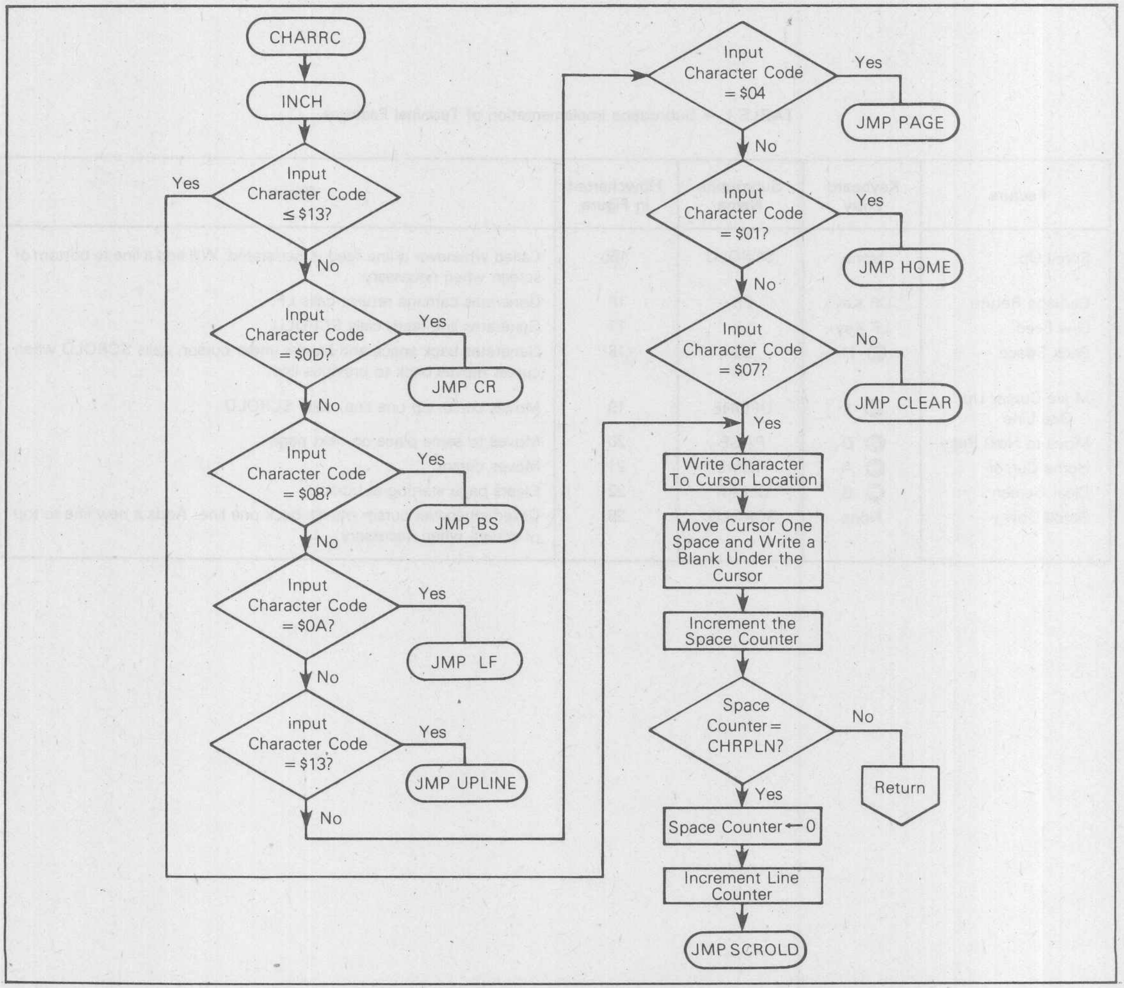

CHARRC
INCH
Input Character Code ≤ $13?
Yes
No
Input Character Code = $0D?
Yes
JMP CR
No
Input Character Code = $08?
Yes
JMP BS
No
Input Character Code = $0A?
Yes
JMP LF
No
input Character Code = $13?
Yes
JMP UPLINE
No
Input Character Code = $04
Yes
JMP PAGE
No
Input Character Code = $01?
Yes
JMP HOME
No
Input Character Code = $07?
No
JMP CLEAR
Yes
Write Character To Cursor Location
Move Cursor One Space and Write a Blank Under the Cursor
Increment the Space Counter
Space Counter = CHRPLN?
No
Return
Yes
Space Counter ← 0
Increment Line Counter
JMP SCROLD

FIGURE 13 — CHARRC Subroutine
Accepts characters from keyboard, moves cursor, and decodes all special characters.

The special functions are implemented using control characters which are not normally utilized by CRT terminals. Table 1 lists the feature and its control character and indicates which routine processes the command. Each time one of the special characters is received, a jump to the appropriate routine occurs. All characters received from the keyboard, except for the special control characters, are written to the current cursor location, the cursor is moved one space, and a blank is written under the cursor.

To facilitate carriage returns, a space counter (SPACES) is used. It keeps track of the cursor displacement from the beginning of the current line. The counter (SPACES) is used whenever a carriage return key is pressed. The cursor is moved back to the beginning of the line by subtracting the number of spaces from the Cursor Registers (R14 and R15). A line feed is then generated by adding the number of characters per line to the Cursor Register.

The CRT controller treats the screen memory as a linear array such that the last space of a line and the first space of the next line are located at adjacent memory locations. When the cursor is at the end of a line and another character is input, the cursor moves to the first of the next line. The space counter (SPACES) must be reset.

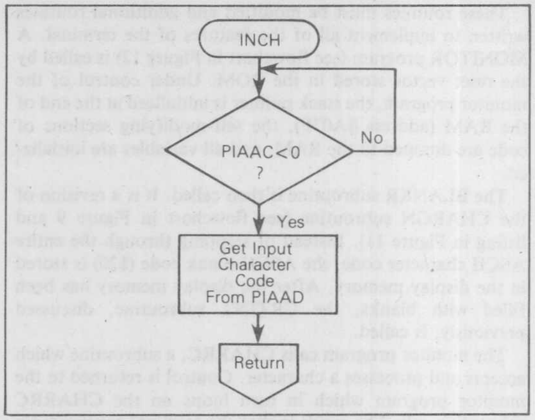


FIGURE 14 — INCH Subroutine Flowchart

Polls PIA A Control Register until IRQA1 is set, then the data is retrieved from the PIA A Data Register.

TABLE 1 — Subroutine Implementation of Terminal Features

Feature	Keyboard Entry	Subroutine Name	Flowcharted in Figure	Result
Scroll Up	None	SCROLU	15b	Called whenever a line feed is generated. Will add a line to bottom of screen when necessary.
Carriage Return	CR Key	CR	16	Generates carriage return, calls LF.
Line Feed	LF Key	LF	17	Generates line feed, calls SCROLU.
Back Space	© H	BS	18	Generates back space and blanks under cursor, calls SCROLD when cursor moves back to previous line.
Move Cursor Up One Line	© $	UPLINE	19	Moves cursor up one line, calls SCROLD.
Move to Next Page	© D	PAGE	20	Moves to same place on next page.
Home Cursor	© A	HOME	21	Moves cursor.
Clear Screen	© G	CLEAR	22	Clears page starting at cursor.
Scroll Down	None	SCROLD	23	Called whenever cursor moves back one line. Adds a new line to top of screen when necessary.

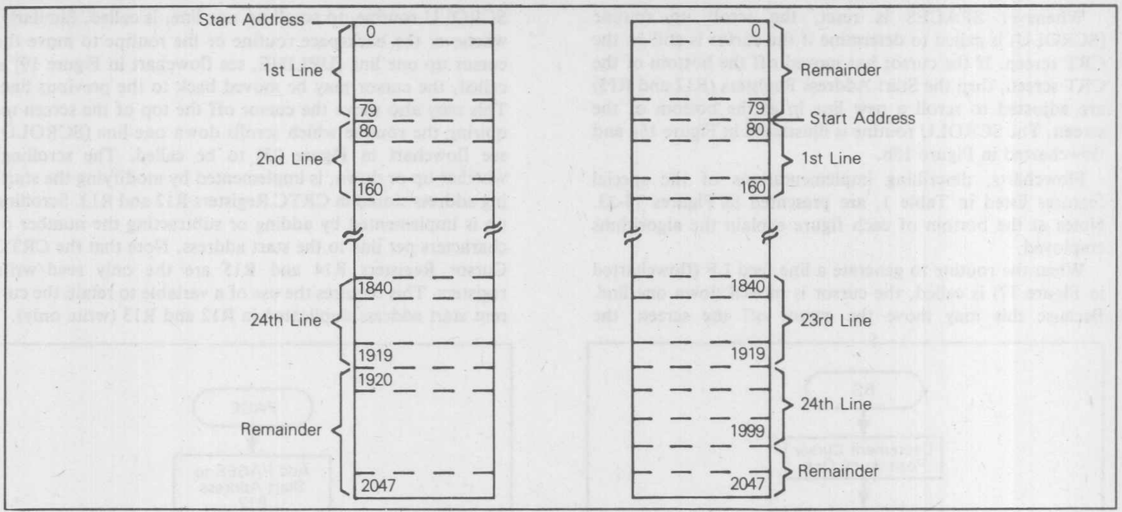


FIGURE 15a — Scrolling

Performed by changing the Start Address in R12 and R13 in the CRTC. This example shows how an 80 x 24 display is scrolled up one line.

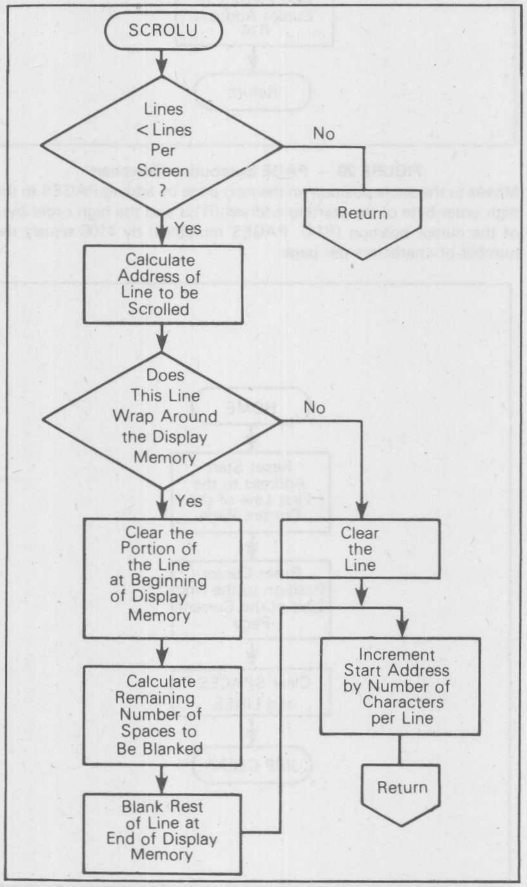

SCROLU
Lines < Lines Per Screen
No
Return
Yes
Calculate Address of Line to be Scrolled
Does This Line Wrap Around the Display Memory
No
Yes
Clear the Portion of the Line at Beginning of Display Memory
Clear the Line
Calculate Remaining Number of Spaces to Be Blanked
Increment Start Address by Number of Characters per Line
Blank Rest of Line at End of Display Memory
Return

FIGURE 15b — SCROLU Subroutine Flowchart

The 14-bit cursor address is checked to see if cursor has moved off the screen. If so, the 14-bit start address is incremented to add a new line (with the cursor) at the bottom.

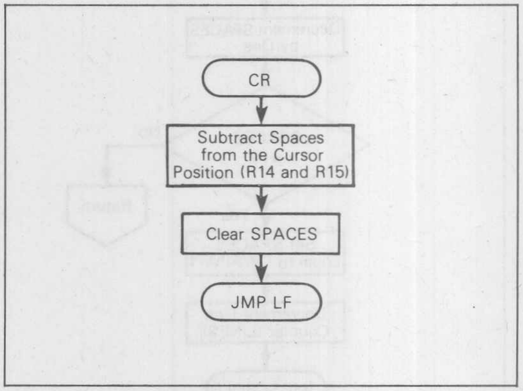


FIGURE 16 — CR Subroutine Flowchart

Generates a cursor return by subtracting SPACES (the space counter) from the current cursor position in R14 and R15 of the CRT. Jumps to LF to generate a line feed.

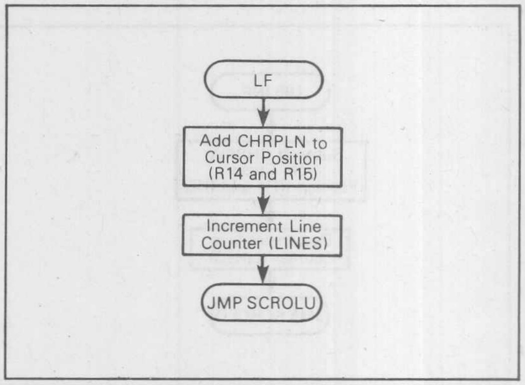

LF
Add CHRPLN to Cursor Position (R14 and R15)
Increment Line Counter (LINES)
JMP SCROLU

FIGURE 17 — LF Subroutine Flowchart

Generates a line feed by adding the number of characters per line to the current cursor position stored in R14 and R15 of the CRTC. Jumps to SCROLU to see if a new line should be scrolled on the page.

Whenever SPACES is reset, the scroll up routine (SCROLU) is called to determine if the cursor is still on the CRT screen. If the cursor has moved off the bottom of the CRT screen, then the Start Address Registers (R12 and R13) are adjusted to scroll a new line in at the bottom of the screen. The SCROLU routine is illustrated in Figure 15a and flowcharted in Figure 15b.

Flowcharts, describing implementations of the special features listed in Table 1, are presented in Figures 15-23. Notes at the bottom of each figure explain the algorithms employed.

When the routine to generate a line feed LF (flowcharted in Figure 17) is called, the cursor is moved down one line. Because this may move the cursor off the screen, the SCROLU routine, to scroll up one line, is called. Similarly, whenever the backspace routine or the routine to move the cursor up one line (UPLINE, see flowchart in Figure 19) is called, the cursor may be moved back to the previous line. This may also move the cursor off the top of the screen requiring the routine which scrolls down one line (SCROLD, see flowchart in Figure 23) to be called. The scrolling, whether up or down, is implemented by modifying the starting address stored in CRTC Registers R12 and R13. Scrolling up is implemented by adding or subtracting the number of characters per line to the start address. Note that the CRTC Cursor Registers R14 and R15 are the only read/write registers. This requires the use of a variable to retain the current start address duplicated in R12 and R13 (write only).

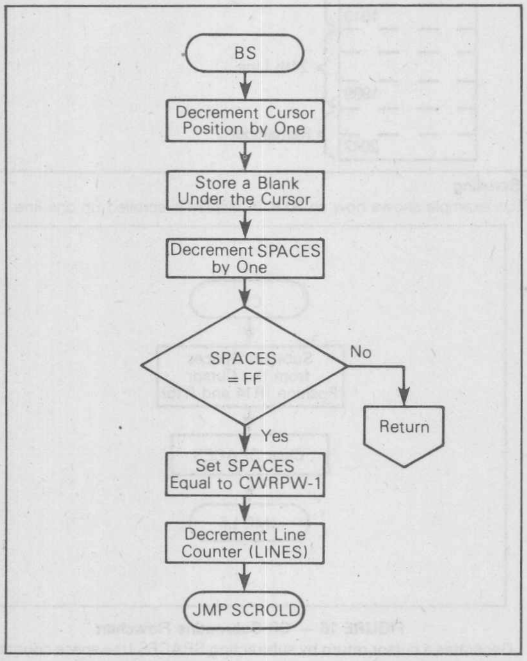

BS
Decrement Cursor Position by One
Store a Blank Under the Cursor
Decrement SPACES by One
SPACES = FF
No
Return
Yes
Set SPACES Equal to CWRPW-1
Decrement Line Counter (LINES)
JMP SCROLD

FIGURE 18 — BS Subroutine Flowchart
Backspaces and blanks under cursor. Jumps to SCROLD and checks if the cursor has moved off the top of the screen.

