Rev. 2, July 1981

Microprocessor/Memory Applications Briefs

A compendium of the most useful CDP1800 Series microprocessor/memory application hints that have appeared in recent issues of the RCA Solid State Newsletter.

RCA Solid State | Brussels ● Buenos Aires ● Hamburg ● Madrid ● Mexico City ● Milan Montreal ● Paris ● Sao Paulo ● Somerville NJ ● Stockholm Sunbury-on-Thames ● Taipei ● Tokyo

INTRODUCTION

This manual is an outgrowth of application-related activities at RCA as they pertain to the needs of our 1800 Series customer base. The inputs for the articles have come directly from customers, indirectly through our field sales and applications force, and as a spinoff of new product definition work. Because of this, these briefs should answer application-type questions that may not be sufficiently explained in our current product literature. New material will periodically update this publication, and, as subject material expands, articles about specific device types will be combined into general Application Notes and appear in future Data Book releases.

We welcome your comments and suggestions for new mini-notes, based on your experiences in developing hardware and software around the 1800 series. We would appreciate it if you could pass along to your local RCA Representative any material that is not of a proprietary nature - we will consider developing any ideas that meet the general needs of our customer base.

TABLE OF CONTENTS

Table of Contents (cont'd)

Page Table of Contents (cont'd)

OSCILLATOR DESIGN CONSIDERATIONS FOR THE CDP1802

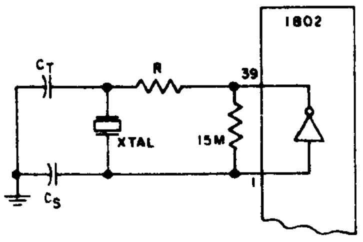
Despite the widespread use of crystal-controlled oscillators for microprocessors, crystal selection may still pose problems for many designers. Most of these problems can be minimized by an understanding of the various properties and specifications needed to define an oscillator circuit for the CDP1802 microprocessor.

Clock Frequency and Accuracy

Quartz crystal oscillators will provide frequency stability better than 0.01%. However, many microprocessor applications do not require an exact clock frequency -- therefore, the use of RC or LC type oscillators may be a wise cost-effective choice. If a crystal is to be used, there are two basic low-cost types to consider: Parallel resonant AT cut quartz crystals typically ranging from 0.8 MHz to 6 MHz, and low frequency tuning fork type crystals available in several stock frequencies from 10 kHz to 240 kHz, including the popular 32.768 kHz digital watch frequency. Statek Corp., for instance, specializes in low-cost tuning fork crystals, and they have numerous free application notes to aid in design.

The Oscillator Circuit

Figure 1 shows the basic cyrstal oscillator circuit for the CDP1802. The 15 megohm resistor is used to bias the gate in its linear region so that it behaves like an amplifier. Capacitors C_S and C_T provide the required



92CS-32713

Figure 1

capacitive loading for the crystal and act as high-frequency filters to avoid overtone oscillations. The values of C_S and C_T can be calculated using the following equations found in ICAN-6086:

$$C_T = \frac{4\ C_L}{1 - 5f\ R_e C_L}, \text{ and}$$

$$C_S = \frac{4\ C_L}{3 + 5f\ R_e C_L}$$

R_e is the equivalent resistance of the crystal. Figure 2 shows the approximate relation of R_e to frequency for AT type crystals.

C_L is the load capacitance for the crystal, generally a standard value set by the crystal vendor between 10 and 32 pF.

f is the frequency in Hz.

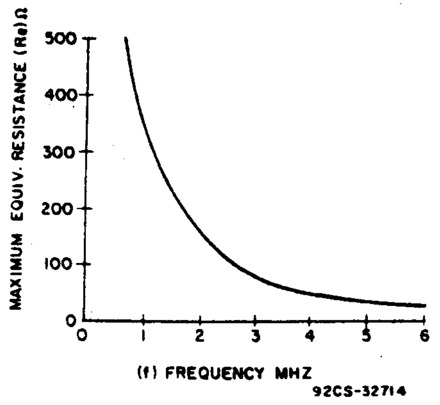


Figure 2.

The actual value C_S used should be about 4 pF less than the calculated value to allow for the amplifier input capacitance.

R can usually be 0 ohms unless low power drain or stability during variable V_{DD} voltage is important. ICAN-6086 and ICAN-6539 explain in detail the need for and calculation of R.

Crystal Specifications

Frequency and Tolerance -- ±0.01% from -20 to 100°C is common. Tighter tolerances down to ±0.001% are available at additional cost (check with vendor).

Mode of Oscillation -- Fundamental, parallel resonance.

Load Capacitance (C_L) -- Typically between 10 pF and 32 pF - choose a stock value the vendor has. Higher values of C_L will improve frequency stability, but lower values will decrease oscillation power consumption.

Maximum Equivalent Resistance (R_e) -- This number is related to the frequency. Most vendors will supply crystals with R_e values lower than the curve shown in Fig. 2.

Max. Drive Level -- The crystal should be able to dissipate 5 milliwatts of power. If the crystal cannot handle this level, frequency drift or even damage to the crystal may result. Crystals with lower drive capability, such as the tuning fork type, can be used if R is increased to reduce the drive level. ICAN-6086 gives details on how to compute R, also Statek has numerous free Application Notes about this.

Can Type -- HC33 and HC18 are popular types.

CAUTION

Don't over spec. Since virtually every parameter is a derivative of another, it is easy to specify a crystal that cannot be manufactured. It's best to work with the vendor by explaining the application.

Partial List of Crystal Vendors

International Crystal Mfg.
10 N. Lee Street
Oklahoma City, OK 73102

405-236-3741

Valtec Corp.
75 South Street
Hopkinton, Mass. 01748

617-435-6831

Statek Corp.
1200 Alvarez Ave.
Orange, CA 92668

714-639-7810

M-Tron Ind.
P.O. Box 630
Yankton, S.D. 57078

605-665-9321

Partial List of Ceramic Resonator Vendors

Vernitron Piezoelectric Div.
232 Forbes Road
Bedford, Ohio 44146

216-232-8600

Radio Materials Corp.
4242 W. Bryn Mawr Ave.
Chicago, Ill. 60646

312-478-3600

Murata Mfg. Co. Ltd.
1148 Franklin Rd. S.E.
Marietta, GA. 30067

404-952-9777

For additional information contact Jerry Johnson - X6776.

Alternative Oscillator Types

RC Type -- A simple RC oscillator is shown in Fig. 3 using a CD4093. The approximate frequency (f) is determined as follows:

$$f = \frac{1.25}{RC}, \text{ at } V_{DD} = 5V$$

Unlike the CDP1802, the CDP1804 microprocessor uses a Schmitt inverter for the oscillator amplifier so that it can be used directly for the RC oscillator.

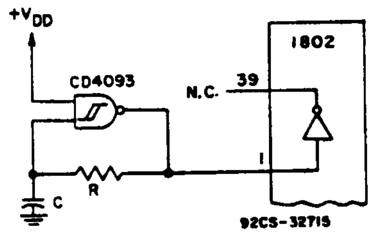


Figure 3

LC Type -- A parallel-resonant LC circuit may be used as the frequency determining network for the CDP1802. (See Fig. 4.) The frequency and component values have the following relationship:

$$f = \frac{1}{2\pi \sqrt{LC}}$$

The high-impedance secondary of a small 455 KC IF transformer similar to those found in most portable transistor radios makes an excellent LC oscillator (the C is built in). Furthermore, it is tunable (about ±10%) using the slug.

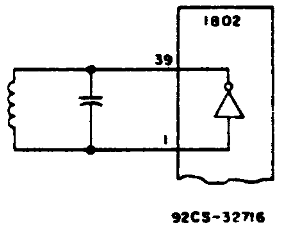


Figure 4

Ceramic Resonators

Made of piezo-electric material, ceramic resonators behave similar to crystals, requiring two capacitors and a bias resistor. Frequency tolerances of ±1% are typical minimums. Values of C_1, C_2, and R depend on the device used; check vendor literature.

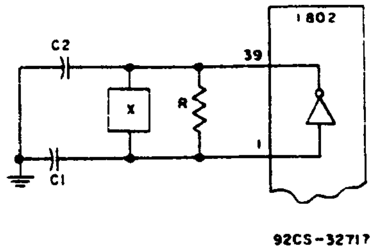


Figure 5

Low Voltage Operation of the 1802

Tests indicate that a typical 1802 can operate with V_{CC} and V_{DD} as low as 2.5V, provided clock frequencies are limited to the values shown in Fig. 1 and temperature is held at 25°C. Guaranteed operation, of course, requires a custom selection.

Twenty 1802's having various date codes and package types, were tested at 25°C in a switch box without memory components. An external square wave generator was used for the clock signal, thereby eliminating crystal oscillator limitations. Details about low voltage crystal oscillator design can be found in ICAN-6539 and ICAN-6086.

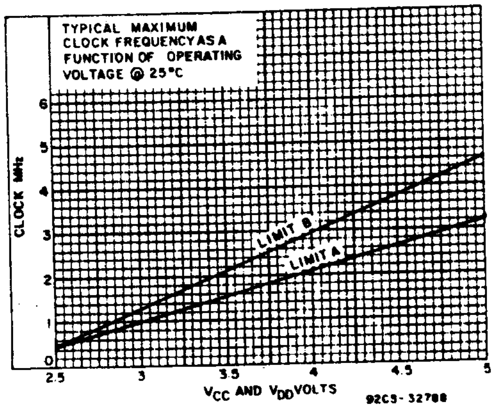


Figure 1

Two instructions, SEX to F, and IRX, were loaded into the CPU by switching in the HEX codes and single stepping the clock. When running, the CPU generates addresses from 0000 to FFFF and then repeats over and over. The frequency was increased until the high address byte to TPA shift became critical. The results, shown as "limit A", indicate the frequency at which the leading edge of the high byte occurs at the trailing edge of TPA (Zero Setup Time). The November '79 Newsletter describes this phenomenon in detail.

If TPA is delayed externally to provide more set up time for the high byte, or if the high byte is not required, then the maximum frequency is merely the internal logic speed limitation of the CPU shown as "limit B".

For more information or specific applications contact Jerry Johnson, X6776.

Data Bus Contention During CDP1802 Register-to-Register Operations

In 1802 based systems using various ROM's (CDP1832, 1834) or EPROM's (2708, 2758, 2716) bus contention problems have been found to occur during internal data transfer operations (GHI, PHI, GLO, PLO). As a result, data is lost in one or more register.

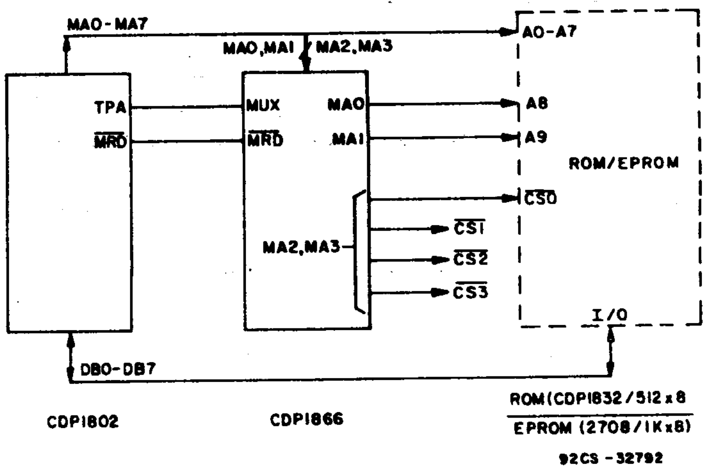
The 1802 generates a valid 16-bit address and a TPA signal during these operations. If the chip-select signals for the ROM or EPROM is only controlled by higher order address bits, then it is very probable that these memories can be selected and have their output drivers "turned on", creating a bus contention problem with the 1802 data bus drivers.

The solution to this problem is to either gate the chip-select functions with $\overline{\text{MRD}}$ externally, or find a spare input on the memories for $\overline{\text{MRD}}$. During these register operations $\overline{\text{MRD}}$ is held high. (See Table 1 on p. 90 of the MPM-201B Manual). See below for specific suggestions.

A. <u>Wiring $\overline{\text{MRD}}$ to a Spare Input</u>

- CDP1834 (CS1 or CS2) - only if they are "active low"
- 2716, 2732, 2758 ($\overline{\text{OE}}$)

B. <u>Gating $\overline{\text{MRD}}$ Externally with the Chip-Select Function</u>



For further information, contact Joe Paradise, X7352.

1802 Interrupt Control Circuits

The circuits shown in Figures 1 and 2 generate a separate vectored starting address, allocating a 32 byte block of memory, for each of 8 prioritized interrupt inputs:

Circuit #1 - operates on a first come-first served basis, with priority arbitration for coincident input pulses. The first input pulse is asynchronously latched. Once an interrupt is initiated all inputs are ignored until the vectoring address is read by the CPU; at this time a new input may be accepted but will not be serviced until the conclusion of the existing interrupt routine. If DMA is to be used, SC0 and SC1 must be AND-ed together to distinguish S2 from S3 states.

Circuit #2 - the highest priority peripheral will be serviced. Inputs are not self latching and therefore must be maintained until the CPU begins the INP instruction to read Port B. The peripheral device must remove its interrupt request before the end of the interrupt routine; peripherals could use a daisy chained interrupt acknowledge scheme.

Both circuits assume no subroutine nesting during interrupt.

Input.	0	1	2	3	4	5	6	7
Vector Address Low Byte	00	20	40	60	80	A0	C0	E0

For additional information contact Jerry Johnson - X6776

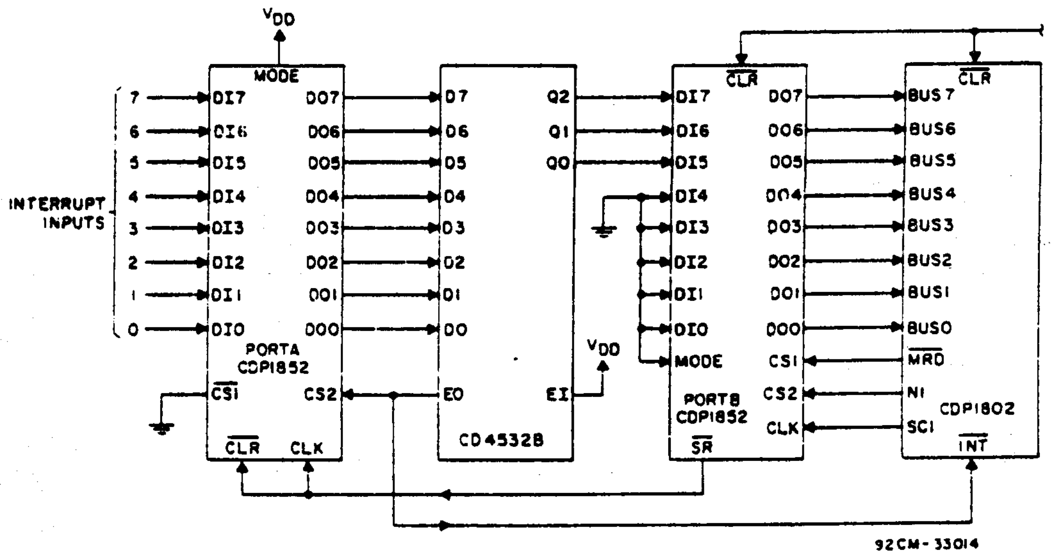


Figure 1

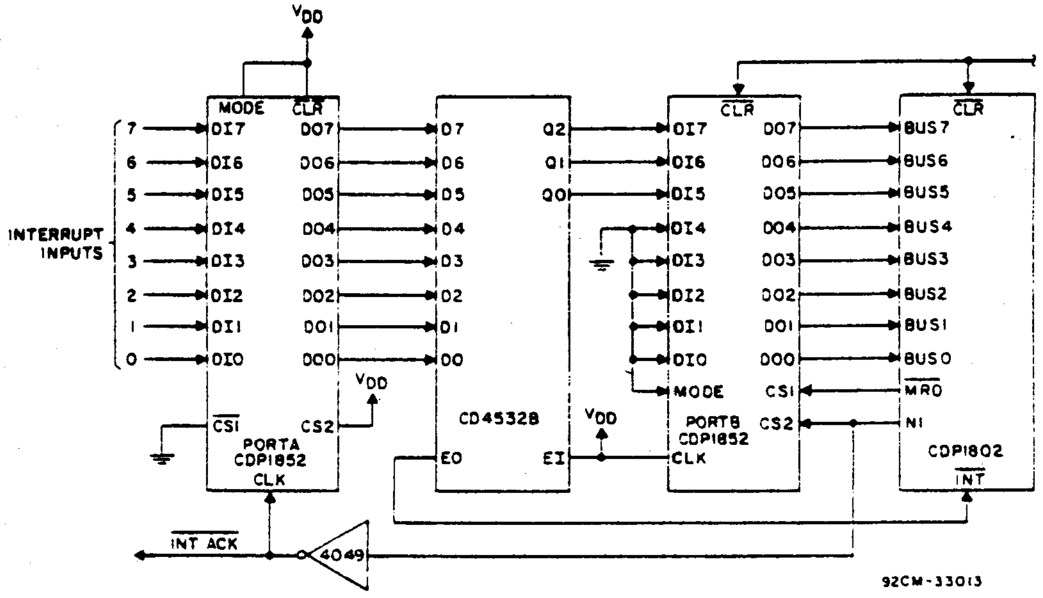
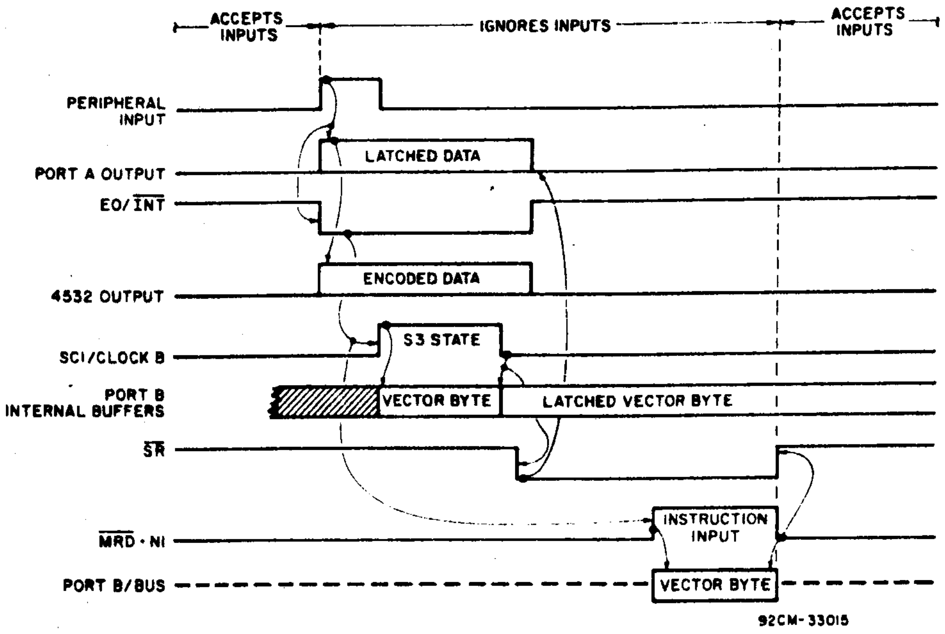
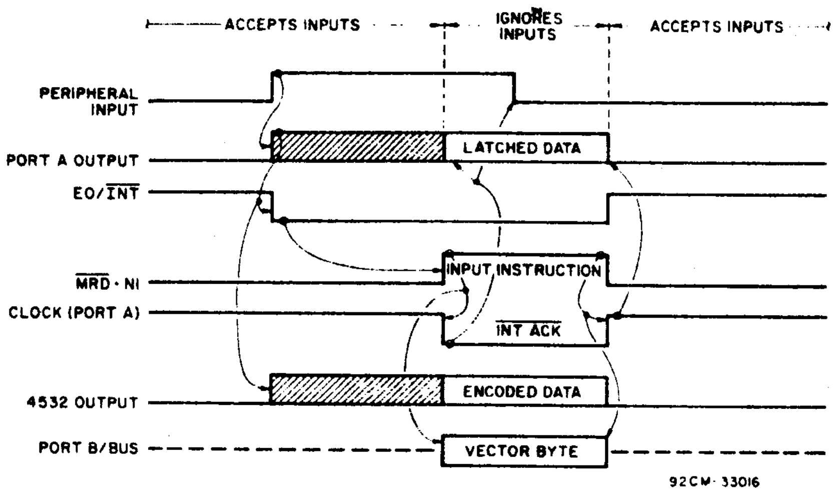


Figure 2

Timing Diagrams



Circuit #1



Circuit #2

Using The 1802 Scratchpad To Store RAM Variables

Small systems with modest RAM requirements can sometimes be implemented without external RAM by using a portion of the 1802 scratchpad register array to store variable data. Since the scratchpad can be configured for 16-bit addresses or 8-bit data, a typical small system could allocate 8 registers for pointer addressing, and still leave 8 registers for up to 16 bytes of "RAM" storage.

A difficulty arises when attempting to perform an arithmetic or ALU operation on register data: these operations require M(R(X)) and the D register as operands. Register to D manipulations can be performed with PHI, GHI, PLO, and GLO instructions, but R(X) cannot point to an internal register to complete the operation.

The problem can be solved if one page of ROM is available for use as a lookup table. With this method, one register operand becomes the lower order table pointer address, while the other operand is transferred to D. The lookup table contains sequential bytes from ØØ to FF, and when the arithmetic or ALU operation is performed, the table contents and D are operated upon, with the result in the D register.

EXAMPLE

- Register data is stored in R(9).1 and R(B).0
- An XOR instruction is to be performed $[M(R(X)) \oplus D \rightarrow D]$
- Lookup table is located in locations Ø3ØØ - Ø3FF
- R(7) is dedicated as lookup table pointer
- R(7).1 has already been initialized to Ø3

```
MACRO:

GHI R9 . . . Get first operand into D

PLO R7 . . . Use first operand to lookup table value

GLO RB . . . Get second operand into D

XOR    . . . Exclusive-OR D with contents of table address
```

if R(9).1 = AA and R(B).0 = FF

then R(7) would point to address Ø3AA

Contents of Ø3AA = AA

$M(R(X)) \oplus D = AA \oplus FF = 55$

D will contain 55 when operation is complete

The idea for this article was submitted by John Stahler, RCA, Des Plaines.

For information on related topics, contact Joe Paradise, X7352.

RCA
Solid State
Division

Microprocessors
Application Note
ICAN-6704

Optimizing Hardware/Software Trade-Offs In RCA CDP1802 Microprocessor Applications

By L.A. Solomon and D. Block

One of the chief reasons for choosing to design with a microprocessor rather than standard IC's is to reduce a system's parts count. To make the best choice requires a careful analysis of hardware/software trade-offs. This analysis usually narrows down to the ratio of ROM to I/O devices in the system. Economics indicates that the more functions handled in software, the less expensive and more flexible the system will be. Thus, a good design practice is to attempt to do everything in software initially and then relegate functions to hardware only as the speed/processing capability of the CPU becomes taxed. This Note will develop some examples of processor interfaces that not only minimize external hardware but, through judicious programming techniques, also minimize speed requirements on the CPU.

The RCA CDP1802 microprocessor is particularly well suited to minimum-cost interfacing because it has a significant number of terminal connections dedicated to I/O operations and an extensive set of I/O instructions. It has three I/O selection lines, called the "N" lines, that are controlled by I/O instructions plus four general-purpose flag input lines testable with branch instructions. There are also DMA-in, DMA-out, and Interrupt Request line inputs as well as two state code and two timing pulse outputs to synchronize I/O devices to the CPU. A single bit output (Q) which can be set or reset under program control is also provided. In all, 15 of the CDP1802 terminal connections are dedicated exclusively to I/O control. In addition, the CDP1802 has other unique architectural features, such as built-in DMA, that can be used to advantage. These features will also be discussed.

A CLASSICAL SYSTEM

A simple system having a keyboard input and a digital display output is shown in Fig. 1. The specific functions are

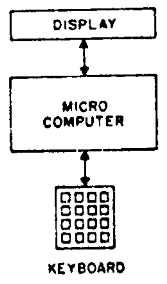


Fig. 1 - Simple microcomputer system.

omitted because the immediate concerns are only the microprocessor and I/O interfaces. These interfaces will be constrained by the programming technique choosen for the system. In the classical software control flowchart for this system, shown in Fig. 2, the standard

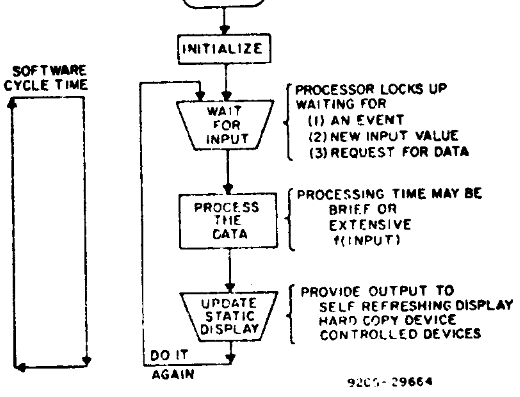

START
INITIALIZE
SOFTWARE CYCLE TIME
WAIT FOR INPUT
PROCESSOR LOCKS UP WAITING FOR (1) AN EVENT (2) NEW INPUT VALUE (3) REQUEST FOR DATA
PROCESS THE DATA
PROCESSING TIME MAY BE BRIEF OR EXTENSIVE f(INPUT)
UPDATE STATIC DISPLAY
PROVIDE OUTPUT TO SELF REFRESHING DISPLAY HARD COPY DEVICE CONTROLLED DEVICES
DO IT AGAIN
92CS-29664

Fig. 2 - Classical software control flowchart for the system of Fig. 1.

Printed in USA/10-78

initialization block is followed by an input, processing, and output procedure with a final loop back to repeat the action. Even without the details of the hardware or software design or the specific application, certain predictions can be made about this system.

First, consider the software cycle time, that is the time to go completely through one loop of the procedure. The cycle time is the sum of the time spent in each portion of the software, including input, processing, and output. Because the program apparently waits for an input, the time spent in the input block is indeterminate. The system cycle time, therefore, is indeterminate. This parameter has immediate impact on the selection of both input and output devices used in the system. The output device, for instance, must be capable of operating for prolonged periods without processor attention. Therefore, it must be a device that is self-refreshing or contains a latch. It certainly cannot be dynamic because no provision for refreshing is apparent in the simple software structure shown thus far. Because dynamically refreshed displays have the potential for lower cost, the static requirement is a serious drawback.

Next, consider an input device. A keyboard, being human operated, will present data to the processor at an uneven rate. The time between keystrokes may vary from a few milliseconds to several seconds or minutes. With the flowchart given, the processor must complete its processing before the next input can be received. If each keystroke requires some analysis by the microprocessor, a choice between using a very fast (and expensive) processor or lengthening the minimum time between keystrokes must be made. The first alternative would be very wasteful since the processor's very fast speed would only be needed in short bursts; most of the time it would be idling waiting for an input. The second alternative leads to an unresponsive system, one in which the operator will have to adjust to the system rather than the other way around. A third alternative is to design in an "intelligent" keyboard controller or buffering device to smooth out the input rate as depicted in Fig. 3. This alternative, however, is not ideal either because it requires additional hardware expense.

Resorting to additional hardware, however, may not be necessary if the flowchart of Fig. 3 is restructured. By doing the controller functions in the software the controllers can be eliminated at only the cost of enlarging the system ROM. Moreover, because ROM's come in fixed increments, it may be no more expensive to have a program that is 1024 bytes long than one that is 527 bytes, even though one is nearly twice as long as the other. In fact, if there is unused space in the system ROM, the controller function may be had for "free". Even if an additional ROM is required, it may cost less than the MSI or LSI controller being replaced.

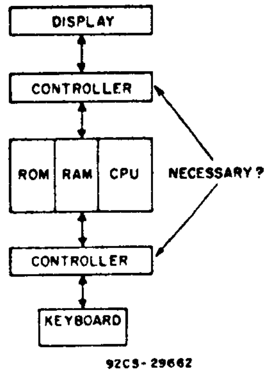


Fig. 3 - Addition of controllers to smooth out input rate.

To take advantage of software control, the approach is changed, so that instead of waiting for an input to take place, the system simply looks at the input periodically. If no input is present, it skips the input operation and goes on to something else. That something else could be the refreshing of a dynamic display, for example, or some processing required as the result of the last input. If an input is present, then it is accepted and acted on. There are several options available for handling the processing associated with this input. If the input is small and can be handled immediately, the system will do so. If not, it can be saved for later when there will be time to handle it, or it can be broken up into small computational blocks interspersed among other tasks such as display refresh. These approaches are flowcharted in Fig. 4. The latter approach is the idea behind a powerful technique called interpretive programming in which functions such as display refresh and keyboard scan are written as modular subroutines. Calls to these subroutines, which pass or pick up parameters from the main program, can be interspersed throughout the main program wherever required by the system timing considerations.

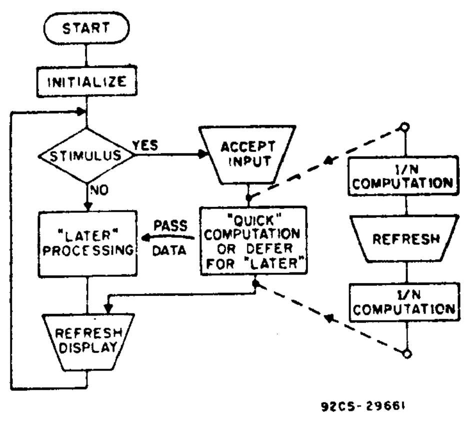


Fig. 4 - Controller function transferred to software and input loads interspersed.

HANDLING A DYNAMIC DISPLAY

Fig. 5 shows a typical multiplexed display system and Fig. 6 gives the details on the display refresh rate. The minimum refresh rate for any digit should be 100 Hz, which is fast enough to prevent flicker under most stationary display conditions.

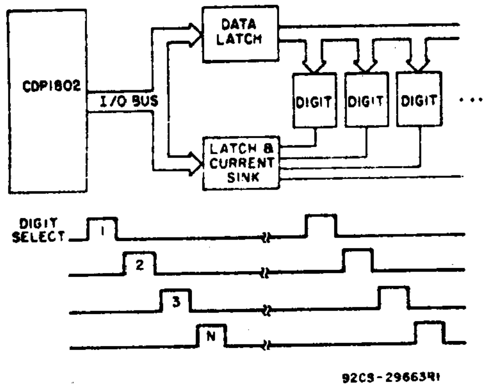


Fig. 5 - Typical multiplexed display system.

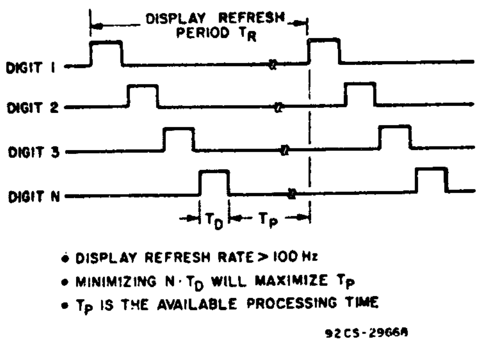


Fig. 6 - Details of the display refresh rate.

The actual ON time (T_D) of any digit is a trade-off between the intensity of the display and the time remaining within the 100-Hz refresh period for the processor to do some other work. It is desirable to minimize T_D so that a maximum of processing time (T_P) is left for the rest of the processing load.

It is customary to "overdrive" multiplexed LED displays to increase their apparent brightness. The extent to which overdrive is practical is a function of the duty cycle

$$\frac{T_D}{T_R}$$

of the display. This technique, however, is not without risk. Should the program crash or hang up (because of a program bug or noise injected into the system, or component failure, etc.), it is quite probable that a digit driver will be incinerated. Because of this hazard appropriate precautions, particularly when debugging a system, should be taken.

The segment information for a 7-segment display can be handled in either of two ways. If the data is in BCD, a device such as the CD4511 which contains a latch, BCD-to-7-segment decoder, and drivers can be used as shown in Fig. 7. Or,

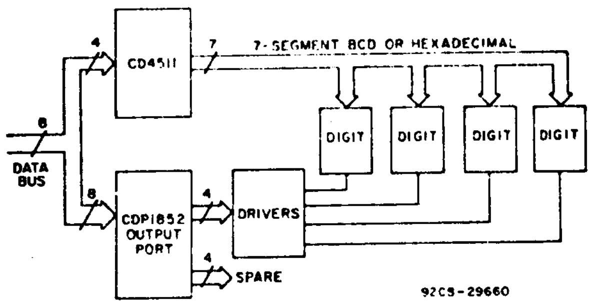


Fig. 7 - Handling segment information in hardware by means of a CD4511 BCD-to-7-segment latch decoder driver.

instead of the CD4511 that does code conversion in hardware, a software conversion via a look-up table can be used along with a simple output port as shown in Fig. 8. Hexadecimal or other codes are also easily accommodated in the table look-up method. But, because the output ports may not have sufficient drive to directly handle LED's, an intermediate stage of buffering may be necessary. No clear-cut recommendation can be made because variables such as the number of devices and the type of display chosen are significant.

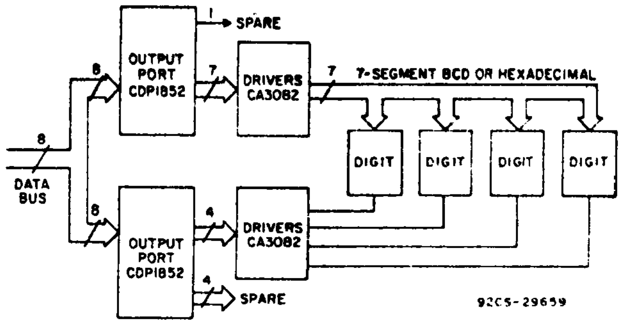


Fig. 8 - Handling segment information in software by means of an output port and lookup table.

SINGLE-SIGNAL INPUTS

The CDP1802 has four flag input lines that can be tested with branch instructions. These inputs are general purpose and can be used for such functions as interrupt vectoring, status indicators, or as single-bit inputs for slowly varying signals such as that of an ASCII terminal having a moderate baud rate. As an example, one of the flag lines is used as an input for a switch in Fig. 9. To signal

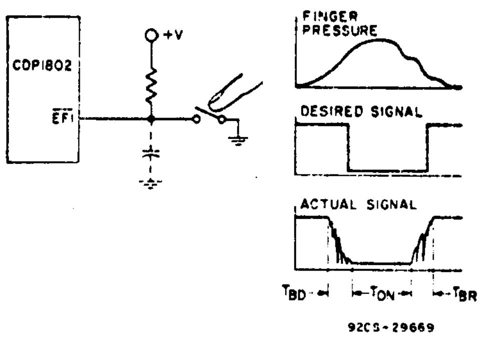


Fig. 9 - Basic switch circuit using microprocessor flag line.

the processor, a change on the flag line from a logic 1 to logic 0 level is used. However, the tendency of mechanical switches to "bounce" prevents this simplistic solution. The actual signal presented to the microprocessor consists of three parts - an initial bounce, a stable ON period, and a release bounce. A program looking only for a simple 1 to 0 to 1 transition may sense many switch closures because of the bounce noise. Although there are hardware solutions to this problem, software techniques may prove more cost-effective. Fig. 10 is a flowchart of a subroutine to debounce a mechanical switch. A test is made on the input signal to test for a switch closure. If none is found, a "switch down" software flag is

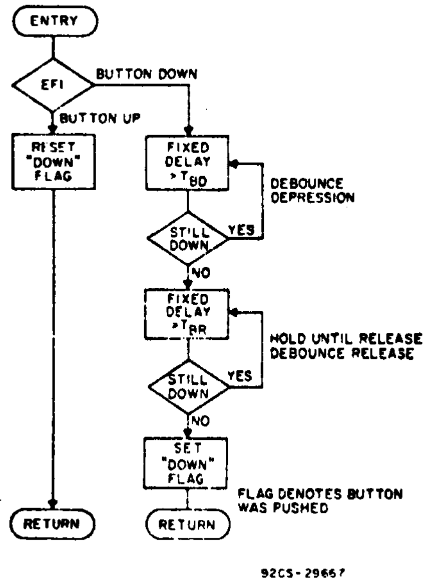


Fig. 10 - Flowchart of subroutine for debouncing a mechanical switch.

reset. This flag may be some convenient bit in one of the CDP1802's sixteen general purpose CPU registers or a bit in a RAM status word. If the switch is down, then the software will loop, waiting for the button to be released. The wait is performed to insure that the switch is not "seen" again for the current depression and to allow for the initial bounce period T_{BD}. Once the switch is released, the switch is again interrogated until it reaches a stable OFF condition. The software flag indicating a "switch down" condition is set, and the program returns to the caller. Although this program is easy to understand, it is, like the earlier simple solutions, not without its problems. For instance, the processor again wastes valuable time. The execution time (see Fig. 9) for this subroutine is at least

$$T_{BD} + T_{BR}$$

and does, in fact, last as long as the button is depressed. Thus, it is obviously not suitable for systems having dynamically refreshed displays. A further drawback, from the human-engineering standpoint, is that a response is made on the release of the switch rather than on its depression, the opposite of what one would normally expect.

Fig. 11 shows a flowchart for an im-

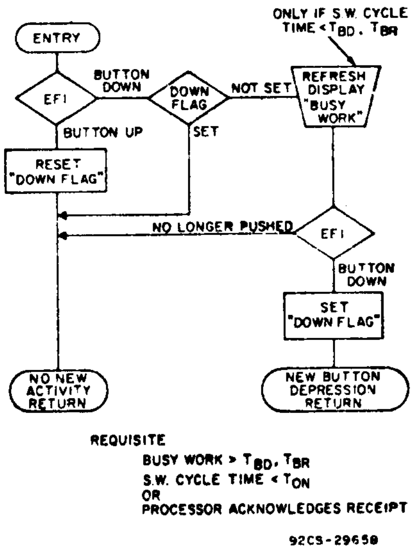

ONLY IF S.W. CYCLE TIME < T_{BD}, T_{BR}
ENTRY
EF1
BUTTON DOWN
DOWN FLAG
NOT SET
REFRESH DISPLAY "BUSY WORK"
BUTTON UP
SET
RESET "DOWN FLAG"
NO LONGER PUSHED
EF1
BUTTON DOWN
SET "DOWN FLAG"
NO NEW ACTIVITY RETURN
NEW BUTTON DEPRESSION RETURN
REQUISITE
BUSY WORK > T_{BD}, T_{BR}
S.W. CYCLE TIME < T_{ON}
OR
PROCESSOR ACKNOWLEDGES RECEIPT
92CS-29658

Fig. 11 - Flowchart of improved subroutine for debouncing a mechanical switch.

proved method that overcomes both of these drawbacks. Here, the subroutine that looks at the input signal has the capability of remembering what that signal was the last time it looked. This information is saved in a software flag called the "down flag". The routine operates as follows. If the button is now down and was also down the last time, then it is assumed that the system sees the same button depression seen earlier. A return is made to the caller with an indication of no new activity. If the button is not now down, but was the last time, then the switch has been released. In this case, the "down flag" is reset and a return made to the caller indicating no new activity (it is assumed that the processor is interested only in switch depressions and not their duration). But, if the switch is down now and was not down the last time, then there is a new depression. The switch must be debounced, the "down flag" set, and a message returned to the caller. Notice in the flowchart that a second test was made after the delay generated in the "busy work" block. This delayed second test is a debouncing technique to determine that the switch has been in the same state for two successive samplings before a decision is made on the true state of the switch. This method is still not optimal because the program is waiting (and therefore wasting time) during the debounce period. If some additional constraints are placed on the software cycle time, however, the program can be further optimized. For example, if the cycle time is greater than the bounce time (T_{BD}) but less than the switch ON time (T_{ON}), then the flowchart can be simplified to Fig. 12. Here there are no timewasting loops because switch bounce, in effect, will not be seen within the given timing restraints.

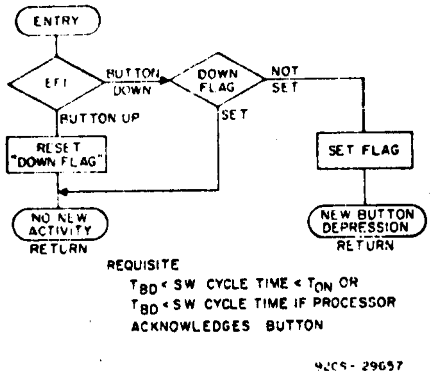

ENTRY
EF1
BUTTON DOWN
DOWN FLAG
NOT SET
BUTTON UP
SET
RESET "DOWN FLAG"
SET FLAG
NO NEW ACTIVITY RETURN
NEW BUTTON DEPRESSION RETURN
REQUISITE
T_{BD} < SW CYCLE TIME < T_{ON} OR
T_{BD} < SW CYCLE TIME IF PROCESSOR ACKNOWLEDGES BUTTON
92CS-29657

Fig. 12 - Flowchart of simplified debouncing subroutine benefitting from additional constraints.

MULTIPLE INPUTS

Up to four inputs can be handled as described above with each switch connected to a separate flag line of the CDP1802. Another technique is a multiplexing scheme in which the four switches are connected to one flag input, as shown in Fig. 13, and sequentially scanned as described in the flowchart of Fig. 14. This technique is readily expandable to additional scanned functions and, therefore, is discussed in detail. The

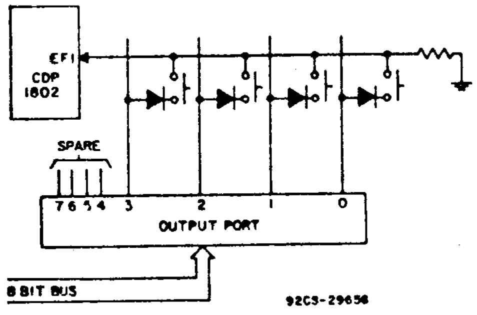


Fig. 13 - Hardware for handling four switch inputs on one flag line by means of a scanning routine.

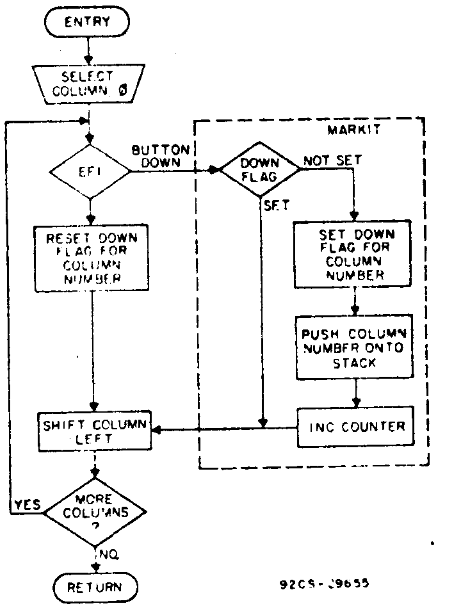


Fig. 14 - Flowchart of scanning routine for handling four switch inputs.

subroutine is designed to look for new switch closures and report them to the main program by "pushing" the switch number of a newly closed switch onto a stack and incrementing a counter. The main program will "pop" switch numbers off the stack and decrement the counter whenever the count is greater than zero. In the CDP1802 any one of the 16 general-purpose registers can be conveniently used as a counter because each has its own increment and decrement instruction.

The auxiliary functions for the subroutine are shown in Fig. 15. It is

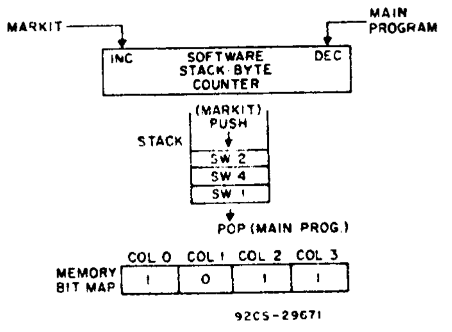


Fig. 15 - Auxiliary functions for the scanning subroutine of Fig. 14.

assumed that the timing constraints of Fig. 12 are met by this routine also, so that Fig. 14 is an extension of the basic flowchart previously developed. Upon entry into the subroutine, the first switch column is selected by outputting a 1 in bit position 0 of the data bus and examining the switch associated with that position. If a new depression is detected, the "down flag" is set for that switch in the memory bit map, the column number is pushed onto the stack, and the counter incremented. Next, the column is shifted and, if more columns remain to be scanned, the process is repeated. No switch closure or no new switch closure simply results in a column shift and continuation. When all columns have been scanned, a return to the main program is executed. The main program detects if any new switch closures have occurred by seeing if the counter has a value greater than zero. If so, the main program successively "pops" a switch number from the stack and decrements the counter until it reaches zero.

A section of the flowchart in Fig. 14 has been partitioned off and labeled "MARKIT". This routine is a common one that can be used as an expanded keyboard scan rountine discussed in the next section. It should be noted that the approach taken above lends itself well to a multi-processor system in which one processor handles the keyboard scanning and puts key numbers in a stack accessible to the other processors as well.

KEYBOARD SCANNING TECHNIQUES

Fig. 16 shows an arrangement for

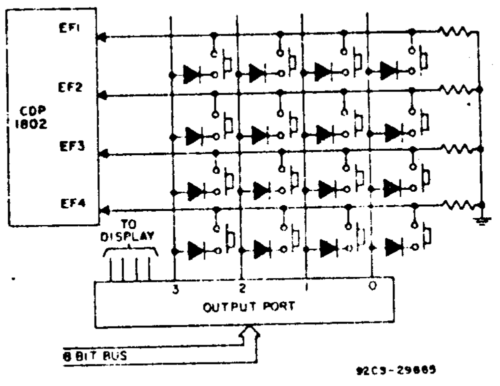
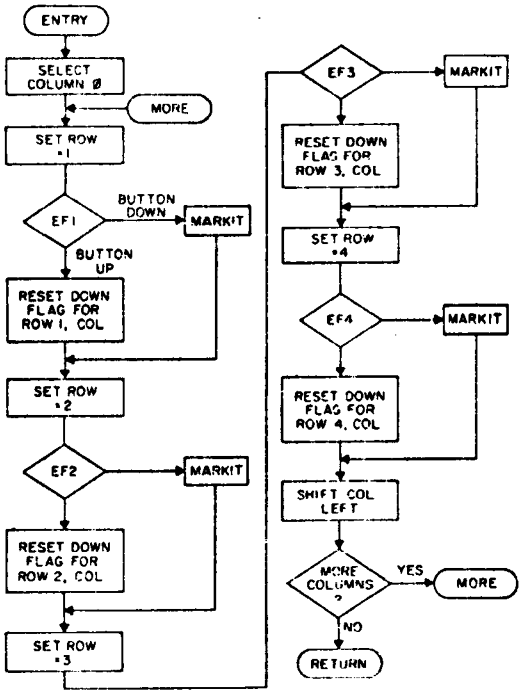


Fig. 16 - Hardware arrangement for handling a 16-key matrix using a scanning routine.

scanning a 16-key matrix. It is a simple extension of the arrangement just discussed. The horizontal lines can go directly into the four flag inputs of the CDP1802 as shown. Fig. 17 gives a flowchart of the software in which "MARKIT" is now responsible for handling row as well as column information. The basic interface between the main program and the keyboard scan subroutine remains the same; the subroutine place new key depressions on the stack and from there they are passed to the main program. Note that a key number's position on the stack does not necessarily represent when a given key was depressed with respect to the other keys on the stack, but merely indicates the order in which the keys were scanned. Because the stack is emptied on each cycle by the main program and only new key depressions are entered, the presence of two key numbers on the stack tells only that both keys were down when the scan took place. To discriminate in time between rapid key depressions, a short software cycle time is necessary. But, remember that this time must be kept within the constraints of T_{BR}, T_{BD}, and T_{ON}. There is a limitation to the technique discussed in that the software does not indicate to the main program when a key has been released. Thus, it can not be used in a system requiring lockout of other keys when any one key is down.

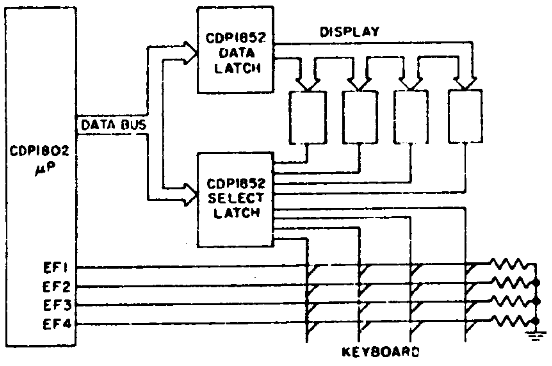
COMBINED DISPLAY AND KEYBOARD

The whole system of Fig. 1 is shown with its component blocks filled in on Fig. 18. The original objective to minimize



92CS-29666

Fig. 17 - Flow chart of software for handling row and column information utilizing "MARKIT" routine.



92CS-29654

Fig. 18 - Simple microcomputer system of Fig. 1 with component blocks expanded.

hardware has been realized in that only two 8-bit output ports are required in this design besides digit drivers (not shown).

A further improvement can be made in the system by combining the keyboard scan and display multiplexing signals as shown in Fig. 19. Here, a single-byte

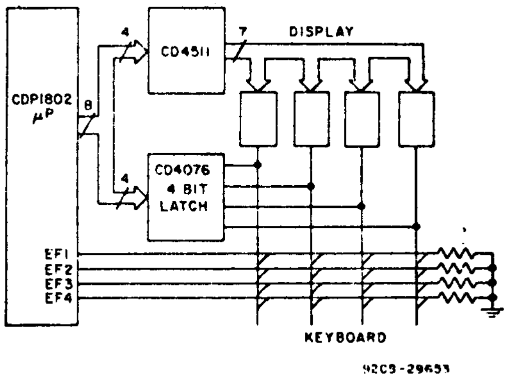


Fig. 19 - System improvement made by combining keyboard scan and display multiplexing signals.

output is used with the upper-order 4-bits being BCD data for the display and the lower-order 4 bits used to simultaneously select a display digit and keyboard column. This arrangement does not reduce the parts count, but does give smaller packages if space is a consideration and cuts down on the number of output operations and output bytes stored. A ready expansion of the system shown in Fig. 20 still uses only two IC's but permits scanning two 16-key keyboards and an 8-digit display.

TIMING GENERATION

In many applications it may be necessary to have some time-keeping ability in the microprocessor system. The requirements may range from having a time-of-day or elapsed-time clock to microsecond timing resolution for generating precision pulse widths. Here again, of the many approaches possible to timekeeping, a cost/performance-optimized one can be found.

Consider an example, shown in Fig. 21, for generating an output pulse of width T_1 each time switch S_1 is closed. The CD-

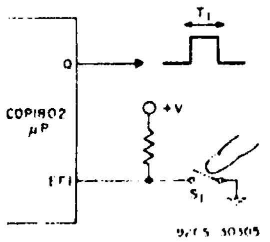


Fig. 21 - System for generating an output pulse for each switch closure.

P1802 has a single-bit output called the Q flip-flop that can be set or reset under program control to perform this function. The simplest technique for generating a fixed delay is by executing a series of "no-ops" in the program as illustrated in Fig. 22. If each "no-op" takes 5 microseconds to execute, for example, and T_1 is 50 microseconds long, then ten "no-ops" would do the job. This technique is obviously not a realistic one for long timing intervals because it is extremely wasteful of memory and fully occupies the processor with a non-productive task.

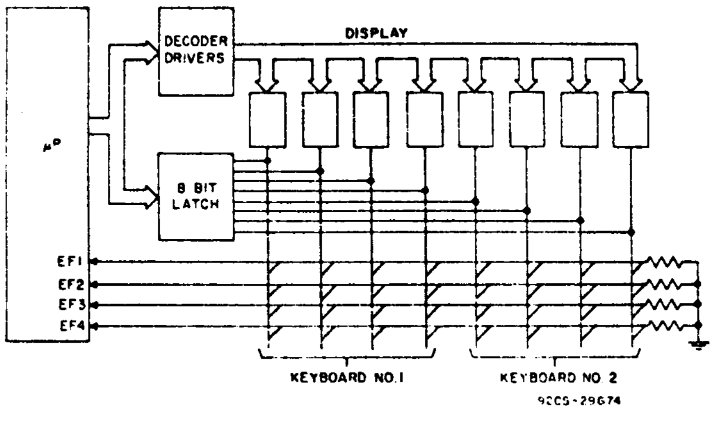


Fig. 20 - An expanded system with 8-digit display and two 16-key keyboards.

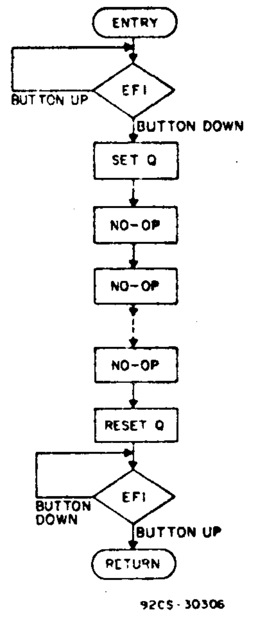


Fig. 22 - Primitive programming technique for pulse generation.

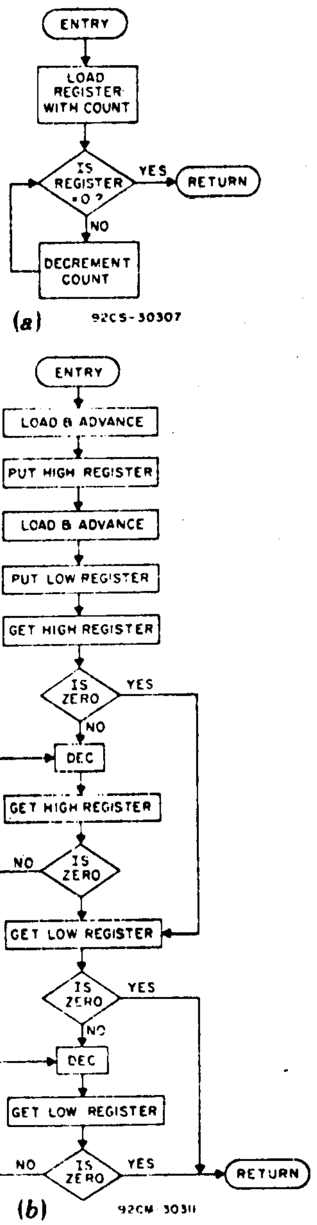


Fig. 23 - (a) Basic flowchart of improved technique for pulse generation. (b) Specific instruction sequence for improved pulse-generation technique.

A better technique is shown in Fig. 23(a). Here a counter is preset with a given value and continually decremented until it reaches zero. In the CDP1802 one of the 16-bit general-purpose registers can be used for this function. Fig. 23(b) shows the specific instruction sequence. This technique saves a lot of memory bytes but still ties up the processor. For maximum processor efficiency it would be best to load an external counter that would count at some preset rate and generate an interrupt to the processor when it reaches zero. Meanwhile, the processor could be doing some useful work. Such a system is shown in Fig. 24, but it does not minimize system hardware.

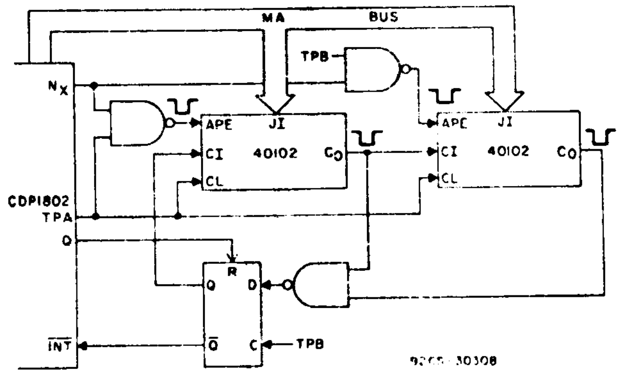


Fig. 24 - System for pulse generation using an external counter.

An interval timer can readily be made for the CDP1802 with no external parts by making use of an internal register that can be automatically incremented. General-purpose register R0 is used as a pointer for DMA operations in the CDP1802 and, as such, is automatically incremented on each DMA (in or out) cycle. By connecting the State Code 1 (SC1) line output back to the DMA Out request line ($\overline{DMAO}$), the CDP1802 performs one DMA cycle for each Fetch and Execute cycle, as shown in Fig. 25, thereby providing a built-in timer and instruction counter. With a clock frequency of 1.57 MHz, the most significant bit of R0 will change each 1/2 second, providing a real-time clock. CPU operation is, of course, slowed by 1/3 with this scheme, but with an upper clock frequency of 6.4 MHz the system can be made fast enough for many applications.

With the circuitry of Fig. 26, a general-purpose interval timer can be realized. This circuit will cause an interrupt when the most significant bit of register R0 goes to a "one". Thus by preloading R0 with a desired count, a timing interval with a range of 2^{15} and a resolution of up to 3.75 microseconds (with a clock frequency of 6.4 MHz) can be obtained with a minimum of external hardware and yet leave the processor free to do useful work.

ACKNOWLEDGMENT

The authors wish to acknowledge the contributions of Messrs. K. Karstad and F. Thorley for developing some of the techniques described.

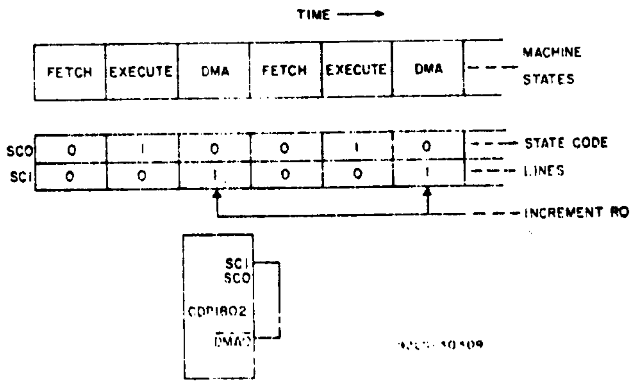


Fig. 25 - Use of DMA-Out line to implement an internal timer.

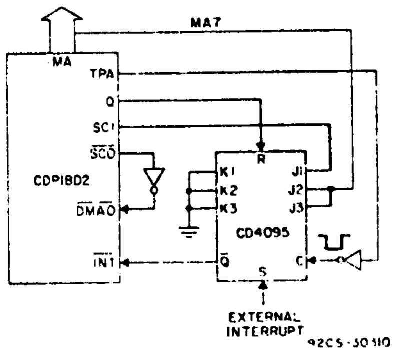


Fig. 26 - Use of interrupt line to implement a general-purpose timer.

CDP1802 Expanded Output Control Using the CD4099

For simple I/O control applications, the only available 1802 output control line with latched set and reset capability is the Q output, controlled with a single SEQ or REQ software instruction. With the addition of a CMOS CD4099/CD4724 and a single inverter, a comparable single instruction can expand this capability to 8 outputs.

Conventional I/O control usually involves an I/O port, which latches control data from the data bus line. The primary disadvantage is that several instruction bytes (and instruction cycles) are required to modify a single control bit based on a conditional computed result. Besides the disadvantage of additional ROM code, the response time of a peripheral device to a CPU command is delayed by the time it takes to execute this instruction sequence.

The circuit and waveforms of Figures 1 and 2 illustrate the implementation of a CD4099/CD4724 as an alternate means of expanding I/O control. The CD4099/CD4724 is an inexpensive addressable latch in a 16 lead package that can use the N lines to select an internal latch for modification. Depending on the level of the data input at the time WD is low, the selected output can be set or reset without modifying any previous condition on the other outputs. This can be easily accomplished with the I/O instructions on the 1802, which can generate 7 combinations of N line codes, with $\overline{\text{MRD}}$ high for an INP instruction, and low for an OUT instruction. These combinations result in the table of single instructions shown in Table 1 that can selectively set or reset individual control bits in the expanded 1802 system.

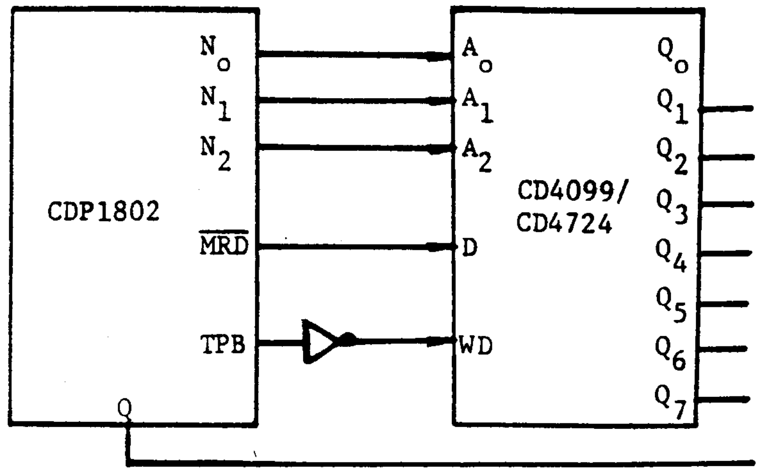


Fig. 1 - CD4099/CD4724 Hardware Implementation

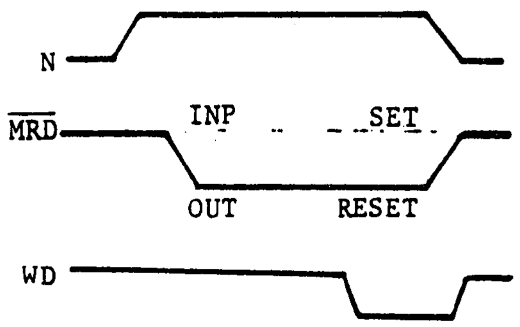


Fig. 2 - Timing Diagram

TABLE I - Single Bit Output Control Codes

	Set With		Reset With	
Output Pin	Mnemonic	Machine Code	Mnemonic	Machine Code
Q_1	INP 1	69	OUT 1	61
Q_2	INP 2	6A	OUT 2	62
Q_3	INP 3	6B	OUT 3	63
Q_4	INP 4	6C	OUT 4	64
Q_5	INP 5	6D	OUT 5	65
Q_6	INP 6	6E	OUT 6	66
Q_7	INP 7	6F	OUT 7	67
Q	SEQ	7B	REQ	7A

CDP1802 Versus CDP1802A - Performance Enhancements

In the past, latching of the high-order address byte determined system speed for most 1802 based designs. The original CDP1802 CPU had a high byte set up before TPA trailing edge time (Tsu) which would diminish to zero at a clock speed of 2.5 MHz (see Fig. 1), thrus limiting system performance to 2.5 MHz maximum.

The new CDP1802A CPU has modified internal logic and art work changes, which reduce the memory address propagation delay, and provide a more generous set up time (Tsu) than its predecessor. The maximum clock frequency is no longer limited by Tsu, but depends entirely on internal logic speed limitations. Operation at 5V, 3.2 MHz, from -40°C to +85°C is guaranteed, with a minimum Tsu of 75 nsec. (See Fig. 1).

Another design improvement is the addition of an internal Schmitt Trigger buffer to the CLEAR input, eliminating the need for external logic devices for power on reset. The internal Schmitt input provides a hysteresis voltage of approximately 1/2 volt, so that the RC network may be connected directly to pin 3 as shown in Fig. 2.

The CDP1802A CPU is pin for pin compatible and functionally equivalent to the older CDP1802 CPU, and will perform in any existing 1802 based designs.

Figure 1:

Minimum High-Order-Memory-Address Byte setup to TPA time, Tsu

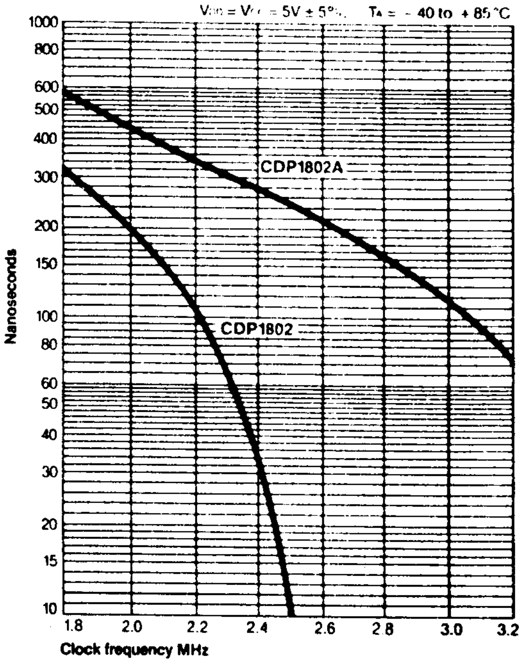
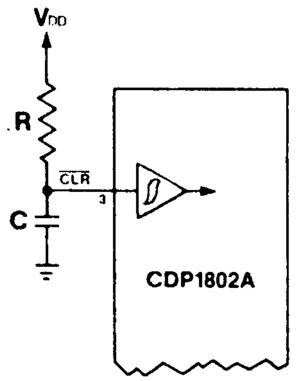


Figure 2:



RCA
Solid State Division

Microprocessor Products
Application Note
ICAN-6842

16-Bit Operations in the CDP1802 Microprocessor

by D. Block

Although the CDP1802 microprocessor is an 8-bit machine, it contains mostly 16-bit registers. Its sixteen 16-bit registers are all general-purpose types, giving the CDP1802 a great deal of flexibility and the flavor of a 16-bit microprocessor in many respects. This paper describes various software routines and a few interface circuits that can be used to manipulate full 16-bit values in the CDP1802.

The areas of logical, shift, and arithmetic operations are all considered along with I/O and loop counters. In sophisticated systems where multiplications and/or divisions are required, a hardware approach using the CDP1855 Multiply/Divide unit should be investigated. These 8-bit units can be cascaded, so that up to 32-bit operations can be performed. For simpler systems, the software approach described below is probably the best choice.

General

In many of the following examples, a 16-bit data word will be located in memory. Since the CDP1802 is an 8-bit machine, of necessity the word will have to be stored as two 8-bit bytes. As a convention, assume that the 16-bit word is located in two consecutive locations with the most significant byte occupying the higher address.

Logic Operations

The logic operations provided by the CDP1802 are AND, OR, and EXCLUSIVE OR. These three operations are normally performed between an operand in the D register and a byte from memory. With the addition of a few instructions, register-to-register operations can also be accomplished, as shown in Example 2, below.

Example 1: Register/Memory Operations

In this example, a 16-bit OR is performed between the contents of an R register called REG1 and two bytes of memory pointed to by another register called POINTR. The results will be in REG1.

```
SEX POINTR     ..POINT X TO FIRST
                 (LOWEST)
                 MEMORY BYTE
GLO REG1
OR             ..OR LOW BYTES
PLO REG1
INC POINTR     ..POINT TO
               ..HIGHEST
                 MEMORY BYTE
GHI REG1
OR             ..OR HI BYTES
PHI REG1       ..RESTORE
                 RESULTS
```

Example 2: Register-to-Register Operations

If the two operands are in two registers called REG1 and REG2, the code below can be used; it is only three bytes longer than that of example 1. Here, again, the results will be returned to REG1.

```
SEX POINTR     ..SET A POINTER
                 TO A FREE BYTE
GLO REG1
STR POINTR     ..STORE LOW BYTE
                 OF REG1
GLO REG2
OR             ..OR LOW BYTES
PLO REG1       ..RESULTS
                 RETURNED TO
                 REG1
GHI REG1
STR POINTR     ..STORE HI BYTE
                 OF REG1
GHI REG2
OR             ..OR HI BYTES
PHI REG1       ..RESULTS TO REG1
```

Shift Operations

In the CDP1802, shifts are performed on the contents of the D-register in either circular shifts, with data bits circulating through DF, or open shifts, in which zeros are shifted into one end of the D-register.

Printed in USA/7-80

Either type of operation can be performed on the full 16 bits of an R-register, as shown below. Right shifts are performed by substituting the appropriate shift-right commands for the shift-left commands given in the examples.

Example 3: Shift Left (Open)

```
GLO REG1        ..GET LOW BYTE
SHL             ..SHIFT LEFT
PLO REG1        ..RESTORE
GHI REG1        ..GET HI BYTE
SHLC            ..SHIFT LEFT WITH
                  CARRY BRINGS IN
                  LSB
PHI REG1        ..RESTORE
```

Example 4: Shift Left Circular

```
GLO REG1
SHL             ..SET UP CARRY
                  INTO HI BYTE
GHI REG1
SHLC            ..SHIFT IN CARRY
PHI REG1        ..CARRY INTO LOW
                  BYTE LEFT IN DF
GLO REG1
SHLC            ..SHIFT CARRY
                  INTO LOW BYTE
PLO REG1
```

A 16-bit shift of two consecutive memory bytes is performed similarly by replacing the GET and PUT statement by memory reference commands as follows:

Example 5: Shift Left-Memory Location

```
LDN REG1        ..REG POINTS TO
                  LOW BYTE
SHL
STR REG1
INC REG1
LDN REG1
SHLC
STR REG1
```

Example 6: Shift Left Circular — Memory Locations

```
LDA REG1
SHL
LDN REG1
SHLC
STR REG1
DEC REG1
LDN REG1
SHLC
STR REG1
```

Arithmetic Operations

Sixteen-bit arithmetic is straightforward in the CDP1802 since arithmetic operations including the value of the DF flag are included in the instruction set. Arithmetic operations are performed between a byte in the D-register and a byte from memory but, again, the addition of a few extra instructions makes register-to-register operations possible.

Example 7: Register/Memory Addition

```
SEX POINTR      .. POINT TO
                  MEMORY
                  LOW BYTE
                  OPERAND
GLO REG1
ADD
PLO REG1        ..RESULTS
                  RESTORED TO
                  REG1
IRX             ..POINT TO HIGH
                  BYTE
GHI REG1
ADC
PHI REG1
```

Example 8: Register/Register Addition

```
SEX POINTR      ..STORE REG2 HI
                  BYTE
GHI REG2
STXD
GLO REG2
STR POINTR      ..STORE REG2 LOW
                  BYTE
GLO REG1        ..SAME AS
                  PREVIOUS
                  EXAMPLE FROM
                  HERE
ADD
PHI REG1
IRX
GHI REG1
ADC
PHI REG1
```

Note that if each byte had been operated on in sequence, as was done in Example 2, rather than storing both halves of REG2 and then operating on them, the code could have been reduced by one instruction. This reduction could be significant in some applications.

Counters

An increment or decrement instruction to an R-register operates on the full 16 bits of the register. Thus, loop counters of up to 65k counts are readily available. The most direct implementation of a loop counter involves the presetting of an R-register with the desired number, decrementing it once each time through the loop, and testing for zero in the counter. The following example sets up a loop counter for 512 counts.

Example 9: Output 512 Bytes

```
LDI #02; PHI COUNTR
                ..SET COUNTER =
                  #0200
```

```
LDI #00; PLO COUNTR
LOOP: OUT1       ..DO AN OUTPUT
                   OPERATION
DEC COUNTR       ..DECREMENT THE
                   COUNT
GLO COUNTR       ..CHECK LOW
                   HALF OF
                   COUNTER
BNZ LOOP
GHI COUNTR       ..CHECK HI HALF
                   OF COUNTER
BNZ LOOP
XX               ..NEXT INSTRUC-
                   TION AFTER LOOP
```

Note that an output instruction outputs MR(X) and then increments R(X), so that this example would output 512 consecutive bytes from memory. Of course, for smaller loops of less than 256 counts, only the low half of a register need be examined, and two instructions can be removed from the loop.

Care must be exercised, when designing timing loops, to equalize the various branch path lengths. The method shown in example 9 would not be suitable for real-time loops. Instead, a loop of the form shown in example10 should be used.

Example 10: Timing Loop

```
LDI #20; PHI TIMER
                 ..SET COUNT
LDI #00; PLO TIMER
LOOP: DEC TIMER
                 ..DECREMENT
                   TIMER
GLO TIMER
BZ ENDTST        ..TEST LOW BYTE
GLO TIMER        ..DUMMY, IN-
                   STRUCTION TO
                   MATCH DELAYS
BR LOOP
ENDTST: GHI
 TIMER           ..TEST HI BYTE
BNZ LOOP
XX               ..NEXT IN-
                   STRUCTION
```

Two notes of caution should be mentioned here. First, a short branch instruction takes two machine cycles, whereas a long branch requires three -- avoid mixing them in a loop. In particular beware of the trap caused by editing long branches into a file in response to assembler-generated "branch out of page" errors. Make sure first that these branches are not in a timing loop. Second, beware of inserting NOP instructions to match delays, as these are three-machine-cycle instructions

A Hardware Approach to Timing Generation

A general-purpose time-delay subroutine can be devised which takes a passed parameter, puts its value into a register, counts it down to zero, and returns to the caller. Significant time delays can be generated by this method when a 16-bit value is passed, as shown below:

Example 11: General-Purpose Time Delay Subroutine

The main program, using the Standard Call and Return Technique, would look like this when #OFFF is the value being passed:

```
.
.
.
.
SEP R4, A(DELAY), #OFFF
.
.
```

The Subroutine itself follows:

```
DELAY:LDA R6
 PHI TIMER       ..LOAD PASSED
                   PARAMETER
LDA R6; PLO TIMER
SKP              ..THIS ENABLES
                   ZERO TO BE
                   PASSES
LOOP: DEC TIMER
                 ..DECREMENT
GLO TIMER        ..TEST LOW BYTE
BZ ENDTST
GLO TIMER        ..DUMMY INST TO
                   MATCH DELAYS
BR  LOOP
ENDTST: GHI TIMER
                 ..TEST HI BYTE
BNZ LOOP
SEP R5           ..RETURN-WHEN
                   ZERO REACHED
```

A method of creating an on-board time in the CDP1802 by using its built-in DMA facilities is described in Reference 1.

