4.2 SYSTEM DIAGNOSIS USING HARDWARE PROGRAMMER AIDS

In addition to the techniques described in which the user applies oscilloscopes and his own innovative techniques for analyzing data, Rockwell makes available to the user several hardware aids which assist in the debugging of a microcomputer system and also a software aid called the "Emulator." The hardware aids are a Keyboard Input Monitor (KIM) and a Teletype Input Monitor (TIM). Each of these aids attempts to reduce the problem of debugging the code to the same techniques that are available on a large computer system, and each is designed to allow the debugging of microprocessor code without need to resort to scopes or other data-trapping techniques.

The basic assumption of each of both the KIM and TIM hardware and the Emulator software, is that the microprocessor system is connected correctly, that all of the electrical characteristics have already been checked and met, and that the only problem to be solved is one of debugging programs and I/O hardware which have been entered into the microcomputer.

Each of the hardware techniques assumes that the user will start his design sequence with all of his programming being done in some form of random-access memory which is loadable from an I/O device, examinable by the I/O device, and changeable by the I/O device. This is the normal first step in developing a microcomputer system, and one that should be used prior to committing any of the hardware to PROMs or alterable memory. The only exception to this is if the user is taking advantage of the software Emulator and if his program is such that the Emulator can give him a significant degree of confidence in his coding. In this case the user of the KIM or TIM devices is primarily that of allowing him to have final debugging access to his various memory locations. Therefore, the common characteristic of all these approaches is that by some technique (in the case of the Emulator by reading an input file, in the case of TIM by reading an input tape from the output Cross-Assembler, and in the case of KIM loading a program into memory by hand) the program has been entered into a program storage. Each of these techniques allows the user to initialize various memory and register locations and to

"start execution" of this program at a memory location. Techniques are implemented which permit the user to stop his program at a particular point and analyze the results of the operations which have just been completed. If the results are correct, the coding between the start point and the stop point is correct. If the coding is incorrect, the user analyzes the data which he displays by means of the I/O device and the hardware or software that interfaces it, and determines by inspection of the data and analysis of his coding the error which could cause the results detected.

If the technique of merely analyzing coding is insufficient, each of these systems has the ability to allow the user to go in and re-execute the code with new data or the original data, stopping only at earlier stop-points until he is able to trap the operation that causes the erroneous data to occur. The Emulator has additional features which permit the user to analyze the operation of instructions as they occur which is very useful in determining which part of the program causes operations to be performed incorrectly.

The normal design cycle should actually include a combination of techniques. The user should write his code on a Cross-Assembler and debug much of his loops and non-I/O programming using the Emulator. The Emulator has been designed to allow very easy analysis of data paths, loops and performance of program on a non-hardware basis. It is particularly valuable for the user who is developing routines which have significant loop and subroutining or any completed algorithm.

The use of emulation has the following advantages:

1. It gives the power of a large machine to allow tracing operations which are not feasible at the hardware level.
2. It may indicate prior to the time that the hardware is committed that more memory or more time is required to perform an operation which may dramatically change the hardware implementation to be committed.

In any case, attempting to bring up the microprocessor system without assemblers and an interface module such as TIM is not the most efficient use of the designer's time.

For the user who is just "starting out" with microprocessors, the KIM technique is acceptable because the length and complexibility of the programs to be written should be shorter, and the user can program directly in Hex and debug using KIM exclusively.

4.2.1 KIM -- Keyboard Input Monitor

KIM allows the user to key in Hex values into specified memory locations and to monitor results.

KIM is available to the system designer in several forms. In its simplest form, a single device of the R6530 type including 1024 bytes of preprogrammed ROM may be included as a component in an existing system. The array includes a monitor program which provides the following features:

1. Data input and output control from serial teletypewriters (ASR 33, Silent 700, etc.)
2. Data input and system control from a 22-key keyboard
3. Address and data display on a 6-digit, 7-segment type display

A microprocessor system designed to include the KIM array will allow the designer to perform the following operations:

1. The user may select keyboard (KB) or teletypewriter (TTY) mode for entry, display and control.
2. If in KB mode, the user may enter address or data fields from the keyboard. The user may display the contents of any address location in the system and can modify the contents of any address location (other than preprogrammed ROM locations). The step operation (STEP key) provides a convenient method for displaying the data contained in successive memory locations. Program execution may be authorized to begin from any selected starting address using the RUN key.
3. If in the TTY mode, the user may obtain a printing of the data at any memory location. He can modify the data contained in any memory location. Program listing from any start address to any end address may be authorized. Paper tapes may be loaded or generated automatically. Finally, program execution may be initiated from any selected starting address.
4. In either mode, the user terminates program execution by depressing the STOP key to return control of the system to the KIM program. Alternatively, a depression of the RST key causes a total reset of the system and a return of the system to KIM program control.

The KIM array is also available to the system designer as a part of a special design-in subsystem provided in the form of a printed-circuit card. Included on this card are the following functional elements:

1. R6502 microprocessor array
2. R6530-002 array (containing the KIM monitor program)
3. 22-key keyboard and mode-select switch
4. 6-digit, 7-segment LED display
5. 1024 x 8 RAM
6. R6530-003 array providing an interval timer, 16 I/O pins, and 64 bytes of RAM
7. All interface circuits for operation with serial teletypewriters

This subsystem provides the same operating features described earlier, but is supplied as an operating unit requiring the user to provide only the +5-volt power supply in order to commence operating. As a "stand-alone" subsystem, the KIM permits the user to enter and debug programs of up to 1024 steps and to control the action of up to 16 I/O pins.

For further details on physical and operating characteristics of the KIM array and subsystem, the reader is referred to the KIM manual supplied separately.

4.2.2 TIM -- Teletype Input Monitor

TIM is a pre-programmed R6530. Its application is to allow the user to interface to an ASCII device such as a Teletype, CRT, Execuport, etc. using the ASCII serial communication techniques to communicate to and from the the microprocessor. This effectively allows the user to load memory from the keyboard or from paper tape or cassetts which are attached to his device. By the addition of a single TTL package to the system, TIM can be configured so that it is the starting point for the microprocessor, but once the initialization has been accomplished it transfers itself out of the start-up memory, changes the rest of the microprocessor memory to normal configuration, and operates transparent to the microprocessor.

The proper time for using TIM to develop a microprocessor system is primarily after the system is determined to be wired correctly by the techniques already described. TIM is then utilized to debug the user's code by allowing the user to input prespecified values, execute portions of the code, and examine the results.

It should be noted that because I/O devices are extensions of memory debugging techniques are simplified. They can be configured to control I/O devices to test that lights can be lit, switches tested, motors started and stopped, etc. For instance, all of the connections to lights and switches can be checked from the teletype keyboard by writing into the I/O registers the appropriate code for turning on the lights. Correct operation of switches can be checked without the program running by putting the switches in either state and reading the I/O device result indicated to the programmer. This type of checking totally "shakes out" the I/O connections to make sure the I/O device is located in the correct memory address determines that the wiring to in the correct memory address, determines that the wiring to the I/O devices is correct, and checks on the microprocessor bus.

A rational technique for applying either TIM or KIM is to interconnect the device into the system to get the microprocessor to pass the single-step start-up sequence, and then to use the debugging capability of the TIM prior to executing any of the user's code to verify that all input/output connections are correct. In cases concerned with the stopping of the motors and other devices which require timing, the proper connection to the motors and other devices can be checked without the motor itself physically being checked by disconnecting leads, opening up connectors, and verifying with a 'scope or a meter that the microprocessor's influence at that point is as would be expected on a static basis. Therefore, this technique is recommended as the second step of a start-up sequence.

Significant details are given in the section on the use of restart or start sequence and a single cycle operation to verify the interconnection of most of the system. It should be recalled that the instructions were given independent of the coding that was available to the programmer.

The advantage of using the TIM or KIM in the start-up checkout is that there is known code which is guaranteed to be accurate that should be evoked during this start-up sequence. By referring to the coding of the ROM as it appears in the documentation on the TIM or KIM, the user can apply the known sequences from the TIM or KIM program to verify the start-up sequence, thereby removing one more variable. Therefore, all initial system checkout should be accomplished using TIM or KIM program first in the start-up

sequence to make sure that the interconnection to TIM and to memory are correct. Then, once the basic operation of TIM has been verified, there is a known sequence which TIM will follow dynamically that allows the user to verify that TIM is operational. The user should then verify the remainder of the memory and I/O connections by writing and reading in the memory locations using the debugging feature of the TIM or KIM. This procedure verifies the connection and operation of each of the chips of the system as well as all the interconnections to all outboard devices.