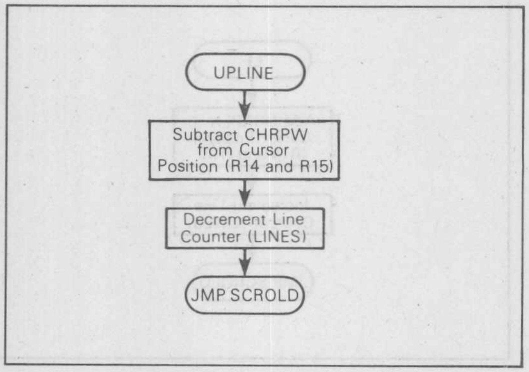

UPLINE
Subtract CHRPW from Cursor Position (R14 and R15)
Decrement Line Counter (LINES)
JMP SCROLD

FIGURE 19 — UPLINE Subroutine Flowchart
Moves the cursor up one line by subtracting the number of characters per line from current cursor position stored in R14 and R15 of the CRTC. Jumps to SCROLD to check if the cursor has moved off the top of the screen.

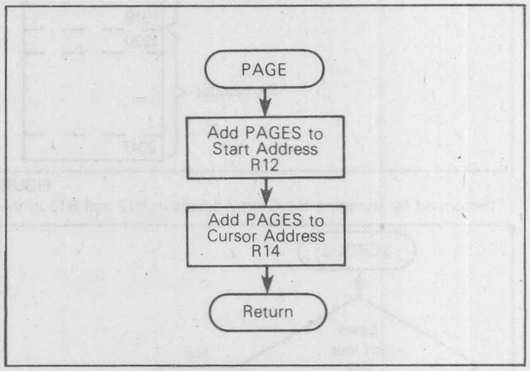

PAGE
Add PAGES to Start Address R12
Add PAGES to Cursor Address R14
Return

FIGURE 20 — PAGE Subroutine Flowchart
Moves to the same position on the next page by adding PAGES to the high order byte of the starting address (R12) and the high order byte of the cursor position (R14). PAGES multiplied by $100 equals the number of characters per page.

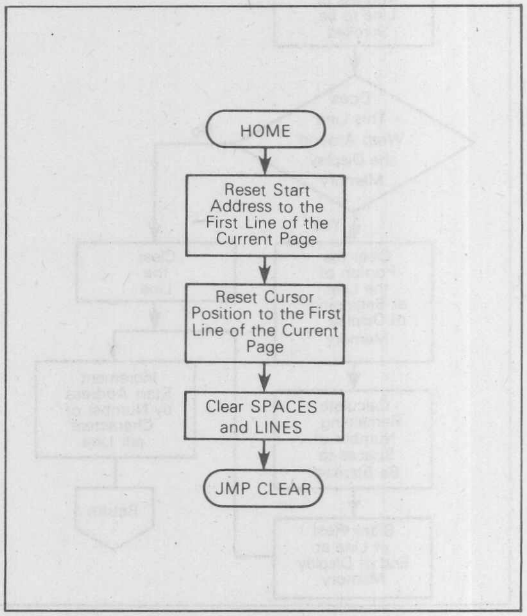

HOME
Reset Start Address to the First Line of the Current Page
Reset Cursor Position to the First Line of the Current Page
Clear SPACES and LINES
JMP CLEAR

FIGURE 21 — HOME Subroutine Flowchart
Reset start address and cursor position to the beginning of the current page, then clear SPACES and jump to CLEAR to put blanks in each display memory element of the current page.

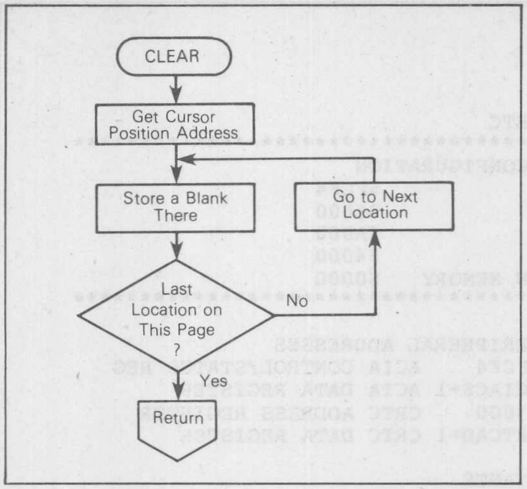


FIGURE 22 — CLEAR Subroutine Flowchart
Stores ASCII blank, code $20, into all memory locations on the current page starting at the cursor.

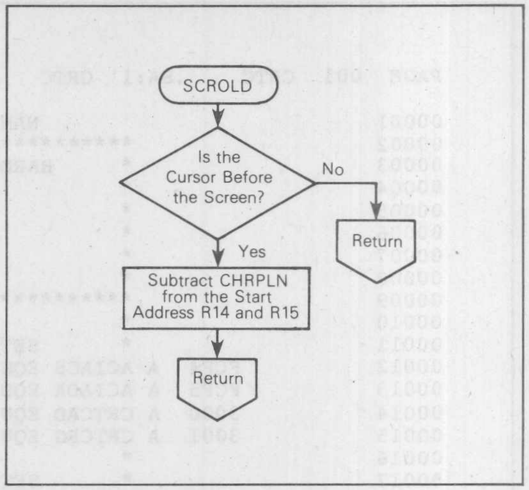
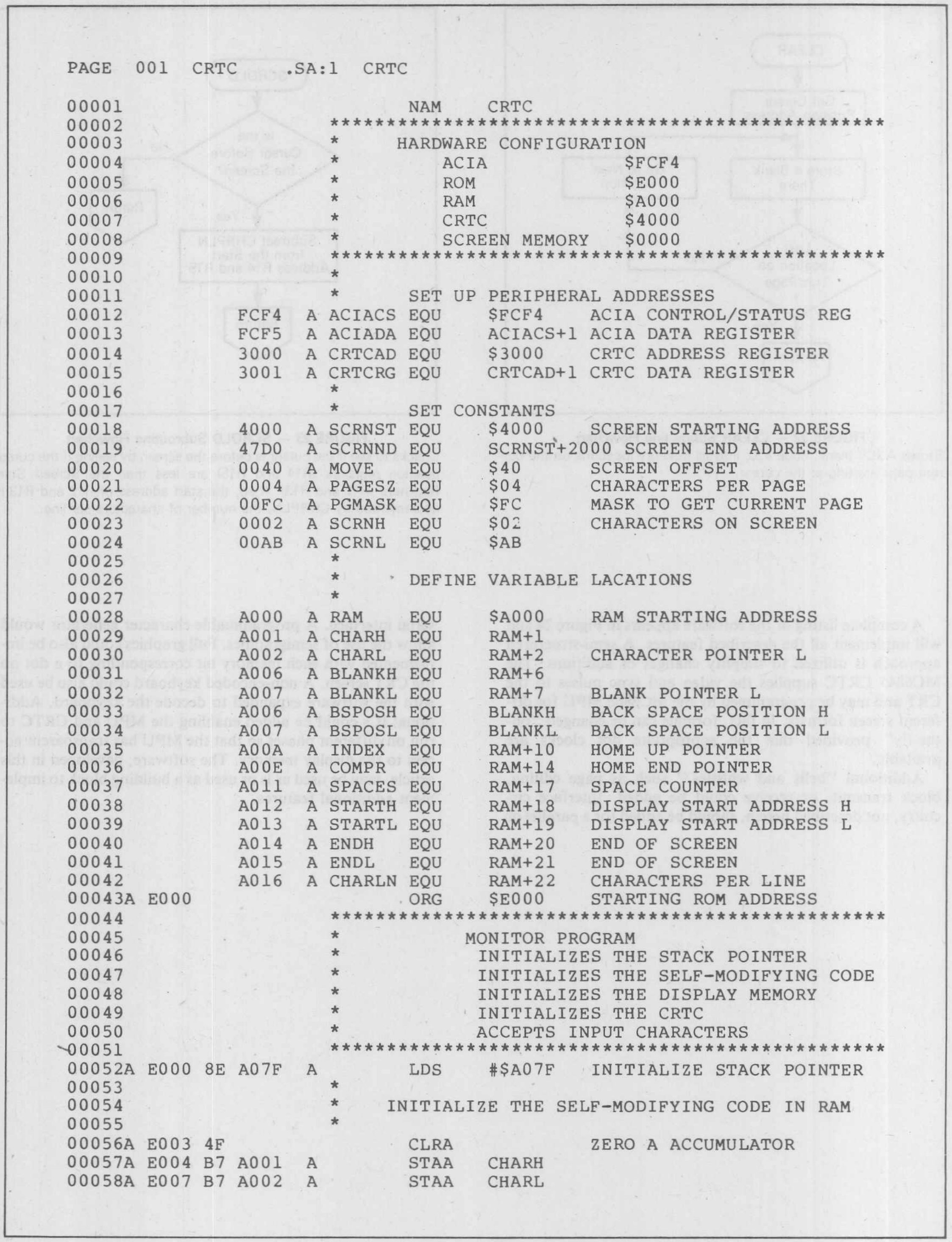

SCROLD
Is the Cursor Before the Screen?
No
Return
Yes
Subtract CHRPLN from the Start Address R14 and R15
Return

FIGURE 23 — SCROLD Subroutine Flowchart
Checks to see if the cursor is before the screen by seeing if the cursor position registers (R14 and R15) are less than the Screen Start Registers (R12 and R13). If so, the start address of R12 and R13 is decremented by CHRPLN, the number of characters per line.

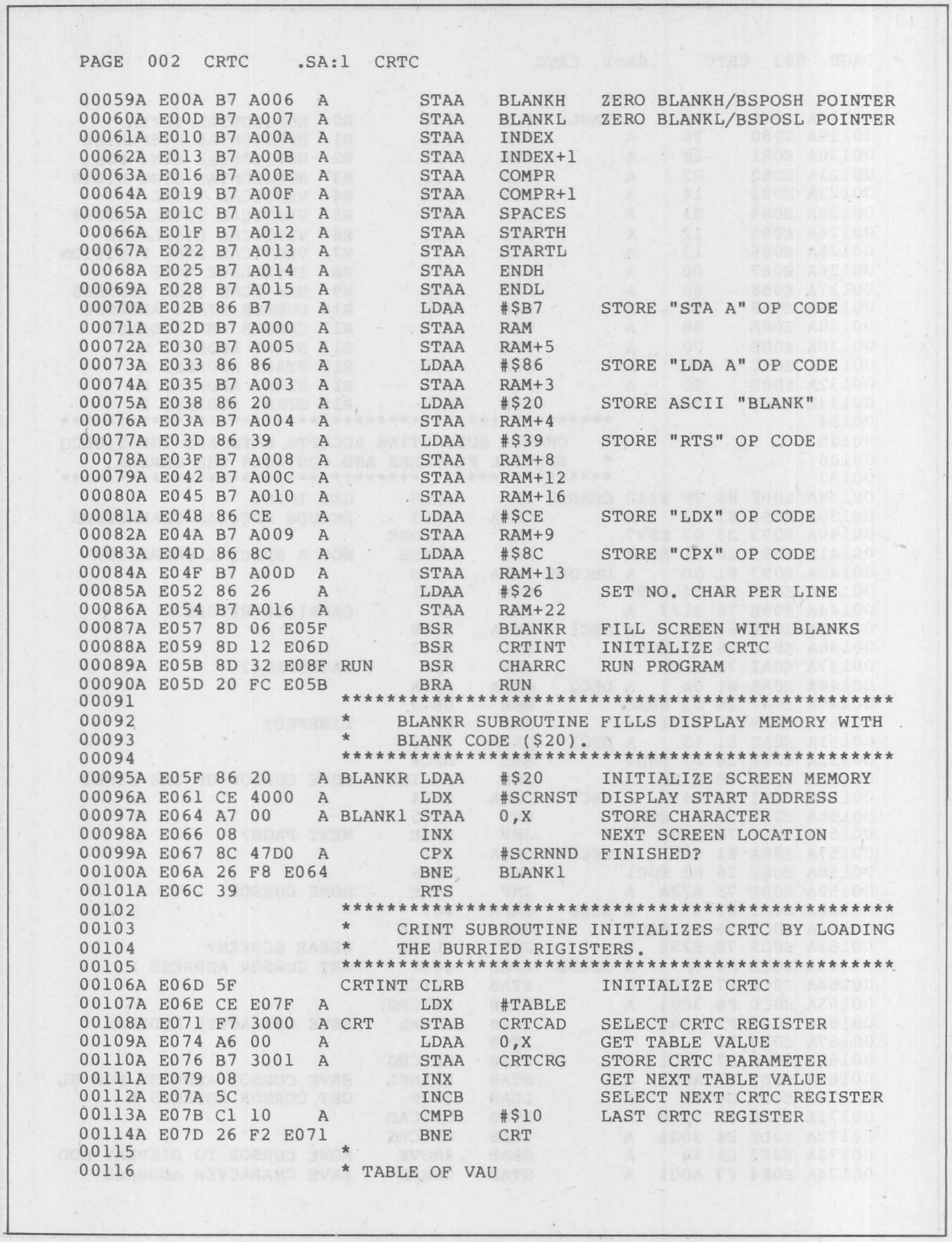
A complete listing of the software appears in Figure 24 and will implement all the described features. A semi-structured approach is utilized to simplify changes or additions. The MC6845 CRTC supplies the video and sync pulses to the CRT and may be programmed by the MC6808 MPU for different screen formats. In fact, formats can be changed "on-the-fly" provided that the appropriate dot clocks are available.

Additional "bells and whistles," such as page editing, block transmit, or receive could be added. Interface circuitry, not described herein, should be added for a parallel or serial interface. A programmable character generator would allow the use of semigraphics. Full graphics could also be implemented with each memory bit corresponding to a dot on the CRT screen. A non-encoded keyboard could also be used with the software expanded to decode the keyboard. Additional ICs could be added enabling the MPU and CRTC to run on different phases so that the MPU has transparent access to the display memory. The software, developed in this article, may be used as is or used as a building block to implement additional features.