Input/Output

Inputting a 16-bit word to one of the R registers is a trivial matter in both hardware and software. Fig. 1 shows two input ports where N0 and N1 have been used to select the low and high bytes, respectively, of a 16-bit word. The code required to load this word into REG1 appears below:

Example 11: Input a 16-Bit Word to a Register

```
INP1             ..BRING LOW BYTE
                   INTO D
PLO REG1         ..TRANSFER IT TO
                   LOW HALF OF
                   REG1
INP2             ..BRING IN HI BYTE
PHI REG1         ..STORE IT
```

A shorter software sequence can be developed to input the word into two con-

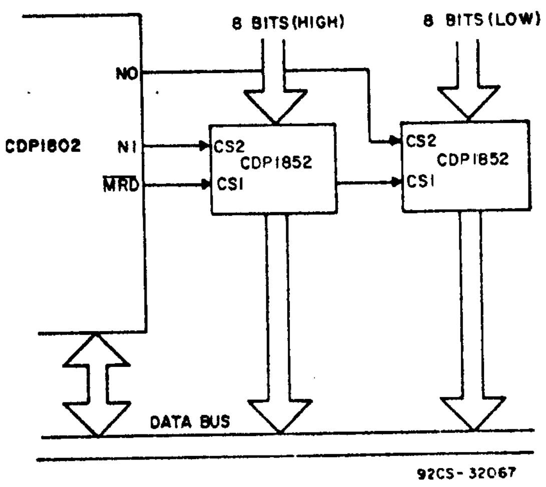


Fig. 1—Basic 16-bit input circuit.

secutive memory locations since an input byte is automatically written into MR(X) as well as the D register. However, hardware can be devised that will automatically input both bytes when a single input instruction is executed. This circuitry, shown in Fig. 2, operates as follows: Execution of input instruction INP 1 will not directly read in a byte. However, it sets the Service Request (SR) of PORT 1, causing the next machine cycle to be given over to a DMA-IN operation which will read in PORT 1. The timing is such that two DMA cycles will actually occur, inputting PORT 1 and PORT 2 to sequential memory locations pointed to by R(0). The software for this sequence would consist only of:

```
SEX R0 ..
INP 1  ..
```

Besides saving code, this approach is faster in the input operation (since DMA operations take only one machine cycle) than a sequence involving two inputs and a register increment.

A full 16-bit output can be obtained from the CDP1802 with one instruction, as shown in Fig. 3. Here, the contents of R(X) will be latched into PORTS 1 and 2 during the execute cycle of the output instruction.[2] Since any one of the 16-bit registers can be output to the address lines, this technique is a very powerful one.

Reference

1. "Optimizing Hardware/Software Trade-Offs in RCA CDP1802 Microprocessor Applications," L.A. Soloman, D. Block, RCA Solid State Application Note ICAN-6704.
2. A more detailed discussion of this "register output" operation can be found in "Register Based Output Function for RCA COSMAC Microprocessors," N. Swales, RCA Solid State Application Note ICAN-6562.

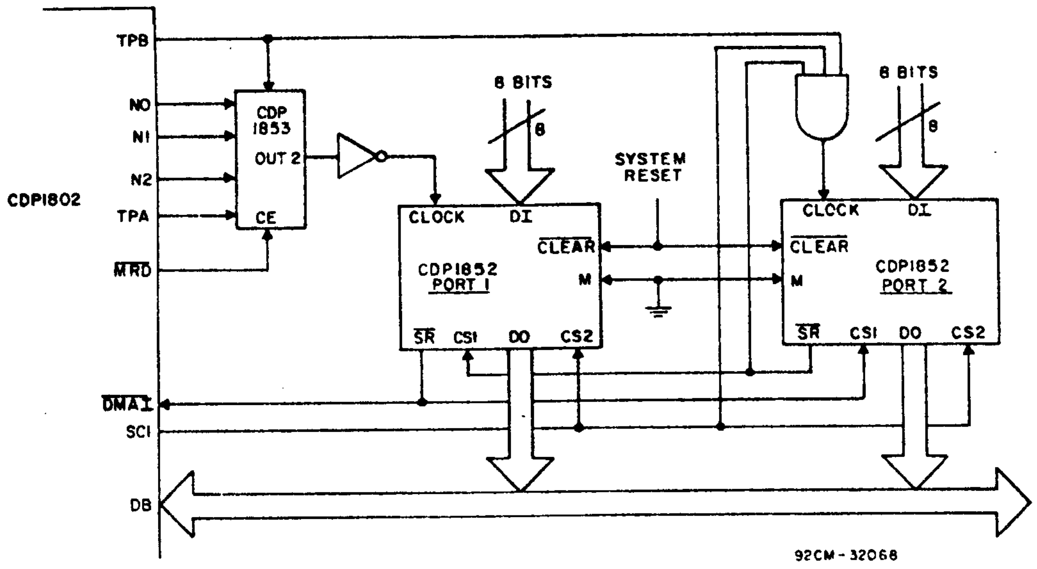


Fig. 2—Circuit for automatic input of a 16-bit value to memory.

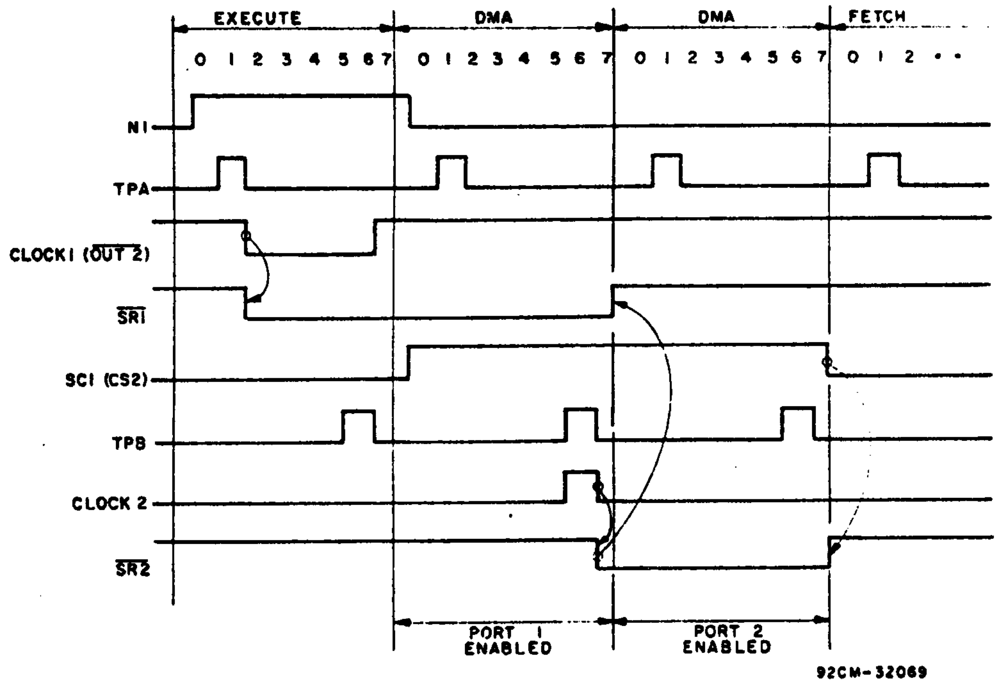


Fig. 3—Timing diagram for the circuit of Fig. 2.

CDP1804 and CDP1805 Processors Improve System Performance and Lower Chip Count

by J. Paradise

ST-6957

Prepared for
and presented at
Southcon/81

RCA|Solid State Division
Somerville, N.J. 08876
Printed in USA/11-80

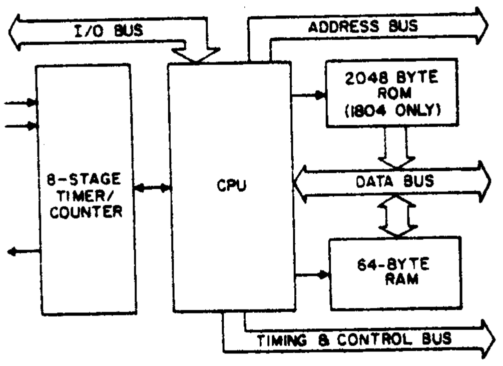
CDP1804 and CDP1805 PROCESSORS IMPROVE SYSTEM PERFORMANCE AND LOWER CHIP COUNT

Joseph Paradise
RCA Solid State Division
Somerville, N.J. 08876

The CDP1804 and CDP1805 processors are RCA's new introductions to the growing CDP1800 family of microprocessor and memory devices. These chips extend the capability of the CDP1802 microprocessor, both in higher performance and additional system functions, while maintaining upward software and hardware compatiblity. To give the system designer a choice between flexibility and minimum chip count, two parts are offered: the CDP1805 for moderate cost and off-the-shelf availability, and the premium CDP1804 for custom VLSI system integration.

FUNCTIONAL DESCRIPTION

The CDP1804 is a CMOS, 8-bit, register-oriented microcomputer designed for use in a wide variety of general-purpose computing and control applications. It contains a 2048-byte mask-programmable ROM, 64 bytes of RAM, an 8-bit presettable down counter, and the same architecture as the CDP1802. The CDP1805 is identical to the CDP1804, with the exception that the ROM is left out for reasons of economy and flexibility. Both devices are capable of dc to 4-MHz operation at 5 volts over the commercial temperature range of -40°C to +85°C, and both have a voltage-range capability of from 4 to 10.5 volts. These added hardware and performance features, in addition to an enhanced instruction set, make the CDP1804 or CDP1805 a suitable choice for customers up-grading present CDP1802 systems for higher system performance or greater system integration or considering the CDP1800-series family for the first time. A block diagram of the CDP1804/CDP1805 processors is shown in Fig. 1.



1. CDP1804/CDP1805 block diagram.

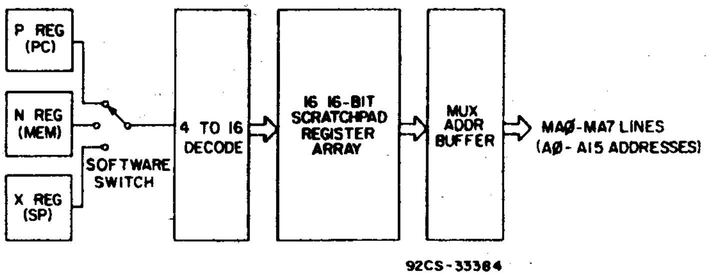
CDP1800 ARCHITECTURE IN SUMMARY

This section of the paper is designed for potential CDP1804/05 users unfamiliar with CDP1800-series architectural features, and explores the software and hardware aspects of data transfer and manipulation, and the control and timing interface to support devices. While established users of other 8-bit machines may find this architecture initially perplexing because of the extreme flexiblity of assignments within register and memory space, familiarity with the device and its capabilities will pay off in compact code generation and efficient use of memory and I/O in most control-oriented applications.

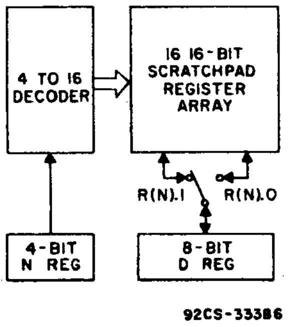
Scratchpad Register Array - The CDP1804/05 devices provide an indirect means of addressing memory through register assignment. Both devices contain sixteen 16-bit internal scratchpad registers (in addition to 64 bytes of RAM) that are user-programmable as program/subroutine counters, memory pointers, or stack pointers for memory addressing, Fig. 2. In addition, these same registers can hold data transferred to or from the accumulator (D register) by means of software instructions, Fig. 3. Finally, the register contents can be incremented or decremented for software loops or time delays.

Memory Addressing - When used to address memory, the registers are selected by software instructions that load 4-bit values into register selectors. The 4-bit P register selects a 16-bit scratchpad as the program counter, the 4-bit X register selects a scratchpad as the stack pointer, and the 4-bit N register selects a scratchpad as the memory pointer during an external data transfer, or as an operand during an internal data transfer (register-accumulator, register-register). Note that all 16 scratchpads are capable of addressing any of the 64K memory locations available to the CDP1804/05 processors; thus, stack space is unrestricted, scratchpads can point to subroutines located anywhere in memory space, and most registers can be dedicated for specific data storage or addressing tasks during subroutine or interrupt processing.

Addressing Modes - As a result of this register-oriented structure, addressing modes include direct, paged direct, immediate, indirect, and inherent. The direct mode applies to all branch instructions, which can



2. Common CDP1802/CDP1804/CDP1805 scratchpad register model, addresses.



3. Common CDP1802/CDP1804/CDP1805 scratchpad register model, data transfer.

cause conditional or unconditional jumps within the current page or anywhere in memory space. Indirect addressing allows transfer of data to or from either general memory or stack, depending on user allocation of memory space. The inherent mode allows for internal register modification with external memory inactive.

Instruction Set - In addition to memory reference instructions, the CDP1804/05 instruction set has a full complement of arithmetic and logic operations, conditional page and long branch instructions that test the contents of the accumulator and carry flag, register instructions that modify the contents of the internal scratchpads or register selectors, and I/O and interrupt handling instructions, Table I.

Bus Structure - The CDP1804/05 processors use a multiplexed address bus to address external memory. The high byte is generated first, with a TPA pulse provided to latch the byte into an external latch or a bus-compatible memory-support chip. The data bus is nonmultiplexed, with a TPB pulse provided to latch stable data into an I/O device. A separate, 3-bit, I/O address bus provides address selection for external peripheral chips.

Table I - Breakdown of 91 Instructions Common to CDP1802/04/05

MEMORY TRANSFER	7
INTERNAL REGISTER	7
LOGIC	10
ARITHMETIC	12
UNCONDITIONAL JUMPS	4
CONDITIONAL JUMPS	27
CONTROL	7
INTERRUPT CONTROL	3
I/O TRANSFER	14

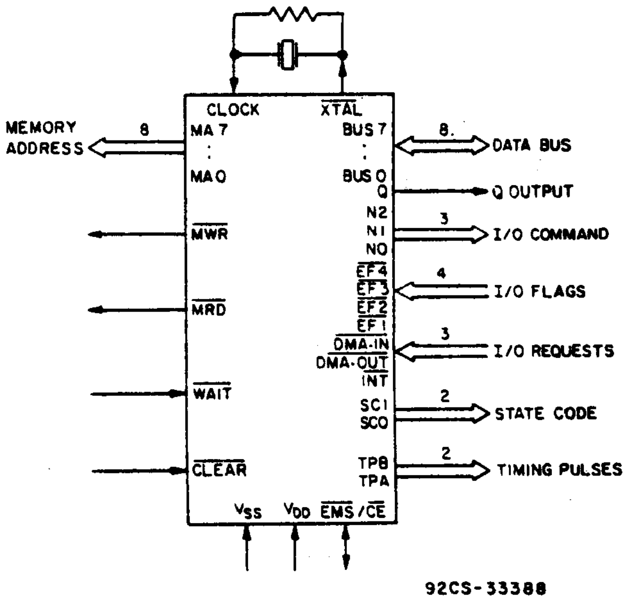
92CS-33392

Control and Status Pins - Control pins are provided for DMA transfer (with register R(0) as the DMA counter), interrupt requests (with register R(1) storing the interrupt vector), four testable flags, and a single-bit software-controlled output port. Two state code lines provide machine status. Separate READ and WRITE signals are provided for memory control, Fig. 4.

Execution Speed - System timing is derived from an external crystal. The crystal is divided so as to generate eight clock cycles per machine cycle. A single FETCH and EXECUTE (16 clock cycles total) is all that is required for two-thirds of the instructions available with the CDP1804/05 devices. The result is a minimum instruction time of four microseconds at maximum frequency, Fig. 5.

CDP1800 ARCHITECTURE - USER ADVANTAGES

The register-based orientation of the CDP1800-series architecture discussed in the previous section is its dominant feature, and

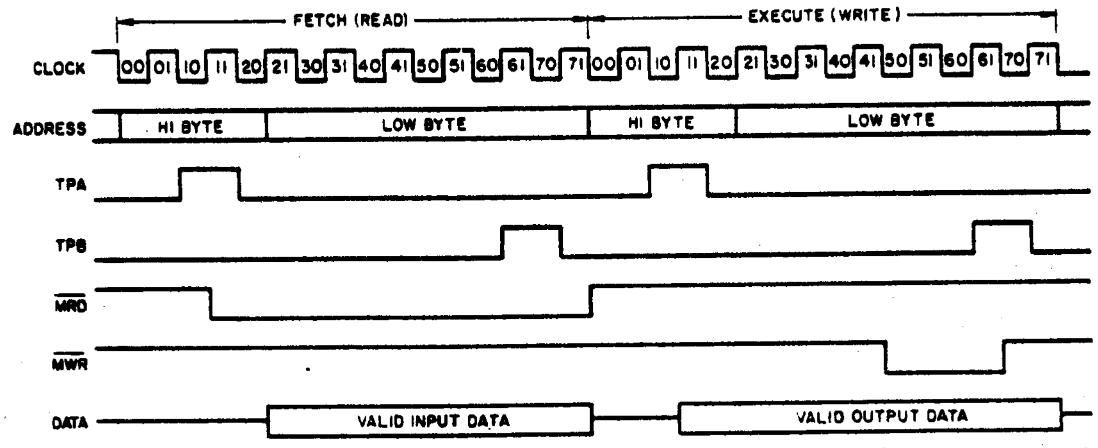


4. CDP1804/CDP1805 functional pinout.

CPU section to generate addresses and control signals for memory/I/O transfer in either a continuous or cycle-stealing mode. This DMA feature also allows the user to perform a real-time clock function without tying up significant software or execution time, Fig. 6. The I/O structure allows for stack transfer of data directly to and from memory, again bypassing the CPU in the process, and the special I/O lines, Flags and Q, allow for software polling of external events and single-bit output control, as well as the use of the lines in combination for bit-banging serial I/O.

CDP1800 SERIES HARDWARE/SOFTWARE SUPPORT

An equally important advantage to potential CDP1804/05 users, besides architectural performance capabilities, is the availability of a complete line of compatible memory and I/O devices for efficient hardware designs, and easy-to-use tools to aid software development. The CDP1800-series is presently the



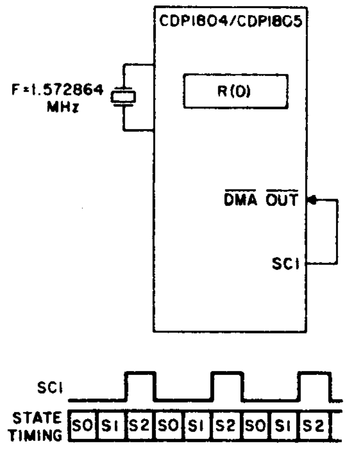
5. Basic dc timing diagram, one instruction cycle.

one not commonly found in the world of microprocessors. Once the user has mapped out a plan to assign registers to perform specific tasks, this flexibility provides him with a rich variety of software techniques for performing his required function. Flexible register assignment results in such structures as multiple program counters for quick subroutine calls, multiple stack pointers for independent data and I/O stacks, and multiple memory pointers that facilitate memory data transfers and the adaptability of the microprocessor to interpretive languages.

In addition, all CDP1800-series processors possess some unique hardware features that reduce system parts count and speed data transfer. The on-chip DMA counter allows the

broadest CMOS LSI line in the industry, and the software and debug support available allows users to reduce the significant software burden required to program a microprocessor family.

The list of support devices, Table II, for the CDP1800-series, which includes the CDP1802, CDP1804, and CDP1805 because of their compatibility, is highlighted by 12 RAM devices with capacities of from 32 to 4096 bits; ROM chips compatible with the CDP1800-series multiplexed bus structure, with user-programmed address decoders that uniquely define memory space without external decoding, and with identical pinout for system upgrade without board changes; EPROMs for system prototyping; and a wide variety of I/O devices



SCI FORCES DMA REQUEST EVERY THIRD MACHINE CYCLE. EACH DMA CYCLE INCREMENTS R(0). FOR FREQUENCY SHOWN, R(0) OVERFLOWS ONCE PER SECOND, WHICH CAN BE CHECKED BY SOFTWARE.

92CS-33389

6. Application of DMA feature to generate real-time clock reference.

Table II - Major CDP1800-Series Support Components

RAMS

	CDP1821	1K X 1
	CDP1822	256 X 4
	MWS5101	256 X 4
	CDP1823	128 X 8
	CDP1824	32 X 8
	CDP1825	1K X 4
	MWS5114	1K X 4
*	CDP1826	64 X 8

ROMS

	CDP1831	512 X 8
	CDP1832	512 X 8
	CDP1833	1K X 8
	CDP1834	1K X 8
*	CDP1835	2K X 8
*	MWS5316	2K X 8

EPROMS

	CDP18U42	256 X 8
*	MWS57U58	1K X 8

SIMPLE I/O

CDP1852	I/O PORT
CDP1853	DECODER
CDP1856	BUFFER
CDP1857	BUFFER
CDP1858	LATCH
CDP1859	LATCH
CDP1863	COUNTER
CDP1866	LATCH
CDP1867	LATCH
CDP1872	I/O PORT
CDP1873	DECODER
CDP1874	I/O PORT
CDP1875	I/O PORT

COMPLEX I/O

	CDP1851	PROG I/O
	CDP1854A	UART
	CDP1855	MDU
	CDP1869	CRT CONTROLLER
	CDP1870	CRT CONTROLLER
	CDP1871	KEYBOARD ENCODER
	CDP1876	CRT CONTROLLER
*	CDP1877	INTERRUPT CONTROLLER

* RESERVED NUMBER FOR 1981 PRODUCTION DEVICES

92CS-33383

ranging from simple buffers and latches to complex peripherals such as multiport programmable I/O, UART's, math chips, and CRT controllers.

Software support is available from products that range from simple prototyping kits to complete development systems. High-level languages, including micro FORTH, PL/M, and BASIC, are available to reduce software development time. In-circuit emulation, through the use of the "Micromonitor," provides comprehensive debug and evaluation capability, either in conjunction with a development system or for standard-alone use in the field. Finally, a complete line of CMOS single-board computers and peripheral subsystems, designed around products in the CDP1800-series family, are available to further ease the cost and the turn-around time of hardware implementation, Table III.

Table III - Major CDP1800-Series System Support

DEVELOPMENT SYSTEM
- COSMAC SYSTEM IV (CDP18S008)
  - CRT-BASED SYSTEM CONTAINING:
    - 64K MEMORY
    - DUAL FLOPPY-DISK DRIVES
    - IN-CIRCUIT EMULATION (MICROMONITOR)
    - BUILT-IN PROM PROGRAMMER
    - COMPLETE DISK OPERATING SYSTEM
      - LEVEL I, LEVEL II ASSEMBLER
      - MACROASSEMBLER
      - FULL-SCREEN EDITOR AND TEXT EDITOR
      - MICROMONITOR OPERATING SYSTEM (MOPS)
      - PROM-PROGRAMMER OPERATING SOFTWARE

SOFTWARE (OPTIONAL)
- BASIC 1 (FIXED POINT) INTERPRETER/COMPILER
- BASIC 2 (FLOATING POINT) INTERPRETER
- PLM-1800 COMPILER
- FIXED AND FLOATING POINT MATH SUBROUTINES

MICROBOARDS — SINGLE BOARD MICROCOMPUTERS
- COORDINATED SET OF COMPUTER, MEMORY AND I/O BOARDS
  - CDP18S606 - 1804/1805 EVALUATION BOARD CONTAINS:
    - 1804 CPU W/2.47 MHZ CLOCK
    - 2K BYTES RAM (FOR 1804 ROM SIMULATION)
    - 2 EACH ROM/EPROM SOCKETS
    - 2 EACH PARALLEL I/O PORTS
    - RS232C SERIAL PORT (UART)

CDP1804/05 ENHANCEMENTS

The preceding discussion has dealt with features and advantages that are common to the CDP1802, CDP1804, and CDP1805. The following sections describe specific enhancements to the CDP1802: increased memory, timer-counter implementation, standard call and return instructions, and enhanced 16-bit data manipulation, all of which should be of particular usefulness to those who are already familiar with CDP1802 capabilities and the advantages of hardware and software enhancements in their system designs.

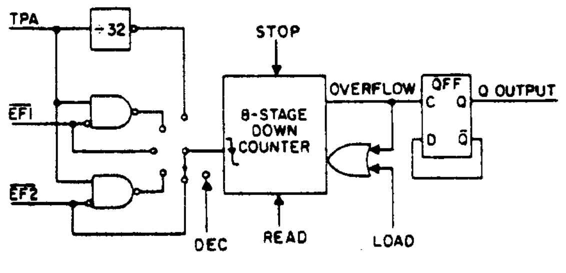
Memory

A significant feature of the upgraded CDP1804 is its memory expandability, which is not compromised as in some other single-chip microprocessors. The same 64K memory address space is available, with 2K of ROM and 64 bytes of RAM on-board. The large on-board ROM size is sufficient for many application programs, and it can hold special firmware such as the CDP18S827 floating-point binary arithmetic subroutine, or a budget interpreter such as TINY BASIC. The internal RAM provides enough locations for stack and auxiliary scratchpad usage; data-acquisition applications can take advantage of the larger memory address space for outboard RAM. Note that both internal ROM and RAM have mask-programmable address spaces, with an $\overline{EMS}$ signal provided to indicate when external memory is being addressed.

The CDP1805 has the same RAM complement and expandability features as the CDP1804. Since the device is an off-the-shelf part, its RAM is accessed through a $\overline{CE}$ input that replaces the CDP1804 $\overline{EMS}$ output. In general, the user will find the absence of ROM on the CDP1805 an advantage in many system applications because of the trend to larger and larger ROM programs in increasingly sophisticated systems. The absence of the ROM allows the user great flexibility in designing his system, and may well provide the most cost-effective approach for the majority of applications. Thus, the CDP1805 should be chosen when anticipated ROM program space exceeds that of the capacity of the CDP1804, when prototyping an experimental application with EPROM, when the application software can change, or when the user is willing to trade off cost for increased chip count.

Timer/Counter

An additional hardware feature of the CDP1804/05 devices, besides on-board memory, is an 8-stage presettable down counter, Fig. 7. This is a full-function timer/counter,



92CS-33385

7. CDP1804/CDP1805 timer/counter model.

with inputs available from an external source or a scaled internal clock, and output overflow indication through the external Q line or an internal counter interrupt request. The counter makes use of ten linked opcode instructions to perform its control functions and to program the counter for its operational modes. LOAD, READ, STOP, and DECREMENT control instructions provide manual control and allow the counter to be used to generate a programmable time delay. Three running modes can be programmed with five additional instructions; the modes are:

1. TIMER - with the input derived from the CDP1804/05 TPA pulse divided by 32, which allows its use as a time base for real-time applications.

2. EVENT COUNTER - with the input from 1 of 2 flag lines (software selectable), for single or dual-channel event counting.

3. PULSE DURATION MEASUREMENT - with the input again from a flag line, with the counter value representing the pulse width at the input. Because two inputs can be sequentially applied, comparison measurements can be made.

In conjunction with each of these modes, a tenth instruction, ETQ, can generate a programmable square wave or provide control to external peripherals by toggling the Q output on every counter overflow.

In addition to the ten timer/counter functions, six additional instructions allow arbitration between external and counter interrupts. Two instructions each are provided for interrupt masking and unmasking, while two others allow for software polling to determine the interrupt source.

Standard Call and Return

The most significant software enhancement of the CDP1804/05 devices over the CDP1802 is the addition of single CALL and RETURN instructions, which save both software and time, and free-up internal scratchpad registers for other uses, Table IV. These instructions, SCAL and SRET, are slightly different than those of other microprocessor families, primarily to allow the user to pass in-line data from the main program or calling routine to the target subroutine. Direct addressing is incorporated as with other families; however, the main program counter is exchanged with a designated scratchpad, and it is the scratchpad contents that are saved on the stack. This technique allows the original PC to point to data following the call

Table IV - Performance Comparisons-Subroutine Call and Return for CDP1802/04/05 Implemented with Various Techniques

	1802 SOFTWARE "SCRT" TECHNIQUE	1802 SOFTWARE "SEP" TECHNIQUE	1804/05 SOFTWARE SCAL / SRET INSTRUCTIONS	1804/05 SOFTWARE "SEP" TECHNIQUE
NUMBER OF MACHINE CYCLES - CALL	32	2	10	2
NUMBER OF MACHINE CYCLES - RETURN	24	4	8	4
CALL TIME (1802 @ 2.5MHz) (1804/05 @ 4 MHz)	102.4 µs	6.4 µs	20 µs	4 µs
RETURN TIME (1802 @ 2.5 MHz) (1804/05 @ 4 MHz)	83.2 µs	12.8 µs	16 µs	8 µs
NUMBER OF BYTES SOFTWARE CALL + RETURN	45	4	6	4

92CS-33391

instruction without having to exchange pointers on the stack.

The SCAL and SRET instructions eliminate the need for dedicated call and return subroutines, as required with the CDP1802, as well as the need to allocate two registers to point to these subroutines. However, as with the CDP1802, the most efficient subroutine call for small programs that do not use nested subroutines is still the SEP instruction, which uses the flexibility of register reallocation to switch program counters from main program to subroutine with one byte of code.

16-Bit Data Transfer

Sixteen-bit data transfers can be easily implemented on all CDP1800-series processors because of the presence of the 16-bit-wide scratchpad register array. In addition to 16-bit register increments and decrements, arithmetic and shift operations can be performed on 16-bit operators stored in the scratchpad registers with a sequence of CDP1804/05 software instructions. An additional hardware feature results from the CDP1804/05 I/O structure, which allows 16-bit data transfer in one machine cycle from scratchpad register R(X) to the address bus, with I/O control lines active to distinguish between this special I/O transfer and normal memory-address operations.

In addition to the above features common to the CDP1802, CDP1804, and CDP1805, the CDP1804/05 devices have four new instructions that supplement the 16-bit data-transfer operations:

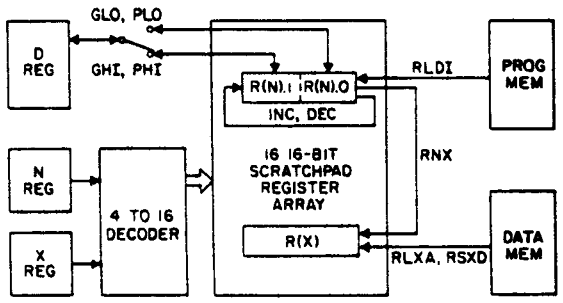
1. A register-load instruction that permits direct loading of any 16-bit scratchpad from two consecutive immediate program locations (RLDI).

2. A register-to-register transfer instruction that permits data transfer from any scratchpad to the R(X) stack pointer (RNX).

3. A register-to-memory transfer instruction that stores any register's contents into two consecutive memory locations (RSXD).

4. A memory-to-register transfer instruction that loads two consecutive memory locations into any scratchpad register (RLXA).

These additional instructions can be used to manipulate either 16-bit data or addresses on the CDP1804/05 devices because of the dual address/data capability of the scratchpad register array. Note that the accumulator (D register) is not involved in any of these new transfer operations, permitting its contents to be preserved without the need for additional manipulations. The enhanced data-transfer capability is illustrated in Fig. 8.



8. Enhanced scratchpad-register data-flow model unique to CDP1804/CDP1805. Compare this figure with figure 3.

WHY WAS I/O LEFT OUT?

The discussion of architectural details presented thus far requires mention of a significant functional block that is not contained within the CDP1804/05 device: a complement of full 8-bit I/O ports. The foremost design goal of the CDP1804/05 processors was to make them upward compatible in both software and hardware with the existing CDP1802 to ease the upgrade transfer for the system designer. This constraint eliminated any chance of adding I/O ports on-board. If the decision to add I/O at the expense of hardware compatibility had been made, then expandaability would have been compromised. Since the power of the CDP1804/05 devices lies within the 16-bit scratchpad array, a full 64K of addressable memory allows this power to be utilized to the fullest.

Some limited I/O is already present on the CDP1804/05 units in the form of the Flag and Q lines. Sixteen-bit data transfer via the address bus is another form of I/O that has already been discussed. External I/O can easily be mapped with CDP1804/05 I/O bus structure, and a wide variety of peripheral devices are available for this purpose. Finally, the high-volume customer can choose to integrate a custom I/O device that can be tailored to his specific needs and that can interface directly with the CDP1804/05 processors.

CONCLUSIONS

The CDP1800-series has been designed into a wide variety of new and existing applications, Table V; this has not been by accident. The announcement of the CDP1800 product line was made in 1976; products in the line are available in high volume. Users have taken advantage of the low power, traditional CMOS features of the line; its architecture, I/O handling capability, and ease of implementation have made the family popular in control-oriented applications. The addition of the more powerful CDP1804 and CDP1805 processors and an increasing array of memory and support chips will encourage users to develop increasingly sophisticated systems around the CDP1800 family, and should attract newcomers to this versatile, broad-based, well-established line.

Table V - CDP1800-Series Microprocessor Applications

SET BACK THERMOSTAT	BOARD TESTERS	MINI PBSX
PORTABLE AIR QUALITY MONITOR	UTILITY METER	HAND HELD MEDICAL TERMINAL
INDUSTRIAL POWER MONITOR	COIN CHANGER	SPACE FLIGHT TAPE RECORDER
ENERGY MEASUREMENT	ARTIFICIAL BREAST	MULTI LINK INTERCOM
LOAD MANAGEMENT (ENERGY MANAGEMENT)	TEST INSTRUMENTS	AUTO ANTI-THEFT
PORTABLE METER READER	IRRIGATION CONTROLLER	INDUSTRIAL CONTROLLERS (NUMERIC MACHINE CONTROL)
PORTABLE BILL CALCULATOR	SALMON COUNTER	REMOTE, HAND HELD, DATA ENTRY TERMINAL
PORTABLE NOISE LEVEL MEASUREMENT	SEISMOGRAPH	AIRBORNE FLIGHT TEST INSTRUMENTATION
AUTOMOTIVE SPARK CONTROL	ENVIRONMENTAL CONTROLS	AIRCRAFT ATTITUDES DISPLAY
AUTOMOTIVE FUEL CONTROL	REMOTE GAUGES AND METERS	FLOOD WATER MONITOR
MILITARY RADIO COMMUNICATION	NAVIGATIONAL CONTROLS	MOTOR-GENERATOR CONTROL SET
ELECTRIC CAR	DATA ACQUISITION SYSTEMS	PORTABLE OIL EXPLORATION EQUIPMENT
MOBILE TELEPHONE	CREDIT CARD VERIFIER	HOME COMPUTER
ULTRASONIC WELD INSPECTION	SONAR SYSTEMS	AIRCRAFT REMOTE CIRCUIT BREAKER CONTROL
HAND HELD MEDICAL MONITOR	FILM SPLICER	SOLAR WATER HEATER CONTROLLER
FLOW METER	SOLAR POWERED DATA LOGGER	BEER TAP CONTROLLER
HOME SECURITY (FIRE & INTRUSION)	PORTABLE GAS ANALYZER	TV GAMES
LOGGING (WELL DRILLING)	KIOSKS	LANGUAGE TRANSLATER
MISSILES	DEEP FAT FRYER	SURVEY EQUIPMENT
AUTOMOTIVE DIAGNOSTICS	VENDING MACHINE	DIVING EQUIPMENT
TELEPHONE DIALER	COPIER CONTROL	UNDERGROUND MINE TELEPHONE
LINE PRINTER CONTROLLER	DIGITAL TUNING	
PRINTING ELAPSED TIME COUNTER	VEHICLE DIAGNOSTICS	
FREQUENCY SYNTHESIZER	TV CAMERA	
BATTERY CHARGER CONTROLLEP	MOTOR CONTROL	

The NEW CDP1804A

The present 1804 design will be modified for the production version in order to provide additional flexibility in the customer's application. Two objectives will be satisfied with the design change:

1. The customer will be able to utilize the internal 1804 RAM when the internal ROM is disabled. (The present 1804 TEST mode disables both RAM and ROM).
2. The customer will be able to plug an 1804 into an existing 1802 socket and be able to utilize the 1804 as a CPU without wiring changes. (The present 1804 requires PIN 16 - $\overline{\text{EMS}}$ output - to be disconnected from the 1802 V_{CC} line, and requires that $\overline{\text{CLEAR}}$ and $\overline{\text{WAIT}}$ be tied low for use in the TEST mode).

The essential changes in the design are summarized in the Table below, followed by a list of differences from the present 1804:

$\overline{\text{CLEAR}}$	$\overline{\text{WAIT}}$	Mode	PIN 16 Function
L	L	RUN (RAM/ROM)	$\overline{\text{EMS}}$ Output
L	H	RESET	Active High Output
H	L	PAUSE	Previous State
H	H	RUN (RAM ONLY)	$\overline{\text{ME}}$ Input

1. The 1804 RUN mode has been renamed RUN(RAM/ROM) mode and is activated with $\overline{\text{CLEAR}}$ = $\overline{\text{WAIT}}$ = Low instead of High.
2. The 1804 TEST mode has been renamed RUN(RAM ONLY) mode and is activated with $\overline{\text{CLEAR}}$ = $\overline{\text{WAIT}}$ = High instead of Low.
3. In the RUN(RAM ONLY) mode, PIN 16 becomes an input.
4. This input is used to select (on an active low level) or deselect the internal 1804 RAM.
5. In this mode, the pre-programmed address mapping for the RAM is eliminated.
6. In the RESET mode, PIN 16 is an active high output.
7. In the PAUSE mode, PIN 16 retains the function of the previous state - if previously in the RUN(RAM/ROM) mode, PIN 16 will remain an output, and if previously in the RUN(RAM ONLY) mode, PIN 16 will remain an input when placed in the PAUSE mode.

The above details will be reflected in the new 1804A data sheet and will be expanded upon in an upcoming Application Note.

In addition, a new 1805 version will be available. This device will be a ROMless version of the 1804 and have all the features described in the 1804 data sheet and modified in this article, except for the fact that there will be no RUN(RAM/ROM) mode, and PIN 16 will always be an input pin.

For more details contact Joe Paradise, X7352.

The EXPANDED 1804 Instruction Set

INTRODUCTION

The 1804 and 1805 microcomputers include a number of new instructions not found in the present 1802 instruction set, which are designed to increase the versatility of the machine and reduce the amount of code needed to write software programs. The 22 new instructions all use a linked "68" opcode, the only free opcode in the present 1802 set. Thus all new instructions require a format which includes the following: 68 (linked opcode) XX (Instruction opcode) YY, ZZ (one or two immediate bytes if required by the instruction). All new instructions also require a minimum of 3 machine cycles (2 SØ cycles and 1 or more S1 cycles).

The 22 new instructions can be divided into the following 4 groups or classifications:

1. Register Instructions - RLDI, RLXA, RSXD, RNX
2. Counter Control Instructions - LDC, GEC, STPC, DTC, STM, SCM1, SCM2, SPM1, SPM2, ETQ
3. Interrupt Control Instructions - XIE, XID, CIE, CID, BCI, BXI
4. Subroutine Instructions - SCAL, SRET

Each group is treated separately in the following discussion.

Register Instructions

MNEMONIC	Instruction	Opcode	Mach. Cyc.
RLDI	Register Load Immediate	68CN (2 imm.bytes)	5
RLXA	Register Load via X and Advance	686N	5
RSXD	Register Store via X & Decrement	68AN	5
RNX	Register N to Register X copy	68BN	4

These additional register instructions add register to register and register to memory transfer capability to the 1804, and reduce coding and machine cycles when compared to equivalent 1802 code. Note that all 4 instructions destroy the previous contents of the T register due to internal data manipulation.

RLDI

The register load immediate instruction provides a means to load one of the 16 16-bit scratchpad registers with one instruction, and without using the D register in the manipulation. The RLDI instruction saves 2 bytes of code and 3 machine cycles over equivalent 1802 instructions, except where upper byte or lower byte register data is loaded into several registers with successive PHI or PLO instructions. Since the D register is not used to perform the 16 bit load, its contents do not have to be saved as with equivalent 1802 instructions.

The instruction format is 68 CN YY ZZ, where N is the chosen scratchpad register (0-F), YY is the R(N).1 immediate data byte, and ZZ is the R(N).0 immediate data byte.

The equivalent 1802 code is as follows:

```
        LDI YY )
        PHI RN )   6 bytes of code
        LDI ZZ )   8 machine cycles
        PLO RN )
```

RLXA

The register load via X and advance instruction allows direct memory to register transfer while by-passing the D register. It is equivalent to the LDXA instruction, except that it is the scratchpad rather than the D register which is being loaded. It can be used in a stack operation where X points to an unused location at the top of the stack. In this mode, an IRX instruction is issued, followed by a RLXA. The RLXA instruction pops the first byte at the top of stack and loads it into R(N).1; R(X) advances; the second byte is popped and loaded into R(N).0; R(X) then advances to point to the next data byte on the stack. Note that since this instruction always operates on 2 data bytes, conventional single byte memory to register transfers require careful programming to keep track of R(X) and to insure that register data is not destroyed. Thus this instruction is most useful when a 16 bit address has been stored on the stack and is to be loaded into a scratchpad register, or when a 16 bit data word needs to be manipulated.

The instruction format is 686N, where N is the loaded scratchpad register (0-F). The equivalent 1802 code is as follows:

```
LDXA     )
PH1 RN   )     4 bytes of code
LDXA     )     8 machine cycles
PLO RN   )
```

RSXD

The register store via X and decrement instruction allows direct register to memory transfer while by-passing the D register. It is equivalent to the STXD instruction, except that it is the scratchpad rather than the D register which is the data source. The instruction complements the RLXA instruction described previously. In a stack operation, X initially points to an unused location at the bottom of the stack. An RSXD instruction pushes R(N).0 data onto the stack; R(X) decrements; the second byte, R(N).1, is pushed onto the stack; R(X) then decrements to point to an unused location at the top of the stack. As with the RLXA instruction, it is always 16 bit data which is being manipulated, so it is usually a 16 bit address or 16 bit data word which is used with this instruction.

The instruction format is 68AN, where N is the scratchpad register (0-F) used in the data transfer.

The equivalent 1802 code is as follows:

```
GLO RN )
STXD   )     4 bytes of code
GH1 RN )     8 machine cycles
STXD   )
```

RNX

The register N to register X copy instruction allows register to register transfer without using the D register. Although in many cases the same operation can be performed with a SEX instruction, it can be useful if the present value of R(N) needs to be stored as a memory pointer or stack pointer address before being modified by a subsequent operation. It can also be used when executing an output instruction, in cases where using a SEX instruction would result in an undesirable increment of R(N).

The instruction format is 68BN, where N is the scratchpad register (0-F) used in the data transfer.

The equivalent 1802 code is as follows:

```
                GHI RN  )
                PHI RX  )     4 bytes of code
                GLO RN  )     8 machine cycles
                PLO RX  )
```

Counter Control Instructions

MNEMONIC	Instruction	Opcode	Machine Cycles
LDC	Load counter	6806	3
GEC	Get counter	6808	3
STPC	Stop counter	6800	3
DTC	Decrement counter	6801	3
STM	Set timer mode and start	6807	3
SCM1	Set counter mode 1 and start	6805	3
SCM2	Set counter mode 2 and start	6803	3
SPM1	Set pulse width mode 1, start	6804	3
SPM2	Set pulse width mode 2, start	6802	3
ETQ	Enable toggle Q	6809	3

The 10 1804 counter control instructions allow the 8-bit presettable down counter on the chip to be used in a variety of modes, with several sources of clock inputs, and various ways of using information processed by the counter circuit.

LDC
The load counter instruction sets the presettable down counter with an 8 bit count from the D register. On every clock transition, the counter will decrement this loaded value by 1, and will set an interrupt flip-flop when it has decremented

to 0. If the counter is preset to $(00)_{16}$ a full 256 counts will occur. During a load instruction to the counter, the counter and its buffer register are loaded, the prescaler reset, the mode reset and any previous interrupts cleared.

GEC

The get counter instruction loads the present value of the counter into the D register. The counter can be read at any time, and a counter read does not affect any counter operation.

STPC

The stop counter instruction gates off the clock source to the counter and freezes it at its present count. The counter mode is also cleared at this time. It is not necessary to stop the counter to do a counter read.

DTC

The decrement counter instruction decrements the present counter count by 1. It enables the user to count in software, but the instruction should be used only after the mode has been cleared by a stop counter instruction. Its advantage over a decrement register N instruction is that a scratchpad register is saved, and GLO BZ instructions are not required to test for 0. Since the D register is not used, its contents do not have to be saved as when performing this manipulation with the 1802.

STM

The set timer mode and start instruction designates the TPA pulse divided by the internal ÷ 32 prescaler as the counter clock source. When the STM instruction is issued, the prescaler will be decremented on every subsequent low-to-high transition of TPA. The clock is then decremented at a rate equal to f/256, where f is the 1804 crystal or RC clock frequency. When the counter has decremented to 0 and the interrupt request is latched, the counter will return to its initial value and continue to decrement on subsequent counts. Note that a start instruction must be re-issued after a LDC or STPC instruction.

SCM1

The set counter mode 1 and start instruction designates the $\overline{\text{EF1}}$ input as the counter clock source. This by-passes the prescaler and causes the counter to decrement on every high-to-low transition of $\overline{EF1}$. This instruction allows the counter to be used as an event counter, or as a real-time counter with a clock source other than the crystal clock ÷ 256.

<u>SCM2</u>
The set counter mode 2 and start instruction designates the $\overline{EF2}$ input as the counte clock source, and causes the counter to decrement on every high-to-low transition of $\overline{EF2}$.

<u>SPM1</u>
The set pulse width mode 1 and start instruction allows the counter to perform pulse width measurements. In this mode the pulse to be measured is applied to the $\overline{EF1}$ input. The TPA pulse is designated as the counter clock source. Each low-to-high transition of TPA decrements the counter if the input signal at $\overline{EF1}$ is low. On the transition of this signal to the positive state, the count is stopped, the mode is cleared, and the interrupt request latched. If the counter decrements to 0 while the input is low, interrupt will also be set, but the counter will continue.

<u>SPM2</u>
The set pulse width mode 2 and start instruction is the same as the SPM1 instruction described above, except that the pulse to be measured is applied to the $\overline{EF2}$ input. Note that for any of these modes which use $\overline{EF1}$ and $\overline{EF2}$ terminals as inputs, $\overline{EF1}$ and $\overline{EF2}$ can still be tested as flags for branch instructions.

<u>ETQ</u>
The enable toggle Q instruction connects the Q-line flip-flop to the output of the counter. Each time the counter decrements from 01 to its next value, the Q-line changes state. The counter clock source is determined by the counter mode, and thus can be used in timer, event counter, or pulse width measurement modes. The ETQ instruction is cleared by an LDC instruction, a CPU reset, or a BCI instruction with CI = 1.

Interrupt Control Instructions

MNEMONIC	Instruction	Opcode	Machine Cycles
XIE	External interrupt enable	680A	3
XID	External interrupt disable	680B	3
CIE	Counter interrupt enable	680C	3
CID	Counter interrupt disable	680D	3
BCI	Short branch on counter interrupt	683E	3
BXI	Short branch on external interrupt	683F	3

These instructions provide a means for identifying and controlling interrupts generated externally, as with the 1802, or from the counter circuitry added in the 1804.

XIE

The external interrupt enable instruction sets an external interrupt enable flip-flop in the 1804. If a return instruction has set the common interrupt enable flip-flop, an external interrupt request will be immediately serviced by the CPU. Note that a hardware reset will also set both flip-flops and accomplish the same objective. Both flip-flops must be set for an interrupt to take place. The external interrupt, as in the 1802, is not latched by the CPU, and must remain active low until the interrupt is serviced.

XID

The external interrupt disable instruction resets the external interrupt enable flip-flop in the 1804, and prevents an external interrupt from being serviced.

CIE

The counter interrupt enable instruction sets a counter interrupt enable flip-flop in the 1804. If a return instruction has set the common interrupt enable flip-flop, a counter interrupt request will be immediately serviced by the CPU. Note that a hardware reset will also set both flip-flops and accomplish the same objective. Both flip-flops must be set for an interrupt to take place. The counter interrupt request is latched, and will remain active until the request is serviced, disabled, or reset by a hardware CPU reset signal.

CID

The counter interrupt disable instruction resets the counter interrupt enable flip-flop in the 1804, and prevents a counter interrupt from being serviced. Note that the CID instruction does not reset the counter interrupt request.

BCI

A short branch on counter interrupt instruction provides a means of identifying and prioritizing the interrupt source : if both the external interrupt and counter interrupt flip-flops are enabled, the BCI instruction can be placed in the user's interrupt service routine. A page branch will then be performed if the counter was the source of the interrupt. The BCI instruction can also be used if program interrupt is not desired, and a polling technique is used instead. If a counter interrupt request is pending, the short branch will be taken, regardless of the state of any of the interrupt enable flip-flops.

BXI

The short branch on external interrupt instruction provides a means of identifying an external interrupt if both external interrupt and counter interrupt flip-flops are enabled, as explained above. Note, however, that since the external interrupt request is not latched, it must remain low until the short branch operation is performed. If the BXI instruction is used for software polling, the $\overline{\text{INT}}$ pin becomes a flag line similar to $\overline{\text{EF1}}$ - $\overline{\text{EF4}}$, but with the exception that only a short branch can be performed.

Subroutine Instructions

MNEMONIC	Instruction	Opcode	Machine Cycles
SCAL	Standard call	688N (2 immed.bytes)	10
SRET	Standard return	689N	8

These two instructions implement the function of the standard call and return technique described in MPM-201B, but without the need to tie up the D register and several scratchpad registers. These instructions are similar in nature to those found in other processors, except that a technique is provided to pass parameters from the main program to the subroutine without additional instructions. Note that both instructions destroy the previous contents of the T register.

program to the subroutine without additional instructions. Note that both instructions destroy the previous contents of the T register.

SCAL

The standard call instruction allows the user to call a subroutine anywhere in memory, while saving the present program counter in a scratchpad register for parameter passing during the subroutine execution. In operation, a 68 8N YY ZZ instruction is issued, where N is the scratchpad register chosen to hold the main program counter, and YY, ZZ are the high and low byte, respectively, of the subroutine call address. The present contents of the selected scratchpad register N are pushed onto the stack (R(N).0 and then R(N).1) as designated by stack pointer R(X). This frees this N register to be used to store the main program counter. After this is accomplished, R(P) is loaded with the instruction's immediate bytes, and execution of the subroutine begins. During subroutine execution, R(N) will point back to any list of inline parameters, and they can be passed to the subroutine with LDA RN instructions. This execution will load the successive data bytes into the D register for use by the subroutine and increment R(N) up to the proper address for a return operation. Subroutines can be nested with this same instruction; R(N) contents (which can be same R(N) used by main subroutine) are pushed onto the stack for retrieval when the subroutine is finished.

SRET

The standard return instruction allows the user to return from a subroutine back to the main program, or from a nested subroutine back to the calling subroutine. In operation, a 689N instruction is issued, where N is the scratchpad register that is holding the main program counter. The contents of R(N) are restored to R(P), and the contents of the stack are popped and placed in R(N), restoring its original contents.

The attached example illustrates the use of these subroutine call and return instructions in a practical program:

Example

Assume a subroutine is to be written to add a main program inline parameter to a memory location's contents:

Solution

Address	Opcode	Mnemonic	Comments
0000 | 004F	71 | C4	DIS | NOP	Main Program 1
0050 0051 0052 0053	68 84 01 80	SCAL R4 0180	Calling routine
0054	23		Inline data
0055 | 017F	C4 | C4	NOP | NOP	Main program 2
0180 0181 0182 0183 0184 0185	E5 44 F4 55 68 94	SEX R5 LDA R4 ADD STR R5 ← SEX R2 SRET R4	Subroutine

Program execution will proceed from main program 1 to subroutine to main program 2. The register table shown below illustrates the data manipulation used in the process.

REGISTER AND MEMORY CONTENTS DURING SUBROUTINE EXECUTION

		Before Call		After Call		After STR R5		After Return	
Function	Register	R(N)	M(R(N))	R(N)	M(R(N))	R(N)	M(R(N))	R(N)	M(R(N))
Stack Pointer	R2	02FF	00	02FD	00	02FD	00	02FF	CD
Main Prog. Ctr.	R3	0050	68	0180	E5	0184	68	0055	C4
Inline Register	R4	ABCD	00	0054	23	0055	C4	ABCD	00
Mem Pointer	R5	0335	15	0335	15	0335	38	0335	38

For more information, contact Joe Paradise, X7352.

A Brief 1804 - 8048 Architecture Comparison

Characteristic	1804	8048/8049
On-board ROM	2K Bytes	1K/2K Bytes
Working Registers	16 16-bit general purpose registers for PC, SP, index registers, data	2 banks of 8 general purpose registers - select only one at a time
On-board RAM	64 Bytes	64/128 Bytes - but includes working registers & stack area
I/O Lines	Q output port 4 EF inputs $\overline{\text{INT}}$ input	2 quasi bi-directional 8 bit ports, 1 true 8 bit port, 2 test inputs, $\overline{\text{INT}}$ input
On-board Timer	8 bit down-counter timer/event/pulse measurement functions	8 bit up-counter timer/event functions
Directly Expandable External Memory	64K total program AND/OR data memory (1804 does not distinguish)	4K total program memory 320/384 bytes total data memory
External Memory Addressing Machine	Std. mux address bus or single pin $\overline{\text{EMS}}$ output. Uses common 8 bit data bus.	4 bits of port 2 used. For high order address (lower nibble) 8 bit bus port used for mux addr/data bus. (12 of 24 avail. port pins tied up for ext. memory)
Test External Mem. Mode	$\overline{\text{CLEAR}}$ = L, $\overline{\text{WAIT}}$ = L	EA = H
Slow External Memory Provision	$\overline{\text{WAIT}}$ line to stop internal clock	No provision
External Bus Configuration	Multiplexed address bus	Multiplexed address/ data bus
External I/O Capability	3 N-lines available for 1 or 2 level I/O to external 8 bit ports	4 internal I/O pins available for special I/O chip expander- 4 4-bit ports addl.
Clock Frequency (5V)	4MHz (4MHz xtal)	2 MHz (6MHz xtal)

A Brief 1804 - 8048 Architecture Comparison (Cont'd)

Characteristics	1804	8048/8049
Cycle Time Range	2 μs - ∞ (static logic)	2.5 - 15 μs (dynamic logic)
Machine Cycle	8 pulses/cycle. Min 2 cycle/instruction	5 pulses/cycle. 1 or 2 cycles/instruction
DMA Capability	DMA in/out request lines; dedicated register for DMA xfer	No internal or external DMA capability
Interrupt Capability	External or timer interrupt; 4 EF flags for polling	External or timer interrupt; 2 testable inputs (T_0, T_1)
Interrupt Mechanism	P+ X REG stored in T R(1) becomes PC R(2) becomes stack PTR	PC+status stored on stack, PGM vectors to LOC 3 of PGM memory (treated as subroutine call)
Interrupt Acknowledge Signal	Decode from S_0, S_1 lines	Must use I/O port pin
Interrupt Code Restrictions	Anywhere in 64K memory space	lower 2K of memory only
Addressing Data Memory	16 16-bit pointers available	2 8-bit pointers available
Stack Pointer, Area	Software designated, can reside in internal or external RAM, any length (16 bit SP)	Dedicated internal data memory, 8 levels only (3 bit SP)
Program Counter	16 bits - software designated, 64K bytes address capability	Dedicated 12 bits, but only address instructions within 2K byte space
Testable Inputs/Registers	Accumulator, EF inputs, carry, Q output	Accumulator, accumulator bits, carry, T_0,T_1,F_0,F_1
Conditional Jumps	Short or long branch instructions for paged or full addressing	Branch only within current page of program memory
Software Instructions (unique to either type)	Subtract, conditional long branch, skip	Jump on accum. bit, decimal adjust. swap accum. nibbles

For more information on this or related subjects contact Joe Paradise, X7352.

RCA Solid State February 1981

CDP1805 Operation in the CDP18S020 Evaluation Kit

Because of the upwards software and hardware compatibility features of the CDP1805 to the CDP1802 microprocessor, the CDP1805 can, in most cases, be plugged directly into an existing CDP1802 socket and emulate CDP1802 operation. This feature conveniently allows the CDP1805 to be evaluated in the CDP18S020 Evaluation Kit. However, 2 relatively minor problems do occur which may require minor modifications for proper use.

The first problem concerns the oscillator section. The Schmitt Trigger used in the oscillator section is designed to operate at 15 pF and is sensitive to any additional capacitive loading at pin 39 (XTAL). Bare board capacitance measures 7 pF at pin 39 and care must be taken when additional wiring is added at this point or when the optional crystal capacitor (C7) is used. A simple solution is to remove the optional capacitors or replace them with values less than 8 pF. If frequency stability is critical and/or capacitive loading at pin 39 is unavoidable, lowering the feedback resistor (R14) to 1 megohm will correct the problem.

The second problem occurs when the single-step mode is used. The CPU responds to a low on the $\overline{\text{WAIT}}$ line on the leading edge of either TPA or TPB. However, the leading edge of TPB also closes the window created in the control circuitry to sample a single step signal. The CDP1805, in effect, creates its own window and the control circuitry will not allow single step operation. Removing the TPB signal at U15 pin 13 and then grounding this point will allow single-step operation.

Interrupt Control Logic in the CDP1805

With the addition of the timer-counter in the 1805, additional internal logic is required for the counter interrupt, and arbitration between external and counter interrupt. This logic is shown in Figure 1. A number of somewhat subtle points need to be made about the new logic, which will illustrate some new interrupt and branch features found in the 1805:

1. All interrupts are enabled with reset.
2. A pending counter interrupt is cleared with reset.
3. Simple enable and disable instructions - XIE, XID, CIE, CID - are available for individual control of external and counter interrupts.
4. The external interrupt can not be polled with a BXI instruction - however, this interrupt signal is not internally latched.
5. A counter underflow or termination of a pulse in either PULSE DURATION MODE 1 or 2 will latch a counter interrupt request.
6. This request can be polled with a BCI instruction.
7. Servicing a counter interrupt request does not automatically disable the interrupt - when an interrupt occurs, software must arbitrate between external and counter interrupt requests.
8. This can be performed by a BCI instruction, which resets the counter interrupt latch.
9. If only the counter interrupt is used, the interrupt routine must still contain either a BCI or LDC instruction to reset the latch.

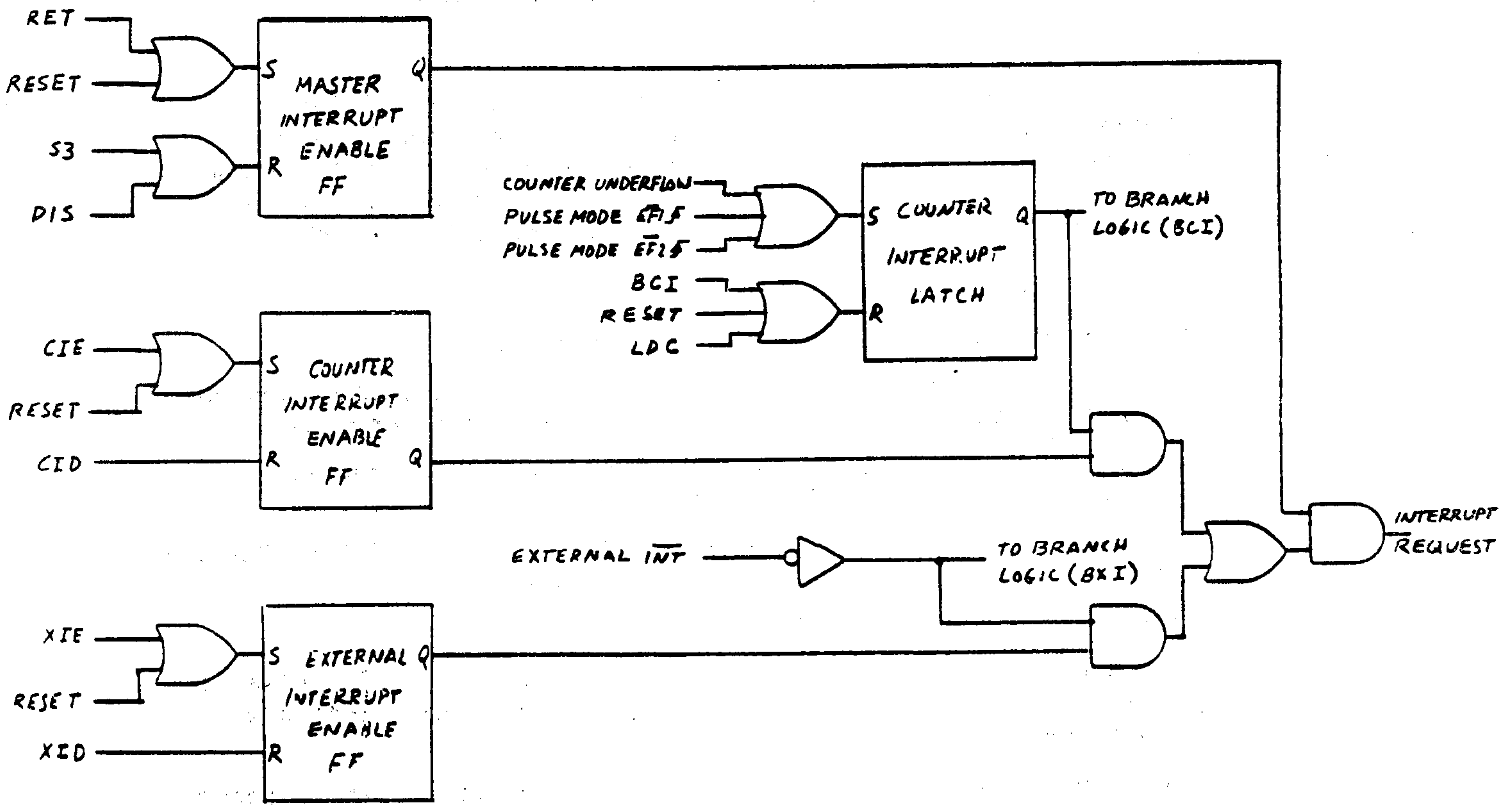
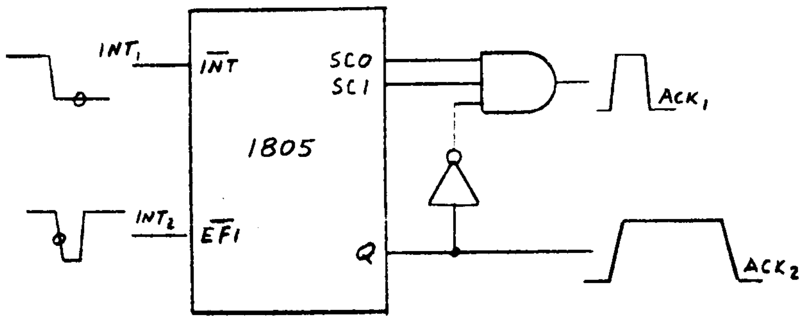
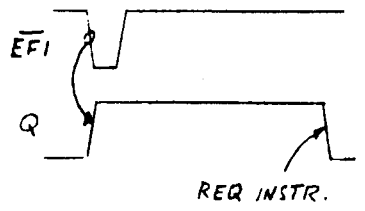


Fig. 1 - CDP1805 Interrupt Control Logic

Adding a Second Interrupt to the CDP1805

The 1805 timer-counter can be configured to provide a second interrupt input in real-time control applications. With the proper instruction sequence, an edge-sensitive latched input is provided, complete with interrupt acknowledge for peripheral handshaking.





CDP1805 INTERRUPT EXPANSION

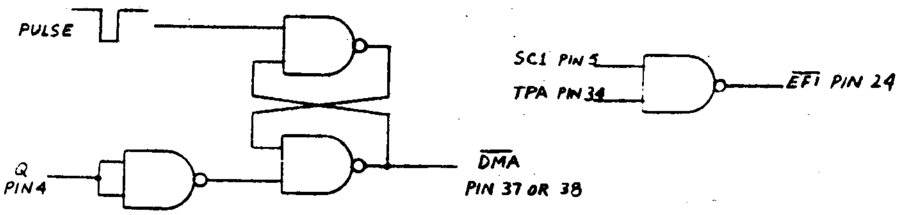
To realize this feature, advantage is taken of the internal interrupt provided by the timer-counter in the COUNTER MODE. With the timer-counter preloaded to 01, a single transition on $\overline{\text{EF1}}$ or $\overline{\text{EF2}}$ will cause a counter underflow and generate a counter interrupt. Arbitration between external and counter interrupt takes place in the interrupt service routine, with the use of the BCI instruction, which will allow branching to the counter interrupt service routine and also reset the counter interrupt.

If the ETQ instruction is used, an acknowledge signal can be provided via the Q output. The Q line will go high on the high-to-low transition of $\overline{\text{EF1}}$, and will be reset during the counter interrupt service routine. Such a signal can be fed to a peripheral as a DATA READY indicator, and provides more information than a decoded state code acknowledge signal.

```
!M
0000 ;
0000 ;                        ..SAMPLE PROGRAM TO DEMO USE OF 1805
0000 ;                        ..COUNTER-TIMER TO CREATE A 2ND
0000 ;                        ..INTR INPUT WITH HANDSHAKE
0000 ;
0000 ;           STACK EQU 0050H
0000 ;           MAINST EQU 0060H
0000 ;
0000 680B;       BEGIN    XID                  ..DISABLE XTNL INTR
0002 680D;                CID                  ..DISABLE CNTR INTR
0004 68C30009;            RLDI R3, INIT
0008 D3;                  SEP R3               ..SET R3 AS MAIN PC
0009 ;
0009 68C10026;   INIT     RLDI R1, INT         ..SET INTERRUPT VECTOR
000D 68C20050;            RLDI R2, STACK       ..INTERRUPT STACK
0011 68C40060;            RLDI R4, MAINST      ..MAIN STACK
0015 E4;                  SEX R4
0016 ;
0016 F801;       INITC    LDI 01               ..COUNTER WILL DETECT
0018 6806;                LDC                  ..1 TRANSITION
001A 6805;                SCM1                 ..SET COUNTER MODE
001C 6809;                ETQ                  ..ENABLE TOGGLE Q
001E 680A;                XIE                  ..ENABLE XTNL INTR
0020 680C;                CIE                  ..ENABLE CNTR INTR
0022 ;
0022 00;         WAIT     IDL                  ..THIS IS BODY OF MAIN
0023 00;                  IDL                  ..PROGRAM IN A REAL
0024 00;                  IDL                  ..ROUTINE
0025 ;
0025 70;         INTRET   RET                  ..RETURN TO MAIN
0026 ;
0026 22;         INT      DEC R2               ..INT VECTORS HERE
0027 78;                  SAVE                 ..SAVE P AND X
0028 683E;                BCI CINT             ..WHICH INTERRUPT?
002A 30;                  DC A.0(CINT)         ..FUDGE FOR ASSEMBLER
002B E4;                  SEX R4               ..THIS WOULD BE ROUTINE
002C 61;                  OUT 1                ..FOR EXTERNAL INT IN
002D E2;                  SEX R2               ..A REAL PROGRAM
002E 3025;                BR INTRET            ..END ROUTINE
0030 ;
0030 7A;         CINT     REQ                  ..DATA READY TO PERIPH
0031 E4;                  SEX R4               ..SOME GENERAL OUTPUT
0032 62;                  OUT 2                ..ROUTINE TO PERIP
0033 E2;                  SEX R2               ..DEVICE
0034 6809;                ETQ                  ..RESET INSTRUCTION
0036 3025;                BR INTRET            ..END ROUTINE
0000
```

Using the CDP1805 Timer-Control for Multiple DMA Transfers

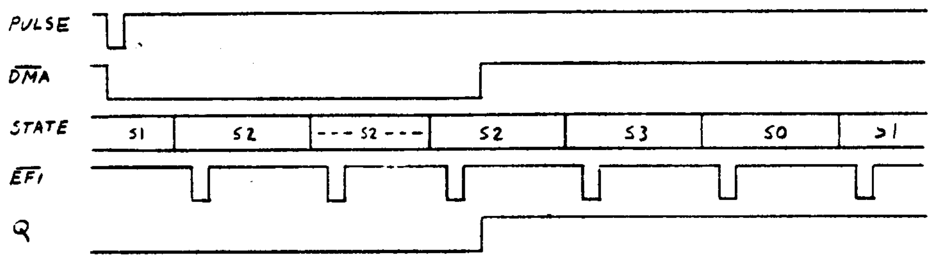
The 1805 timer-counter, along with a single inexpensive CD4011, can be used to count a desired number of DMA transfers with the application of a single short pulse. The software initialization and maintenance is also modest, providing a good alternative to equivalent hardware techniques used in 1802-based designs.



HARDWARE CONNECTIONS TO CDP1805

The hardware shown above consists of a simple RS latch, inverter, and gate, implemented with the CD4011. Initially with the Q signal low, the latch accepts an external pulse and forces the $\overline{\text{DMA}}$ line low. After execution of the present instruction, a DMA S2 cycle begins, forcing SC1 high, and gating TPA to EF1. With the timer-counter set to COUNTER MODE 1, the counter will decrement on every TPA, which will occur once per DMA cycle. With the counter initially loaded with the desired number of DMA transfers +1, and with the ETQ instruction enabled, $\overline{\text{DMA}}$ will remain low until the counter underflows. The underflow sets Q high, externally resets the RS latch, and reloads the counter to its initial value. The underflow also generates a counter interrupt, which is detected by the CPU, and forces an S3 cycle. The S3 cycle causes the counter to decrement once, because the simple gating used does not uniquely decode a DMA cycle. Thus the counter is loaded with the number of transfers +1 if the hardware scheme shown is used.

The interrupt service routine resets the Q line and clears the counter interrupt request. When the interrupt routine is complete, the circuitry is reinitialized for another asynchronous DMA transfer.



TIMING DIAGRAM

The attached ASMS assembly listing illustrates the minimal software needed for program execution.

```
>!M
0000 ;
0000 ;                         ..SAMPLE PROGRAM TO DEMONSTRATE 1805 COUNTER
0000 ;                         ..USED TO COUNT DMA TRANSFERS TO OR FROM MEM
0000 ;                         ..SINGLE 4011 IS ONLY EXTERNAL HW REQD
0000 ;                         ..TRANSFER CONTROLLED BY SINGLE EXTNL PULSE
0000 ;
0000 ;             DMAMEM EQU 0100H
0000 ;             DCOUNT EQU 15H
0000 ;             CSTACK EQU 0050H
0000 ;             MAINST EQU 0060H
0000 ;
0000 68C30005;     BEGIN   RLDI R3, INIT
0004 D3;                   SEP R3            ..SET R3 AS MAIN PC
0005 ;
0005 68C00100;     INIT    RLDI R0, DMAMEM ..MEM BLOCK TO BE TRANS
0009 68C10024;             RLDI R1, CINT   ..INTERRUPT VECTOR
000D 68C20050;             RLDI R2, CSTACK ..INTERRUPT STACK
0011 68C40060;             RLDI R4, MAINST ..MAIN STACK
0015 E4;                   SEX R4
0016 ;
0016 F815;         INITC   LDI DCOUNT        ..NO. OF DMA TRANSFER BYTES
0018 6806;                 LDC               ..LOAD COUNTER
001A 6805;                 SCM1              ..SET COUNTER MODE 1
001C 6809;                 ETQ               ..Q TOGGLES WHEN FINISHED
001E ;
001E 00;           WAIT    IDL               ..WAIT FOR DMA TO COME IN
001F 00;                   IDL               ..GET DCOUNT TRANSFERS
0020 00;                   IDL               ..FOR EVERY DMA PULSE
0021 ;
0021 6809;         INITQ   ETQ               ..ETQ WAS RESET BY BCI
0023 70;                   RET               ..RETURN TO MAIN PROGRAM
0024 ;
0024 22;           CINT    DEC R2            ..INTERRUPT VECTORS HERE
0025 78;                   SAVE              ..SAVE MAIN PC
0026 ;
0026 7A;                   REQ               ..RESET Q
0027 683E;                 BCI INITQ         ..RQD TO RESET COUNTER INT
0029 21;                   DC A.0(INITQ)     ..FUDGE FOR ASSEMBLER
0000
```

Multiple DMA Transfer Under Software Control

The ubiquitous 1805 timer-counter, helped along with a simple inverter, can play another role in DMA transfers by giving the user software control over the operation. The circuit in Figure 1 shows an inverted Q signal being fed back to both the $\overline{DMA}$ and $\overline{EF1}$ pins. With the counter properly initialized and set to PULSE MODE 1, an SEQ instruction will begin the DMA transfer. The counter will decrement with each TPA pulse while $\overline{EF1}$ is low. When an underflow occurs, the ETQ instruction will cause Q to go low, forcing $\overline{EF1}$ high and terminating the PULSE MODE.

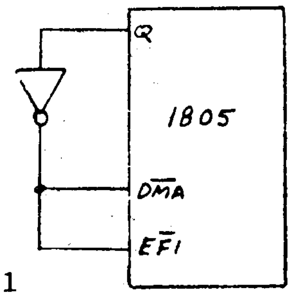


Figure 1

The simple software required to execute this operation is attached. It is set up in the form of a subroutine call, which also illustrates the new SCAL and SRET instructions of the 1805. A unique feature of SCAL, shown in the routine, is that constant parameters can be passed to the subroutine from the main program. In this case, the DMA routine is called with the number of bytes to be transferred set by the byte succeeding the SCAL instruction. This is possible because the main program counter is stored in a designated register (R5) rather than on a stack. The LDA instruction in the subroutine passes the value (DCOUNT) to the subroutine and increments R5 to the next byte in the main program.

Note that the subroutine sequence shown is required for each DMA transfer. The SPM1 mode is terminated when a low-to-high EF1 transition occurs, requiring a reload of the counter and resetting of the desired mode. The LDC instruction also resets ETQ, so this instruction must also be repeated.

```
>!M
0000 ;
0000 ;                 ..SAMPLE PROGRAM TO DEMONSTRATE 1805 COUNTER
0000 ;                 ..USED TO COUNT DMA TRANS TO OR FROM MEM
0000 ;                 ..SINGLE INVERTER IS ONLY EXTERNAL HW REQD
0000 ;                 ..TRANSFER CONTROLLED BY SOFTWARE INIT
0000 ;
0000 ;         DMAMEM EQU 0100H
0000 ;         DCOUNT EQU 15H
0000 ;         SUB1 EQU 0200H
0000 ;         SUB3 EQU 0300H
0000 ;         MAINST EQU 0050H
0000 ;
0000 68C30005; BEGIN   RLDI R3, INIT
0004 D3;               SEP R3          ..SET R3 AS MAIN PC
0005 ;
0005 68C00100; INIT    RLDI R0, DMAMEM ..MEM BLOCK TO BE TRANS
0009 68C40050;         RLDI R4, MAINST ..MAIN STACK
000D E4;               SEX R4
000E 680D;             CID             ..DISABLE COUNTER INT
0010 ;
0010 68850200; MAIN1   SCAL R5, SUB1   ..CALL SOME SUBROUTINE
0014 ;
0014 6885001D; MAIN2   SCAL R5, SUB2   ..CALL DMA ROUTINE
0018 15;               DC (DCOUNT)     ..R5 POINTS TO CTR LOAD VAL
0019 ;
0019 68850300; MAIN3   SCAL R5, SUB3   ..CALL SOME SUBROUTINE
001D ;
001D 45;       SUB2    LDA R5          ..PASS DMA LOAD VAL TO SUB2
001E 6806;             LDC             ..LOAD COUNTER
0020 6804;             SPM1            ..SET PULSE WIDTH MODE 1
0022 6809;             ETQ             ..ENABLE TOGGLE Q
0024 7B;               SEQ             ..INIT DMA TRANS OF DCOUNT
0025 6895;             SRET R5         ..RETURN TO MAIN PROG
0000
```

Understanding The CDP1851 Programmable I/O

Overview

The CDP1851 is a general purpose programmable I/O device, having 20 I/O pins which may be used in several different modes of operation. (See Table 2).