Now the problem is truly reduced to making sure that the programmer's code is correct and the user's program can be loaded by means of either through-the-keyboard or through-the-auxiliary devices.

The program can be debugged as a program rather, with no concern as to whether or not the problem is one of hardware or software. By definition other than incorrect timing to I/O devices, the problem is reduced to one of programming mistakes.

For a more detailed discussion on the programming on TIM, the user is referred to the TIM manual supplied separately.

4.3 MICROPROCESSOR START-UP PROCEDURE

This section attempts to tie together all of the techniques previously discussed into one ordered procedure. This procedure is based on experience gained in bringing up systems using processors from several different manufacturers. While it is certainly true that no single procedure can be expected to catch all of the software and hardware errors that can exist in microcomputer systems, it is hoped that this step-by-step approach will allow the designer to bring up his system with an absolute minimum of difficulty.

This procedure assumes the existence of Single Cycle and/or Single Instruction logic. Any of the System Development tools discussed in Section 4.2 will assist the user in bringing up his system. These devices allow convenient entry of test programs as well as modification of the system program and data.

Each step in the procedure includes the following information:

- Section of the system hardware/software to be checked.
- Hardware, test equipment, etc. required to perform the test.
- Action to be taken in implementing the test.
- Expected results.
- Suggested procedures for analyzing failure modes.

It cannot be emphasized too strongly that one must utilize a very methodical, step-by-step procedure aimed at solving a single problem at a time within the system. It is very easy for several problems to amplify each other to such an extent that nothing within the system seems to be operating properly. Correcting problems one-at-a-time will ultimately yield a complete working system with minimum frustration.

4.3.1 System Power

It is generally recommended that first prototypes of microcomputer systems be built using sockets for the ICs (processor, memories, etc.). One distinct advantage of this technique is that it permits the designer to verify that V_{CC} and V_{SS} are properly connected to each socket before the chips are inserted. The V_{CC} line should be within the tolerances specified about the 5 volt nominal relative to V_{SS}. This basic first step can help avoid power supply connections which may be fatal to the chips in the system.

After checking power connections with a voltmeter or oscilloscope, to insert the processor into its socket and verify that the additional current drain is within specifications for this device.

Before inserting the other devices, examine the address lines, SYNC line and the output clocks to make sure that the processor is generating signals. The address lines should be incrementing and the sync line should be generating regular, positive going pulses. The $\overline{RES}$ line and the RDY line should be high (> +2.4V) for this test.

If the processor appears to be operating and power consumption is reasonable, the rest of the devices in the system can be inserted into their sockets.

4.3.2 Basic System Timing

Before one can expect a microprocessor system to function, proper operation of the basic system timing signals (ϕ_1, ϕ_2, etc.) must be verified. The most important of these signals is the system clock.

In the 6502, both phases (ϕ_1 and ϕ_2) are available for driving the rest of the system. In this system it is necessary to check the clock timing very carefully to assure that the timing of the clock signals within the processor is compatible with that used on the support chips.

Using an oscilloscope, compare the ϕ_1 input clock and the ϕ_2 clock presented to the support chips to verify that the delay due to clock buffering does not exceed the allowable maximum.

4.3.3 System Reset

Static and dynamic analysis of the Reset function can provide very detailed information on how the system is operating. In fact, it is this step which will verify the operation of most of the basic system hardware. The tools required are:

- Single-Cycle/Single-Instruction Logic
- Oscilloscope
- Signal Generator (for Driving RESET)

STATIC ANALYSIS OF SYSTEM DETAILS

Depress the HALT button and then the manual RESET switch: then push the single-cycle switch six times. This will step the processor through the first part of the BRK sequence and into the RESET vector fetch. At this time the processor should be generating FFFC on the address bus and the ROM should have put the low-order byte of the RESET vector onto the data bus in response to this address. This is an excellent time to check the following very basic items:

1. Address Lines

   Using the oscilloscope, verify that the logic levels on the address lines are proper and that they are reflected properly through any bus expanders onto the memory and peripheral chips. This is a very common circuit fault.

2. ROM/PROM Chip Select

   Using the oscilloscope, verify that the address FFFC does select the ROM which contains the low-order byte of the RESET vector.

3. Data Bus

   Using the oscilloscope, verify that the voltages on the data bus pins of the processor are proper. It is important for these signals to be analyzed at the processor to ensure proper operation of any bidirectional bus expanders in the system. In this test, the most common indication of improper operation of the data bus expanders is "floating" processor data bus pins, i.e., the processor data bus pins are being driven neither high nor low because the bus expanders are in the open-circuit condition or are reversed.

4. Miscellaneous Processor Pins

   Using the oscilloscope, briefly examine the other processor pins (R/W, $\overline{IRQ}$, $\overline{NMI}$, etc.) to assure that there are no voltage level problems detectable at this point. Both of the interrupt inputs and the R/W output should be high. Examine the R/W signal on the input to the memory and peripheral devices.

After these initial tests have been performed, it should be possible to press the single-step switch once more to fetch the high-order byte of the interrupt vector from address FFFD. On the next actuation of the single-cycle switch, the processor address bus should contain the RESET vector which was fetched from memory.

At this point, the processor is ready to execute the system initialization routine. During initialization, it can be expected that program memory will be accessed, peripheral registers will be loaded, and internal processor registers will be cleared or set to a starting value. It is extremely useful to execute this routine one instruction at a time to determine that each time program memory is accessed, the proper instruction is returned. However, unless a data trap is provided, it will be more meaningful to utilize dynamic analysis techniques to analyze the operation of peripheral devices, since most peripheral accesses will be for the purpose of writing either the I/O control or the control registers in the peripheral devices.

DYNAMIC ANALYSIS OF SYSTEM DETAILS

The general technique of dynamic analysis is discussed in Section 4.1.2. The discussion which follows will apply this technique to analyze many of the details of the system operation.

Set up the system as described in Section 4.1.2. After the test equipment is operating properly, most of the system operation can be verified using only the oscilloscope.

ADDRESS BUS VERIFICATION

The first item which must be checked is the specific timing of the address lines. These lines will change during the first part of $\emptyset_1$ but after the specified period, they should stabilize and remain stable through the rest of the cycle. Figure 4-6a shows the waveform which one should expect to see while examining $\emptyset_1$, $\emptyset_2$ and two address lines. In this illustration, one address line is going high and the other is going low. These lines are being generated within the processor and are guaranteed to operate properly provided the total loading on the pins is within specifications. The most common cause of both voltage-level and rise-time problems is overloading. Voltage-level problems are commonly evidenced by the "zero" level being too high, i.e., the address buffer is being asked to sink too much current. Excess capacitance is usually evidenced by the rise and fall times being too long (Figure 4-6b).

In examining the address lines, it is important that the data be examined on the processor and directly on the various support chips. This will assure that any bus expanders in the system are operating properly and that the addresses are valid where they are actually being used.

DATA BUS VERIFICATION

After the addresses have been verified, the next step is to examine the data bus to verify the validity of data being transferred both from the processor to the support chips and from the support chips back into the processor.

Figure 4-7 illustrates the waveform which one can expect to see on the data bus lines. It is very important to note that during $\emptyset_1$ there is no way to predict the voltage on the data bus, since neither the processor nor the support chips are driving these lines. However, during $\emptyset_2$ the data bus pins should go either high or low. It is only during $\emptyset_2$ (high) that the validity of the data can be verified.

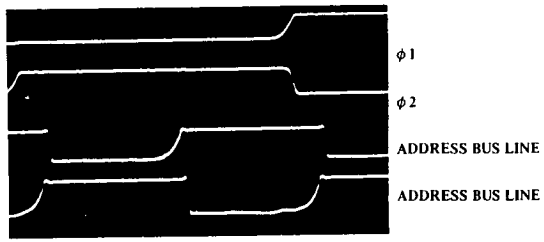


FIGURE 4-6a – Proper Address Lines

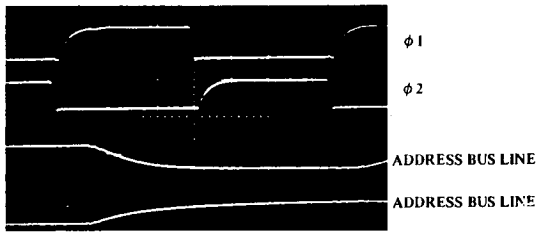
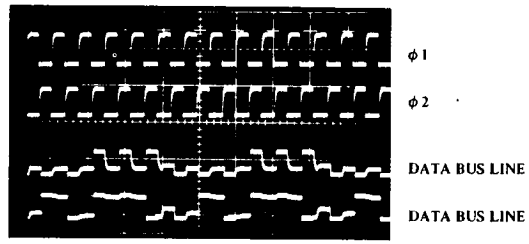
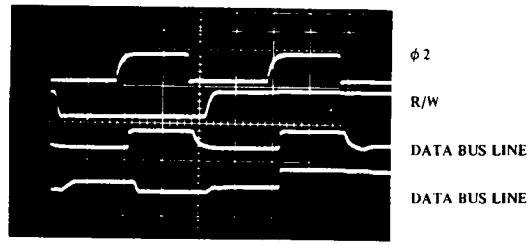


FIGURE 4-6b – Excess Address Line Loading

Address Lines In R650X Systems
FIGURE 4-6





The Data Bus in R650X Systems
FIGURE 4-7

Three very important parameters must be considered when examining the data bus. These are the voltage levels, the time at which the data is valid and the delay from the trailing edge of $\emptyset_2$ to data becoming invalid.