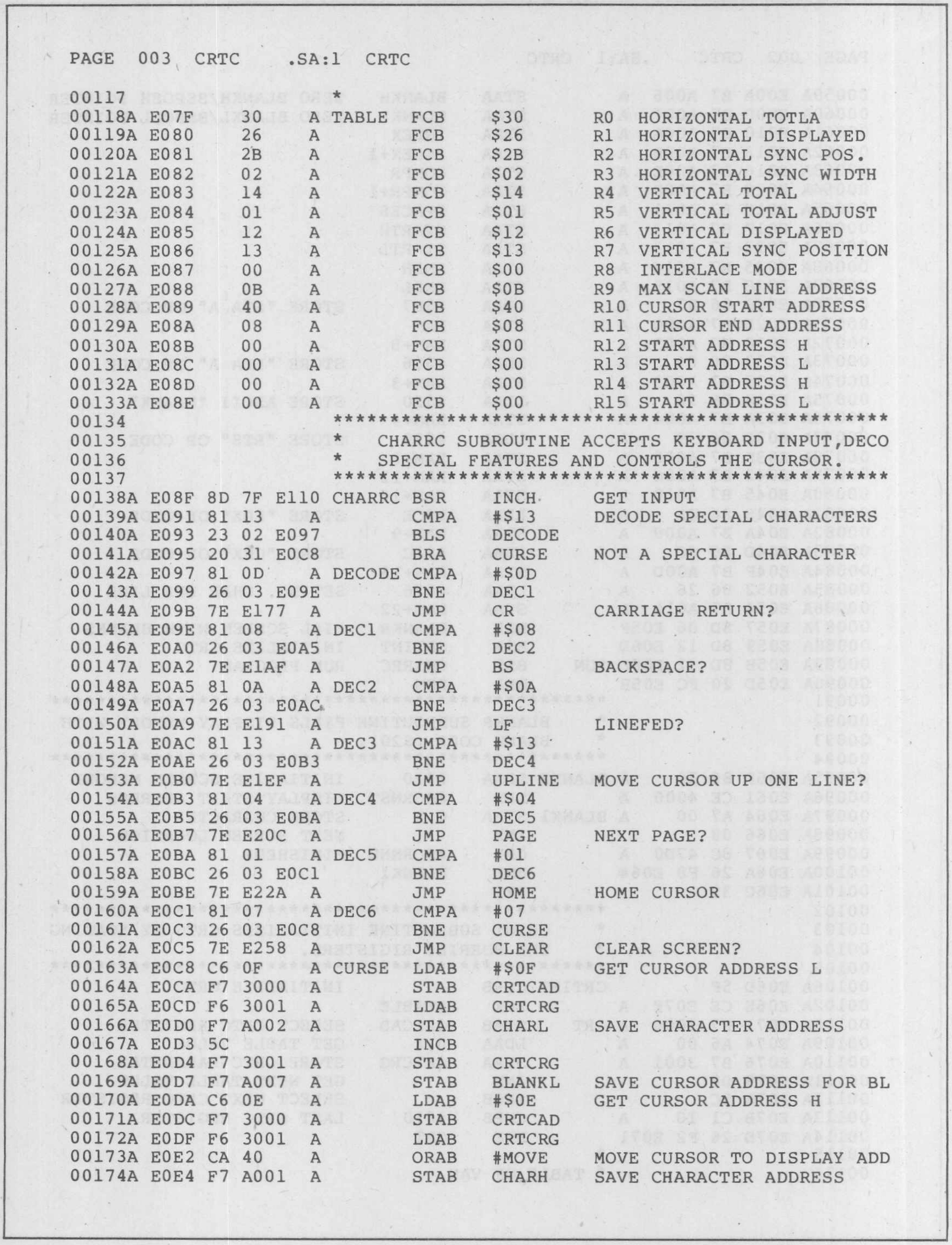
```
PAGE  001  CRTC      .SA:1  CRTC

                         NAM    CRTC
                  *************************************************
                  *     HARDWARE CONFIGURATION
                  *         ACIA            $FCF4
                  *         ROM             $E000
                  *         RAM             $A000
                  *         CRTC            $4000
                  *         SCREEN MEMORY   $0000
                  *************************************************
                  *
                  *      SET UP PERIPHERAL ADDRESSES
          FCF4  A ACIACS EQU    $FCF4    ACIA CONTROL/STATUS REG
          FCF5  A ACIADA EQU    ACIACS+1 ACIA DATA REGISTER
          3000  A CRTCAD EQU    $3000    CRTC ADDRESS REGISTER
          3001  A CRTCRG EQU    CRTCAD+1 CRTC DATA REGISTER
                  *
                  *      SET CONSTANTS
          4000  A SCRNST EQU    $4000    SCREEN STARTING ADDRESS
          47D0  A SCRNND EQU    SCRNST+2000 SCREEN END ADDRESS
          0040  A MOVE   EQU    $40      SCREEN OFFSET
          0004  A PAGESZ EQU    $04      CHARACTERS PER PAGE
          00FC  A PGMASK EQU    $FC      MASK TO GET CURRENT PAGE
          0002  A SCRNH  EQU    $02      CHARACTERS ON SCREEN
          00AB  A SCRNL  EQU    $AB
                  *
                  *      DEFINE VARIABLE LACATIONS
                  *
          A000  A RAM    EQU    $A000    RAM STARTING ADDRESS
          A001  A CHARH  EQU    RAM+1
          A002  A CHARL  EQU    RAM+2    CHARACTER POINTER L
          A006  A BLANKH EQU    RAM+6
          A007  A BLANKL EQU    RAM+7    BLANK POINTER L
          A006  A BSPOSH EQU    BLANKH   BACK SPACE POSITION H
          A007  A BSPOSL EQU    BLANKL   BACK SPACE POSITION L
          A00A  A INDEX  EQU    RAM+10   HOME UP POINTER
          A00E  A COMPR  EQU    RAM+14   HOME END POINTER
          A011  A SPACES EQU    RAM+17   SPACE COUNTER
          A012  A STARTH EQU    RAM+18   DISPLAY START ADDRESS H
          A013  A STARTL EQU    RAM+19   DISPLAY START ADDRESS L
          A014  A ENDH   EQU    RAM+20   END OF SCREEN
          A015  A ENDL   EQU    RAM+21   END OF SCREEN
          A016  A CHARLN EQU    RAM+22   CHARACTERS PER LINE
A E000                   ORG    $E000    STARTING ROM ADDRESS
                  *************************************************
                  *           MONITOR PROGRAM
                  *            INITIALIZES THE STACK POINTER
                  *            INITIALIZES THE SELF-MODIFYING CODE
                  *            INITIALIZES THE DISPLAY MEMORY
                  *            INITIALIZES THE CRTC
                  *            ACCEPTS INPUT CHARACTERS
                  *************************************************
A E000 8E A07F  A        LDS    #$A07F   INITIALIZE STACK POINTER
                  *
                  *    INITIALIZE THE SELF-MODIFYING CODE IN RAM
                  *
A E003 4F                CLRA            ZERO A ACCUMULATOR
A E004 B7 A001  A        STAA   CHARH
A E007 B7 A002  A        STAA   CHARL
```

FIGURE 24 — Complete Listing of CRTC Software



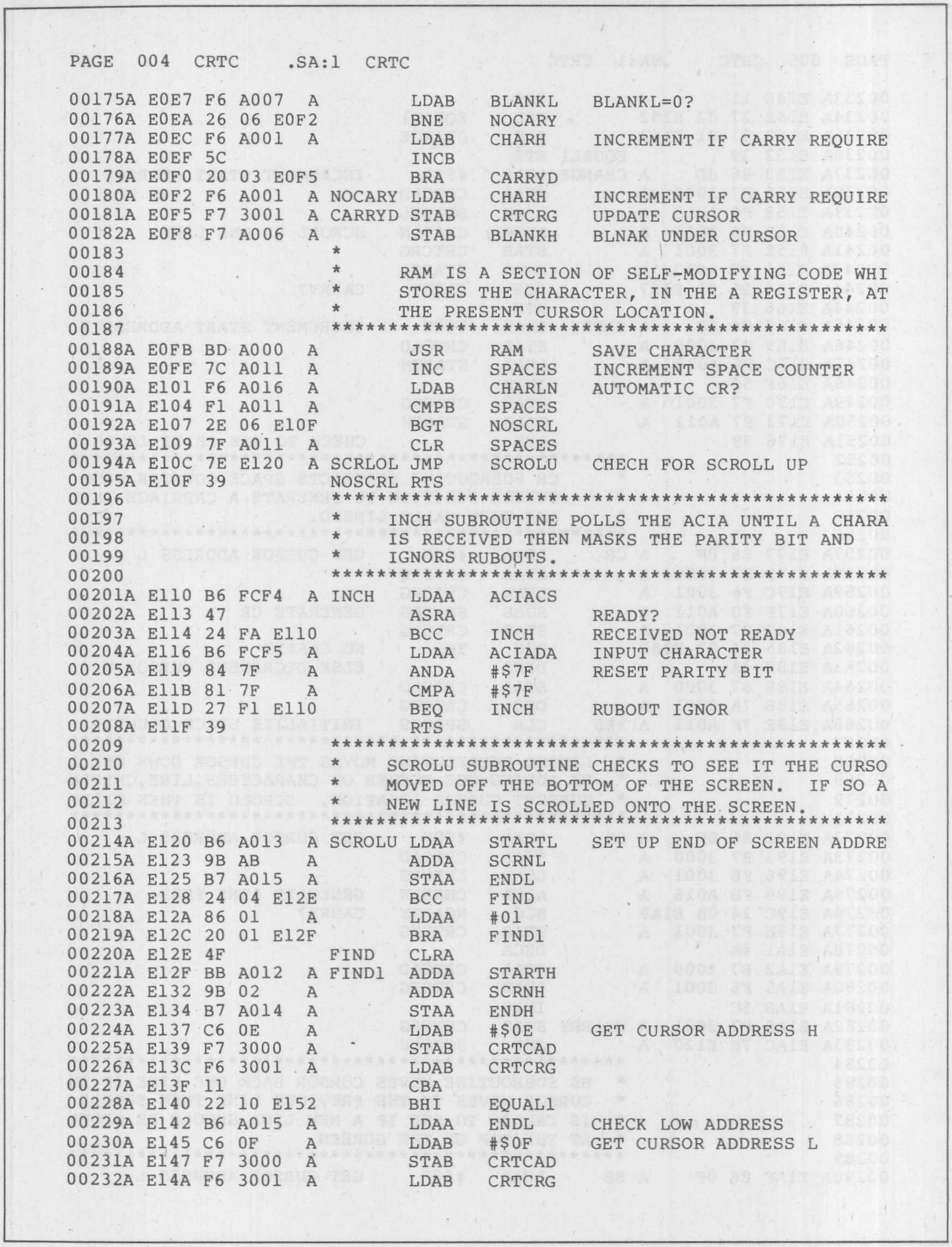
```
PAGE  002  CRTC    .SA:1  CRTC

00059A E00A B7 A006  A        STAA   BLANKH    ZERO BLANKH/BSPOSH POINTER
00060A E00D B7 A007  A        STAA   BLANKL    ZERO BLANKL/BSPOSL POINTER
00061A E010 B7 A00A  A        STAA   INDEX
00062A E013 B7 A00B  A        STAA   INDEX+1
00063A E016 B7 A00E  A        STAA   COMPR
00064A E019 B7 A00F  A        STAA   COMPR+1
00065A E01C B7 A011  A        STAA   SPACES
00066A E01F B7 A012  A        STAA   STARTH
00067A E022 B7 A013  A        STAA   STARTL
00068A E025 B7 A014  A        STAA   ENDH
00069A E028 B7 A015  A        STAA   ENDL
00070A E02B 86 B7    A        LDAA   #$B7      STORE "STA A" OP CODE
00071A E02D B7 A000  A        STAA   RAM
00072A E030 B7 A005  A        STAA   RAM+5
00073A E033 86 86    A        LDAA   #$86      STORE "LDA A" OP CODE
00074A E035 B7 A003  A        STAA   RAM+3
00075A E038 86 20    A        LDAA   #$20      STORE ASCII "BLANK"
00076A E03A B7 A004  A        STAA   RAM+4
00077A E03D 86 39    A        LDAA   #$39      STORE "RTS" OP CODE
00078A E03F B7 A008  A        STAA   RAM+8
00079A E042 B7 A00C  A        STAA   RAM+12
00080A E045 B7 A010  A        STAA   RAM+16
00081A E048 86 CE    A        LDAA   #$CE      STORE "LDX" OP CODE
00082A E04A B7 A009  A        STAA   RAM+9
00083A E04D 86 8C    A        LDAA   #$8C      STORE "CPX" OP CODE
00084A E04F B7 A00D  A        STAA   RAM+13
00085A E052 86 26    A        LDAA   #$26      SET NO. CHAR PER LINE
00086A E054 B7 A016  A        STAA   RAM+22
00087A E057 8D 06 E05F        BSR    BLANKR    FILL SCREEN WITH BLANKS
00088A E059 8D 12 E06D        BSR    CRTINT    INITIALIZE CRTC
00089A E05B 8D 32 E08F RUN    BSR    CHARRC    RUN PROGRAM
00090A E05D 20 FC E05B        BRA    RUN
00091                  ********************************************
00092                  *    BLANKR SUBROUTINE FILLS DISPLAY MEMORY WITH
00093                  *    BLANK CODE ($20).
00094                  ********************************************
00095A E05F 86 20    A BLANKR LDAA   #$20      INITIALIZE SCREEN MEMORY
00096A E061 CE 4000  A        LDX    #SCRNST   DISPLAY START ADDRESS
00097A E064 A7 00    A BLANK1 STAA   0,X       STORE CHARACTER
00098A E066 08                INX              NEXT SCREEN LOCATION
00099A E067 8C 47D0  A        CPX    #SCRNND   FINISHED?
00100A E06A 26 F8 E064        BNE    BLANK1
00101A E06C 39                RTS
00102                  ********************************************
00103                  *    CRINT SUBROUTINE INITIALIZES CRTC BY LOADING
00104                  *    THE BURRIED RIGISTERS.
00105                  ********************************************
00106A E06D 5F         CRTINT CLRB             INITIALIZE CRTC
00107A E06E CE E07F  A        LDX    #TABLE
00108A E071 F7 3000  A CRT    STAB   CRTCAD    SELECT CRTC REGISTER
00109A E074 A6 00    A        LDAA   0,X       GET TABLE VALUE
00110A E076 B7 3001  A        STAA   CRTCRG    STORE CRTC PARAMETER
00111A E079 08                INX              GET NEXT TABLE VALUE
00112A E07A 5C                INCB             SELECT NEXT CRTC REGISTER
00113A E07B C1 10    A        CMPB   #$10      LAST CRTC REGISTER
00114A E07D 26 F2 E071        BNE    CRT
00115                  *
00116                  * TABLE OF VAU
```

FIGURE 24 — Complete Listing of CRTC Software
(Continued)