The I/O lines are grouped into two sections, A and B, each having 10 lines; 8 data and 2 handshaking lines (ready and strobe).

In essence, the CPU programs the PIO by asserting the proper address (if memory mapped) or the proper N-line code (if I/O mapped) on the PIO control lines, and outputs a sequence of control bytes on the data bus to the control register of the PIO. The control bytes contain information to define port mode, interrupt enable/disable, I/O bit assignment, bit masking, etc. (see Codes A-P, Table 1).

The CPU transfers data bytes to and from each port by asserting codes S, T, U or V, given in Table 1. Modes may be combined so that their functional definition can be tailored to almost any I/O requirement.

Modes of Operation

a. Normal Input/Output Mode

Ports A and B can be separately programmed to be 8 bit input or output parts with handshaking control lines, ready and strobe.

b. Bi-Directional Mode

Port A can be programmed to be a bi-directional port. This configuration provides a means for communicating with a peripheral device or structure on a single 8 bit bus for both transmitting and receiving data. Handshaking signals are provided to maintain proper bus flow discipline. The handshaking lines for Port A are used in the normal manner for input control. The handshaking lines for Port B are used by Port A for output control; consequently, Port B I/O lines must be in the bit programmable mode where handshaking is not used.

c. Bit-programmable Mode

Ports A and B can be separately bit programmed so that each individual line can be designated as an input or output line.

An additional feature of the bit programmable mode is that the four handshaking lines, A RDY, A STROBE, B RDY, and B STROBE can be individually programmed as input or output lines (see Code K, Table 1).

In the input, output, and bi-directional modes the STROBE lines give the PIO the trigger signals needed to generate interrupt requests. However, in the bit programmable mode the handshake lines are not used to carry strobe and ready signals, but carry I/O data if programmed to do so.

Interrupts

Interrupt requests are generated differently depending on the port mode.

a. Input and Output Modes

The falling edge of the strobe pulse from the peripheral device sets the $\overline{INT}$ line, and the rising edge of TPB, during the requested read or write operation resets the $\overline{INT}$ signal.

b. Bi-Directional Mode (Port A only)

Set and reset of interrupt requests are done as explained in the input and output modes. However, since the $\overline{A\ INT}$ line is used for both input and output interrupts the CPU must read the status register to determine what condition caused the interrupt request.

c. Bit-Programmable Mode

Interrupt requests can be generated by programming the PIO to re-act to specified logic functions (AND, OR, NAND, or NOR) on selected I/O lines. The CPU must issue two control bytes; the first will select the logic condition, and the second will contain masking information indicating which bit(s) of eight the PIO will monitor for the logic condition. The $\overline{INT}$ signal will exist while the logic condition is present. (See Codes I & J, Table 1.)

d. Interrupt Enable/Disable

To enable or disable the $\overline{INT}$ lines in all modes, the CPU must issue a control byte for each port (see Codes L M N & P, Appendix II). A and B interrupt status can be read from bit D_0 and D_1 of the status register to determine which $\overline{INT}$ line is causing the request if they are wired together (OR'd).

Timing

a. Input Data Transfer

Refer to Input Port Sequential Timing diagram. Assume an I/O mapped I/O system similar to Figure 1. The peripheral presents data to the I/O port and outputs a strobe pulse to the PIO. The strobe pulse causes three things to happen:

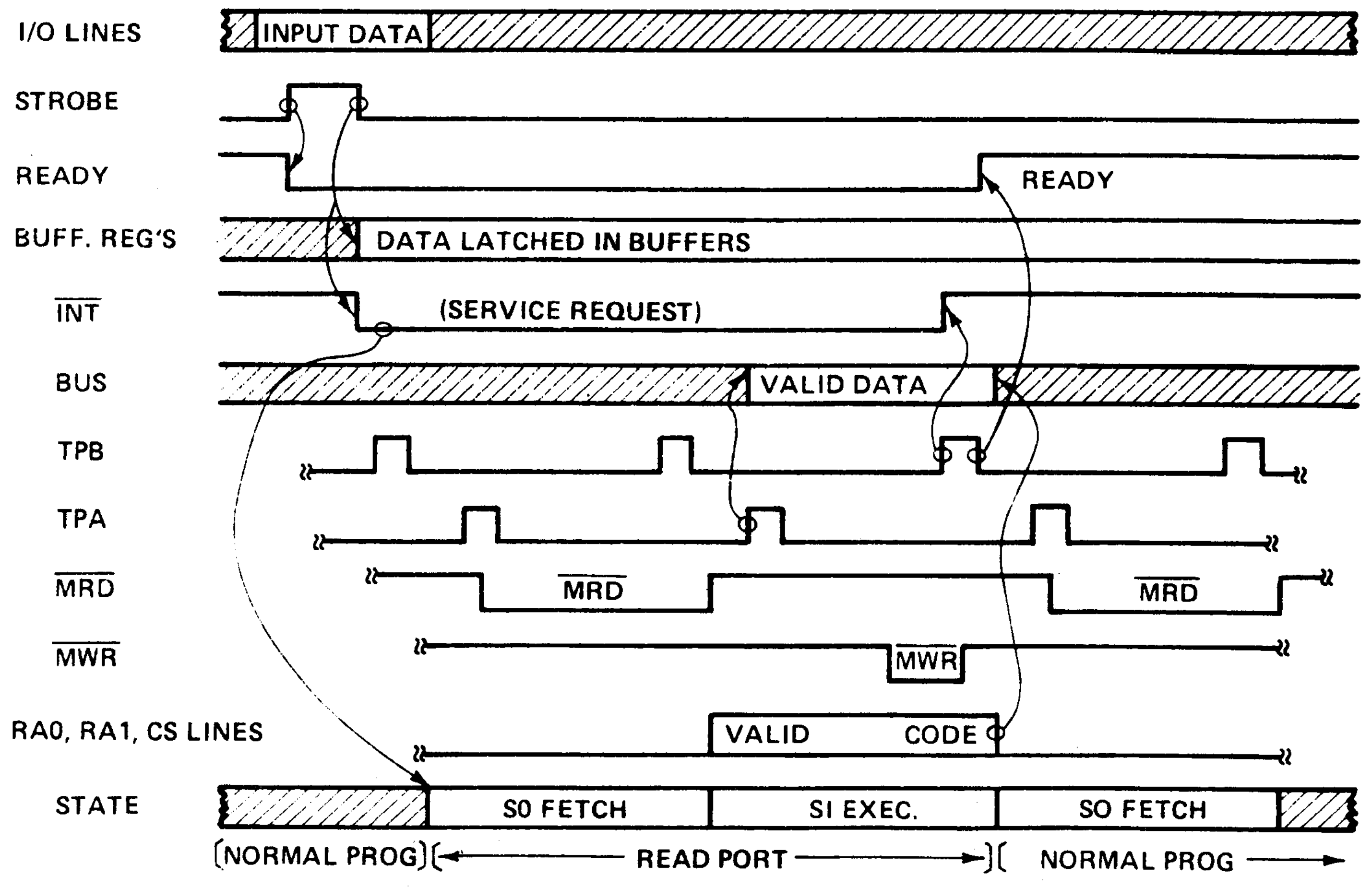
1. The READY signal is reset inhibiting further transmission from the peripheral.

2. The input data is latched in the buffer register.

3. The $\overline{INT}$ line is set low signalling the CPU to read the data.

The A and B $\overline{INT}$ lines of the PIO may be wired to the $\overline{INT}$ pin on the CPU to signal program interrupts, or they can be wired to separate $\overline{EF}$ pins where periodic polling of the $\overline{EF}$ pins is required to check for service requests.

INPUT PORT SEQUENTIAL TIMING*



*I/O SPACE I/O

In either case, the program will branch to a subroutine and execute an input instruction (INP6 or INP7, see Codes S & T, Table 1) which will assert the proper code on the RA0, RA1, and CS pins of the PIO. The PIO will place data onto the system bus so it can be used by the CPU and/or written into memory.

The TPB pulse that occurs during the $\overline{\text{MWR}}$ pulse terminates the interrupt request and sets the READY line, indicating to the peripheral that the PIO is ready to accept a new data byte.

b. Output Data Transfer

Refer to Output Port Sequential Timing diagram. Assume an I/O mapped I/O system similar to example #1. A strobe pulse from a previous output sequence or a dummy strobe causes the $\overline{\text{INT}}$ to go low signalling the CPU to output a data byte to the PIO.

The PIO $\overline{\text{INT}}$ line can be wired to the CPU $\overline{\text{INT}}$ pin or an $\overline{\text{EF}}$ pin as explained in Input Data Transfer, above. The program will branch to a subroutine and execute an output instruction (OUT6 or OUT7) which will assert the proper code on the RA0, RA1, and CS pins of the PIO.

Data will be read from memory and placed on the bus and latched into the port buffers on the trailing edge of the TPB. The READY line is also set at this time. The peripheral will transmit a strobe pulse indicating the reading process is completed. The rising edge of the strobe pulse causes the READY signal to reset, and the falling edge sets the interrupt request signalling the CPU to output another data byte.

c. Data Transfer, Bit-Programmable Mode

The CPU loads a data byte to the 8 bit port as in the norman output mode. I/O lines programmed as outputs will accept and latch data bits, however, I/O lines programmed as inputs will ignore the loaded data (see Codes H, U, and V of Table 1).

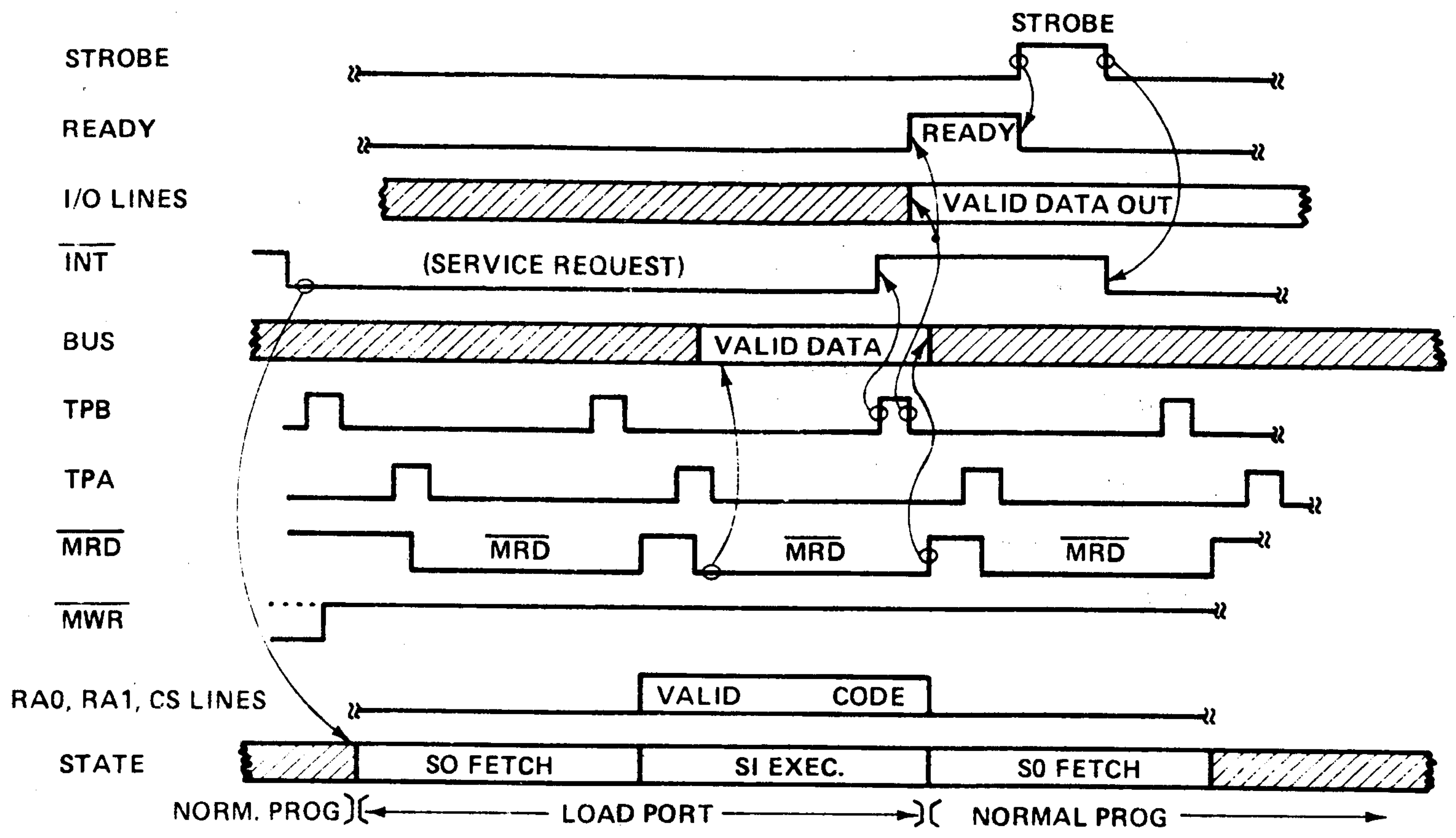
The CPU reads the 8 bit port as in the normal input mode. I/O lines are non-latching and therefore input data must be stable while the CPU reads. All 8 I/O lines are read whether they are programmed as input lines or output lines. Data read from the lines programmed as outputs will be data bits latched during the last output cycle (see Codes H, S, & T, Table 1).

Examples

Example #1 - Simple Input/Output Mode

Figure 1 shows an I/O space I/O system involving the 1802 CPU, the 1851 PIO, and two peripheral devices both with handshaking. The $\overline{\text{INT}}$ lines are individually connected to $\overline{\text{EF}}$ lines 1 and 2. Therefore, the CPU is not truly interrupted but must poll the flag lines periodically during the main program.

OUTPUT PORT SEQUENTIAL TIMING*



*I/O SPACE I/O

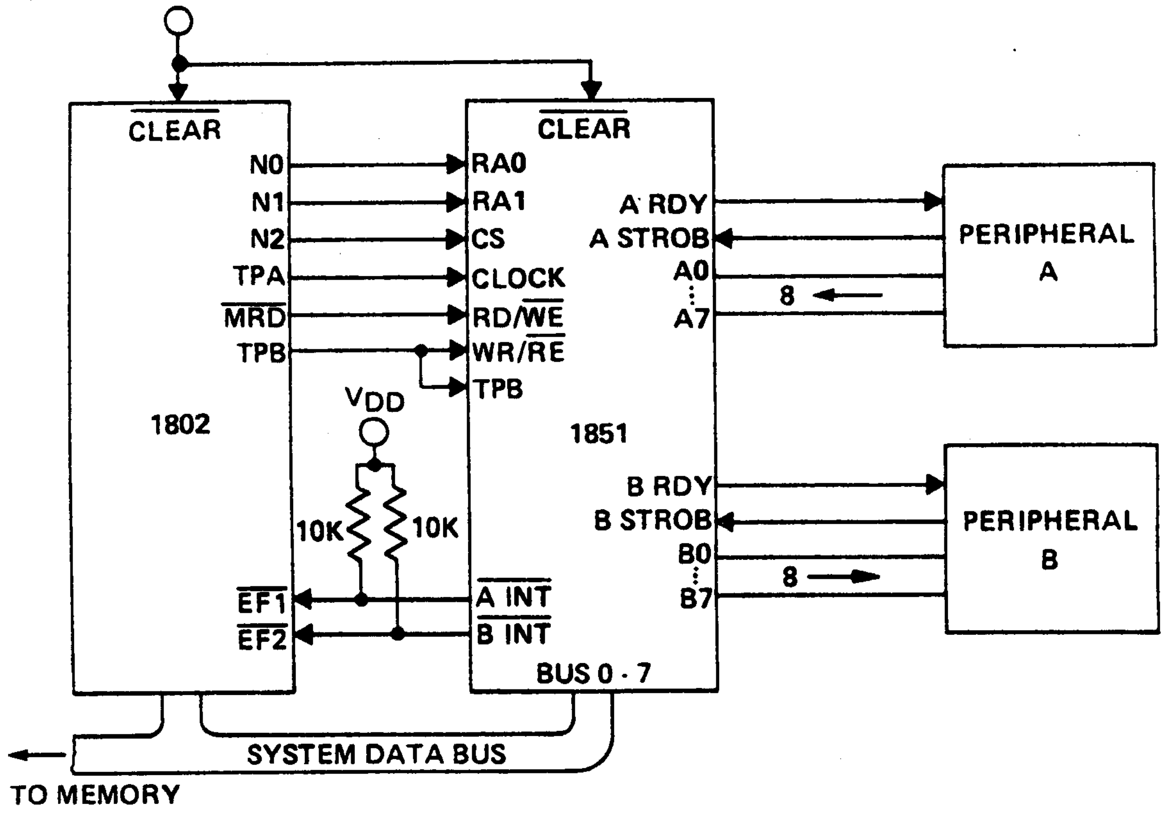
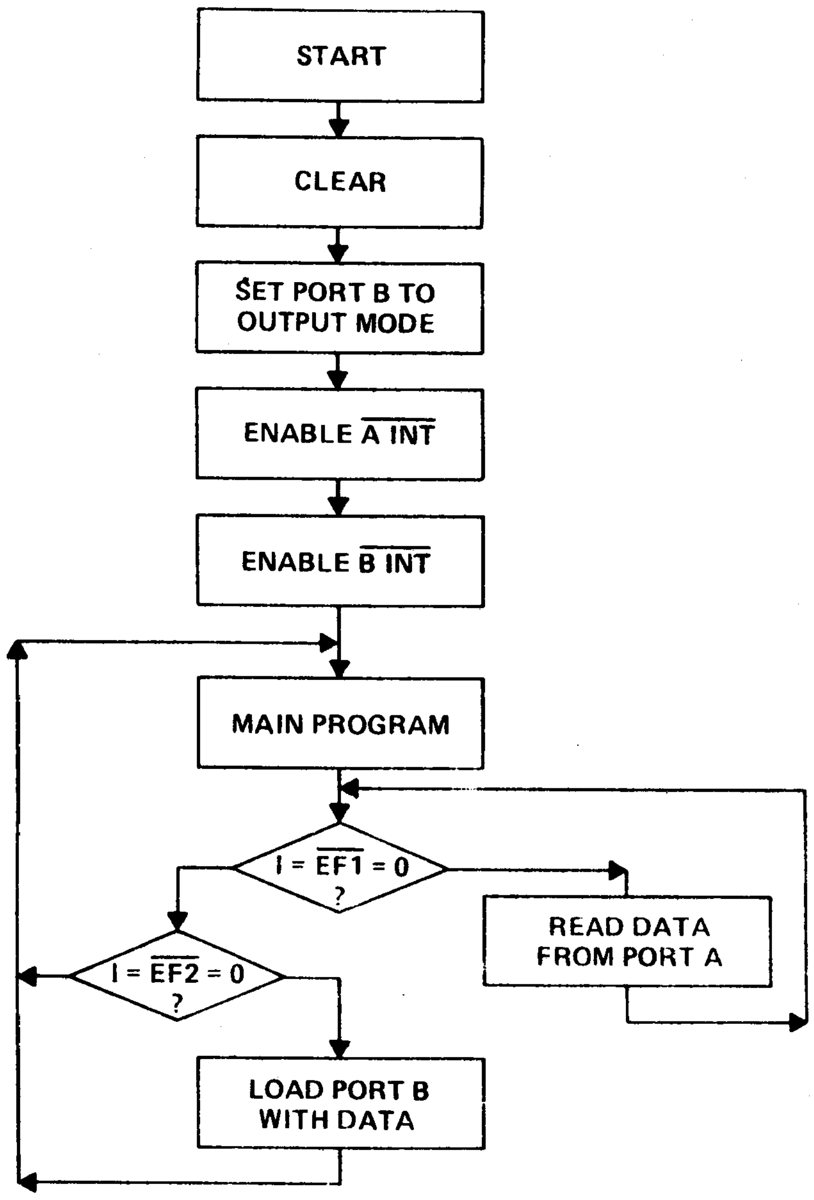


Fig. 1 - Example 1


START
CLEAR
SET PORT B TO OUTPUT MODE
ENABLE $\overline{\text{A INT}}$
ENABLE $\overline{\text{B INT}}$
MAIN PROGRAM
I = $\overline{\text{EF1}}$ = 0 ?
READ DATA FROM PORT A
I = $\overline{\text{EF2}}$ = 0 ?
LOAD PORT B WITH DATA

Flow Chart for Example 1

Step by Step - Refer to Flow Chart for example 1.

1. To begin using this system the device must be cleared which automatically programs both Port A and B to the input mode.

2. Port B is set to the output mode by loading the control register with contro byte given in Code D.

3. Port A interrupt line is enabled by loading control byte given in Code L, Table 1.

4. Port B interrupt line is enabled by loading control byte given in Code M, Table 1.

5. Now the main program can begin running.

6. Sometime during or at the end of the main program the $\overline{EF1}$ flag is polled. If it is low the program will branch to a subroutine to read Port A. The CPU must output the proper N-line Code to read A. (see Code S Table 1, and Input Port Timing diagram). When this step is completed the flag is again polled to check for more incoming data.

7. If $\overline{EF1}$ is = 1 then $\overline{EF2}$ is considered. If $\overline{EF2}$ is = 0 then the program will branch to a subroutine to load Port B with data. The CPU must output the proper N-line code, and place the 8 bit data word on the bus. (see Code V, Table 1 and Output Port Timing diagram). When this step is completed the CPU returns to the main program.

8. If $\overline{EF2}$ = 1 the CPU returns to the main program.

Example #2

Fig. 2 shows an I/O space I/O system involving the 1802 CPU, the 1851 PIO. and two peripheral devices. Peripheral A has a bi-directional bus with handshaking capability for transmit and receive. Peripheral B has a 4 bit receiver and a 4 bit transmitter.

Port A will be programmed for bi-directional mode with interrupt capability.

Port B will be programmed for bit programmable mode with interrupt capability for output data when a logic NOR (all (0's) occurs on the 4 input lines.

Priorities will be as follows:

1. Port A input data, via interrupt and subroutine

2. Port A output data, via interrupt and subroutine

3. Port B output data, via interrupt and subroutine

4. Port B input data, part of main program.

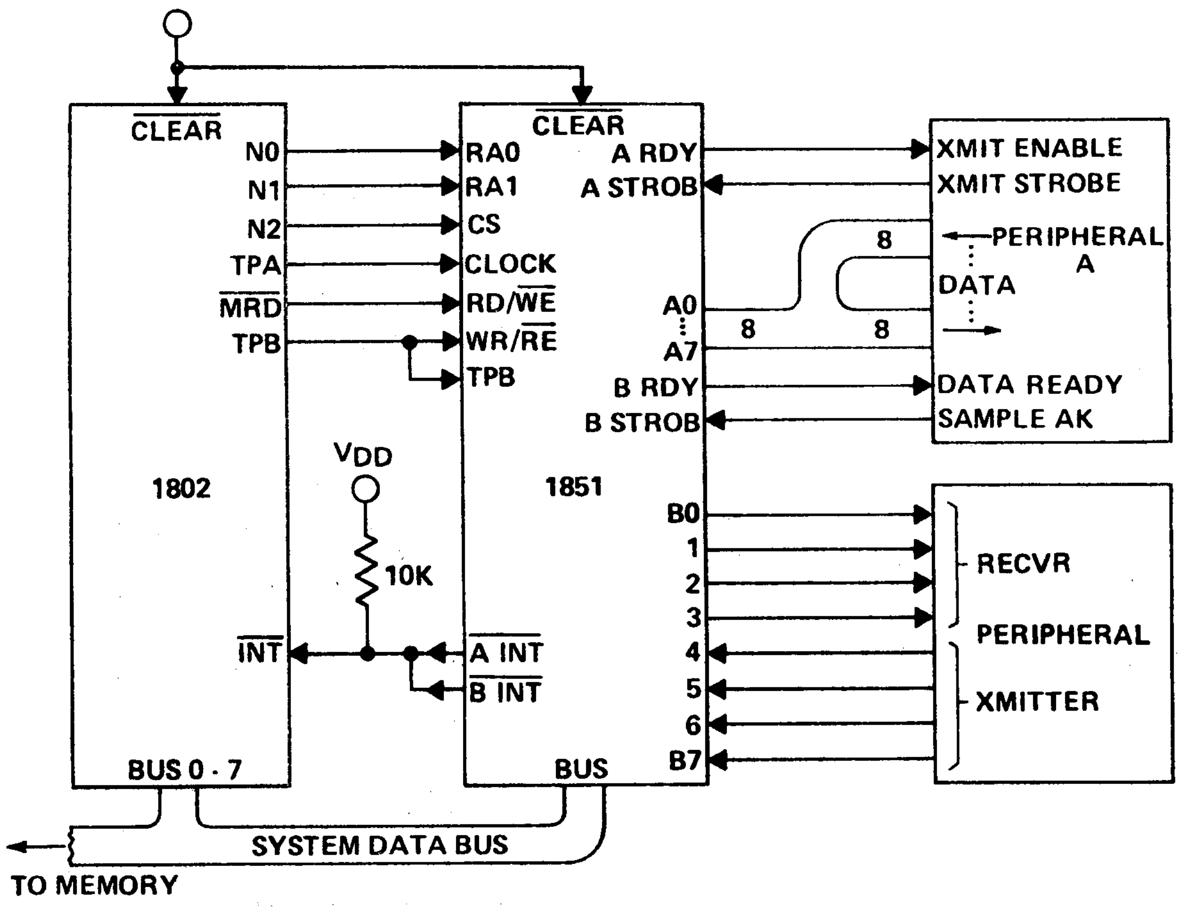
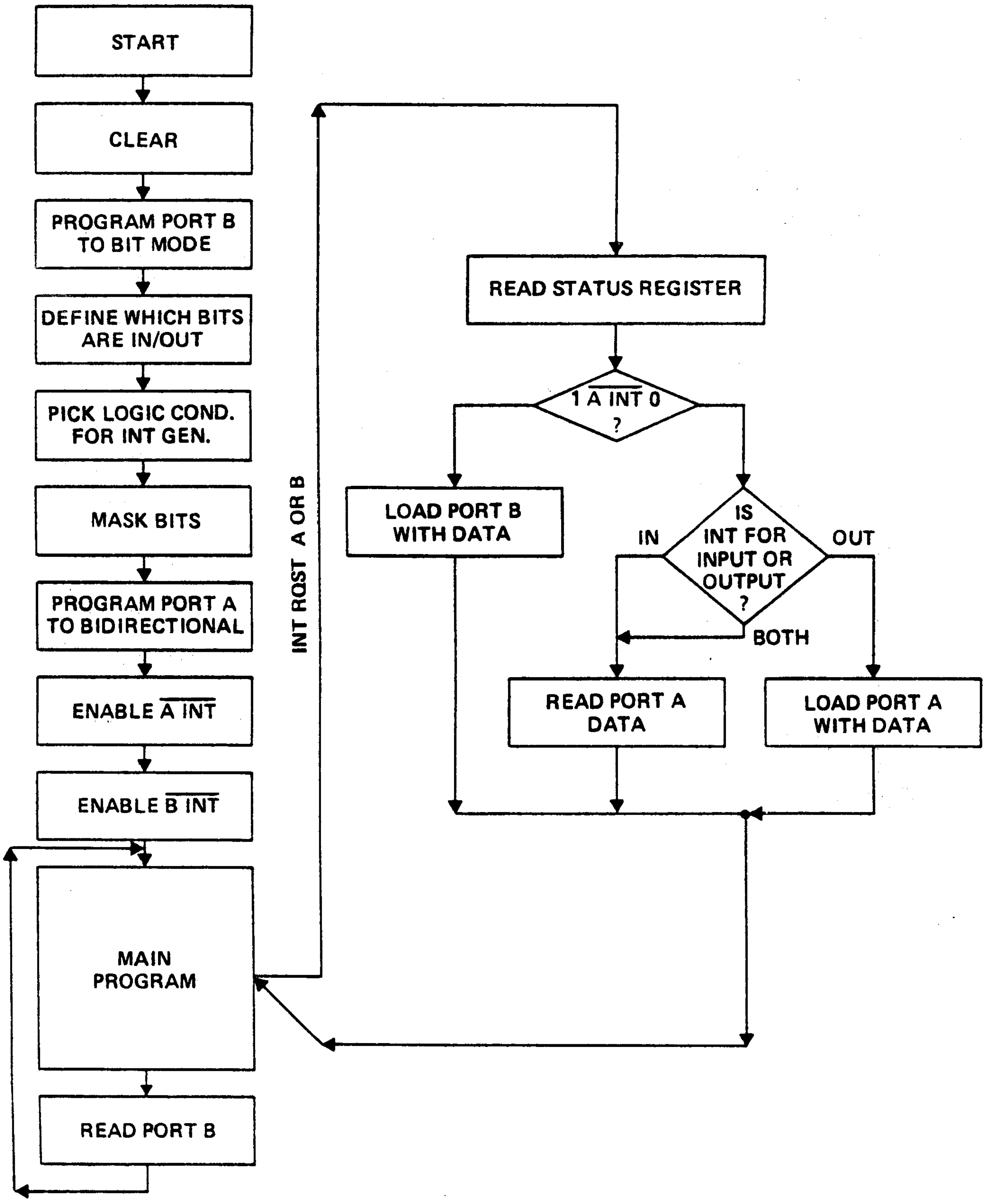


Fig. 2 - Example 2


START
CLEAR
PROGRAM PORT B TO BIT MODE
DEFINE WHICH BITS ARE IN/OUT
PICK LOGIC COND. FOR INT GEN.
MASK BITS
PROGRAM PORT A TO BIDIRECTIONAL
ENABLE $\overline{A\ INT}$
ENABLE $\overline{B\ INT}$
MAIN PROGRAM
READ PORT B
INT RQST A OR B
READ STATUS REGISTER
1 $\overline{A\ INT}$ 0 ?
LOAD PORT B WITH DATA
IS INT FOR INPUT OR OUTPUT ?
IN
OUT
BOTH
READ PORT A DATA
LOAD PORT A WITH DATA

Flow Chart for Example 2

Port A is used to interface with a bi-directional device, therefore the handshaking lines A READY and A STROBE are used for control of incoming data (from peripheral to CPU). B READY and B STROBE handshaking lines are used for output control (from CPU to peripheral).

Port B is used in the bit programmable mode having bits B0, B1, B2, and B3 as outputs (data from CPU to peripheral) and Bits B4, B5, B6, and B7 as inputs (data from peripheral to CPU).

The proper coding of the CPU N-lines will select whether the data will be read in, or written out to the PIO, or whether the status register is to be read, or whether the control register is to be loaded with a control byte. (See Table 1).

Step by Step

1. To begin using this system the device must be cleared which automatically programs both Port A and B to the input mode.

2. Since Port A is going to be in the bi-directional mode, Port B must be programmed in the bit programmable mode. Port B is set to the bit mode by loading control byte given in Code G, Table 1.

3. Code H control byte must be loaded. This byte determines which bits are input/output.

4. Code I control byte must be loaded which determines the logical condition of bits required to generate an interrupt request from Port B (in this case a NOR condition).

5. Code J control byte must be loaded. This byte tells which of the 8 bits in Port B are monitored and which are masked for interrupt generation. (In this case all of the input lines B4, B5, B6, and B7 will be monitored).

6. Now Port A is set to bi-directional mode by loading Code E control byte.

7. Port A interrupt line is enabled by loading the control register with Code L control byte.

8. Port B interrupt line is enabled by loading the control register with Code M control byte.

9. Now the main program can begin running.

10. Port B will be read periodically as the main program repeats its loop.
11. When an interrupt request is received by the CPU it will branch to a subroutine, and automatically de-activate the interrupt enable to inhibit further interruptions.
12. The CPU must read the status register by outputting the proper N-line Code R. The status register will contain information describing which interrupt has occured A and/or B, and also indicated whether $\overline{\text{A INT}}$ was caused by an input or an output demand from peripheral A.
NOTE that $\overline{\text{A INT}}$ and $\overline{\text{B INT}}$ lines are wired (OR'd together (see Fig. 2).

13. Several decisions will be made based on the information contained in the status register.

14. To read Port A the CPU must output the proper N-line Code S.

15. To load Port A with data the CPU must output the proper N-line code, and place the 8 bit data word on the bus (see Table 1).

16. To load Port B (bits B0, B1, B2, and B3) with data the CPU must output the proper N-line code, and place the 4 bit data on the bus (see Table 1).

17. Once the interrupt subroutines are completed the CPU returns to the main program and the interrupt enable (CPU) is activated.

For more details or specific application requirements, Contact Jerry Johnson, (X6776).

Table 1

CODE	DESIRED ACTION	CS	RA1	RA0	RD/WE	WR/RE	D7	D6	D5	D4	D3	D2	D1	D0
							DATA BUS							
A	SET PORT A TO INPUT MODE	1	0	1	0	1	0	0	X	0	1	X	1	1
B	SET PORT B TO INPUT MODE	1	0	1	0	1	0	0	X	1	0	X	1	1
C	SET PORT A TO OUTPUT MODE	1	0	1	0	1	0	1	X	0	1	X	1	1
D	SET PORT B TO OUTPUT MODE	1	0	1	0	1	0	1	X	1	0	X	1	1
E	SET PORT A TO BIDIRECTIONAL (PORT B MUST BE IN BIT MODE FIRST)	1	0	1	0	1	1	0	X	0	1	X	1	1
F	SET PORT A TO BIT - PROG MODE	1	0	1	0	1	1	1	X	0	1	X	1	1
G	SET PORT B TO BIT - PROG MODE	1	0	1	0	1	1	1	X	1	0	X	1	1
H	I/O BIT ASSIGNMENT (A OR B)	1	0	1	0	1	(0 = INPUT, 1 = OUTPUT)							
I	SET LOGICAL CONDITION FOR INTERRUPT GENERATION (BIT - MODE) (D3) 0 = PORT A, 1 = PORT B (D4) 0 = NO MASK, 1 = MASK BYTE FOLLOWS NEXT ↓	1	0	1	0	1	0	D6	D5	D4	D3	1	0	1
J	SET MASKING OF BITS (PRECEEDED BY CODE I)	1	0	1	0	1	(0 = MONITORED, 1 = MASKED)							
K	SET RDY AND /OR STROBE LINES TO I/O LINES (BIT - MODE ONLY)	1	0	1	0	1	D7	D6	D5	D4	D3	D2	D1	0

D6	D5	LOGICAL CONDITION
0	0	NAND
0	1	OR
1	0	NOR
1	1	AND

Code K:

(D1) 0 = PORT A, 1 = PORT B

(D2) 0 = NO CHANGE TO RDY LINE FUNCTION,
1 = CHANGE PER BIT D6

(D3) 0 = NO CHANGE PER BIT D7
1 = CHANGE PER BIT D7

(D4) RDY LINE <u>OUTPUT DATA TO BE LOADED</u>

(D5) STROBE LINE <u>OUTPUT DATA TO BE LOADED</u>

(D6) RDY LINE USED AS
0 = INPUT LINE
1 = OUTPUT LINE

(D7) STROBE LINE USED AS
0 = INPUT LINE
1 = OUTPUT LINE

CODE	DESIRED ACTION	CS	RA1	RA0	RD/WE	WR/RE	D7	D6	D5	D4	D3	D2	D1	D0
L	ENABLE A $\overline{INT}$ OUTPUT	1	0	1	0	1	1	X	X	X	0	0	0	1
M	ENABLE B $\overline{INT}$ OUTPUT	1	0	1	0	1	1	X	X	X	1	0	0	1
N	DISABLE A $\overline{INT}$ OUTPUT	1	0	1	0	1	0	X	X	X	0	0	0	1
P	DISABLE B $\overline{INT}$ OUTPUT	1	0	1	0	1	0	X	X	X	1	0	0	1
R	READ STATUS REGISTER	1	0	1	1	0	D7	D6	D5	D4	D3	D2	D1	D0

Code R:

(D0) $\overline{B\ INT}$ STATUS (1 MEANS SET) } ALL MODES
(D1) $\overline{A\ INT}$ STATUS (1 MEANS SET) }

(D2) 1 = $\overline{A\ INT}$ WAS CAUSED BY B STROBE } BIDIRECT MODE A ONLY
(D3) 1 = $\overline{A\ INT}$ WAS CAUSED BY A STROBE }

(D4) A RDY INPUT DATA } BIT PROGRAMMABLE MODE
(D5) A STROBE INPUT DATA }
(D6) B RDY INPUT DATA }
(D7) B RDY INPUT DATA }

CODE	DESIRED ACTION	CS	RA1	RA0	RD/WE	WR/RE	DATA BUS
S	READ PORT A	1	1	0	1	0	(INPUT DATA BYTE)
T	READ PORT B	1	1	1	1	0	(INPUT DATA BYTE)
U	LOAD PORT A	1	1	0	0	1	(OUTPUT DATA BYTE)
V	LOAD PORT B	1	1	1	0	1	(OUTPUT DATA BYTE)

Table 2 - CDP1851 Programming Modes

Mode	(8) Port A Data Pins	(2) Port A Handshaking Pins	(8) Port B Data Pins	(2) Port B Handshaking Pins
Input	Accept Input data	Ready Strobe	Accept Input data	Ready Strobe
Output	Output Data	Ready Strobe	Output Data	Ready Strobe
Bi-directional (Port A only)	Transfer input/output Data	Input Handshaking for Port A	Must be previous-ly set to bit-programmable mode	Output Handshaking for Port A
Bit-Programmable	Programmed individually as inputs or outputs	Programmed individually as inputs or outputs	Programmed individually as inputs or outputs	Programmed individually as inputs or outputs

Memory Mapping The 1851

When using the 1851 in a memory mapped application, with connections as shown in Fig. 1, some difficulties in programming the device may exist. The bit program mode is most prone to having errors.

Known Errors

1. Bits BØ and B1 will not program as outputs but remain as inputs, when the device is used at V_{DD} voltages above approximately 7 volts.
2. The logical conditions (AND, NOR, OR, NAND) for interrupt response (programmed by byte #9 in the data sheet) do not load correctly. This phenomenon occurs at all voltages.

Reason for Errors

When configured as shown in Fig. 1, $\overline{\text{MRD}}$ drives WR/$\overline{\text{RE}}$ (pin 38), and $\overline{\text{MWR}}$ drives RD/$\overline{\text{WE}}$ (pin 39). During a memory write cycle the $\overline{\text{MRD}}$ signal stays high. $\overline{\text{MWR}}$ goes low enabling the data bus input buffers, allowing data to be presented to an internal register. See Fig. 2. When the $\overline{\text{MWR}}$ signal goes high again it latches the data in the register, but also shuts off the data at the input buffers, creating a race condition of data hold and latching the data. The race condition is independent of system clock speed.

The Fix

Fig. 3 shows the added hardware required to avoid the race condition. By inverting $\overline{\text{MRD}}$ and driving RD/$\overline{\text{WE}}$ (pin 39) the bus buffers remain on during and well after the inverted $\overline{\text{MWR}}$ signal clocks the internal data register, thereby avoiding the race condition during the write operation. Switching the $\overline{\text{MRD}}$ and $\overline{\text{MWR}}$ signals and adding inverters makes no difference logically during a read operation.

Additional Hardware for High Speed Systems

Two inverters are inserted in the TPA line to compensate for the added inverter delay, introduced in the $\overline{\text{MRD}}$ line, so that the flip-flop can successfully latch the READ signal at high system speeds.

Since CS is generated from the high address byte, and latched with TPA, CS must also be delayed to compensate for the added TPA delay.

The entire change can be implemented with one 4069 package.

For additional information contact G. Johnson, X6776.

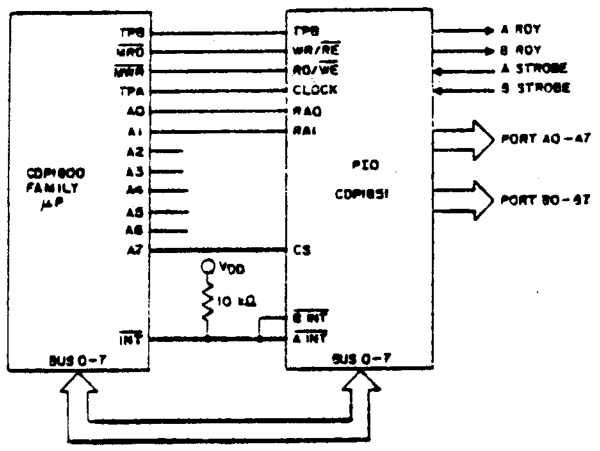


Figure 1

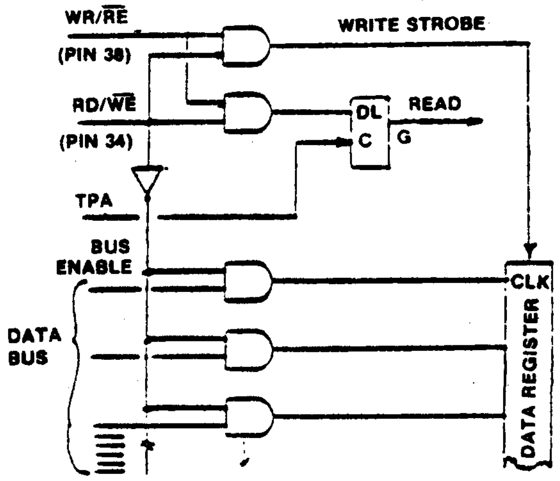


Figure 2

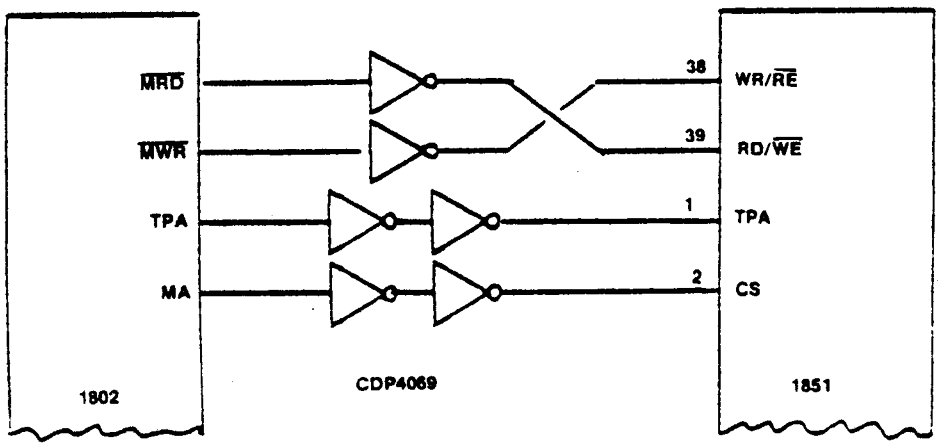


Figure 3

Using The CDP1852 I/O Port for DMA Operations

At present the MPM-201B CDP1802 Users Manual does not consider the use of the CDP1852 for DMA-IN operations. Instead it gives an example (page 79) using 2 CD4076 4-bit registers, 1/2 of a CD4013 Dual Flip-Flop, and various logic gates: a total of at least four packages are required to facilitate the DMA input. Using the versatile CDP1852 reduces the complexity of the circuit and the number of DIP packages.

One possible circuit configuration is shown in Figure 1. In system operation, data is strobed into the port's 8-bit register by a high (1) level on the clock pin. The negative high-to-low transition of the clock sets the internal service request latch which causes a low (0) on $\overline{SR}$. The $\overline{SR}$ signal causes a DMA-IN command at the CPU. The CPU then goes to a DMA-input cycle (S2) where SC0 = 0 and SC1 = 1. SC0 is inverted so that the CS1 and CS2 pins both receive a logic (1) during the S2 cycle. Additional logic is used to gate off the $\overline{\text{DMA-IN}}$ command after the S2 cycle begins. When the S2 cycle is completed the CPU returns to normal fetch (S0) and execute (S1) states. The absence of a logic (1) on either the CS1 or CS2 pins will reset the $\overline{SR}$ flip-flop and disable the output of the CDP1852.

References:

1. Users Manual for the RCA CDP1802 (MPM-201B)
2. CDP1852 data sheet (File # 1166)

For more information on this article or related topics, contact Jerry Johnson, X6776

USING THE CDP1852 AS AN INPUT PORT FOR DMA OPERATION

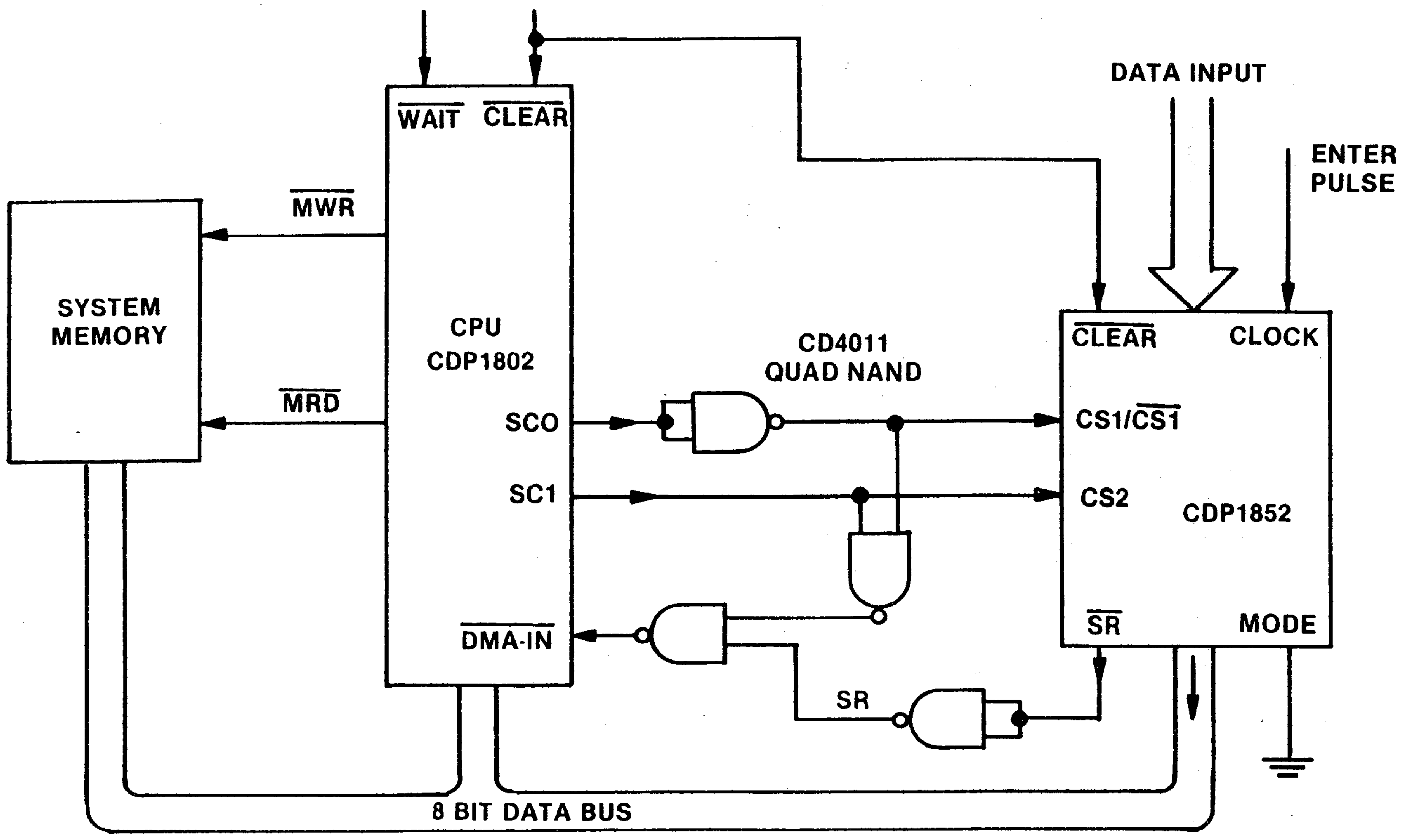
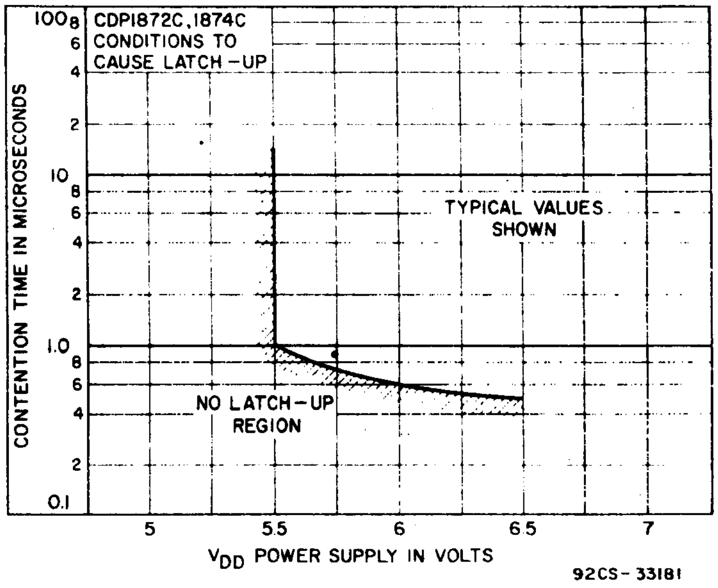


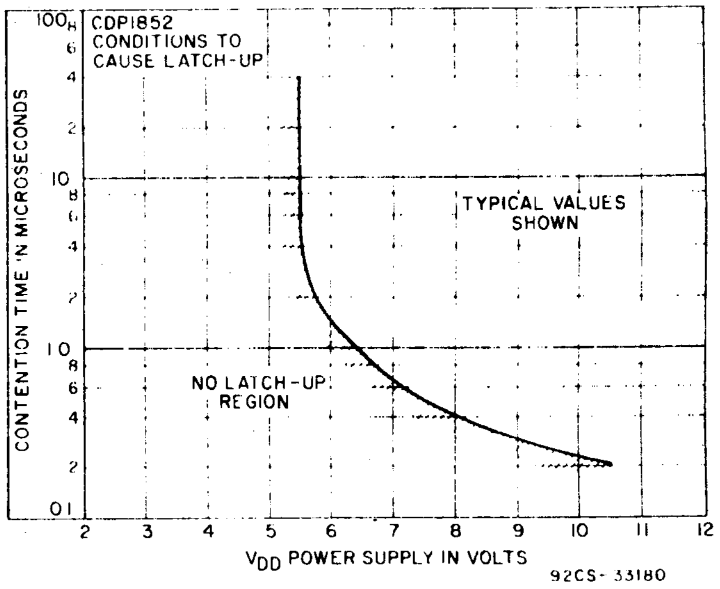
Figure 1

Bus Contention Effects on CMOS I/O Ports

Because of the parasitic transistors inherent to bulk CMOS devices, care must be taken with high output current devices to avoid potential problems in systems designed with known bus contention. It has been observed that such parts are prone to SCR latch-up when outputs are shorted to their opposite state for any significant amounts of time, especially when working at high voltages.

Laboratory measurements have confirmed that latch-up can occur with the CDP1852 byte-wide I/O port, and with the new CDP1872C and CDP1874C high-speed I/O ports. The latch-up phenomenon is a function of voltage shorting time, as shown in the typical curves below, and can occur if 4 or more outputs are simultaneously shorted.





Even though the conditions for latch-up are rather severe, and will rarely occur in a properly designed system, especially at low voltages (<5.5V), the designer should take the precaution during his system design to analyze his bus configuration for known long bus contention times or undefined bus conditions during power initialization. If a latch-up does occur, the problem is more serious than an instantaneous current spike generated during the contention time : these devices can draw hundreds of milliamps and pull the power supply down below normal operating level, with power turn-off needed to end the mechanism. Worst of all, latch-up can cause possible damage to the device itself.

For more information, contact Joe Paradise, X7352.

1854A vs. 1854 · Design Improvements For Higher System Speeds

Abstract: Design differences between the CDP1854, CDP1854A and other industry types are detailed from an applications point of view. The improved 1854A performance has been demonstrated to be competitive with other CMOS UARTs at maximum 1802 system speeds.

Background: The original 1854 UART could not tolerate the same amount of distortion on received data as competitive CMOS UARTs, possibly resulting in loss of synchronization and/or data permutation. The causes for the above mentioned behavior are divided into three distinct areas as follows:

1. Difficulty of operation at high clock speeds.

   On some units the maximum operating clock speed was less than the industry standard of 3.2 MHz at 5 volts. This sluggishness was caused by a relatively slow parity checking circuit which has been redesigned in the 1854A to improve performance.

2. Off-centered detection of data bits.

   Industry standard UARTs such as the Western Digital TR1602 use 16 clock pulses per data bit and sample the bit in the middle at clock pulse 7-1/2. The original 1854, sampled at 6-1/2 clock pulses. This minor skew accounted for less tolerance for distortion at the beginning of a bit, but was more forgiving of distortion at the end of the bit. The new 1854A samples at 7-1/2 clock pulses.

3. Clock counter reset delay.

   The major source of trouble was caused by a 6-1/2 clock pulse delay between the time a stop bit was detected and the time the clock counter, which enabled the UART to search for the next start bit was reset. The 1854 did not reset its counter until clock pulse 13. Other industry standard UARTs reset their counters at 7-1/2 clock pulses, the exact time at which the first stop bit is detected, which permits the receiver to immediately begin searching for the next start bit. When using the 1854 loss of synchronization may result if transmitter and receiver frequencies are allowed to differ by more than 1.25% and/or the stop pulse is distorted or too narrow. The redesigned 1854A resets the clock counter at 9 clock pulses - an improvement over the 1854, as evidenced in the tolerance requirements listed in Table 1, attached.

Conclusions: The 1854A will operate properly at 3.2 MHz to the frequency tolerances noted in Table 1. The 1854 is limited to speeds below 3 MHz, and has less tolerance to line distortion caused by envelope delay, phase jitter, etc., for the reasons detailed in the above discussion.

For more details or specific application requirements, contact

Jerry Johnson, X6776

TABLE 1

Theoretical Tolerance to
Frequency Difference Between
Transmitter and Receiver Clocks *

	1854	1854A	Others
$f_r < f_t$	1.25%	3.75%	5.31%
$f_r > f_t$	4.06%	4.69%	4.69%

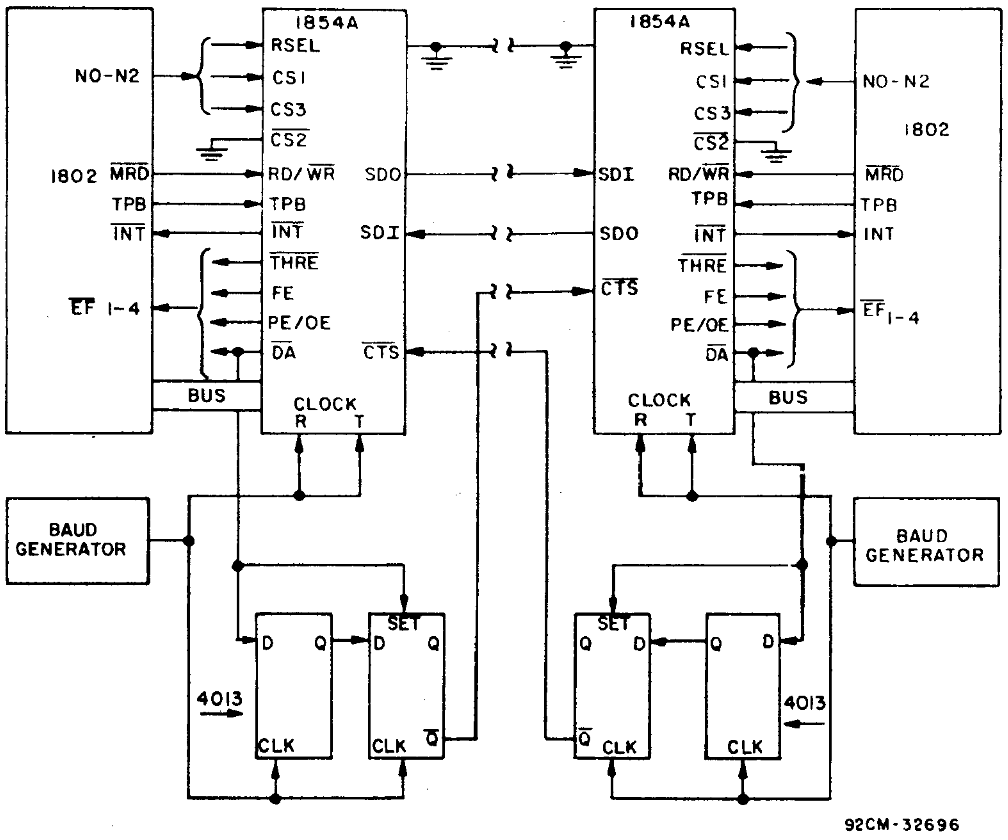
* Data Train consisting of one start bit, 8 data bits, and 1 stop bit.

TABLE 2

Published Specifications for CMOS UART's

	RCA				HARRIS		INTERSIL	
Parameter	CDP1854D	CDP1854CD	CDP1854A	CDP1854AC	HD-6402A	HD-6402	IM6402A	IM6402
Nominal Operating Voltage	10	5	10	5	10	5	10	5
Absolute max. Voltage	15	7	11	7	12	8	12	8
Maximum Clock Freq. MHz	6.4 @ 10V	3.0 @ 5V	6.4 @ 10V	3.2 @ 5V	4.0 @ 10V	2.0 @ 5V	4.0 @ 10V	2.0 @ 5V

Interfacing Two COSMAC Systems Using 1854A UART's



Asynchronous full duplex data communication between two independent COSMAC systems can be achieved using two 1854A UART's wired as shown. Baud rate generators must be the same frequency for both systems.

When the UART receives a serial data character it generates a data available ($\overline{DA}$) and an interrupt request signal ($\overline{INT}$). Until the CPU responds by reading the data the $\overline{DA}$ line inhibits word transmission from the other UART, and vice versa.

Internal timing of the 1854A is such that a pulse at its $\overline{CTS}$ input at the time that $\overline{DA}$ of the receiving UART is activated will cause the stop bit to be reset early. To avoid errors, the 4013 Flip-Flops provide a 2 clock period delay between $\overline{DA}$ (low) and $\overline{CTS}$ (high).

If the 1802 clock is used to drive the UART clocks the Flip-Flops can be eliminated and replaced with a simple inverter since the CPU will take many more than 2 clock periods from when it is flagged to when it services (Reads) the UART.

For additional information or specific applications, contact Jerry Johnson, X6776.

Interfacing the 1854A with the 8085

Circuit Description:

The 1854A UART can be interfaced with the 8085 CPU with the addition of several logic devices. The lack of a convenient timing pulse to strobe the TPB pin on the 1854A UART during read or write cycles is resolved by OR'ing $\overline{WR}$ and $\overline{RD}$ CPU signals to create a quasi TPB pulse (see schematic diagram). Inverters are required on the $\overline{DA}$ and the RESET OUT signals as shown. Note the 1854A is operated in Mode 1 (the COSMAC mode).

A working test circuit was developed using an Intel SDK-85 evaluation kit having a system clock speed of 3.072 MHz, thus the 1854A was operated near its maximum clock speed of 3.2 MHz at 5 volts.

Timing Considerations: Refer to timing diagrams

A worst case timing analysis based on data sheet values for both parts reveals two possible problems.

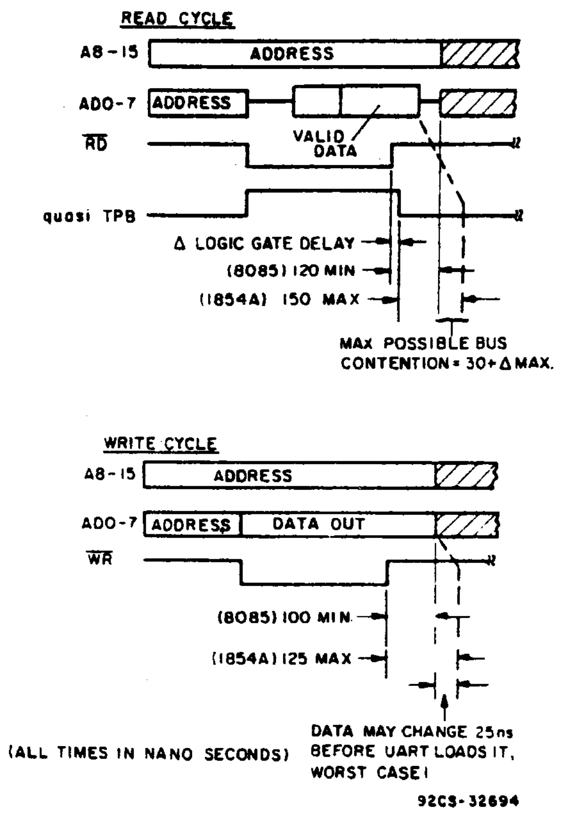
1. Read Cycle - The 8085 specifies a minimum of 120 ns from the trailing edge of read to the output of the next address on the ADO-7 lines (Trae). The 1854A has a maximum data hold time of 150 ns after TPB goes low (Ttrs). Since TPB is created via a logic gate which OR's $\overline{WR}$ and $\overline{RD}$, any gate delay time must also be considered. Possible bus contention of 30 ns plus gate delay may exist, worst case.

2. Write Cycle - The 8085 specifies a minimum of 100 ns for data hold time after the trailing edge of $\overline{WR}$ (Twd). The 1854A requires a maximum data hold time after $\overline{WR}$ (Twr) of 125 ns, therefore the 8085 may be unable to load data into the UART worst case.

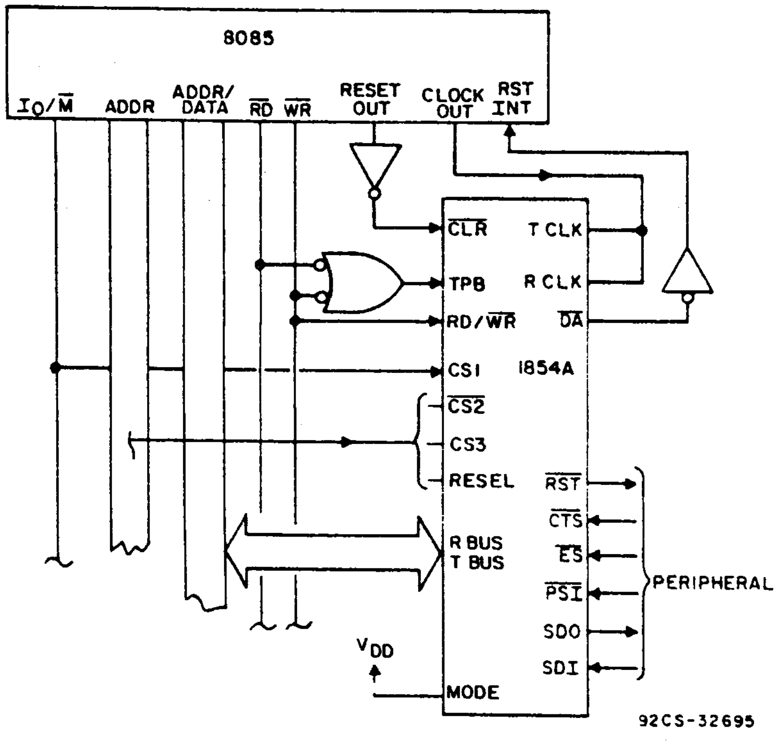
Summary:

The 1854A can be used with the 8085 as configured in this article provided the parts are selected to avoid worst case timing problems.

For more details or specific application requirements, contact Jerry Johnson, X6776.



TIMING DIAGRAMS



INTERFACE SCHEMATIC

Competitive UARTs - An Industry Comparison

Some popular competitive industry standard UARTs are listed here to show relative differences in parameters affecting interchangeability. Except for variations in the function of pin 2, all types have the same pin out.

The RCA CDP1854A is unique in that it has two operating modes selectable via pin 2. Mode 0 (pin 2 tied to V_{SS}) configures the UART to industry standard operation and pin out. Mode 1 (pin 2 tied to V_{DD}) modifies certain signal polarities to allow easy interface with the 1800 series microprocessors, without additional components.

INDUSTRY STANDARD UARTs

Manufacturer & Type	Technology	Voltage(s)	Quiescent Current	Pin 2	Max Clock Freq.	Notes
AMI						
S-1602	N Channel	+5V only	70 mA	N.C	0.8 MHz	TTL compatible
G.I.						
AY-5-1013A	P Channel	+5V & -12V	16 mA	-12V	0.64 MHz	TTL compatible
AY-6-1013	P Channel	+5V & -12V	19 mA	-12V	0.36 MHz	TTL compatible
AY-3-1014A	P Channel	+5V to +14V	20 mA	N.C.	0.48 MHz	TTL compatible
AY-3-1015D	P Channel	+5V only	15 mA	N.C.	0.48 MHz	TTL compatible
HARRIS						
HD-6402	CMOS	+5V only	100 uA	N.C.	2 MHz	
INTERSIL						
IM-6402	CMOS	+5V only	500 uA	N.C.	1 MHz	
IM-6402A	CMOS	+5V to +10V	800 uA	N.C.	4 MHz	
IM-6402-1	CMOS	+5V only	100 uA	N.C.	2 MHz	
RCA						
CDP1854AC	CMOS	+5V only	200 uA	Mode	3.2 MHz	Pin 2 must be @ V_{SS}, Mode = 0
CDP1854A	CMOS	+5V to +10V	50 to 200uA	Mode	6.4 MHz	Pin 2 must be @ V_{SS}, Mode = 0
STANDARD MICRO						
COM-8017	N Channel	+5V only	25 mA	N.C.	0.64 MHz	TTL compatible
WESTERN DIGITAL						
TR-1602	N. Channel	+5V & -12V	25 mA	-12V	0.32 MHz	TTL compatible
TR-1863	N Channel	+5V only	25 mA	N.C.	3.5 Mhz	TTL compatible

Using the MDU

COSMAC systems using the CDP1855 MDU can perform an 8 bit x 8 bit multiply in 8% of the machine time required by systems using a software multiply algorithm. Programs for both methods are explained below:

Software Method

Software implementation of an 8x8 multiply algorithm, based on a method of add and shift right, is listed in Program #1. Normal overhead of initializing the CPU registers is done in the main program. Since 9 shifts are required to complete the 8x8 multiply operation, 18 instructions of the subroutine must be repeated nine times, resulting in a total of 344 machine cycles. Assuming a clock frequency of 3 MHz, the time required to complete the operation is 917 uSec.

Hardware Method Using the CDP1855

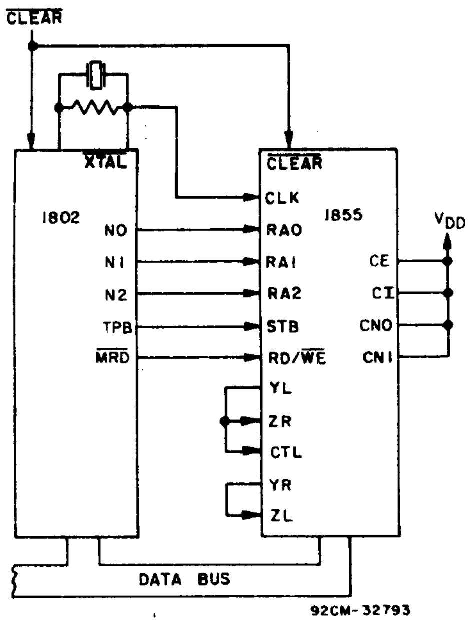


Figure 1

Figure 1 shows the pin connections for using the 1855 with the 1802/1804. A subroutine for performing an 8x8 multiply is listed in Program #2. The overhead of register initializing is about the same as in the 1st program, however, the actual 13 instruction subroutine must be executed only once since the MDU does all the shifting and adding.

Again assuming a clock frequency of 3MHz, the time required to load two 8 bit numbers, multiply, and read the 16 bit result is only 69 uSec.

Step-By-Step Operation of the MDU

1. Load the control register by executing an OUT 7 instruction followed by the

control byte, in this case F0 (11110000). Logic highs on bits 4 and 5 define the number of cascaded MDU's as one. Bit 6 resets the internal sequence counters, and bit 7 selects the optional clock prescaler, which causes the shift frequency to be 1/2 of the clock frequency. This is required when the clock frequency is higher than 1.5 MHz at 5V operation or 3 MHz at 10V operation.

2. Load the X register by executing an OUT 4 instruction followed by one of the 8 bit operands.

3. Load the Z register by executing an OUT 5 instruction followed by the other 8 bit operand. The order of steps 2 and 3 can be interchanged if desired.

4. Load the control register as in step 1 but this time with F9 (11111001). Bit 3 resets register Y to zero and the code of 01 on bits 1 and 0 cause the multiply operation to begin.

5. The MDU will automatically perform the 8x8 multiply and store the most significant half of the 16 bit answer in register Y and the lesser significant half in the Z register. The time required to perform the actual multiply operation is 8N+1 shifts, where N = the number of MDU's, in this case one. Therefore, it takes 9 shift cycles, and since the prescaler is being used, this translates to 18 clock pulses of the CPU.

   To provide enough time for the multiply operation, software should be arranged so that one 2-cycle instruction occurs between the multiply command and the reading of the answer.

6. Read the most significant half of the 16 bit answer from register Y by executing an INP6 instruction.

7. Read the lesser significant half of the 16 bit answer from register Z by executing an INP5 instruction. The order of steps 6 and 7 can be interchanged if desired.

PROGRAM # 1

```
                                   ASSEMBLY
M ADDRESS         BYTE             PROGRAM          COMMENTS
0000              F800             LDI 00           R0 = Prog. counter
0002              B7               PHI R7           R7 = Product register
0003              B8               PHI R8           R7.1 = 00
0004              F8FF             LDI FF                                              Initial-
0005              A8               PLO R8           R8 = 00FF                          ization
0007              F803             LDI 03

0009              AE               PLO RE
000A              F802             LDI 02
000C              BE               PHI RE           RE = Sub. Prog. counter
000D              DE               SEP RE           Call subroutine
000E              (data)           (data)           8 bit Multiplicand
000F              (data)           (data)           8 bit multiplier

0202              D0               SEP R0           Return to main prog.
0203              F809             LDI 09
0205              A6               PLO R6           R6 = Loop counter
0206              FE               SHL              Presets DF = 0
0207              40               LDA R0           Load M'cand

0208              58               STR R8           into 00FF
0209              40               LDA R0           Load M'lier
020A              A7               PLO R7           into R7.0
020B              E8               SEX R8                                              Sub-
020C              97               GHI R7                                              routine

020D              76               SHRC             Shift low byte
020E              B7               PHI R7
020F              87               GLO R7
0210              76               SHRC             Shift high byte
0211              A7               PLO R7

0212              3B17             BNF 17           Branch if DF = 0          Loop
0214              97               GHI R7                                      9
0215              F4               ADD              Add M'lier to R7.1       times
0216              B7               PHI R7           Return sum to R7.1
0217              26               DEC R6           Index loop count

0218              86               GLO R6
0219              3202             BZ 02            Loop count = 9?
021A              A6               PLO R6
021B              300C             BR0 0C           Continue looping
```

PROGRAM # 2 *

M ADDRESS	BYTE	ASSEMBLY PROGRAM	COMMENTS
0000	F801	LDI 01	Load 0101
0002	BE	PHI E	into
0003	AE	PLO E	RE of the CPU
0004	DE	SEP E	Go to subroutine at 0101
0005			Main program continues
0100	D0	SEP 0	Return to main program
0101	F802	LDI 02	Load
0103	BF	PHI F	RF of the CPU
0104	F800	LDI 00	with
0106	AF	PLO F	0200
0107	EE	SEX E	
0108	67F0	OUT 7, F0	Load MDU control register
010A	EF	SEX F	
010B	64	OUT 4	Load X register of MDU
010C	65	OUT 5	Load Z register of MDU
010D	EE	SEX E	
010E	67 F9	OUT 7, F9	Load MDU control register
0110	EF	SEX F	
0111	6E	INP 6	Read Y register of MDU
0112	60	IRX	
0113	6D	INP 5	Read Z register of MDU
0114	3000	BR 00	Branch to 0100
0200	(data)		8 bit Multiplicand loaded in X
0201	(data)		8 bit Multiplier loaded in Z
0202	(data)		Storage space for result, high byte
0203	(data)		Storage sapce for result, low byte

* Based on program examples submitted by J. Pintaske, RCA Brussels.

For additional information contact Jerry Johnson, X6776.

Interfacing a "Full House" MDU Board with the CDS

Figure 1 is a schmatic diagram showing four CDP1855 MDU's in cascade, with connector wiring for interfacing to the development system. Most of the connections are to existing back plane wiring, with additional jumpers to the CPU clock and to the decode module.

A universal program is listed which facilitates Multiplication, Multiplication with Addition, and Division operations.