1. Voltage Levels

   The logic levels on the processor data bus must always be greater than 2.4 volts for a logic 1 and less than 0.4 volts for a logic 0. This is a very basic concept, but a quick check on these levels very early in the checkout procedure can help the designer avoid hours of attempting to make a system operate with signals which are actually marginal but which on the surface appear to be satisfactory.

   Another very important item to check is whether or not the logic "0" voltage is actually going negative (below GND). It is very important that the logic signals going into all the chip inputs not be allowed to go below -0.3 volt as indicated in the specifications.

2. Data Valid Time

   The time at which data becomes valid indicates the total time which the processor or memory has available to stabilize the gates and latches used to trap the data within the chip. For this reason the data must not take too long to reach either a valid high "1" or a valid low "0." The primary cause of slow signals on the data bus is excessive loading, either resistive or capacitive. Carefully check the devices which are attached to the bus to make sure that the total loading is within specifications.

3. Hold Time

   The last important consideration,"hold time," is defined as the time between the trailing edge of the $\emptyset_2$ pulse and the point at which data is no longer valid. A minimum of 10 ns hold time is required for the processor to trap the data into its internal input latches. The processor internal $\emptyset_2$ pulse is used to gate the contents of the data bus into these latches. Hold time is also required by the various support chips within the system. Carefully check the

signals as they appear on the RAMs, ROMs, etc. to verify that each is being operated in accordance with its specification.

4.3.4 Detailed Component Check

After the dynamic check of the reset routine, the next step is to attempt to run the system program. The success of this operation will determine whether or not a further detailed component check is necessary. It is important to note that the checkout of the system program should proceed one step at a time in much the same manner as we have approached the hardware checkout. If a careful examination has been made of all of the devices, data paths, etc. in the system, the software checkout can proceed under the assumption that the hardware is fully operational. However, it is inevitable that doubts will arise. There are times in the software checkout process that the program will appear to be incorrect ... data will not be going into memory as it should ... or, in general, some hardware failure will be indicated. As soon as this happens, the suspected components should be examined in detail. In keeping with the policy of "one step or one problem at a time," it is important that potential hardware problems not be allowed to invalidate the effort being put into the software checkout.

Component problems can be one of two types: component failure, i.e., a part not operating per specifications; or system failure, i.e., a part being used wrong in the system. The latter problem can be a result of incorrect system design or incorrect wiring. The problem of functional components not operating properly in the system is the one which will be addressed here. In fact, if there is any doubt about a component being functional, it should be replaced immediately upon verification of proper signals to all inputs. If it still does not operate properly, the problem is most likely system-related.

The detailed component check is performed most effectively by loading a small looping program into the system RAM. For this reason, the TIM or KIM debug software (see TIM and KIM Manuals) can be of significant value in this process. The procedure involves static and dynamic operation of a small test program which exercises each of the components in the system. The goal of this step should be a complete verification that all chip selects are operating properly, that all data address lines are operating properly, and that the support chips are driving the processor properly.

The suggested procedures for checkout of each type of component are discussed below.

1. ROMs (PROMs)

The most straightforward component in any microprocessor system is the ROM. This device simply puts out an 8-bit word onto the the data bus in response to an address. Difficulty with ROMs is usually caused by improper chip selects or by misapplication of devices which are not part of the R6500 family. For this reason, static testing of ROMs is usually a very effective first step. This requires entering a test program into RAM and executing this program using the single-cycle switch. The program itself should simply perform a READ (for example, an LDA or LDX instruction) of a selected word for each ROM chip to be tested. The chip selects can then be examined and, at the same time, the address lines presented to the chips can be examined along with the data put on the data bus.

After the chip select, address bus and data bus have been verified statically, it may be necessary to execute the same test program dynamically to assure that all chips in the system are operating at system speed. At this point, it may be necessary to include a WRITE operation (STA, STX, STY, etc.) in the loop to provide a sync signal.

Analysis of the dynamic operation of the ROMs should involve first looking at each address and data bus lines directly on the processor chip. It is here that the address is being generated, and it is here that the data must meet a speed specification. If data are not valid at the proper time, the next step is to determine where excessive delay has been introduced into the data path from address output, through the ROM and back to the processor data bus. It should be kept in mind that it is this entire path which must operate at speed to assure proper processor operation. In fact, if the delays are excessive, it may be necessary to slow down the system clock rate to allow the program data to reach the processor in time. An alternative solution to this problem is the implementation of the RDY signal to hold the processor for one cycle each time it fetches data or program from the ROMs.

Although the problems discussed above may be encountered at this point, it is much more likely that a wiring error will cause a single address or data line to be excessively loaded so that it operates slowly or not at all. This problem can usually be detected and fixed quite easily by looking at each component in the data path.

2. RAMs

Operation of the RAMs in a microprocessor system can be checked in much the same manner as the ROMs. Execution of a test loop program both statically and dynamically for each chip in the system should be sufficient to verify proper operation of the RAMs in the system. For each RAM, both a WRITE and a READ operation should be included in the test loop. This will allow checkout of data transfers in both directions.

During single-cycle execution of the test loop, the processor will stop only in the RAM read operations. However, this will allow a static check of the chip select logic and the address and data lines. Running the program dynamically will allow verification that the data and address signals presented to the RAMs during the WRITE operation are within specification for the RAM being used in the system, and that the total delays through the address, RAM, and data bus path are within specifications for the processor during the READ operations. As with the ROMs, the most likely problem to be encountered at this point is concerned with wiring errors which cause a specific device to operate improperly. A careful check of each pin will enable detection of this type of problem.

3. PIAs

The peripheral interface devices (6520, 6530, etc.) can all be checked out in the manner described above. However, since these chips do many different operations, the test program must be much more complex than that required for the ROM and RAM.

However, it can usually be limited to testing only those functions which are used in the system.

A large part of the operation of the peripheral interface devices can be verified by performing a WRITE followed by a READ for each register on the chip. This will permit a complete checkout to be made of the data paths between the processor and the chips, as well as of all the chip select functions. However, a more complete analysis may be required to verify that data are appearing properly on the output pins of the peripheral chip, and that data on the inputs are being reflected properly back into the processor. This will involve disconnecting the peripheral subsystem which the processor is attempting to drive, and manually putting data into the inputs. A separate test can verify the validity of output data.

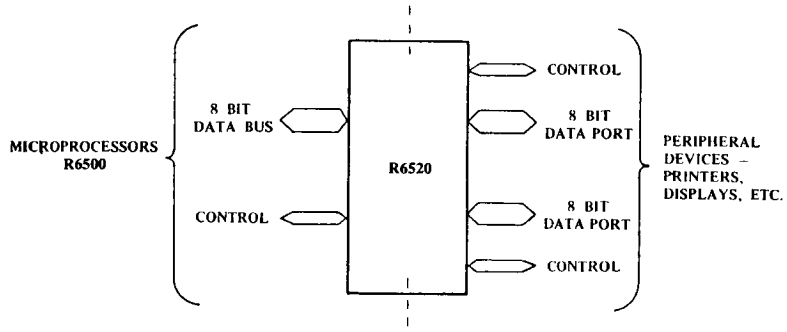
After the system hardware has been examined to the detailed degree discussed above, the designer will have developed confidence that his system can operate properly once the system program is completely debugged. Verification of the system program should proceed with a section-by-section checkout as discussed above. Each subroutine, interrupt routine, etc. should be examined separately. Subroutines, routines, etc. can then be combined to form the major peripheral operating routines, arithmetic routines, etc. which make up the system program. The final result should be a functioning program that has been examined in all of its details, running on a system which is fully operational.