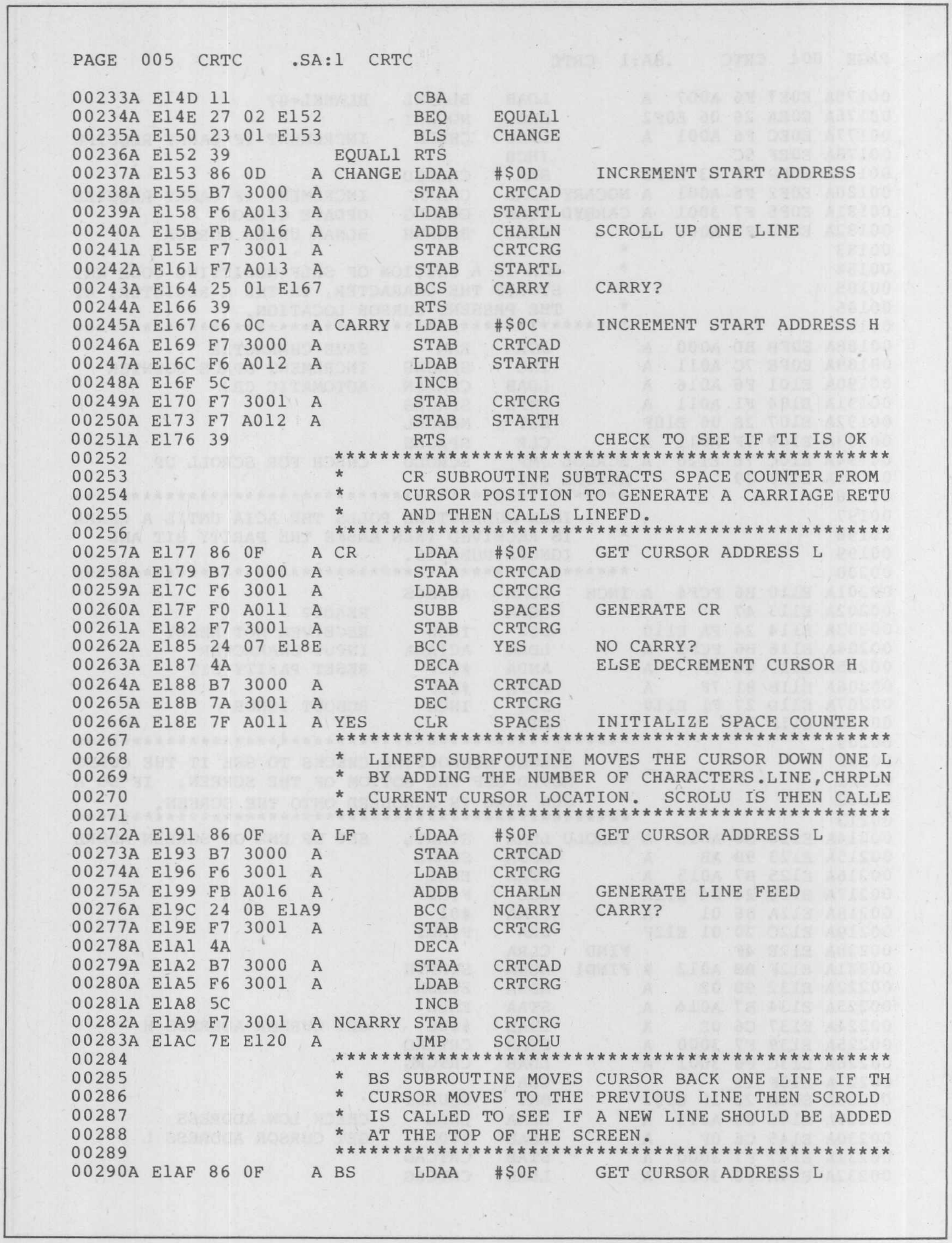
```
PAGE  003  CRTC     .SA:1  CRTC

00117                    *
00118A E07F     30    A TABLE  FCB    $30       R0  HORIZONTAL TOTLA
00119A E080     26    A        FCB    $26       R1  HORIZONTAL DISPLAYED
00120A E081     2B    A        FCB    $2B       R2  HORIZONTAL SYNC POS.
00121A E082     02    A        FCB    $02       R3  HORIZONTAL SYNC WIDTH
00122A E083     14    A        FCB    $14       R4  VERTICAL TOTAL
00123A E084     01    A        FCB    $01       R5  VERTICAL TOTAL ADJUST
00124A E085     12    A        FCB    $12       R6  VERTICAL DISPLAYED
00125A E086     13    A        FCB    $13       R7  VERTICAL SYNC POSITION
00126A E087     00    A        FCB    $00       R8  INTERLACE MODE
00127A E088     0B    A        FCB    $0B       R9  MAX SCAN LINE ADDRESS
00128A E089     40    A        FCB    $40       R10 CURSOR START ADDRESS
00129A E08A     08    A        FCB    $08       R11 CURSOR END ADDRESS
00130A E08B     00    A        FCB    $00       R12 START ADDRESS H
00131A E08C     00    A        FCB    $00       R13 START ADDRESS L
00132A E08D     00    A        FCB    $00       R14 START ADDRESS H
00133A E08E     00    A        FCB    $00       R15 START ADDRESS L
00134                    ************************************************
00135                    *    CHARRC SUBROUTINE ACCEPTS KEYBOARD INPUT,DECO
00136                    *    SPECIAL FEATURES AND CONTROLS THE CURSOR.
00137                    ************************************************
00138A E08F 8D 7F E110 CHARRC BSR    INCH      GET INPUT
00139A E091 81 13    A        CMPA   #$13      DECODE SPECIAL CHARACTERS
00140A E093 23 02 E097        BLS    DECODE
00141A E095 20 31 E0C8        BRA    CURSE     NOT A SPECIAL CHARACTER
00142A E097 81 0D    A DECODE CMPA   #$0D
00143A E099 26 03 E09E        BNE    DEC1
00144A E09B 7E E177  A        JMP    CR        CARRIAGE RETURN?
00145A E09E 81 08    A DEC1   CMPA   #$08
00146A E0A0 26 03 E0A5        BNE    DEC2
00147A E0A2 7E E1AF  A        JMP    BS        BACKSPACE?
00148A E0A5 81 0A    A DEC2   CMPA   #$0A
00149A E0A7 26 03 E0AC        BNE    DEC3
00150A E0A9 7E E191  A        JMP    LF        LINEFED?
00151A E0AC 81 13    A DEC3   CMPA   #$13
00152A E0AE 26 03 E0B3        BNE    DEC4
00153A E0B0 7E E1EF  A        JMP    UPLINE    MOVE CURSOR UP ONE LINE?
00154A E0B3 81 04    A DEC4   CMPA   #$04
00155A E0B5 26 03 E0BA        BNE    DEC5
00156A E0B7 7E E20C  A        JMP    PAGE      NEXT PAGE?
00157A E0BA 81 01    A DEC5   CMPA   #01
00158A E0BC 26 03 E0C1        BNE    DEC6
00159A E0BE 7E E22A  A        JMP    HOME      HOME CURSOR
00160A E0C1 81 07    A DEC6   CMPA   #07
00161A E0C3 26 03 E0C8        BNE    CURSE
00162A E0C5 7E E258  A        JMP    CLEAR     CLEAR SCREEN?
00163A E0C8 C6 0F    A CURSE  LDAB   #$0F      GET CURSOR ADDRESS L
00164A E0CA F7 3000  A        STAB   CRTCAD
00165A E0CD F6 3001  A        LDAB   CRTCRG
00166A E0D0 F7 A002  A        STAB   CHARL     SAVE CHARACTER ADDRESS
00167A E0D3 5C                INCB
00168A E0D4 F7 3001  A        STAB   CRTCRG
00169A E0D7 F7 A007  A        STAB   BLANKL    SAVE CURSOR ADDRESS FOR BL
00170A E0DA C6 0E    A        LDAB   #$0E      GET CURSOR ADDRESS H
00171A E0DC F7 3000  A        STAB   CRTCAD
00172A E0DF F6 3001  A        LDAB   CRTCRG
00173A E0E2 CA 40    A        ORAB   #MOVE     MOVE CURSOR TO DISPLAY ADD
00174A E0E4 F7 A001  A        STAB   CHARH     SAVE CHARACTER ADDRESS
```

FIGURE 24 — Complete Listing of CRTC Software
(Continued)



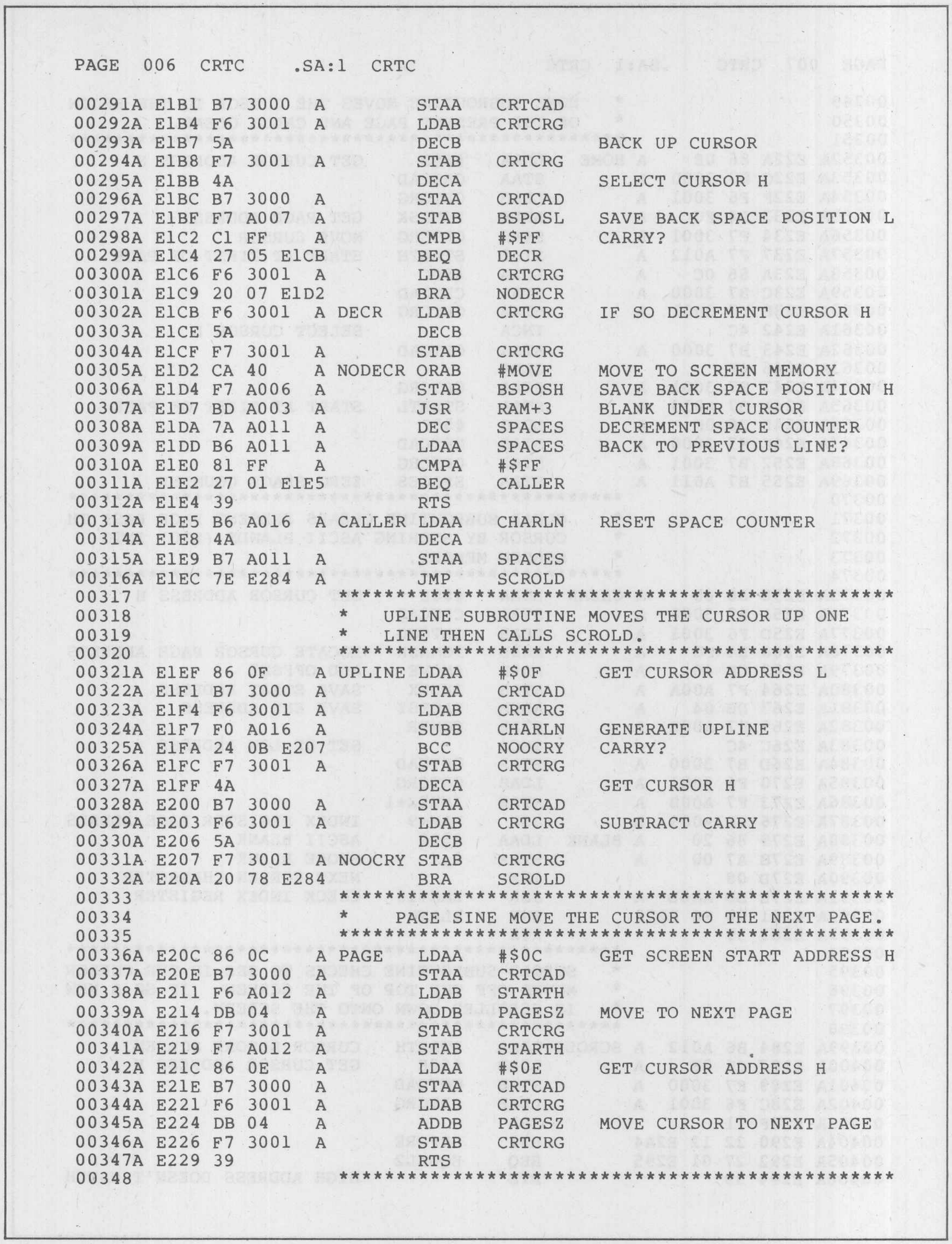
```
PAGE  004  CRTC    .SA:1  CRTC

00175A E0E7 F6 A007  A          LDAB   BLANKL    BLANKL=0?
00176A E0EA 26 06 E0F2          BNE    NOCARY
00177A E0EC F6 A001  A          LDAB   CHARH     INCREMENT IF CARRY REQUIRE
00178A E0EF 5C                  INCB
00179A E0F0 20 03 E0F5          BRA    CARRYD
00180A E0F2 F6 A001  A NOCARY   LDAB   CHARH     INCREMENT IF CARRY REQUIRE
00181A E0F5 F7 3001  A CARRYD   STAB   CRTCRG    UPDATE CURSOR
00182A E0F8 F7 A006  A          STAB   BLANKH    BLNAK UNDER CURSOR
00183                  *
00184                  *        RAM IS A SECTION OF SELF-MODIFYING CODE WHI
00185                  *        STORES THE CHARACTER, IN THE A REGISTER, AT
00186                  *        THE PRESENT CURSOR LOCATION.
00187                  *************************************************
00188A E0FB BD A000  A          JSR    RAM       SAVE CHARACTER
00189A E0FE 7C A011  A          INC    SPACES    INCREMENT SPACE COUNTER
00190A E101 F6 A016  A          LDAB   CHARLN    AUTOMATIC CR?
00191A E104 F1 A011  A          CMPB   SPACES
00192A E107 2E 06 E10F          BGT    NOSCRL
00193A E109 7F A011  A          CLR    SPACES
00194A E10C 7E E120  A SCRLOL   JMP    SCROLU    CHECH FOR SCROLL UP
00195A E10F 39         NOSCRL   RTS
00196                  *************************************************
00197                  *       INCH SUBROUTINE POLLS THE ACIA UNTIL A CHARA
00198                  *       IS RECEIVED THEN MASKS THE PARITY BIT AND
00199                  *       IGNORS RUBOUTS.
00200                  *************************************************
00201A E110 B6 FCF4  A INCH     LDAA   ACIACS
00202A E113 47                  ASRA             READY?
00203A E114 24 FA E110          BCC    INCH      RECEIVED NOT READY
00204A E116 B6 FCF5  A          LDAA   ACIADA    INPUT CHARACTER
00205A E119 84 7F               ANDA   #$7F      RESET PARITY BIT
00206A E11B 81 7F    A          CMPA   #$7F
00207A E11D 27 F1 E110          BEQ    INCH      RUBOUT IGNOR
00208A E11F 39                  RTS
00209                  *************************************************
00210                  *       SCROLU SUBROUTINE CHECKS TO SEE IT THE CURSO
00211                  *       MOVED OFF THE BOTTOM OF THE SCREEN.  IF SO A
00212                  *       NEW LINE IS SCROLLED ONTO THE SCREEN.
00213                  *************************************************
00214A E120 B6 A013  A SCROLU   LDAA   STARTL    SET UP END OF SCREEN ADDRE
00215A E123 9B AB    A          ADDA   SCRNL
00216A E125 B7 A015  A          STAA   ENDL
00217A E128 24 04 E12E          BCC    FIND
00218A E12A 86 01    A          LDAA   #01
00219A E12C 20 01 E12F          BRA    FIND1
00220A E12E 4F         FIND     CLRA
00221A E12F BB A012  A FIND1    ADDA   STARTH
00222A E132 9B 02    A          ADDA   SCRNH
00223A E134 B7 A014  A          STAA   ENDH
00224A E137 C6 0E    A          LDAB   #$0E      GET CURSOR ADDRESS H
00225A E139 F7 3000  A          STAB   CRTCAD
00226A E13C F6 3001  A          LDAB   CRTCRG
00227A E13F 11                  CBA
00228A E140 22 10 E152          BHI    EQUAL1
00229A E142 B6 A015  A          LDAA   ENDL      CHECK LOW ADDRESS
00230A E145 C6 0F    A          LDAB   #$0F      GET CURSOR ADDRESS L
00231A E147 F7 3000  A          STAB   CRTCAD
00232A E14A F6 3001  A          LDAB   CRTCRG
```

FIGURE 24 — Complete Listing of CRTC Software (Continued)