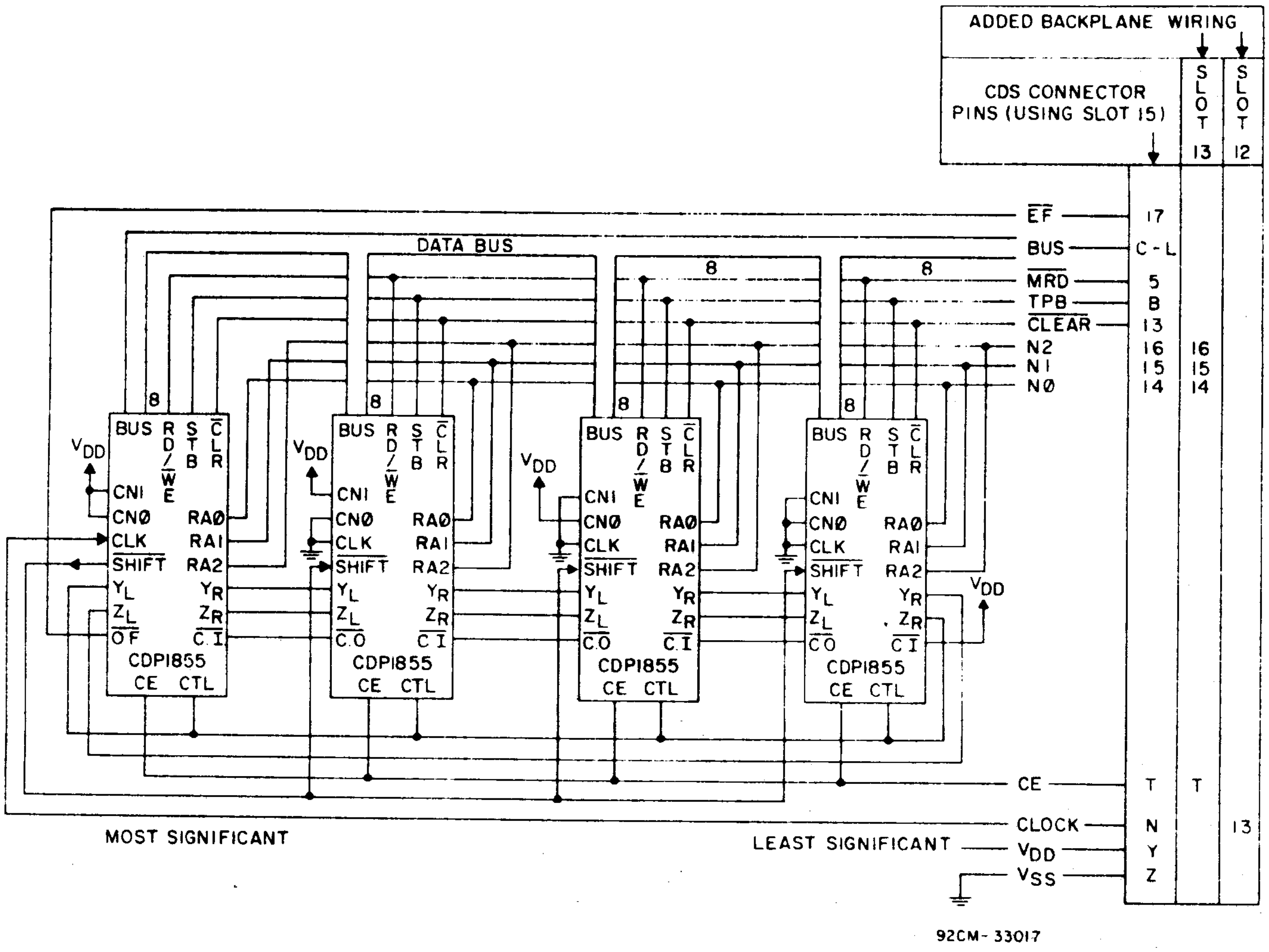


Figure 1 - Cascaded CDP1855 configuration for CDS Interface.

MULTIPLY/DIVIDE PROGRAM

M Address	Byte	Assem. Prog.	Comments
0 0 0 0	6120	OUT 1, 20	Enable two level I/O
0 0 0 2	F804	LDI 00	
0 0 0 4	BF	PHI F	
0 0 0 5	F800	LDI 40	Reg. F = 0040, storage space
0 0 0 7	AF	PLO F	for answer
0 0 0 8	67C4	OUT 7, CC	Load control register with 11001100
0 0 0 A	64XX	OUT 4, data	Load X register of MS unit
0 0 0 C	64XX	OUT 4, data	
0 0 0 E	64XX	OUT 4, data	
0 0 1 0	64XX	OUT 4, data	Load X register of LS unit
0 0 1 2	65ZZ	OUT 5, data	Load Z register of MS unit
0 0 1 4	65ZZ	OUT 5, data	
0 0 1 6	65ZZ	OUT 5, data	
0 0 1 8	65ZZ	OUT 5, data	Load Z register of LS unit
0 0 1 A	66YY	OUT 6, data	Load Y register of MS unit
0 0 1 C	66YY	OUT 6, data #	
0 0 1 E	66YY	OUT 6, data	
0 0 2 0	66YY	OUT 6, data	Load Y register of LS unit
0 0 2 2	67C1	OUT 7, C *	Load control register with 11000001
0 0 2 4	C4C4C4C4C4	10 - NOP's	Allow 33 machine cycles for the
0 0 2 9	C4C4C4C4C4		multiply/divide operation before
0 0 2 E	EF	SEX F	reading the answer (6E execute)
0 0 2 F	6E60	INP 6, IRX	Read Y register of MS unit
0 0 3 1	6E60	INP 6, IRX	
0 0 3 3	6E60	INP 6, IRX	
0 0 3 5	6E60	INP 6, IRX	Read Y register of LS unit
0 0 3 7	6D60	INP 5, IRX	Read Z register of MS unit
0 0 3 9	6D60	INP 5, IRX	
0 0 3 B	6D60	INP 5, IRX	
0 0 3 D	6D60	INP 5, IRX	Read Z register of LS unit
0 0 3 F	00	IDLE	STOP

NOTES:

0040 thru 0043 - Storage for high order 32 bits of multiply result, or 32 bit remainder from divide operation.

0044 thru 0047 - Storage for low order 32 bits of multiply result, or 32 bit result from divide operation.

# - Required for divide; optional for multiply when number is to be added to result.

* - 1 for multiply; 2 for divide operation.

Multiplication

A 32x32 bit multiplication is possible, resulting in a 64 bit product.

Operations involving numbers less than 32 bit are accomplished by loading the more significant bits with zeros.

A number may be added to the product by pre-loading the Y registers with up to 32 bits before the multiplication. The final result appears in Y (most significant) and Z (least significant).

Register X is unaltered by the operation, so that repeated multiplication by a constant number is possible without re-loading the X register each time.

Division

A 64÷32 bit division is possible resulting in a 32 bit result (Z registers), and a 32 bit remainder (Y registers).

The $\overline{\text{CO}}$ pin of the most significant MDU is used as an overflow indicator. The status register also contains a bit for overflow indication, and can be read by executing an INP 3 instruction. Overflow occurs if the division result has more than 32 bits.

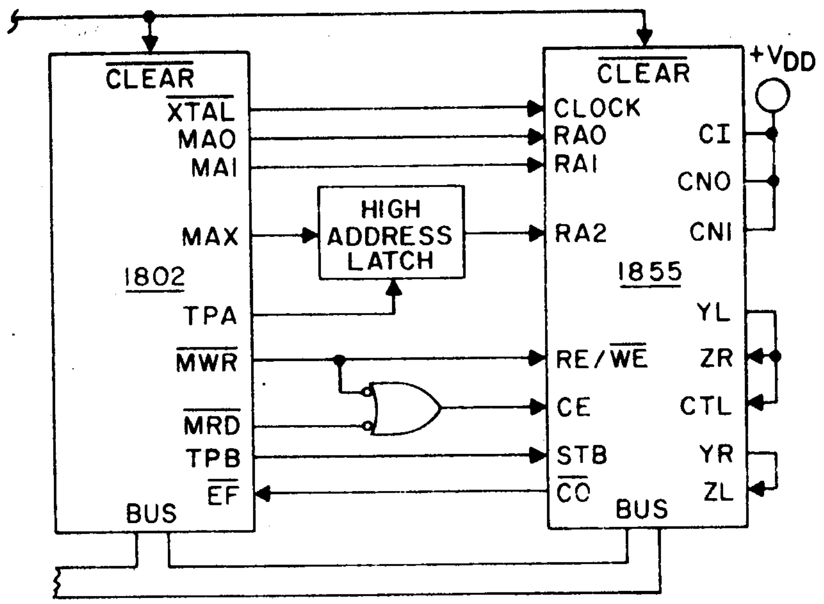
For additional information or specific applications, contact Jerry Johnson, X6776.

Memory Mapping the MDU

The current CDP1855 data sheet does not currently show how to use the device in a memory mapped configuration; however, it does mention the use of $\overline{\text{MWR}}$ for RE/$\overline{\text{WE}}$, and address lines or functions of address lines for RA0, 1 and 2. This will work provided certain non-memory operations, such as PHI, GLO, etc., do not cause spurious selection of the MDU when the contents of the internal registers appear on the address bus.

To prevent the MDU from responding to spurious addresses, and thereby avoiding bus contention, the $\overline{\text{MWR}}$ and $\overline{\text{MRD}}$ signals must be OR'd to produce a valid CE signal. The IRX instruction is a special case; it should not be used when R(X) is pointing to any of the MDU register addresses, since this instruction not only presents the contents of R(X) on the address bus, but also forces $\overline{\text{MRD}}$ low, which enables the MDU and causes inadvertent advancing of the sequence counters.

The required "OR'd" CE signal, as well as latched high order address bits, are usually available in most memory intensive systems using an 1866 or 1867 decoder, or similar hardware.



92CS-33173

For more information contact G. Johnson, X6776.

Faster MDU Throughput Using "DMA Method"

When using the MDU in the normal configuration shown in the CDP1855 data sheet, a large portion of the throughput time is consumed by loading and unloading data and control bytes. Throughput speed can be increased by 45% by using a DMA in/out technique to load and unload the MDU. Figure 1 shows the hardware needed to: sequentially address the MDU; control the timing of read/write; and issue DMA in/out signals to the CPU.

With the normal I/O mapped MDU, the CPU must execute 6 I/O instructions to facilitate a multiply/divide operation. The "DMA method" requires only one I/O instruction, leaving more unused I/O mapping space for larger systems.

Speed Comparison

Table 1 shows a comparison of multiply time (not including loading and unloading the stack), and system throughput frequency (including loading and unloading the stack) for 4 different multiply methods.

* 1. The "software method" using only software (no MDU).

* 2. The "normal hardware method" using the MDU addressed by the CPU.

3. The "DMA method" using the MDU as shown in Fig. 1.

4. The "theoretical minimum DMA method" using the MDU; assuming multiplication by a constant number. Special hardware would avoid reloading X and Y registers. A pulse at the $\overline{\text{CLEAR}}$ pin would replace loading the last control byte (FC).

TABLE 1

Multiply Method	Multiply Time in Micro Seconds	System Throughput Frequency
Software Only	917 *	1 KHz
Hardware MDU (normal operation)	69 *	7.8 KHz
Hardware MDU using DMA technique	29	11.4 KHz
Theoretical Minimum	21.3	14.4 KHz

Values given are for system operation at 5 volts V_{DD}, and 3 MHz clock frequency.

* Ref: Newsletter article "Using the MDU" May 1980.

Added Hardware (Fig. 1)

Both flip-flops are normally in the reset condition, causing no DMA action. A (0111) code is jammed into the CD4516, keeping the MDU de-selected (CE = 0).

When an output 2 instruction occurs, the N_1 line goes high, setting flip-flop B, which asserts a $\overline{\text{DMA-OUT}}$ signal at the CPU. Later in the output cycle, during TPB, flip-flop A is set, which removes the preset (JAM) signal from the CD4516. The CD4516 merely counts up at each TPA pulse, sequencially addressing the MDU. Additional logic decodes the CD4516 output to produce the $\overline{\text{DMA-IN}}$ signal ($\overline{\text{DMA-IN}}$ has priority over $\overline{\text{DMA-OUT}}$). At the end of the address sequence, the carry out ($\overline{\text{CO}}$) restores the flip-flops to their original states; ending the DMA action, and de-selecting the MDU.

The MDU $\overline{\text{SHIFT}}$ pin is connected to the CPU $\overline{\text{WAIT}}$ pin causing a stretched CPU cycle during the multiply/divide operation, so that the MDU receives at least 18 clock pulses before the answer is read from the Z and Y registers.

Software Program

The program starts by initializing register R3, designating it as the program counter, and freeing R0 for the DMA pointer. RX is also set to R0 so that the stack can be loaded/unloaded using INP/OUT instructions.

The program enters the multiply routine, and loops continuously performing successive multiply operations. The looping scheme was chosen to demonstrate the throughput rate of the system (Fig. 2) in terms of multiplies/sec.

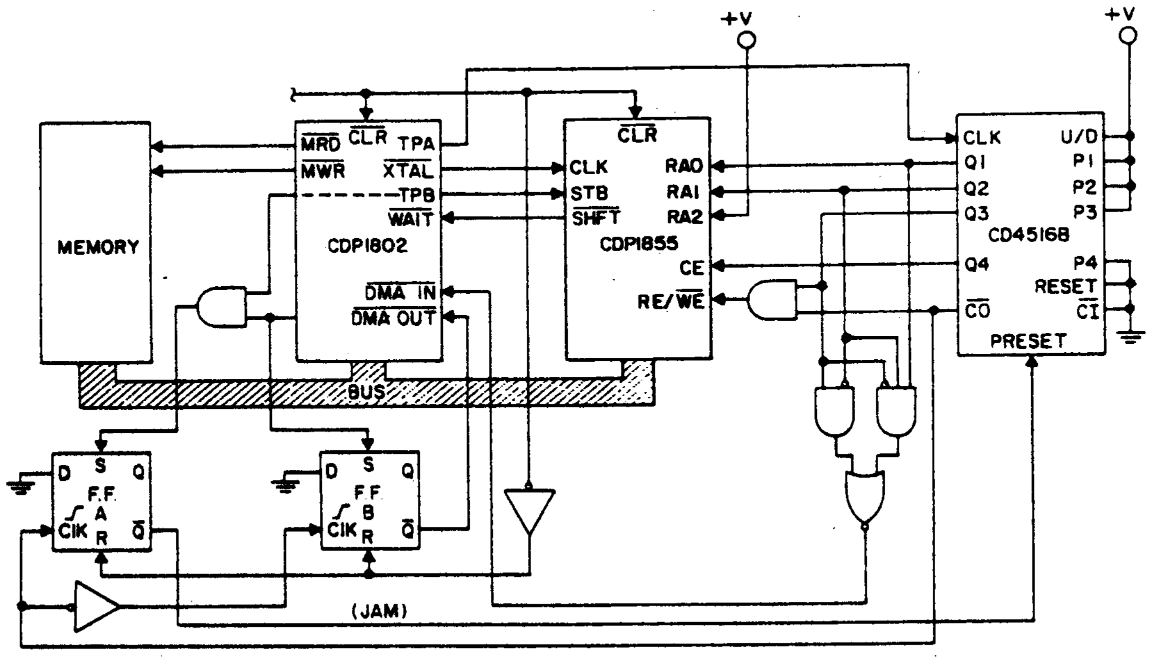
Although not shown, the software can be modified to facilitate the use of branch and interrupt techniques to get in and out of the multiply loop. With the addition of a SEP instruction, the multiply operation can become a one-shot subroutine.

For additional information contact Jerry Johnson, X6776.

PROGRAM FOR MULTIPLY USING "DMA METHOD"

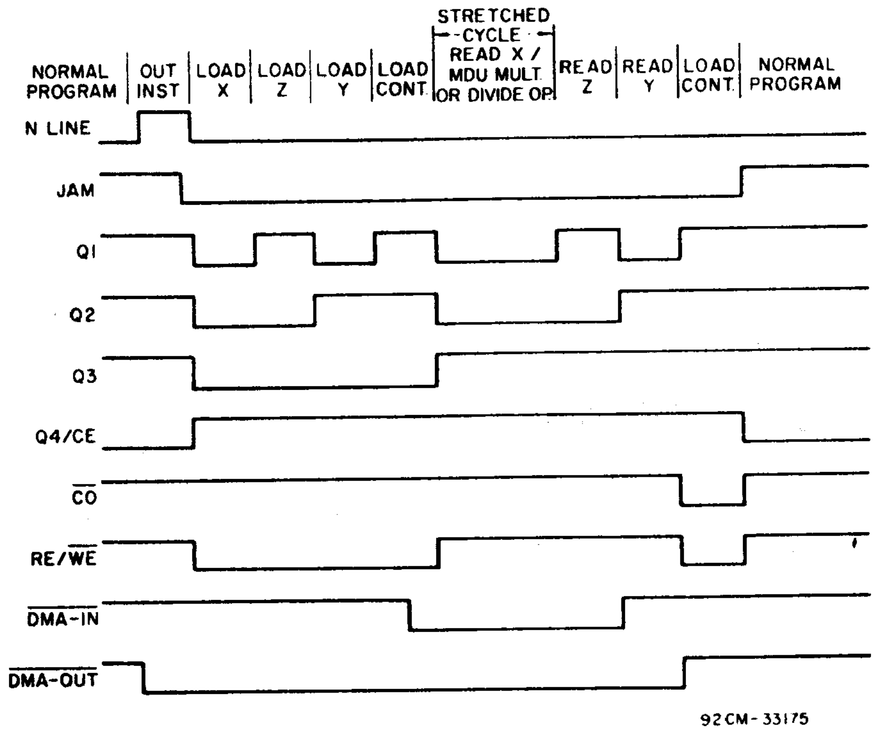
```
                                        ASSEMBLY
M ADDRESS          BYTE                 PROGRAM        COMMENTS

0000               F800                 LDI 00         RO = Prog. Counter
0002               B3                   PHI R3
0003               F807                 LDI 07
0005               A3                   PLO R3         R3 = 0007
0006               D3                   SEP R3

0007               E3                   SEX RO         RO = RX
0008               F807                 LDI 07
000A               AO                   PLO RO
000B               F802                 LDI 02
000D               BO                   PHI RO

000E               F8FC                 LDI FC         Put FC in Stack
0010               50                   STR RO         at 0207
0011               F803                 LDI 03
0013               AO                   PLO RO
0014               F8F1                 LDI F1         Put F1 in Stack
0016               73                   STXD           at 0203

0017               F800                 LDI 00         Put 00 in Stack
0019               73                   STXD           at 0202


001A               F801       routine,  LDI 01
001C               AO                   PLO RO
001D               69                   INP 9          Port A data to Stack 0201
001E               20                   DEC RO
001F               6A                   INP C          Port B data to Stack 0200

0020               20                   DEC RO
0021               62                   OUT 2 #        Initiate DMA and multiply
0022               F805                 LDI 05         action
0024               AO                   PLO RO
0025               61                   OUT 1          STACK 0205 to Port C
0026               64                   OUT 4          Stack 0206 to Port D
0027               301A                 BR routine


01FF  \                                                Dummy byte
0200   |                                               Load X register
0201   /                                               Load Z register
0202  <    STACK                                       Load Y register
0203   \                                               Load control register
0204   |                                               Read X register
0205   |                                               Read Z register
0206   |                                               Read Y register
0207  /                                                Load control register
```

# Only one instruction required to activate a complete MDU operation



92CM-33174

Figure 1



TIMING DIAGRAM

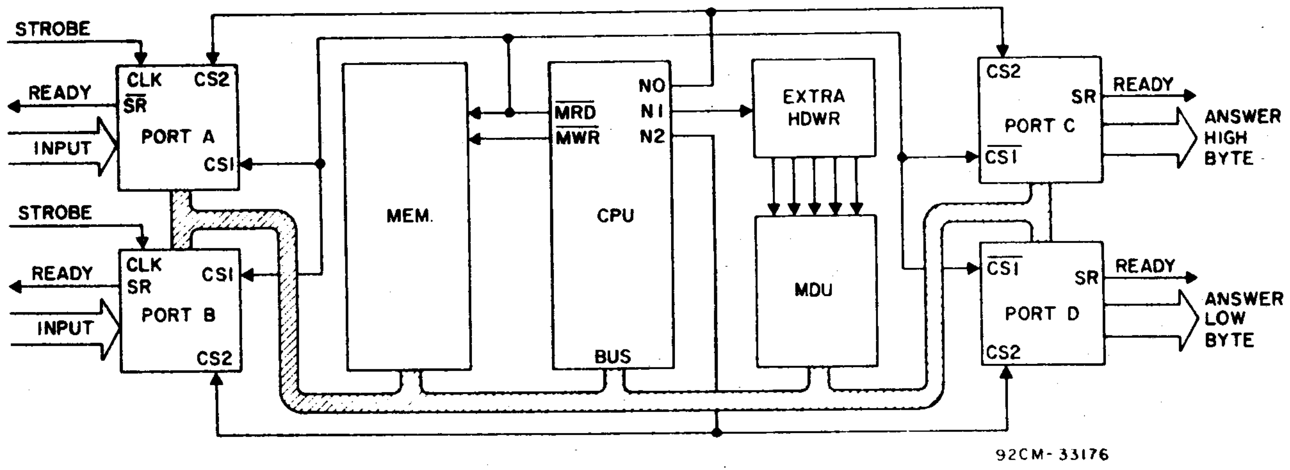


Figure 2

Digital Filter Using the CDP1855

Figure 1 shows a simple low-pass filter composed of a resistor and a capacitor. The output voltage can be described by the differential equation:

1. $$\frac{dVO}{dt} + \frac{VO(t)}{RC} = \frac{VI(t)}{RC}$$

by using Euler's method of sampling integration, equation 1 is transformed to:

2. $V_{O(n)} = kV_{I(n)} + m\ V_{O(n-1)}$; where n is any specific sample period,

$$k = \frac{1}{1 + RC/T},\ \text{and}\ m = \frac{1}{1 + T/RC}$$

T is the sampling time interval in seconds, R is in ohms, and C is in farads.

Equation 2 can be implemented with a recursive digital filter that uses multiply, add, and delay functions as shown in Fig. 2. Fig. 3 shows two multiply operations of the CDP1855. The second operation adds the result of the first operation to its product. The feedback delay is obtained by a read-in and then read-out of a scratch pad memory cell.

The program listing shows a routine used to do real time recursive digital filtering, where the incoming analog signal is digitized by an A/D converter, processed by the digital filter, and reconstructed by a D/A converter (Fig. 4). The beginning of the program initializes CPU registers, and loads constant data into the STACK area. The program then moves into a continuous loop of data input, data processing (MDU action), and data output.

The sample rate (T) is the time it takes the program to execute the routine loop of 15 instructions.

If the CPU clock is 3.2 MHz, the time required to complete one loop is 75 usec, or a sampling frequency of about 13 KHz. Based on the Nyquist criterion of at least two samples per cycle, the input signal frequency should be limited to 6.5 KHz.

Example

Assume that Fig. 1 has R = 30K Ohms, and C = .1 uF, and that T = 75 usec;

then $k = \frac{1}{1 + 300/75} = .2 = .00110011$ Binary = 33 hex,

and $m = \frac{1}{1 + 75/300} = .8 = .11001100$ Binary = CC hex.

Assume now that the input signal goes from 0 to a full scale voltage of 1 volt, and remains at 1 volt for 10 sample periods. Using equation 2 the corresponding output voltages are listed below, and if plotted would produce a response curve similar to that of the analog RC circuit.

Sample Time n	0	1	2	3	4	5	6	7	8	9	10
Input Voltage V_I	0	1	1	1	1	1	1	1	1	1	1
Output Voltage V_0	0	.20	.36	.49	.59	.67	.74	.79	.83	.86	.90

For additional information contact Jerry Johnson, X6776.

```
                                 Assembly
M. Address              Byte     Program          Comments

0 0 0 0                 F8 04    LDI 04           RP = R0
0 0 0 2                 AB       PLO RB
0 0 0 3                 F8 05    LDI 05
0 0 0 5                 AA       PLO RA
0 0 0 6                 F8 01    LDI 01

0 0 0 8                 BA       PHI RA           RA = 0105
0 0 0 9                 BB       PHI RB           RB = 0104
0 0 0 A                 EA       SEX A            RX = RA
0 0 0 B                 F8 FD    LDI FD           Load data
0 0 0 D                 73       STXD             in Stack 0105

0 0 0 E                 F8 00    LDI 00           Initialize storage space
0 0 1 0                 73       STXD             data to zero (0104)
0 0 1 1                 F8 CC    LDI CC           Load data
0 0 1 3                 73       STXD             in stack 0103
0 0 1 4                 F8 F9    LDI F9           Load data
0 0 1 6                 73       STXD             in stack 0102

0 0 1 7                 73       STXD             Decrement RX again
0 0 1 8                 F8 33    LDI 33           Load data
0 0 1 A                 5A       STR A            in stack 0100

0 0 1 B Routine )       64       OUT 4            Load X register
0 0 1 C                 69       INP 9            READ V_I from A/D
0 0 1 D                 66       OUT 6            Load Z register
0 0 1 E                 67       OUT 7            Load control register
0 0 1 F                 7A       REQ              Dummy instruction during multiply

0 0 2 0                 64       OUT 4            Load X register
0 0 2 1                 66       OUT 6            Load Z register
0 0 2 2                 67       OUT 7            Load control register
0 0 2 3                 7A       REQ              Dummy instruction during multiply
0 0 2 4                 6A       INP A            READ Y register

0 0 2 5                 5B       STR B            STORE in stack at 0104
0 0 2 6                 61       OUT 1            Load D/A with same data
0 0 2 7                 F8 00    LDI 00
0 0 2 9                 AA       PLO RA           Set RX back to 0100
0 0 2 A                 30 1B    BR Routine       Branch to "Routine"



0 1 0 0                 33      \                 K constant value
0 1 0 1                 (data)   )                V_I data from A/D
0 1 0 2                 F9       (  STACK         Control byte, reset Y and multiply
0 1 0 3                 CC       (  AREA          m constant value
0 1 0 4                 (data)   \                Storage space for result V_O
0 1 0 5                 FD       )                Control byte, multiply and add Y
```

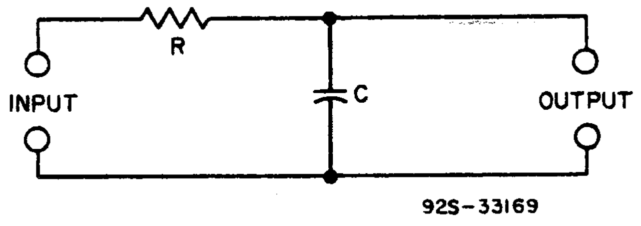


Figure 1

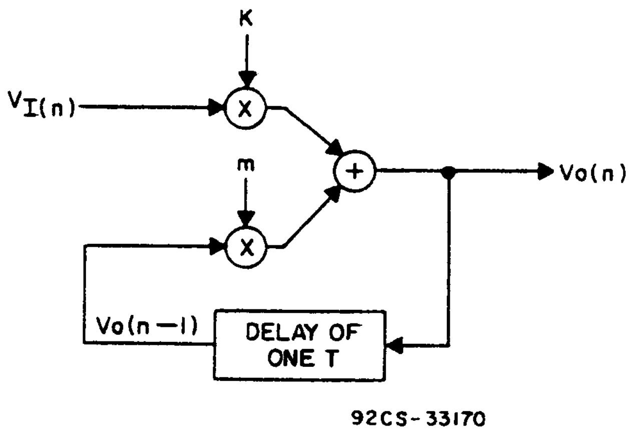


Figure 2

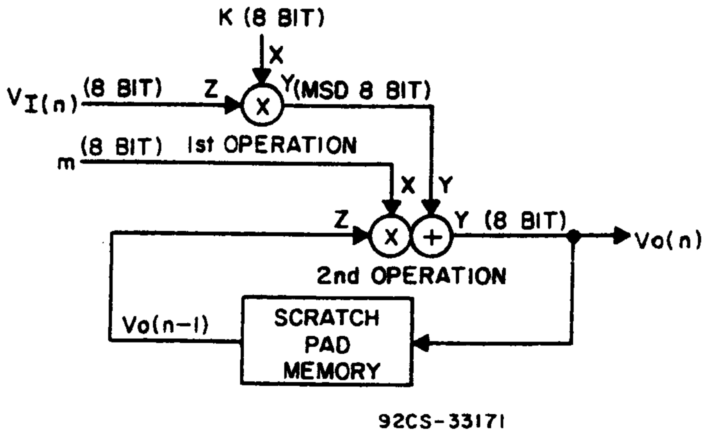


Figure 3

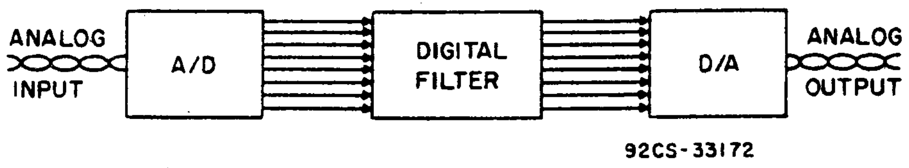


Figure 4

RCA

Solid State Division

Microsystems Products Application Note ICAN-6953

An Introduction to the Video Interface System (VIS) Devices - CDP1869 and CDP1870

By W. H. Schilp

This Note describes a circuit and the software required to mate the RCA-CDP1869 and CDP1870 VIS (Video Interface System) chip set[1] to the Evaluation Kit, CDP18S020. The capabilities of the VIS chip set are demonstrated by employing the set in the video portion of an intelligent terminal. Also included in the Note is the circuitry for a CPU controller which, combined with the video board, permits implementation of a stand-alone video output from serial ASCII input. The program shown can also be employed with a Microboard system using the VIS Microboard CDP18S661A.[2] The VIS circuitry can be constructed on a 44-pin, 4.5 by 7.5-inch PC board, which will mate to the P-2 connector on the Evaluation Kit.

The sample software program displays ASCII encoded alphanumeric characters and symbols, responds to line feed and carriage return, and uses control characters to move the cursor up, down, right, left and home, and to clear the screen. The program will operate from a ROM or PROM with a 6-byte RAM stack outside the program area.

THE VIDEO INTERFACE SYSTEM

The RCA VIS chip set, CDP1869 and CDP1870, is a two-chip system that contains all of the circuitry necessary to generate the video signals needed to drive a monitor in color or black and white. Sound output is also available.

THE VIS CIRCUIT

The video-output circuit of which the CDP1869 and CDP1870 are a part, is shown in Fig. 1. It is recommended that the video driver transistor and the crystals be placed close to the CDP1870. The CMOS parts should be handled in accordance with recommended procedures.[3] The numbers, P2-1, etc., in Fig. 1, refer to the Evaluation Kit connector P2. The MWS5114's and the 2758 must have access times less than 300 nanoseconds for the memory to access properly.

THE CONTROLLER CIRCUIT

The controller, Fig. 2, consists of a CDP1802 Microprocessor, a 2758 EPROM for program storage, a CDP1824 RAM stack, and a CDP1852 for high-order address latching. The data is entered through the CDP1854 UART. The CDP1863 counter generates the clock frequency for the BAUD rate. As shown, the counter is set for 300 BAUD, but it can be rewired to generate other rates. It is recommended that the rate be limited to 1200 BAUD or that suitable hardware be added between UART and I/O device to signal the input device that a new character can be entered. The comments on wiring and handling mentioned above apply here also. The connections P-1, etc., mate with those of the VIS circuit (Fig. 1).

MODIFICATIONS TO EVALUATION KIT

The VIS chip set is memory mapped into memory locations F800 through FFFF. To separate the FXXX memory locations for the VIS chip set, the following changes must be made to the Evaluation Kit, Fig. 3:

1. On the top side of the board:
   - Under U6, cut the connection between pins 11 and 12.
   - Cut the line from U23/15 to U6/8.
2. On the bottom side of the board:
   - Cut line from U6/9 and U22/15 at U22/15.
3. Add a CD4011 in the user area and make the following circuit connections:
   - U23/15 to CD4011/1.
   - CD4011/3 to U22/15.
   - U6/1 to CD4011/2.
   - U21/12 to CD4011/5
   - U3/21 to CD4011/6.
   - CD4011/4 to U6/8.
   - Don't forget V_{CC} and ground connections.

Printed in USA/5/81

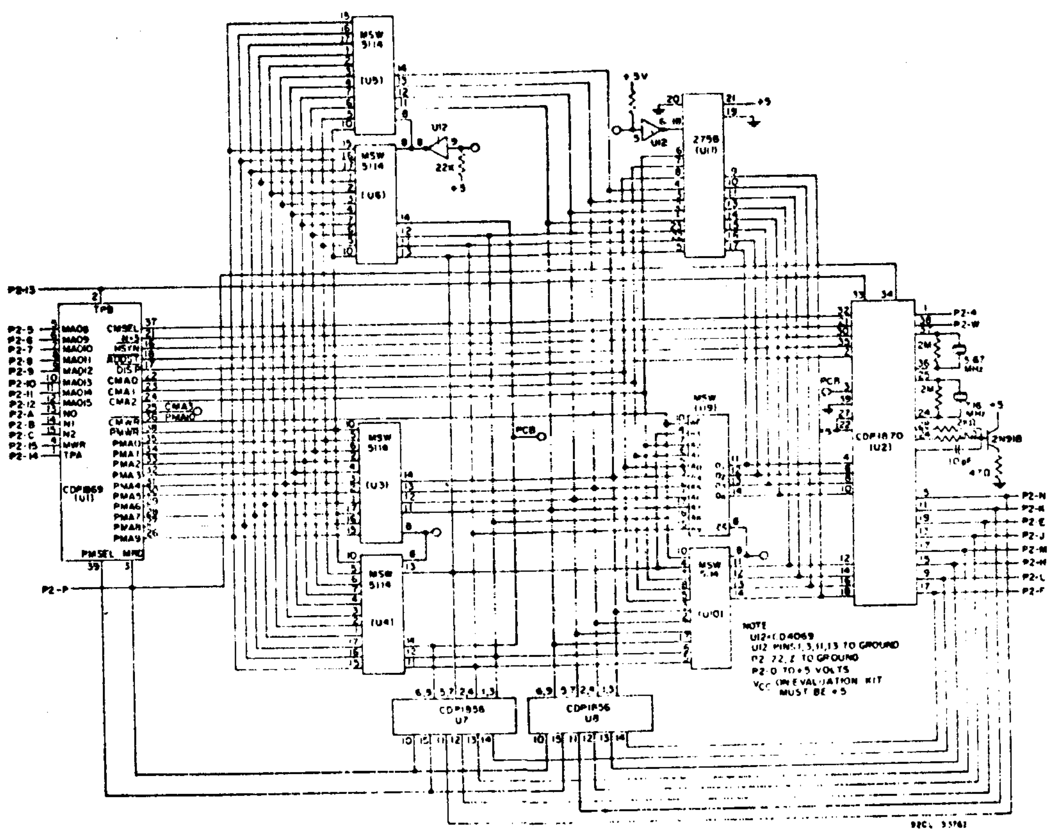


Fig. 1 - Video output circuit employing the CDP1869 and the CDP1870.

OPTIONS FOR THE VIS CIRCUIT

The following represent options for the VIS circuit of Fig. 1.

Page Memory

For two pages of page memory:

- Connect U1/25 to U12/9 and U3/8 and U4/8.

For one page of page memory:

- Do not use U5 or U6. Connect U3/8 and U4/8 to ground.

NOTE: In the two-page mode, the double-page bit must be set high.

Character Memory

The VIS circuit of Fig. 1 is designed to be used in any of three character-memory configurations: up to 64 characters in RAM, up to 64 characters in ROM, or the combination of the two, the 128-character mode. If the RAM-only configuration is used, U11 is not used and U4/14 and U6/14 are connected to U2/3; U9/8 and U10/8 are connected to ground. In the ROM-only version, U9 and U10 are not used, and U4/14 and U6/14 are connected to U2/3. In these configurations, the system can be programmed for all eight color combinations.

In the 128-character memory version, U4/14 and U6/14 are connected to U12/5, U9/8 is connected to U10/8, and U2/3 is connected to ground. In this mode, the PCB (page color bit) is used to select the RAM (PCB at ground) or ROM (PCB at V_{CC}). In the 128-character mode, only the color combinations available with the character color bits can be used. In black-and-white operation only, the 7.16-MHz crystal and the capacitor on U2/26 are not needed.

NOTE: In this system, EF1 is used during screen refresh to determine whether the display is active.

SOFTWARE PROGRAM

The program example given in the Appendix is one that initializes the system, loads the bit patterns into memory, clears the screen, homes the cursor, and waits for a character input. Each time a displayable character is input, it is displayed in the position of the cursor, which is then incremented one location to the right. If the cursor was on the right margin, it moves down one row and to the left margin. Automatic scrolling is accomplished each time the cursor is indexed past the lower right corner of the screen or above the top displayed line.

The program shown in the Appendix is written for the Evaluation Kit. To utilize the program with the controller board of Fig. 2, the routine that inputs the ASCII characters must be changed to that shown in Fig. 4. In addition, for a Microboard system, bytes

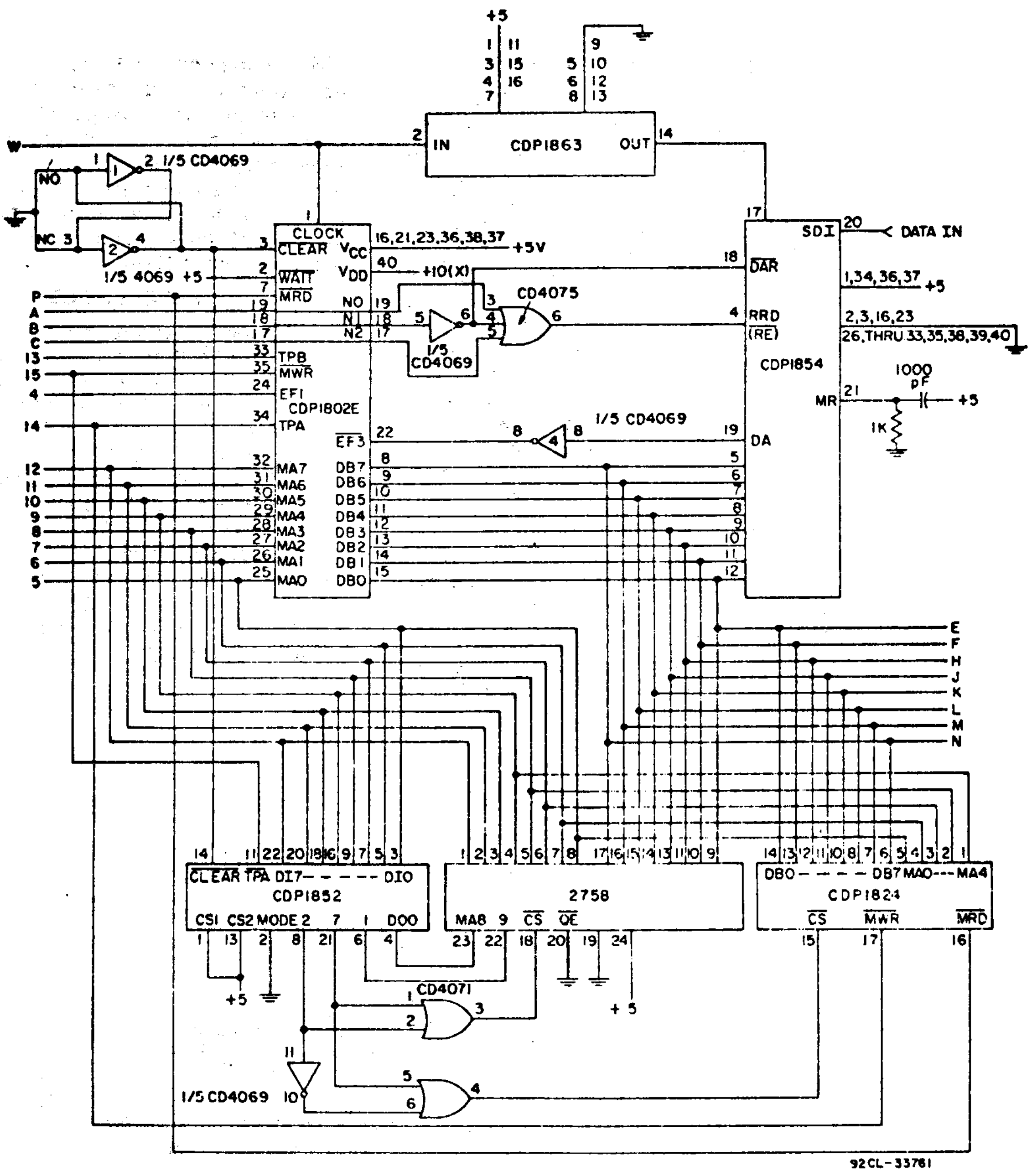


Fig. 2 - A CPU controller for the VIS that permits implementation of a stand-alone video output from serial ASCII input.

0002 and 0003 must be changed to 6180 to enable the two-level I/O, and the program will run as described. A CPU board such as the CDP18S601 is also required; either its on-board memory or a separate memory card must be available to the program at locations 0000 through 03FF. This memory area can be supplied by a RAM, ROM or PROM. A stack is required at locations 0600 through 0606; a CDP18S640 control and display card equipped with the appropriate utility ROM must also be used. The Prototyping Kit, CDP18S691, contains all that is required with the exception of the VIS board.

Program Information

Register Usage

R(2) Stack pointer
R(3) Utility program counter
R(4) Stores home address
R(5) Main program counter
R(7.0) Character counter for storing bit patterns
R(8.0) Character row counter for storing bit patterns, then ASCII value storage for previously stored character
R(9) CMEM location pointer for bit pattern storage, then column counter
R(A) Page memory location pointer
R(B) Bit pattern pointer
R(F.1) Storage for ASCII input from utility program

RUNNING THE PROGRAM

To run the program with either the Evaluation Kit or Microboard System, depress RESET, RUN U, and then the carriage return to set up the timing and registers; then depress either RESET, RUNP or enter $P0. Each time an ASCII printable character is input, it is displayed in the position of the cursor, which, again, is incremented one position. Line feed, carriage return, cursor up (control O), cursor left (control N), and cursor right (control L), all move the cursor as indicated and restore the previously displayed character to the position from which the cursor came. Clear screen (control K) clears all characters and homes the cursor. Control H causes the screen to scroll down, and Control I causes it to scroll up.

To run the program with the stand-alone system, depress the momentary contact switch; the program will come up running and waiting for an input.

This Note is meant to describe the circuitry and software relating to the CDP1869/70 VIS chip set and to introduce the reader to its potential. Therefore, the program example given in the Appendix is written to be instructive, and as such is not memory or speed efficient. However, an understanding of the program and the VIS data sheet will allow one to utilize more of the many features of the set, such as color, sound, graphics, and motion.

REFERENCES

1. "COS/MOS Video Interface System," RCA Solid State File No. 1197 on the CDP1869C/CDP1870C or "COS/MOS Memories, Microprocessors, and Support Systems," RCA Solid State Databook SSD-260.
2. "RCA COSMAC Microboard Video-Audio-Keyboard Interface CDP-18S661V1 and CDP18S661V3," RCA Solid State publication MB-661.
3. "Guide to Better Handling and Operation of CMOS Integrated Circuits," RCA Solid State Application Note ICAN-6525.

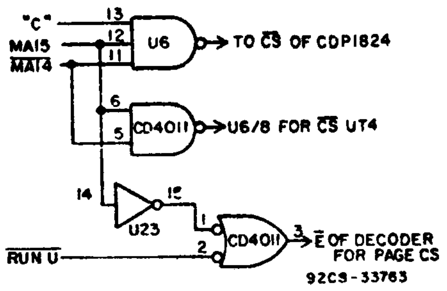

"C"
MA15
$\overline{\text{MA14}}$
13
12
11
U6
TO $\overline{\text{CS}}$ OF CDP1824
6
5
CD4011
U6/8 FOR $\overline{\text{CS}}$ UT4
14
15
U23
1
2
CD4011
3
$\overline{\text{RUN U}}$
$\overline{\text{E}}$ OF DECODER FOR PAGE CS
92CS-33763

Fig. 3 - Schematic diagram of circuit used to modify the Evaluation Kit to separate the FXXX memory locations for the VIS chip set.

```
0068 EE;          0120 REEN: SEX E           ..GET RID OF X
0069 3E69;        0121       BN3 *           ..WAIT FOR DATA AVAILABLE FROM UART
006B 6ABF;        0122       INP 2; PHI AIN  ..STORE IT IN REG F
006D C4C4;        0123       NOP; NOP        ..WASTED SPACE
```

Fig. 4 - Modification of the program shown in the Appendix to make it usable with the Microboard system of Fig. 2.

Appendix - Program Used with the Evaluation Kit

```
!M
0000 ;
0000 ;         ..**********************************************************************..
0000 ;         ..      PROGRAM FOR THE VIS TO EVALUATION KIT BOARD                     ..
0000 ;         ..      ASSUMES THE CHARACTER BIT PATTERNS ARE AT 0200 TO 03FF          ..
0000 ;         ..**********************************************************************..
0000 ;
0000 ;         ..**********************************************************************..
0000 ;         ..                             DEFINE CONSTANTS                         ..
0000 ;         ..**********************************************************************..
0000 ;
0000 ;         READ=#813B               ..ADDRESS OF READ ROUTINE IN UTILITY
0000 ;         STK=2                    ..REG 2 IS THE STACK POINTER
0000 ;         UTCT=3                   ..REG 3 IS THE PROGRAM COUNTER FOR UT4
0000 ;         HA=6                     :.STORE THE HOME ADDRESS IN REG 6
0000 ;         PPTR=#0A                 ..PAGE POINTER IS REG A
0000 ;         CTR=5                    ..COUNTER REG FOR THIS PROGRAM IS REG 5
0000 ;
0000 7100;     DIS,#00                  ..DISABLE THE INTERUPTS
0002 C4C4;     NOP;NOP                  ..LEAVE SPACE FOR 2 LEVEL I/O FOR THE MICROBOARD
0004 ;
0004 F800B5;.  LDI A.1(INIT); PHI CTR   ..SET UP REG 5
0007 F80BA5;   LDI A.0(INIT); PLO CTR
000A D5;       SEP CTR                  ..START COUNTING IN REG 5
000B ;
000B E5;       INIT: SEX CTR            ..SET X FOR OUT IMMED
000C 6380;           OUT 3; ,#80        ..TURN ON DISPLAY, SET UP 40 CHARACTERS
000E F880BA;   LDI #80; PHI PPTR        ..SET CMEM HIGH, NOISE OFF, FULL RES VERT, NTSC
0011 F8C9AA;   LDI #C9; PLO PPTR        ..DBL PG BIT FOR 1ST PG OF MEM
0014 EA65;     SEX PPTR; OUT 5          ..SET X TO PG PTR, OUT 5 SETS THE BITS
0016 ;
0016 ;         CHST=7                   ..REG 7 HAS NUMBER OF CHARACTERS STORED
0016 ;         ROCT=8                   ..REG 8 IS THE ROW COUNTER WITHIN EACH
0016 ;                                  ..CHARACTER BIT PATTERN
0016 ;         CMEM=9                   ..REG 9 POINTS TO CHARACTER MEMORY
0016 ;         CPAT=#0B                 ..REG B POINTS TO CHARACTER PATTERN
0016 ;
0016 F8F8BA;   LDI #F8; PHI PPTR        ..POINT THE PAGE POINTER AND HOME ADDRESS TO
0019 B6;       PHI HA                   ..F800
001A F8F4B9;   LDI #F4; PHI CMEM        ..POINT THE CHARACTER MEM AT F400
001D F800AB;   LDI #00; PLO CPAT        ..PUT 00 IN LOW BYTE OF ALL NEEDED REG'S
0020 AAA9;     PLO PPTR; PLO CMEM
0022 ABA7;     PLO ROCT; PLO CHST
0024 A6;       PLO HA
0025 ;
0025 F802BB;   LDI #02; PHI CPAT        ..CHARACTER PATTERN IS AT 0200
0028 ;
0028 F806B2;   LDI #06; PHI STK         ..STACK IS AT 0606
002B A2;       PLO STK
002C ;
002C 672A;     OUT 7; DEC PPTR          ..SET THE HOME ADDRESS
002E 662A;     OUT 6; DEC PPTR          ..SET THE PAGE MEMORY POINTER
0030 ;
0030 ;
0030 ;         ..************************************************************************..
0030 ;         ..                        CHARACTER LOAD ROUTINE                          ..
0030 ;         ..************************************************************************..
0030 ;
0030 87;       CHLD: GLO CHST           ..GET THE ADDRESS OF THE 1ST CHARACTER PATTERN
0031 3431;           B1*                ..WAIT FOR NON-DISPLAY
0033 5A;             STR PPTR           ..STORE IT IN PAGE MEMORY
0034 4B;       BTLD: LDA CPAT           ..GET THE FIRST ROW OF THE 1ST CHARACTER BIT PATTERN
0035 3435;           B1 *               ..WAIT FOR NON-DISPLAY


0037 59;             STR CMEM           ..STORE IT IN CHARACTER MEMORY
0038 ;
0038 1918;     INC CMEM; INC ROCT       ..INCREMENT TO NEXT ROW OF 1ST CHARACTER BIT PATTERN
003A ;                                  ..AND KEEP A COUNT ON THE NUMBER OF ROWS
003A 88FB08;   GLO ROCT; XRI #08        ..SEE IF PUT IN 8 ROWS OF BIT PATTERNS
003D 3A34;     BNZ BTLD                 ..IF NOT GO BACK FOR ANOTHER
003F ;
003F F800A8;   LDI #00; PLO ROCT        ..RESET THE ROW COUNT TO 0
0042 17;       INC CHST                 ..INCREMENT FOR THE NEXT CHARACTER BIT PATTERN
0043 87FB40;   GLO CHST; XRI #40        ..SEE IF STORED ALL 64 CHARACTER PATTERNS
0046 3A30;     BNZ CHLD                 ..IF NOT GO BACK
0048 ;
0048 ;                         ..BIT PATTERN ARE NOW STORED IN CHARACTER MEMORY
0048 ;
0048 F880BA;   LDI #80; PHI PPTR        ..TURN OFF THE CHARACTER MEMORY ACCESS
004B F8C8AA;   LDI #C8; PLO PPTR        ..MODE
004E 65;       OUT 5
004F ;                         ..NOW WE CLEAR THE SCREEN
004F ;
004F F8FFBA;   LDI #FF; PHI PPTR        ..POINT THE PAGE POINTER AT THE TOP
0052 AA;       PLO PPTR                 ..OF THE PAGE MEMORY
0053 F800;     CLPG: LDI #00            ..GET THE CHARACTER MEMORY ADDRESS FOR A SPACE
0055 3455;           B1 *               ..WAIT
0057 73;             STXD               ..STORE IT IN THE PAGE MEMORY AND POINT TO NEXT
```

```
0058 ;                                     ..LOWER PAGE MEM LOCATION
0058 9AFBF7;              GHI PPTR; XRI #F7 ..SEE IF REACHED THE BOTTOM OF PAGE MEMORY
005B 3A53;                BNZ CLPG          ..IF NOT GO BACK
005D ;
005D 1A;           INC PPTR                 ..MOVE THE PAGE POINTER TO 1ST LOCATION THAT IS
005E ;                                      ..UPPER LEFT CORNER OF THE SCREEN
005E ;
005E ;
005E ;             ..*****************************************************************..
005E ;             ..                         INITALIZATION COMPLETE                  ..
005E ;             ..                NOW WE ARE READY TO BRING IN CHARACTERS          ..
005E ;             ..*****************************************************************..
005E ;
005E ;
005E ;             AIN=#0F                  ..REG F WILL CONTAIN THE ASCII BYTE INPUTTED
005E ;             TSTR=8                   ..REG 8 WILL BE NEEDED FOR TEMPORARY STORAGE
005E ;             COCT=9                   ..REG 9 WILL KEEP COUNT OF THE COLUMN IN WHICH THE
005E ;                                      ..CURSOR IS
005E ;
005E F800A9;       LDI #00; PLO COCT        ..ZERO OUT THE COLUMN COUNTER
0061 ;
0061 ;
0061 ;             ..*****************************************************************..
0061 ;             ..                THIS ROUTINE SAVES THE CHARACTER IN THE NEW LOCATION ..
0061 ;             ..                AND DISPLAYS THE CURSOR THERE                    ..
0061 ;             ..*****************************************************************..
0061 ;
0061 ;
0061 0AA8;         NEWC: LDN PPTR; PLO TSTR..PUT THE CHARACTER INTO TEMPORARY STORAGE
0063 F83F;                LDI #3F           ..LOAD THE CMEM ADDRESS OF THE CURSOR
0065 3465;                B1 *              ..WAIT
0067 5A;                  STR PPTR          ..DISPLAY THE CURSOR
0068 ;
0068 F838A3;       REEN: A.0(READ)->UTCT.0 ..LOAD THE ADDRESS OF READ ROUTINE INTO
006B F881B3;             A.1(READ)->UTCT.1 ..UTILITY'S PROGRAM COUNTER
006E D3;                  SEP UTCT          ..DO THE READ
006F ;
006F EA;                  SEX PPTR          ..RESET X
0070 9F;                  GHI AIN           ..GET THE ASCII CHARACTER FROM REG F
0071 FA60C2010A;          ANI #60; LBZ CTLC ..SEE IF BITS 5 AND 6 ARE LOW, IF SO MUST BE A
0076 ;                                      ..CONTROL CHARACTER SO GO TO THAT ROUTINE
0076 FB603268;            XRI #60; BZ REEN  ..SEE IF BITS 5 AND 6 ARE HIGH, IF SO WE DON'T
007A ;                                      ..NEED IT SO GO BACK FOR A NEW ENTRY
007A ;
007A ;                             ..WE MUST HAVE A DISPLAYABLE CHARACTER SO DO IT
007A ;
007A 9F;           GHI AIN                  ..GET THE ASCII VALUE
007B FF20;         SMI #20                  ..SUBTRACT HEX 20 TO MAKE IT EQUAL TO ITS
007D ;                                      ..CHARACTER MEMORY ADDRESS
007D 347D;         B1 *                     ..WAIT
007F 5A;           STR PPTR                 ..DISPLAY IT
0080 ;
0080 ;
0080 ;             ..*****************************************************************..
0080 ;             ..                REMAINDER OF THE PROGRAM TAKES CARE OF THE POSITION ON ..
0080 ;             ..                THE SCREEN FOR THE NEXT CHARACTER                ..
0080 ;             ..                AND THE SCROLLING                                ..
0080 ;             ..*****************************************************************..
0080 ;
0080 ;
0080 ;                             ..HAS THE PAGE POINTER REACHED END OF LAST LINE ?
0080 ;
0080 ;             LLCK:
0080 8AFF7F;       GLO PPTR; SMI #7F        ..SEE IF THE PAGE POINTER IS AT THE LAST DISPLAYABLE
0083 3A91;         BNZ INCR                 ..LOCATION IF NOT WE CAN GO AND INCREMENT IT
0085 9AFFFF;       GHI PPTR; SMI #FF
0088 3A91;         BNZ INCR
008A ;
008A AAA9;         PLO PPTR; PLO COCT       ..MUST HAVE BEEN AT LAST LOCATION SO RESET PAGE
008C F8F8BA;       LDI #F8; PHI PPTR        ..POINTER TO BEGINNING THAT IS F800
008F 3099;         BR USCK                  ..GO TO CHECK IF WE MUST SCROLL UP
0091 ;
0091 ;                             ..INC PG PTR AND CHECK POSITION
0091 ;
0091 1A;           INCR: INC PPTR           ..INCREMENT PAGE POINTER AND COLUMN COUNTER
0092 19;                 INC COCT           ..TO NEXT LOCATION
0093 89;                 GLO COCT           ..SEE IF WE REACHED THE 40TH COLUMN
0094 FF28;               SMI #28
0096 3A61;               BNZ NEWC           ..IF NOT GO GET ANOTHER CHARACTER
0098 ;
0098 A9;           PLO COCT                 ..MUST HAVE BEEN 40TH COLUMN SO ZERO OUT COLUMN
0099 ;                                      ..COUNTER
0099 ;
0099 ;
0099 ;             ..*****************************************************************..
0099 ;             ..                         SCROLL UP ROUTINE                        ..
0099 ;             ..*****************************************************************..
0099 ;
0099 ;                             ..SCROLL UP CHECK
0099 ;
0099 E2;           USCK: SEX STK            ..POINT X TO STACK
009A 86FFC0;             GLO HA; SMI #C0    ..SEE IF PRESENT HOME ADDRESS IS IN SECOND
009D 967FF8;             GHI HA; SMBI #F8   ..PAGE OF PAGE MEMORY
```

```
00A0 33AC;              BDF SCPG            ..IF SO GO TO SECOND PAGE ROUTINE
00A2 ;
00A2 ;                              ..1ST PAGE ROUTINE
00A2 ;
00A2 86FCC073;     GLO HA; ADI #C0; STXD   ..STORE AN OFFSET HOME ADDRESS ON THE STACK
00A6 967C03;       GHI HA; ADCI #03        ..WHICH IS THE ADDRESS OF THE NEXT LOCATION
00A9 52;           STR STK                 ..JUST BELOW THE PRESENTLY DISPLAYED SCREEN
00AA 30BE;         BR PPCK                 ..GO TO THE PAGE POINTER CHECK
00AC ;


00AC ;                                ..2ND PAGE ROUTINE
00AC ;
00AC 8652;         SCPG: GLO HA; STR STK   ..SUBTRACT THE HOME ADDRESS FROM THE PAGE
00AE 8AF7;               GLO PPTR; SM      ..POINTER TO SEE IF THE PAGE POINTER IS STILL
00B0 9652;               GHI HA; STR STK   ..ON SCREEN
00B2 9A77;               GHI PPTR; SMB
00B4 3361;               BDF NEWC          ..IF SO CAN GET A NEW CHARACTER
00B6 ;
00B6 86FFC0;             GLO HA; SMI #C0   ..MIGHT BE OFF SCREEN SO STORE THE OFFSET
00B9 73;                 STXD              ..HOME ADDRESS ON THE STACK
00BA 967F03;             GHI HA; SMBI #03
00BD 52;                 STR STK
00BE ;                              ..PAGE PTR CHECK
00BE ;
00BE 12;           PPCK: INC STK           ..SUBTRACT THE OFFSET HOME ADDRESS FROM
00BF 8AF7;               GLO PPTR; SM      ..THE PAGE POINTER
00C1 22;                 DEC STK
00C2 9A77;               GHI PPTR; SMB
00C4 12;                 INC STK
00C5 EA;                 SEX PPTR          ..RESET X TO THE PAGE POINTER
00C6 3B61;               BL NEWC           ..IF PAGE POINTER STILL ON SCREEN GO BACK
00C8 ;                                     ..FOR A NEW CHARACTER
00C8 ;
00C8 ;                              ..WE WILL HAVE TO SCROLL
00C8 ;                              ..BUT FIRST CLEAR THE NEXT LINE
00C8 ;
00C8 ;             TEMP=7                  ..NEED TEMPORARY STORAGE
00C8 89A7;         GLO COCT; PLO TEMP      ..FOR THE COLUMN COUNT
00CA 8932D1;       ZERO: GLO COCT; BZ CLLN ..MOVE THE PAGE POINTER TO THE LEFT MARGIN
00CD 292A;               DEC COCT; DEC PPTR
00CF 30CA;               BR ZERO
00D1 ;
00D1 F800;         CLLN: LDI #00           ..GET THE CHEM ADDRESS OF A SPACE
00D3 34D3;               B1*               ..WAIT
00D5 5A;            .    STR PPTR          ..STORE IT IN PAGE MEMORY
00D6 1A19;               INC PPTR; INC COCT..MOVE OVER ONE
00D8 89FF28;             GLO COCT; SMI #28 ..SEE IF WE DID THE WHOLE LINE
00DB 3AD1;               BNZ CLLN          ..IF NOT GO BACK
00DD ;
00DD 2A;           REST: DEC PPTR          ..RESET THE PAGE POINTER TO THE BEGINNING
00DE 29;                 DEC COCT          ..OF THE LINE
00DF 89;                 GLO COCT
00E0 3ADD;               BNZ REST
00E2 ;
00E2 8732EA;       RSTR: GLO TEMP; BZ USCL ..REPOSITION THE PAGE POINTER AND COLUMN
00E5 27;                 DEC TEMP          ..COUNTER TO THE PROPER POSITION ON THE NEW LINE
00E6 191A;               INC COCT; INC PPTR..WHEN DONE GO TO SCROLL UP ROUTINE
00E8 30E2;               BR RSTR
00EA ;
00EA ;                                    ..SCROLL UP ROUTINE
00EA ;
00EA ;                              ..BUT FIRST SEE IF WE HAD THE LAST LINE OF THE
00EA ;                              ..SECOND PAGE ON THE BOTTOM OF THE SCREEN
00EA ;
00EA 86FF58;       USCL: GLO HA; SMI #58   ..SEE IF LAST LINE WAS ON BOTTOM
00ED 3AFB;               BNZ NLST
00EF 96FFFF;             GHI HA; SMI #FF
00F2 3AFB;               BNZ NLST          ..IF NOT GO TO NOT LAST LINE ROUTINE
00F4 A6;                 PLO HA            ..MUST HAVE BEEN LAST LINE SO SET UP
00F5 F8F8B6;             LDI #F8; PHI HA   ..HOME ADDRESS
00F8 C00103;             LBR OUTP          ..GO TO OUTPUT IT
00FB ;
00FB ;                             ..NOT LAST LINE
00FB ;
00FB 86FC28;       NLST: GLO HA; ADI #28   ..ADD HEX 28 TO THE HOME ADDRESS
00FE A6;                 PLO HA
00FF 967C00;             GHI HA; ADCI #00
0102 B6;                 PHI HA
0103 E667;         OUTP: SEX HA; OUT 7     ..SET THE NEW HOME ADDRESS
0105 26EA;               DEC HA; SEX PPTR  ..DEC THE HA CAUSE OUT INSTRUCTION INCR'S IT
0107 ;                                     ..RESET X TO THE PAGE POINTER
0107 C00061;             LBR NEWC          ..GO GET THE NEXT CHARACTER
010A ;
010A ;
010A ;             ..*******************************************************************..
010A ;             ..                 THE FOLLOWING ROUTINE LOOKS AT THE CONTROL          ..
010A ;             ..                 CHARACTERS FOR CURSOR MOVEMENT OR SCROLL            ..
010A ;             ..                 CONTROL CHARACTER ROUTINE                           ..
010A ;             ..*******************************************************************..
010A ;
010A ;
```

```
010A 88;            CTLC: GLO TSTR              ..RESTORE THE CHARACTER WHICH USED TO
010B 340B;                 B1 *                 ..BE UNDER THE CURSOR
010D 5A;                   STR PPTR
010E ;
010E 9F;            GHI AIN                     ..GET THE ASCII VALUE
010F FF0832A5;      SMI #08; BZ DSCL            ..CONTROL H IS SCROLL DOWN
0113 FF01C200EA;    SMI #01; LBZ USCL           ..CONTROL I IS SCROLL UP
0118 FF013235;      SMI #01; BZ LIFD            ..THIS IS LINE FEED
011C FF01C2000B;    SMI #01; LBZ INIT           ..CONTROL K IS CLEAR SCREEN AND HOME CURSOR
0121 FF01C20080;    SMI #01; LBZ LLCK           ..CONTROL L IS CURSOR RIGHT
0126 FF013250;      SMI #01; BZ CART            ..THIS IS CARRIAGE RETURN
012A FF0132BF;      SMI #01; BZ CULT            ..CONTROL N IS CURSOR LEFT
012E FF013258;      SMI #01; BZ CUUP            ..CONTROL O IS CURSOR UP
0132 C00061;        LBR NEWC                    ..NOT USING ANY OTHERS RIGHT NOW SO
0135 ;                                          ..GO BACK FOR NEXT CHARACTER
0135 ;
0135 ;
0135 ;              ..*******************************************************************..
0135 ;              ..                    LINE FEED ROUTINE                              ..
0135 ;              ..*******************************************************************..
0135 ;
0135 ;
0135 8AFF58;        LIFD: GLO PPTR; SMI #58 ..SEE IF CURSOR IS ON LAST LINE OF SECOND PAGE
0138 9A7FFF;              GHI PPTR; SMBI #FF
013B 3348;                BDF LILL              ..IF SO GO TO LAST LINE ROUTINE
013D ;
013D 8AFC28;        GLO PPTR; ADI #28           ..NOT ON LAST LINE SO ADD HEX 28
0140 AA;            PLO PPTR
0141 9A7C00;        GHI PPTR; ADCI #00
0144 BA;            PHI PPTR
0145 C00099;        LBR USCK                    ..AND GO CHECK TO SEE IF WE NEED TO SCROLL UP
0148 ;
0148 ;                                ..LAST LINE ROUTINE
0148 ;
0148 89AA;          LILL: GLO COCT; PLO PPTR..PAGE POINTER IS F800 PLUS THE COLUMN COUNT
014A F8F8BA;              LDI #F8; PHI PPTR
014D C00099;              LBR USCK              ..GO CHECK FOR SCROLL UP
0150 ;
0150 ;
0150 ;              ..*******************************************************************..
0150 ;              ..                  CARRIAGE RETURN ROUTINE                          ..
0150 ;              ..*******************************************************************..
0150 ;
0150 ;
0150 89;            CART: GLO COCT              ..SEE IF CURSOR IS IN LEFT MARGIN


0151 C20061;              LBZ NEWC              ..IF SO GO BACK FOR NEXT CHARACTER
0154 2A29;                DEC PPTR; DEC COCT..IT WASN'T, MOVE LEFT TILL WE GET THERE
0156 3050;                BR CART
0158 ;
0158 ;
0158 ;              ..*******************************************************************..
0158 ;              ..                  CURSOR UP ROUTINE                                ..
0158 ;              ..*******************************************************************..
0158 ;
0158 ;
0158 8AFF28;        CUUP: GLO PPTR; SMI #28 ..SEE IF CURSOR IS IN 1ST LINE OF 1ST PAGE
015B 9A7FF8;              GHI PPTR; SMBI #F8..OF PAGE MEMORY
015E 3B6A;                BNF CUFL              ..IF SO GO TO 1ST LINE ROUTINE
0160 ;
0160 ;                              ..MOVE CURSOR UP
0160 ;
0160 8AFF28;        GLO PPTR; SMI #28           ..NOT IN 1ST LINE SO SUBTRACT HEX 28
0163 AA;            PLO PPTR
0164 9A7F00;        GHI PPTR; SMBI #00
0167 BA;            PHI PPTR
0168 3087;          BR DSCK                     ..CHECK FOR SCROLL
016A ;
016A ;                                ..1ST LINE ROUTINE
016A ;
016A F8FFBA;        CUFL: LDI #FF; PHI PPTR ..PUT CURSOR IN PROPER POSITION OF LAST
016D 89FC58;              GLO COCT; ADI #58 ..LINE OF SECOND PAGE OF PAGE MEMORY
0170 AA;                  PLO PPTR
0171 ;                                          ..NOW CHECK FOR SCROLL
0171 ;
0171 ;
0171 ;              ..***************************************************************..
0171 ;              ..                  DOWN SCROLL ROUTINE                           ..
0171 ;              ..***************************************************************..
0171 ;
0171 ;                                ..DID WE GO OFF SCREEN ?
0171 ;
0171 86FB00;        DSFC: GLO HA; XRI #00   ..IS THE HOME ADDRESS F800 ?
0174 3A83;                BNZ GTCH
0176 96FBF8;              GHI HA; XRI #F8
0179 3A83;                BNZ GTCH              ..IF NOT DON'T SCROLL
017B 8AFF58;              GLO PPTR; SMI #58 ..ARE WE ON THE LAST LINE OF
017E 9A7FFF;              GHI PPTR; SMBI #FF..2ND PAGE ?
0181 33B8;                BGE DSFL              ..IF SO, SCROLL
0183 EA;            GTCH: SEX PPTR
0184 C00061;              LBR NEWC              ..GO BACK FOR MORE
0187 ;
```

```
0187 ;
0187 ;                            ..DOWN SCROLL CHECK
0187 ;
0187 E2;             DSCK: SEX STK           ..SET X TO THE STACK
0188 8AFC28;               GLO PPTR; ADI #28 ..SEE IF THE HOME ADDRESS
018B 73;                   STXD              ..IS THE NEXT LINE
018C 9A7C00;               GHI PPTR; ADCI #00
018F 5212;                 STR STK; INC STK  ..THAT IS, DOES HA=PPTR+#28-COCT ?
0191 89F5;                 GLO COCT; SD
0193 73;                   STXD
0194 F07F00;               LDX; SMBI #00
0197 5212;                 STR STK; INC STK
0199 86F7;                 GLO HA; SM
019B 3A8322;               BNZ GTCH; DEC STK
019E 96F7;                 GHI HA; SM
01A0 123A83;               INC STK; BNZ GTCH
01A3 30AF;                 BR DSNF           ..IT DOES, SO WE MUST SCROLL
01A5 ;
01A5 ;
01A5 ;                            ..SCROLL DOWN
01A5 ;
01A5 86FB00;         DSCL: GLO HA; XRI #00   ..IF THE HOME ADDRESS IS
01A8 3AAF;                 BNZ DSNF          ..F800 GO TO FIRST LINE
01AA 96FBF8;               GHI HA; XRI #F8   ..ROUTINE
01AD 32B8;                 BZ DSFL
01AF ;
01AF ;                         ..DOWN SCROLL FOR NOT 1ST LINE
01AF ;
01AF 86FF28;         DSNF: GLO HA; SMI #28   ..SUBTRACT HEX 28 FROM THE HOME ADDRESS
01B2 A6;                   PLO HA
01B3 967F00;               GHI HA; SMBI #00
01B6 3002;                 BR OUTP-1         ..GO SET THE NEW HOME ADDRESS
01B8 ;
01B8 ;
01B8 ;                      ..DOWN SCROLL FOR 1ST LINE
01B8 ;
01B8 F858A6;         DSFL: LDI #58; PLO HA   ..WE WERE ON THE LAST LINE SO NEW HOME
01BB F8FF;                 LDI #FF           ..ADDRESS WILL BE LAST LINE OF SECOND PAGE
01BD ;                                       ..OF PAGE MEMORY
01BD 3002;                 BR OUTP-1         ..GO SET THE HOME ADDRESS
01BF ;
01BF ;               ..****************************************************************..
01BF ;               ..                          CURSOR LEFT                           ..
01BF ;               ..****************************************************************..
01BF ;
01BF ;
01BF 8932C7;         CULT: GLO COCT; BZ CLLS ..SEE IF WE ARE ON THE LEFT MARGIN, IF SO
01C2 ;                                       ..GO TO CURSOR LEFT LEFT SIDE
01C2 2A29;                 DEC PPTR; DEC COCT..NOT ON LEFT MARGIN SO MOVE LEFT ONE SPOT
01C4 C00061;               LBR NEWC          ..GO FOR NEXT CHARACTER
01C7 ;
01C7 ;                          ..CURSOR ON LEFT MARGIN ROUTINE
01C7 ;
01C7 8A3ADA;         CLLS: GLO PPTR; BNZ CLNF..SEE IF CURSOR IS IN 1ST LINE OF 1ST PAGE
01CA 9AFFF8;               GHI PPTR; SMI #F8
01CD 3ADA;                 BNZ CLNF          ..IF NOT GO TO CURSOR LEFT NOT 1ST LINE
01CF F8FFBA;               LDI #FF; PHI PPTR ..MUST HAVE BEEN 1ST LINE SO CURSOR GOES IN
01D2 F87FAA;               LDI #7F; PLO PPTR ..LAST LOCATION OF SECOND PAGE OF PAGE MEMORY
01D5 F827A9;               LDI #27; PLO COCT ..MUST RESET COLUMN COUNTER TO RIGHT MARGIN
01D8 3071;                 BR DSFC           ..GO CHECK IF WE MUST SCROLL DOWN
01DA ;
01DA ;                          ..CURSOR LEFT FROM LEFT MARGIN BUT NOT 1ST LINE
01DA ;
01DA 2A;             CLNF: DEC PPTR          ..THIS WILL MOVE IT TO THE RIGHT MARGIN
01DB ;                                       ..OF THE PREVIOUS LINE
01DB F827A9;               LDI #27; PLO COCT ..RESET THE COLUMN COUNTER
01DE 3087;                 BR DSCK           ..GO CHECK FOR SCROLL DOWN
01E0 ;
01E0 ;
01E0 ;               ..****************************************************************..
01E0 ;               ..      THE FOLLOWING ARE THE HEX CODES FOR THE ASCII BIT PATTERNS  ..
01E0 ;               ..      THEY ARE ORG'D TO BE AT MEMORY 0200                         ..
01E0 ;               ..****************************************************************..
01E0 ;
01E0 ;               PAGE                    ..START THE CHARACTER PATTERN AT 0200
0200 ;
0200 00000000000000; ,#0000000000000000C8C8C8C8C808C800D4D4000000000000D4FED4D4FED4D400;
0207 00C8C8C8C8C808;
020E C800D4D4000000;
0215 000000D4FED4D4;


021C FED4D400;
0220 ;
0220 ;
0220 ;
0220 ;
0220 DCEAE8DCCAEADC; ,#DCEAE8DCCAEADC00F2F2C4C8D0E6E600DCE2E0DCE0E2DC00C8C8C0C0C0C0C000;
0227 00F2F2C4C8D0E6;
022E E600DCE2E0DCE0;
0235 E2DC00C8C8C0C0;
023C C0C0C000;
```

```
0240 ;              0443
0240 ;              0444
0240 ;              0445
0240 ;              0446
0240 C2C4C8C8C8C4C2;0447 ,#C2C4C8C8C8C4C200E0D0C8C8C8D0E000E2D4C8FEC8D4E20000C8C8FEC8C80000;
0247 00E0D0C8C8C8D0;0447
024E E000E2D4C8FEC8;0447
0255 D4E20000C8C8FE;0447
025C C8C80000;      0447
0260 ;              0448
0260 ;              0449
0260 ;              0450
0260 ;              0451
0260 0000000000C8C8;0452 ,#0000000000C8C8D0000000FE000000000000000000000C80000C2C4C8D0E00000;
0267 D0000000FE0000;0452
026E 00000000000000;0452
0275 00C80000C2C4C8;0452
027C D0E00000;      0452
0280 ;              0453
0280 ;              0454
0280 ;              0455
0280 ;              0456
0280 DCE2E6EAF2E2DC;0457 ,#DCE2E6EAF2E2DC00C8D8C8C8C8C8DC00DCE2C2C6D8E0FE00DCE2C2CCC2E2DC00;
0287 00C8D8C8C8C8C8;0457
028E DC00DCE2C2C6D8;0457
0295 E0FE00DCE2C2CC;0457
029C C2E2DC00;      0457
02A0 ;              0458
02A0 ;              0459
02A0 ;              0460
02A0 ;              0461
02A0 E0E8E8FEC8C8C8;0462 ,#E0E8E8FEC8C8C800FEE0E0FCC2E2DC00CCD0E0ECF2E2DC00FEE2C2C4C8D0E000;
02A7 00FEE0E0FCC2E2;0462
02AE DC00CCD0E0ECF2;0462
02B5 E2DC00FEE2C2C4;0462
02BC C8D0E000;      0462
02C0 ;              0463
02C0 ;              0464
02C0 ;              0465
02C0 ;              0466
02C0 DCE2E2DCE2E2DC;0467 ,#DCE2E2DCE2E2DC00DCE2E6DAC2C4D800C0C0C8C0C8C0C000C0C0C8C0C0C8C8D0;
02C7 00DCE2E6DAC2C4;0467
02CE D800C0C0C8C0C8;0467
02D5 C0C000C0C0C8C0;0467
02DC C0C8C8D0;      0467
02E0 ;              0468
02E0 ;              0469
02E0 ;              0470
02E0 ;              0471
02E0 C4C8D0E0D0C8C4;0472 ,#C4C8D0E0D0C8C400C0C0FEC0FEC0C000E0D0C8C4C8D0E000DCE2C4C8C8C0C8C0;
02E7 00C0C0FEC0FEC0;0472
02EE C000E0D0C8C4C8;0472
02F5 D0E000DCE2C4C8;0472
```

```
02E7 00C0C0FEC0FEC0;0472
02EE C000E0D0C8C4C8;0472
02F5 D0E000DCE2C4C8;0472
```

```
02FC C8C0C8C0;      0472
0300 ;              0473
0300 ;              0474
0300 ;              0475
0300 ;              0476
0300 DCE2E2EAECE0DE;0477 ,#DCE2E2EAECE0DE00DCE2E2FEE2E2E200FCE2E2FCE2E2FC00DEE0E0E0E0E0DE00;
0307 00DCE2E2FEE2E2;0477
030E E200FCE2E2FCE2;0477
0315 E2FC00DEE0E0E0;0477
031C E0E0DE00;      0477
0320 ;              0478
0320 ;              0479
0320 ;              0480
0320 ;              0481
0320 FCE2E2E2E2E2FC;0482 ,#FCE2E2E2E2E2FC00FEE0E0FCE0E0FE00FEE0E0FCE0E0E000DCE2E0E0EEE2DC00;
0327 00FEE0E0FCE0E0;0482
032E FE00FEE0E0FCE0;0482
0335 E0E000DCE2E0E0;0482
033C EEE2DC00;      0482
0340 ;              0483
0340 ;              0484
0340 ;              0485
0340 ;              0486
0340 E2E2E2FEE2E2E2;0487 ,#E2E2E2FEE2E2E200DCC8C8C8C8C8DC00CEC4C4C4C4E4D800E2E4E8F0E8E4E200;
0347 00DCC8C8C8C8C8;0487
034E DC00CEC4C4C4C4;0487
0355 E4D800E2E4E8F0;0487
035C E8E4E200;      0487
0360 ;              0488
0360 ;              0489
0360 ;              0490
0360 ;              0491
0360 E0E0E0E0E0E0FE;0492 ,#E0E0E0E0E0E0FE00E2F6EAEAE2E2E200E2F2EAEAE6E2E200DCE2E2E2E2E2DC00;
0367 00E2F6EAEAE2E2;0492
036E E200E2F2EAEAE6;0492
```

```
0375 E2E200DCE2E2E2;0492
037C E2E2DC00;        0492
0380 ;                0493
0380 ;                0494
0380 ;                0495
0380 ;                0496
0380 FCE2E2FCE0E0E0;0497 ,#FCE2E2FCE0E0E000DCE2E2E2EADCC200FCE2E2FCEBE4E200DCE2E0DCC2E2DC00;
0387 00DCE2E2E2EADC;0497
038E C200FCE2E2FCEB;0497
0395 E4E200DCE2E0DC;0497
039C C2E2DC00;        0497
03A0 ;                0498
03A0 ;                0499
03A0 ;                0500
03A0 ;                0501
03A0 FEC8C8C8C8C8;0502 ,#FEC8C8C8C8C800E2E2E2E2E2DC00E2E2E2E2D4C800E2E2E2EAEAF6E200;
03A7 00E2E2E2E2E2;0502
03AE DC00E2E2E2E2;0502
03B5 D4C800E2E2EA;0502
03BC EAF6E200;        0502
03C0 ;                0503
03C0 ;                0504
03C0 ;                0505
03C0 ;                0506
03C0 E2E2D4C8D4E2E2;0507 ,#E2E2D4C8D4E2E200E2E2D4C8C8C8C800FEC2C4C8D0E0FE00CEC8C8C8C8CE00;
03C7 00E2E2D4C8C8C8;0507
03CE C800FEC2C4C8D0;0507
03D5 E0FE00CEC8C8C8;0507


03DC C8C8CE00;        0507
03E0 ;                0508
03E0 ;                0509
03E0 ;                0510
03E0 ;                0511
03E0 C0E0D0C8C4C2C0;0512 ,#C0E0D0C8C4C2C000DCC4C4C4C4C4DC00C8DCFEC8C8C800FEFEFEFEFEFEFE00;
03E7 00DCC4C4C4C4C4;0512
03EE DC00C8DCFEC8C8;0512
03F5 C8C800FEFEFEFE;0512
03FC FEFEFE00;        0512
0400 ;                0513
0400 ;                0514
0400 ;                0515
0400 ;                0516
0400 ;                0517
```

When incorporating RCA Solid State Devices in equipment, it is recommended that the designer refer to "Operating Considerations for RCA Solid State Devices", Form No. 1CE-402, available on request from RCA Solid State Division, Box 3200, Somerville, N.J. 08876.