SECTION 5

R6520 PERIPHERAL INTERFACE ADAPTER (PIA)

The R6520 is a direct pin-for-pin replacement for the Motorola M6820 Peripheral Interface Adapter, the "PIA." As such, it meets all of the "PIA" electrical specifications and is totally hardware-compatible with the M6820.

The R6520 is an I/O device which acts as an interface between the microprocessor and peripherals such as printers, displays, keyboards, etc. The prime function of the R6520 is to respond to stimuli from each of the two worlds it is serving. On the one side, the R6520 is interfacing with peripherals via two 8-bit bidirectional peripheral data ports. On the other side, the device interfaces with the microprocessor through an 8-bit data bus (this is the same data bus discussed at length in Section 1.2.2). It is, therefore, simplest to view the basic function of the R6520 as illustrated in the block diagram of Figure 5-1.



Basic R6520 Interface Diagram
FIGURE 5-1

In addition to the lines described above, the R6520 provides four interrupt input/peripheral control lines and the logic necessary for simple, effective control of peripheral interrupts. No external logic is required for interfacing the R650X microprocessor to most peripheral devices. Figure 5-2 shows the R6520 pinout designations for the Peripheral Interface Adaptor.

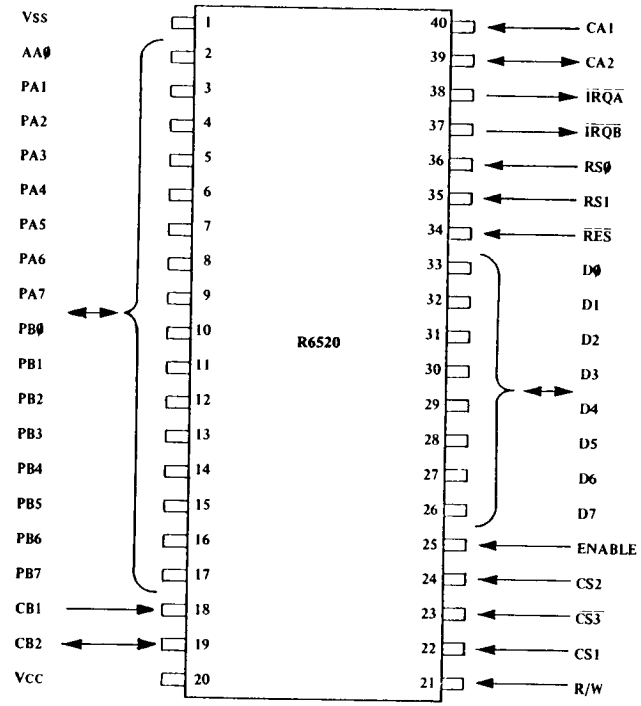
The functional configuration of the R6520 is programmed by the microprocessor during systems initialization. Each of the peripheral data lines is programmed to act as an input or output, and each of the four control/interrupt lines may be programmed for one of four possible control modes. This allows a high degree of flexibility in the overall operation of the interface.

Some of the more important features of the R6520 are the following:

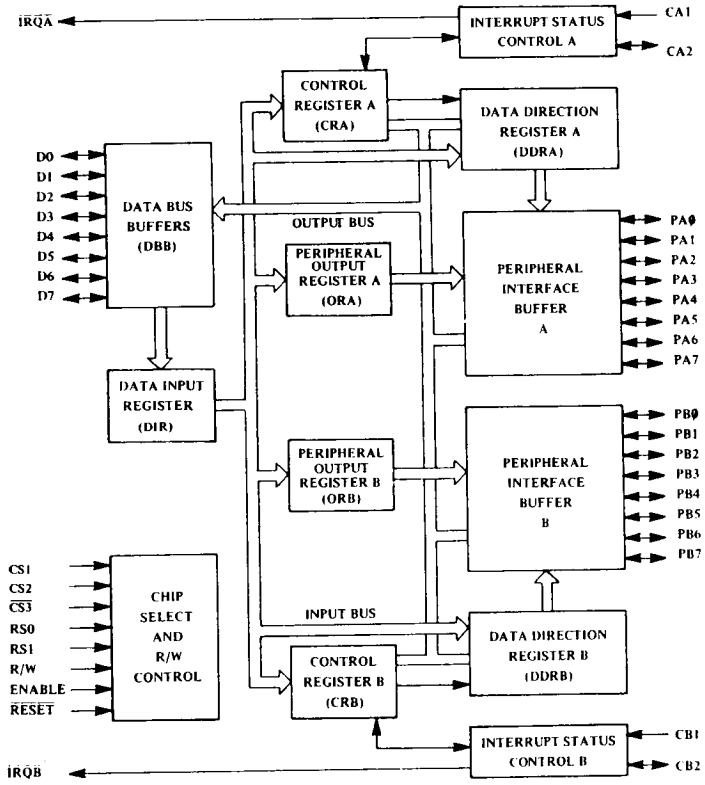
- Compatibility with the R6500 microprocessors (CPUs).
- Eight-bit bidirectional data bus for communication with the microprocessor.
- Two 8-bit bidirectional ports for interface to peripherals.
- Two programmable control registers.
- Two programmable Data Direction Registers.
- Four individually controlled interrupt input lines -- two usable as peripheral control outputs.
- Handshake control logic for input and output peripheral operation.
- High-impedance three-state and direct transistor drive peripheral lines.
- Program-controlled interrupt and interrupt mask capability.

5.1 R6520 ORGANIZATION

Figure 5-3 contains a block diagram of the R6520 showing the internal registers and data paths and the various inputs and outputs on the device. This section contains a general description of the internal organization of the device, along with a discussion of how the various registers affect one another. The following sections discuss the details



R6520 Pinout Designations, Peripheral Interface Adaptor
FIGURE 5-2



R6520 Internal Architecture

FIGURE 5-3

of the inputs and outputs on the chip, along with a detailed discussion of the operation of each register. The final section discusses the R6520 from an operational viewpoint, describing the interaction of the register bits, input/output lines, etc.

The R6520 is organized into two independent sections referred to as the "A Side" and the "B Side." Each section consists of a Control Register (CRA, CRB), Data Direction Register (DDRA, DDRB), Output Register (ORA, ORB), Interrupt Status Control and the buffer necessary to drive the Peripheral Interface busses.

5.1.1 Data Input Register

When the microprocessor writes data into the R6520, the data which appear on the data bus during the Phase 2 clock pulse is latched into the Data Input Register. It is then transferred into one of six internal registers of the R6520 after the trailing edge of Phase 2. This assures that the data on the peripheral output lines will not "glitch" -- i.e., the output lines will make smooth transitions from high to low or from low to high, and the voltage will remain stable except when it is going to the opposite polarity.

5.1.2 Control Registers (CRA and CRB)

The Control Registers allow the microprocessor to control the operation of the interrupt lines (CA1, CA2, CB1, CB2), and peripheral control lines (CA2, CB2). A single bit in each register controls the addressing of the Data Direction Registers (DDRA, DDRB) and the Output Registers (ORA, ORB) discussed below. In addition, two bits (bit 6 and 7) are provided in each control register to indicate the status of the interrupt input lines (CA1, CA2, CB1, CB2). These interrupt status bits ($\overline{IRQA}$, $\overline{IRQB}$) are normally interrogated by the microprocessor during the interrupt service program to determine the source of an active interrupt. These are the interrupt lines which drive the interrupt input ($\overline{IRQ}$, $\overline{NMI}$) of the microprocessor. The other bits in CRA and CRB are described in the discussion of the interface to the peripheral device (Section 1.5.4).

The various bits in the control registers will be accessed many times during a program to allow the processor to enable or disable interrupts, change operating modes, etc. as required by the peripheral device being controlled.

5.1.3 Data Direction Registers (DDRA, DDRB)

The Data Direction Registers allow the processor to program each line in the 8-bit Peripheral I/O port to act as either an input or an output. Each bit in DDRA controls the corresponding line in the Peripheral A port, and each bit in DDRB controls the corresponding line in the Peripheral B port. Placing a "0" in the Data Direction Register causes the corresponding Peripheral I/O line to act as an input, while a "1" causes it to act as an output.

The Data Direction Registers are normally programmed only during the system initialization routine which is performed in response to a Reset signal; however, the contents of these registers can be altered during system operation. This allows very convenient control of some peripheral devices such as keyboards.

5.1.4 Peripheral Output Registers (ORA, ORB)

The Peripheral Output Registers store the output data which appear on the Peripheral I/O port. Writing an "0" into a bit in ORA causes the corresponding line on the Peripheral A port to go low (< 0.4V) if that line is programmed to act as an output. A "1" causes the corresponding output to go high. The lines of the Peripheral B port are controlled by ORB in the same manner.