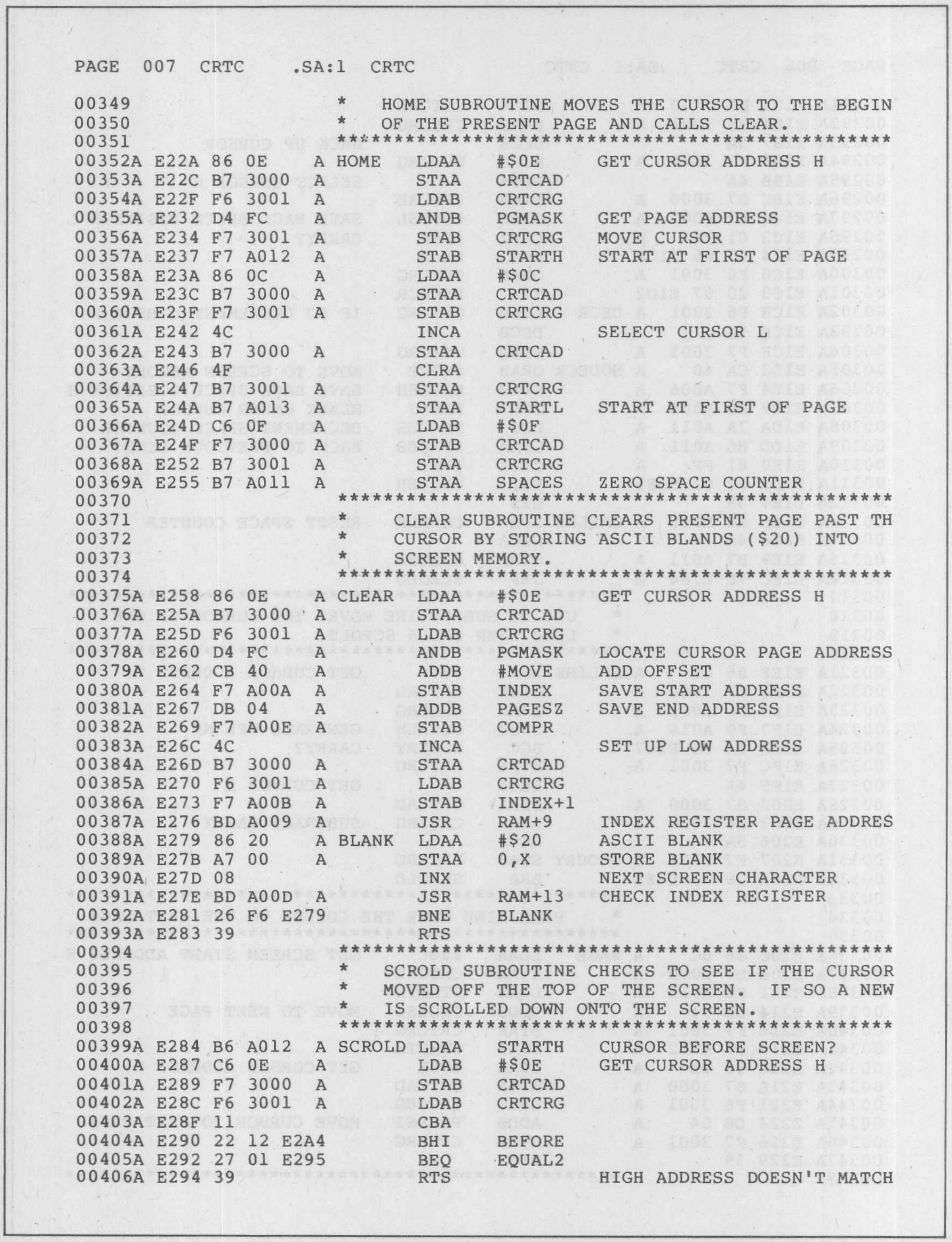
```
PAGE  005  CRTC     .SA:1  CRTC

00233A E14D 11                   CBA
00234A E14E 27 02 E152           BEQ   EQUAL1
00235A E150 23 01 E153           BLS   CHANGE
00236A E152 39          EQUAL1 RTS
00237A E153 86 0D     A CHANGE LDAA  #$0D      INCREMENT START ADDRESS
00238A E155 B7 3000   A          STAA  CRTCAD
00239A E158 F6 A013   A          LDAB  STARTL
00240A E15B FB A016   A          ADDB  CHARLN    SCROLL UP ONE LINE
00241A E15E F7 3001   A          STAB  CRTCRG
00242A E161 F7 A013   A          STAB  STARTL
00243A E164 25 01 E167           BCS   CARRY     CARRY?
00244A E166 39                   RTS
00245A E167 C6 0C     A CARRY    LDAB  #$0C      INCREMENT START ADDRESS H
00246A E169 F7 3000   A          STAB  CRTCAD
00247A E16C F6 A012   A          LDAB  STARTH
00248A E16F 5C                   INCB
00249A E170 F7 3001   A          STAB  CRTCRG
00250A E173 F7 A012   A          STAB  STARTH
00251A E176 39                   RTS             CHECK TO SEE IF TI IS OK
00252                   ************************************************
00253                   *     CR SUBROUTINE SUBTRACTS SPACE COUNTER FROM
00254                   *     CURSOR POSITION TO GENERATE A CARRIAGE RETU
00255                   *     AND THEN CALLS LINEFD.
00256                   ************************************************
00257A E177 86 0F     A CR       LDAA  #$0F      GET CURSOR ADDRESS L
00258A E179 B7 3000   A          STAA  CRTCAD
00259A E17C F6 3001   A          LDAB  CRTCRG
00260A E17F F0 A011   A          SUBB  SPACES    GENERATE CR
00261A E182 F7 3001   A          STAB  CRTCRG
00262A E185 24 07 E18E           BCC   YES       NO CARRY?
00263A E187 4A                   DECA            ELSE DECREMENT CURSOR H
00264A E188 B7 3000   A          STAA  CRTCAD
00265A E18B 7A 3001   A          DEC   CRTCRG
00266A E18E 7F A011   A YES      CLR   SPACES    INITIALIZE SPACE COUNTER
00267                   ************************************************
00268                   * LINEFD SUBFROUTINE MOVES THE CURSOR DOWN ONE L
00269                   * BY ADDING THE NUMBER OF CHARACTERS.LINE,CHRPLN
00270                   * CURRENT CURSOR LOCATION.  SCROLU IS THEN CALLE
00271                   ************************************************
00272A E191 86 0F     A LF       LDAA  #$0F      GET CURSOR ADDRESS L
00273A E193 B7 3000   A          STAA  CRTCAD
00274A E196 F6 3001   A          LDAB  CRTCRG
00275A E199 FB A016   A          ADDB  CHARLN    GENERATE LINE FEED
00276A E19C 24 0B E1A9           BCC   NCARRY    CARRY?
00277A E19E F7 3001   A          STAB  CRTCRG
00278A E1A1 4A                   DECA
00279A E1A2 B7 3000   A          STAA  CRTCAD
00280A E1A5 F6 3001   A          LDAB  CRTCRG
00281A E1A8 5C                   INCB
00282A E1A9 F7 3001   A NCARRY STAB  CRTCRG
00283A E1AC 7E E120   A          JMP   SCROLU
00284                   ************************************************
00285                   *  BS SUBROUTINE MOVES CURSOR BACK ONE LINE IF TH
00286                   *  CURSOR MOVES TO THE PREVIOUS LINE THEN SCROLD
00287                   *  IS CALLED TO SEE IF A NEW LINE SHOULD BE ADDED
00288                   *  AT THE TOP OF THE SCREEN.
00289                   ************************************************
00290A E1AF 86 0F     A BS       LDAA  #$0F      GET CURSOR ADDRESS L
```

FIGURE 24 — Complete Listing of CRTC Software
(Continued)



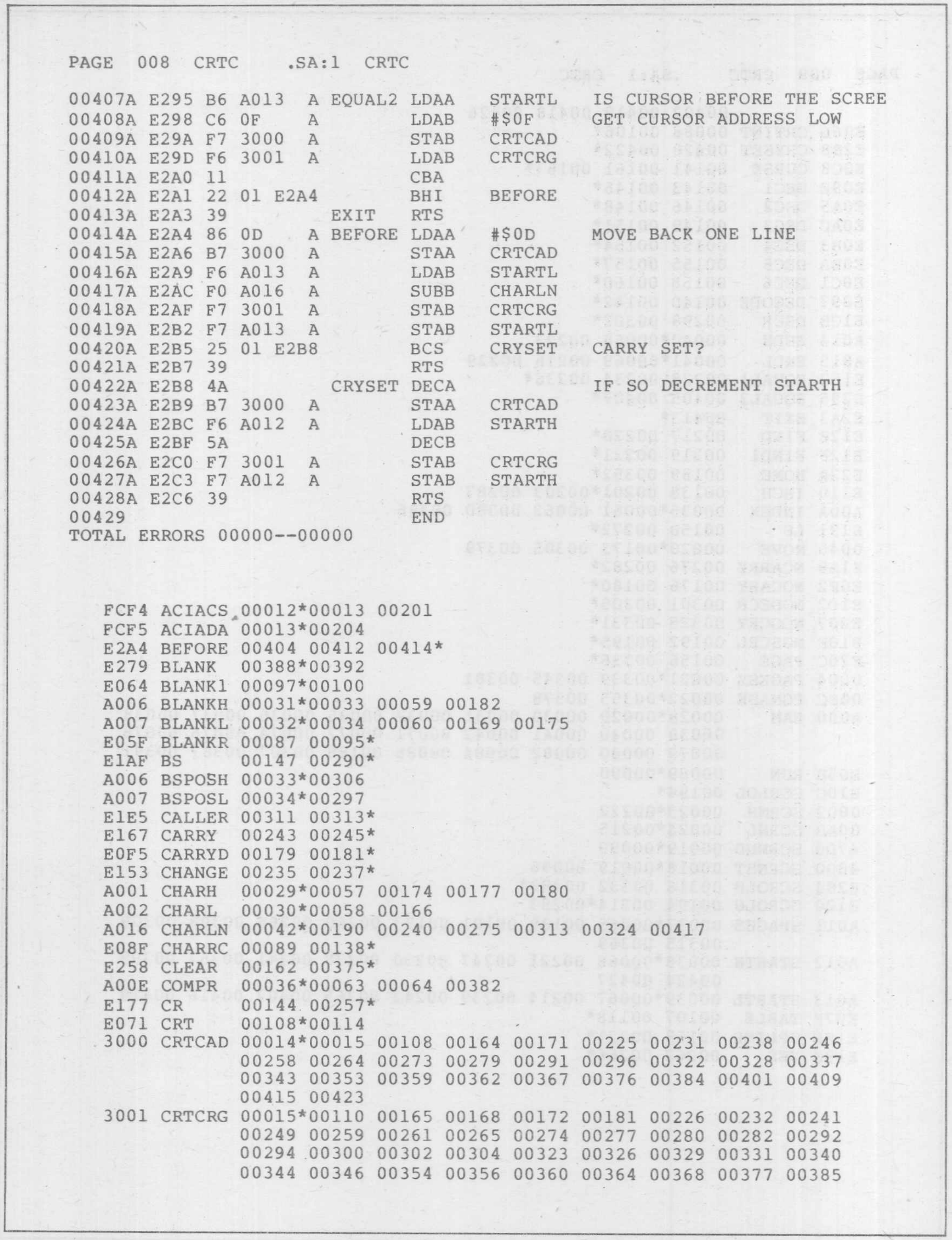
```
PAGE  006  CRTC     .SA:1  CRTC

00291A E1B1 B7 3000  A        STAA   CRTCAD
00292A E1B4 F6 3001  A        LDAB   CRTCRG
00293A E1B7 5A                DECB            BACK UP CURSOR
00294A E1B8 F7 3001  A        STAB   CRTCRG
00295A E1BB 4A                DECA            SELECT CURSOR H
00296A E1BC B7 3000  A        STAA   CRTCAD
00297A E1BF F7 A007  A        STAB   BSPOSL   SAVE BACK SPACE POSITION L
00298A E1C2 C1 FF    A        CMPB   #$FF     CARRY?
00299A E1C4 27 05 E1CB        BEQ    DECR
00300A E1C6 F6 3001  A        LDAB   CRTCRG
00301A E1C9 20 07 E1D2        BRA    NODECR
00302A E1CB F6 3001  A DECR   LDAB   CRTCRG   IF SO DECREMENT CURSOR H
00303A E1CE 5A                DECB
00304A E1CF F7 3001  A        STAB   CRTCRG
00305A E1D2 CA 40    A NODECR ORAB   #MOVE    MOVE TO SCREEN MEMORY
00306A E1D4 F7 A006  A        STAB   BSPOSH   SAVE BACK SPACE POSITION H
00307A E1D7 BD A003  A        JSR    RAM+3    BLANK UNDER CURSOR
00308A E1DA 7A A011  A        DEC    SPACES   DECREMENT SPACE COUNTER
00309A E1DD B6 A011  A        LDAA   SPACES   BACK TO PREVIOUS LINE?
00310A E1E0 81 FF    A        CMPA   #$FF
00311A E1E2 27 01 E1E5        BEQ    CALLER
00312A E1E4 39                RTS
00313A E1E5 B6 A016  A CALLER LDAA   CHARLN   RESET SPACE COUNTER
00314A E1E8 4A                DECA
00315A E1E9 B7 A011  A        STAA   SPACES
00316A E1EC 7E E284  A        JMP    SCROLD
00317                  ************************************************
00318                  *   UPLINE SUBROUTINE MOVES THE CURSOR UP ONE
00319                  *   LINE THEN CALLS SCROLD.
00320                  ************************************************
00321A E1EF 86 0F    A UPLINE LDAA   #$0F     GET CURSOR ADDRESS L
00322A E1F1 B7 3000  A        STAA   CRTCAD
00323A E1F4 F6 3001  A        LDAB   CRTCRG
00324A E1F7 F0 A016  A        SUBB   CHARLN   GENERATE UPLINE
00325A E1FA 24 0B E207        BCC    NOOCRY   CARRY?
00326A E1FC F7 3001  A        STAB   CRTCRG
00327A E1FF 4A                DECA            GET CURSOR H
00328A E200 B7 3000  A        STAA   CRTCAD
00329A E203 F6 3001  A        LDAB   CRTCRG   SUBTRACT CARRY
00330A E206 5A                DECB
00331A E207 F7 3001  A NOOCRY STAB   CRTCRG
00332A E20A 20 78 E284        BRA    SCROLD
00333                  ************************************************
00334                  *    PAGE SINE MOVE THE CURSOR TO THE NEXT PAGE.
00335                  ************************************************
00336A E20C 86 0C    A PAGE   LDAA   #$0C     GET SCREEN START ADDRESS H
00337A E20E B7 3000  A        STAA   CRTCAD
00338A E211 F6 A012  A        LDAB   STARTH
00339A E214 DB 04    A        ADDB   PAGESZ   MOVE TO NEXT PAGE
00340A E216 F7 3001  A        STAB   CRTCRG
00341A E219 F7 A012  A        STAB   STARTH
00342A E21C 86 0E    A        LDAA   #$0E     GET CURSOR ADDRESS H
00343A E21E B7 3000  A        STAA   CRTCAD
00344A E221 F6 3001  A        LDAB   CRTCRG
00345A E224 DB 04    A        ADDB   PAGESZ   MOVE CURSOR TO NEXT PAGE
00346A E226 F7 3001  A        STAB   CRTCRG
00347A E229 39                RTS
00348                  ************************************************
```

FIGURE 24.— Complete Listing of CRTC Software
(Continued)