Video Information Series

I. A Review of General Video Display Techniques

Introduction

The television industry has been alive and well for over thirty years and yet the method of transmitting and receiving video picture information has remained essentially unchanged. In fact, the only real controversy in all that time occurred around 1953 when the Federal Communication Commission (FCC) was assigned the task of deciding upon an industry standard, based on the recommendations of the National Television System Committee (NTSC), for color TV operation that would be reciprocally compatible with the existing black and white standard. The NTSC standard decided upon is still intact today and used quite successfully. It is indeed unusual that, within such a well defined and stable framework, most of the design and application effort has been directed toward such a limited area of the TV industry; namely the entertainment/news media. Only within the last ten years have other areas, such as education and scientific research, been seriously explored.

One area, in particular, that has shown increased interest and activity recently is the video display terminal industry. The successful marketing of microprocessors has encouraged much of this current activity. Previously, most video terminals were used primarily in large computer systems as input/output devices or in Computer-Aided Design systems as circuit-design display devices. With the advent of low-cost, powerful microprocessors and peripheral control circuits, the use of video terminals has increased considerably. These circuits have allowed the ability to add features and change functions, transforming discrete-logic 'dumb' video terminals into 'intelligent' work stations capable of sophisticated word processing, scientific analysis, and computer graphics. The expanding home and small business computer market has also increased the popularity of video terminals. The video-games industry has created a whole new field of video design and application interest and the home TV, used as a low-cost video terminal, has replaced the surplus teletype for the computer hobbyist.

In light of this video renaissance, it seems appropriate to review the basic design techniques of video terminals and TV in general. In order to effectively use the LSI CRT controller circuits available today, some background knowledge of TV and video electronics is a necessity.

Video Basics

The primary objective in any video display system is to present information to the viewer, whether it be in a picture format as with standard broadcast TV or in a text or graphic format as with a computer interface. The device used for this purpose is the cathode-ray tube (CRT), a vacuum tube with a large, flat face coated with a special phosphor material that emits light when struck by an electron beam. A high voltage (15,000 - 25,000 volts) is used to accelerate the beam from the rear of the tube to the phosphor-coated face. The horizontal and vertical deflection circuitry determine which area of the screen is illuminated, with additional circuitry used to control the brightness, contrast, and focus of the display.

Two types of horizontal and vertical deflection methods are in wide use today. Electrostatic deflection, used in lab oscilloscopes and high speed graphic displays, provides the highest picture resolution and speed but is also expensive to implement. Electro-magnetic deflection, used in broadcast TV and low-cost video terminal applications, is much less expensive because of large volume production and varied applications.

Whichever method is used, the deflection system may be thought of as an X-Y co-ordinate plot, with the horizontal being the X direction and the vertical being the Y direction. (Fig. 1.) The two variables (X and Y) may be controlled independently by two methods (plotting and sweeping), yielding three distinctly different display systems:

<u>Plot X - Plot Y</u> - In this method, known as stroke writing, a plot of X and Y is performed to produce a vector display. The beam is moved and illuminated only for the screen positions necessary to form the desired display. The CRT used is a storage-type CRT, in which the phosphor material has a long persistance allowing the beam trace to remain visible after the deflection operation.

<u>Sweep X - Plot Y</u> - In this method, used extensively in lab oscilloscopes, a variable time-base generator is used to repetitively move the beam from left to right across the screen (Sweep X). The beam is moved and illuminated in the Y direction by the vertical amplifier circuitry, which responds to external voltages (Plot Y), to produce a time-varying voltage waveform display. Both non-storage and storage-type CRT's are utilized.

<u>Sweep X - Sweep Y</u> - This method, known as a raster scan display, is used exclusively in broadcast TV and in most video terminals. The beam is moved at a constant rate from left to right by the horizontal sweep circuitry and from top to bottom by the vertical sweep circuitry (Fig. 2). Since the raster scan rate is fixed, the display is re-written (refreshed) at a periodic rate equal to the vertical scan frequency, eliminating the need for a storage-type CRT. The horizontal and vertical scan rates are selected to provide refreshing often enough to prevent any visually annoying flicker in the display.

In the NTSC raster scan video display, the electron beam is started in the upper lefthand corner of the screen and deflected horizontally to the right at a frequency of 15,750 Hz (63.5 us). When the beam reaches the right side of the screen a horizontal sync pulse occurs, and the beam to be returned (retraced) very quickly ($\cong$5 us) to the left side of the screen. At the same time, the beam is also being deflected vertically from the top of the screen to the bottom at a frequency of 60 Hz (16.7 ms). When the beam reaches the bottom righthand corner of the screen a vertical sync pulse occurs, and the beam to be retraced very quickly (≅1 ms) to the top of the screen. Since the beam is being deflected in two directions simultaneously, each horizontal scan line is actually moving at an unnoticeable downward slope to the right (Fig. 2). The horizontal deflection occurs much more rapidly than the vertical deflection, allowing 262.5 horizontal lines to be scanned within each vertical field. As the electron beam is moved across the screen its intensity is varied in proportion to the picture information seen by the TV camera.

The scanning frequencies used in NTSC TV were selected to present a flicker-free image with the appearance of smooth, continuous motion, within the allotted transmitting bandwidth. In order to provide motion, the entire screen must be scanned within 1/20 sec (50 ms) or faster. This is the minimum time required for effective persistence of vision, which is the eye's ability to retain an image after it has been removed from direct view. The TV frame rate of 1/30 sec (33.4 ms) easily meets this requirement. However, to prevent flicker, the screen must be illuminated at least twice the 1/20 sec rate. This could be done by simply transmitting all 525 lines (the amount of lines necessary for good picture resolution) at twice the frame rate of 1/60 sec (16.7 ms), but this would require a transmitting bandwidth much higher than the allotted 6 MHz. The method used to overcome this problem is called interlaced scanning, in which two fields of video information are transmitted at 1/30 sec, with every other field started 1/2 horizontal line later than the one before it. The two fields are interlaced at a 2 to 1 rate to produce one complete picture frame. Since the picture content changes at a rate slower than the field time (16.7 ms), the two fields appear to be seen simultaneously. The net effect is a frame every 33.4 ms, which is a composite of two 262.5-line fields and which can be transmitted within the 6 MHz channel bandwidth, resulting in a total screen resolution of 525 lines. (Fig. 3)

To work effectively, the horizontal and vertical deflection system must be precisely controlled by the signal sent from the broadcast TV transmitter. This is done by adding synchronization information to the picture information. The horizontal and vertical sync pulses which are transmitted keep the TV set locked to the exact scene that the camera sees.

Fig. 4 shows the horizontal and vertical sync pulse timing relationship. The horizontal pulses occur at the end of each line and the vertical pulses occur at the end of each field. Although the vertical sync that is produced during the vertical blanking period is composed of shorter pulses at twice the horizontal rate, it is actually converted in the TV chassis to a single vertical sync pulse with a duration of 190.5 us and a repetition rate of 16.7 ms. The equalizing pulses and serrated vertical pulses are used to maintain horizontal sync during the vertical retrace time. Also, during both the horizontal and vertical blanking time, the video information is held at the black level to prevent interference with normal picture quality. Although Fig. 4 shows the two-to-one interlace relationship (Ⓐ), it is not absolutely necessary for video terminal applications and is often omitted.

Since the horizontal and vertical sync are transmitted together, some facility must be provided in the TV set to distinguish between the two. This operation is performed by the sync separator section of the TV. The two basic circuits used are the differentiator and the integrator. The differentiator responds to sync clock-edges at the horizontal timing frequency and triggers an oscillator that is free-running near the 15,750 Hz rate. The integrator responds to the sync pulse width at the vertical timing frequency and triggers an oscillator that is free-running near the 60 Hz rate. The normal horizontal sync pulses are ignored by the vertical integrator circuit because their comparatively low duty cycle cannot charge the integrator to the vertical oscillator trigger point. During vertical retrace, however, the serrated vertical pulses, which are essentially double-frequency inverted horizontal pulses, provide a much higher duty cycle that allows the vertical integrator to charge to the necessary trigger level. Because the equalizing and serrated pulses are at a multiple (2H) of the horizontal scanning frequency, the horizontal differentiator continues to respond to these pulses, maintaining horizontal sync during vertical retrace. After the horizontal and vertical sync pulses are separated they are sent to their respective drive circuits to control the CRT deflection yokes.

The video information is also separated at this time and sent to the video amplifiers and CRT drive circuits. The scene, as it appears to the TV camera, can now be displayed in a synchronized time relation to the viewer. Fig. 5A shows a simplified block diagram of a typical black and white TV chassis. The information presented thus far has followed a route from the CRT to point B in the diagram. The remaining circuitry is used to amplify the RF antenna signal, select the desired channel, and recover the video, sync, and sound information from the modulated RF carrier frequency.

It is at point B that many CRT controller circuits used in video terminal applications are connected. The composite video and sync signal produced by the CRT controller is designed to interface at a standard video input level as shown in Fig. 5C. However, some CRT controllers interface earlier or later in the block diagram signal path. In the earlier path, the composite video and sync signal from the CRT controller is used to drive an RF modulator circuit, which produces a standard output (as shown in Fig. 5B) that contains a modulated Radio Frequency signal with a carrier frequency equal to the TV channel picture carrier (i.e., Channel 2 = 55.25 MHz). This signal is essentially the same as the signal produced by the broadcast TV transmitter and may be connected directly to a standard TV at the antenna terminal inputs. This method provides a simple interface, but requires the use of the TV IF and RF sections shown in Fig. 5A. The signal bandwidth is also limited by the RF section and the 4.5 MHz sound section, preventing more than approximately 40 characters from being displayed horizontally on the screen. Since these signal limiting sections are bypassed using the composite video method, which is connected at point B, higher resolution displays are possible. This method may require minor modifications to a standard TV chassis to interface at point B, although some commercial models are supplied with an external video input for use in monitor applications. Other commercial models are available strictly for monitor use, with the RF and IF sections omitted, offering higher resolution display capabilities.

Standard TV chassis circuitry may be further reduced by interfacing still later in the signal path by using separate horizontal, vertical, and video signals connected directly to the appropriate TV drive circuits. This method is most often used for high resolution color monitors and the drive circuits required often use non-standard signals, making interfacing more difficult, thereby limiting the CRT controller to a particular TV chassis type.

Summary

Obviously, many options are available to both the CRT controller designer and user. Of the many controller circuits on the market, few are directly interchangable. Some are intended for low resolution and simple interfacing, while others are quite complex, in both circuitry and display capability. The user will find it necessary to decide on a CRT controller circuit, based on such factors as display density, ease of interfacing, additional IC circuitry required, cost, flexibility, and second-sourcing. However, since the broadcast TV standards, and subsequently the video terminal requirements, have long been established, new application design pains are somewhat eased. The low-cost, production volume and time-tested equipment produced by the broadcast TV industry is used to advantage by the relatively new video terminal industry, to the point, in fact, of using standard TV chassis as video display units.

In newsletters to follow, CRT controller circuits will be analyzed with respect to sync generation, screen formatting, character generation, microprocessor impact and interfacing, and system design considerations. A practical video terminal system will be presented using the RCA VIS chip-set (CDP1869/70). This system will be used as a basic building block, with options, such as color and sound, added as they are discussed in details.

For additional information contact Rick Vaccarella, X6542.

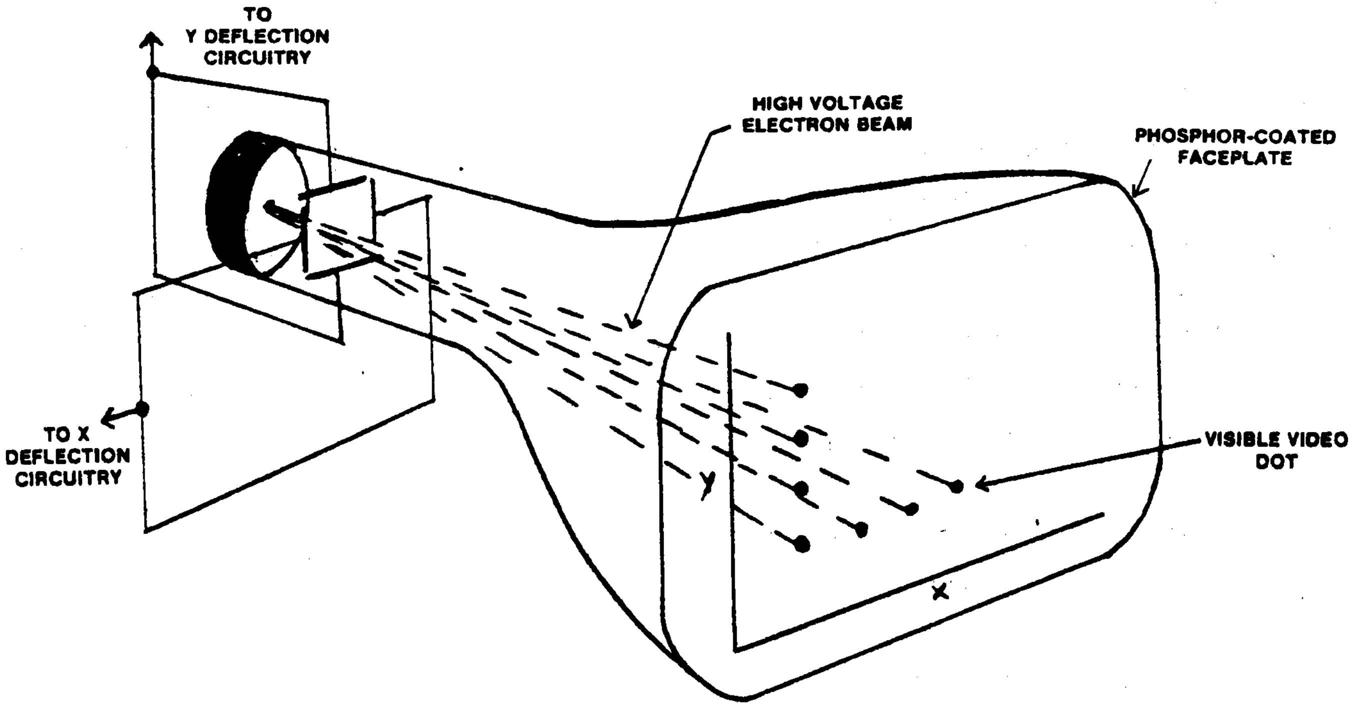


FIG. 1-CRT DEFLECTION SYSTEM

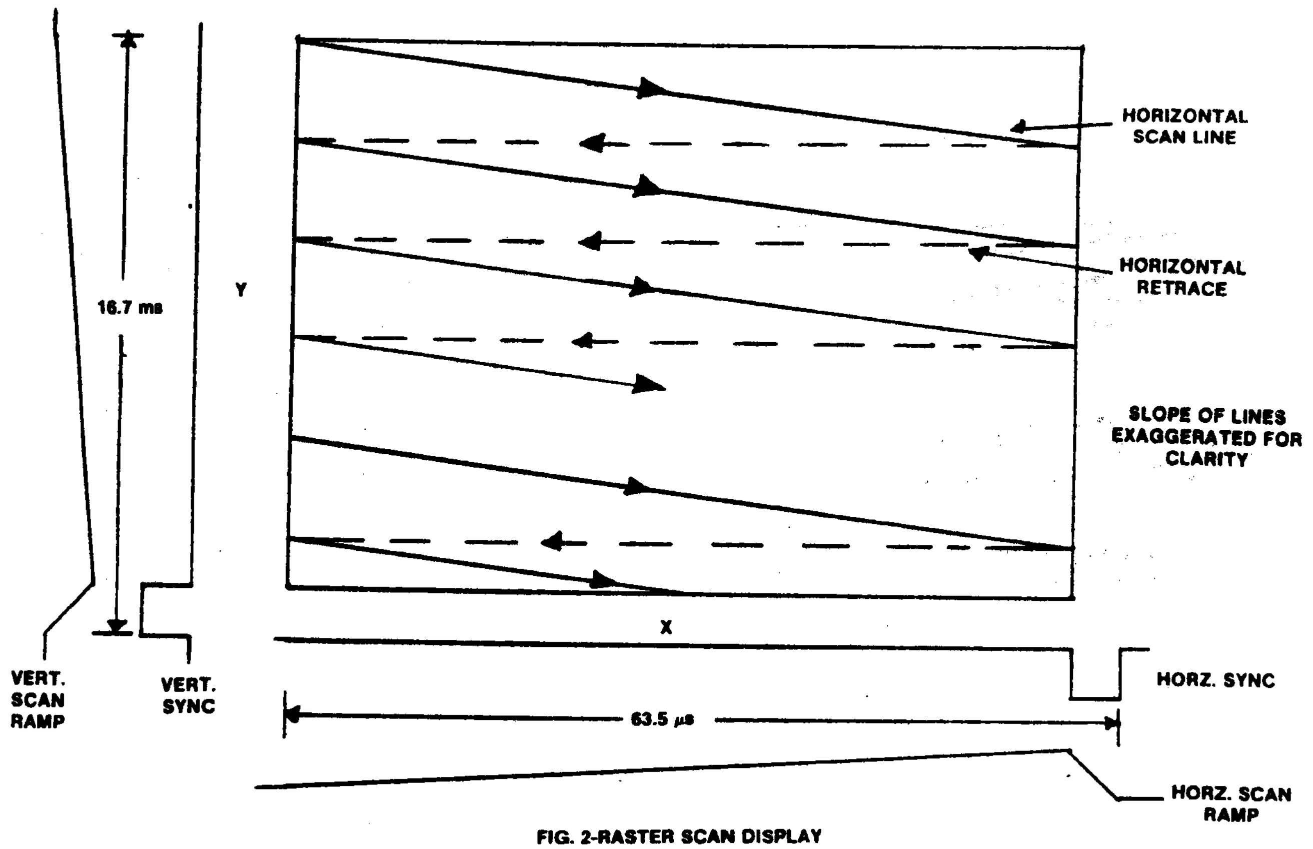


FIG. 2-RASTER SCAN DISPLAY

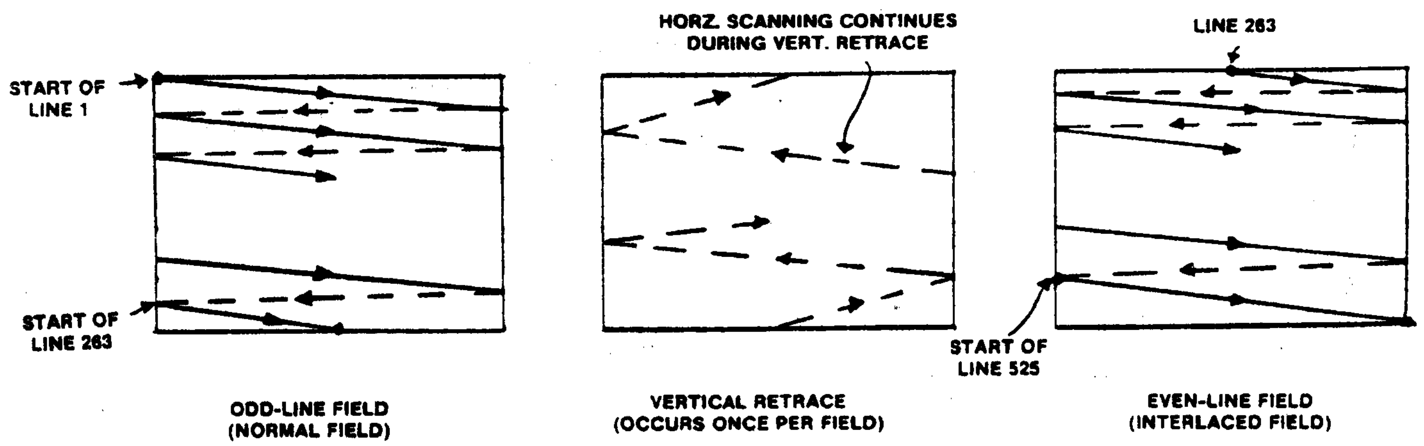
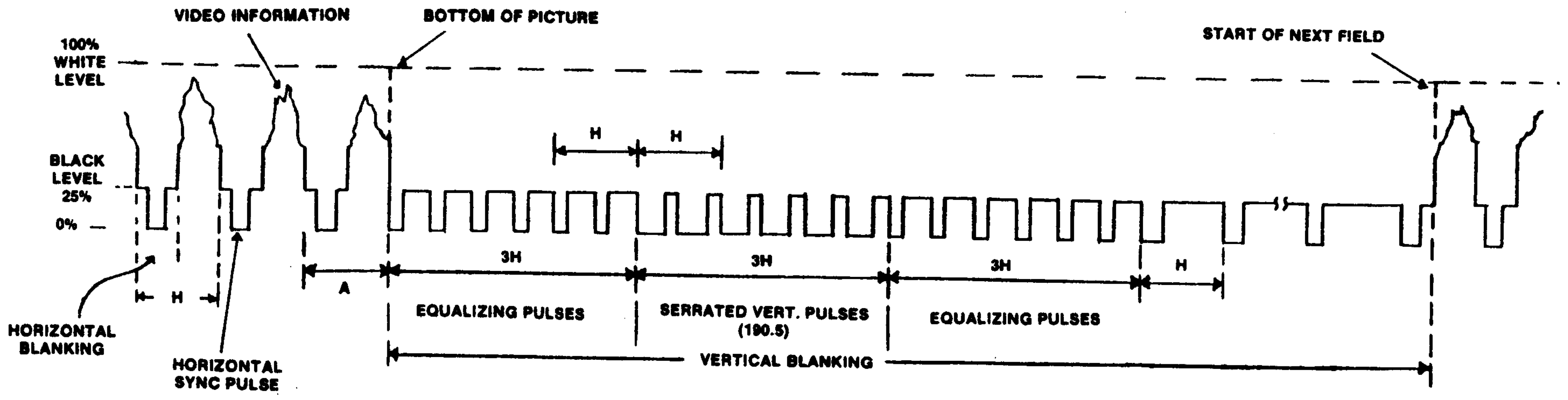

HORZ. SCANNING CONTINUES DURING VERT. RETRACE
LINE 263
START OF LINE 1
START OF LINE 263
START OF LINE 525
ODD-LINE FIELD (NORMAL FIELD)
VERTICAL RETRACE (OCCURS ONCE PER FIELD)
EVEN-LINE FIELD (INTERLACED FIELD)

FIG. 3-2 TO 1 ODD-LINE INTERLACE SCANNING



A = H FOR FULL SCAN LINE
A = 0.5 H FOR HALF SCAN LINE
H = HORIZONTAL SCAN LINE (63.5 μs)

FIG. 4-TV SYNC TIMING

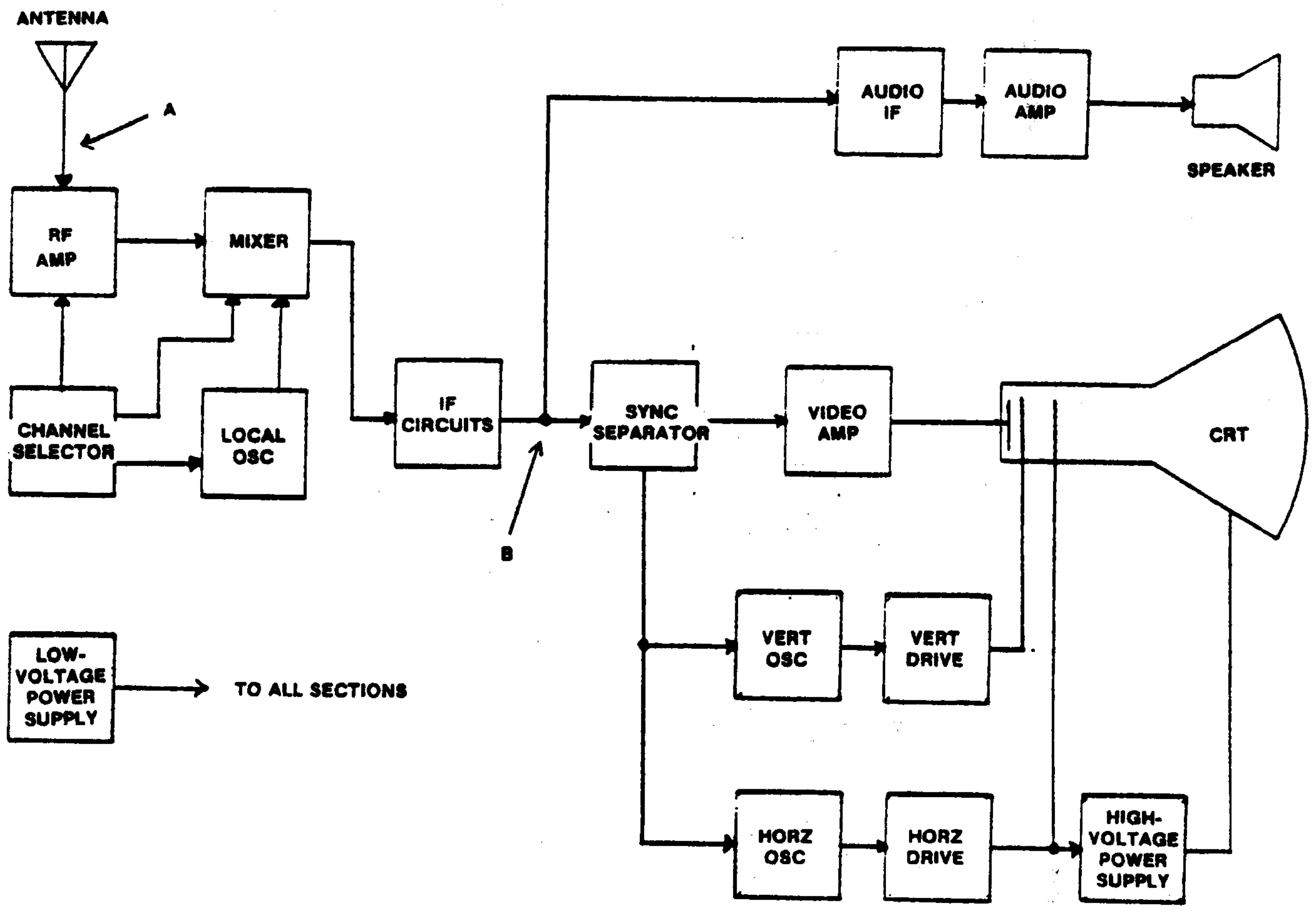


FIG. 5A - SIMPLIFIED TV BLOCK DIAGRAM

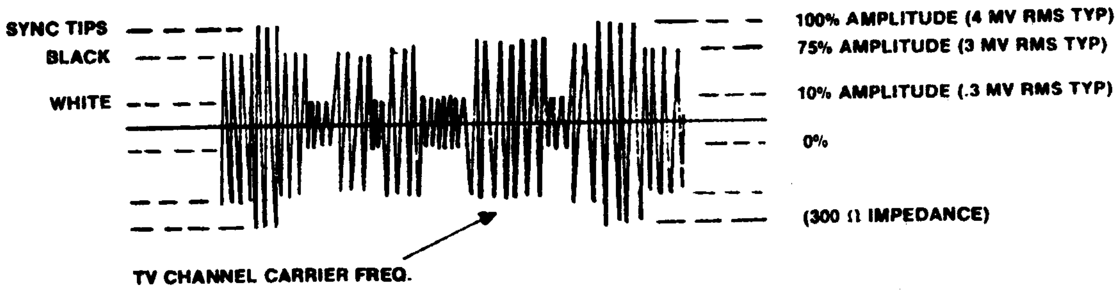

SYNC TIPS
BLACK
WHITE
100% AMPLITUDE (4 MV RMS TYP)
75% AMPLITUDE (3 MV RMS TYP)
10% AMPLITUDE (.3 MV RMS TYP)
0%
(300 Ω IMPEDANCE)
TV CHANNEL CARRIER FREQ.

FIG. 5B-STANDARD RF SIGNAL LEVELS

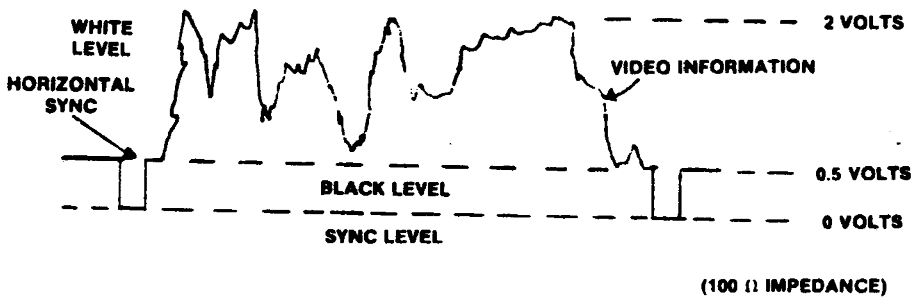


FIG. 5C-STANDARD VIDEO SIGNAL LEVELS

RCA
Solid State Division

Microprocessor Products
Application Note

ICAN-6889

USING SLOWER MEMORIES WITH THE VIS DISPLAY SYSTEM

G. T. Fogarty

The VIS (Video Interface System) Display System (CDP1869 and CDP1870)[1], a minimal-device-count approach to color-character generation, is essentially a CRT controller designed to interface to the CDP1800 series of microprocessors (CDP1802, CDP1804). The system relieves the CPU of the chore of generating screen refresh timing or data. Other capabilities include programmable background and character colors, white-noise and tone generator, and hardware scrolling. The scheme described in this Note, while requiring a few more parts, very nearly doubles the memory access-time requirement of the system, and permits the use of memories approximately half as fast as those normally required with the VIS System.

VIS System Operation

The VIS System was designed with minimal chip count as a goal. A minimum I/O system requires only the CDP1869, CDP1870, page memory, character memory, and two bus separator chips, Fig. 1. The bus separators are required to allow the CPU to access the page memory. The character memory bus multiplexing is internal to the CDP1870.

The character generating scheme is as follows:

The page memory is a sequential list of character positions on the CRT screen. Its data is a pointer to the character to be displayed at that screen position. The character memory contains the actual dot pat-

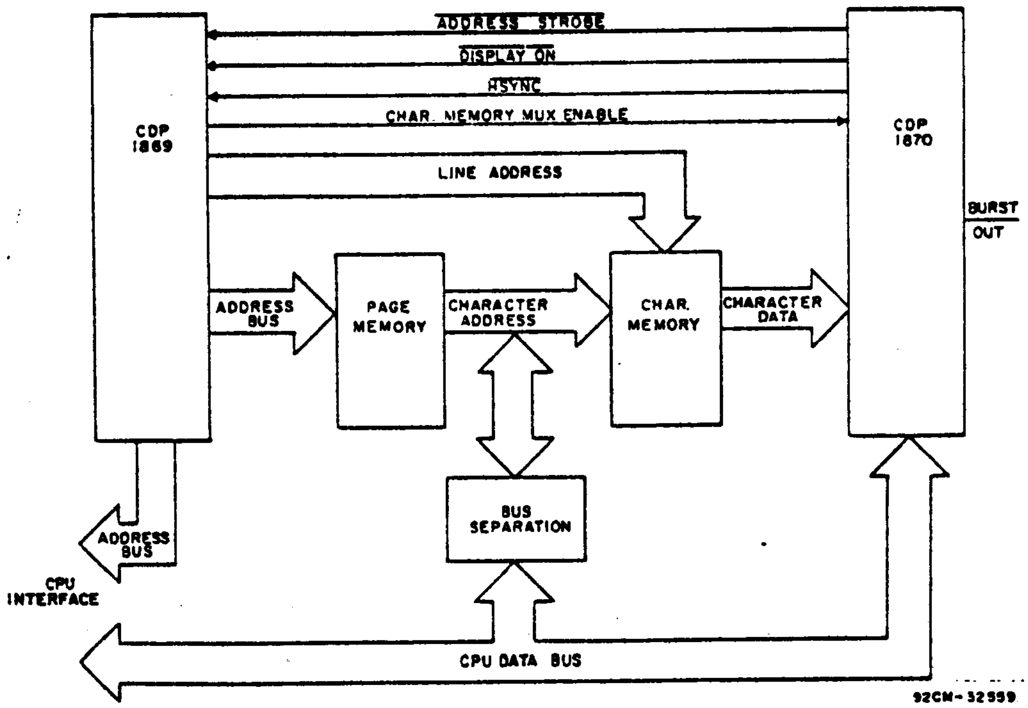


Fig. 1—A minimum I/O system.

tern of all the possible characters that can be displayed. During screen refresh, the CDP1869 generates addresses to the page memory. The page memory output data, in turn, addresses the character memory, whose data is then latched into the CDP1870 for serial output to the CRT screen.

Note that one character cycle involves two memory access times: page address to page memory, and page data (character address) to character memory. In a 40-character-per-line system, the total of these two times is about one microsecond (six dot clocks) minus the page memory address delay from the CDP1869 and the character data setup time into the CDP1870.

The timing diagram in Fig. 2 shows that the signal, ADDRESS STROBE, from the CDP1870 advances the page-memory address in the CDP1869 at the trailing edge, signal. The multivibrator signal is used to advance the address counter in the CDP1869 and to latch page-memory data. The burst signal, occurring once per horizontal sync., is used to generate the "extra" address count. During nondisplay time, the one-shot is disabled to prevent the address counter from advancing, and the latch control is held true. The latter makes the latch feed through, enabling normal character memory access. Gates A and B inhibit the last strobe in half and full resolution, which allows for proper hardware rolling or scrolling. PMA4 through PMA9 are addresses 4 through 9 of the CDP1869 and are used for the page memory.

With this "staggered access" circuit, page memory access starts at the trailing edge of the new strobe pulse (pulse address out delay) and terminates at the trailing edge of the next strobe; the total

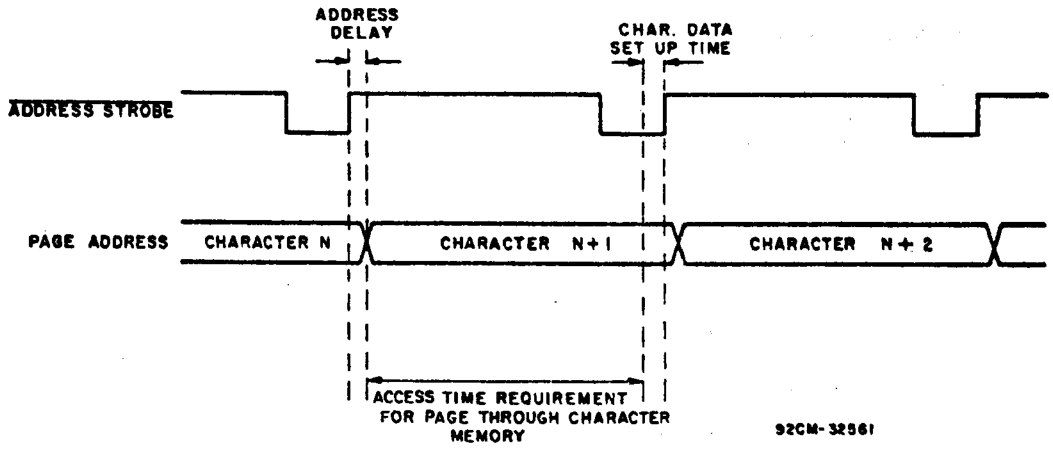


Fig. 2—Timing diagram for Fig. 1.

initiating an address cycle. This signal is analogous in time to the shift-register load signal in the CDP1870. Therefore, the next ADDRESS STROBE terminates the current access, and initiates the next character access.

Adaptation for Slower Memories

If page and character memory were accessed in parallel, the memory speed requirement would be eased substantially. This parallel accessing can be accomplished by latching the page-memory data and keeping the page memory one character ahead of the character memory; the circuit shown in Fig. 3 accomplishes this task. A TTL one-shot multivibrator was chosen for minimal delay. The multivibrator generates a pulse of approximately 200 nanoseconds starting at the trailing edge of the ADDRESS STROBE time required is approximately that to access one full character. Character memory access starts at the **leading** edge of the strobe signal (plus latch delay) and terminates at the data setup time requirement of the CDP1870, also approximately the time to access one full character. Since the delay through the latch is similar to the address out delay of the CDP1870, this scheme, while it requires a few more parts, essentially doubles the memory access-time requirement of the VIS System, and permits the use with it of memories approximately half as fast as those normally required.

References

1. "COS/MOS Video Interface System," types CDP1869 and CDP1870, RCA Solid State Data Sheet No. 1197.

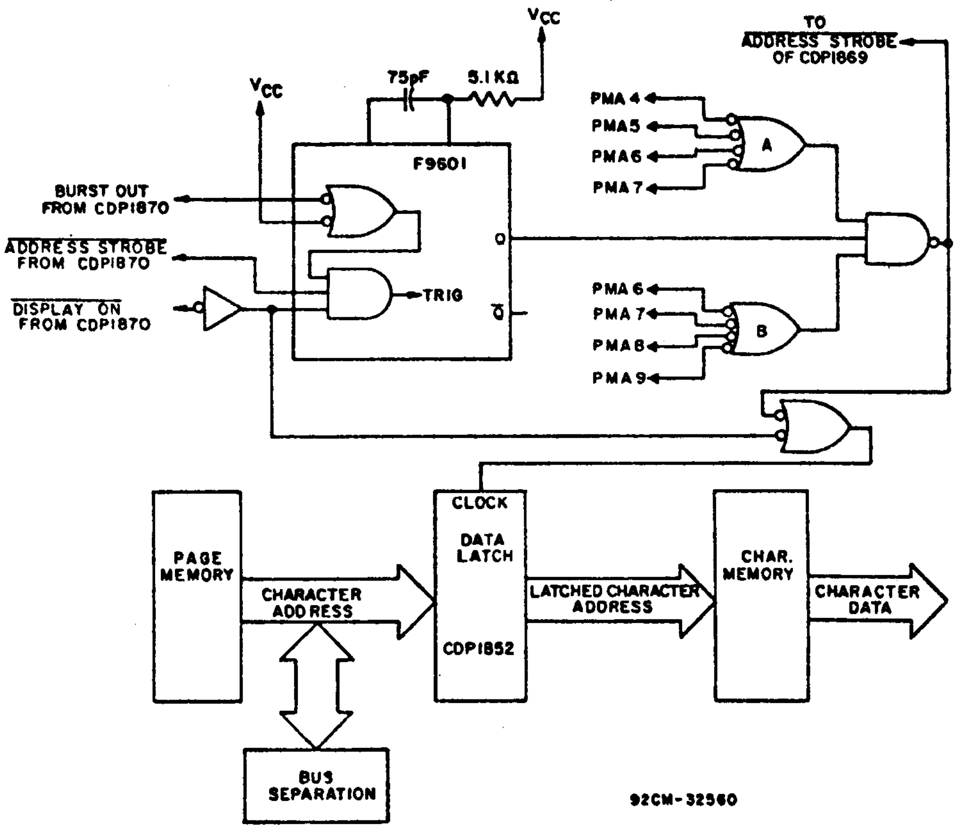


Fig. 3—Circuit used to adapt VIS System to slower memories.

RCA Solid State February 1981

The Technical and Marketing Position of VIS

The CRT display has become very popular in the past few years for use in data terminals, personal computers, video games, graphics displays, and industrial instrumentation and display. In response to the increased demand for CRT controller electronics, much of the discrete control logic has become integrated into single-chip devices. These new devices interface easily to the popular microprocessors and provide increased 'intelligence' and simple operation. Since the majority of the CRT controller available today are complex LSI circuits, the data sheets which describe their use are often difficult to interpret. Selecting the proper controller for a particular application may involve considerable time and expense for evaluation and prototyping.

The attached industry CRT controller comparison chart (Table 1) points out the variety of features offered. As a starting point in choosing a CRT controller, it may be helpful to relate the desired application to the available devices by using the CRT Controller Selection Guide (Table 2) which shows the relative merit of each device for some common applications, based on cost, capability, and overall system chip-count. After the initial selections are made the comparison chart and individual data sheets could be used to further narrow the choices.

Using the above method, it is evident that the RCA CDP1869, CDP1890/96 two-chip Video Interface System provides an economical solution to a variety of CRT display applications. The flexibility and features offered by VIS are especially competitive since most are on-chip functions, contributing to a simplified application system with a very low chip-count.

TABLE 1 — INDUSTRY CRT CONTROLLER COMPARISON

MANUFACTURER	TYPE NO.	TECHNOLOGY/ PACKAGE	μP/μC BUS COMPATIBILITY	PRICE (QTY.)	COLOR CAPABILITY	LIGHT-PEN	SYNC OUTPUTS	VIDEO OUTPUTS	ASYNCHRONOUS SYSTEM OPERATION	ON-CHIP OSC	EXTERNAL PAGE MEMORY	EXTERNAL CHAR. MEMORY
RCA	CDP1869 CDP1870/ CDP1876	CMOS 40-PIN DIP +4-10.5v	CDP1800- SERIES	CDP1869CE $18 (100) CDP1870CE $11 (100)	YES	NO	HORZ AND COMPOSITE	LUMINANCE CHROMINANCE	YES	YES SYNC DOT AND COLOR	2 K X 8 RAM	256 CHARACTERS RAM ROM
INTEL	I8275	NMOS 40-PIN DIP +5v	MCS 80 85	$3[illegible] (100)	NO	YES	NO	NO	NO DMA FROM PROCESSOR	NO	μP LIMITED USES SYSTEM RAM	128 CHARACTERS ROM
INTEL	I8276	NMOS 40-PIN DIP +5v	MCS 80 85	$15 (100)	NO	NO	NO	NO	YES DUAL ROW- BUFFER OPERATION	NO	μP LIMITED USES SYSTEM RAM	128 CHARACTERS ROM
NATIONAL	DP8350	TTL 40-PIN DIP +5v	MCS 80 85	$45 (25-99)	NO	NO	HORZ AND VERT	NO	YES	YES	4 K X 8 RAM	256 CHARACTERS ROM
MOTOROLA SYNERTEK HITACHI AMI COMMODORE SEMICONDUCTOR	MC 6845 SYP6545 HD46505 S68045 MPS6545	NMOS 40-PIN DIP +5v	MC 6800 6500	$26 (100) $21 (100) N/A $16 (100) $11 (1000)	NO	YES	HORZ AND VERT	NO	YES	NO	16 K X 8 RAM	256 CHARACTERS ROM
MOTOROLA AMI FAIRCHILD	MC6847 S68047 F6847	NMOS 40-PIN DIP +5v	MC6800 6500	$20 (100) $15 (100) N/A	YES	NO	HORZ VERT COMPOSITE SYNC LUM	LUMINANCE B-Y R-Y	YES	YES DOT (R-C OSC)	6 K X 8 RAM	512 CHARACTERS RAM ROM INTERNAL MASK ROM (64 CHARS)
SMC TI	CRT5027 TMS9927	NMOS 40-PIN DIP +5v +12v	GENERAL PURPOSE TMS9900	$20 (100) $23 (100)	NO	NO	HORZ VERT COMPOSITE	NO	YES	NO	8 K X 8 RAM	256 CHARACTERS ROM
SMC	CRT9007	NMOS 40-PIN DIP +5v	GENERAL PURPOSE	N/A	NO	YES	HORZ VERT COMPOSITE	NO	YES	NO	16 K X 8 RAM	256 CHARACTERS ROM
SMC THOMSON CSF	CRT96364 96364	NMOS 40-PIN DIP +5v	GENERAL PURPOSE (STAND ALONE CONTROLLER)	$16 (100)	NO	NO	COMPOSITE	NO	YES	YES SYNC GEN	2 K X 6 RAM (EXPANDABLE WITH EXT. HARDWARE)	64 CHARACTERS ROM
TI	TMS 9918	NMOS 40-PIN DIP +5v	GENERAL PURPOSE TMS 9900	$45 (100)	YES (2 TEXT 15 GRAPHIC 4 GRAY SCALE)	NO	COMPOSITE COLOR VIDEO, SYNC, BURST AND BLANKING		YES	YES SYNC DOT COLOR	16 K X 8 DYNAMIC RAM	256 CHARACTERS DYNAMIC RAM

PROGRAMMABLE FEATURES						
LINES CHAR	CHARS/ROWS	ROWS/FRAME	CURSOR	OTHER	EXTERNAL PARTS REQUIRED	COMMENTS
8, 9, 16	20, 40	12, 24	N/A	HOME ADDRESS (SCROLL) 8 BKG COLORS GRAY SCALE 8 DOT COLORS GRAY SCALE TONE/WHITE NOISE GEN	DATA BUS BUFFER SEPARATOR (8 BIT) VIDEO SYNC MIXER CIRCUIT μP/μC	TWO CHIP SET NTSC PAL COMPATIBLE NON INTERLACED SYNC EXTERNAL V AND H INPUTS (TELETEXT OVERLAYS) RGB BOND OUT OPTION (CDP1862)
UP TO 16	UP TO 80	UP TO 64	YES LOCATION BLINK RATE	HIGHLIGHT VIDEO REVERSE VIDEO UNDERLINE 11 NON-MEMORY GRAPHICS SPECIAL SCREEN CONTROL CODES (END-OF-LINE BLANK ETC.)	DMA CONTROLLER CHIP (8257) (40 PIN) DOT CHAR CLOCK GEN CIRCUIT SYNC TIMING GEN CIRCUIT P/S SHIFT REG (VIDEO OUTPUT) VIDEO-SYNC MIXER CIRCUIT μP/μC	OBTAINS DISPLAY DATA FROM μC SYSTEM RAM VIA DMA TECHNIQUE NON INTERLACED SYNC
UP TO 16	UP TO 80	UP TO 64	YES LOCATION BLINK RATE	HIGHLIGHT VIDEO REVERSE VIDEO UNDERLINE 11 NON-MEMORY GRAPHICS SPECIAL SCREEN CONTROL CODES (END-OF-LINE BLANK ETC.)	DOT CHAR CLOCK GEN CIRCUIT SYNC TIMING GEN CIRCUIT P/S SHIFT REG (VIDEO OUTPUT) VIDEO-SYNC MIXER CIRCUIT μP/μC	SIMILAR TO [illegible] BUT DOES NOT USE DMA TECHNIQUE LOWER COST OBTAINS DISPLAY DATA FROM μP/μC SYSTEM RAM VIA ROW BUFFER TECHNIQUE NON INTERLACED SYNC
UP TO 16 ALSO UP TO 16 HORZ DOTS/CHAR	5 TO 110	UP TO 64	YES LOCATION SIZE	HOME ADDRESS (SCROLL) HORZ AND VERT SYNC	ADDRESS MUX (12-BIT) DATA BUS BUFFER SEPARATOR (8-BIT) BUS LATCH (8 BIT) P/S SHIFT REG (VIDEO OUTPUT) VIDEO-SYNC MIXER CIRCUIT μP/μC	ALL PROGRAMMABLE FEATURES (EXCEPT CURSOR LOCATION AND HOME ADDRESS) ARE MASK-PROGRAMMABLE ONLY NON INTERLACED SYNC
UP TO 32	UP TO 256 DOT RATE LIMITED	UP TO 128	YES LOCATION SIZE BLINK RATE	HORZ AND VERT SYNC H AND V DISPLAY WINDOW HOME ADDRESS (SCROLL) FIELD SYNC (NON INTERLACED INTERLACED-SYNC INTER- LACED—SYNC AND VIDEO)	ADDRESS MUX (14-BIT) DATA BUS BUFFER SEPARATOR (8-BIT) BUS LATCH (8 BIT) DOT/CHAR CLOCK GEN CIRCUIT P/S SHIFT REG VIDEO-SYNC MIXER CIRCUIT μP/μC	NON INTERLACED AND INTERLACED SYNC AND VIDEO SY6545 TRANSPARENT ADDRESSING ROW COL ADDRESSING
12 NOT PROGRAMMABLE	32 NOT PROGRAMMABLE	16 NOT PROGRAMMABLE	N/A	ALPHA MODE 4 DOT COLORS 4 BKG COLORS SEMIGRAPHICS MODE UP TO 8 COLORS GRAPHICS MODE UP TO 4 COLORS	ADDRESS MUX (12 BIT) DATA BUS BUFFER SEPARATOR (8 BIT) PIA (6820) 4-BIT COUNTER (FOR EXT. CHAR. ROM) VIDEO-SYNC MIXER CIRCUIT μP/μC	MC6847 AND S68047 ARE FUNCTIONALLY THE SAME BUT HAVE DIFFERENT PIN OUTS INTERLACE OPTION AVAILABLE (S68047Y) CERTAIN MODES MAY BE CHANGED ON A CHARACTER BY CHARACTER BASIS (MINOR-MODE SWITCHING) HIGH DENSITY COLOR GRAPHICS CAPABILITY (256 X 192 ELEMENTS)
UP TO 16	20 to 132 (16 PRE- DEFINED COMBINATIONS)	UP TO 64	YES LOCATION	HORZ AND VERT SYNC HORZ AND VERT DISPLAY WINDOW LAST DISPLAYED DATA— ROW (SCROLL) INTERLACE NON-INTERLACED	ADDRESS MUX (13 BIT) DATA BUS BUFFER SEPARATOR (8-BIT) DOT/CHAR CLOCK GEN CIRCUIT P/S SHIFT REG (VIDEO OUTPUT) VIDEO SYNC MIXER CIRCUIT μP/μC	PROGRAMMABLE FEATURES ARE PROCESSOR, MASK OR PROM PROGRAMMABLE SELF LOAD MODE (USES PROM TO LOAD REGISTERS) BALANCED BEAM CURRENT OPTION (5037) LINE LOCK OPTION ([illegible]) PRE-PROGRAMMED VERSION OF THE 5037 (5047) COMPATIBLE CHARACTER GEN. AVAILABLE (7004) COMPATIBLE CHAR GEN. WITH GRAPHICS AVAILABLE (8002)
UP TO 16	8 - 240	2 - 256	YES LOCATION	HORZ AND VERT SYNC DMA BURST MODE INTERLACE MODES (2) HORZ SPLIT SCREEN HOME REGISTER (SCROLL) ROW-TABLE MODE INTERRUPT ENABLE	ADDRESS MUX (14 BIT) DATA BUS BUFFER SEPARATOR (8 BIT) DOT CHAR CLOCK GEN CIRCUIT P/S SHIFT REG (VIDEO OUTPUT) VIDEO-SYNC MIXER CIRCUIT μP/μC	NON INTERLACED AND INTERLACED SYNC AND VIDEO [illegible] SCREEN FORMAT [illegible] MASK ROM VERSION AVAILABLE DMA OR ROW-BUFFER OPERATION COMPATIBLE ROW [illegible] AVAILABLE [illegible] COMPATIBLE CHAR GEN WITH GRAPHICS AVAILABLE (8002)
8 NOT PROGRAMMABLE	16 NOT PROGRAMMABLE	64 NOT PROGRAMMABLE	YES LOCATION	PAGE ERASE AND HOME END-OF-LINE ERASE/RETURN LINE FEED LINE ERASE CURSOR LEFT RIGHT NORMAL CHAR ADVANCE CURSOR	DATA GATE BUFFER (8-BIT) CHARACTER DATA LATCH (8 BIT) DOT CHAR CLOCK GEN CIRCUIT P/S SHIFT REG (VIDEO OUTPUT) VIDEO SYNC MIXER CIRCUIT COMMAND DECODE ROM/PROM μP/μC (OPTIONAL)	SCREEN FORMAT NOT PROGRAMMABLE AUTOMATIC SCROLL CAPABILITY STAND ALONE SYSTEM CAPABILITY COMPATIBLE CHARACTER GEN AVAILABLE (7004) COMPATIBLE CHAR GEN WITH GRAPHICS AVAILABLE (8002) [illegible] VERSION AVAILABLE [illegible]
8 NOT PROGRAMMABLE	40 NOT PROGRAMMABLE	24 NOT PROGRAMMABLE	N/A	MODES PATTERN GRAPHICS 32 BLOCKS X 24 LINES MULTICOLOR GRAPHICS 64 BLOCKS X 48 LINES TEXT GRAHPICS 40 CHARS X 24 LINES 36 VIDEO PLANES (SPRITES) EXT. VIDEO MIXING COLOR, PATTERN, BASE— ADDRESS REGISTERS	CONTROLLER SELECT LOGIC μP/μC	AUTO DYNAMIC RAM REFRESH [illegible] VIDEO PLANES (3-D EFFECT) BIT MAPPED GRAPHICS EXTERNAL VIDEO MIXING INPUT EXTERNAL SYNC INPUT EXTERNAL COLOR PHASE LOCKING (INTERLACED AND NON INTERLACED EXT. SOURCES) SEPARATE PATTERN AND COLOR TABLES HIGH RESOLUTION COLOR GRAPHICS CAPABILITY

-139-

TABLE 2 - CRT CONTROLLER SELECTION GUIDE

	RCA	Intel	Intel	National	Mot'la, AMI	Mot'la,AMI	TI, SMC	SMC	SMC	TI
Display Application	CDP1869, 70, 76	I8275	I8276	DP8350	MC6845	MC6847	TMS9927, CRT5027	CRT9007	CRT 96364	TMS9918
Low-Resolution Dumb Terminal	F	F	F	P	P	P	F	P	E	P
Low-Resolution Smart Terminal	G	G	G	F	G	G	E	G	P	G
High-Resolution Smart Terminal	P	G	E	G	G	P	G	G	P	P
High-Resolution Video Games	F	P	P	P	P	G	P	F	P	E
Broadcast TV Text Overlays	E	P	P	P	P	F	F	P	P	G
Low-Cost Industrial Displays	E	F	F	P	F	F	G	P	G	P
Low-Power Portable Displays (External CRT)	G	P	P	P	P	P	P	P	P	P
Automotive Test Equipment	G	F	F	F	F	F	F	F	P	F

E = Excellent G = Good F = Fair P = Poor

SUMMARY OF VIS COMPETITIVE FEATURES

- STANDARD CMOS ADVANTAGES - Low power, high noise immunity, etc.
- ASYNCHRONOUS OPERATION - Independent screen-refresh operation
- PROGRAMMABLE DISPLAY FORMATS - More than adequate for industrial/commercial display applications and low-end video terminals
- LOW SYSTEM CHIP COUNT - On-board osc., sync outputs, etc.
- COLOR AND SOUND CAPABILITY - 8 dot/8 bkg colors, RGB-color option, tone and white-noise generators
- NTSC AND PAL COMPATIBLE - US/European market
- ROM OR RAM CHARACTER MEMORY - For non-standard character sets
- TELETEXT COMPATIBLE - V and H inputs available for picture overlays
- SIMPLE CDP1800-SERIES INTERFACING - CDP1802, CDP1804, CDP1805
- LOW SYSTEM COST - $30 chip set - competitive system may require significant amount of support devices

CURRENT VIS HARDWARE/SOFTWARE

Parts

CDP1869	(TA10684)	- Address/Sound
CDP1870	(TA10685)	- Color Video - composite
CDP1876	(TA10891)	- Color Video - RGB
TA10890		- Color Video - RGB - 80 char. - 10V (89943 - for CDP18S008, CDP18S040)

Microboards

CDP18S661 V1	Microboard Video-Audio-Keyboard Interface (NTSC)
CDP18S661 V3	" " " " " (PAL)
CDP18S661 V1	" " " " " (NTSC) to replace existing board. Improved color circuitry, 2K page memory (MWS5114), ROM/EPROM (linked for various +5 ROM/PROM types)

Systems

CDP18S008	CRT-Based Development System
CDP18S040	Video Terminal (for CDP18S007 Development System up-grade to CDP18S008)

Demo Products

Microboard-based, simple

Microboard-based, improved - B&W Monitor, VP-3301 keyboard (not complete - R. Rhodes)

Lancaster Products

VP-601/611	Discrete Logic Keyboard (parallel output - 611 = hex key pad)
VP-606/616	" " " (serial output - 616 = hex key pad)
VP-3301	Interactive Data Terminal (VIS)

Software

VIS Interpreter (3K, disk-based) CDP18S835

ROM (2Kx8) - ASCll character set - ?

CDP1871 Keyboard Encoder Enhancements

Design changes are presently under way to extend the performance and improve the operation of the CDP1871 keyboard encoder chip. These changes will be reflected in the preliminary data sheet, which is now in the Commercial Publications Department. The currently available Objective Data Sheet (File No. 1232) describes the operation of the older CDP1871 design.

The new design improvements include:

Case A	Improved data-out access time from Chip Select.
Case B	Reduced Drive Line noise sensitivity via the use of active pull-down devices on the D1 - D11 outputs.
Case C	Elimination of the glitch pulse present on the $\overline{\text{RPT}}$ output ($\overline{\text{EFXB}}$) during the occurence of the $\overline{\text{DA}}$ output ($\overline{\text{EFXA}}$).
Case D	Internal latch circuits on the Shift and Control inputs to eliminate erroneous key code outputs.

For customers who have been sampled with the older parts or who are designing with the Objective Data Sheet, external circuits may be used to upgrade existing application designs. These circuits can subsequently be eliminated or disabled as the newer production parts become available.

Case B - Drive Line noise sensitivity may cause erroneous key codes because of stray capacitance on the drive line outputs. For example, when the key switch connecting D1 and S1 is closed, that key is detected when the scan counter output enables D1, connecting it to V_{DD}. When the key switch is released, the scan count resumes and D1 is disconnected from V_{DD}. Since the Drive Lines do not contain any active or passive devices to V_{SS}, D1 is now left unterminated and any stray capacitance is discharged only via leakage paths. If sufficient charge has accumulated on D1 and another key switch that is tied to D1 is depressed, a valid key detect results and the scan counter is stopped. However, the scan count will be different from the depressed key switch that stopped the scan and an erroneous key code output is read.

The effect of the added active pull-down devices on the new design is to discharge any capacitance on the Drive Lines when a depressed character key is released and the scan count advances.

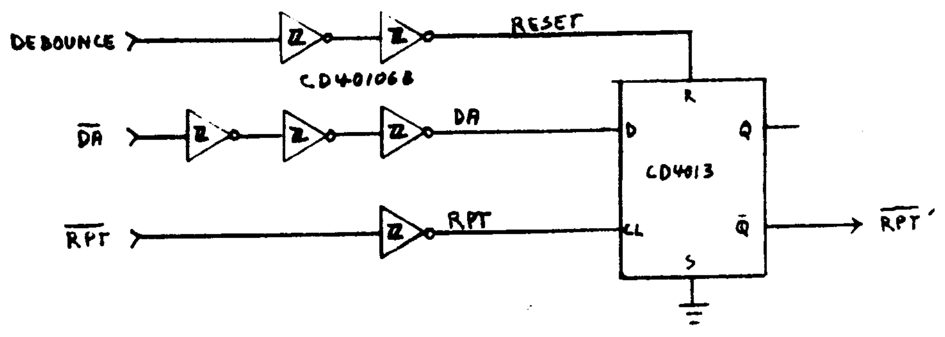
On the older devices, Drive Line noise sensitivity can be reduced by minimizing any stray capacitance on the keyboard PC board and the connecting cables associated with the Drive and Sense lines.

Case C - The glitch pulse on $\overline{\text{RPT}}$ (Pin 35) is caused by the skewing of timing of an internal NOR gate. The result is a short pulse on the $\overline{\text{RPT}}$ output occurring from the time a key detect is latched until $\overline{\text{DA}}$ becomes valid. This may cause a problem in a system that uses the $\overline{\text{RPT}}$ output in an edge-triggered circuit. The glitch pulse might initiate an invalid repeat condition.

The problem is eliminated on the new design by replacing the NOR gate with a flip-flop. The same technique may be used externally with the older devices as shown in Fig. 1. A CD4013 D F/F, which is clocked by $\overline{\text{RPT}}$, produces a signal, $\overline{\text{RPT}}'$, only during a valid repeat condition. The D input is provided by a delayed $\overline{\text{DA}}$ signal, which allows the CD4013 $\overline{\text{Q}}$ output to change only after a valid $\overline{\text{DA}}$ output condition. Since the glitch pulse on $\overline{\text{RPT}}$ occurs before a valid $\overline{\text{DA}}$ output is available, the outputs of the CD4013 remain unchanged during this invalid $\overline{\text{RPT}}$ time. The delayed $\overline{\text{DA}}$ signal ensures sufficient set-up and hold-time for the CD4013 D input with respect to the Clock input during a valid $\overline{\text{RPT}}$ condition. The CD4013 is reset by a Schmitt-trigger buffer driven by the debounce input (Pin 36) of the CDP1871, so that a valid $\overline{\text{RPT}}$ condition remains active until the key is released and the debounce RC time has terminated.

Case D - Erroneous key code outputs can result during certain Shift or Control key operations. A normal sequence would be to press the Shift or Control key and the desired character key. However, if the Shift or Control key is then released the code for the unshifted or non-control function is produced at the outputs. This is because the Shift and Control inputs are connected directly to the internal decoder gates which drive the D1 - D11 outputs. If the Shift or Control input is released before the character key is released, that decoder gate is disabled and the key detect signal is removed allowing the scan counters to advance to the unshifted or non-control code for the depressed character key. A similar action results if the Shift or Control input is depressed after the character key is depressed.

The problem is eliminated on the new design by using feed-through latch circuits on the Shift and Control inputs. These inputs are latched internally by a valid key detect condition. Any activity on the Shift or Control inputs is then ignored until the character key is released and the debounce RC time has terminated. The same technique may also be used externally with the older devices as shown in Fig. 2. This circuit is identical to that used in the CDP18S008 Development System IV. A CD4042B feed-through latch is used to latch the levels on the Shift and Control switches, if either of the $\overline{\text{DA}}$ or $\overline{\text{RPT}}$ signals are valid. When $\overline{\text{DA}}$ and $\overline{\text{RPT}}$ are not valid, the CD4042B returns to a feed-through condition, in which the Q outputs follow the D inputs.



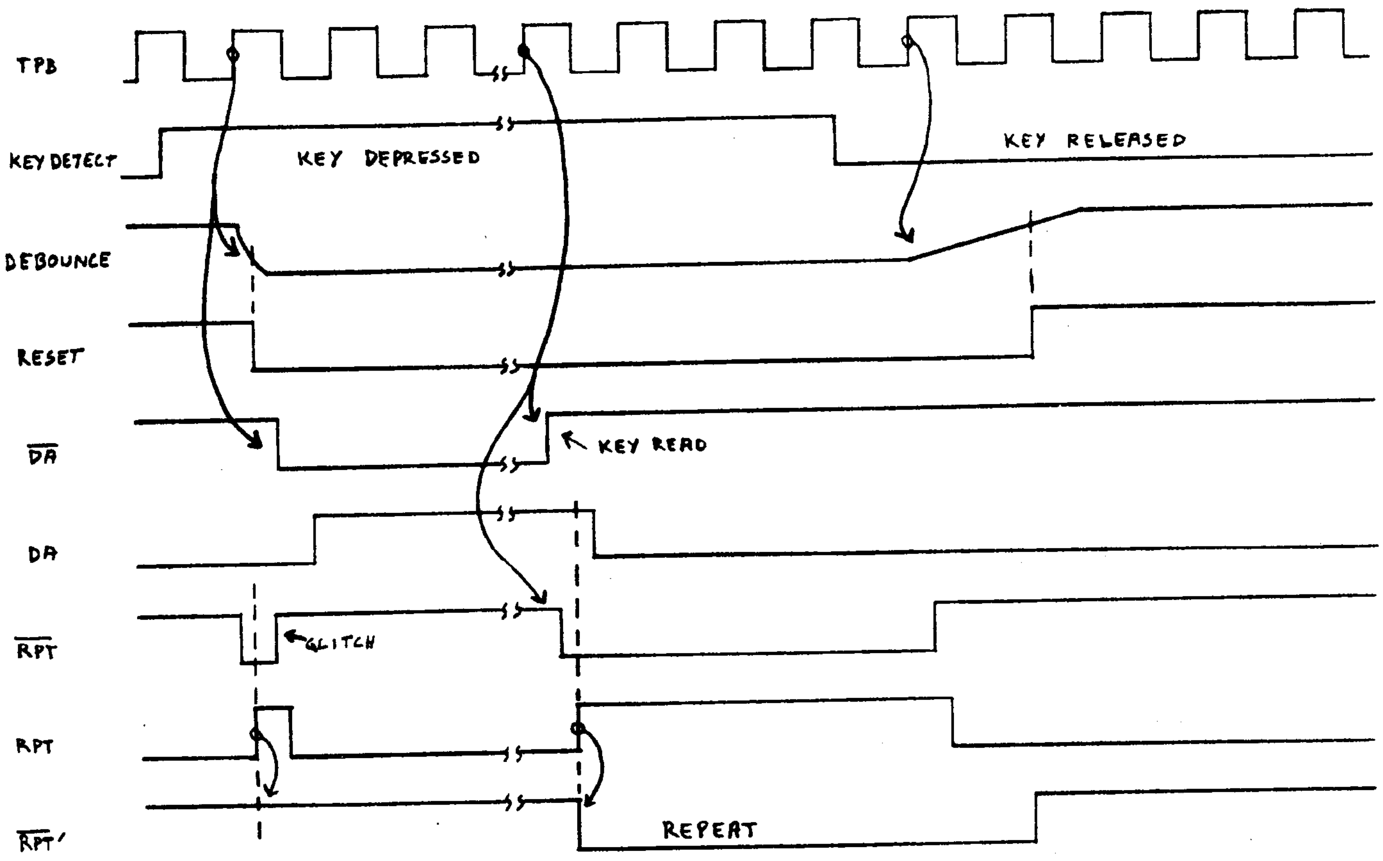
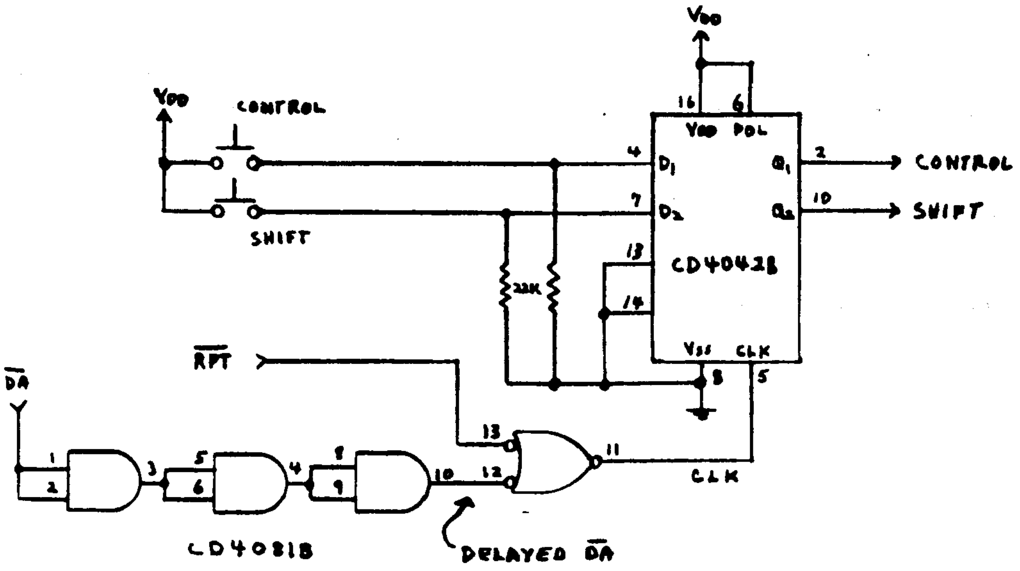
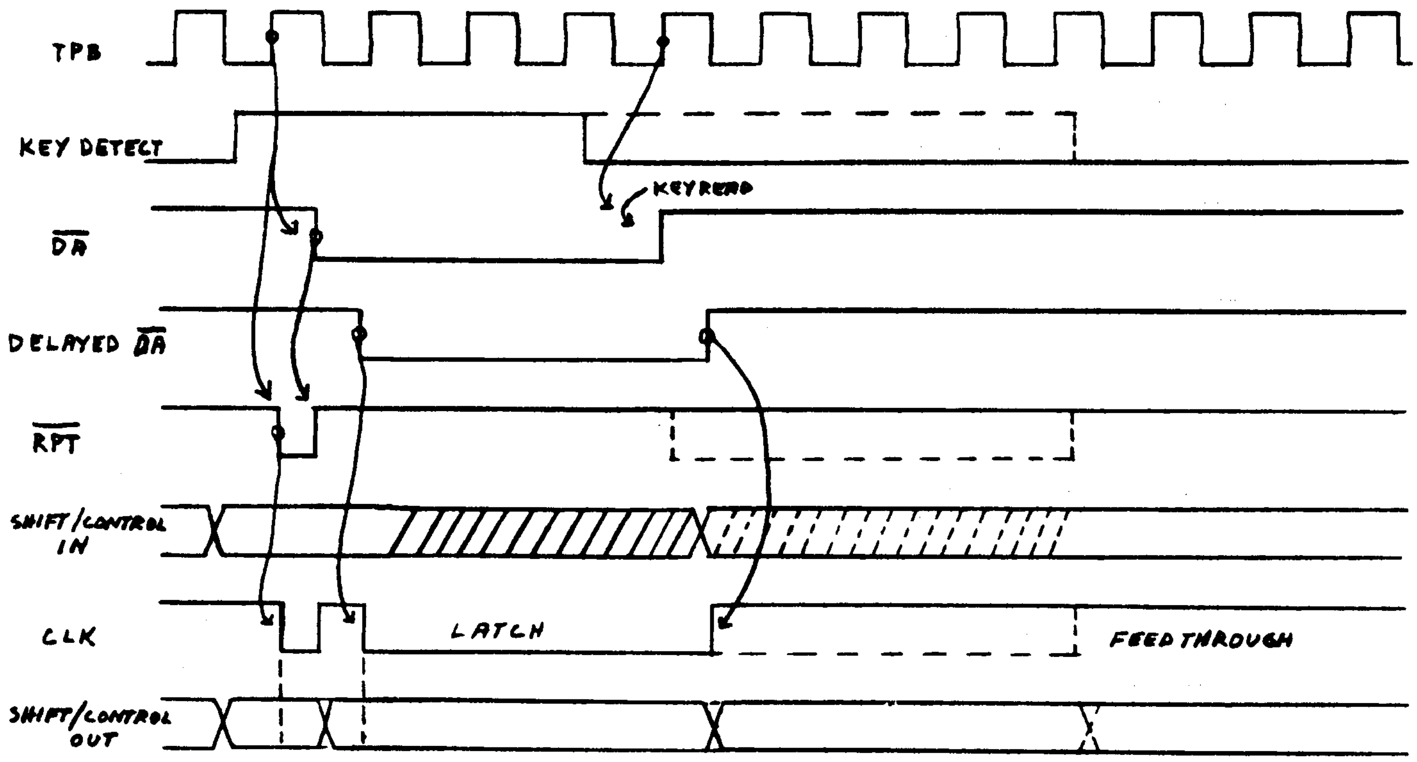


Fig. 1 - CDP1871 External $\overline{\text{RPT}}$-Glitch Eliminator Circuit





▨ = DON'T CARE

DOTTED LINES SHOW REPEAT OPERATION

(Used in CDP18S008 Development System)

Fig. 2 - CDP1871 External Shift/Control Latch Circuit

A Competitive Analysis of the 5101 CMOS Static RAM
or
How to Interpret Manufacturers' Data Sheets for Power Consumption

The RCA5101 (256x4 configuration) is a shining example of the benefits of SOS technology. In comparisons with cometitive CMOS versions of this device, the RCA part, by virtue of the high speed and packing densities achievable with SOS, is believed to be one of the smallest and fastest 5101 RAM's in the industry. Now, a recent study has also shown it to consume the least amount of total power in small and medium system applications

This characteristic is illustrated in Figures 1 and 2. Figure 1 charts total power consumption for a theoretical system design operating at 1 μs and 3.2 μs cycle times. (1 μs is a standard spec sheet value while 3.2 μs represent an 1802 machine cycle at a maximum CPU operating frequency of 2.5 MHz). The chart reflects the individual power components listed in Figure 2 and assumes selection of a single page of RAM (2 - 5101 devices) for each cycle. The RCA device enjoys the considerable saving in total power, even when comparing with a relatively high 500 μA quiescent current (-8) device, because of the internal design structure during the time the chip is selected.