Addressing of these registers is discussed in Section 5.2.4.

5.1.5 Interrupt Status Control

The four interrupt/peripheral control lines (CA1, CA2, CB1, CB2) are controlled by the Interrupt Status Control (A, B). This logic interprets the contents of the corresponding Control Register, detects active transitions on the interrupt inputs and performs those operations necessary to assure proper operation of these four peripheral interface lines. The operation of these lines is described in detail in Section 5.3.2

5.1.6 Peripheral Interface Buffers (A, B) and Data Bus Buffers (DBB)

The Buffers which drive the peripheral I/O ports and the data bus provide the current and voltage drive necessary to ensure proper system operation and to meet the device specifications.

5.2 PROCESSOR INTERFACE

The R6520 interfaces to the microprocessor with an 8-bit bidirectional data bus, 3 chip-select lines, 2 register-select lines, 2 interrupt request lines, read/write line, enable line, and reset line.

5.2.1 Data Bus (DO-D7)

The 8-bit, bidirectional data bus allows the transfer of data between the microprocessor and the R6520. The data bus output drivers are 3-state devices that remain in the high impedance state except when the microprocessor reads data from the peripheral adapter. This data bus is the same as discussed in Section 1.2.2, "Bus Structure."

5.2.2 Enable (E)

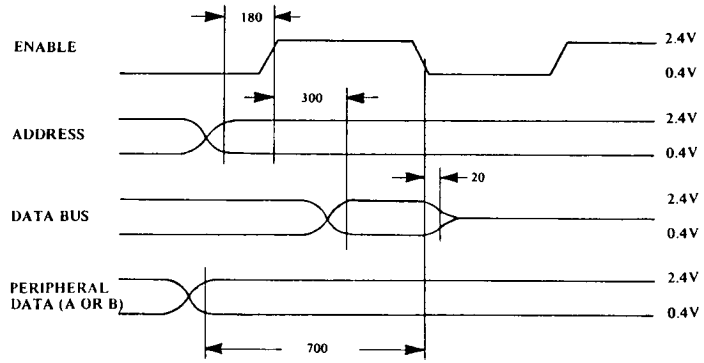
The Enable input is the only microprocessor interface timing input on the peripheral interface device. All data transfers into and out of the R6520 are controlled by this signal. In normal operation, this input should be connected to the phase two clock signal. In the case of the R6512 through R6515, this is the Phase 2 clock generated externally to the microprocessor chip. For on-chip oscillator products the enable pulse becomes Ø2(OUT).

5.2.3 Read/Write (R/W)

This signal is generated by the microprocessor to control the direction of data transfers on the data bus. A "low" (< 0.4V) on this line enables the input buffers (microprocessor Write), and data are transferred from the microprocessor to the R6520 under control of Enable input if the device has been chip-selected. A "high" on the R/W line allows the R6520 to transfer data to the data bus buffers. The data bus buffers are enabled when the proper chip-select and Enable signals are present. Figure 1.23 illustrates the Read/Write timing.

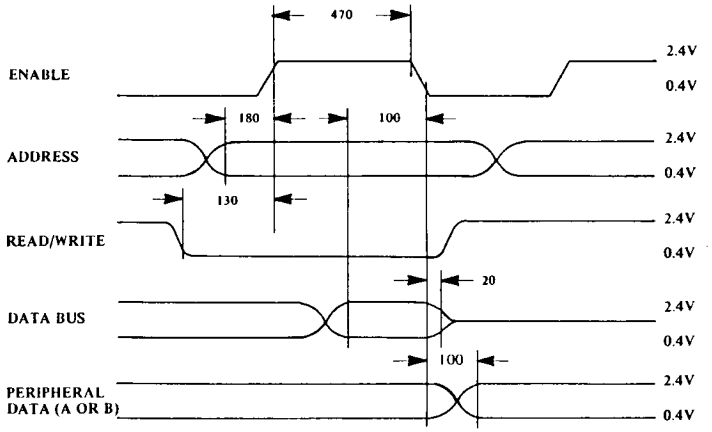
5.2.4 Chip Select Lines (CSO, CS1, CS2)

These three inputs allow the microprocessor to select the proper peripheral interface device. CSO and CS1 must be high and CS2 must be low for selection of the device. Data transfers are then performed under control of the Enable and R/W signals. These lines are normally connected to the address lines on the microprocessor, either directly or through address decoders.



Microprocessor Interface Timing - Read

FIGURE 5-4 a



***NOTE: ALL TIMES SPECIFIED ARE IN nSEC FOR 1MHZ OPERATION.**

Microprocessor Interface Timing - Write

FIGURE 5-4 b

Microprocessor Interface Timing

FIGURE 5-4

As described in Section 5.4.2, a single bit in each Control Register (CRA and CRB) controls access to the Data Direction Register or the Peripheral interface. If bit 2 in the Control Register is a "1," a Peripheral Output register (ORA, ORB) is selected, and if bit 2 is a "0," the Data Direction Register is selected. Internal registers are selected by the Register Select lines (RSØ, RS1) and the Data Direction Register Access Control bit as follows:

RS1	RSØ	Data Direction Register Access Control Bit CRA-2	CRB-2	Register Selected
Ø	Ø	1	-	Peripheral Interface A (See Section 5.2.5)
Ø	Ø	Ø	-	Data Direction Register A
Ø	1	-	-	Control Register A
1	Ø	-	1	Peripheral Interface B (See Section 5.2.5)
1	Ø	-	Ø	Data Direction Register B
1	1	-	-	Control Register B

If the programmer wishes to write the data into DDRA, ORA, DDRB, or ORB, he must first set bit 2 in the proper Control Register. The desired register can then be accessed with the address determined by the address interconnect technique used.

5.2.5 Register Select Lines (RS0), (RS1)

These two register select lines are used to select the various registers inside the R6520. These input lines are used in conjunction with internal control registers to select a particular register that is to be accessed by the microprocessor. These lines are normally connected to microprocessor address output lines. These lines operate in conjunction with the chip-select inputs to allow the microprocessor to address a single 8-bit register within the microprocessor address space. This register may be an internal register (CRA, ORA, etc.) or it may be a Peripheral I/O port.

The processor can write directly into the Control Registers (CRA, CRB), the Data Direction Registers (DDRA, DDRB) and the Peripheral Output Registers (ORA, ORB). In addition, the processor can directly read the

contents of the Control Registers and the Data Direction Registers. Accessing the Peripheral Output Register for the purpose of reading data back into the processor operates differently on the ORA and the ORB registers, and the two procedures are discussed separately below.

READING THE PERIPHERAL A I/O PORT

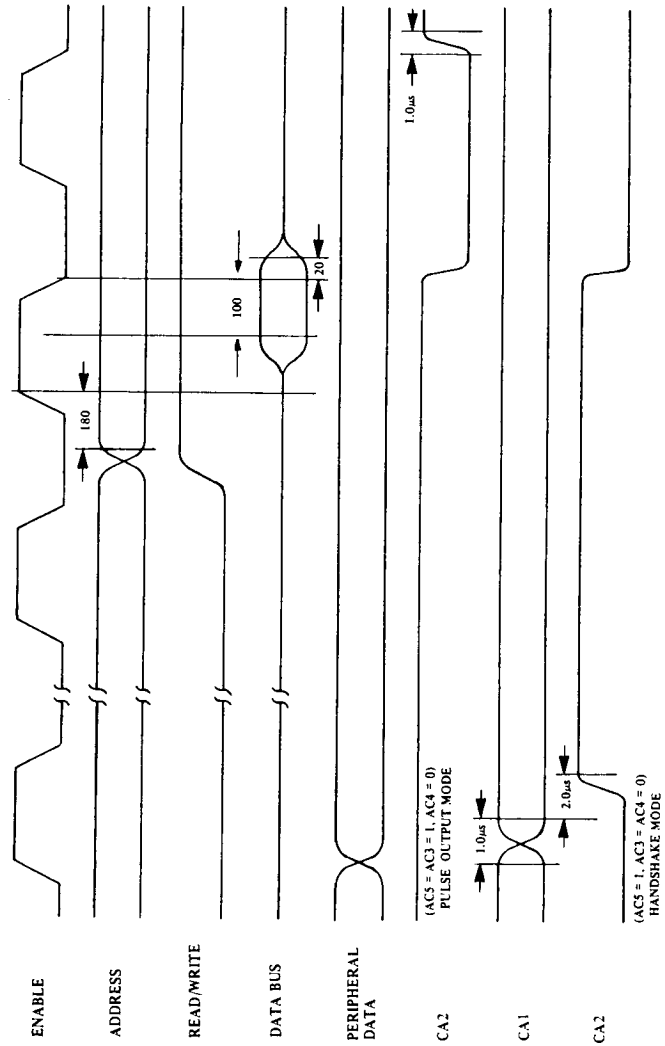
The Peripheral A I/O port consists of 8 lines which can be programmed to act as inputs or outputs. When programmed to act as outputs, each line reflects the contents of the corresponding bit in the Peripheral Output Register. When programmed to act as an input, these lines will go high or low depending on the input data. The Peripheral Output Register (ORA) has no effect on those lines programmed to act as inputs. The eight lines of the Peripheral A I/O port therefore contain either input or output data depending on whether the line is programmed to act as an input or an output. Figure 5-5 illustrates the interface timing.