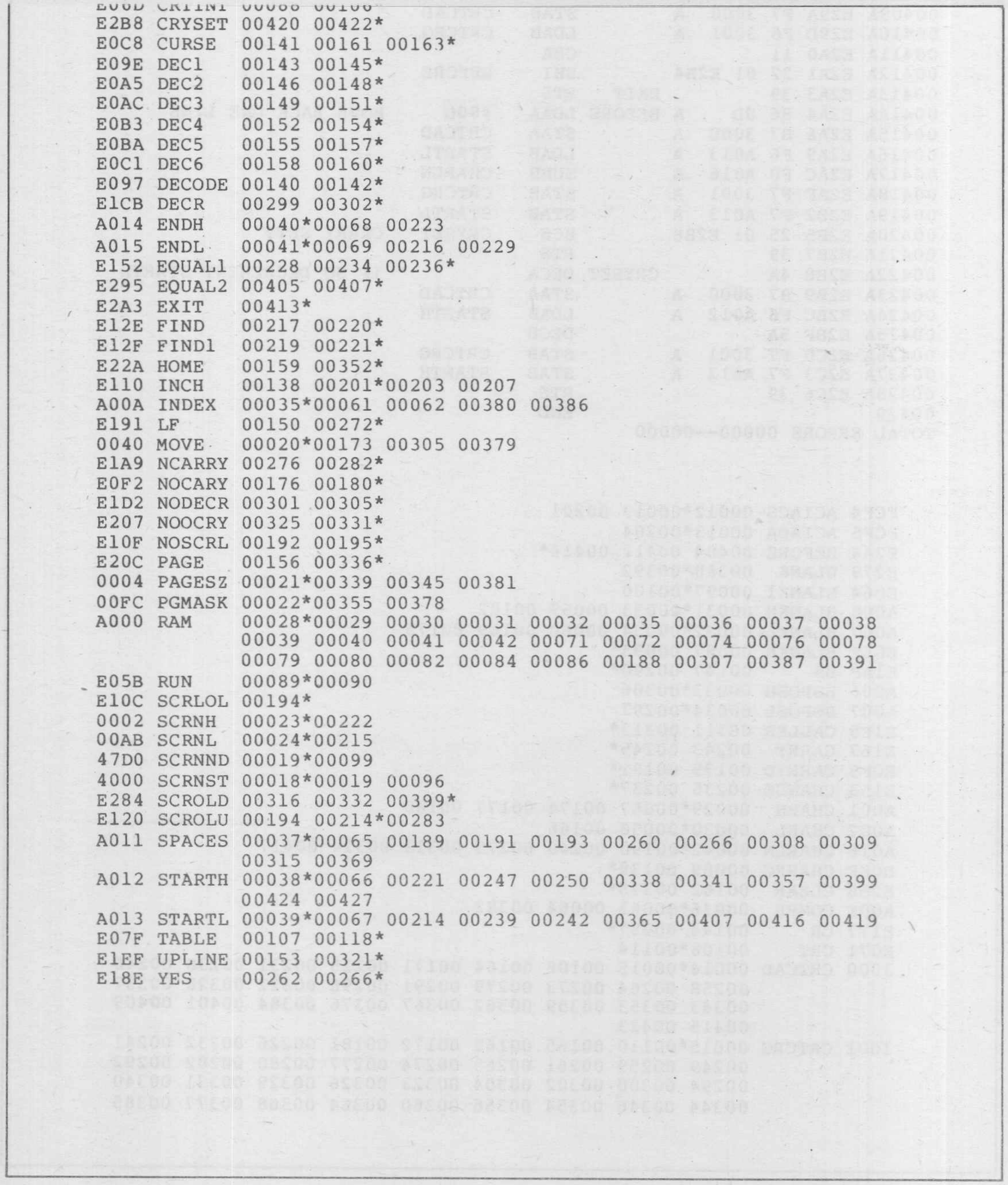
```
PAGE  007  CRTC    .SA:1  CRTC

00349                     *    HOME SUBROUTINE MOVES THE CURSOR TO THE BEGIN
00350                     *    OF THE PRESENT PAGE AND CALLS CLEAR.
00351                     ************************************************
00352A E22A 86 0E     A HOME   LDAA   #$0E      GET CURSOR ADDRESS H
00353A E22C B7 3000  A         STAA   CRTCAD
00354A E22F F6 3001  A         LDAB   CRTCRG
00355A E232 D4 FC    A         ANDB   PGMASK    GET PAGE ADDRESS
00356A E234 F7 3001  A         STAB   CRTCRG    MOVE CURSOR
00357A E237 F7 A012  A         STAB   STARTH    START AT FIRST OF PAGE
00358A E23A 86 0C    A         LDAA   #$0C
00359A E23C B7 3000  A         STAA   CRTCAD
00360A E23F F7 3001  A         STAB   CRTCRG
00361A E242 4C                 INCA             SELECT CURSOR L
00362A E243 B7 3000  A         STAA   CRTCAD
00363A E246 4F                 CLRA
00364A E247 B7 3001  A         STAA   CRTCRG
00365A E24A B7 A013  A         STAA   STARTL    START AT FIRST OF PAGE
00366A E24D C6 0F    A         LDAB   #$0F
00367A E24F F7 3000  A         STAB   CRTCAD
00368A E252 B7 3001  A         STAA   CRTCRG
00369A E255 B7 A011  A         STAA   SPACES    ZERO SPACE COUNTER
00370                     ************************************************
00371                     *     CLEAR SUBROUTINE CLEARS PRESENT PAGE PAST TH
00372                     *     CURSOR BY STORING ASCII BLANDS ($20) INTO
00373                     *     SCREEN MEMORY.
00374                     ************************************************
00375A E258 86 0E     A CLEAR  LDAA   #$0E      GET CURSOR ADDRESS H
00376A E25A B7 3000  A         STAA   CRTCAD
00377A E25D F6 3001  A         LDAB   CRTCRG
00378A E260 D4 FC    A         ANDB   PGMASK    LOCATE CURSOR PAGE ADDRESS
00379A E262 CB 40    A         ADDB   #MOVE     ADD OFFSET
00380A E264 F7 A00A  A         STAB   INDEX     SAVE START ADDRESS
00381A E267 DB 04    A         ADDB   PAGESZ    SAVE END ADDRESS
00382A E269 F7 A00E  A         STAB   COMPR
00383A E26C 4C                 INCA             SET UP LOW ADDRESS
00384A E26D B7 3000  A         STAA   CRTCAD
00385A E270 F6 3001  A         LDAB   CRTCRG
00386A E273 F7 A00B  A         STAB   INDEX+1
00387A E276 BD A009  A         JSR    RAM+9     INDEX REGISTER PAGE ADDRES
00388A E279 86 20     A BLANK  LDAA   #$20      ASCII BLANK
00389A E27B A7 00    A         STAA   0,X       STORE BLANK
00390A E27D 08                 INX              NEXT SCREEN CHARACTER
00391A E27E BD A00D  A         JSR    RAM+13    CHECK INDEX REGISTER
00392A E281 26 F6 E279         BNE    BLANK
00393A E283 39                 RTS
00394                     ************************************************
00395                     *    SCROLD SUBROUTINE CHECKS TO SEE IF THE CURSOR
00396                     *    MOVED OFF THE TOP OF THE SCREEN.  IF SO A NEW
00397                     *    IS SCROLLED DOWN ONTO THE SCREEN.
00398                     ************************************************
00399A E284 B6 A012  A SCROLD LDAA   STARTH    CURSOR BEFORE SCREEN?
00400A E287 C6 0E    A         LDAB   #$0E      GET CURSOR ADDRESS H
00401A E289 F7 3000  A         STAB   CRTCAD
00402A E28C F6 3001  A         LDAB   CRTCRG
00403A E28F 11                 CBA
00404A E290 22 12 E2A4         BHI    BEFORE
00405A E292 27 01 E295         BEQ    EQUAL2
00406A E294 39                 RTS              HIGH ADDRESS DOESN'T MATCH
```

FIGURE 24 — Complete Listing of CRTC Software
(Continued)



```
PAGE  008  CRTC     .SA:1  CRTC

00407A E295 B6 A013  A EQUAL2 LDAA   STARTL    IS CURSOR BEFORE THE SCREE
00408A E298 C6 0F    A        LDAB   #$0F      GET CURSOR ADDRESS LOW
00409A E29A F7 3000  A        STAB   CRTCAD
00410A E29D F6 3001  A        LDAB   CRTCRG
00411A E2A0 11                CBA
00412A E2A1 22 01 E2A4        BHI    BEFORE
00413A E2A3 39         EXIT   RTS
00414A E2A4 86 0D    A BEFORE LDAA   #$0D      MOVE BACK ONE LINE
00415A E2A6 B7 3000  A        STAA   CRTCAD
00416A E2A9 F6 A013  A        LDAB   STARTL
00417A E2AC F0 A016  A        SUBB   CHARLN
00418A E2AF F7 3001  A        STAB   CRTCRG
00419A E2B2 F7 A013  A        STAB   STARTL
00420A E2B5 25 01 E2B8        BCS    CRYSET    CARRY SET?
00421A E2B7 39                RTS
00422A E2B8 4A         CRYSET DECA             IF SO DECREMENT STARTH
00423A E2B9 B7 3000  A        STAA   CRTCAD
00424A E2BC F6 A012  A        LDAB   STARTH
00425A E2BF 5A                DECB
00426A E2C0 F7 3001  A        STAB   CRTCRG
00427A E2C3 F7 A012  A        STAB   STARTH
00428A E2C6 39                RTS
00429                         END
TOTAL ERRORS 00000--00000


   FCF4 ACIACS 00012*00013 00201
   FCF5 ACIADA 00013*00204
   E2A4 BEFORE 00404 00412 00414*
   E279 BLANK  00388*00392
   E064 BLANK1 00097*00100
   A006 BLANKH 00031*00033 00059 00182
   A007 BLANKL 00032*00034 00060 00169 00175
   E05F BLANKR 00087 00095*
   E1AF BS     00147 00290*
   A006 BSPOSH 00033*00306
   A007 BSPOSL 00034*00297
   E1E5 CALLER 00311 00313*
   E167 CARRY  00243 00245*
   E0F5 CARRYD 00179 00181*
   E153 CHANGE 00235 00237*
   A001 CHARH  00029*00057 00174 00177 00180
   A002 CHARL  00030*00058 00166
   A016 CHARLN 00042*00190 00240 00275 00313 00324 00417
   E08F CHARRC 00089 00138*
   E258 CLEAR  00162 00375*
   A00E COMPR  00036*00063 00064 00382
   E177 CR     00144 00257*
   E071 CRT    00108*00114
   3000 CRTCAD 00014*00015 00108 00164 00171 00225 00231 00238 00246
               00258 00264 00273 00279 00291 00296 00322 00328 00337
               00343 00353 00359 00362 00367 00376 00384 00401 00409
               00415 00423
   3001 CRTCRG 00015*00110 00165 00168 00172 00181 00226 00232 00241
               00249 00259 00261 00265 00274 00277 00280 00282 00292
               00294 00300 00302 00304 00323 00326 00329 00331 00340
               00344 00346 00354 00356 00360 00364 00368 00377 00385
```

FIGURE 24 — Complete Listing of CRTC Software
(Continued)



```
E06D CRTINI 00088 00106*
E2B8 CRYSET 00420 00422*
E0C8 CURSE  00141 00161 00163*
E09E DEC1   00143 00145*
E0A5 DEC2   00146 00148*
E0AC DEC3   00149 00151*
E0B3 DEC4   00152 00154*
E0BA DEC5   00155 00157*
E0C1 DEC6   00158 00160*
E097 DECODE 00140 00142*
E1CB DECR   00299 00302*
A014 ENDH   00040*00068 00223
A015 ENDL   00041*00069 00216 00229
E152 EQUAL1 00228 00234 00236*
E295 EQUAL2 00405 00407*
E2A3 EXIT   00413*
E12E FIND   00217 00220*
E12F FIND1  00219 00221*
E22A HOME   00159 00352*
E110 INCH   00138 00201*00203 00207
A00A INDEX  00035*00061 00062 00380 00386
E191 LF     00150 00272*
0040 MOVE   00020*00173 00305 00379
E1A9 NCARRY 00276 00282*
E0F2 NOCARY 00176 00180*
E1D2 NODECR 00301 00305*
E207 NOOCRY 00325 00331*
E10F NOSCRL 00192 00195*
E20C PAGE   00156 00336*
0004 PAGESZ 00021*00339 00345 00381
00FC PGMASK 00022*00355 00378
A000 RAM    00028*00029 00030 00031 00032 00035 00036 00037 00038
            00039 00040 00041 00042 00071 00072 00074 00076 00078
            00079 00080 00082 00084 00086 00188 00307 00387 00391
E05B RUN    00089*00090
E10C SCRLOL 00194*
0002 SCRNH  00023*00222
00AB SCRNL  00024*00215
47D0 SCRNND 00019*00099
4000 SCRNST 00018*00019 00096
E284 SCROLD 00316 00332 00399*
E120 SCROLU 00194 00214*00283
A011 SPACES 00037*00065 00189 00191 00193 00260 00266 00308 00309
            00315 00369
A012 STARTH 00038*00066 00221 00247 00250 00338 00341 00357 00399
            00424 00427
A013 STARTL 00039*00067 00214 00239 00242 00365 00407 00416 00419
E07F TABLE  00107 00118*
E1EF UPLINE 00153 00321*
E18E YES    00262 00266*
```

FIGURE 24 — Complete Listing of CRTC Software
(Continued)

MONITOR FOR THE MC146805G2L1 MICROCOMPUTER

Prepared by
David Bush
Microprocessor Product Engineer
and
Ed Rupp
Microprocessor System Design Engineer

INTRODUCTION

The MC146805G2 is a fully static single-chip CMOS Microcomputer. It has 112 bytes of RAM, 2106 bytes of user ROM, four 8-bit input/output ports, a timer, and an on-chip oscillator. The MC146805G2L1 ROM contains a monitor routine which provides the user with the ability to evaluate the MC146805G2 using a standard RS232 terminal. The user can enter short programs into the on-chip RAM and execute them via the monitor. A description of the monitor operation follows along with an assembled listing of the actual program.

MONITOR MODE

In this mode the MC146805G2L1 Microcomputer is connected to a terminal capable of running at 300, 1200, 4800, or 9600 baud. Figure 1 contains a schematic diagram of the monitor mode connections and a table showing C0 and C1 switch settings to obtain a baud rate that matches the terminal. Be sure the oscillator frequency is 3.579545 MHz. Any area of RAM from location $18 to $7A may be used for program storage; however, upper locations may be needed for user stack.

When the microcomputer is reset, a power-up message is printed. Following the message, the prompt character "." is printed and the monitor waits for a response. The response may consist of single letter commands with some commands requiring additional input. Unrecognized commands respond by printing "?". Valid commands are:

R — Display the Register
A — Display/Change the Accumulator
X — Display/Change the Index Register
M — Display/Change Memory
C — Continue Program Execution
E — Execute Program at Address
S — Display State of I/O and Timer

R — Display the Register

The processor registers are displayed as they appear on the stack. The format of the register print is:

HINZC AA XX PP

The first field shows the state of the condition code register bits. Each bit in the register has a single letter corresponding to the bit name. If the letter is present, the bit is 1. If a "." is printed in place of the letter, that bit is 0. For example, "H..ZC" means that the H, Z, and C bits are 1 and that the I and N bits are 0. The remainder of the line shows the status of the accumulator, index register, and program counter, respectively. The stack pointer is always at a fixed address (in this case $7A). The values shown are the values loaded into the CPU when a "C" or "E" command is executed. All register values except the condition code register can be changed with other commands. To change the condition code register, it is necessary to use the memory change command and modify location $7B.

A — Examine/Change the Accumulator

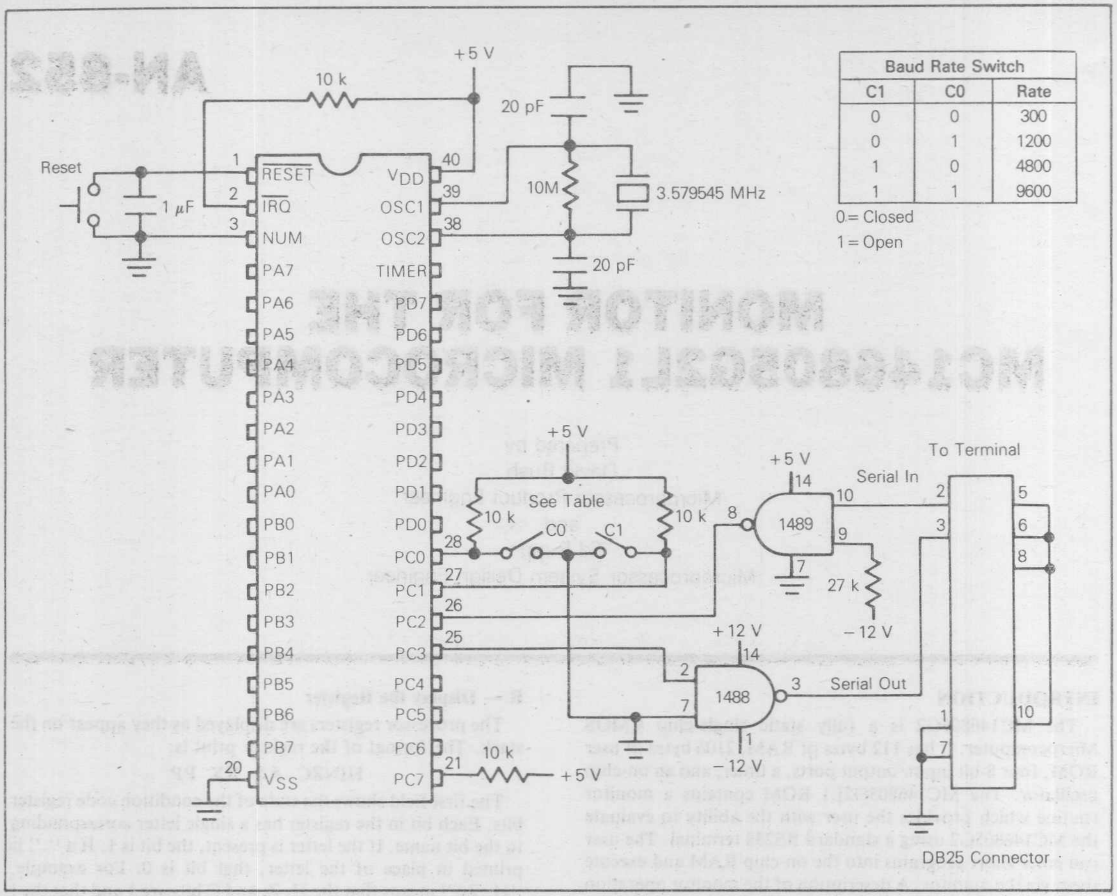
This command begins by printing the current value of the accumulator and then waits for more input. In order to change the current value, type in a new value (two hex digits). To leave the accumulator unchanged, type any non-hex digit (a space is a good choice).

X — Examine/Change the Index Register

This procedure is the same as the "A" command, but affects the index register instead.