Unfortunately, this dramatic realization is not apparent when reading manufact-uurers' data sheets, and is difficult to assess without an understanding of the internal chip design. Attachment 1 attempts to give a short course on power considerations for RAM systems, to aid the user in choosing the optimum device for his system. The conclusion is that the RCA device has power advantages over competitive devices analyzed except when the device is used in a battery back-up, deselected, static address bus mode where quiescent current values are the only power consideration in the system design.

For more information on this or related subjects, contact Dick Peck, X7376.

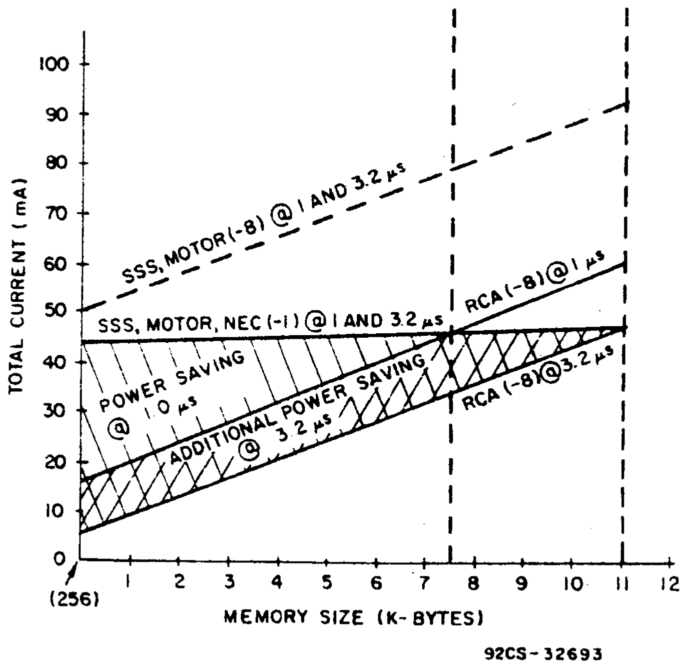


Fig. 1 - Power savings for small and medium systems (using the MWS5101EL-8).

MFR	TYPE DESIGNATION	VERSION	I_{Quies} (μA)	I_{OPER}* (mA)
RCA	MWS5101	-2 -3 -8	50 200 500	8 8 8
SSS	SCM5101	-1 -3 -8	10 100 500	22 22 25
Motorola	MCM5101P	-1 -3 -8	10 200 500	22 22 25
NEC	μPD5101L	-1	10	22
Hitachi	HM435101	-1	15	22

* 1 μs Cycle time

Fig. 2 - Manufacturers' data sheet values for I_{QUIES} and I_{OPER}.

ATTACHMENT # 1

HOW TO DETERMINE YOUR SYSTEM'S POWER REQUIREMENTS AND CHOOSE A RAM BASED ON PUBLISHED DATA

FACT 1. A memory device can operate in one of 3 modes in a real system

MODE	CHIP SELECT	ACTIVITY ON INPUT LINES
Quiescent	Deselected	No
Standby	Deselected	Yes
Operating	Selected	Yes

FACT 2. It has been observed that manufacturers generally specify current in the quiescent and operating modes only, and may use quiescent and standby modes interchangeably.

FACT 3. Depending on the manufacturer, specified current may or may not be a function of cycle time.

MODE	IS CURRENT A FUNCTION OF CYCLE TIME?	
	COMPETITION	RCA
Quiescent	No	No
Standby	Yes	No
Operating	No	Yes

The above chart requires some explanation and is the key to understanding how to interpret data sheets. The quiescent mode is straightforward - quiescent implies no activity, and the quiescent values shown in the spec sheets will accurately reflect worst-case values for inactive, battery-back-up menory systems. In the standby mode, lab measurements have shown that competitive devices exceed quiescent values, even though this is generally not stated. The RCA design uses small input buffers, with negligible effect on current as frequency is increased. Since any system that is not in a back-up mode generally has activity on the address but even during deselection, this consideration is important and may involve independent characterization of the selected vendor's devices.

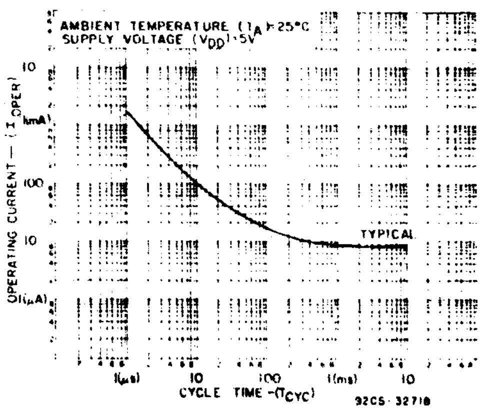
The effect of operating mode current is the most dramatic in this analysis. The competitive devices analyzed drew currents listed in Figure 2 under DC conditions - chip selected, with no activity on the input lines. Under the same conditions, the RCA devices again drew only quiescent current and thus save considerable power during select conditions. The competitive devices are essentially unaffected by cycle time during chip select because of the high initial currents; the RCA devices possess a linear power vs. frequency characteristic and draw maximum current only at minimum cycle times.

The above considerations indicate that if a system is in a standby or operating mode (with 1 pair of devices operating and the others in a standby mode), the RCA device number savings will increase as system memory is decreased and/or operating frequency is decreased. It also indicates that published specifications alone may not be adequate in analyzing a vendor's device for power consumption characteristics.

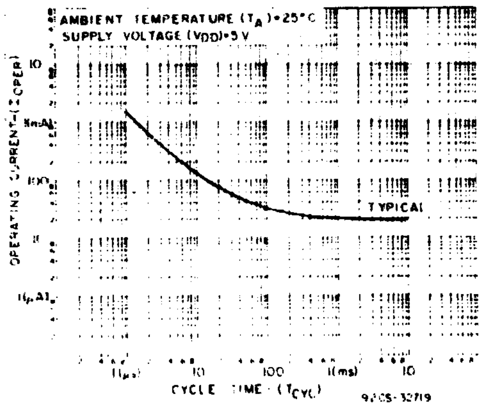
OPERATING CURRENT DERATING CURVES FOR THE 256x4 RAM

(I_{OPER} VS T_{CYC})

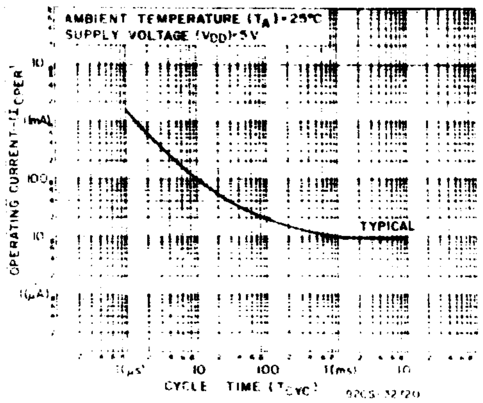
To allow users of the RCA 256x4 CMOS RAM's a means of derating the I_{OPER} values in the data sheet to match their specific application, derating curves for the MWS5101EL-2, MWS5101EL-3, and the CDP1822E are presented below:



MWS5101EL-2

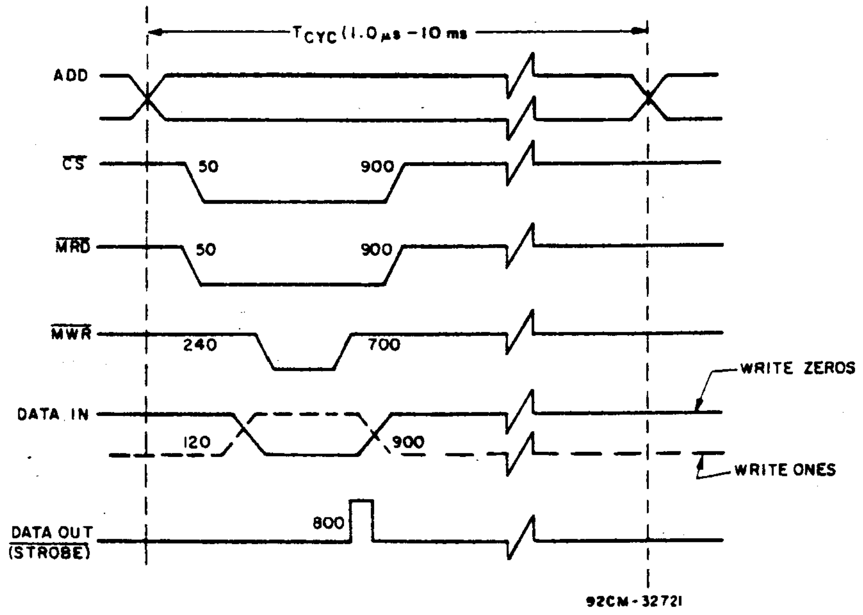


MWS5101EL-3



CDP1822E

The data for these curves was generated under the following conditions:



TIMING DIAGRAM

NOTE: All timing values are in nano-seconds, unless otherwise specified.

Voltage/Temp. V_{DD} = 5.0 V, T_A = 25°C

No. of Devices Tested

MWS5101EL-2	- 38
MWS5101EL-3	- 19
CDP1822E	- 20

Pattern Used

Writing either ONES or ZEROS using the above timing was found to be the worst case. Maximum activity (transitions) on all input lines is more important than a specific pattern.

These curves and the following I_{OPER} (Typ) values will be added to the next update of the 1822 and 5101 data sheets:

Cycle Time	Description	Present Data Sheet Value	New Data Sheet Value
1 μs	Present Data Sheet Condition	4 mA	2 mA
3.2 μs	Min. Time in 1802 System @ 2.5 MHz Clock Rate	(not spec'd)	400 μA

For more details or specific applications information contact Joe Paradise, X7352.

Synchronous versus Asynchronous RAMs . . . Which is Better?

Let's start out with some definitions:

Asynchronous - Refers to ALL RCA RAM devices which accept address inputs which remain valid for an entire READ or WRITE cycle. These address inputs hook up directly to on-board address decoders which select the addressed memory cell.

Synchronous - Refers to RAM's (6501, 6508, 6514) and EPROMS (6653, 6654) made by Harris, Intersil, et al which accept address inputs which are valid for only part of a READ or WRITE cycle. These address inputs hook up to on-board address latches which require a strobe pulse (↓) to latch. Addresses can then become invalid without affecting memory addressing.

Now that we understand these terms, what are the system implications to the user? The synchronous devices were originally designed to be used with the 6100, a 12-bit multiplexed address/data bus processor that generates the appropriate timing signal to latch addresses before they change into data. For processors like the 6100, synchronous memories are the appropriate choice.

For processors that generate stable addresses for a full machine cycle and the appropriate chip select ↓ edge, synchronous and asynchronous devices can be used interchangeably. So RCA 5101 and 5114 parts can find use in these system designs.

A problem with synchronous devices develops when they are used in an 1800 system. The 1802 employs a multiplexed address bus, with a TPA pulse to latch the high-order address. However, there is no convenient pulse to latch the low-order byte when it becomes valid. This is not a problem for an asynchronous RCA RAM, but it does require external devices and loss in performance when using the synchronous devices. The general procedure is to delay TPA with external flip-flops so the TPA pulse lines up with the low-order address byte (details on specific implementation will follow in a future Newsletter). Since this cuts into available RAM or EPROM access time, it may require that the CPU frequency be reduced to accommodate this arrangement. So where an equivalent RCA RAM function is available - 5101 or 5114 - the RCA <u>asynchronous</u> device is the <u>better</u> choice.

Conclusion? Use RCA <u>asynchronous</u> RAMS whenever available for use in 1800 series or other 8-bit processor systems. No additional components are required to accommodate these devices, and no system performance compromise is needed with an 1802 CPU.

For more information, contact Joe Paradise - X7352.

The Synchronous RAM Syndrome

In a previous newsletter some basics on synchronous and asynchronous RAM's and ROM's were defined, concluding that asynchronous devices are the better choice for 1800-Series systems. It is recognized, however, that customer applications will, in some cases, require the use of synchronous devices. In response to any inquiries, the following information should provide the necessary design limitation specifications.

The basic problem encountered when using synchronous devices in CDP1802-based systems is that of providing a latch pulse for the address and enable inputs. The CDP1802 provides two consistent timing pulses (TPA and TPB), although each have inherent limitations in this application. TPB occurs too late in the machine cycle to allow sufficient access time and TPA is not valid during the lower order address time of the multiplexed address information. The ideal solution is to delay the TPA pulse enough to provide valid address information without reducing the available read access time. (See Fig. 1).

In practice, however, some design trade-offs are required to meet this objective. In order to illustrate these trade-off requirements, a fully buffered COS/MOS CDP1802 system (5V), interfacing with eight (8) IM6508 (Intersil) 1Kx1 synchronous RAM's and one (1) IM6654 (Intersil) 512x8 synchronous EPROM, is used, along with a worst-case analysis of the circuit. Although many circuit options may be used to delay TPA, the example shown indicates the problem areas for worst-case operation.

The schematic for this example (Fig. 2) has three problem areas of concern for worst-case design:

Prob. 1 - A delayed TPA pulse must be provided to latch the lower-order address and chip enable inputs for the synchronous RAM/ROM devices.

Prob. 2 - The higher-order address byte must be latched by the high-to-low transition of TPA to provide the A8 and A9 addresses and the chip enable inputs to the IM6508's and the IM6654.

Prob. 3 - Proper write timing must be provided for the IM6508 RAM's.

Each area is examined separately, with references to Fig. 2, Fig. 3A, Fig. 3B, and Table I, in the following text. Table I provides a summary of delay times for the devices used.[1]

Prob. 1 - A CD4013B (Dual D Flip-Flop) is used to provide a latch strobe for the IM6508 RAM's and the IM6654 EPROM. The latch strobe is gated with the proper chip select inputs from the CDP1859. The CD4013 circuit used delays TPA by 2 CPU clock cycles. This is accomplished by limiting the CPU Clk frequency to 2 MHz (T = 500 ns), which prevents TPA (450 ns TPLH -max.) from overlapping Clk cycle 20. Since TPA is the data input (D1) of the first flip-flop, the required data set-up time (TS = 48 ns -min.) is maintained. Clk cycle 20-low sets Q1 high, which is the data input (D2) of the second flip-flop. Clk cycle 30-low then sets Q2 high. Q2 is gated with the $\overline{\text{CE0}}$ and $\overline{\text{CE2}}$ inputs to provide the latch strobes ($\overline{\text{E}}$, $\overline{\text{E1}}$) to the IM6508C and IM6654M synchronous memories on the Clk cycle 40-low. (See Fig. 2A).

The only restriction with this delayed-TPA circuit is that the CD4050 buffers used in the CLK and TPA lines must be in the same package and/or selected such that the propagation delay (TPLH, TPHL) of the TPA buffer does not exceed the propagation delay of CLK buffer.

The available access time for this circuit is given below and is adequate for the synchronous memories (TELQV = TE_1LQV = 600 ns).

Available access time: T_{ACC} = 3.5T-t5-t1-t8-t11

= 1750 - 360 - 132 - 300 - 150

= 808 ns

Prob. 2 - A CDP1859 (4-bit latch with decode) is used to latch a higher-order address byte (HOB) and provide two chip enable outputs. MAØ and MA1 are latched by TPA and become A8 and A9 for the IM6508 RAM's, and A8 for the IM6654 EPROM. MA6 and MA7 are latched by TPA and decoded to become $\overline{\text{CE0}}$ - $\overline{\text{CE3}}$. $\overline{\text{CE0}}$ is used to select the IM6508 RAM's at address 0000_{16} - $3FFF_{16}$ $\overline{\text{CE2}}$ is used to select the IM6654 EPROM at address 8000_{16} - $BFFF_{16}$ (See Fig. 2A). The design limitations in this case are the memory address-to-clock set-up time (t_{MACL}) of the CDP1859 and the HOB set-up-to-TPA (t_{SU1}) of the CDP1802:

$$t_{MACL} = 40 \text{ ns (max)}$$

$$t_{SU1} = 2T - 800 \text{ (max)} \qquad T = \frac{1}{f_{CLOCK}}$$

$$t_{MACL} = t_{SU1} = 2T - 800$$

$$40 = 2T - 800$$

$$T = \frac{800 + 40}{2} = 420 \text{ ns (max CPU clock freq.)}$$

Prob. 3 - The write timing for the IM6508 RAM's is provided by the CDP1802 $\overline{\text{MWR}}$ signal. The IM6508 data in (D) and data out (Q) and the IM6654 Q0 - Q7 lines are buffered with two CDP1856's (4-bit bus buffers/separators) to prevent bus contention during non-memory CPU cycles. (See Fig. 2B). Although the system shown easily meets the data sheet write timing, The design limitations in this case are the IM6508 $\overline{W}$ pulse width (T_{WLWH}) and the data set-up time (T_{DVEH}):

IM6508 Specifications:

T_{WLWH} = 395 ns (min)

T_{DVEH} = 395 ns (min)

For the circuit shown:

t_{WLWH} = 2 T = 1000 ns (min)

t_{DVEH} = 5.5 T-t14-t16+t12+t1

= 2750-650-150+450+132

= 2532 (min)

Conclusion

The worst-case example shown should provide the necessary design guidelines for other similar systems, where one or more of the problem areas discussed may be critical to proper system operation. Although circuit operation is limited to 2.00 MHz, its use should still be valuable in many customer applications.

1 - RCA COS/MOS Integrated Circuits Manual (SSD-250A)
RCA COS/MOS Memories, Microprocessors, and Support Systems Manual (SSD-260)
Intersil Data Book (July 1979)

For more information contact R. Vaccarella, X6542.

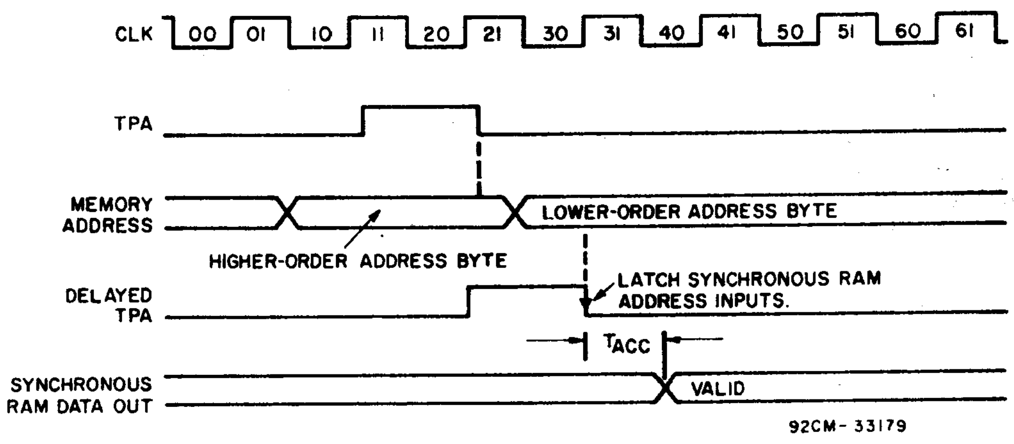


Fig. 1 - Delayed TPA Timing

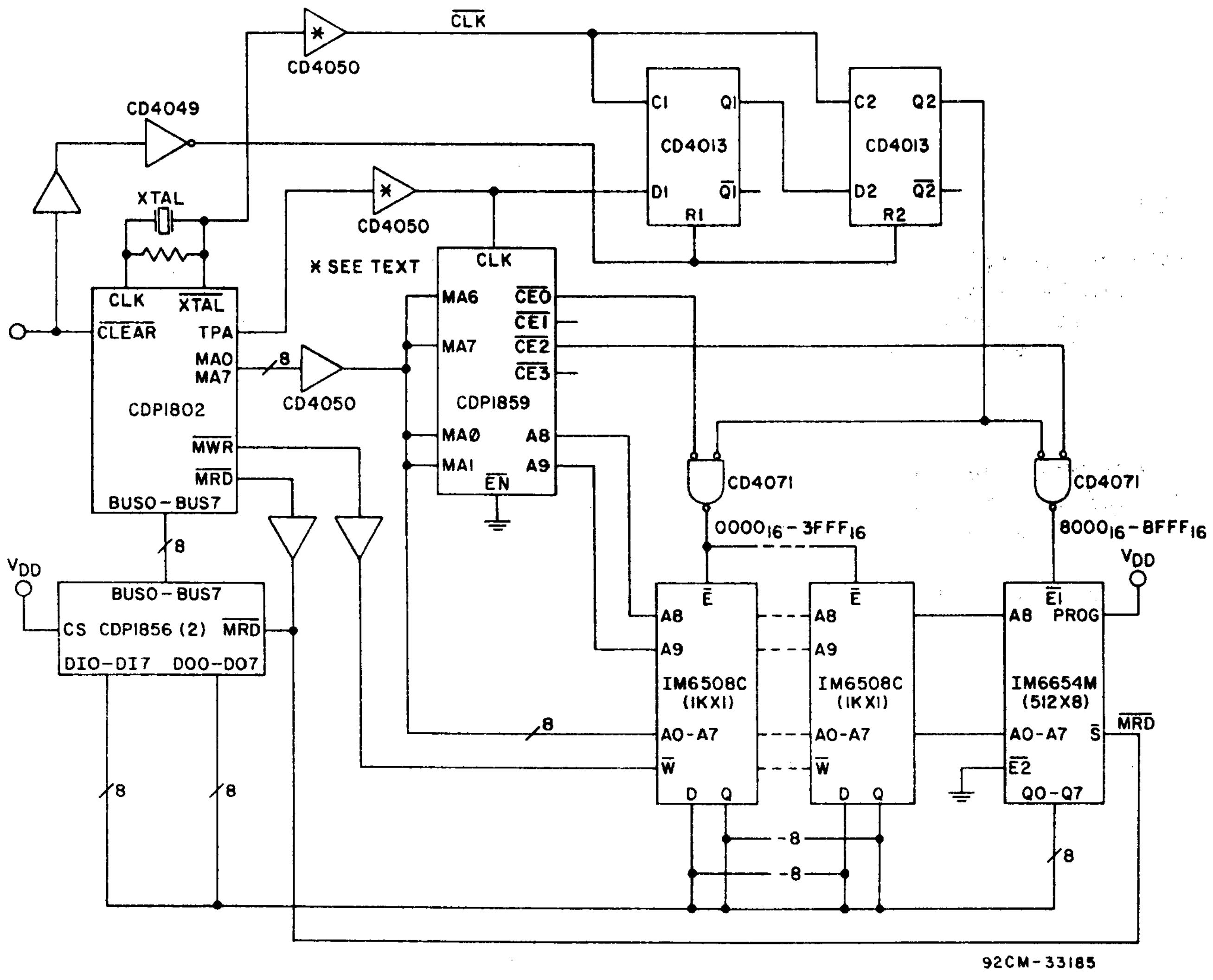


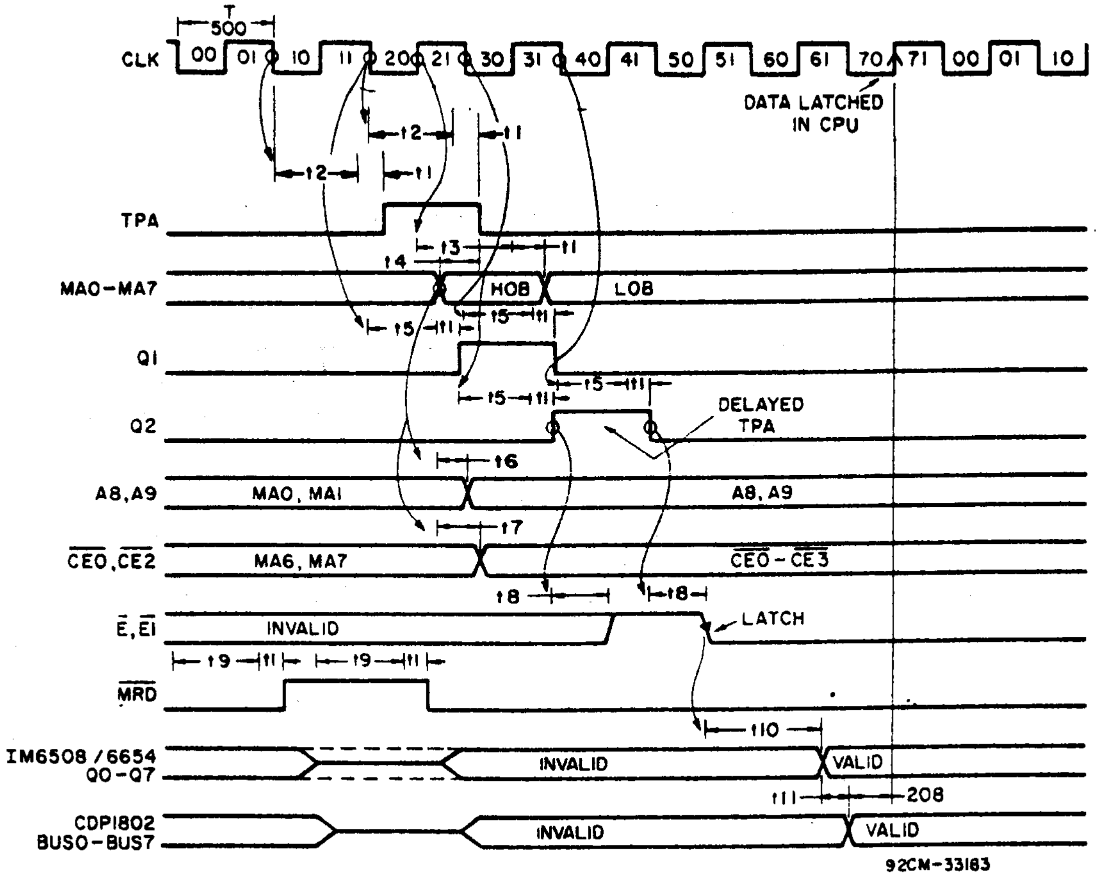
Fig. 2 - CDP1802 Synchronous RAM/ROM System

CDP1802 Synchronous RAM/ROM System

TABLE I - Delay Times

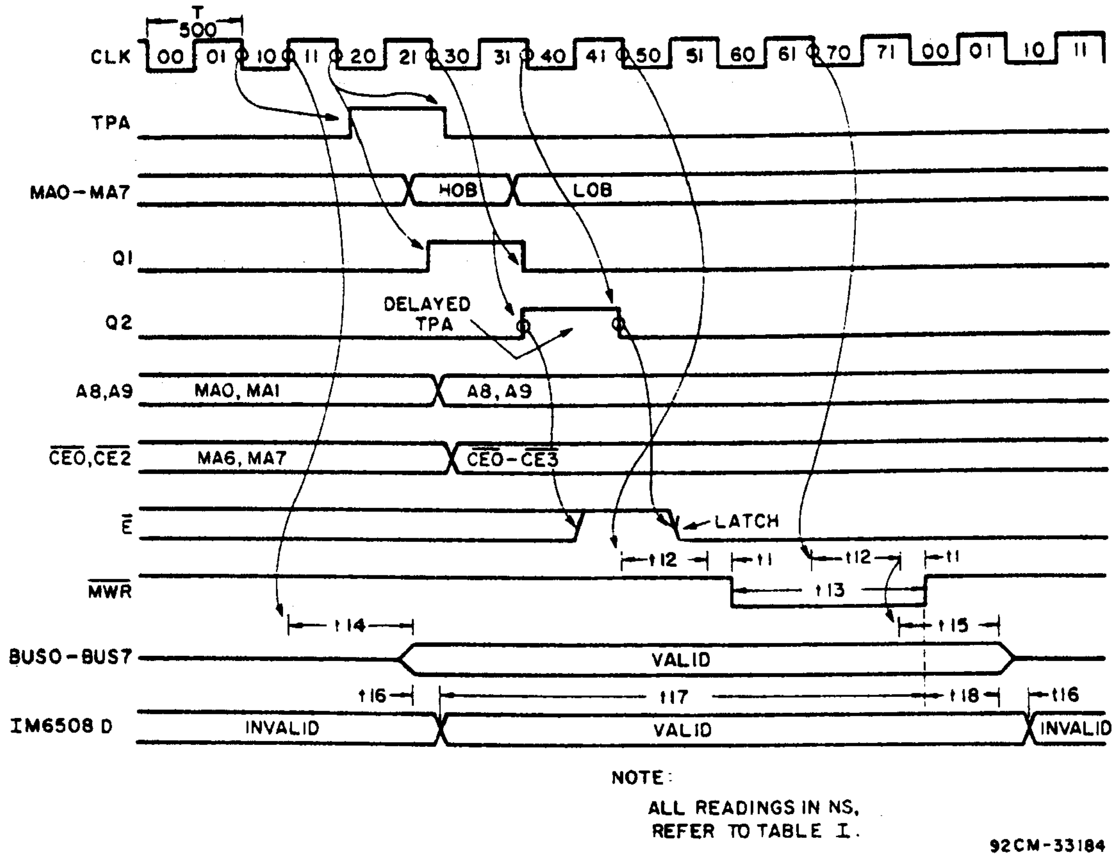
V_{DD}, V_{CC} = 5.0V ● Typical values are for T_A = 25°C

Device	Ref.	Delay (ns) -40 to +85°C Min.	Typ.●	Max.
CDP1802				
Clk to TPA (T_{PLH}, T_{PHL})	t_2		300	450
Clk to Lower Order Address (T_{PLH}, T_{PHL})	t_3		350	500
Higher Order Address Set-up to $\overline{TPA}$ (T_{SU1})	t_4	2T-800	2T-600	
Clk to $\overline{MRD}$ (T_{PLH}, T_{PHL})	t_9		300	450
Clk to $\overline{MWR}$ (T_{PLH}, T_{PHL})	t_{12}		300	450
Clk to CPU Data Bus	t_{14}		450	650
CPU Data to Bus Hold After WR Time (T_H)	t_{15}	T + 25	T + 120	
Data Set-up (T_{SU2})	-		-30	0
CDP1859				
Memory Address to Address (T_{MAA})	t_6		100	150
Memory Address to $\overline{CE}$ (T_{MACE})	t_7		150	225
Memory Address to Clock Set-up (T_{MACL})	-		25	40
$\overline{ENABLE}$ to $\overline{CE}$ (T_{ECE})	-		125	200
CDP1856				
DI to DB Delay (T_{IB})	t_{11}		100	150
DB to DO Delay (T_{BD})	t_{16}		100	150
CD4013B				
Clk to Q, $\overline{Q}$ (T_{PLH}, T_{PHL})	t_5		150	360
Data Set-up Time (T_S)	-	48	20	
CD4049UB				
Propagation Delay (T_{PLH}/T_{PHL})	-		60/32	144/78
CD4050UB				
Propagation Delay (T_{PLH}/T_{PHL})	t_1		70/55	168/132



NOTE:
ALL READINGS IN NS, REFER TO TABLE I.

Fig. 3(a) - READ Timing



NOTE:
ALL READINGS IN NS, REFER TO TABLE I.

Fig. 3(b) - WRITE Timing

CDP1802 Synchronous RAM/ROM System

TABLE I - Delay Times (cont'd)

V_{DD}, V_{CC} = 5.0V ● Typical Values are for T_A = 25°C

Device	Ref.	Delay (ns) -40 to +85°C		
		Min.	Typ●	Max.
CD4071B				
Propagation Delay (T_{PLH}, T_{PHL})	t_8		125	300
IM6508C				
Access Time from $\overline{E}$ (T_{ELQV})	t_{10}			600
$\overline{W}$ Pulse Width (T_{WLWH})	t_{13}	395		
Data Set-up Time (T_{DVEH})	t_{17}	395		
Data Hold Time (T_{EHDX})	t_{18}	0		
Address Set-up Time (T_{AVEL})	-	20		
Address Hold Time (T_{ELAX})	-	170		
IM6654M				
Access Time from $\overline{E}_1$ (T_{E1LQV})	t_{10}			600
Address Set-up Time (T_{AVE1L})	-	0		
Address Hold Time (T_{E1LAX})	-	100		
$\overline{E}_1$ Pulse Width (positive) (T_{E1HE1L})	-	150		

MASKED ROM PROGRAMMING INSTRUCTIONS

Because of the confusion associated with some of the nomenclature in the present PD-30 Guide, and because alternate methods for submitting ROM data are now available to CDP1800 Series users, the intent of this article is to clearly and simply state the procedures required to submit masked ROM data to RCA for fabrication of custom ROM devices. A detailed, expanded, revised PD-30 Guide with specific information on each of the masked ROM devices offered by RCA is currently in preparation.

METHODS OF SUBMITTING ROM DATA

ROM program/format data can be submitted to RCA in three ways: CARD DECK, DISKETTE, or EPROM.

CARD DECK METHOD - Use standard 80 column computer cards. Deck must contain title card, option card, format card, and data cards. All cards must be punched per the instructions in the PD-30 publication.

DISKETTE METHOD - The diskette contains the ROM address and data information only. Title, option, and data format information which would otherwise be punched in card deck format can be submitted via the attached information sheet. In addition, the user must specify the RCA development system used in generating the diskette - CDP18S005 or 007 (CDOS) - as well as track number or file name. Finally, a diskette listing should be submitted, to verify its contents.

EPROM METHOD - Data can be submitted via EPROMS, programmed using the CDP18S480 PROM programmer; e.g., 2704, 2708, 2716, 2732, or 2758 types (RCA CDP18U42 by Jan., 1981). If the ROM to be manufactured is smaller (in memory size) than the EPROM carrying the data pattern, or if more than one ROM data pattern is stored in that EPROM, then the starting address and size of each pattern must be stated on the information sheet, along with normal title, option, and data format information. In addition, an EPROM listing, including address and data information, should be submitted to verify the EPROM contents.

INTERPRETING THE PD-30 OPTION CARD TABLES

Experience has shown this to be the most confusing aspect of the information presented in the ROM programming guide. Allowable options for ROM control signals and addresses (specifically, for the CDP1833) are explained below:

CHIP SELECT OPTION - Chip select pins CS1 and CS2 can be enabled by an active high (logic 1) level - programmed by P, an active low (logic 0) level - programmed by N, or always enabled (don't care) - programmed by X, in columns 28 and 29. If X is chosen, this means selection of the device depends only on the remaining CS and or MA lines that are not defined as X (don't care).

$\overline{\text{MRD}}$ HAS NO OPTION - Always enter N in column 30 of the card/sheet.

CEO HAS NO OPTION - always enter X in column 31 of the card/sheet.

CEI OPTION - this input will be active if a P is entered in column 32. CEI will have no effect on circuit operation if an X is entered.

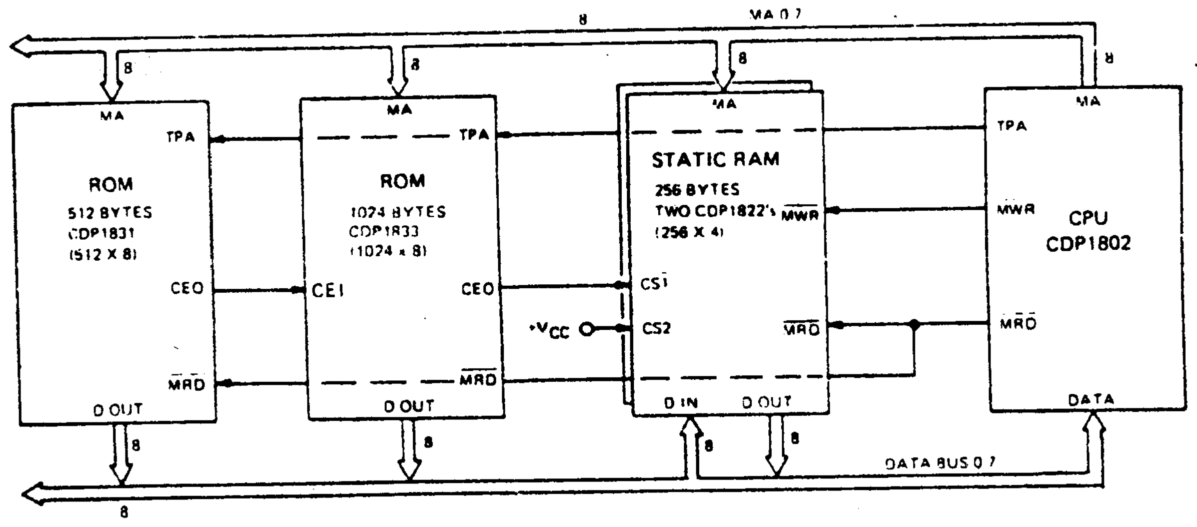
TPA OPTION - enter P in column 34 if the CDP1802 TPA pulse is fed directly to this input. An N can be entered if a different, active low, pulse is to be used.

HIGH-ORDER ADDRESS OPTION - the ROM data will be located in one contiguous block of memory (1024 words for the CDP1833). That block, via user option, can be located anywhere in the 65K of memory space addressed by the CDP1802. The user selects 1 of 64 blocks by programming the MA15-MA1Ø lines with the appropriate code. Entering a P corresponds to an active high (logic 1) level for that address decoder input, while an N corresponds to an active low (logic 0) level. Thus, entering all N's in columns 36-41 locates the block in the lowest 1K of memory space (address locations 0000-03FF). Entering all P's locates the block in the highest 1K memory space (address locations FC00-FFFF). If X is chosen for one or more of the MA pins, multiple mapping will occur : the same block will appear in several locations throughout the 65K memory space.

ALL OPTIONS - must be defined. Do not leave blank!

Note that pins defined as X are still CMOS inputs and must be tied high or low on the circuit board, even though their function is meaningless. For lowest power consumption all pins defined as X should be tied to V_{SS}, and all unused memory bits should be programmed as logic zeros.

EXAMPLE



The above diagram shows a typical system with RAM and ROM memory. The customer desires memory mapping as follows:

Address	
0 0 0 0 to 0 3 F F	1024 bytes of ROM, CDP1833
0 4 0 0 to 0 5 F F	512 bytes of ROM, CDP1831
* 0 0 to * F F	256 bytes of RAM, two CDP 1822's

* any high order address 06 to FF.

Since the RAM $\overline{\text{CS1}}$ is driven from the daisy chained CEO of the ROM's, the RAM is addressed only if both ROM's are not addressed.

Information Sheet for ROM Programming Using DISKETTE or EPROM
(Example sheets attached)

The customer is submitting data for the CDP1833 on a diskette, prepared using the MEM SAVE software program on a CDS18S007 development system. A file name of "ROMDAT" has been assigned to the ROM data file. An information sheet has been filled out as required.

The data for the CDP1831 ROM is submitted via a 2704 EPROM. Another information sheet has been filled out for this ROM.

For additional information, see PD-30 Guide, or contact Jerry Johnson, X6776.

Information Sheet for ROM Programming Using DISKETTE or EPROM

TITLE

COLUMN		
1	T	Customer Name (start at left)
6 - 30		
35 - 54		Division
59 - 63		RCA Custom Number
65 - 71		ROM Type
72	(	
73	8	
74	)	
79 - 80		ROM #

HOW IS ROM DATA SUBMITTED?

☐ DISKETTE) or) Check One ☐ EPROM)

OPTIONS

COLUMN		
1 - 6	O P T I O N	
8 - 17		ROM Type
28		
29		
30		STARTING ADDRESS
31		OF DATA BLOCK
32		IN ROM TO BE
34		MANUFACTURED ☐☐☐☐
36		
37		
38		
39		
40		
41		
42		
79 - 80		ROM #

ANSWER THE FOLLOWING QUESTIONS IF DISKETTE IS SUBMITTED:

Type of System?

☐ CDS18S005)
☐ CDS18S007) Check One
☐ CDS18S008)

■ If CDS18S005 System was used, specify:

TRACK # ☐☐

Software used:

☐ ROM SAVE } Check One
☐ SAVE (PROM)

■ If CDS18S007/8 System was used, specify:

FILE NAME ____________

Software used: ☐ (MEM SAVE
check one ☐ (SAVE (PROM)

DATA FORMAT

COLUMN		
1 - 11	D A T A F O R M A T	
13 - 15		
17 - 19		
79 - 80		ROM #

If EPROM is submitted, state

Type: ____________

Starting Address of Data Block in EPROM: ☐☐☐☐

USE SEPARATE SHEET FOR EACH ROM PATTERN

Information Sheet for ROM Programming Using DISKETTE or EPROM

TITLE

COLUMN		
1	T	Customer Name (start at left)
6 - 30	ALPHA CONTROLS CO	
35 - 54	DISPLAYS DIVISION	Division
59 - 63	94321	RCA Custom Number
65 - 71	1833CE	ROM Type
72	(	
73	8	
74	)	
79 - 80	01	ROM #

Example

HOW IS ROM DATA SUBMITTED?

[✓ DISKETTE] or EPROM — Check One

OPTIONS

COLUMN		
1 - 6	OPTION	
8 - 17	CDP1833CE	ROM Type
28	X	
29	X	
30	N	STARTING ADDRESS
31	X	OF DATA BLOCK
32	P	IN ROM TO BE
34	P	MANUFACTURED 0000
36	N	
37	N	
38	N	
39	N	
40	N	
41	N	
42		
79 - 80	01	ROM #

ANSWER THE FOLLOWING QUESTIONS IF DISKETTE IS SUBMITTED:

Type of System?

CDS18S005 [✓ CDS18S007] CDS18S008 — Check One

■ If CDS18S005 System was used, specify:

TRACK #

Software used:

ROM SAVE / SAVE (PROM) — Check One

■ If CDS18S007/8 System was used, specify:

FILE NAME ROMDAT

Software used: check one [✓ (MEM SAVE] (SAVE (PROM)

DATA FORMAT

COLUMN		
1 - 11	DATA FORMAT	
13 - 15	HEX	
17 - 19	POS	
79 - 80	01	ROM #

If EPROM is submitted, state

Type: ____________

Starting Address of Data Block in EPROM: ☐☐☐☐

USE SEPARATE SHEET FOR EACH ROM PATTERN

Information Sheet for ROM Programming Using DISKETTE or EPROM

TITLE

COLUMN		
1	T	Customer Name (start at left)
6 - 30	ALPHA CONTROLS CO	
35 - 54	DISPLAYS DIVISION	Division
59 - 63	84322	RCA Custom Number
65 - 71	1831CE	ROM Type
72	(	
73	8	
74	)	
79 - 80	02	ROM #

Example

HOW IS ROM DATA SUBMITTED?

☐ DISKETTE or [✓ EPROM] Check One

OPTIONS

COLUMN		
1 - 6	OPTION	
8 - 17	CDP1831CE	ROM Type
28	X	
29	X	
30	N	STARTING ADDRESS
31	X	OF DATA BLOCK
32	X	IN ROM TO BE
34	P	MANUFACTURED 0400
36	N	
37	N	
38	N	
39	N	
40	N	
41	P	
42	N	
79 - 80	02	ROM #

ANSWER THE FOLLOWING QUESTIONS IF DISKETTE IS SUBMITTED:

Type of System?
☐ CDS18S005
☐ CDS18S007
☐ CDS18S008
Check One

■ If CDS18S005 System was used, specify:
TRACK #

Software used:
☐ ROM SAVE
☐ SAVE (PROM)
Check One

■ If CDS18S007/8 System was used, specify:
FILE NAME ____________

Software used: ☐ (MEM SAVE
check one ☐ (SAVE (P

DATA FORMAT

COLUMN		
1 - 11	DATA FORMAT	
13 - 15	HEX	
17 - 19	POS	
79 - 80	02	ROM #

If EPROM is submitted, state
Type: 2704

Starting Address of Data
Block in EPROM: 0000

USE SEPARATE SHEET FOR EACH ROM PATTERN

CDP18U42 - 1702 EPROM Comparison

Since the RCA CDP18U42 EPROM is a functional replacement (except for the power supplies) for the 1700 series UV erasable PROM's, the following comparison of these 2 types may be helpful to our Sales and FTS organization. Clearly, the CDP18U42 is considerably easier to use.

Programming Summary:

CDP18U42

Initially, all 2048 bits of the PROM are in the "0" state (output low). Programming is done on a byte basis, in which any erased bits may be selectively programmed to a "1" state (output high) at any address location or in any sequence. The V_{CC}, V_{SS}, and address inputs function the same as in the read mode. One program voltage (18-25V) is used to pulse (10 ms) the V_{DD} and V_{sat} inputs simultaneously. The data output terminals become the true data inputs and, as with the address inputs, have the same logic and voltage levels as in the read mode. Since each location need only be programmed once, the total programming time is 2.6 sec. (Refer to the CDP18U42 data sheet for specific information).

1700 Series

"Initially, all 2048 bits of the PROM are in the "0" state (output low). Information is introduced by selectively programming "1"'s (output high) in the proper bit locations. Word address selection is done by the same decoding circuitry used in the read mode. All 8 address bits must be in the binary complement state when pulsed V_{CC} and V_{GG} move to their negative levels. The addresses must be held in their binary complement state for a minimum of 25 μs after V_{DD} and V_{GG} have moved to their negative levels. The addresses must then make the transition to their true state a minimum 10 μs before the program pulse is applied. The addresses should be programmed in the sequence 0 through 255 for a minimum of 32 times. The eight output terminals are used as data inputs to determine the information pattern in the 8 bits of each word. A low data input level (1-48V) will program a "1" and a high data input level (ground) will leave a "0". All 8 bits of one word are programmed simultaneously by setting the desired bit information patterns on the data input terminals. During the programming, V_{GG}, V_{DD} and the program pulse are pulsed with signals."[1] (Refer to the 1700 series data sheet for specific information).

[1] Intel Component Data Catalog 1979, p. 4-68

TABLE 1

CDP18U42 - 1702 EPROM Comparison

CDP18U42	1702
256x8	256x8
Static Operation	Static Operation
Access Time ≤ 1 μs	Access Time .650 - 1.5 μs
24 pin DIP	24 pin DIP
Tri-state Outputs	Tri-state Outputs
Erase Time 5-20 mins.	Erase Time 10-45 mins.
Program Time 2.6 secs. (1 cycle)	Program Time 2 mins. (32 cycles min
Erased to all 0's	Erased to all 0's
Address Inputs (Pins 1-3, 17-21)	**Address Inputs**
Read $=V_{SS} - V_{CC}$ (0, 5V)	Same pin/function (0, 5V)
Program $=V_{SS} - V_{CC}$ (0, 5V) (True)	*Complement/True (0, -48V)
Data Outputs (Pins 4-11)	**Data Outputs**
Read (output) $=V_{SS} - V_{CC}$ (0, 5V)	Same Pin/Function (0, 5V)
Program (input) $=V_{SS} - V_{CC}$ (0, 5V)	*Input = (0, -48V)
V_{CC} (Pin 12)	**V_{CC}**
Read $=V_{CC}$ (5V)	V_{CC} (5V)
Program $=V_{CC}$ (5V)	*Grd (0V)
$\overline{\text{PGM}}$ (Pin 13)	**Program**
Read $= V_{CC}$ (5V)	V_{CC} (5V)
Program $= V_{SS}$ (0V)	*Program Pulse (-48V)
$\overline{\text{CS1}}$ (Pin 14)	**$\overline{\text{CS}}$**
Read $= V_{SS}$ (0V)	Same Pin/Function (0V)
Program $= V_{SS}$ (0V)	Same Pin/Function (0V)

TABLE 1

CDP18U42 - 1702 EPROM Comparison (Cont'd)

CDP18U42	1702
CS2 (Pin 15)	V_{BB}
Read = V_{CC} (5V)	V_{CC} (5V)
Program = V_{CC} (5V)	*V_{BB} (+12V)
$\overline{CS3}$ (Pin 16)	V_{GG}
Read = V_{SS} (0V)	*V_{GG} (-9V)
Program = V_{SS} (0V)	*Pulsed V_{GG} (-40V)
V_{sat} (Pin 22)	V_{CC}
Read = V_{CC} (5V)	V_{CC} (5V)
Program = Pulsed (22V)	*Grd (0V)
V_{DD} (Pin 23)	V_{CC}
Read = V_{CC} (5V)	V_{CC} (5V)
Program = Pulsed (22V)	*Grd (0V)
V_{SS} (Pin 24)	V_{DD}
Read = (0V)	*V_{DD} (-9V)
Program = (0V)	*Pulsed V_{DD} (-48V)

* = Voltage or Function Different

For more information on this or related subjects contact Rick Vaccarella, X6542.

CDP18U42 - IM6654 EPROM Comparison

Since the RCA CDP18U42 and the Intersil IM6654 are both CMOS, single voltage UV erasable PROM's, the following comparison of these two types may be helpful to our Sales and FTS organization. Although the parts are similar, it should be noted that the CDP18U42 does not require an address latch strobe (static operation) and is considerably easier to program.

Programming Summary:

CDP18U42

Initially, all 2048 bits of the EPROM are in the "0" state (output low). Programming is done on a byte basis, in which any erased bits may be selectively programmed to a "1" state (output high) at any address location or in any sequence. The V_{CC}, V_{SS}, and address inputs function the same as in the read mode. One program voltage (18-25V) is used to pulse (10 ms typ.) the V_{DD} and V_{sat} inputs simultaneously. The data output terminals become the true data inputs and, as with the address inputs, have the same logic and voltage levels as in the read mode. Since each location need only be programmed once, the total programming time is 2.6 sec. (Refer to the CDP18U42 data sheet for specific information).

IM6654

Initially, all 4096 bits of the EPROM are in the "1" state (output high). Selective programming of proper bit locations to "0's" is performed electrically at any address or in any sequence.

In the PROGRAM mode, V_{CC} and V_{DD} are tied together to the normal operating supply. High logic levels at all of the appropriate chip inputs and outputs must be set at V_{DD} - 2V minimum. Low logic levels must be set at V_{SS} + 0.8V maximum. Addressing of the desired location in the PROGRAM mode is done as in the READ mode. Address and data lines are set at the desired logic levels, and PROGRAM and chip select $\overline{(S)}$ pins are set high. The address is latched by the high-to-low transition of the strobe input ($\overline{EI}$). During valid DATA IN time, the PROGRAM pin is pulsed (20 ms typ.) from V_{DD} to -40V. This pulse initiates the programming of the device to the levels set on the data outputs. Each location should be programmed four times.

Intelligent programmer equipment with successive READ/PROGRAM/VERIFY sequence is recommended. (Refer to the IM6654 data sheet for specific information).

TABLE I

CDP18U42 - IM6654 EPROM COMPARISON

CDP18U42	IM6654
256x8	512x8
Static Operation	Synchronous operation
Access Time $\leq$ 1 μs	Access Time = .300 - .450 μs
24 Pin DIP	24 Pin DIP
Tri-State Outputs	Tri-State Outputs
Single 5V Supply (READ Mode)	Single 5V Supply (READ Mode) - 10V Part Available
Power Dissipation = 50 μW	Power Dissipation = 5 μW
Erase Time = 5 - 20 mins.	Erase Time = 5 - 20 mins.
Erased to all 0's	Erased to all 1's
Program Time = 2.6 secs. (1 cycle)	Program Time = 40 secs. (4 cycles)
Address Inputs (Pins 1-3, 17-21) (AØ - A7)	Address Inputs (Pins 1-8, 23 (A1 - A8)
Read/Program = V_{SS}-V_{CC} (0, 5V)	Same funct. (0, 5V) - latched by $\overline{E1}$
Data Outputs (Pins 4 - 11)	Data Outputs (Pins 8-11, 13-17)
Read (Output) = V_{SS}-V_{CC} (0, 5V)	Same function (0, 5V)
Program (Input) = V_{SS}-V_{CC} (0, 5V)	Same function (0, 5V)
V_{CC}(Pin 12)	* V_{CC} (Pin 24)
Read = V_{CC} (5V)	Same function (5V)
Program = V_{CC} (5V)	Same function (5V)
$\overline{PGM}$ (Pin 13)	* Program (Pin 18)
Read = V_{CC} (5V)	V_{DD} (5V)
Program = V_{SS} (0V)	* Program Pulse (-40V)
$\overline{CS1}$ (pin 14) (Chip Select)	* $\overline{E1}$ (Pin 20) (Strobe Pulse)
Read = V_{SS} (0V)	* Latches address lines & $\overline{E2}$ (V_{CC}-V_{SS})
Program = V_{SS} (0V)	* Latches address lines & $\overline{E2}$ (V_{CC}-V_{SS})
CS2 (Pin 15) (Chip Select)	* $\overline{E2}$ (Pin 22) (Chip Enable)
Read = V_{CC} (5V)	* Latched by E1 (V_{SS})
Program = V_{CC} (5V)	* V_{SS} (0V)

TABLE I

CDP18U42 - IM6654 EPROM COMPARISON (CONT'D)

CDP18U42	IM6654
$\overline{CS3}$ (Pin 16) ($\overline{\text{Chip Select}}$)	* $\overline{S}$ (Pin 21) ($\overline{\text{Chip Select}}$)
Read = V_{SS} (0V)	Same function
Program = V_{SS} (0V)	* V_{CC} (5V)
V_{sat} (Pin 22)	* N/A
Read = V_{CC} (5V)	
Program = pulsed (22V)	
V_{DD} (Pin 23)	* V_{DD} (Pin 19)
Read = V_{CC} (5V)	Same function (5V)
Program = pulsed (22V)	* V_{CC} (5V)
V_{SS} (Pin 24)	* V_{SS} (Pin 12)
Read = (0V)	Same function (0V)
Program = (0V)	Same function (0V)

* = Voltage, Function, or pin different

For more information on this or related subjects, contact Rick Vaccarella, X6542.

CDP18U42 PROM Programmer Circuit Options

Since the introduction of the RCA CDP18U42 EPROM there have been a number of inquiries as to what hardware may be used to program these parts. The following describes the options available to CDP18U42 users.

Also included is a schematic diagram of a low cost (<$50) programmer circuit. I designed and built this circuit to illustrate the simple programming procedure required with the CDP18U42 EPROM. Its operation in intentionally limited to the basic requirements of the CDP18U42 to reduce design and construction costs.

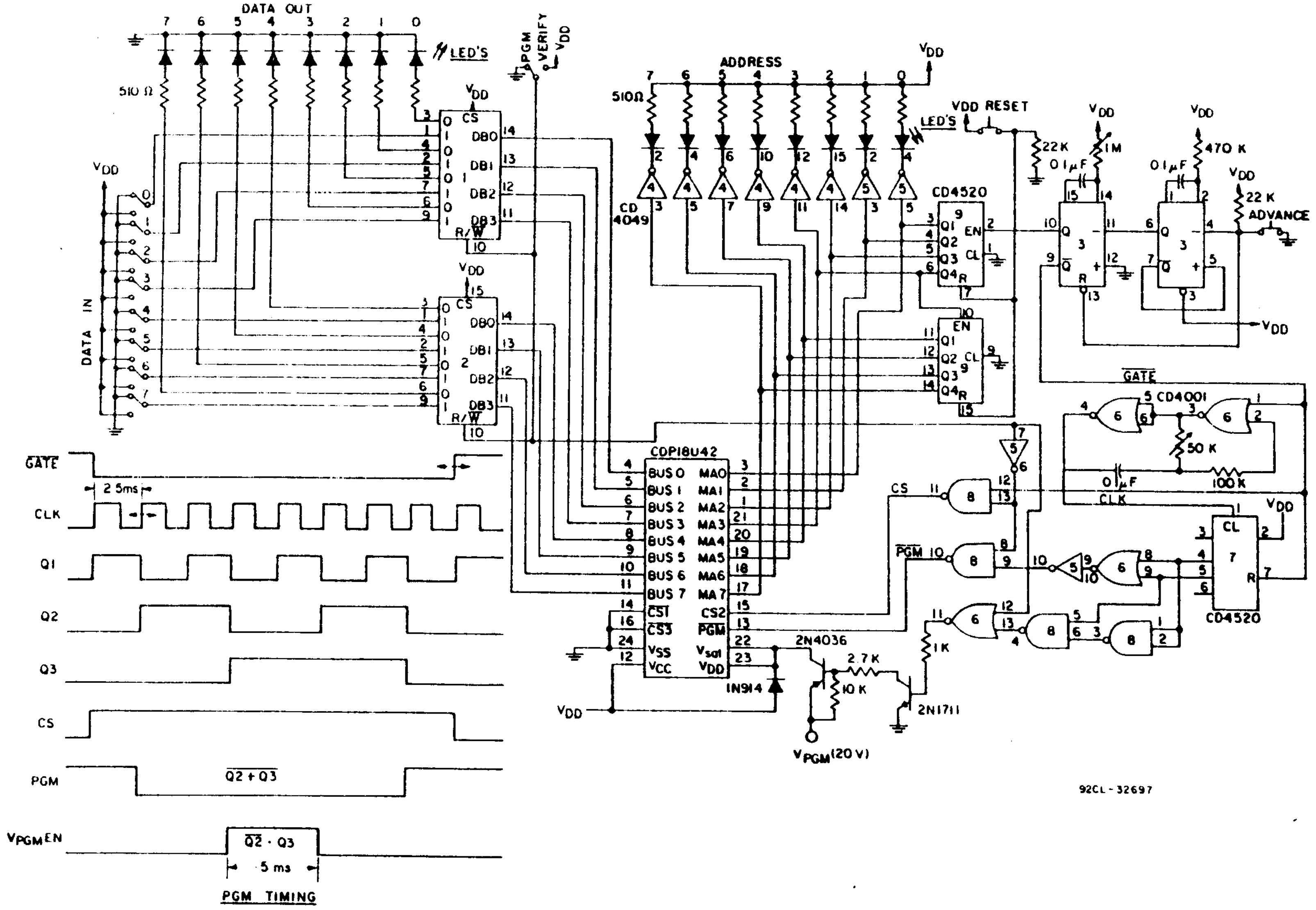
Two modes of operation are available; program and verify. In the program mode, data is entered in binary using eight toggle switches at the address indicated by the ADDRESS LED's. After releasing the ADVANCE switch, a timing cycle is initiated in which the CDP18U42 is put in the program mode and the program voltage (20V) is applied to the V_{DD}, V_{sat} pins. At the end of this timing cycle the address counter is incremented by one and the CDP18U42 is ready to accept the next data byte.

After programming is completed, the CDP18U42 may be checked by placing the PGM/VERIFY switch in the VERIFY position. In this mode the CDP18U42 is selected in the read mode and the program voltage circuit is disabled. After depressing the RESET switch, each data byte may be read on the DATA OUT LED's, starting at address zero, using the ADVANCE switch.

Although the basic timing requirements of the CDP18U42 are provided by this circuit, other features may be easily added, such as a hex keypad circuit, to enhance programming efficiency.

As indicated by the other programming options, this circuit can provide an effective low cost alternative for the low volume CDP18U42 EPROM user.

Other circuits similar to this may also be used with the CDP18S020 Evaluation Kit to provide more sophisticated programming and verifying operations with user written software.



CDP18U42 PROM Programmer

ATTACHMENT # 1

CDP18U42 PROGRAMMING CIRCUIT OPTIONS

1. COSMAC Development System II (CDP18S005)

- Sophisticated software techniques (auto-reprogram, listings, RAM backup, etc.)
- Requires only the PROM programmer card (CDP18S402) and the appropriate software, available in diskette (CDP18S480), paper tape (CDP18S490VI), and cassette (CDP18S480V2).
- Universal programmer approach
- External power supplies - 22-26V - 50 mA (1842, 2704,2708,2716,2758)
  9V - 70 mA)1702, Read Only)
- OEM support
- All CDS features available to user
- User must buy or already own a CDS and provide the appropriate interface (disk, paper tape, or cassette).

- CDS II	$3200.00
- PROM Programmer	700.00
- Floppy Disc	3600.00 (Optional)

2. COSMAC Microboard Prototyping System (CDP18S691)

- Sophisticated software techniques (auto-reprogram, listings, RAM backup, etc.)
- Requires only the PROM programmer card (CDP18S402) and the appropriate software, available in paper tape (CDP18S480VI) and cassette (CDP18S480V2)
- Universal programmer approach
- External power supplies - 22-26V - 50 mA (1842, 2704, 2708, 2758, 2716)
  9V - 70 mA (1702, Read Only)
- OEM support
- All Microboard features available to user
- No floppy disc interface available; utility program does not provide a disk loader routine.

- Microboard Prototyping System	$990.00
- PROM Programmer	700.00

3. COSMAC Evaluation Kit (CDP18S020)

- Relatively inexpensive ($249.00)
- Easy to use
- User prototyping area provided on-board
- OEM support
- No floppy disk interface available
- User supplied ASCII terminal or COSMAC microterminal (CDP18S021 - $140.00)
- User designed and built PROM programmer circuit
- User written software
- External power supplies required (+5V, +20V)

4. Commercial PROM Programmers (None presently available for the CDP18U42, although contact has been made with PRO-LOG and DATA I/O)

- A number of OEM programmer sources are available to the user, offering a variety of self-contained units
- OEM support
- Universal programmer designs, allowing for new type introductions via plug-in modules
- Ease of use by non-technical personnel
- Some lag time associated with the introduction of plug-in modules for new types.
- Because of the universal design approach, the user pays for options and features he may not utilize

5. User Designed PROM Programmer Circuit

- Ease of programming the CDP18U42 minimizes the circuit design and software requirements
- User can tailor the circuit to his particular requirements
- Cost would be determined by a price/performance tradeoff
- The attached schematic is an example of a minimal programming circuit costing about fifty dollars.
- User must invest time and mony for design and construction
- User must develop his own software
- No OEM support of his design

For more information on this or related subjects contact Rick Vaccarella, X6542.

CDP1802-Based PROM Programmer Circuit for the CDP18U42 EPROM

Based on a circuit suggested by F. Thorley (FTS - Birmingham) and J.A. Stahler (FTS - Des Plaines) a microprocessor-based PROM programmer for the low-end CDP18U42 EPROM user has been designed and built. The cost of the programmer is approximately $50. (excluding the terminal or MICROTERMINAL). The circuit can be easily constructed in the user-area of the RCA Evaluation Kit (EK) or as a stand-alone design, as shown in the attached diagrams.

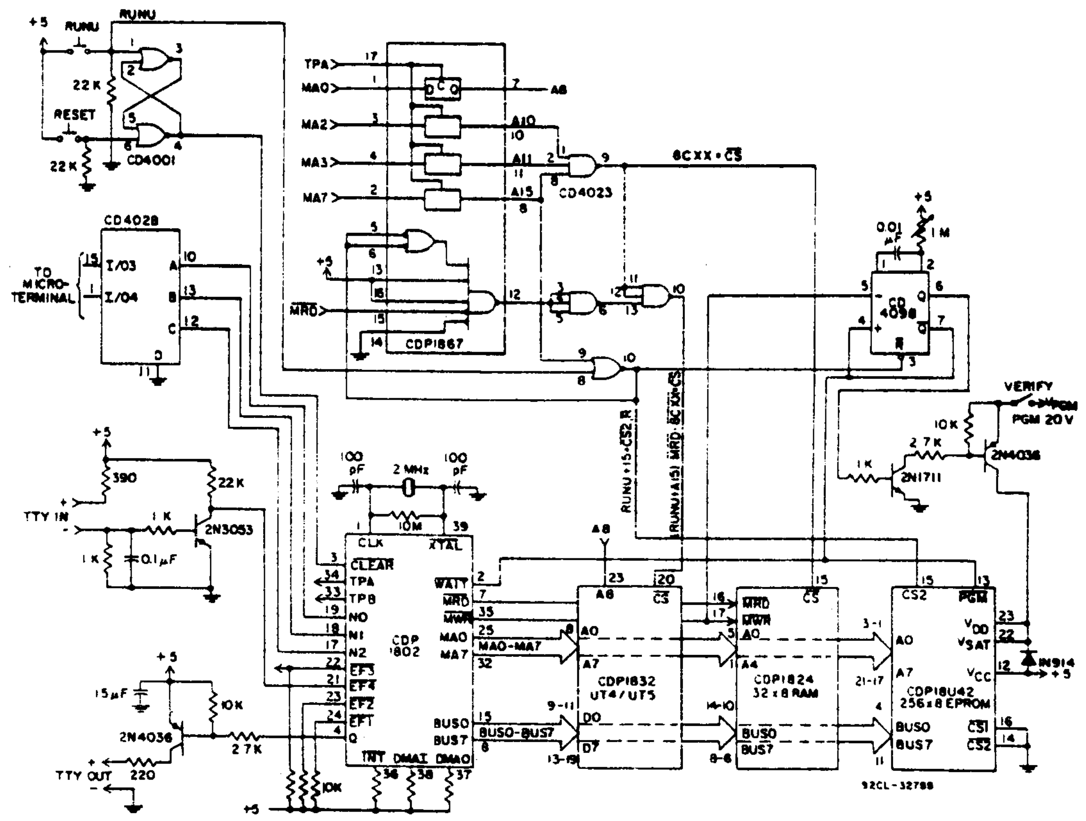
Operation is simplified by using the standard UT4 (CPR512) or UT5 (CPR522) utility program software. The CDP18U42 EPROM verify and program functions are implemented by utilizing the ?M and !M utility program functions. The CDP18U42 is treated as RAM UT4 and UT5.

The verify function (?M) is performed by locating the CDP18U42 below address space $8ØØØ_{16}$. Since the utility program is defined as starting at address $8ØØØ_{16}$, any read (?M) operation done below this address will read data from the CDP18U42 EPROM. The necessary address decoding is already in place on the RCA-EK and is handled by the CDP1867, CD4023, and CD4001 in the stand-alone design. When using the RCA EK, it is necessary to disable the first page of RAM.

The program function (!M) is performed by taking advantage of the WAIT-STATE feature of the CDP1802, to extend the CPU machine cycle to provide the 5 ms program time required by the CDP18U42 EPROM, when a write cycle (!M) is initiated below address $8ØØØ_{16}$, the $\overline{\text{MWR}}$ output from the CDP1802 triggers the CD4098 one-shot. The one-shot Q output puts the CPU in the WAIT-STATE and the CDP18U42 in the PROGRAM mode. Since the WAIT-STATE occurs late in the machine cycle, the address and data bits are held valid to provide the required CDP18U42 programming information. The one-shot Q output is used to enable the program voltage (20V) when the PGM/VERIFY switch is in the PGM position.

When this circuit is used with the RCA MICROTERMINAL, CDP18S021 (optional price $140), data is entered from the hex keypad and displayed on the LED display. The UT5 utility program used with the MICROTERMINAL, also contains a COUNT subroutine that increments the address once per second, which is useful as an automatic verify of PROM data.

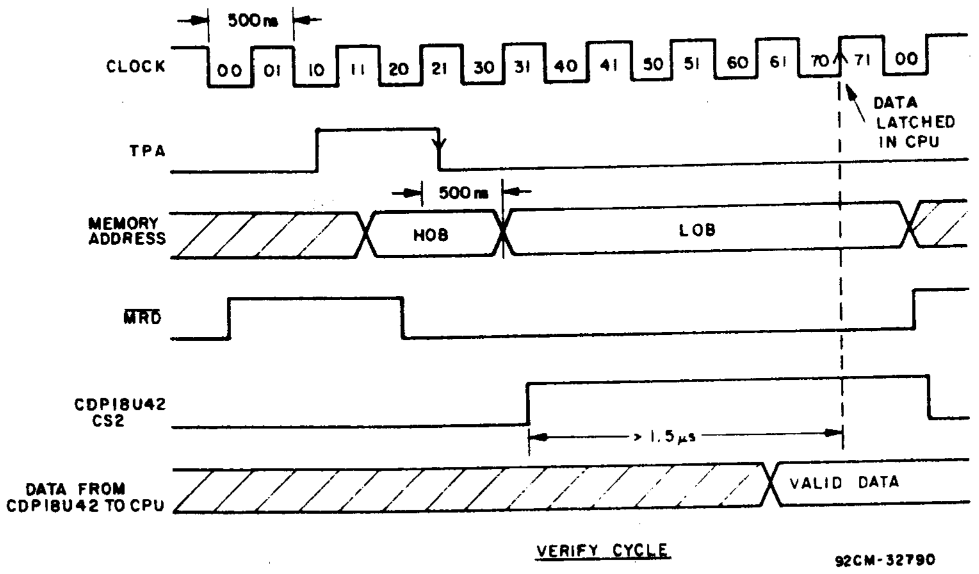
The circuit also provides a serial TTY (20 mA current loop) interface for use with a standard ASCII terminal. (UT4 is required for this option). This mode of operation provides keyboard data entry and hard-copy printout of EPROM data.



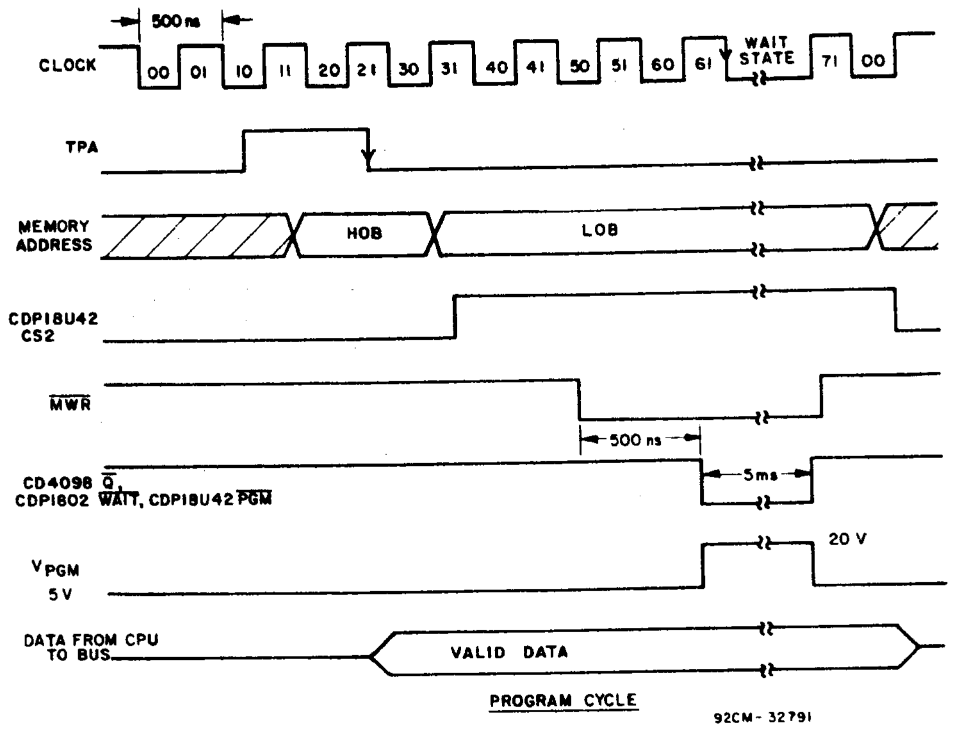
CDP1802-Based PROM Programmer Circuit

For more information, contact Rick Vaccarella, X6542.