Performing a Read operation with RS1 = 0, RS0 = 0 and the Data Direction Register Access Control bit (CRA-2) = 1, directly transfers the data on the Peripheral A I/O lines into the processor (via the data bus). This will contain both the input and output data. The processor must be programmed to recognize and interpret only those bits which are important to the particular peripheral operation being performed.

Since the processor always reads the Peripheral A I/O port pins instead of the actual Peripheral Output Register (ORA), it is possible for the data read into the processor to differ from the contents of the Peripheral Output Register for an output line. This is true when the I/O pin is not allowed to go to a full +2.4V DC when the Peripheral Output register contains a logic 1. In this case, the processor will read a zero from the Peripheral A pin, even though the corresponding bit in the Peripheral Output register is a 1.

READING THE PERIPHERAL B I/O PORT

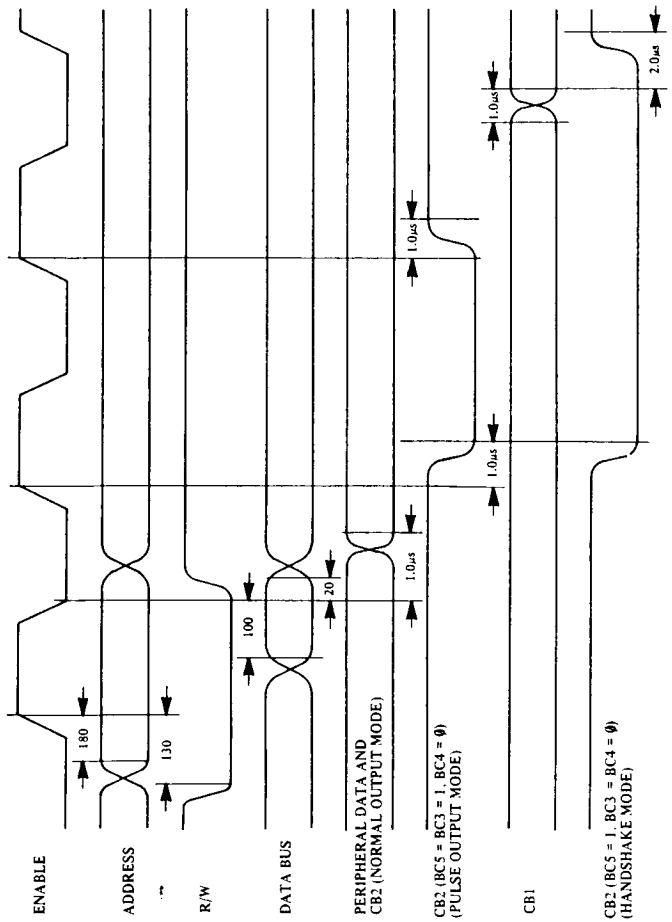
Reading the Peripheral B I/O port yields a combination of input and output data in a manner similar to the Peripheral A port. However, data are read directly from the Peripheral B Output Register (ORB) for those lines programmed to act as outputs. It is, therefore, possible to load down the Peripheral B Output lines without causing incorrect data to be transferred back into the processor on a Read operation. Figure 5-6 illustrates the timing.



NOTE: ALL TIMES SPECIFIED ARE IN nSEC FOR 1MHZ OPERATION.

Peripheral A Interface Timing

FIGURE 5-5



Peripheral B Interface Timing
FIGURE 5-6

The details of the Peripheral A and Peripheral B ports will be discussed in the next section under the discussion of the interface between the R6520 and the Peripheral Devices.

5.2.6 Reset ($\overline{RES}$)

The active low Reset line resets the contents of all R6520 registers to a logic zero. This line can be used as a power-on reset or as a master reset during system operation.

5.2.7 Interrupt Request Line ($\overline{IRQD}$, $\overline{IRQB}$)

The active low Interrupt Request lines ($\overline{IRQA}$ and $\overline{IRQB}$) act to interrupt the microprocessor either directly or through external interrupt priority circuitry. These lines are "open source" (no load device on the chip) and are capable of sinking 1.6 milliamps from an external source. This permits all interrupt request lines to be tied together in a "wired-OR" configuration. The "A" and "B" in the titles of these lines correspond to the "A" peripheral port and the "B" peripheral port. Hence each interrupt request line services one peripheral data port.

Each Interrupt Request line has two interrupt flag bits which can cause the Interrupt Request line to go low. These flags are bits 6 and 7 in the two Control Registers. These flags act as the link between the peripheral interrupt signals and the microprocessor interrupt inputs. Each flag has a corresponding interrupt disable bit which allows the processor to enable or disable the interrupt from each of the four interrupt inputs (CA1, CA2, CB1, CB2).

The four interrupt flags are set by active transitions of the signal on the interrupt input (CA1, CA2, CB1, CB2). Controlling this active transition is discussed in the next section under the discussion of the interface between the R6520 and the peripheral device.

CONTROL OF $\overline{IRQA}$

Control Register A bit 7 is always set by an active transition of the CA1 interrupt input signal. Interrupting from this flag can be disabled by setting bit 0 in the Control Register A (CRA) to a logic 0. Similarly, Control Register A bit 6 can be set by an active transition of the CA2 interrupt input signal. Interrupting from this flag can be disabled by setting bit 3 in the Control Register to a logic 0.

Both bit 6 and bit 7 in CRA are reset by a "Read Peripheral Output Register A" operation. This is defined as an operation in which the proper chip-select and register-select signals are provided to allow the processor to read the Peripheral A I/O port.

CONTROL OF $\overline{\text{IRQB}}$

Control of $\overline{\text{IRQB}}$ is performed in exactly the same manner as that described above for $\overline{\text{IRQA}}$. Bit 7 in CRB is set by an active transition on CB1; interrupting from this flag is controlled by CRB bit 0. Likewise, bit 6 in CRB is set by an active transition on CB2; interrupting from this flag is controlled by CRB bit 3.

Also, both bit 6 and bit 7 are reset by a "Read Peripheral B Output Register" operation.

SUMMARY:

IRQA goes low when CRA-7 = 1 and CRA-0 = 1 or when CRA-6 = 1 and CRA-3 = 1.

IRQB goes low when CRB-7 = 1 and CRB-0 = 1 or when CRB-6 = 1 and CRB-3 = 1.

The use of these interrupt flags and interrupt disable bits is discussed in more detail in Section 5.3.

It should be stressed at this point that the flags act as the link between the peripheral interrupt signals and the processor interrupt inputs. The interrupt disable bits allow the processor to control the interrupt function.

5.3 PERIPHERAL INTERFACE

The R6520 provides two 8-bit bidirectional ports and four interrupt/control lines for interfacing to peripheral devices. These ports and the associated interrupt/control lines are referred to as the "A" side and the "B" side. Each side has its own unique characteristics and will be discussed separately below.

5.3.1 Peripheral I/O Ports

The Peripheral A and Peripheral B I/O ports allow the microprocessor to interface to the input lines on the peripheral device by loading data into the Peripheral Output Register. They also allow the processor to interface with the peripheral device output lines by reading the data on

the Peripheral Port input lines directly onto the data bus and into the internal registers of the processor.

PERIPHERAL A I/O PORT (PAØ-PA7)

As discussed in Section 5.1.3 each of the Peripheral I/O lines can be programmed to act as an input or an output. This is accomplished by setting a "1" in the corresponding bit in the Data Direction Register for those lines which are to act as outputs. A "0" in a bit of the Data Direction Register causes the corresponding Peripheral I/O lines to act as an input.