M — Examine/Change Memory

Any memory location may be examined or changed with this command (except of course, ROM). To begin, type "M" followed by a hexadecimal address in the range $0000-$1FFF. The monitor responds by beginning a new line



Baud Rate Switch		
C1	C0	Rate
0	0	300
0	1	1200
1	0	4800
1	1	9600

0 = Closed
1 = Open

FIGURE 1. Monitor Mode Schematic Diagram

and printing the memory address followed by the current contents of that location. At this point you may type:

1. "." and re-examine the same byte. (Try this with location $0008.)
2. "^" and go to the previous byte. Typing "^" at location $0000 causes the monitor to go to $1FFF.
3. "CR" and go to the next byte. "CR" is the carriage return character. The byte after $1FFF is $0000.
4. "DD", where "DD" is a valid 2-digit hexadecimal number. The new data is stored at the current address and the monitor then goes to the next location. This means that to enter a program it is only necessary to go to the starting address of the program and start typing in the bytes. To see if the byte was really inputted, you can use the "^" character to return to the last byte typed in.
5. Finally, any character other than those described above causes the memory command to return to the prompt level of the monitor and prints ".".

C — Continue Program Execution

The "C" command merely executes an RTI instruction. This means that all the registers are reloaded exactly as they are shown in the register display. Execution continues until the reset switch is depressed or the processor executes an SWI. Upon executing an SWI, the monitor regains control and prints the prompt character. This feature can be used for an elementary form of breakpoints. Since there is really no way to know where the stack pointer is after an SWI, the monitor assumes that it is at $7A. This will not be the case if an SWI is part of a subroutine. In this case, the monitor will be re-entered but the stack pointer will point to $78. This is perfectly valid and typing "C" will pick up the program from where it left off. However, the A, X, R, and E commands all assume the stack starts at $7A and will not function properly. If the stack location is known, it is still possible to examine the registers by using the M command.

E — Start Execution at Address

The "E" command waits for a valid memory address

($0000-$1FFF) and places the address typed on the stack at locations $7E and $7F. The command then executes an RTI just like the "C" command. If the address typed is not a valid memory address, the command exits to the monitor without changing the current program counter value.

S — Display I/O States and Timer

The "S" command displays ports A, B, C, and D data along with the timer data and control register contents. The format of the display is:

A B C D TIM TCR

The data displayed is simply memory (RAM) locations $0000-$0003 with $0008 and $0009. Ports A, B, and D may be written to by first making them all outputs; i.e., for port A, change location $0004 (port A DDR) to $FF. Port C and the timer registers cannot be changed as they are used by the monitor.

MONITOR PROGRAM

A flowchart for the monitor mode program is provided in Figure 2. A listing for the ROM monitor program is attached to the end of this application note.