CDP1802-Based PROM Programmer Circuit Timing



Verify Cycle



Program Cycle

LSI CDP1800 Series Data Sheets

TYPE	FILE #	LATEST ISSUE	SAME AS DATA BOOK?	COMMS
CDP1802	1023	6 -79	Y	
CDP1804	1147	8 -79	Y	
CDP1821	1200	8 -79	Y	
CDP1822	1074	12-80	N	REVS
MWS5101	1106	8 -80	N	REVS
CDP1823	1198	7 -79	Y	
CDP1824	1103	11-79	Y	
CDP1825	1247	6 -80	N	NEW
MWS5114	1207	12-80	N	REVS
CDP1831	1104	12-80	N	REVS
CDP1832	1145	11-80	N	REVS
CDP1833	1135	11-80	N	REVS
CDP1834	1143	12-80	N	REVS
TA10906	1267	11-80	N	NEW
TA10736	1276	12-80	N	NEW
CDP1842	1202	8 -79	-	DROP
CDP18U42	1219	11-79	N	NEW
CDP1851	1056	7 -79	Y	
CDP1852	1166	12-80	N	REVS
CDP1853	1189	6 -79	Y	
CDP1854A	1193	8 -79	Y	
CDP1855	1053	7 -79	Y	
CDP1856,7	1192	6 -79	Y	
CDP1858,9	1127	7 -79	Y	
CDP1861	1201	1 -80	N	REVS
CDP1862	1181	12-80	N	REVS
CDP1863	1179	5 -79	Y	
CDP1864	1196	7 -79	Y	
CDP1866,7,8	1112	6 -79	Y	
CDP1869,70,76	1197	3 -80	N	REVS
CDP1871	1232	1 -80	N	NEW
CDP1873	1248	6 -80	N	NEW
CD4036,9	613	10-72	Y	
CD4061A	768	3 -76	Y	
CD40061A	1188	12-76	Y	
CD40114B	1116	8 -78	Y	

RCA
Solid State
Division

Microprocessor Products
Application Note
ICAN-6622

Pro-Log PROM Programmer* Interface for the COSMAC Development System

by J. Kowalchik

This Note describes an interface and utility routine that allows operation of the COSMAC Development System with a state-of-the-art PROM programmer, the Series 90 with 9104 Parallel Interface Option, by Pro-Log Corporation. The interface requires a minimum of circuitry and is easily installed by the user.

FEATURES

The interface and utility routine:

1. Allows direct transfer of assembled object code from CDS to PROM
2. Allows direct transfers of PROM data into CDS memory
3. Compares PROM data to CDS memory and prints errors
4. Includes additional utility routines:
   a. fill memory with a constant,
   b. invert data in CDS memory.

The Series 90, a stand alone unit, can program, list, verify and duplicate. It is portable and can be configured for a wide variety of PROM types by installing the appropriate personality module.

Fig. 1 shows the block diagram of the system. The byte I/O board provided with the CDS is used to transfer data to and from the Series 90. An additional circuit board is required for control of the system. The board uses only four COSMOS integrated circuits and can be disabled when not in use. Fig. 2 shows the wiring diagram of the interface board and Fig. 3 the interconnect diagram of the system.

Software routines (PROG 1) are provided to support the interfa are loaded in higher order mem ing lower order memory location to be transferred to or from

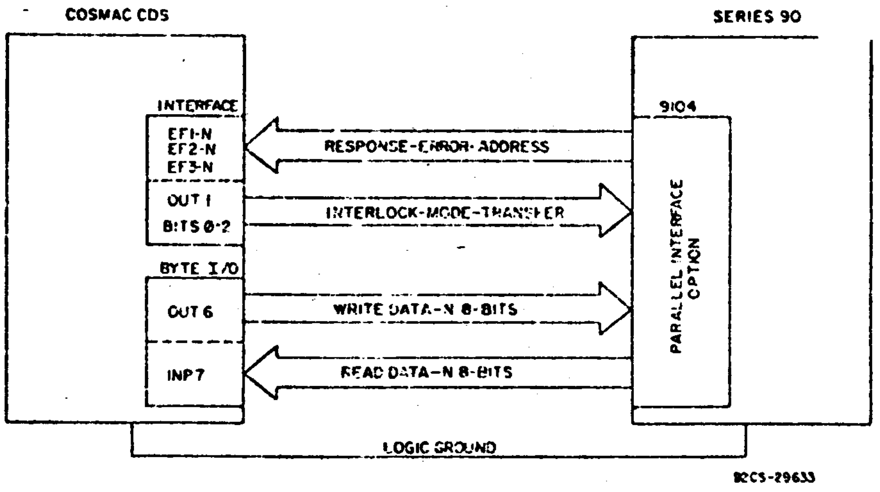


Fig. 1 — COSMAC-Series 90 block diagram.

* Pro-Log Corporation Series 90 with 9104 Parallel Interface Option.

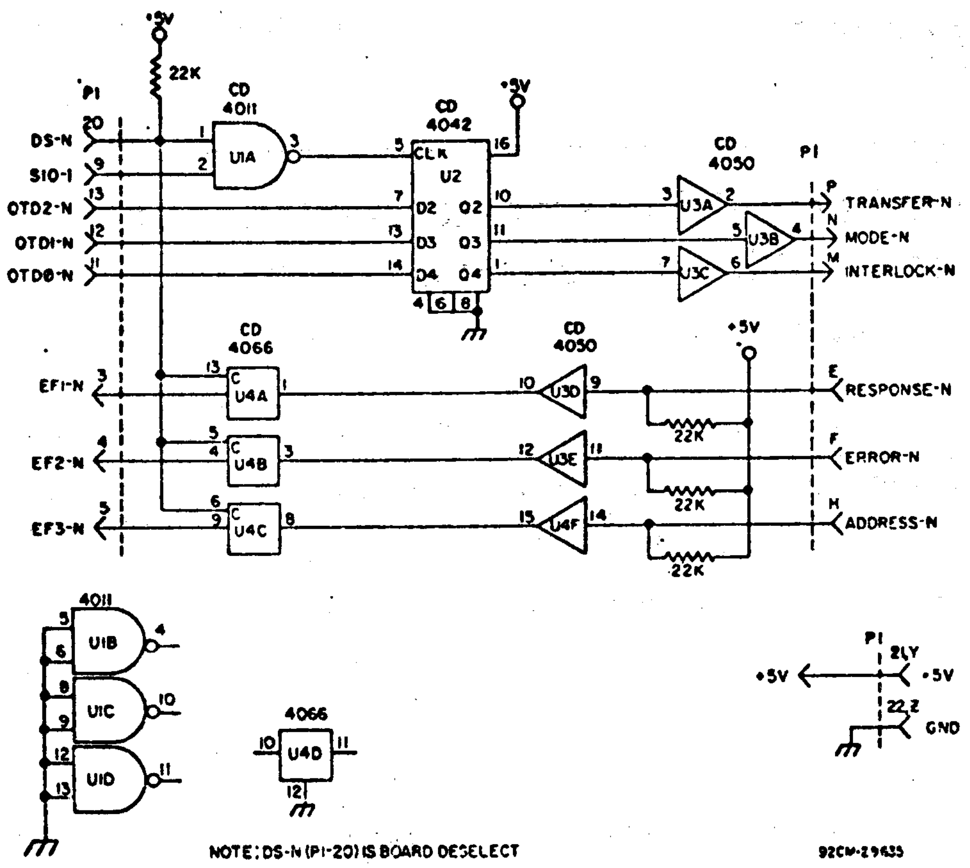


Fig. 2 – CDS-Series 90 interface board.

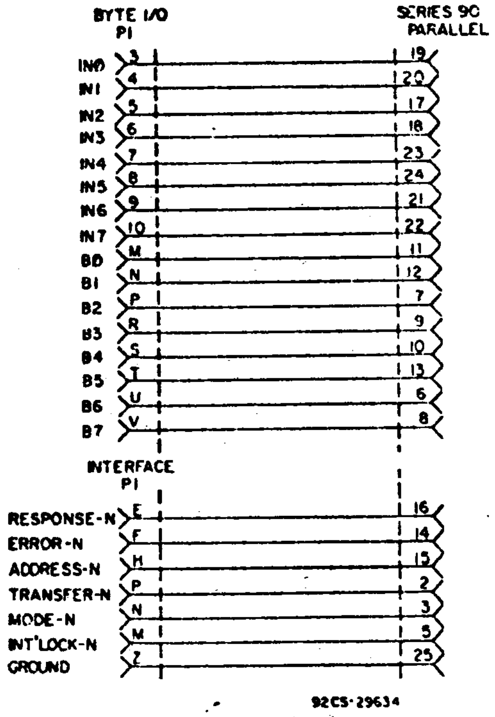


Fig. 3 – CDS-Series 90 interconnection.

PROG 1 requires only 512 bytes of memory and can be loaded into RAM or PROM, depending on the option of the user and the size of the CDS on hand. Some possible starting addresses are shown in Table I.

CONSTRUCTION

The interface circuitry is constructed on a standard-size plugboard that has been cut to match the COSMAC printed boards. The wiring diagram is shown in Fig. 2. Only the back plane signals indicated need to be wired to this board, which can be installed in any unused I/O slot. Refer to the RCA Application Note concerning handling of CMOS devices and installation precautions before beginning construction.

I/O PORT ASSIGNMENTS

PROG 1 uses output ports 1 and 6 and input port 7, as well as EF1, EF2, and EF3 to communicate with the Series 90. The port assignments are listed in Table II. Output port 6 and input port 7 are provided with the development system on the byte I/O board. Output port 1 and the external flag input circuitry are located on the user wired interface board, Fig. 1.

PROM Utility Program

PROG 1 was designed to provide the user with maximum flexibility in his application. The program recognizes the following commands:

1. PROGRAM PROM (P)

Format:	P (Starting Address), (number of bytes) cr (cr=carriage return)
Example:	P100,200cr Programs into PROM memory data from 100 to 2FF
Operation:	Data in CDS memory beginning at the starting address is programmed into PROM for the specified number of bytes starting at PROM address 0000. If 0000 is selected as the CDS starting address, a NOP instruction (C4) is loaded into PROM at 0000 because the CDS reserves memory address 0000 for the UT3 program workspace. The Series 90 tests for non-erased bits in the PROM to be programgrammed. An error message will be generated on the CDS terminal if a non-blank field is encountered. The "not erased" indicator on the Series 90 will also be lit in such a situation.

Table I - Some Possible Starting Addresses

Load Into	System Size	PROG 1 Starting Address	Workspace Available to User
RAM	4K	0C00	3K
RAM	8K	1C00	7K
PROM	4 or 8K	7000	4 or 8K

Table II - Port Assignments

OUT 6 (66)	IN 7 (67)	OUT 1 (61)
B0-P WRITE DATA 0-N	IN0-P READ DATA 0-N	B0-P INTERLOCK-N
B1-P WRITE DATA 1-N	IN1-P READ DATA 1-N	B1-P MODE-N
B2-P WRITE DATA 2-N	IN2-P READ DATA 2-N	B2-P TRANSFER-N
B3-P WRITE DATA 3-N	IN3-P READ DATA 3-N	
B4-P WRITE DATA 4-N	IN4-P READ DATA 4-N	
B5-P WRITE DATA 5-N	IN5-P READ DATA 5-N	
B6-P WRITE DATA 6-N	IN6-P READ DATA 6-N	
B7-P WRITE DATA 7-N	IN7-P READ DATA 7-N	

EXT. Flags
EF1-N RESPONSE-N
EF2-N ERROR-N
EF3-N ADDRESS-N

2. **TRANSFER PROM DATA TO CDS MEMORY (T)**
   Format: T (Starting Address), (number of bytes)cr
   Example: T100,200cr Transfers 200 hex bytes into memory starting at address 100.
   Operation: Data in PROM is loaded into CDS memory beginning at the starting address for the specified number of bytes. If 0000 is specified as the starting address, data will be transferred there, but will be modified by UT3 if an attempt is made to examine it.
3. **COMPARE PROM DATA WITH MEMORY DATA (C)**
   Format: C (Starting Address), (number of bytes)cr
   Example: C100,200cr Compares PROM data with that in CDS memory starting at 100 and continuing for 200 hex bytes.
   Operation: Data programmed in PROM is compared to that loaded into CDS memory beginning at the starting address and continuing for the given number of bytes. If no errors are found, the normal prompt character (.) will be printed. If any locations are found not to match the equivalent PROM location, an error message will be generated in the following format:

CDS M.A.	Memory Data	PROM DATA
1023	FF	69
01A0	23	F8

4. **FILL CDS MEMORY WITH A CONSTANT VALUE (F)**
   Format: F (Starting Address), (number of bytes), (data)cr
   Example: F100,100cr Fills memory 100-1FF with the hex data value.
   Operation: The data specified is loaded into CDS memory beginning at the starting address and continuing for the given number of bytes. The data can be any hex constant from 00 through FF. This routine is useful to "clear memory" areas prior to loading data for transfer into PROM. Any locations in the block to be burned into PROM will contain the value 00 or FF if they are not holding any object code.
5. **INVERT CDS MEMORY DATA (I)**
   Format: I (Starting Address), (number of bytes)cr
   Example: I100,200cr Inverts data in memory 100-2FF.
   Operation: Some systems require that data in PROM be negative true. This is the case for the CDS system itself. Positive true data can be inverted in memory in blocks using the I command before programming into PROM. Data beginning at the starting address and continuing for the given number of bytes is inverted.
6. **UTILITY (RETURN TO UT3) (U)**
   Format: Ucr
   Operation: Returns control to the UT3 program

PROG 1 Operation

PROG 1 is similar in operation to UT3. It is, however, a distinct program that can be stored on tape, disc or in PROM memory until needed. Like UT3, the only RAM required by PROG 1 is memory location 0000. It uses a period as its normal prompt character to distinguish it from UT3. Because many CDS resident subroutines are used by PROG 1, only the last four digits entered are valid, and leading zeros are assumed on all address and number of bytes entries. The last two digits typed are valid on data entries, also with leading zeros assumed. The comma is used by PROG 1 rather than the space of UT3 to add clarity on multiple entry commands.

TYPICAL USAGE OF THE SYSTEM

Assume that a program has just been assembled and that its object file resides on disc. The program occupies slightly less than 1K of memory and is to be loaded into two 2704 PROMS. The origin of the program is 0000, and its current disc location is U/TR # 0100. A typical sequence of operations is as follows:

1. Load the PROM Utility Routine into CDS memory (assume a 4K CDS).
2. Start the PROM Utility package running by typing $P0C00 cr. The program will respond with its characteristic prompt, period (.), which distinguishes it from UT3.
3. Load the object code into CDS memory using the following:

*$P8600 cr	Start the disc loader.
READ?0100 cr	Select U/TR# of the object file
$P0C00 cr	Go back to the PROM Utility Period indicated PROM Utility running

4. Insert a blank 2704 into the "copy" socket.
5. Program the data from 000 to 1FF by typing P0,200cr. It may be necessary to press "reset" on the Series 90 if this is the first operation.
6. Confirm that the data is correct by typing C0,200cr. Location 000 in CDS memory was the starting address, so the data in PROM location 0000 is C4 as noted under the "P" command operation. The compare routine will not detect this as an error. Since memory location 0000 is used as scratchpad memory by PROG 1 and UT3, errors are not reported. Therefore, differences are likely to occur between memory and PROM at this location. This is a normal condition only for location 0000. All other locations should compare and the PROM Utility will respond with a period.
7. Program the second PROM with data from 200 to 3FF by typing: P200,200cr.
8. Verify that the data is correct by typing: C200,200cr.
   NOTE: If the PROM to be programmed required that the data be negative true, the I command would be employed to invert the entire block of data in CDS memory before programming.

SPECIAL CONSIDERATIONS

1. Some PROM types require special programming techniques as specified by the manufacturer. These special considerations may affect the operation of the CDS interface; refer to the Series 90 operating manual or contact Pro-Log Corporation. The interface described in Note was operated using 1702A and 2704/2708 parts, and functioned successfully.
2. The problem of location 0000 in CDS memory being unavailable to the user could be critical where the Disable instruction is to be programmed at location 0000 in PROM. The solution is to load the object code into CDS starting at 0000. The UT3 program will clobber location 0000. After the chip has been programmed, disable the interface and manually copy the burned PROM into another blank PROM after changing location 0000 to the disable op code.
3. The programmer must be sure that the Series 90 unit is wired for the 9104 Parallel Interface. The presence of a connector on the unit does not assure that it is wired with this option. For those units not so equipped, the option is field installable (contact Pro-Log Corp. for additional information).

Reference

1. "Guide to Better Handling and Operation of CMOS Integrated Circuits," J. Flood and H.L. Pujol, RCA Solid State Application Note ICAN-6525.

PROG 1

```
!M
0000 ;            ..FROM UTILITY ROUTINE PACKAGE
0000 ;            ..WRITTEN FOR CDP1802 1/77
0000 ;            ..
0000 ;            ..MAIN PC= R5, USES R3 FOR
0000 ;            ..LINKAGE TO UT3, R(0).1
0000 ;            ..HOLDS BRANCH INFO
0000 ;            ..
0000 ;            ..
0000 ;            ..CONFIGURED FOR 8K SYSTEM
0000 ;            ..
0000 ;            ORG #1C00
1C00 F81C;        LDI    A.1(BEGIN)
1C02 B5;          PHI    R5
1C03 F807;        LDI    A.0(BEGIN)
1C05 A5;          PLO    R5
1C06 D5;          SEP    R5
1C07 F881; BEGIN: LDI    #81
1C09 B3;          PHI    R3
1C0A F89C; ENTRY: LDI    #9C
1C0C A3;          PLO    R3
1C0D D30D;        SEP    R3,#0D   ..RETURN
1C0F D30A;        SEP    R3,#0A   ..LINE FEED
1C11 D32E;        SEP    R3,#2E   ..PERIOD
1C13 F83B;        LDI    #3B
1C15 A3;          PLO    R3
1C16 D3;          SEP    R3       ..READ A CHARACTER
1C17 FB46;        XRI    #46      ..IS IT AN F?
1C19 3A20;        BNZ    *+7
1C1B F800;        LDI    00
1C1D B0;          PHI    R0
1C1E 3051;        BR     FILL
1C20 9F;          GHI    RF
1C21 FB49;        XRI    #49      ..IS IT AN I?
1C23 3A2A;        BNZ    *+7
1C25 F801;        LDI    01
1C27 B0;          PHI    R0
1C28 3051;        BR     FILL
1C2A 9F;          GHI    RF
1C2B FB55;        XRI    #55
1C2D C21DF3;      LBZ    UTIL     ..IS IT A U?
1C30 9F;          GHI    RF       ..IS IT A P?
1C31 FB50;        XRI    #50      ..YES, GO TO PROG
1C33 3A3A;        BNZ    *+7
1C35 F800;        LDI    00
1C37 B0;          PHI    R0
1C38 30A8;        BR     PROG
1C3A 9F;          GHI    RF       ..IS IT AN L?
1C3B FB54;        XRI    #54
1C3D 3A44;        BNZ    *+7
1C3F F801;        LDI    01
1C41 B0;          PHI    R0
1C42 30A8;        BR     PROG
1C44 9F;          GHI    RF
1C45 FB43;        XRI    #43      ..IS IT A C?
```

```
1C47 3A4E;      0055            BNZ     *+7
1C49 F802;      0056            LDI     02
1C4B B0;        0057            PHI     R0
1C4C 30A8;      0058            BR      PROG
1C4E C01DEB;    0059            LBR     SINERR
1C51 ;          0060 .......................................
1C51 F800;      0061 FILL:      LDI     00
1C53 AD;        0062            PLO     RD
1C54 BD;        0063            PHI     RD
1C55 F83B;      0064            LDI     #3B
1C57 A3;        0065            PLO     R3
1C58 D3;        0066 LOOP1:     SEP     R3
1C59 3358;      0067            BDF     LOOP1
1C5B FB2C;      0068            XRI     #2C
1C5D CA1DE4;    0069            LBNZ    SYNERR
1C60 9D;        0070            GHI     RD
1C61 B7;        0071            PHI     R7
1C62 8D;        0072            GLO     RD
1C63 A7;        0073            PLO     R7
1C64 F800;      0074            LDI     00
1C66 BD;        0075            PHI     RD
1C67 AD;        0076            PLO     RD
1C68 D3;        0077 LOOP2:     SEP     R3
1C69 3368;      0078            BDF     LOOP2
1C6B 90;        0079            GHI     R0
1C6C 3A73;      0080            BNZ     BYPAS
1C6E 9F;        0081            GHI     RF
1C6F FB2C;      0082            XRI     #2C
1C71 3076;      0083            BR      BP1
1C73 9F;        0084 BYPAS:     GHI     RF
1C74 FB0D;      0085            XRI     #0D
1C76 CA1DE4;    0086 BP1:       LBNZ    SYNERR
1C79 8D;        0087            GLO     RD
1C7A 3A80;      0088            BNZ     BP2
1C7C 9D;        0089            GHI     RD
1C7D C21DE4;    0090            LBZ     SYNERR
1C80 9D;        0091 BP2:       GHI     RD
1C81 B8;        0092            PHI     R8
1C82 8D;        0093            GLO     RD
1C83 A8;        0094            PLO     R8
1C84 90;        0095            GHI     R0
1C85 3A93;      0096            BNZ     LOOP4
1C87 F800;      0097            LDI     00
1C89 AD;        0098            PLO     RD
1C8A BD;        0099            PHI     RD
1C8B D3;        0100 LOOP3:     SEP     R3
1C8C 338B;      0101            BDF     LOOP3
1C8E FB0D;      0102            XRI     #0D          ..IS IT A RETURN?
1C90 CA1DE4;    0103            LBNZ    SYNERR
1C93 28;        0104 LOOP4:     DEC     R8
1C94 90;        0105            GHI     R0
1C95 3A9B;      0106            BNZ     INVERT
1C97 8D;        0107 FILLM:     GLO     RD
1C98 57;        0108            STR     R7
1C99 309F;      0109            BR      FILINV
1C9B 07;        0110 INVERT:    LDN     R7
1C9C FBFF;      0111            XRI     #FF
```

```
1C9E 57;              STR    R7
1C9F 17;     FILINV:  INC    R7
1CA0 88;              GLO    R8
1CA1 3A93;            BNZ    LOOP4
1CA3 98;              GHI    R8
1CA4 3A93;            BNZ    LOOP4
1CA6 300A;            BR     ENTRY
1CA8 ;       ..........................................
1CA8 ;                ..PROM PROGRAM UTILITY ROUTINES
1CA8 ;                ..P=R5, R7 HOLDS MEM. STARTING ADDR
1CA8 ;                ..R1= PC COMP SUB, R(0).1 = BRANCH
1CA8 ;                ..RD HOLDS # OF BYTES, R6= PC DELAY
1CA8 ;                ..RA= PC TRANSF, R9= PC INPREP
1CA8 ;                ..X= R5 FOR I/O AND RB OTHERWISE
1CA8 ;                ..
1CA8 ;                ..FOR PRO-LOG SERIES 90 MACHINE
1CA8 ;                ..
1CA8 ;       LIST=#08
1CA8 ;       IPREP=#09
1CA8 ;       DLY=#06
1CA8 ;       TRAN=#0A
1CA8 ;       ..
1CA8 F81D;   PROG:    LDI    A.1(INPREP)
1CAA B9;              PHI    IPREP
1CAB B1;              PHI    R1
1CAC B6;              PHI    DLY
1CAD BA;              PHI    TRAN
1CAE B8;              PHI    LIST
1CAF F8C8;            LDI    A.0(INPREP)
1CB1 A9;              PLO    IPREP
1CB2 F89A;            LDI    A.0(COMP)
1CB4 A1;              PLO    R1
1CB5 F8BB;            LDI    A.0(DELAY)
1CB7 A6;              PLO    DLY
1CB8 F8AF;            LDI    A.0(TRANSF)
1CBA AA;              PLO    TRAN
1CBB F8D2;            LDI    A.0(LISTD)
1CBD A8;              PLO    LIST
1CBE D9;              SEP    IPREP
1CBF D3;     PROG1:   SEP    R3        ..CALL READ
1CC0 33BF;            BDF    PROG1
1CC2 FB2C;            XRI    #2C
1CC4 CA1DE4;          LBNZ   SYNERR
1CC7 9D;              GHI    RD
1CC8 B7;              PHI    R7
1CC9 8D;              GLO    RD
1CCA A7;              PLO    R7
1CCB D9;              SEP    IPREP     ..CALL IPREP
1CCC D3;     PROG2:   SEP    R3
1CCD 33CC;            BDF    PROG2
1CCF FB0D;            XRI    #0D       ..IS IT A CR?
1CD1 CA1DE4;          LBNZ   SYNERR
1CD4 E5;              SEX    R5
1CD5 6100;            OUT    1,00      ..RESET
1CD7 D610;            SEP    R6,#10    ..CALL DELAY
1CD9 6101;            OUT    1,#01     ..GAIN CONTROL OF 90
1CDB 3EDB;            BN3    *         ..WAIT TILL ADDR LO
```

```
1CDD 6103;              OUT    1,#03     ..SET MODE LO
1CDF F800;              LDI    #00
1CE1 BB;                PHI    RB
1CE2 AB;                PLO    RB
1CE3 66FF;              OUT    6,#FF     ..OUTPUT 00
1CE5 DA;                SEP    RA
1CE6 DA;                SEP    RA
1CE7 DA;                SEP    RA        ..CALL TRANSF 3 TIMES AAA
1CE8 EB;                SEX    RB
1CE9 9D;                GHI    RD        ..GET MSB
1CEA FA0F;              ANI    #0F
1CEC FBFF;              XRI    #FF       ..INVERT
1CEE 5B;                STR    RB
1CEF 66;                OUT    6
1CF0 2B;                DEC    RB
1CF1 DA;                SEP    RA        ..CALL TRANSF
1CF2 EB;                SEX    RB
1CF3 8D;                GLO    RD
1CF4 F6;                SHR
1CF5 F6;                SHR
1CF6 F6;                SHR
1CF7 F6;                SHR
1CF8 FBFF;              XRI    #FF       ..INVERT
1CFA 5B;                STR    RB
1CFB 66;                OUT    6
1CFC 2B;                DEC    RB
1CFD DA;                SEP    RA
1CFE EB;                SEX    RB
1CFF 8D;                GLO    RD
1D00 FA0F;              ANI    #0F
1D02 FBFF;              XRI    #FF
1D04 5B;                STR    RB
1D05 66;                OUT    6
1D06 2B;                DEC    RB
1D07 DA;                SEP    RA        ..CALL TRANSF ADDR DONE
1D08 ;          ..........................................................
1D08 90;                GHI    R0        ..IF 0 DO PROGRAM, IF
1D09 CA1D49;            LBNZ   LOAD      ..NOT 0 DO COMP OR LIST
1D0C EB;                SEX    RB
1D0D 340D;              B1     *         ..WAIT RESP HI
1D0F 360F;              B3     *         ..CK ADDR HI
1D11 3515;              B2     *+4       ..CK NON BL FLD ERR
1D13 3017;              BR     FIX
1D15 303B;              BR     ERROR
1D17 ;          ..........................................................
1D17 97;          FIX:  GHI    R7
1D18 3A29;              BNZ    AHEAD     ..FIX UP LOC 0000
1D1A 87;                GLO    R7
1D1B 3A29;              BNZ    AHEAD     ..PUT C4 THERE IF NEEDED
1D1D 2D;                DEC    RD
1D1E F8C4;              LDI    #C4
1D20 FBFF;              XRI    #FF
1D22 5B;                STR    RB
1D23 66;                OUT    6
1D24 2B;                DEC    RB
1D25 DA;                SEP    RA        ..CALL TRANSF
1D26 17;                INC    R7
```

```
1D27 3032;    0226           BR     TEST
1D29 2D;      0227 AHEAD:    DEC    RD
1D2A 47;      0228           LDA    R7
1D2B FBFF;    0229           XRI    #FF
1D2D 5B;      0230           STR    RB
1D2E EB;      0231           SEX    RB       ..FIX   X
1D2F 66;      0232           OUT    6
1D30 2B;      0233           DEC    RB
1D31 DA;      0234           SEP    RA       ..TRANSF
1D32 9D;      0235 TEST:     GHI    RD
1D33 3A29;    0236           BNZ    AHEAD
1D35 8D;      0237           GLO    RD
1D36 3A29;    0238           BNZ    AHEAD
1D38 C01C0A;  0239           LBR    ENTRY    ..PRINT CR,LF,PD
1D3B ;        0240 ...................................
1D3B F89C;    0241 ERROR:    LDI    #9C
1D3D A3;      0242           PLO    R3
1D3E D30A;    0243           SEP    R3,#0A   ..LINE FEED
1D40 D345;    0244           SEP    R3,#45
1D42 D352;    0245           SEP    R3,#52
1D44 D352;    0246           SEP    R3,#52
1D46 C01C07;  0247           LBR    BEGIN
1D49 ;        0248 ......................................
1D49 3449;    0249 LOAD:     B1     LOAD     ..WAIT RESP HI
1D4B 364B;    0250           B3     *        ..WAIT ADDR HI
1D4D E5;      0251           SEX    R5       ..FIX X AFTER TRANSF
1D4E 6101;    0252           OUT    1,#01    ..SET MODE HI
1D50 90;      0253           GHI    R0       ..IF 1 DO LIST, ELSE
1D51 FF01;    0254           SMI    01       ..DO COMPARE
1D53 3A60;    0255           BNZ    COMPAR
1D55 ;        0256 ......................
1D55 2D;      0257 LD1:      DEC    RD
1D56 D8;      0258           SEP    LIST
1D57 8D;      0259           GLO    RD
1D58 3A55;    0260           BNZ    LD1
1D5A 9D;      0261           GHI    RD
1D5B 3A55;    0262           BNZ    LD1
1D5D C01C0A;  0263           LBR    ENTRY    ..PRINT CR,LF,PD
1D60 2D;      0264 COMPAR:   DEC    RD
1D61 D1;      0265           SEP    R1       ..CALL COMP
1D62 3A6D;    0266           BNZ    PRINT
1D64 8D;      0267 AGAIN:    GLO    RD
1D65 3A60;    0268           BNZ    COMPAR
1D67 9D;      0269           GHI    RD
1D68 3A60;    0270           BNZ    COMPAR
1D6A C01C0A;  0271           LBR    ENTRY
1D6D ;        0272 ..
1D6D F89C;    0273 PRINT:    LDI    #9C      ..PC=R5
1D6F A3;      0274           PLO    R3
1D70 D30A;    0275           SEP    R3,#0A   ..L. F.
1D72 27;      0276           DEC    R7       ..FIX R7 AFTER COMP
1D73 97;      0277           GHI    R7
1D74 BF;      0278           PHI    RF
1D75 F8AE;    0279           LDI    #AE
1D77 A3;      0280           PLO    R3
1D78 D3;      0281           SEP    R3
1D79 87;      0282           GLO    R7
```

```
1D7A BF;    0283           PHI      RF
1D7B F8AE;  0284           LDI      #AE
1D7D A3;    0285           PLO      R3
1D7E D3;    0286           SEP      R3
1D7F D320;  0287           SEP      R3,#20
1D81 D320;  0288           SEP      R3,#20   ..SPACE TWICE
1D83 E7;    0289           SEX      R7
1D84 F0;    0290           LDX
1D85 BF;    0291           PHI      RF
1D86 17;    0292           INC      R7       ..RESTORE R7
1D87 F8AE;  0293           LDI      #AE
1D89 A3;    0294           PLO      R3
1D8A D3;    0295           SEP      R3
1D8B D320;  0296           SEP      R3,#20
1D8D D320;  0297           SEP      R3,#20   ..SPACE TWICE
1D8F 0B;    0298           LDN      RB
1D90 BF;    0299           PHI      RF
1D91 F8AE;  0300           LDI      #AE
1D93 A3;    0301           PLO      R3
1D94 D3;    0302           SEP      R3
1D95 D30D;  0303           SEP      R3,#0D   ..C. R.
1D97 3064;  0304           BR       AGAIN
1D99 ;      0305 .........................................
1D99 ;      0306           ..PROM UTILITY SUBROUTINES
1D99 ;      0307           ..
1D99 D5;    0308 EXCOMP:   SEP      R5
1D9A 349A;  0309 COMP:     B1       COMP     ..COMP SUB PC=R1
1D9C E1;    0310           SEX      R1       ..SET X=P
1D9D 6105;  0311           OUT      1,#05
1D9F 3C9F;  0312           BN1      *        ..WAIT RESP LO
1DA1 EB;    0313           SEX      RB
1DA2 6F;    0314           INP      7        ..GET CHAR FROM PROM
1DA3 FBFF;  0315           XRI      #FF      ..INVERT THE DATA
1DA5 5B;    0316           STR      RB
1DA6 E7;    0317           SEX      R7       ..POINT TO MEM
1DA7 F7;    0318           SM                ..CHECK IF =
1DA8 17;    0319           INC      R7
1DA9 E1;    0320           SEX      R1
1DAA 6101;  0321           OUT      1,#01
1DAC 3099;  0322           BR       EXCOMP
1DAE D5;    0323 EXTRFR:   SEP      R5
1DAF 34AF;  0324 TRANSF:   B1       TRANSF   ..WAIT RESP HI
1DB1 EA;    0325           SEX      RA
1DB2 6107;  0326           OUT      1,#07    ..SET TRANS LO
1DB4 3CB4;  0327           BN1      *        ..WAIT RESP LO
1DB6 6103;  0328           OUT      1,#03    ..RESET TRANS
1DB8 30AE;  0329           BR       EXTRFR
1DBA ;      0330 ..
1DBA D5;    0331 EDLY:     SEP      R5
1DBB 45;    0332 DELAY:    LDA      R5
1DBC BB;    0333           PHI      RB
1DBD AB;    0334           PLO      RB
1DBE 2B;    0335 DLY1:     DEC      RB
1DBF 8B;    0336           GLO      RB
1DC0 3ABE;  0337           BNZ      DLY1
1DC2 9B;    0338           GHI      RB
1DC3 3ABE;  0339           BNZ      DLY1
```

```
1DC5 30BA;   0340           BR     EDLY
1DC7 ;       0341 ..
1DC7 D5;     0342 EXIN:     SEP    R5
1DC8 F800;   0343 INPREP:   LDI    00
1DCA BD;     0344           PHI    RD
1DCB AD;     0345           PLO    RD
1DCC F83B;   0346           LDI    #3B
1DCE A3;     0347           PLO    R3
1DCF 30C7;   0348           BR     EXIN
1DD1 ;       0349 ..
1DD1 D5;     0350 EXLIST:   SEP    R5
1DD2 34D2;   0351 LISTD:    B1     LISTD     ..WAIT RESP HI
1DD4 E8;     0352           SEX    R8        ..FIX X
1DD5 6105;   0353           OUT    1,#05     ..SET TRANS LO
1DD7 3CD7;   0354           BN1    *         ..WAIT RESP LO
1DD9 E7;     0355           SEX    R7
1DDA 6F;     0356           INP    7
1DDB FBFF;   0357           XRI    #FF       ..INVERT INPUT DATA
1DDD 57;     0358           STR    R7        ..STORE IN MEM.
1DDE 17;     0359           INC    R7
1DDF E8;     0360           SEX    R8
1DE0 6101;   0361           OUT    1,#01
1DE2 30D1;   0362           BR     EXLIST
1DE4 F89C;   0363 SYNERR:   LDI    #9C
1DE6 A3;     0364           PLO    R3
1DE7 D30D;   0365           SEP    R3,#0D    ..C.R.
1DE9 D30A;   0366           SEP    R3,#0A    ..L.F.
1DEB F89C;   0367 SINERR:   LDI    #9C
1DED A3;     0368           PLO    R3
1DEE D33F;   0369           SEP    R3,#3F    ..?
1DF0 C01C07; 0370           LBR    BEGIN
1DF3 D3;     0371 UTIL:     SEP    R3
1DF4 9F;     0372           GHI    RF
1DF5 FB0D;   0373           XRI    #0D
1DF7 3AE4;   0374           BNZ    SYNERR
1DF9 00;     0375           IDL              ..CAUSE INTERRUPT TO
1DFA ;       0376                            ..RETURN CONTROL TO UT3
1DFA ;       0377           END
0000
```

NOTE:

PROG 1 as written was designed for use with personality modules requiring a 3-digit address field, such as the 2704/2708. To use PROG 1 with modules requiring a two digit address field, such as the 1702A, lines 174 through 203, should be deleted and replaced with the following lines.

```
1CE5 DA:     0174           SEP  RA
1CE6 DA:     0175           SEP  RA       ..CALL TRANSF 2 TIMES
1CE7 EB:     0176           SEX  RB
1CE8 9D:     0177           GHI  RD       ..GET MSB
1CE9 A0:     0178           PLO  R0       ..SAVE IN R0.0
1CEA 32ED:   0179           BZ   GOAHED   ..IF MSB=0 GO ON
1CEC 2D:     0180           DEC  RD       ..IF MSB=1 DEC
1CED 8D:     0181  GOAHED:  GLO  RD
1CEE F6:     0182           SHR
1CEF F6:     0183           SHR
1CF0 F6:     0184           SHR
1CF1 F6:     0185           SHR
1CF2 FBFF:   0186           XRI  #FF
1CF4 5B:     0187           STR  RB
1CF5 66:     0188           OUT  6
1CF6 2B:     0189           DEC  RB
1CF7 DA:     0190           SEP  RA       ..CALL TRANSF
1CF8 EB:     0191           SEX  RB       ..FIX X
1CF9 8D:     0192           GLO  RD
1CFA FA0F:   0193           ANI  #0F
1CFC FBFF:   0194           XRI  #FF
1CFE 5B:     0195           STR  RB
1CFF 66:     0196           OUT  6
1D00 2B:     0197           DEC  RB
1D01 DA:     0198           SEP  RA       ..CALL TRANSF
1D02 80:     0199           GLO  R0
1D03 3207:   0200           BZ   LABEL
1D05 1D:     0201           INC  RD
1D06 C4:     0202           NOP
1D07 C4:     0203  LABEL:   NOP
```

-193-

RCA
Solid State
Division

Microprocessor Products
Application Note
ICAN-6928

Interfacing PLM Code to CDOS System Functions

by W. Fritchie

This Application Note defines a method for interfacing PLM programs to CDOS[1] system functions without the need for assembly language; the interface is an array of PLM procedures (which can be included in a PLM library) and supportive macro definitions, all of which are described in detail and used in a sample program.

Background

The system functions in CDOS are described fully in the CDOS manual.[2] The description shows the programmer how to interface his applications to the system functions in assembly language. Since PLM allows assembly language statements to be mixed with PLM statements, by using the $A characters, the programmer can interface his PLM applications with the CDOS system functions in this way. However, it is more desirable to have the entire application written in PLM, as this approach leads to shorter development time and prevents the user from having to learn the mechanics of applying the system functions in assembly language. For example, if the user wanted to type a message to the console, the PLM statement would be:

```
CALL TYPE, (MESSAGE);
```

where MESSAGE is an address variable containing the pointer to the message to be typed. The equivalent code in assembly language presents some problems. As an example, it would be very difficult to determine the address of the variable message that would have to be passed to the CDOS system function that types the characters associated with the variable MESSAGE. A solution would be to declare the variable MESSAGE in assembly language and not in a PLM statement; however, this solution would have to be accomplished with care so that the message would not be positioned in the mainstream of instructions generated by the PLM compiler.

Another drawback to mixing assembly language with PLM statements is that the resulting overall program is less readable and, therefore, more difficult to maintain. PLM is a block-oriented language in which the block is emphasized on the listing by indentation. Example 1, Fig. 1, the block format of a program written in PLM, shows how indentation is used to group the statements of a block. This example is very simple; in more complex examples, the block structure becomes much more important in understanding and maintaining the program. In Example 1, it is assumed that a procedure exists called TYPE which is contained in a library of commonly used procedures. It is easy to see where the TYPE statement appears in the overall flow of the program. If the TYPE statement were not in the correct logical sequence, it could easily be moved into the correct block.

Example 2, Fig. 2, shows assembly-language instructions mixed with PLM instructions to accomplish the same results as the program in Example 1. Example 2 also illustrates some of the problems incurred when linking CDOS system functions and PLM programs with assembly language. First, the programmer

```
DO;
DECLARE DATA (5) BYTE INITIAL ('PRINT');
DECLARE (X,Y,Z) BYTE;
DECLARE MESSAGE ADDRESS;
MESSAGE = ADDR(DATA);
X=Y;
IF X=Z
    THEN
      DO;
          Y=3;
          X=1;
    END;
  ELSE
      DO;
        X=1;
        CALL TYPE (MESSAGE);
    END;
END EXAMPLE 1;
```

Fig. 1 - Example 1.

must understand how to set up the values of UCALL and TYPE. Second, he must learn where to position the assembly-language statement for MESSAGE so as not to cause any run-time problem. Example 2, like Example 1, is straightforward; it is relatively easy to read and maintain the code. However, again, when the application becomes more complex, the readability and maintainability of the program, when assembly language code is mixed with PLM code, becomes much more of a problem. The PLM procedures and the macros described below will permit the programmer to interface his PLM program with CDOS system functions without having to use assembly language.

```
    DO;
    DECLARE (X, Y, Z) BYTE:
      X=Y;
      IF X=Z
        THEN
          DO;
            Y=3;
            X=1;
          END;
        ELSE
          DO;
            X=1;
  $AUCALL=#B453
  $ATYPE=14
  $ACALL UCALL, TYPE, A (MESSAG)
          END;
  $AMESSAG:   ,T'PRINT'
  END EXAMPLE 2;
```

Fig. 2 - Example 2.

Supportive Macros

The following interfaces between PLM statements and the CDOS system functions are discussed in detail:

- Typing a message to the console - TYPE
- Accepting input from the keyboard - CREAD
- Typing out a CDOS error message - CDERR
- Parsing a file name from an input stream into an IOCB - SRNAM
- Returning to CDOS - CDENT
- Setting up the X and P registers
- Setting up the entire PLM library as a macro

In order to perform these interfaces, the macro definitions described below and in Fig. 3 are required.

ARG and PARM1: Macro definitions ARG and PARM1 deal with passing the addresses of parameters to the assembly language calls for CDOS. For example, Example 1 declared the variable MESSAGE in a declaration statement, but CDOS must know the memory address of the variable MESSAGE. The macro definitions ARG and PARM1 assist in this linkage.

CDOS: The macro definition CDOS returns control to the operating system when the application program has finished its task.

ENTRY: The macro definition ENTRY is used to set up the X and P registers. This code generates a label called START. When the output from PLM has been assembled, the address of START must be ascertained from the listing; this address is used as the starting address when making a binary file from the ASCII-HEX file (the conversion of ASCII-HEX files to binary is done by the CDOS command CDSBIN).

EQUATE: The macro definition EQUATE contains values for the constants used in the assembly-language statements in the Code of Procedure sections described under the heading "PLM Procedures," below.

PLMMAC: The macro definition PLMMAC contains the standard PLM library in a macro-definition form. The advantage of keeping the PLM library as a macro definition rather than as a CDOS file is that the former is easier and faster to use, primarily because the MERGE command does not have to be used to combine the output of the PLM compiler with the PLM library. When the PLMMAC macro definition is used, the library is automatically combined with the output of the PLM compiler during the assembly process to produce the desired object code. This method is faster because the step involving the combination of the PLM output with the PLM library is eliminated. In addition, the problems associated with merging, such as disk-full conditions, are also eliminated.

PLM Procedures

The PLM procedures associated with the supportive macros described above are specified in the following format:

- Name
- Function
- Assumptions
- Call to Procedure
- Code of Procedure

Name: TYPE
Function: Types out a message to the console.
Assumptions: MSG$ADR is an address variable.
MESSAGE contains the message to type.
Call to Procedure:

```
MSG$ADR=ADDR (MESSAGE);
CALL TYPE (MSG$ADR);
```

Code of Procedure:

```
TYPE: PROCEDURE (TYPE$PARM);
  DECLARE (TYPE$PARM,X) ADDRESS;
  X=TYPE$PARM;
  $AARG 'TYPEAD'..SETTING UP CDOS
     TYPE PARAMETER
  $ACALL UCALL,ZTYPE..CALL TO
     CDOS
  $ATYPEAD:  ORG *+2..STORAGE FOR
     MESSAGE ADR.
END TYPE;
```

Name: KEYBD
Function: Reads a record from the console into a buffer.
Assumptions: BUFFER is declared as the input buffer. BUF$LENGTH is declared as a byte variable.
BUF$ADR is an address variable.
Call to Procedure:
```
BUF$ADR=ADDR (BUFFER);
CALL KEYBD (BUF$ADR,
   BUF$LENGTH);
```
Code of Procedure:
```
KEYBD:  PROCEDURE
   KEYBUF,LGT);
 DECLARE (KEYBUF,X) ADDRESS;
 DECLARE (Y,LGT) BYTE;
 X=KEYBUF;
 $AARG 'KEYADR'
 Y=LGT;
 $APARM1 'LENGTH'
 $ACALL UCALL,KEYIN
 $AKEYADR:  ORG *+2
 $ALENGTH:  ORG *+1
END KEYBD;
```

Name: PRINT$ERROR
Function: Print out a CDOS error message.
Assumptions: BUFFER is an array with a minimum of 2 bytes; the error message number is in the second byte.
MSG$ADR is an address variable.
Call to Procedure:
```
MSG$ADR=ADDR (BUFFER);
CALL PRINT$ERROR (MSG$ADR);
```
Code of Procedure:
```
PRINT$ERROR:  PROCEDURE
   (IOCB1),
 DECLARE (IOCB1,X) ADDRESS;
 X=IOCB1;
 $AARG 'ERADR'
 $ACALL UCALL,CDERR
 $AERADR:  ORG *+2
END PRINT$ERROR;
```

Name: PARSE
Function: Searches a specified input buffer for a file name and reformats the information into the appropriate area of an IOCB. PARSE is designed to help in setting up an IOCB by taking file name information from a line buffer (put there by KEYBD) and relocating it into an IOCB.
Assumptions: SRBLK is an address array with two entries. The first entry points to the address of the input buffer. The second entry points to the logical unit number byte in an IOCB.[3] SRNAM-$STATUS is a byte variable that will hold the status of the PARSE operation.[4] MSG$ADR is an address variable.
Call to Procedure:
```
MSG$ADR=ADDR (SRBLK);
SRNAM$STATUS=SEARCH
 $FILENAME (MSG$ADR);
```
Code of Procedure:
```
PARSE:  PROCEDURE (BLOCK)
   RETURNS (NAME);
 DECLARE NAME BASED (NAME+PTR)
   BYTE
 DECLARE (BLOCK,X) ADDRESS
 X=BLOCK
 $AARG (SRADR)
 $ACALL UCALL,BRNAME
 $ASRADR ORG*+2
 RETURN (NAME);
END PARSE;
```

Name: CDOS
Function: Returns control to the CDOS operating system.
Assumptions: None
Call to Procedure:
```
CALL CDOS;
```
Code of Procedure:
```
CDOS:  PROCEDURE;
 $ACDOS
END CDOS;
```

Name: ENTRY
Function: Set up X and P registers.
Assumptions: None
Call to Procedure:
```
$AENTRY
```
Code of Procedure: None

Name: PLMMAC
Function: To automatically load the PLM library using macros.
Assumptions: None
Call to Procedure:
```
$APLMMAC
```
Code of Procedure: None

Application

The following program, Example 3, Fig. 4, called EXAMPL, shows how to use all of the above-mentioned macros and PLM procedures. The program reads in a file name from the keyboard of the console. If no file name was typed, an error message is printed and control is returned to CDOS. If a file name was typed, the input data is retyped on the console's printer. Control stays with the program and is signaled by the issuance of another prompt. The prompt character is a percent sign (%). Note that the procedures TYPE, KEYBOARD, PRINT$ERROR, PARSE and CDOS, which comprise about 50 percent of the program, do not have to be written by the programmer. The description of the macro definition file, MACRO, is shown in Fig. 5. The CDOS commands required to compile, assemble and execute the sample program are shown in Fig. 6.

References

1. **Operator Manual for the RCA COSMAC CDP18S007,** RCA Solid State publication MPM-232.
2. See Reference 1, Chapter 6.
3. For further information on the IOCB, see Reference 1, Chapter 6, Fig. 18, pp 66.
4. See Reference 1, Chapter 6, pp 71, "IOCB Setup Aid Routine," for a discussion of status values.

```
MACRO                          MACRO
ARG %LABEL                     ENTRY
ORG            *-5             A.1(START)—>R3.1
A.1(%LABEL)—>R7.1              A.0(START)—>R3.0
A.0(%LABEL)—>R7.0              A.1(STACK)—>R2.1
RF.1—>@R7                      A.0(STACK)—>R2.0
INC  R7                        SEP            R3
RF.0—>@R7                      START:  ORG    *
MEND                           MEND

MACRO                          MACRO
PARM1 %LABEL1                  EQUATE
ORG            *-5             CDERR=#28
A.1(%LABEL1)—>R7.1             CDENT=#1E
A.0(%LABEL1)—>R7.0             ZTYPE=#14
RF.0—>@R7                      SRNAME=#24
MEND                           UCALL=#B453
                               KEYIN=#12
MACRO                          MEND
CDOS
CALL   UCALL,CDENT             MACRO
MENT                           PLMMAC
                               [The Standard PLM Library]

                               MEND
```

Fig. 3 - Macro definitions required in PLM/ CDOS interfaces.

```
DO;
/*      DATA DECLARATIONS FOR EXAMPLE   */
        $AEQUATE
        DECLARE PROMPT ADDRESS INITIAL (2500H);
        DECLARE BUFFER (20) BYTE ;
        DECLARE END$BUF BYTE INITIAL (ODH);
        DECLARE BUF$LENGTH BYTE INITIAL (20);
        DECLARE CONTINUE$PROCESSING BYTE INITIAL (0);
        DECLARE NAME$PTR ADDRESS INITIAL (0B452H);
        DECLARE TRUE BYTE INITIAL (0);
        DECLARE FALSE BYTE INITIAL (1);
        DECLARE (MSG$ADR,BUF$ADR) ADDRESS ;
        DECLARE SRBLK(2) ADDRESS ;
        DECLARE IOCB(36) BYTE ;
        DECLARE (SRNAM$STATUS,I) BYTE ;
        DECLARE NULL BYTE INITIAL (80H);
        $AEJECT
/*                                                             */
/*      THE PROCEDURES TYPE, KEYBD, PRINT$ERROR, PARSE AND CDOS */
/*      ARE FROM A LIBRARY OF PROCEDURES AND DO NOT HAVE TO BE  */
/*      WRITTEN BY THE USER                                     */
/*                                                             */
TYPE: PROCEDURE (TYPE$PARM);
        DECLARE (TYPE$PARM,X) ADDRESS;
        X=TYPE$PARM;
        $AARG 'TYPEAD'  ..SETTING UP CDOS TYPE PARAMETER
        $ACALL UCALL,ZTYPE  ..CALL TO CDOS
        $ATYPEAD: ORG *+2  ..STORAGE FOR MESSAGE ADR.
END TYPE;
```

Fig. 4 - Example 3, program EXAMPL.

```
KEYBD: PROCEDURE (KEYBUF,LGT);
        DECLARE (KEYBUF,X) ADDRESS;
        DECLARE (Y,LGT) BYTE;
        X=KEYBUF;
        $AARG 'KEYADR'
        Y=LGT;
        $APARM1 'LENGTH'
        Y=LENGTH(X);
        $ACALL UCALL,KEYIN
        $AKEYADR: ORG *+2
        $ALENGTH: ORG *+1
END KEYBD;
PRINT$ERROR: PROCEDURE (IOCB1);
        DECLARE (IOCB1,X) ADDRESS ;
        X=IOCB1;
        $AARG 'ERADR'
        $ACALL UCALL,CDERR
        $AERADR: ORG *+2
END PRINT$ERROR;
PARSE: PROCEDURE (BLOCK) RETURNS (NAME);
        DECLARE NAME BASED (NAME$PTR) BYTE ;
        DECLARE (BLOCK,X) ADDRESS ;
        X=BLOCK;
        $AARG 'SRADR'
        $ACALL UCALL,SRNAME
        $ASRADR: ORG *+2
        RETURN (NAME);
END PARSE;
CDOS: PROCEDURE;
        $ACDOS
END CDOS;
        $AEJECT
/*                                                           */


/*      THIS IS THE BEGINNING OF THE MAIN PROGRAM            */
/*                                                           */
$AENTRY
DO WHILE CONTINUE$PROCESSING=TRUE;
        DO I=1 TO 20; /*INITIALIZE INPUT BUFFER*/
          BUFFER(I)=0;
        END;
        MSG$ADR=ADDR(PROMPT); /*OUTPUT PROMPT*/
        CALL TYPE(MSG$ADR);
        BUF$ADR=ADDR(BUFFER); /*INPUT FILENAME*/
        CALL KEYBD (BUF$ADR,BUF$LENGTH);
        SRBLK(1)=ADDR(BUFFER); /*SETUP FOR PARSING FILENAME*/
        SRBLK(2)=ADDR(IOCB)+11;
        MSG$ADR=ADDR(SRBLK);
        SRNAM$STATUS=PARSE(MSG$ADR); /*PARSE FILENAME INTO IOCB*/
        IF SRNAM$STATUS=NULL    /*NULL IMPLIES NO FILENAME INPUT*/
          THEN
            DO;
              IOCB(2)=11;       /*SETUP ERROR NUMBER FOR SYNTAX ERROR*/
              MSG$ADR=ADDR(IOCB);
              CALL PRINT$ERROR(MSG$ADR); /*PRINT SYNTAX ERROR MESSAGE*/
              CONTINUE$PROCESSING=FALSE; /*SET FLAG TO STOP PROGRAM*/
            END;
          ELSE
            DO;
              MSG$ADR=ADDR(BUFFER);
              CALL TYPE(MSG$ADR); /*RETYPE INPUT FILENAME*/
            END;
        END;
        CALL CDOS;
        $APLMMAC
END PROGRAM;
EOF
```

Fig. 4 - Example 3, program EXAMPL [cont'd].

MACRO ARG
MACRO PARM1
MACRO CDOS
MACRO ENTRY
MACRO PLMMAC
MACRO EQUATES
EOM STATEMENT

Fig. 5 - Description of macro definition file.

```
PLM EXAMPL (CR)
[PLM output file is called EXAMPL.ASM]
ASM4 (CR)
TYPE SOURCE FILENAME  EXAMPL.ASM (CR)
WRITE TO DSK OR PRINTER (D/P)? D
WRITE?  OUTPUT (CR)
?R,B,M,L,H,U=M
MACRO READ?MACRO (CR)
?R,B,M,L,H,U=L
?R,B,M,L,H,U=U
[Now get a listing and ascertain address of label start. Assume value is
F9. Using CDSBIN, get a binary executable file.]
CDSBIN OUTPUT; F9 (CR)
(EXECUTE FILE).
OUTPUT (CR)
%
```

Fig. 6 - CDOS commands required to compile, assemble, and execute the program of Fig. 5. Underlined sections are printed by the computer.

RCA
Solid State
Division

Microsystems Products
Application Note
ICAN-6918

A Methodology for Programming COSMAC 1802 Applications Using Higher-Level Languages

by W. Fritchie

This Note defines a method of optimizing the time-critical portions of programs written in higher-level languages for COSMAC 1802 applications by recoding those portions in assembly language.

The Dilemma

The three main reasons for using higher-level languages in the development of microprocessor applications are to assure the development of highly reliable software, to reduce overall program development time (including program design, coding, and debugging) and thus the cost of software development, and to aid in program maintenance. Once a program is written in a higher-level language, its maintenance becomes much simpler because its logic is described in a much more readable and understandable form.

Of special importance in the maintenance area is program documentation. The earliest step in program development involves its design, and some form of design notes or documentation, such as flowcharts, are always formulated before the coding begins. However, unless a great deal of time is spent on updating the design information while an assembly language program is being developed, the program, when finally debugged and running, will probably not match the initial set of design notes. To a degree, the higher-level language eliminates the need to go back and update the original design notes because the syntax of the language not only accurately defines the logic of the program but also becomes the design information. Therefore, the use of higher-level languages assures more reliable software, speeds up program development, and aids in program maintenance.

It would seem obvious, then, that microprocessor applications should be designed in a structured higher-level language, and there are a large number of non-time-critical applications that can be written totally in higher-level languages like PLM. Moreover, if time-critical applications are programmed for hardware that is not finalized, it may be possible to redesign the hardware so that its overall performance will match the application requirements when it is used with a program written entirely in the higher-level language. However, because it is not always possible to redesign the hardware, or for other reasons described below, it may not be possible to code the entire application in a higher-level language and still achieve the desired performance. Therefore, the only alternative is to rewrite portions of the program, or the entire program, in assembly language.

This performance dilemma was verbalized in a U.S. Government report[1] as follows:

> "Unfortunately, machine language insertions are necessary for interfacing special-purpose devices, for accessing special-purpose hardware capabilities, and for certain code optimizations on time-critical paths. Here we have an example of Dijkstra's dilemma [see below], in which the mismatch between higher-level language programming and the underlying hardware is unacceptable and there is no feasible way to reject the hardware. The only remaining alternative is to "continue bit pushing in the old way, with all the know ill effects." Those ill effects can, however, be constrained to the smallest possible perimeter, in practice, if not in theory."

Dijkstra's dilemma is stated as follows:

> "In the past, when we used 'low-level language' (assembly language) it was considered to be the purpose of our programs to instruct our machines; now,

Printed in USA/11-80

when using 'high-order language', we would like to regard it as the purpose of our machines to execute our programs. Run-time efficiency can be viewed as a mismatch between the program as stated and the machinery executing it. The difference between past and present is that, in the past, the programmer was always blamed for such a mismatch: he should have written a more efficient, more 'cunning' program! With the programming discipline acquiring some maturity, with a better understanding of what it means to write a program so that the belief in its correctness can be justified, we tend to accept such a program as 'a good program' if matching hardware is thinkable, and if, with respect to a given machine, the aforementioned mismatch then occurs, we now tend to blame that computer as ill-designed, inadequate, and unsuitable for proper usage. In such a situation, there are only a few ways out of the dilemma: (1) accept the mismatch, (2) continue bit pushing in the old way, with all the known ill-effects, and (3) reject the hardware, because it has been identified as inadequate."

Thus, the problem of having to mix assembly-language code with higher-level language code is well known. For example, an application cannot be programmed in a higher-level language if its characters must be sent to a terminal at 19.2 kilobaud; the loop required to process these characters would not be fast enough to handle the baud rate. The solution to this problem is to recode the data-communications driver in assembly language and leave the non-time-critical paths in the higher-level language.

An example of a case in which none of the final application can be programmed in a higher-level language is one in which the program must reside in 500 bytes; most programs require a library of routines larger than 1000 bytes. Therefore, the application in question can be designed in the higher-level language, but the final program must be recoded entirely in assembly language.

Method of Solution

This Note defines a methodology for programming COSMAC 1802 applications using PLM, or for programming any microprocessor application using a structured language like PLM. (It should be noted that PLM itself is not developed to the extent that it is usable in all applications of the COSMAC 1802.) The first step in the method is to design, code, and debug the program in PLM (using structured programming techniques) with the goal of highlighting time-critical paths. In the second step, those time-critical paths that cannot perform fast enough must be recoded in assembly language using the PLM code as an overall design guide. At this point, the application program is complete. The final result of this approach is that the maximum amount of code will be written in PLM. This should be a major goal in any application in order to reduce design time, generate a maintainable program, produce good documentation, and develop the program in a structured way that will assure the highest degree of reliability.

A Practical Example

Higher-Level Language Program

The following example, Fig. 1, shows how the methodology described above can be applied to the writing of a microprocessor application. This example has a time-critical path that was first written in PLM, found to be too slow, and then rewritten in assembly language. The time-critical path is the one involving the transfer of characters from the read file to the write file; the code is found in the innermost DO WHILE loop. The purpose of the program is to combine one or more files into a single file. The multiple files that are to be combined are referred to as the READ$FILE; the single file created is referred to as the WRITE$FILE.

The code in Fig. 1 shows only the program's main section. The code for the associated declarations, procedures, and macros is not shown, but is straightforward and not difficult. The variable CONTINUE$PROCESSING is a global variable that is set to false when an abort condition or a read or write error occurs, or when all programs have been combined. Assembly language statements can be added to PLM source programs by starting the statements with the characters $A. Thus, the first two statements of the program are assembly language statements, where EJECT instructs the assembler to issue a TOP OF FORM character to the line printer and ENTRY is a macro definition that sets up the X-register to register 2 and the program counter to register 3. The macro capability in the assembler can aid considerably in writing applications in PLM for the COSMAC 1802 microprocessors. DC3 is the character that ends the source file. The constant ASCII is equal to the value assigned to a file when the file is ascii. INPUT$FILENAME is a global variable that keeps track of whether a file is opened for reading; this variable is used mainly in the OPEN$READ$FILE procedure. The array FNAME contains the name of the file that is currently opened for reading. Each time a file has been

```
DO;
/* PLACE ASSOCIATED DECLARATIONS AND PROCEDURES HERE */
  $AEJECT
  $AENTRY
/* BEGIN MAIN LOOP */
  CALL SETUP$WRITE$FILE;
  DO WHILE CONTINUE$PROCESSING = TRUE;
    CALL OPEN$READ$FILE;
    CALL OPEN$WRITE$FILE;
    IF CONTINUE$PROCESSING = TRUE
      THEN
        DO;
          DO WHILE CHAR NE DC3;
            PARAMETER = ADDR (READ$IOCB);
            CHAR = GET$CHAR (PARAMETER);
            PARAMETER = ADDR (WRITE$IOCB);
            CALL PUT$CHAR (PARAMETER);
          END;
          PARAMETER = ADDR (READ$IOCB);
          CALL CLOSE (PARAMETER);
          INPUT$FILENAME = NOT$OPENED;
          PARAMETER = ADDR (FNAME);
          CALL TYPE (PARAMETER);
        END;
  END;
  IF OUTPUTS$FILENAME = OPENED
      THEN
        DO;
          IF MTYPE = ASCII
            THEN CALL PLACE$DC3;
          PARAMETER = ADDR (WRITE$IOCB);
          CALL CLOSE (PARAMETER);
        END;
  $ACDOS
  END;
/* END MAIN LOOP */
EOF
```

Fig. 1 · The sample program coded in PLM.

combined, the name of the file, found in FNAME, is typed. OUTPUT$FILENAME is a global variable that keeps track of whether the output file has been opened. The arrays READ$IOCB and WRITE$IOCB contain the input/output control block for the files being read and written, respectively. MFILE is a variable that contains the output file type.

A description of the procedures used in the example follows:

SETUP$WRITE$FILE: This procedure does some initialization for the setting up of the write file.

OPEN$READ$FILE: Before any READ$FILE can be used, the file must be opened. This procedure takes a specific source name representing one of the files to be combined and tries to open that file on a diskette. If the open is successful, control returns to the main loop. If there is a problem, the user is informed of the specific problem and has the option of retyping the file name or aborting the program and returning to CDOS. If the user types another file name, control returns to the beginning of the OPEN$READ$FILE procedure and checking resumes. This loop continues to be executed until a successful open or abort condition occurs.

OPEN$WRITE$FILE: Before WRITE$FILE can be written to, it must also be opened. This procedure works in the same way as OPEN$READ$FILE except that an attempt is made to open the specified file for writing. Control remains with this procedure until a successful open or abort condition occurs.

GET$CHAR: This procedure takes one character from the opened read file and returns it in such a way that it is stored in the BYTE variable CHAR. If any fatal errors occur while the character is being retrieved, an error message is printed and control returned to CDOS.

PUT$CHAR: This procedure outputs the current character stored in CHAR to the WRITE$FILE. In the program under discussion, CHAR is a global variable that can be referenced by the PUT$CHAR procedure. This type of global definition makes it unnecessary to pass the character to output through a parameter. However, it is possible to rewrite the procedure call and the procedure so that the character can be accepted by output only through a parameter. Once again, if any fatal errors occur while the character is being retrieved, an error message is printed and control returned to CDOS.

CLOSE: The close procedure will close the file referenced by the accompanying parameter. If any errors occur during the closing function, an error message is printed and control returned to CDOS.

Two more points should be made concerning the program. First, the statement $ACDOS, found at the end of the program, is another macro definition. PLM transforms this statement into the macro call CDOS. This macro definition is a member of the macro definition file that contains the return mechanism to CDOS. Second, the argument passed to some of the procedures is the ADDRESS variable PARAMETER. This arrangement is necessary because PLM does not allow an address to be generated in a parameter list. Therefore, the address has to be placed in a variable, and that variable used as the parameter.

Mixed Language Program

When the coding and debugging of the application under consideration were complete it was discovered, as mentioned above, that the innermost DO WHILE loop performed too slowly; the following steps have been taken to increase its speed. The DO WHILE loop has been replaced by one PLM statement $ATRANSF, Fig. 2. From this statement the PLM compiler generates one assembly language TRANSF. TRANSF is a macro definition that contains an assembly language routine that transfers data on a sector rather than a character basis. When the output of the PLM compiler is to be assembled, the macro option of the COSMAC 1802 assembler is selected to read the macro definition file that contains the macro definition TRANSF. When the assembler processes the statement TRANSF, it substitutes the code from the previously loaded macro definition. Thus, the final application program, Fig. 2, becomes a blend of PLM and assembly language statements.

```
DO;
/* PLACE ASSOCIATED DECLARATIONS
                   AND PROCEDURES HERE */
  $AEJECT
  $AENTRY
/* BEGIN MAIN LOOP */
  CALL SETUP$WRITE$FILE;
  DO WHILE CONTINUE$PROCESSING = TRUE;
    CALL OPEN$READ$FILE;
    CALL OPEN$WRITE$FILE;
    IF CONTINUE$PROCESSING = TRUE
      THEN
        DO;
          $ATRANSF
            PARAMETER = ADDR (READ$IOCB);
            CALL CLOSE (PARAMETER);
            INPUT$FILENAME = NOT$OPENED;
            PARAMETER = ADDR (FNAME);
            CALL TYPE (PARAMETER);
        END;
  END;
  IF OUTPUT$FILENAME = OPENED
      THEN
        DO;
          IF MTYPE = ASCII
            THEN CALL PLACE$DC3;
          PARAMETER = ADDR (WRITE$IOCB);
          CALL CLOSE (PARAMETER);
        END;
  $ACDOS
END MAIN LOOP;
/* END MAIN LOOP */
EOF
```

Fig. 2 - The sample program of Fig. 1 recoded in assembly language to optimize the speed-critical loop DO WHILE of Fig. 1.

Summary

Skill in making these tradeoffs is the key to writing effective applications in PLM. The end result of the methodology advanced in this Note can be compared with the original motivations for using higher-level languages as discussed above. First, design time was reduced because a large portion of the code was written in PLM. Second, the final program is a rather easy program to maintain. Third, the PLM code can double for a portion of the design documentation because it is much easier to read and understand than a program composed only of assembly language, and it accurately reflects the logic of the program. Finally, the flow of the program is in a structured form. Note that the use of the DO WHILE statement eliminates GOTO's from the code; one of the main objectives in the production of structured programs is the elimination of GOTO's.

In summary, all applications can be *designed* in PLM, but not all applications, if they are to perform optimally, can be completely *written* in PLM. However, the time spent in coding the application in PLM is not wasted because most of the PLM code will be used (assembly language will be required in only small segments of the program), and the PLM code will represent the overall logic and documentation of the program, both of which contribute to its reliability.

Reference

1. "A Common Programming Language for The Department of Defense—Background and Technical Requirements," D.A. Fisher, U.S. Government Report P-1191, U.S. Government Printing Office, Washington, D.C.

-203-

RCA
Solid State Division

Microsystems Products
Application Note
ICAN-6955

Using the COSMAC Microboard Battery-Backup RAM, CDP18S622

by P. H. Merl

The RCA-CDP18S622, COSMAC Microboard 8-Kilobyte Battery-Backup RAM,[1] pictured in Fig. 1, is a Microboard RAM card equipped with rechargeable batteries. The batteries are maintained in the charged state either by the user's system power supply or by an entirely separate, external dc or ac supply. Provisions have been made on the card for an on-board rectifier and regulator and the necessary capacitors to accommodate this external supply, Fig. 2. There are also provisions for an additional, fourth, battery. Linking options provide the user with the ability to power a small system from the battery pack.[2] Logic diagrams of the battery-backup card are provided in Figs. 3 and 4. This Note discusses the application of the board as a standard-power backup medium, a nonvolatile transport medium, and as an efficient means of aiding the testing of new or prototype boards.

APPLICATIONS

Standard-Power Backup

Perhaps the primary application of the CDP18S622 Microboard is in systems subject to power failure. As power in the system fails, the on-board voltage comparator disables memory read (MRD) and memory write (MRW), and all information on the board remains intact in spite of the power failure. The drivers to the bus are held in the high-impedance state, so that no current is supplied to the system backplane. Battery drain is negligible and the information in memory is safe for days because of the low-power requirements of the CMOS RAM. The MRD and MRW functions are actively driven to the false state with system power removed so that the battery-backup RAM card can be removed from the system, transferred elsewhere, installed in another system, and run.

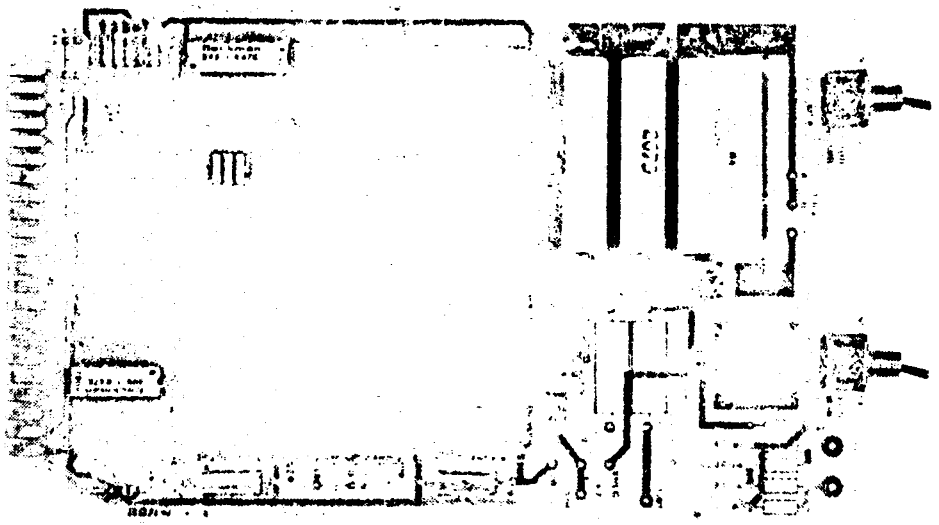
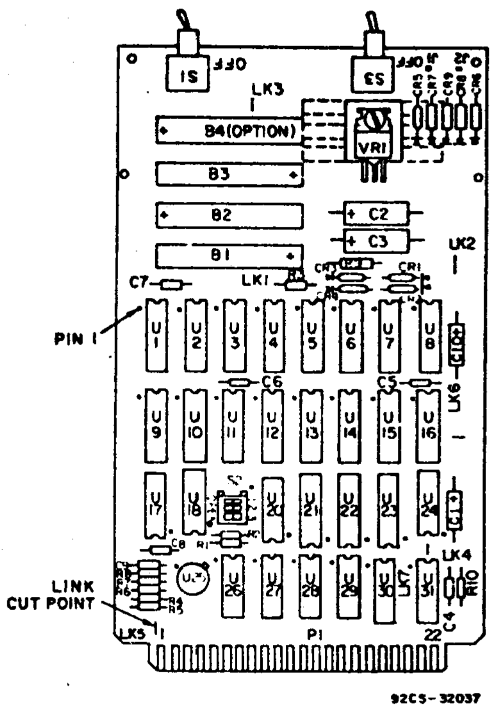


Fig. 1 - The CDP18S622, COSMAC Microboard 8-Kilobyte Battery-Backup RAM.



Parts List

B1 – B3 = nickel-cadmium, 180 mAh, AAA
*B4 = nickel-cadmium, 180 mAh, AAA (Panasonic NR-AAA-U)
C1, C10 = 15 µF, 50 V
*C2, C3 = 220 µF, 20 V (Sprague 137D227C7020F2)
C4 = 0.33 µF, 50 V
C5 – C8 = 0.1 µF, 50 V
C9 = 22 pF
CR1 – CR4 = 1N270
*CR5 = 1N270
*CR6 – CR9 = 1N4001 (RCA D1201F)
*H1 = Heat sink (Thermalloy 6070B)
J1, J2 = terminal, optional 12.6 V ac
R1, R2, R3, R6, R7 = 47 kΩ, ¼ W, 5%
R4 = 5.1 MΩ, ¼ W, 5%
R5 = 10.0 MΩ, ¼ W, 5%
R8 = 1.1 MΩ, ¼ W, 5%
R9 = 15 Ω, ½ W
R10 = 100 kΩ, ¼ W, 5%
S1, S3 = SPDT
S2 = 3-rocker DIP
U1 – U16 = MWS5114E
U17, = resistor module 22 kΩ, 16 pin
U18 = CD4001BE
U20 = CD4070BE
U21 = CDP1867CE
U22, U23 = CDP1866CE
U24 = CD4075BE
U25 = CA3078S
U26, U27 = CDP1856CE
U28, U29 = CD4050BE
U30 = resistor module 22 kΩ, 14 pin
U31 = CD4093BE
*VR1 = 5-V voltage regulator (Fairchild 7805)

*User-supplied components for optional power supply.

Fig. 2 - Layout diagram for the CDP18S622. B4, VR1, CR5-CR9, C2 and C3 are optional items not supplied with the board.

The voltage comparators allow the system voltage to drop below battery voltage before the comparator disables the RAM. This feature assures that small power-supply fluctuations will not gate the RAM off during normal operation, and that only an actual power loss will disable the RAM. Conversely, on power-up, the voltage is allowed to rise above battery voltage before the comparator gates the RAM on. Hysteresis in the comparator circuit prevents oscillation on, and diminishes the noise sensitivity of, the enable signal when power-supply and battery voltage are nearly equal.