The buffers which drive the Peripheral A I/O lines contain "passive" pull-ups as shown in Figure 5-7a. These pull-up devices are resistive in nature and therefore allow the output voltage to go to Vdd for a logic 1. The switches can sink a full 1.6 ma, making these buffers capable of driving one standard TTL load.

In the input mode, the pull-up devices shown in Figure 5-7a are still connected to the I/O pin and still supply current to this pin. For this reason, these lines represent one standard TTL load in the input mode.

PERIPHERAL B I/O PORT (PBØ-PB7)

The Peripheral B I/O port duplicates many of the functions of the Peripheral A port. The process of programming these lines to act as an input or an output has been discussed previously. Also, the effect of reading or writing this port has been discussed. However, there are several characteristics of the buffers driving these lines which affect their use in peripheral interfacing. These will be discussed below.

The Peripheral B I/O port buffers are push-pull devices as shown in Figure 5-7b. The pull-up devices are switched "OFF" in the "0" state and "ON" for a logic 1. Since these pull-ups are active devices, the logic "1" voltage is not guaranteed to go higher than +2.4V. They are TTL compatible but are not CMOS compatible.

However, the active pull-up devices can sink up to 1 ma at 1.5V. This current drive capability is provided to allow direct connection to Darlington transistor switches. This permits very simple control of relays, lamps, etc.

Because these outputs are designed to drive transistors directly, the output data is read directly from the Peripheral Output Register for those lines programmed to act as inputs.

The final characteristic which is a function of the Peripheral B push-pull buffers is the high-impedance input state. When the Peripheral B I/O lines are programmed to act as inputs, the output buffer enters the high-impedance state. These inputs will then have an impedance of greater than 1 megohm.

5.3.2 Interrupt Input/Peripheral Control Lines (CA1, CA2, CB1, CB2)

The four interrupt input/peripheral control lines provide a number of special peripheral control functions. These lines greatly enhance the power of the two general purpose interface ports (PAØ-PA7, PBØ-PB7).

PERIPHERAL A INTERRUPT INPUT/PERIPHERAL CONTROL LINES (CA1, CA2)

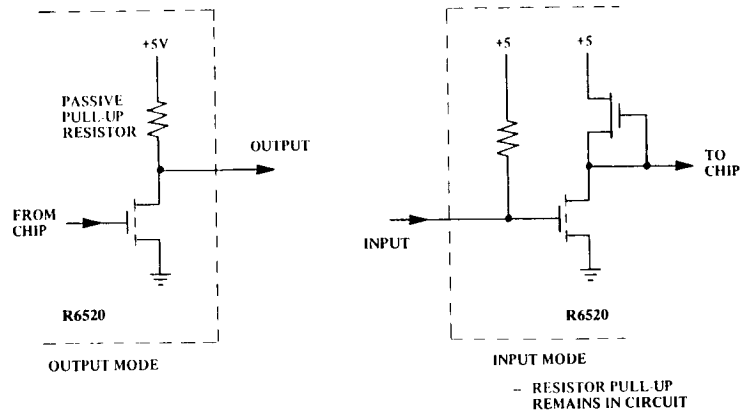
CA1 is an interrupt input only. An active transition of the signal on this input will set bit 7 of Control Register A to a logic 1. The active transition can be programmed by the microprocessor by setting a "Ø" in bit 1 of the CRA if the interrupt flag (bit 7 of CRA) is to be set on a negative transition of the CA1 signal or a "1" if it is to be set on a positive transition. Note: A negative transition is defined as a transition from a high (> 2.4V) to a low (< 0.4V), and a positive transition is defined as a transition from a low to a high voltage.

Setting the interrupt flag will interrupt the processor through IRQA if bit Ø of CRA is a 1 as described previously.

CA2 can act as a totally independent interrupt input or as a peripheral control output. As an input (CRA, bit 5 = Ø) it acts to set the interrupt flag, bit 6 of CRA, to a logic 1 on the active transition selected by bit 4 of CRA.

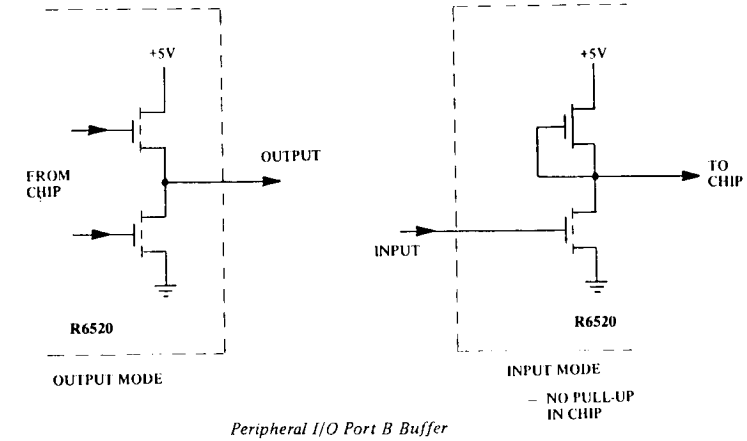
These control register bits and interrupt inputs serve the same basic function as that described above for CA1. The input signal sets the interrupt flag which serves as the link between the peripheral device and the processor interrupt structure. The interrupt disable bit allows the processor to exercise control over the system interrupts.

In the Output mode (CRA, bit 5 = 1), CA2 can operate independently to generate a simple pulse each time the microprocessor reads the data on the Peripheral A I/O port. This mode is selected by setting CRA, bit 4 to a "0" and CRA, bit 3 to a "1." This pulse output can be used to control the counters, shift registers, etc. which make sequential data available on the Peripheral input lines.



Peripheral I/O Port A Buffer

FIGURE 5-7a



Peripheral I/O Port B Buffer

FIGURE 5-7b

Peripheral I/O Port Buffers

FIGURE 5-7

A second output mode permits CA2 to be used in conjunction with CA1 to "handshake" between the processor and the peripheral device. On the A side, this technique allows positive control of data transfers from the peripheral device into the microprocessor. The CA1 input signals the processor that data is available by interrupting the processor. The processor reads the data and sets CA2 low. This signals the peripheral device that it can make new data available. This technique is discussed in detail in Section 3.

The final output mode can be selected by setting bit 4 of CRA to a 1. In this mode, CA2 is a simple peripheral control output which can be set high or low by setting bit 3 of CRA to a 1 or a 0, respectively.

The operation of CA1 and CA2 is summarized in the next section.

PERIPHERAL B INTERRUPT INPUT/PERIPHERAL CONTROL LINES (CB1, CB2)

CB1 operates as an interrupt input only in the same manner as CA1. Bit 7 of CRB is set by the active transition selected by bit 0 of CRB. Likewise, the CB2 input mode operates exactly the same as the CA2 input modes. The CB2 output modes, CRB, bit 5 = 1, differ somewhat from those of CA2. The pulse output occurs when the processor writes data into the Peripheral B Output Register. Also, the "handshaking" operates on data transfers from the processor into the peripheral device.

The operation of CB1 and CB2 is summarized in the next section. A more detailed discussion of handshaking on the Peripheral B I/O port is contained in Section 3 of this manual.

5.4 R6520 OPERATION

5.4.1 Control Register Operation

	7	6	5 4 3	2	1 Ø
CRA	IRQA1	IRQA2	CA2 Control	DDRA Access	CA1 Control

	7	6	5 4 3	2	1 Ø
CRB	IRQB1	IRQB2	CB2 Control	DDRB Access	CB2 Control

Control Register Bit Designations

FIGURE 5-8

TABLE 5-1

Control of Interrupt Inputs CA1, CB1

CRA (CRB) Bit 1	CRA (CRB) Bit 0	Active Transition of Input Signal*	IRQA (IRQB) Interrupt Outputs
0	0	Negative	Disable--remain high
0	1	Negative	Enabled--goes low when bit 7 in CRA (CRB) is set by active transition of signal on CA1 (CB1)
1	0	Positive	Disable--remain high
1	1	Positive	Enable--as explained above

*Note 1: Bit 7 of CRA (CRB) will be set to a logic 1 by an active transition of the CA1 (CB1) signal. This is independent of the state of Bit 0 in CRA (CRB).

TABLE 5-2

Control of CA2 (CB2) as Interrupt Inputs (Bit 5 = "0")

CRA (CRB) Bit 5	CRA (CRB) Bit 4	CRA (CRB) Bit 3	Active Transition of Input Signal*	IRQA (IRQB) Interrupt Output
0	0	0	Negative	Disable--remains high
0	0	1	Negative	Enabled--goes low when bit 6 in CRA (CRB) is set by active transition of signal on CA2 (CB2)
0	1	0	Positive	Disable--remains high
0	1	1	Positive	Enable--as explained above

*Note: Bit 6 of CRA (CRB) will be set to a logic 1 by an active transition of the CA2 (CB2) signal. This is independent of the state of Bit 3 in CRA (CRB).