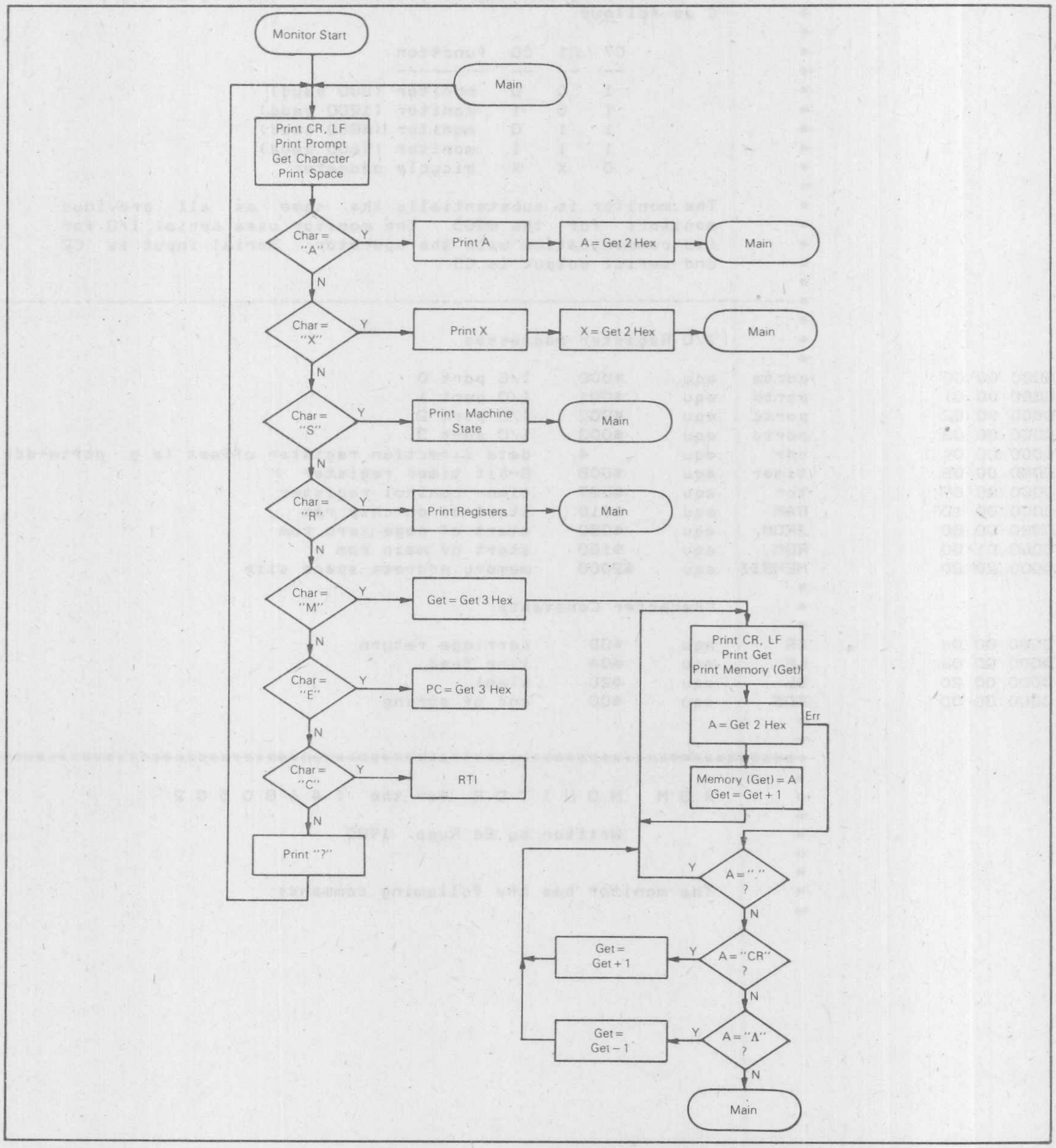


FIGURE 2. Monitor Mode Operating Flowchart

```
Sep  8 15:10 1981  146805G2 ROM Monitor Listing Page 1

                     *
                     *       M C 1 4 6 8 0 5 G 2   R O M   P A T T E R N
                     *
                     *       The MC6805G2 single-chip microcomputer is a  40-pin  CMOS
                     *       device  with  2096  bytes  of ROM, 112 bytes of RAM, four
                     *       8-bit I/O  ports,  a  timer  and  an  external  interrupt
                     *       input   The  ROM contains two separate programs.  Either
                     *       of these programs may be selected on reset by wiring port
                     *       C as follows:
                     *
                     *               C7  C1  CO  function
                     *               --  --  --  --------
                     *                1   0   0   monitor (300 baud)
                     *                1   0   1   monitor (1200 baud)
                     *                1   1   0   monitor (4800 baud)
                     *                1   1   1   monitor (9600 baud)
                     *                0   X   X   bicycle odometer
                     *
                     *       The monitor is substantially the  same  as  all  previous
                     *       monitors  for  the 6805.  The monitor uses serial I/O for
                     *       its communication with the operator.  Serial input is  C2
                     *       and serial output is C3.
                     *
                     *----------------------------------------------------------------------
                     *
                     *       I/O Register Addresses
                     *
0000 00 00           porta   equ     $000    I/O port 0
0000 00 01           portb   equ     $001    I/O port 1
0000 00 02           portc   equ     $002    I/O port 2
0000 00 03           portd   equ     $003    I/O port 3
0000 00 04           ddr     equ        4    data direction register offset (e.g. porta+ddr)
0000 00 08           timer   equ     $008    8-bit timer register
0000 00 09           tcr     equ     $009    timer control register
0000 00 10           RAM     equ     $010    start of on-chip ram
0000 00 80           ZROM    equ     $080    start of page zero rom
0000 01 00           ROM     equ     $100    start of main rom
0000 20 00           MEMSIZ  equ    $2000    memory address space size
                     *
                     *       Character Constants
                     *
0000 00 0d           CR      equ     $0D     carriage return
0000 00 0a           LF      equ     $0A     line feed
0000 00 20           BL      equ     $20     blank
0000 00 00           EOS     equ     $00     end of string
                     *
                     *
                     **********************************************************************
                     *
                     *       R O M   M O N I T O R  for the  1 4 6 8 0 5 G 2
                     *
                     *               Written by Ed Rupp, 1980
                     *
                     *
                     *
                     *       The monitor has the following commands:
                     *
```

```
                    *        R --  Print registers.
                    *              format is CCCCC AA XX PPP
                    *
                    *        A --  Print/change A accumulator.
                    *              Prints the register value, then
                    *              waits for new value.  Type
                    *              any non-hex character to exit.
                    *
                    *        X --  Print/change X accumulator.
                    *              Works the same as 'A',. except modifies X instead.
                    *
                    *        M --  Memory examine/change.
                    *              Type M AAA to begin,
                    *                then type:  .  -- to re-examine current
                    *                            ^  -- to examine previous
                    *                            CR -- to examine next
                    *                            DD -- new data
                    *                Anything else exits memory command.
                    *
                    *        C --  Continue program.  Execution starts at
                    *                the location specified in the program
                    *                counter, and
                    *                continues until an  swi is executed
                    *                or until reset.
                    *
                    *        E --  Execute from address.  Format is
                    *                E AAAA.  AAAA is any valid memory address.
                    *
                    *        S --  Display Machine State.  All important registers are
                    *                displayed.
                    *
                    *
                    *        Special Equates
                    *
0602 00 2e          PROMPT   equ     '.       prompt character
0602 00 0d          FWD      equ     CR       go to next byte
0602 00 5e          BACK     equ     '^       go to previous byte
0602 00 2e          SAME     equ     '.       re-examine same byte
                    *
                    *        Other
                    *
0602 00 7f          initsp   equ     $7F      initial stack pointer value
0602 00 7a          stack    equ     initsp-5 top of stack
                    *
                    *        ram variables
                    *
0602 00 10          get      equ     RAM+0    4-byte no-mans land, see pick and drop subroutines
0602 00 14          atemp    equ     RAM+4    acca temp for getc,putc
0602 00 15          xtemp    equ     RAM+5    x reg. temp for getc,putc
0602 00 16          char     equ     RAM+6    current input/output character
0602 00 17          count    equ     RAM+7    number of bits left to get/send
                    *
                    *        state --- print machine state
                    *
                    *         A  B  C  D TIM TCR
                    *        dd dd dd dd  dd  dd
```

```
                        *
                        *       header string for I/O register display
                        *
0602 0d 0a              iomsg   fcb     CR,LF
0604 20 41 20 20 42 20          fcc     / A  B  C  D TIM TCR/
     20 43 20 20 44 20
     54 49 4d 20 54 43
     52
0617 0d 0a 00                   fcb     CR,LF,EOS
                        *
061a 5f                 state   clrx
061b d6 06 02           state2  lda     iomsg,x get next char
061e a1 00                      cmp     #EOS    quit?
0620 27 06                      beq     state3  yes, now print values
0622 cd 08 01                   jsr     putc    no, print char
0625 5c                         incx            bump pointer
0626 20 f3                      bra     state2  do it again
0628                    state3
                        *
                        *       now print values underneath the header
                        *
0628 5f                         clrx
0629 f6                 pio     lda     ,x      start with I/O ports
062a cd 07 5e                   jsr     putbyt
062d cd 07 8b                   jsr     puts
0630 5c                         incx
0631 a3 04                      cpx     #4      end of I/O?
0633 26 f4                      bne     pio     no, do more
                        *
0635 cd 07 8b                   jsr     puts
0638 b6 08                      lda     timer   now print the value in the timer
063a cd 07 5e                   jsr     putbyt
063d cd 07 8b                   jsr     puts
0640 cd 07 8b                   jsr     puts
0643 b6 09                      lda     tcr     the control register too
0645 cd 07 5e                   jsr     putbyt
0648 20 48                      bra     monit   all done
                        *
                        *       pcc --- print condition codes
                        *
                        *       string for pcc subroutine
                        *
064a 48 49 4e 5a 43     ccstr   fcc     /HINZC/
                        *
064f b6 7b              pcc     lda     stack+1 condition codes in acca
0651 48                         asla            move h bit to bit 7
0652 48                         asla
0653 48                         asla
0654 b7 10                      sta     get     save it
0656 5f                         clrx
0657 a6 2e              pcc2    lda     #'.
0659 38 10                      asl     get     put bit in c
065b 24 03                      bcc     pcc3    bit off means print .
065d d6 06 4a                   lda     ccstr,x pickup appropriate character
0660 cd 08 01           pcc3    jsr     putc    print . or character
0663 5c                         incx            point to next in string
```

```
0664 a3 05                      cpx     #5      quit after printing all 5 bits
0666 25 ef                      blo     pcc2
0668 81                         rts
                        *
                        *       seta --- examine/change accumulator A
                        *
0669 ae 7c              seta    ldx     #stack+2 point to A
066b 20 02                      bra     setany
                        *
                        *       setx --- examine/change accumulator X
                        *
066d ae 7d              setx    ldx     #stack+3 point to X
                        *
                        *       setany --- print (x) and change if necessary
                        *
066f f6                 setany  lda     ,x      pick up the data, and
0670 cd 07 5e                   jsr     putbyt  print it
0673 cd 07 8b                   jsr     puts
0676 cd 07 94                   jsr     getbyt  see if it should be changed
0679 25 17                      bcs     monit    error, no change
067b f7                         sta     ,x      else replace with new value
067c 20 14                      bra     monit    now return
                        *
                        *       regs --- print cpu registers
                        *
067e ad cf              regs    bsr     pcc     print cc register
0680 cd 07 8b                   jsr     puts    separate from next stuff
0683 3f 11                      clr     get+1   point to page zero,
0685 a6 7c                      lda     #stack+2
0687 b7 12                      sta     get+2
0689 cd 07 4b                   jsr     out2hs  continue print with A
068c cd 07 4b                   jsr     out2hs  X and finally the
068f cd 07 43                   jsr     out4hs  Program Counter
                        *
                        *       fall into main loop
                        *
                        *       monit --- print prompt and decode commands
                        *
0692 cd 07 7d           monit   jsr     crlf     go to next line
0695 a6 2e                      lda     #PROMPT
0697 cd 08 01                   jsr     putc    print the prompt
069a cd 07 c3                   jsr     getc    get the command character
069d a4 7f                      and     #%1111111 mask parity
069f cd 07 8b                   jsr     puts    print space (won't destroy A)
06a2 a1 41                      cmp     #'A     change A
06a4 27 c3                      beq     seta
06a6 a1 58                      cmp     #'X     change X
06a8 27 c3                      beq     setx
06aa a1 52                      cmp     #'R     registers
06ac 27 d0                      beq     regs
06ae a1 45                      cmp     #'E     execute
06b0 27 16                      beq     exec
06b2 a1 43                      cmp     #'C     continue
06b4 27 21                      beq     cont
06b6 a1 4d                      cmp     #'M     memory
06b8 27 1e                      beq     memory
```

```
06ba a1 53                  cmp      #'S      display machine state
06bc 26 03                  bne      monit2
06be cc 06 1a               jmp      state    commands are getting too far away
                   *
06c1 06 c1         monit2   equ      *
06c1 a6 3f                  lda      #'?      none of the above
06c3 cd 08 01               jsr      putc
06c6 20 ca                  bra      monit     loop around
                   *
                   *        exec --- execute from given address
                   *
06c8 cd 07 94      exec     jsr      getbyt   get high nybble
06cb 25 c5                  bcs      monit     bad digit
06cd 97                     tax               save for a second
06ce cd 07 94               jsr      getbyt   now the low byte
06d1 25 bf                  bcs      monit     bad address
06d3 b7 7f                  sta      stack+5  program counter low
06d5 bf 7e                  stx      stack+4  program counter high
                   *
                   *        cont --- continue users program
                   *
06d7 80            cont     rti               simple enough
                   *
                   *       memory --- memory examine/change
                   *
06d8 cd 07 94      memory   jsr      getbyt   build address
06db 25 b5                  bcs      monit     bad hex character
06dd b7 11                  sta      get+1
06df cd 07 94               jsr      getbyt
06e2 25 ae                  bcs      monit     bad hex character
06e4 b7 12                  sta      get+2    address is now in get+1&2
06e6 cd 07 7d      mem2     jsr      crlf     begin new line
06e9 b6 11                  lda      get+1    print current location
06eb a4 1f                  and      #$1F     mask upper 3 bits (8K map)
06ed cd 07 5e               jsr      putbyt
06f0 b6 12                  lda      get+2
06f2 cd 07 5e               jsr      putbyt
06f5 cd 07 8b               jsr      puts     a blank, then
06f8 ad 2c                  bsr      pick     get that byte
06fa cd 07 5e               jsr      putbyt   and print it
06fd cd 07 8b               jsr      puts     another blank,
0700 cd 07 94               jsr      getbyt   try to get a byte
0703 25 06                  bcs      mem3     might be a special character
0705 ad 25                  bsr      drop     otherwise, put it and continue
0707 ad 33         mem4     bsr      bump     go to next address
0709 20 db                  bra      mem2     and repeat
070b a1 2e         mem3     cmp      #SAME    re-examine same?
070d 27 d7                  beq      mem2     yes, return without bumping
070f a1 0d                  cmp      #FWD     go to next?
0711 27 f4                  beq      mem4     yes, bump then loop
0713 a1 5e                  cmp      #BACK    go back one byte?
0715 26 0c                  bne      xmonit   no, exit memory command
0717 3a 12                  dec      get+2    decrement low byte
0719 b6 12                  lda      get+2    check for underflow
071b a1 ff                  cmp      #$FF
071d 26 c7                  bne      mem2     no underflow
```

```
071f 3a 11                       dec     get+1
0721 20 c3                       bra     mem2
                          *
                          *      convenient transfer point back to monit
                          *
0723 cc 06 92             xmonit   jmp      monit      return to monit
                          *
                          *     utilities
                          *
                          *     pick --- get byte from anywhere in memory
                          *              this is a horrible routine (not merely
                          *              self-modifying, but self-creating)
                          *
                          *     get+1&2 point to address to read,
                          *     byte is returned in A
                          *     X is unchanged at exit
                          *
0726 bf 15                pick   stx     xtemp    save X
0728 ae d6                       ldx     #$D6     D6=lda 2-byte indexed
072a 20 04                       bra     common
                          *
                          *
                          *      drop --- put byte to any memory location.
                          *               has the same undesirable properties
                          *               as pick
                          *      A has byte to store, and get+1&2 points
                          *      to location to store
                          *      A and X unchanged at exit
                          *
072c bf 15                drop   stx     xtemp    save X
072e ae d7                       ldx     #$D7     d7=sta 2-byte indexed
                          *
                          *
0730 bf 10                common stx     get      put opcode in place
0732 ae 81                       ldx     #$81     81=rts
0734 bf 13                       stx     get+3    now the return
0736 5f                          clrx             we want zero offset
0737 bd 10                       jsr     get      execute this mess
0739 be 15                       ldx     xtemp    restore X
073b 81                          rts              and exit
                          *
                          *      bump --- add one to current memory pointer
                          *
                          *      A and X unchanged
                          *
073c 3c 12                bump   inc     get+2    increment low byte
073e 26 02                       bne     bump2    non-zero means  no carry
0740 3c 11                       inc     get+1    increment high nybble
0742 81                   bump2  rts
                          *
                          *
                          *      out4hs --- print word pointed to as an address, bump pointer
                          *                 X is unchanged at exit
                          *
0743 ad e1                out4hs bsr     pick     get high nybble
0745 a4 1f                       and     #$1F     mask high bits
```

```
0747 ad 15               bsr     putbyt    and print it
0749 ad f1               bsr     bump      go to next address
                    *
                    *    out2hs --- print byte pointed to, then a space. bump pointer
                    *         X is unchanged at exit
                    *
074b ad d9          out2hs  bsr     pick      get the byte
074d b7 10               sta     get       save A
074f 44                  lsra
0750 44                  lsra
0751 44                  lsra
0752 44                  lsra                shift high to low
0753 ad 16               bsr     putnyb
0755 b6 10               lda     get
0757 ad 12               bsr     putnyb
0759 ad e1               bsr     bump      go to next
075b ad 2e               bsr     puts      finish up with a blank
075d 81                  rts
                    *
                    *    putbyt --- print A in hex
                    *         A and X unchanged
                    *
075e b7 10          putbyt  sta     get       save A
0760 44                  lsra
0761 44                  lsra
0762 44                  lsra
0763 44                  lsra                shift high nybble down
0764 ad 05               bsr     putnyb    print it
0766 b6 10               lda     get
0768 ad 01               bsr     putnyb    print low nybble
076a 81                  rts
                    *
                    *    putnyb --- print lower nybble of A in hex
                    *         A and X unchanged, high nybble
                    *         of A is ignored.
                    *
076b b7 13          putnyb  sta     get+3     save A in yet another temp
076d a4 0f               and     #$F       mask off high nybble
076f ab 30               add     #'0       add ascii zero
0771 a1 39               cmp     #'9       check for A-F
0773 23 02               bls     putny2
0775 ab 07               add     #'A-'9-1 adjustment for hex A-F
0777 cd 08 01       putny2  jsr     putc
077a b6 13               lda     get+3     restore A
077c 81                  rts
                    *
                    *     crlf --- print carriage return, line feed
                    *          A and X unchanged
                    *
077d b7 10          crlf    sta     get       save
077f a6 0d               lda     #CR
0781 cd 08 01            jsr     putc
0784 a6 0a               lda     #LF
0786 ad 79               bsr     putc
0788 b6 10               lda     get       restore
078a 81                  rts
```

```
                        *       puts --- print a blank (space)
                        *                A and X unchanged
                        *
078b b7 10              puts    sta     get     save
078d a6 20                      lda     #BL
078f ad 70                      bsr     putc
0791 b6 10                      lda     get     restore
0793 81                         rts
                        *
                        *       getbyt --- get a hex byte from terminal
                        *
                        *       A gets the byte typed if it  was  a  valid  hex  number,
                        *       otherwise  A gets the last character typed.  The c-bit is
                        *       set  on  non-hex  characters;   cleared   otherwise.   X
                        *       unchanged in any case.
                        *
0794 ad 0f              getbyt  bsr     getnyb  build byte from 2 nybbles
0796 25 0c                      bcs     nobyt   bad character in input
0798 48                         asla
0799 48                         asla
079a 48                         asla
079b 48                         asla            shift nybble to high nybble
079c b7 10                      sta     get     save it
079e ad 05                      bsr     getnyb  get low nybble now
07a0 25 02                      bcs     nobyt   bad character
07a2 bb 10                      add     get     c-bit cleared
07a4 81                 nobyt   rts
                        *
                        *       getnyb --- get hex nybble from terminal
                        *
                        *       A gets the nybble typed if  it  was  in  the  range  0-F,
                        *       otherwise  A  gets the character typed.  The c-bit is set
                        *       on  non-hex  characters;   cleared   otherwise.   X   is
                        *       unchanged.
                        *
07a5 ad 1c              getnyb  bsr     getc    get the character
07a7 a4 7f                      and     #%1111111 mask parity
07a9 b7 13                      sta     get+3   save it just in case
07ab a0 30                      sub     #'0     subtract ascii zero
07ad 2b 10                      bmi     nothex  was less than '0'
07af a1 09                      cmp     #9
07b1 23 0a                      bls     gotit
07b3 a0 07                      sub     #'A-'9-1 funny adjustment
07b5 a1 0f                      cmp     #$F     too big?
07b7 22 06                      bhi     nothex  was greater than 'F'
07b9 a1 09                      cmp     #9         check between 9 and A
07bb 23 02                      bls     nothex
07bd 98                 gotit   clc             c=0 means good hex char
07be 81                         rts
07bf b6 13              nothex  lda     get+3   get saved character
07c1 99                         sec
07c2 81                         rts             return with error
                        *
                        *       S e r i a l   I / O   R o u t i n e s
                        *
                        *       These subroutines are modifications of the original NMOS
                        *       version.  Differences are due to the variation in cycle
                        *       time of CMOS instructions vs. NMOS.
                        *
                        *       Since the INT and TIMER interrupt vectors are used in the
                        *       bicycle odometer, the I-bit should  always  be  set  when
                        *       running  the  monitor.  Hence, the code that fiddles with
                        *       the I-bit has been eliminated..
                        *
                        *
                        *       Definition of serial I/O lines
                        *
                        *       Note: changing `in' or `out' will necessitate changing the
                        *       way `put' is setup during reset.
                        *
07c3 00 02              put     equ     portc   serial I/O port
07c3 00 02              in      equ     2       serial input line#
07c3 00 03              out     equ     3       serial output line#
                        *
                        *       getc --- get a character from the terminal
                        *
                        *       A gets the character typed, X is unchanged.
                        *
07c3 bf 15              getc    stx     xtemp   save X
07c5 a6 08                      lda     #8      number of bits to read
07c7 b7 17                      sta     count
07c9 04 02 fd           getc4   brset   in,put,getc4 wait for hilo transition
                        *
                        *       delay 1/2 bit time
                        *
07cc b6 02                      lda     put
07ce a4 03                      and     #%11    get current baud rate
07d0 97                         tax
07d1 de 08 4b                   ldx     delays,x get loop constant
07d4 a6 04              getc3   lda     #4
07d6 9d                 getc2   nop
07d7 4a                         deca
07d8 26 fc                      bne     getc2
07da 5d                         tstx            loop padding
07db 14 02                      bset    in,put  ditto
07dd 14 02                      bset    in,put  CMOS ditto
07df 5a                         decx
07e0 26 f2                      bne     getc3   major loop test
                        *
                        *       now we should be in the middle of the start bit
                        *
07e2 04 02 e4                   brset   in,put,getc4 false start bit test
07e5 7d                         tst     ,x      more timing delays
07e6 7d                         tst     ,x
07e7 7d                         tst     ,x
                        *
                        *       main loop for getc
                        *
07e8 ad 46              getc7   bsr     delay   (6) common delay routine
07ea 05 02 00                   brclr   in,put,getc6 (5)  test input and set c-bit
07ed 7d                 getc6   tst     ,x       (4) timing equalizer
```

```
07ee 9d                         nop             (2) CMOS equalization
07ef 9d                         nop             (2) CMOS equalization
07f0 9d                         nop             (2) CMOS equalization
07f1 9d                         nop             (2) CMOS equalization
07f2 9d                         nop             (2) CMOS equalization
07f3 9d                         nop             (2) CMOS equalization
07f4 36 16                      ror     char    (5) add this bit to the byte
07f6 3a 17                      dec     count   (5)
07f8 26 ee                      bne     getc7   (3) still more bits to get(see?)
                        *
07fa ad 34                      bsr     delay   wait out the 9th bit
07fc b6 16                      lda     char    get assembled byte
07fe be 15                      ldx     xtemp   restore x
                        *
0800 81                         rts             and return
                        *
                        *       putc --- print a on the terminal
                        *
                        *       X and A unchanged
                        *
0801 b7 16              putc    sta     char
0803 b7 14                      sta     atemp   save it in both places
0805 bf 15                      stx     xtemp   don't forget about X
0807 a6 09                      lda     #9      going to put out
0809 b7 17                      sta     count   9 bits this time
080b 5f                         clrx            for very obscure reasons
080c 98                         clc             this is the start bit
080d 20 02                      bra     putc2   jump in the middle of things
                        *
                        *       main loop for putc
                        *
080f 36 16              putc5   ror     char    (5) get next bit from memory
0811 24 04              putc2   bcc     putc3   (3) now set or clear port bit
0813 16 02                      bset    out,put
0815 20 04                      bra     putc4
0817 17 02              putc3   bclr    out,put (5)
0819 20 00                      bra     putc4   (3) equalize timing again
081b dd 08 30           putc4   jsr     delay,x (7) must be 2-byte indexed jsr
                        *                           this is why X must be zero
081e 43                         coma            (3) CMOS equalization
081f 43                         coma            (3) CMOS equalization
0820 43                         coma            (3) CMOS equalization
0821 3a 17                      dec     count   (5)
0823 26 ea                      bne     putc5   (3) still more bits
                        *
0825 14 02                      bset    in,put   7 cycle delay
0827 16 02                      bset    out,put send stop bit
                        *
0829 ad 05                      bsr     delay   delay for the stop bit
082b be 15                      ldx     xtemp   restore X and
082d b6 14                      lda     atemp   of course A
082f 81                         rts
                        *
                        *       delay --- precise delay for getc/putc
                        *
0830 b6 02              delay   lda     put     first, find out
```

```
0832 a4 03                  and     #%11      what the baud rate is
0834 97                     tax
0835 de 08 4b               ldx     delays,x  loop constant from table
0838 a6 f8                  lda     #$F8      funny adjustment for subroutine overhead
083a ab 09          del3    add     #$09
083c                del2
083c 9d                     nop               CMOS equalization
083d 4a                     deca
083e 26 fc                  bne     del2
0840 5d                     tstx              loop padding
0841 14 02                  bset    in,put    ditto
0843 14 02                  bset    in,put    CMOS ditto
0845 5a                     decx
0846 26 f2                  bne     del3      main loop
0848 9d                     nop               CMOS equalization
0849 9d                     nop               CMOS equalization
084a 81                     rts               with X still equal to zero
                    *
                    *       delays for baud rate calculation
                    *
                    *       This table must not be put on page zero since
                    *       the accessing must take 6 cycles.
                    *
084b 20             delays  fcb     32          300 baud
084c 08                     fcb     8          1200 baud
084d 02                     fcb     2          4800 baud
084e 01                     fcb     1          9600 baud
                    *
                    *      reset --- power on reset routine
                    *
                    *       Based on a port bit, run the bicycle odometer or the monitor.
                    *
084f                reset
084f 0e 02 03               brset   7,portc,other
0852 cc 01 54               jmp     odo       be a bicycle odometer
                    *
                    *       run the monitor
                    *
0855                other
0855 a6 08                  lda     #%1000    setup port for serial io
0857 b7 02                  sta     put       set output to mark level
0859 b7 06                  sta     put+ddr set ddr to have one output
                    *
                    *       print sign-on message
                    *
085b 5f                     clrx
085c d6 08 6c       babble  lda     msg,x     get next character
085f a1 00                  cmp     #EOS      last char?
0861 27 06                  beq     mstart    yes, start monitor
0863 cd 08 01               jsr     putc      and print it
0866 5c                     incx              advance to next char
0867 20 f3                  bra     babble    more message
0869                mstart
0869 83                     swi               push machine state and go to monitor routine
086a 20 e3                  bra       reset   loop around
                    *
```

```
                        *       msg --- power up message
                        *
086c 0d 0a              msg     fcb     CR,LF
086e 31 34 36 38 30 35          fcc     /146805G2/
     47 32
0876 00                         fcb     EOS
                        *
                        *
                        ******************************************************************
                        *
                        *       interrupt vectors
                        *
1ff6                            org     MEMSIZ-10 start of vectors
                        *
1ff6 01 e0                      fdb     onemil  exit wait state     \
1ff8 01 e0                      fdb     onemil  timer interrupt      |- odometer vectors
1ffa 02 46                      fdb     wheel   external interrupt  /
1ffc 06 92                      fdb     monit   swi to main entry point
1ffe 08 4f                      fdb     reset   power on vector
```