The CDP18S622 battery-backup board is involved in system power configurations through five different options. The first option makes use of the board as shipped, with three batteries and no regulator components. In this configuration, the batteries are maintained in the charged state by the user's system power supply. When the system power is off, the batteries provide power for the CDP18S622 only and memory contents are preserved. The second option uses the optional external regulator circuit, which permits an external ac or dc supply to be connected directly to the battery-backup board; this external supply is, again, separate from the user's system supply. In this mode of operation, the external supply maintains the battery pack in the charged state and provides power for the CDP18S622, but not the rest of the system. With a power failure, backup power would be supplied to the battery-backup board only. The third option also makes use of the external supply feature. However, in this option, the entire system is powered from the external supply with the battery-backup capability available to the entire system. The fourth option uses the external supply to power the entire system, but the battery-backup feature applies only to the CDP18S622 RAM. The fifth option

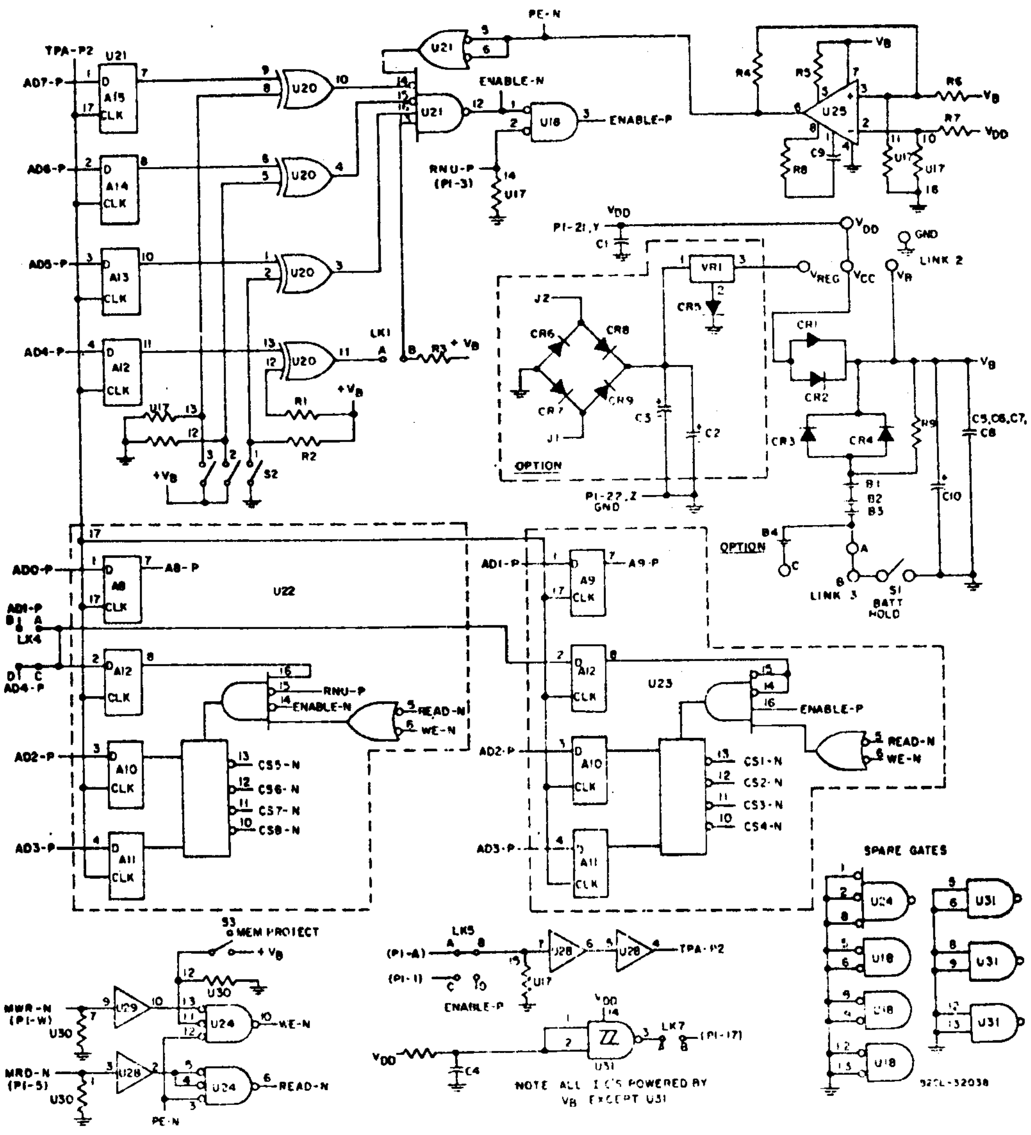


Fig. 3 - Logic diagram of the CDP18S622: control portion and optional power supply.

makes use of the user's system power supply to power the entire system; the battery-pack supply power is available to the entire system during a power failure.[2]

When operating an entire system from the battery pack, it is advisable to install the fourth battery. Installation and linking provisions have been provided on the card.[2]

Non-Volatile Transport Medium

A second application of the battery-backup Microboard is as a non-volatile transport medium. As stated above, the board can be withdrawn from a system, transferred elsewhere, installed in another system, and run without loss of data. Standard conductive packaging should **not** be used during the transfer as this packaging will short out signal, power, and ground lines on the RAM card and run the batteries down prematurely and/or cause stored data to be lost.

As an example of the use of the battery-backup Microboard as a transport medium, consider the case of a development system, such as the CDP18S007 or CDP18S008, for which software has been developed using disk storage, keyboard, and display, software that is now to be run on a remote system that has none of these features. It would take time to program an EPROM, which ultimately might develop bugs, for use on the remote system. The backup Microboard software could be easily debugged with the aid of the Micromonitor, however, and the board inserted in the remote system in running condition.

An example of a typical debugging procedure involving use of the battery-backup Microboard is as follows:

1. Substitute the CDP18S622 RAM card for the RAM in the development system that would normally hold the program.
2. Assure that the memory-protect switch is in the off position
3. Down-load the program into the CDP18S622 RAM card.

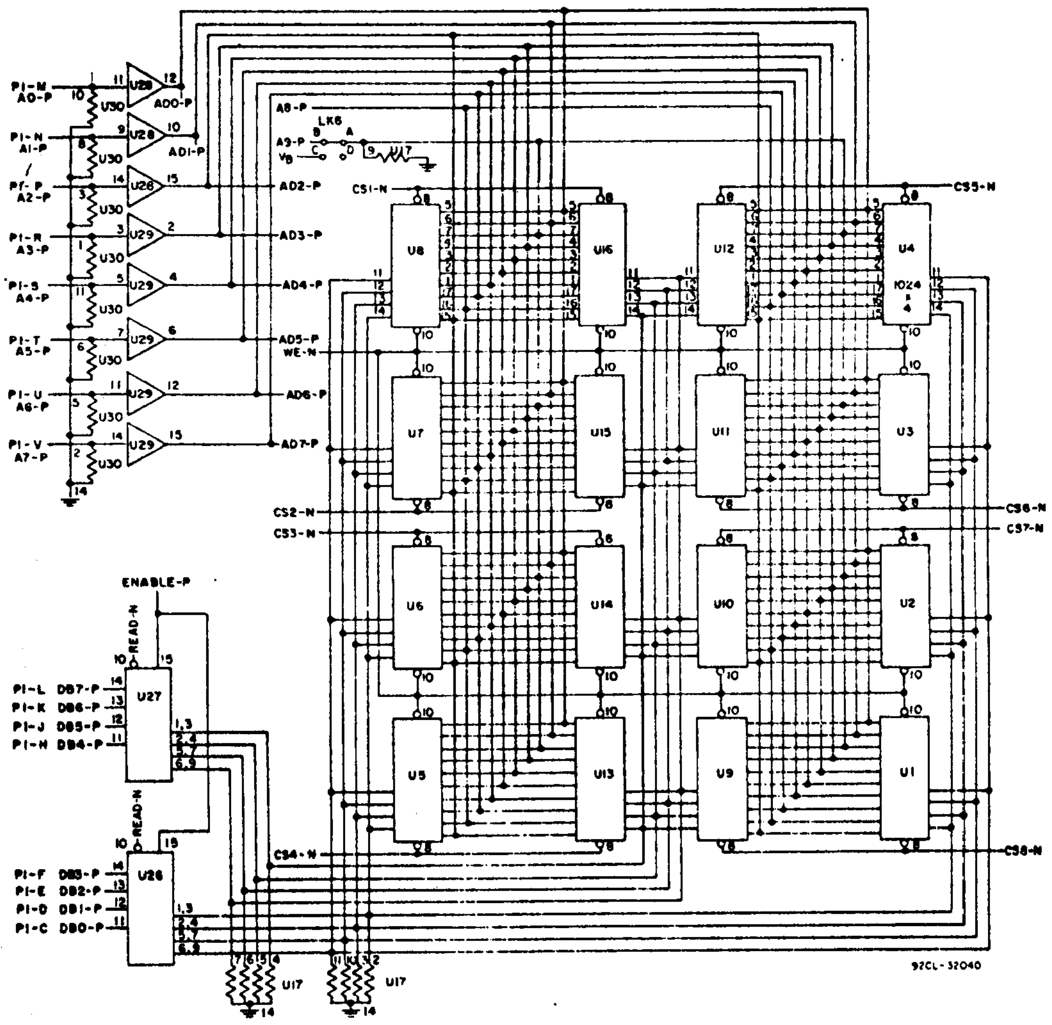


Fig. 4 - Logic diagram for the CDP18S622: memory and buffer portion.

4. Place the memory-protect switch in the on position.
5. Remove the CDP18S622 RAM card from the development system.

   Note: Power need not be removed from the development system when removing or installing the battery-backup RAM cards, although power should be removed when reinstalling the original nonbattery-backup RAM. It is recommended that the development system be placed in the step mode, by using the Step/Continuous switch, before removing the CDP18S622 RAM card; system execution will probably halt if this is not done. After removal of the board, place the switch in the continuous mode and execution will resume.
6. Transport the CDP18S622 battery-backup RAM card to the target system while observing these precautions: Make sure the back of the card does not contact a conductive surface, including Velostat* bags. Do not expose the board to large amounts of static electricity.
7. To change the memory address of the CDP18S622 at the target system, simply change the board switch settings[3] and install the battery-backup RAM card into the target system. See note under step No. 5, above.
8. Reset the target system and run the transported program. At this point it is important to note that if the transported program requires RAM within the bounds of the battery-backup card, the memory-protect switch will have to be placed in the off position.

*Trademark of 3M Company.

The manual memory-protect switch already mentioned above is a significant advantage of the battery-backup RAM card. To test a program written to reside in a ROM or EPROM, the user need not program an EPROM or have a test-run ROM made since a flip of the switch turns the "read/write" memory card into a "read only" memory card. Bugs can be sought out in a simulated ROM environment.

It is important to note that the battery-backup Microboard is not designed to save an executing program if power fails during operation. If the CPU is executing code in the battery-backup RAM card, and if the card is not already in the memory-protect mode, a power outage will cause the memory read and memory write to be disabled even before the CPU stops executing code. Therefore, a system restart will have to be performed whenever power is lost. However, by using the optional power-up reset capability, adding some hardware to detect power failure, and writing a short, fast, interrupt routine, system status could be saved on the battery-backup board so that on the return of power the system could resume operation precisely where the program was interrupted with the power failure.

Testing Other Boards
When testing newly manufactured or prototype boards, the CDP18S622 battery-backup card is a time-saving device. It saves the time spent in reloading the test program for each board tested because it saves the program code; this savings can be substantial in the testing of many boards. The procedure given here should be followed when using the battery-backup Microboard in this application:

1. Put the battery-backup board into the memory area where the test program is to reside.
2. Load the test program from disk, paper tape, magnetic tape, etc. This step is the time consumer!
3. Position the memory-protect switch to on and run the test program, noting hardware failures.
4. Turn off the system power and make the necessary repairs.
5. Return the repaired board to the system or put the next card to be tested into the system, turn the power on, and rerun the test.

Again, no program load phase need be performed because the battery-backup card saves the code.

REFERENCES

1. "RCA COSMAC Microboard 8-Kilobyte Battery-Backup RAM CDP18S622," RCA Solid State publication MB-622.
2. See ref. 1 for parts list and directions for powering an entire system from the battery pack.
3. See ref. 1 for switch settings.

RCA
Solid State
Division

Microprocessors
Application Note
ICAN-6847

Programming 2732 PROM's with the CDP18S480 PROM Programmer

by D. Block

The CDP18S480 PROM Programmer was designed to program a variety of industry-standard PROM's, including the Intel 2704, 2708, 2758, and 2716's and equivalent products from other suppliers. With a simple hardware addition to the PROM Programmer, and without any software changes, the CDP18S480 can also be used to program Intel 2732 PROM's. This addition to the PROM family is organized as 4K × 8 (4 kilobytes x 8 bits) and operates from a single +5-volt supply. All inputs are T^2L compatible except for pin 20 ($\overline{OE}/V_{PP}$), which must be pulsed to 25 volts during programming.

Many electrical and mechanical characteristics have been maintained between the 2716 (organized as 2K × 8) and the 2732 for upwards compatibility. For example, only the definitions of pins 20 and 21 have changed between the two versions, as shown in Fig. 1. In the 2732, pin 20 continues to represent an output enable function, but with V_{PP} (the programming voltage) also impressed on it; pin 21 becomes the A_{11} address pin.

Electrically, the programming voltage (+25 volts) and programming pulse width (50 milliseconds nominal) are the same for both the 2716 and 2732. The technique for 2732 programming described here takes advantage of this fact. The 2732 will be treated as two 2716's for all operations, and the logic circuitry described below will handle necessary logic-level conversions and voltage switching. The hardware is designed to plug into the 2716 socket of the PROM programmer.

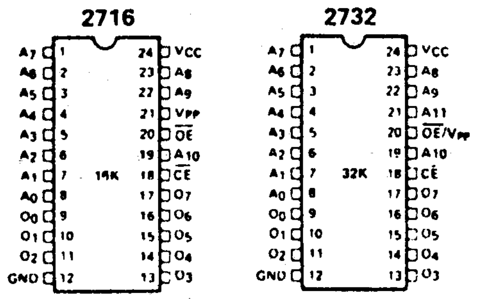


Fig. 1—Pin configurations.

Theory of Operation

Fig. 2 shows the programming waveforms for the 2716 and 2732. Note that the voltages on pin 21 of the 2716 are almost those required for pin 20 of the

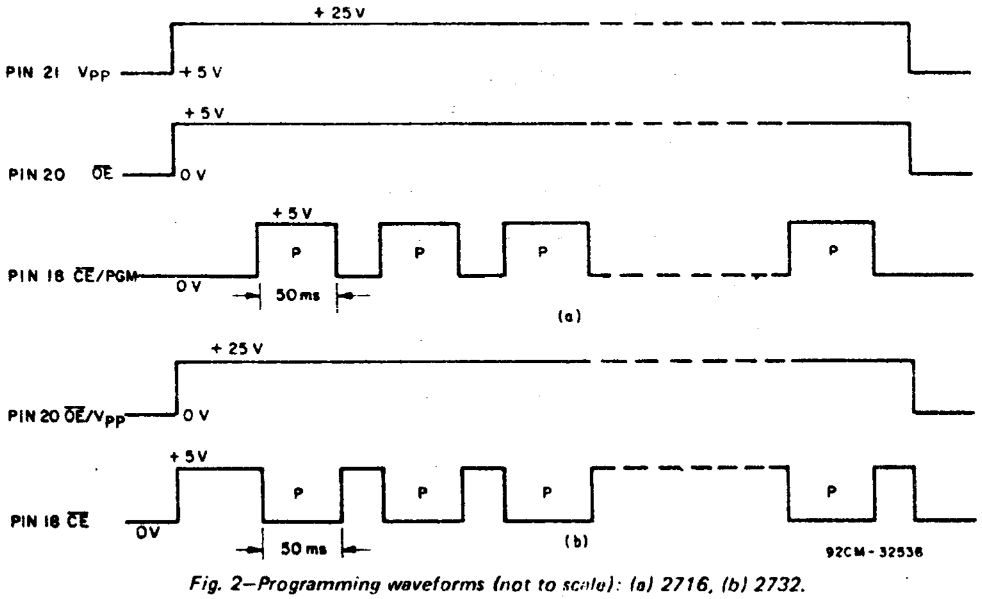


Fig. 2—Programming waveforms (not to scale): (a) 2716, (b) 2732.

Printed in USA/2/80

2732. However, pin 20 must go to ground in the Read mode. The transistor circuitry in Fig. 3 is designed so that when pin 20 of the 2716 is low, the input to pin 20 of the 2732 will also be low. When pin 20 of the 2716 is high, pin 20 of the 2732 is connected to the programming voltage (if a programming operation is in effect) or to +5 volts, disabling the chip.

The programming pulses applied to pin 18 of the two devices differ in polarity. Inversion of the pin 18 signal is accomplished through pin 20 of the 2716 by means of an Exclusive-OR gate. The conditions for reading and programming are met by this arrangement.

As mentioned above, pin 21 on the 2732 is the A_{11} address pin. Fig. 3 shows this pin connected to a toggle switch. The 2732 will be exercised twice for each operation: once with A_{11} low, and then with A_{11} high.

Hardware Construction

The circuitry of Fig. 3 can be constructed on a small vectorboard that plugs directly into the 2716 socket of the PROM Programmer. A ZIF, zero-insertion-force, socket is required for the 2732. It is recommended that solder connections not be made directly to this socket, since the solder may wick up the leads and interfere with operation of the pin clamps. Instead, the ZIF socket should be plugged into a second socket, and connections made to that socket.

Only a CD4049 inverter should be used to drive pin 18 of the 2732. Other CMOS inverters may not have sufficient drive to handle the T^2L input. The use of capacitor C1 is recommended by Intel to prevent transients from exceeding the +26-volt maximum rating of the device.

Operation

The 2732 will be programmed as two 2716's.[1] When programming from a disk file, sequential sections of the file must be loaded in, since the disk-file reader loads only 2 kilobytes at a time into the buffer. Similarly, when copying one 2732 into another, several steps are necessary. The examples below should help clarify the requirements. CDS output is underlined. The CDOS software version is assumed.

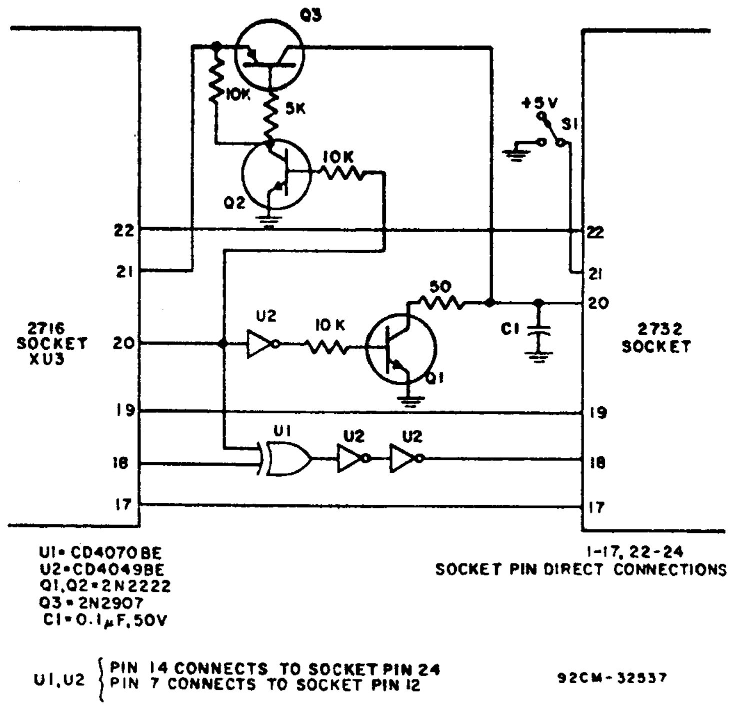


Fig. 3—Circuit diagram.

Example 1:

Program a 2732 from a disk file called FILE.OBJ for the address range 0000-0FFF. First, set S1 so that pin 21 is low.

```
PROM PROGRAMMER, VERSION XX
 ENTER V,C,P,F,S,U: P
 TYPE#: 2716 (CR)
 LOGIC=P,N: P
 (D)ISK OR (R)AM: D
 INPUT FILENAME: FILE.OBJ
 ENTER LOWEST PROM ADDRESS (XXOO): 0000(CR)
 LOADING COMPLETED
 DONE!                   .. The first half of the PROM is done
 REPROGRAM 2716?: N      .. Set S1 in the opposite position
 PROM PROGRAMMER, VERSION XX
 ENTER V,C,P,F,S,U: P
 TYPE#: 2716(CR)
 LOGIC=P,N?: P
 (D)ISK OR (R)AM?: D
 INPUT FILENAME: FILE.OBJ
 ENTER LOWEST PROM ADDRESS (XXOO): 0800(CR)  .. Second half of
                                                address range.
 LOADING COMPLETED
 DONE!                   .. Entire 2732 is now programmed
 REPROGRAM 2716? N
```

Example 2:

Copy two 2716's into one 2732. Insert the lower address 2716 first.

```
 PROM PROGRAMMER, VERSION XX
 ENTER V,C,P,F,S,U: C
 TYPE#: 2716(CR)
 LOGIC=P,N?: P      PAGE#=08(CR)   .. First half=pg 8-F
 DONE!
 PROM PROGRAMMER, VERSION XX   .. Insert higher address 2716
 ENTER V,C,P,F,S,U: C
 TYPE#: 2716(CR)
 LOGIC=P,N?: P      PAGE#=10(CR)   .. Second half=pg 10-17
 DONE!
 PROM PROGRAMMER, VERSION XX   .. Remove 2716, Insert 2732, set S1 low
 ENTER V,C,P,F,S,U: P
 TYPE#: 2716(CR)
 LOGIC=P,N?: P
 (D)ISK OR (R)AM? R      PAGE?: 08(CR)
 DONE!
 REPROGRAM 2716: N                 .. Set S1 high
 PROM PROGRAMMER, VERSION XX
 ENTER V,C,P,F,S,U: P
 TYPE#: 2716(CR)
 LOGIC=P,N?: P
 (D)ISK OR (R)AM? R    PAGE#=10(CR) .. Second half of data
 DONE!
 REPROGRAM 2716? N                 .. Entire 2732 complete
```

Reference:

1. The operating procedure for the 2716 is given in: "Operator's Manual for PROM Programmer CDP18S480," RCA Solid State publication MPM-222A.

RCA
Solid State Division

Microsystems Products
Application Note
ICAN-6934

Cassette Tape I/O For COSMAC Microprocessor Systems

C.D. Smith

This Note describes a circuit and the software needed to add a low-cost cassette-tape input and output to the COSMAC Evaluation Kit (CDP18S020, CDP18S024, and CDP 18S025), the COSMAC Development System (CDP-18S005 and CDP18S007), or the Microboard Prototyping Kit (CDP18S691).

Cassette-Tape Format

The RCA COSMAC VIP cassette-tape format was chosen because it is well documented[1] and requires only a simple software and hardware interface. Bits on the tape consist of one cycle of 2 kHz for a logical 0 and one cycle of 800 Hz for a logical 1. Data is preceded by approximately four seconds of continuous logical 0's for sync followed by a specified number of data bytes. Each byte begins with a logical 1 start bit followed by eight data bits (least significant bit first) and ends with a parity bit; odd parity is used in this code. Fig. 1 shows typical waveforms for recorded data.

Electrical Interface

The electrical interface circuit shown in Fig. 2 is identical to that used with the COSMAC VIP.[2] Bit-serial data is output to U14A-Pin 2

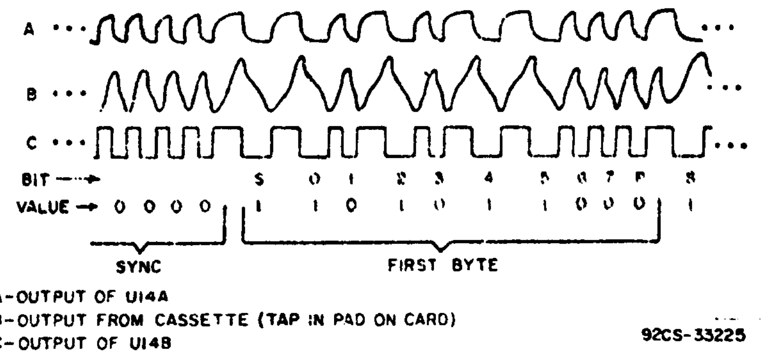


Fig.1 - Typical waveforms of recorded data.

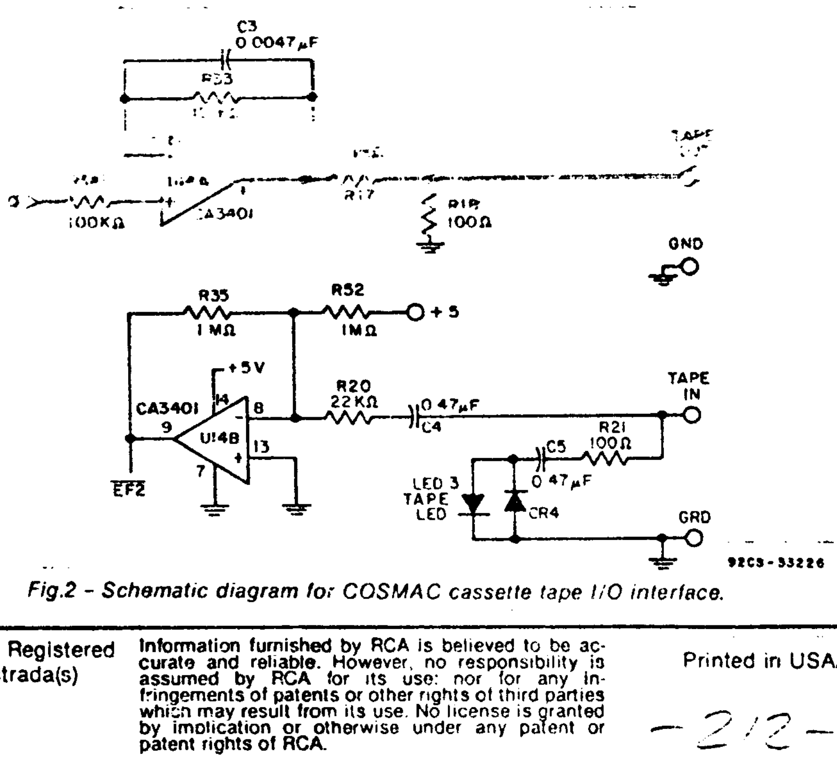


Fig.2 - Schematic diagram for COSMAC cassette tape I/O interface.

Printed in USA/12-80

through the CDP1802 Q bit. Bit-serial data is input from a cassette recorder to U14B-Pin 8 and then to the EF2 input of the CDP1802. Thus, by providing software to control the Q output and sample the EF2 input, data is recorded and recovered from the tape.

Software Interface

Three software routines are described below. The first is an input/output routine for COSMAC systems that use the COSMAC Microterminal as an operator interface. The second is an input/output routine for COSMAC systems that use a standard RS-232 or current loop remote terminal. The third is a boot-loader routine that can be used to load the appropriate input/output routine after power up. This latter routine can be either ROM resident or entered by hand when required.

Using the Microterminal

The use of the cassette input/output routine for the Microterminal-based system requires an understanding of the Microterminal itself.[3] The Microterminal system is divided into three functional sections: control, display, and keyboard.

The control section contains the requisite hardware for controlling the operation of the microprocessor system. The function keys are as follows:

R Reset: Resets the logic of the Microterminal and microprocessor system. Puts the CDP1802 in the reset state.

RU Run Utility: Starts execution of the utility program (UT5), which is at location 8000.

RP Run Program: Starts program execution at location 0000 with R0 as program counter.

CONT/STEP Slide switch to enable continuous or single-step operation of the microprocessor system.

←→ Entry Mode Control: Toggles between the address-entry and data-entry modes.

INC Increment Address: Each depression increments the address shown in the display. In the data-entry mode, it also causes the data byte shown to be written to the address shown before the address is incremented.

$P Start Address Program: Starts program execution at the location shown in the address display.

CA Clear Address: Clears (resets) the address display to 0000.

The keyboard section of the Microterminal contains 16 digit keys (0 through F) that are used to enter hexadecimal numbers into the address or data field. The destination of entered data is controlled by the Entry Mode Control toggle switch (←→). The functions of the hexadecimal digit keys and some of the control function keys are encoded and sent to the microprocessor system on the bidirectional data bus. Logic is provided for scanning and signalling keyboard activity to the microprocessor. The actual scanning, debounce, and decoding algorithms are performed by software routines in the utility program.

The display section consists of an eight-digit, seven-segment, LED display, display drivers, and refresh logic. In either the data-entry or address-entry mode of operation, the display shows a four-digit address field on the left side and a two-digit data field on the right; the fields are separated by blank positions. In other subroutine-oriented display modes, all eight digits are available in two groups of four separated by blank positions. Illuminated decimal points in either the address or data field indicate the current operating mode. Software routines in the utility program perform digit selection, multiplexing, and hexadecimal-to-seven-segment code conversion.

The Microterminal cassette I/O software routine uses the displayed address as the first location to be output or input and the displayed data byte as the number of 256-byte pages to be input or output. If the Microterminal is in the data entry mode when the routine is executed, data is transferred from memory to tape; if it is in the address mode, data is transferred from tape to memory. Initially, the Microterminal is used to enter the cassette I/O routine, Fig. 3, beginning at location #0000. After entry and verification, execute the routine by depressing RESET-RUN P. Then change to the data entry mode by depressing ←→ and enter #01 as data. The address display should be #0000. Start the cassette in the record mode, let the tape advance beyond the nonmagnetic leader, and then depress $P. When output is complete, the routine returns to the monitor program.

Verification of the recorded data is a two-step process:

1. Play the data back into the tape input circuit and adjust the volume to a point where the LED flickers. Then execute the

```
!M
0000 90B3B8F809A3C0810ED4801A3609D4801A3E0ED4801A94BAF822AA6C4AF33227;
0020 8AFF363309301C8AFF23AAFAF03A458CF63338D480E530098E73D4809E60F0AE;
0040 D480D630098AFA0FFC4FAA93BA4AA353560909 1C30091E8EBEF800AE8CF63BAA;
0060 F898A8F80ABDD833632D9D3A66D83B6DF809ADAB9D76BD2BD88B3A741D8DF633;
0080 8A9D5F1F2E9E3A6D30EA9FBA8FAA9EBB8EABD481A63092D3F8103D9A3DA5FF01;
00A0 3A9C1DF880FE35A63097F8D6A8F840BDFC00D82D9D3AB0F810ADF808AB4F8BFF;
00C0 00D89BF6BBD82B8B3AC21D8DF6D82E9E3AB7D830EAD3F80A3BDDF8201D7B73FF;
00E0 0133DF39D57A60F030DFF880B0E0F800A0D0
```

Fig.3 - Object code for Microterminal cassette I/O.

cassette tape I/O routine and load the data into memory. Loading is accomplished by first depressing RESET, RUN, P, then ——. Enter #01 as data and then depress —— and enter #0100 as address. Rewind and start the cassette in the play mode; let the tape advance beyond the non-magnetic leader and then depress $P. After loading, the routine returns to the monitor.

2. Use the Microterminal to verify the content of memory starting at location #0100. If loading is unsuccessful, repeat the loading process while making slight adjustments to the tape recorder volume. After successfully loading the data, make a note of the volume-control position for future reference.

Using a Standard Data Terminal

Using the cassette I/O routine for systems having a ROM based utility program that operates through a standard data terminal (RS-232 or 20 milliampere loop) requires an understanding of the ROM utility program.[4] This program allows the user to examine memory, alter memory, or begin execution at any specified address.

?M(ADDRESS) (SP) (NUMBER OF BYTES) (CR)

— The specified number of bytes in memory, beginning at the specified address, will be transmitted to the data terminal.

!M (ADDRESS) (SP) (DATA) (CR)

— Data bytes consisting of two hexidecimal digits are stored in sequential memory address beginning at the specified address.

$P (ADDRESS)

— Program execution begins at the specified address with P=0 and X=0.

Initially, the !M command is used to enter the cassette I/O routine, shown in Fig. 4, beginning at location #0000. After entry and verification, using the ?M command, execute the routine using the $P command. The system will respond by outputting "VIP CASSETTE I/O FOR RCA EK", followed by "?U, L, S=". Enter U to return to the utility program, L to load memory from tape, or S to save memory on tape, all followed by CR, and the system will respond by outputting "FIRST?". Enter the starting address of the data to be output followed by a CR. The system will respond by outputting "LAST?". Enter the address of the last data byte to be output, but before entering a CR, start the tape in the record mode and let the tape advance beyond the nonmagnetic leader. Unlike the Microterminal I/O routine, which inputs and outputs data in 256-byte blocks, this program allows the user to record and load any number of bytes. However, it is recommended that data be recorded beginning at a page boundary, and that blocks of 256 bytes be output.

To record the cassette I/O program, depress RESET, RUN U, and enter $P0000. Enter S-CR, then #0000 for the "first" address and #01FF for the "last" address. Start the tape in the record mode before entering the last CR. "XFERING" is output, and when the recording of data is complete, the program returns to its beginning.

Verification of the recorded data is accomplished in a manner similar to that used with the Microterminal: Play back the tape, adjust the volume, and execute the program using the $P command. Enter #0200 for the "first" address and #03FF for the "last" address. Start the tape in the play mode, and when the tape advances beyond the leader, enter the last CR. Use the ?M command to verify a successful load.

Boot Strapping After Power Off

To reload either the Microterminal or the data terminal cassette I/O routine after power off, it is necessary to load only the cassette boot routine. After entering and verifying the boot routine in Fig. 5, position the tape just before the saved cassette I/O routine, and start execution by depressing RESET-RUN P, after starting the tape in the

```
!M
0000 710090FC01B8F88CB2F81FA2E290B3B4B5F81BA3F892A4F8B6A5D314D401B10D;
0020 0A56495020434153534554544520492F4F20464F522052434120454B0D0A3F20;
0040 552C4C2C533D203F2000D4813B9FFB5332669FFB4C32729FFB553A1BF880B0F8;
0060 00A0DC12E0D014D40154331B14D400C1301B14D40154331B14D401003B1B14D4;
0080 01B10D0A4C4F41442045525524F5200301BD3C830A39673867393B683A646B346;
00A0 A33091967386739 3B683A646736046A393F4B33091D396B386A36072A6F0B630;
00C0 B5F82EA8F840BDFC00D82D9D3AC7F810ADF808A54ABBFF00D89BF6BBD82B8B3A;
00E0 D91D8DF6D829893ACE993ACED8D50000000000000000000000000000000000000;
0100 F842A8F80ABDD833032D9D3A06D83B0DF809ADAB9D76BD2BD88B3A141D8DF633;
0120 2C9D5A1A29893A0D993A0DFED5D3F80A3B35F8201D7BBFFF013337392D7A9F30;
0140 37D3F8103D443D4FFF013A461DF880FE3550304114D400B10D0A46495253543F;
0160 2000F800BDADD4813B33669FFB0D3AAE9DBA8DAA14D400B10D0A4C4153543F20;
0180 00F800BDADD4813B33859FFB0D3AAE9D73BD7314D400B10D0A58464552494E47;
01A0 2100608AF5A9609A75B919FC00C8FF00D5DC1246BF32BCD481A430B3D5
```

Fig.4 - Object code for data-terminal cassette I/O.

```
!M
0000 7100F801BAF802B990AAA9BFF83BAFF80ABDDF330F2D9D3A12DF3B19F809ADAB;
0020 9D76BD2BDF8B3A201D8DF633379D5A1A29893A19993A19C08000D0F8103D3D3D;
0040 48FF013A3F1DF880FE3549303A
```

Fig.5 - Object code for VIP cassette-tape-format boot loader.

play mode. The most significant byte of the first address is in M(0003), and the number of pages to be input is in M(0006). The contents of these addresses can be changed to suit the individual user's needs.

References

1. **RCA COSMAC VIP CDP18S711 Instruction Manual,** RCA Solid State publication VIP-311.

2. See ref. 1, pp. 70, Fig. E-3-Keyboard, decoding, audio oscillator, and cassette interface circuits.

3. **Instruction Manual for RCA COSMAC Microterminal,** RCA Solid State publication MPM-212.

4. For ROM Utility Program information refer to the following RCA Solid State publications:

For Evaluation Kits:
— **Evaluation Kit Manual for the RCA CDP1802 COSMAC Microprocessor,** MPM-203.
— **Instruction Manual for the RCA COSMAC Evaluation Kit CDP18S020 and the EK/Assembler-Editor Design Kit CDP 18S024,** MPM-224.

For COSMAC Development Systems:
— **Operator Manual for the RCA COSMAC DOS Development System (CDS III) CDP18S007,** MPM-232.
— **Operator Manual for the RCA COSMAC Development System II CDP 18S005,** MPM-216.

For Microboard Prototyping Kits:
— **RCA COSMAC Microboard Prototyping System CDP18S691, CDP18S691V3,** MB-691.

RCA
Solid State
Division

Microprocessors
Application Note
AN-6834

Microprocessor Control for Color-TV Receivers

K. Karstad

A microprocessor control that makes available sophisticated and desirable features not feasible with current standard or custom IC's is described. The TV controller features modular design, which permits essential functions to be implemented on a priority basis while events of secondary importance are held until processing time is available. Physical modularity also permits features to be added in the form of software changes, thus saving manufacturers the expense in time and money of hardware redesign.

The tuning function is implemented by means of a microprocessor controlled phase-locked loop, which constitutes a frequency synthesizer. Scanning and preprogramming of desired channels is possible; for example, a real-time clock allows event programming. The set can be programmed to turn itself on or off or switch stations in a preplanned sequence. User interaction is accomplished through a local keyboard on the set or a remote 35-key unit that transmits pulse-position-modulated infrared pulses. A six-digit LED display provides information and feedback. In this all-CMOS design, a two-level battery back-up system guarantees that stored channels and events will be kept intact for weeks in the event of an ac power failure. An interrupt driven system, with the interrupt load time multiplexed, is combined with a DMA cycle-stealing feature to overcome the difficulties in software structuring.

INTRODUCTION

Until recently the number of user-controllable features on a TV receiver was limited, both for technical and economic reasons, to mechanical channel selection and sound and picture control. The advent of the varactor diode made electronic tuning possible by taking advantage of the voltage-variable capacitance characteristic of the diode. The availability of IC's, in most cases standard MSI circuits or custom MSI/LSI circuits tailored for specific functions, has made feasible the electronic control of channel selection from the keyboard, either remote or part of the set, with the channel number shown either on or off-screen. This type of IC control represents the present trend in the TV industry; however, the availability of the microprocessor makes possible much more sophisticated control of a much wider range of features.

This Note describes a microprocessor control for a color-TV receiver, a control that supports a large number of features and options.[1] As there is no clear industry trend toward the application of sophisticated electronics to control functions, it can be assumed that many of the features discussed will not find their way into the standard home receiver for some time to come. Neverthless, this design, based on the CDP1802 microprocessor chip,[2] illustrates the degree to which the stored program concept can be used to implement receiver control.

The microprocessor control represents a modular approach to TV-control design both in its ability to process functions on a priority basis and in its ability to accept additional or modified functions. Depending on model or market, and without the need for the long-range custom-LSI development phase required in a hardware redesign, manufacturers can quickly implement new features through software changes or additions alone.

GENERAL FUNCTIONAL OPERATIONS

Tuning Control and Preprogramming

The tuning function in this all-electronic system, Fig. 1, is implemented by means of frequency synthesis. A phase-locked loop, PLL, selects the correct local oscillator, L.O., frequency for a specific channel and keeps the receiver tuned, irrespective of changes in the tuner with time and temperature. When any number from 1 to 99 is keyed into the keyboard, the signals are translated by means of data in a ROM look-up table into the 14-bit code, dictated by international channel allocations, required to set the PLL on the constant L.O. frequency for the channel desired.

Printed in USA/8-80

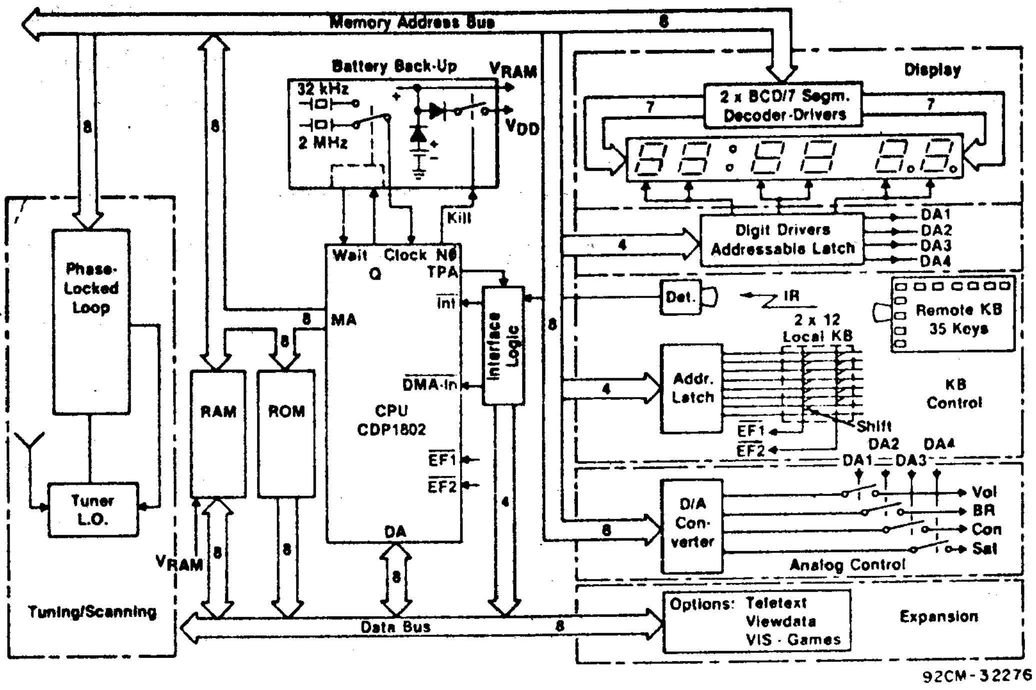


Fig. 1 — The microprocessor-control system. The user interacts with the controller by means of either the local 2 x 12 keyset or the 35-key remote unit. Clock and program information is shown on the six-digit display. Exact tuning is accomplished through the phase-locked loop. The battery back-up system keeps the clock running and stored information intact in the event of a power failure.

The PLL design also permits direct fine tuning over a whole channel. While the traditional need for fine tuning is eliminated by the exactness of the frequency synthesis approach, some users desire it in fringe areas for improved picture quality. There is also, in some areas, a need to tune in the nonallocated channels used for cable TV and master antenna systems; the fine tuning capability makes this possible. In the VHF band, fine tuning is implemented in 25-kHz steps, and in the UHF band, in 100-kHz steps.

An important feature of the microprocessor tuner is channel preprogramming. In the following discussion, the distinction made between programs and channels is that commonly practiced in Europe, where a given program may be assigned to two different channels in two different geographical areas in much the same way as different networks are assigned different channels in the U.S. The tuner can store 16 programs, which can then be called out simply by depressing a key number between 1 and 16.

This preprogramming technique requires that the channel to which a given program is assigned be known. For the more usual case, where the channel is not known, a scanning mode is available. Depression of the scan key will initiate a search starting at the channel to which the set is currently tuned and terminating at the next higher channel available on the air. If the scan button is not depressed again, that channel is automaticaly assigned to the stored-program location currently chosen. With each depression of the scan button, the tuning system searches for the nearest higher available carrier. If no station is found, the scan wraps around and stops on the frequency from which the search began.

Display and Keyboard Control

Channel and program numbers are shown on a six-digit, LED, off-screen display. (Display and keyboards are shown in the Appendix.) For a period of time the on-screen display was popular, but the implementation of the off-screen display is simpler; it can also be on permanently, and does not distract the viewer's attention. The optimum number of digits permitting continuous display of clock and program number, the normal or prioritized display mode, is six. Upon request from the keyboard, both channel and program numbers are shown, the channel number displacing the clock display. In the scanning mode, the channel frequencies assigned to program numbers are automatically displayed.

The availability of a real-time clock permits event programming; i.e., a program can be automatically turned on or off at a specific time. Again, the six-digit display is sufficient for the easy programming of events as well as the listing of events already programmed.

The local keyboard, with 12 keys and shift-key, can handle 24 tasks, including program select scan, analog control (up/down), mute, on/off, and clock set. The master keyboard is remote, has 35 keys, and communicates with the receiver over an infrared link. Each depressed key generates a unique seven-bit code as a pulse-position-modulated (PPM) pulse train;[3] zero and one bits are defined by specific distances between pulses. This pulse train, an asynchronous event with respect to any other task, is processed by the DMA (direct memory access) channel. Hence, a random real-time event is given priority.

The analog functions, volume, brightness, contrast, and saturation, are changed continuously up or down by holding down the appropriate key. A six-bit word in memory, providing 64-step resolution, is output at a predetermined rate and incremented or decremented by one each time. With sample-and-hold multiplexing techniques, one output port is sufficient for serving four different channels.

Event Programming

The six-digit display and 24-hour clock make it possible to preprogram and monitor events for a period beginning at the present and extending 24 hours into the future. An arbitrary number of memory locations, eight in this design, are reserved for event programs. The user, while watching the display, simply keys in the time for the set to turn on (or to switch channels if already on) and the program number. This procedure can be repeated until the list is full, which is indicated by flashing 9's on the display. There is no restriction on the sequence of entering events. The software orders the list in time sequence so that the event nearest in time is always at the top of the list. The software checks every minute to determine whether an event is to be activiated; if it is, the event is executed and then erased from the list, and the next event in time is moved up. A lit decimal point on the display indicates that an event is pending. When the event has been executed, the decimal point begins flashing and continues to flash until the viewer acknowledges by depressing any key. If no acknowledgment is received within five minutes, the set turns itself off.

The sequence above describes an event that occurs only once; it is a simple matter to program an event so that it is automatically repeated every 24 hours, for example, an event that turns a program off at a specified time each day. The programmed information (time, program number), along with an indication of whether it is a one-time-only, off, or repeat event is listed on the display by use of a "list" button on the keyboard. An empty or exhausted list is indicated by flashing eights.

Expansion

The modular structure of the microprocessor tuner makes it possible to add options by adding ports to the data bus and segments of code to the software. Two options of considerable future interest are Teletext[4] and Viewdata[5]. If the receiver in question is equipped with the Teletext option, for example, the keyboard will contain a key marked Teletext. When this key is depressed, the software will check for the existence of this option. If it is present, certain keys will from that moment be redefined and, when depressed, will lead to the execution of tasks different from those of the normal TV mode. The latter mode is recalled by use of another key, NTV.[6]

SYSTEM SOFTWARE

The requirements on the software portion of this real-time controller are to multiplex the six display digits, multiplex four analog channels, update the clock, monitor the two keyboards, and execute the tasks called for. The clock must, of course, be updated regularly and frequently so that it does not lose time. The display must be refreshed at a rate high enough to avoid flicker, i.e., at 50 to 60 Hz at least. The analog values must also be refreshed steadily at a rate high enough to maintain a dc signal and without excessively large filter constants. Certainly, the keyboards must be checked frequently enough to catch any random key depressions. Finally, whatever command is received, fine tuning, scan, mute, etc., must be processed and executed. Note in this context that some of the tasks, such as volume up/down, require continuous depression of a key for an indeterminate amount of time.

These overlapping and partially conflicting specifications are met by an interrupt-driven system together with the DMA channel used for the remote keyboard.[7] A pulse train, derived from the CPU clock, interrupts the software program every four milliseconds, as shown in Fig. 2. The interrupt routine, which must be executed with regular frequency, is divided into three sections to assure distribution of the load. Each time the interrupt service routine is called, a test is made to determine which cycle is to be serviced next. Hence, three cycles comprising a complete frame are sequentially serviced. After 12 milliseconds, a full interrupt service frame repeats.

The six digits of the display are grouped in pairs, two per cycle. This arrangement reduces the multiplex rate to 1:3 and provides twice as much time between interrupts as would otherwise be the case. The refresh rate for the display is now 83 Hz, well above the critical flicker rate.

The sequence of events in the interrupt routine is as follows: An interrupt occurs. The last digit pair is turned off and the next pair is turned on. The four analog channels are then multiplexed and refreshed and the clock is updated. Con-

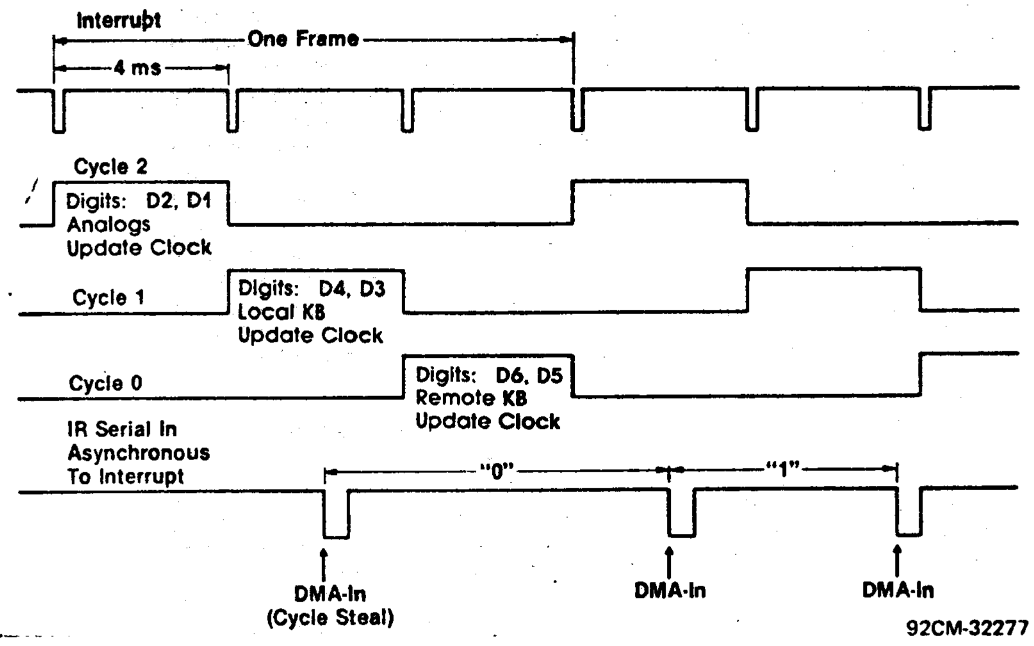


Fig. 2 — *The interrupt system. The system is interrupt driven with the interrupt service routine split and time multiplexed. Remote infrared control pulses are received in real time through direct memory access without interfering with ongoing processing.*

trol then returns to the main program. At the next interrupt the last digit pair is turned off and the next pair is turned on. The clock is updated again, but in this cycle, the local keyboard is also scanned for key closure. At the third interrupt, the last digit pair is handled, and the service routine checks for reception of a command from the remote keyboard. If a character has been received, the command is processed and executed. Once more the clock is updated and the main program resumes execution. All urgent tasks are thus served regularly and frequently: the clock is updated every four milliseconds, and every twelve milliseconds the display and the D/A channels are refreshed. In addition, both keyboards are monitored, and if a key closure has taken place, the required processing is performed for the main program to act upon later.

A key closure on the remote keyboard generates a bit string of pulses, and for some commands, a long character string. These pulse trains, which occur randomly and sometimes last for long periods, are received without tying up the processor and neglecting other real time events by having them fed into the DMA input. Thus the keyed information is conveyed to the CPU through a cycle-stealing process and, to all practical purposes, without slowing, or interfering with, the CPU's current operation.

The interrupt service routine in each four-millisecond time slot may vary in length according to which cycle is on, and particularly according to how much of the clock update routine is required at a specific moment. Nevertheless, the processor is idle during the major part of the time slot, and is available for background processing in the main program.

When control returns to the main program in each time slot, the program determines whether a command was received from one of the keyboards. If it has been, that task, for example, change brightness, is executed. If no command has been received, the CPU is essentially idle until the next interrupt occurs. In most cases, the free time in one of the three cycles is ample time for the processing of any task in the main program: if it is not, the next interrupt simply postpones the background processing until one or a few more time slots are available. This manipulation is not apparent to the user.

The flowchart for the main program, Fig. 3, provides additional information on the points mentioned above. An interrupt can occur at any time in the main program; however, most of the time, if no keyboard command has been received, the program loops through the upper portion of the chart. A few other tests are also regularly performed: are there any events pending, was an event executed but not acknowledged, has ac power failed, or is no station on the air.

THE HARDWARE/SOFTWARE INTERFACE

Clock, Display and Keyboards

Four locations in the RAM are assigned to hold the four clock digits. As shown in the flowchart of Fig. 4, at every interrupt, a seconds counter is incremented. Later,

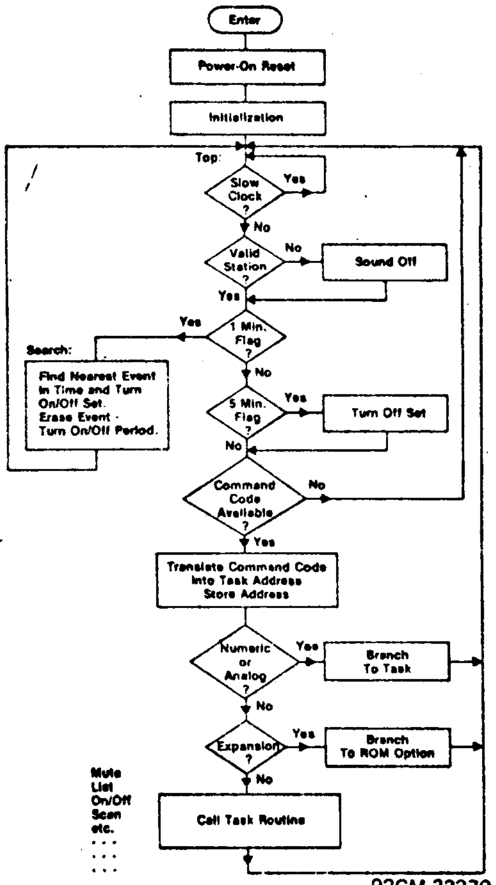


Fig. 3— The main-program flowchart. When a keyboard command is received, the program branches from the upper loop and executes the task called. The main program also checks every minute for a programmed event that calls for action; the user has five minutes to acknowledge an event execution.

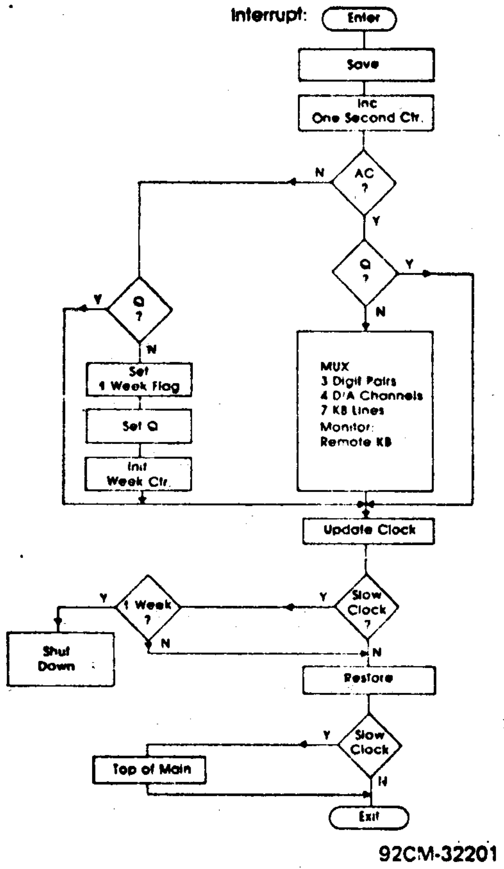


Fig. 4— The interrupt-routine flowchart. The interrupt service routine increments the seconds counter and updates the clock. It also multiplexes the display and the analog values and scans and monitors the keyboards. If ac power fails, the CPU switches to a low-frequency clock and battery power.

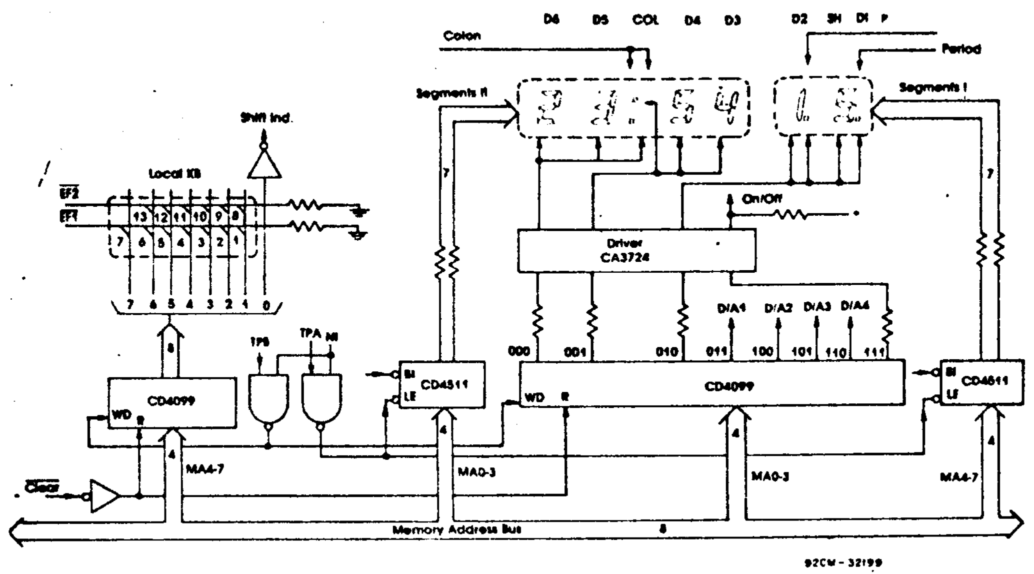
In the update routine, the counter is tested for the value of one second, and an update buffer is loaded.

Another buffer in RAM, the display buffer, contains BCD data for the six display digits. The update routine transfers data for the four clock digits from the update buffer. The display buffer also contains addresses for the digit pairs, the seven scanning lines for the local keyboard, and the four multiplexed D/A lines.

The configuration of the data in the display buffer is closely related to the hardware scheme shown in Fig. 5. Segment data for a digit pair are fed from two separate latch/decoder circuits. An eight-bit addressable latch selects the digit pair or the D/A channel to be sampled. Another addressable latch provides the scan address for testing key closures on the local keyboard. The required 16 bits of data are assembled in one 16-bit CPU register and output over the memory address bus with just one instruction.[8]

Four RAM locations store the six-bit D/A data, which is output before the D/A channel is sampled. The latched data is smoothed by a simple R/2R network; no greater precision is required.

The software section of the interrupt routine, which interacts with the hardware of Fig. 5, is detailed in the flowchart of Fig. 6. The program turns off the last select line for a digit pair and tests whether it was cycle 0. If the answer is yes, a new frame of events is about to be repeated and the display pointer is reset to the top of the display buffer. Segment data for the next digit pair are output, and the pair is selected. In this example, the answer to the test of cycle 2 is yes; therefore, the four D/A channels are sampled in a burst mode, after which the routine exits to its update portion. At the next interrupt, since it was not cycle 0, the program turns on the next digit pair, finds itself in cycle 1, and activates the local keyboard routine. Finally, at the third interrupt (cycle 0), the remote keyboard routine is sequenced.



Output Register Data Format

Segments I Data	Segments II Data	Data	Keyboard Scan Address	Data	Address Digit Pairs D/A Channels
BIT: 15 14 15 12	11 10 9 8	7	6 5 4	3	2 1 10

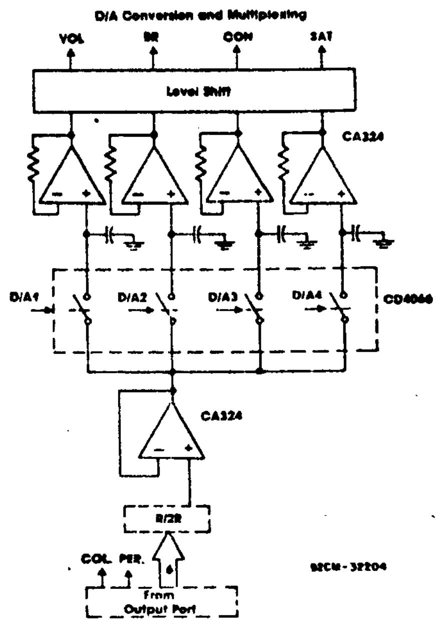
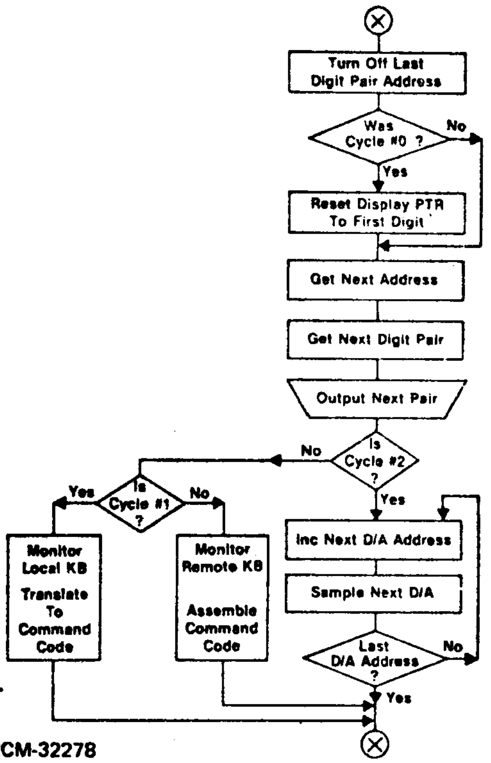


Fig. 5— The multiplexing of the six-digit display and the four analog channels, and the scanning of the local keyboard, is accomplished with a compact 16-bit data format. The eight-bit microprocessor outputs the full data word over the memory address bus with only one instruction.



92CM-32278

Fig. 6— This detail of the interrupt service routine shows how time-division multiplexing is accomplished in three cycles in order to distribute the real-time load.

A key depression on the local keyboard is translated into a command code and stored in memory for the main program to process. The sequence of events is as follows: One of the output lines from the addressable latch is activated; the CPU tests the first flag-line for key closure (Fig. 5). If the output line is line 4 and EF1 is tested (defined as row 0), a possible key closure is identified and marked as key number 4 + 0 = 4. The CPU next tests line 4 and flag EF2 (defined as row 7). A key closure here is identified as key number 4 + 7 = 11. The key number is used as an address pointer to a ROM table that contains the actual command code for that particular key (for example, mute). Because the local keyboard has only 12 keys to handle 24 functions, a shift-key depression must also be verified. If the shift key is down, 12 is added to the table address, causing a jump to the second part of the table containing the shift commands. The keyboard scan repeats every 12 milliseconds. If closure is detected, a debounce routine allows 36 milliseconds for verification of a valid key depression.

The received and detected IR pulse train from the remote unit is a PPM pulse stream composed of six bits plus a parity bit for each character. The software monitors the presence of a bit stream, measures the time between pulses, thereby discriminating between ones and zeros, and assembles the information into a character that represents the actual command code. Finally, the program tags the received command code as to what type it is, a single command (on, off) or a repeat command (volume up), for the duration of the key depression. Ones or zeros are determined by measuring the time between pulses.

A four-bit counter is clocked by a 10-kHz signal. The content of the counter is read at the beginning of each IR pulse, and the counter is restarted. Hence, the number of pulses read represents the time interval between the last two pulses. As the IR pulse itself activates the DMA-in line, the time count is automatically read into a memory location. The DMA pointer advances in readiness for another input byte, which occurs at the next IR pulse. Hence, the seven bits in a command, represented by seven counts, are automatically stored in memory. If the time base is chosen well, it is possible, while maintaining good resolution, to get a count in which the most significant bit reads directly and correctly 1 or 0 for the time interval. It remains only for the CPU to repack the 1's and 0's into a command code word, which is stored for later use. The software code used for servicing the remote unit is in many ways similar to a UART program; it checks for correct number of bits and framing error, and ignores noise pulses and other error conditions.

Tunning and Program selection

A prescaler in the closed loop circuit, Fig. 7, divides the outputs from the L.O. by 64 for VHF and by 256 for UHF; the dividing ratio is defined as K. The output of the prescaler, now within the frequency range of CMOS circuitry, feeds a 14-bit divide-by-N counter, the output of which is compared with a reference frequency. The output from the phase detector is filtered, amplified, and applied to the varactor diode used to tune the L.O. In sum, the L.O. frequency of the tuner is measured, and whatever control voltage is necessary to force the frequency to be correct for a selected channel is generated by the loop. At phase lock, when the inputs to the phase detector are of the same frequency, the L.O. frequency of the tuner is N times K times the reference frequency. Therefore, the divide-by-N counter, which is under software control by the user, selects the channel.

The data format is a 16-bit word; the two most significant bits are decoded to select one of four bands (VHF-I,,VHF-III, VHF, extra). The remaining 14 bits are the binary representation of the number N for a specific channel. The 14 bits provide sufficient resolution to place the L.O. frequencies as little as 25 kHz apart for VHF. Thus, it is possible to use the tuner to select any channel in the world, as well

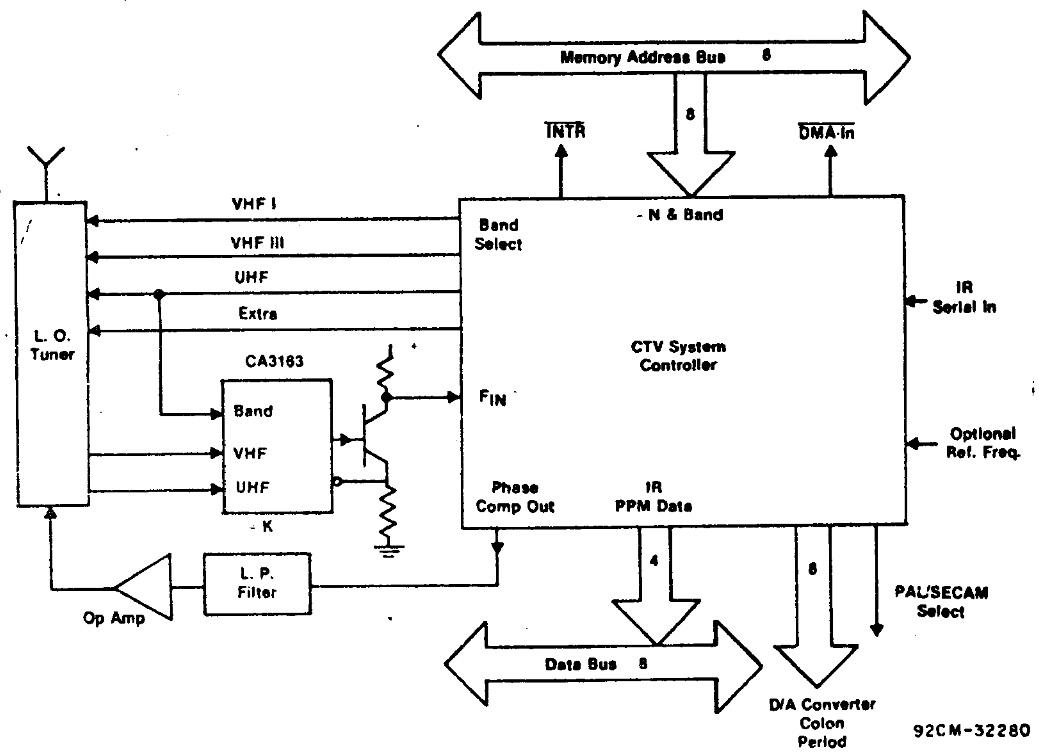


Fig. 7— Precise tuning and station selection is done with a phase-locked loop. The loop is controlled by an LSI chip that contains a divide-by-N counter, band select, phase comparator and reference frequency dividers. Band and channel frequency information is contained in a 16-bit data word sent over the memory address bus.

as unassigned channels used in cable TV and MATV installations. Additional fine tuning is implemented with the same resolution. The 16 bits of data are output over the eight-bit memory address bus with only one instruction, using register-based output. With the exception of the ECL prescaler, the low-pass filter, and operational amplifier, all circuitry is integrated on one CMOS chip. The required interface logic for the remote keyboard, discussed earlier, is also included on the chip.

The PLL system employed in the microprocessor tuner differs fundamentally from the open-loop analog system commonly employed. In the analog system, a predetermined voltage is applied to the tuner with the expectation that the generated frequency will be correct. But because there is no feedback, changes in components with age and temperature influence the frequency. Neither is automatic compensation achieved for the unavoidable differences that occur in tuner characteristics.

Note that with a PLL system, preprogramming of channels and events can be done at any time; no stations have to be on the air. In fact, it is now possible for the dealer to preprogram sets before they are delivered to customers, provided a battery back-up system is implemented.

Battery Back-Up

A two-level battery back-up system takes over in the event of a power failure. Because of the all-CMOS circuitry of the logic and processor, this system can be implemented with only four rechargeable low-capacity NiCd cells.

In order to conserve power, the CPU, upon detecting a power failure, automatically switches from the 2-MHz crystal in normal use to a low-frequency 32-kHz crystal. The switching takes place during a few milliseconds "wait" state which the CPU also enters automatically with a power failure. If power is not restored within the predetermined number of days, seven, the CPU shuts itself down after shutting down the whole system with the exception of the RAM storing user-programmed information. The remaining battery power is sufficient for approximately three months of storage. If power returns before three months, operation resumes as normal with stored information intact.

The slow-clock mode works satisfactorily because, during the absence of ac power, the only task performed by the software at each interrupt is a clock update. Essentially the same update routine is used, except that, in the slow-clock mode, a branch instruction sets a new limit for the number of interrupt pulses required to measure one second. Again, the software ignores all code except update clock and testing for the presence of ac power.

Acknowledgement
The author gratefully acknowledges the contributions of G. Fogarty. The bulk of the material contained in this Note was published in IEEE Transactions on Consumer Electronics, May 1980 (Vol. CE-26, No.2, 0098-3063/80/0149-0156$00.75 ©1980 IEEE).

REFERENCES

1. The few microprocessor-controlled receivers currently on the market are described in: Kleinman, A., "Programmable Color TV," Radio-Electronics (May, 1977).

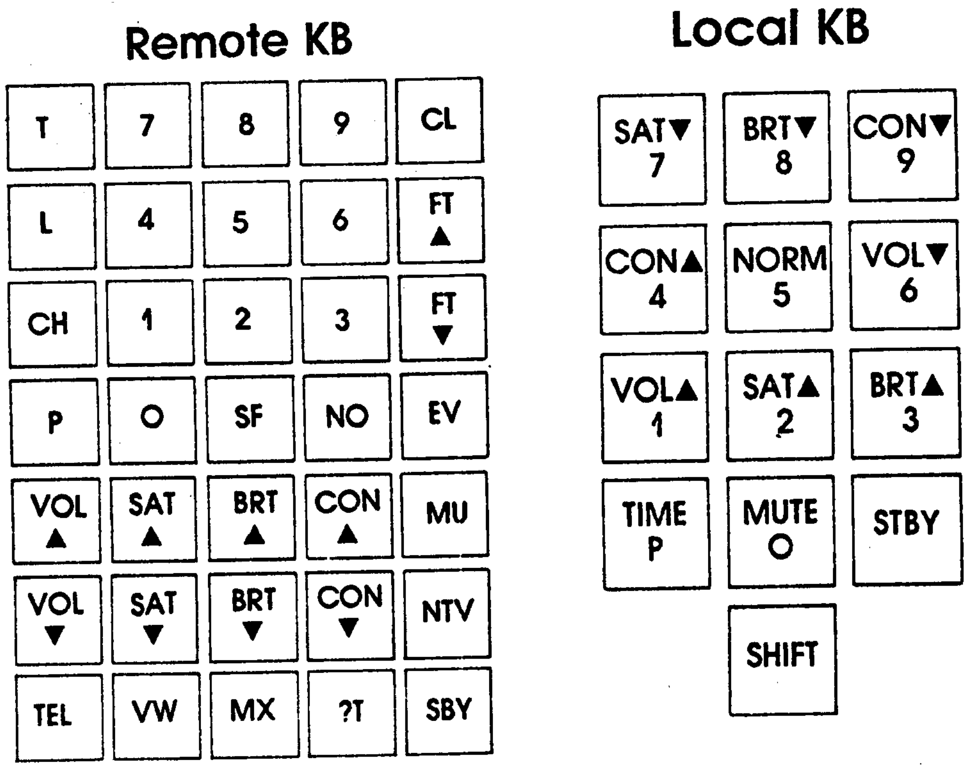
   Baum Wolfgang, "Farbfernsehgerat mit Microprozessor - Steuerung," Funkschau, Heft 17 (1977).
2. The microprocessor used in this design is the CDP1802. If the CDP1804 is substituted for the CDP1802, two kilobytes of ROM and 64 bytes of RAM will be available on the CPU chip.
3. Valvo: Data Sheet SAB 3011.
4. "Teletext - The LSI Solution," Mullard Technical Information TP1606.
5. "Viewdata," Wireless World (Feb. 1977).
6. A video interface system (VIS), which also is Teletext compatible, is another option. This system, built around two LSI CMOS chips (CDP1869 and CDP1870) offers a variety of formats for displaying and modifying data under software control with either NTSC or PAL compatible output signals.
7. User Manual for the CDP1802 Microprocessor, MPM-201B and COSMAC Microprocessor Product Guide, MPM-180B, RCA Solid State Division.
8. "Register-Based Output Function for the RCA COSMAC Microprocessors,"RCA Solid State Application Note ICAN-6562.

APPENDIX

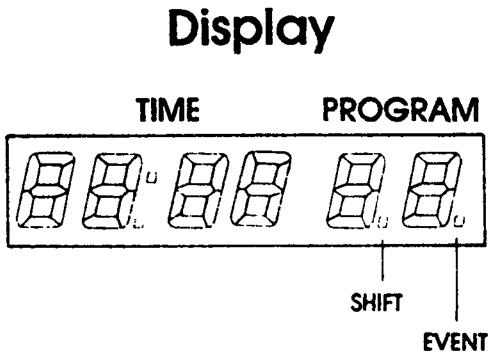
Summary of single and compound keyboard commands. (t and p refer to numerals 0 through 9). Keyboard diagrams are shown on the following page.

Commands	Actions
ANY KEY	Acknowledge automatic turn-on
BRT	Brightness adjust
CH	Channel scan
ppCH	Preprogram channels
CL	Clear. Returns program to previous mode. Instant escape from error mode. Return from listing, scan.
COL	Color adjust
CON	Contrast adjust
EVppPttttT	Preprogram a one-time event
EV00PttttT	Preprogram off
FT	Fine tuning
L	List preprogrammed events
L NO	Cancel preprogrammed event while listing
MU, MUTE	Mute
MX	Mix, expansion command
NO NTV	Four factory analog values are called
NTV	Return to normal TV mode. Four user's analog values are also restored
P	Program scan
ppP	Program selection. Turn set on with pp = 1—16

SAT	Saturation adjust
SF CH	What channel is currently assigned?
SF EVppPttttT	Preprogram a repetitive event
SF NTV	Store user's analog values
SF P	Toggle between PAL or SECAM decoder
STB, STBY	Standby: toggles set on/off
T	Toggle clock display on/off
ttttT	Set clock
TEL	Tele, expansion command
? T	? Time, expansion command
VW	View, expansion command
VOL	Volume adjust



92CM-32202


Display
TIME
PROGRAM
SHIFT
EVENT