5.4.2 R6520 Operation in R6500 Systems

A brief review of the overall operation of the R6520 should serve to tie together many of the details discussed previously.

During the system initialization routine which is executed in response to the processor RESET signal, the microprocessor will write a pattern of 1's and 0's into the Data Direction Registers. This will determine those lines which are to act as inputs and those which are to act as outputs.

This pattern will usually be fixed for the system operation. Therefore, the next step would be to set the various operating modes, active transitions, etc. which are controlled by the Control Registers. At the same time, the Data Direction Register Access Control Bit can be set to a 1 to allow the processor to control the Peripheral Ports during system operation.

The interrupts will normally remain disabled until the entire system is initialized. At this time, the interrupts are enabled and full system operation begins.

During system operation, the microprocessor will interrogate the switches, sensors, etc. in the peripheral device by reading the data on the Peripheral Input lines. Binary or decimal data may be transferred into the microprocessor in the same way. At the same time the various lights, motors, solenoids, etc. on the peripheral device are controlled by writing data into the appropriate bits of the Peripheral Output Registers. The entire sequence of operations is determined by the programmer to control a particular peripheral device in a defined manner. The various registers, gates, etc. in the Interface Device act primarily as a link between the internal processor operations and the various inputs and outputs on the peripheral devices being controlled.

TABLE 5-3

Control of CA2 Output Modes

CRA Bit 5	CRA Bit 4	CRA Bit 3	Mode	Description
1	0	0	"Handshake" on Read	CA2 is set high on an active transition of the CA1 interrupt input signal and set low by a microprocessor "Read A Data" operation. This allows positive control of data transfers from the peripheral device to the microprocessor.
1	0	1	Pulse Output	CA2 goes low for one cycle after a "Read A Data" operation. This pulse can be used to signal the peripheral device that data was taken.
1	1	0	Manual Output	CA2 set low
1	1	1	Manual Output	CA2 set high

TABLE 5-4

Control of CB2 Output Modes

CRB Bit 5	CRB Bit 4	CRB Bit 3	Mode	Description
1	0	0	"Handshake" on Write	CB2 is set low on microprocessor "Write B Data" operation and is set high by an active transition of the CB1 interrupt input signal. This allows positive control of data transfers from the microprocessor to the peripheral device.
1	0	1	Pulse Output	CB2 goes low for one cycle after a microprocessor "Write B Data" operation. This can be used to signal the peripheral device that data is available.
1	1	0	Manual Output	CB2 set low
1	1	1	Manual Output	CB2 set high

SECTION 6

R6522 VERSATILE INTERFACE ADAPTER (VIA)

6.1 R6522 ORGANIZATION

The R6522 Versatile Interface Adapter (VIA) provides all of the capability of the R6520. In addition, this device contains a pair of very powerful interval timers, a serial-to-parallel/parallel-to-serial shift register and input data latching on the peripheral ports. Expanded handshaking capability allows control of bidirectional data transfers between VIAs in multiple processor systems.

Control of peripheral devices is handled primarily through two 8-bit bidirectional ports. Each of these lines can be programmed to act as either an input or an output. Also, several peripheral I/O lines can be controlled directly from the interval timers for generating programmable-frequency square waves and for counting externally generated pulses. To facilitate control of the many powerful features of this chip, the internal registers have been organized into an interrupt flag register, an interrupt enable register, and a pair of function control registers.

Figures 6-1 through 6-3 show the R6522 interfacing, pinout designations, and block diagram, respectively.

6.2 PROCESSOR INTERFACE

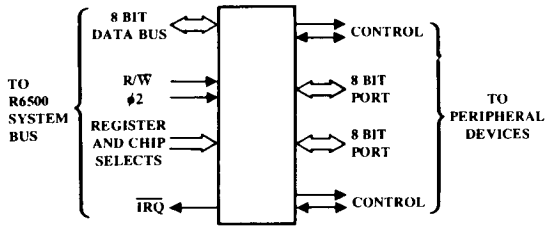
This section contains a description of the buses and control lines which are used to interface the R6522 to the system processor.

6.2.1 Phase Two Clock ($\phi 2$)

Data transfers between the R6522 and the system processor take place only while the Phase Two Clock is high. In addition, $\phi 2$ acts as the time base for the various timers, shift registers, etc. on the chip.

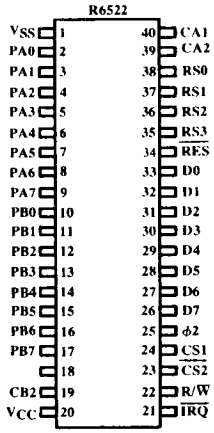
6.2.2 Chip Select Lines (CS1, $\overline{CS2}$)

The two chip select inputs are normally connected to processor address lines either directly or through decoding. The selected R6522 register will be accessed when CS1 is high and $\overline{CS2}$ is low.



R6522 Interface Diagram

FIGURE 6-1



R6522 Pinout Designations

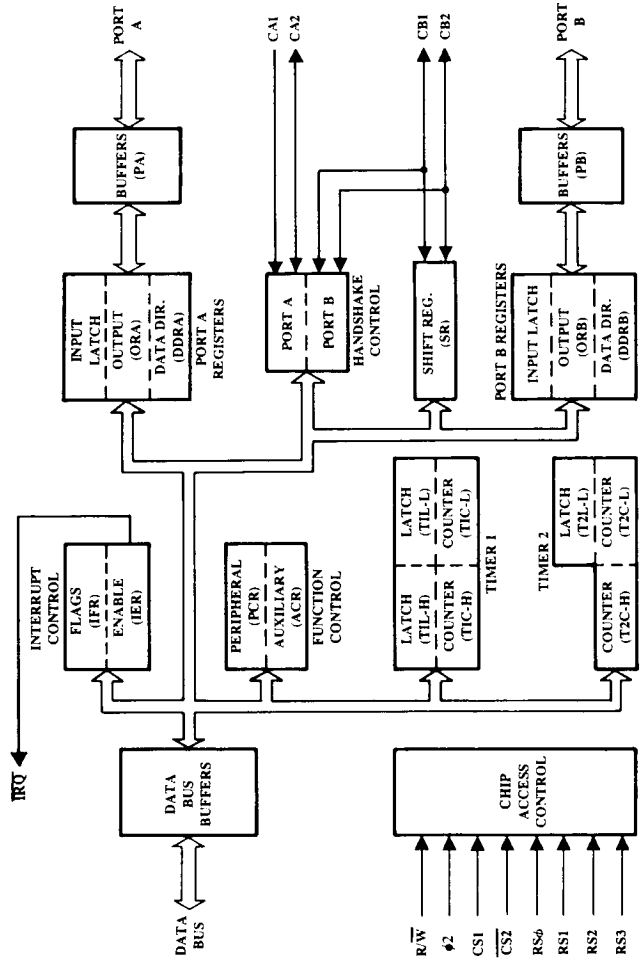
FIGURE 6-2

6.2.3 Register Select Lines (RS0, RS1, RS2, RS3)

The four Register select lines are normally connected to the processor address bus lines to allow the processor to select the internal R6522 register which is to be accessed. The sixteen possible combinations access the registers as follows:

RS3	RS2	RS1	RS0	Register	Remarks	RS3	RS2	RS1	RS0	Register	Remarks
L	L	L	L	ORB		H	L	L	L	T2L-L	Write Latch
L	L	L	H	ORA	Controls Handshake					T2C-L	Read Counter
L	L	H	L	DDRB		H	L	L	H	T2C-H	Triggers T2L-L/ T2C-L Transfer
L	L	H	H	DDRA		H	L	H	L	SR	
L	H	L	L	T1L-L	Write Latch	H	L	H	H	ACR	
				T1C-L	Read Counter	H	H	L	L	PCR	
L	H	L	H	T1C-H	Trigger T1L-L/ T1C-L Transf.	H	H	L	H	IFR	
						H	H	H	L	IER	
L	H	H	L	T1L-L		H	H	H	H	ORA	No Effect on Handshake
L	H	H	H	T1L-H							

Note: L = 0.4V DC, H = 2.4 V DC.



R6522 Block Diagram

FIGURE 6-3

6.2.4 Read/Write Line (R/$\overline{W}$)

The direction of data transfers between the R6522 and the system processor is controlled by the R/$\overline{W}$ line. If R/$\overline{W}$ is low, data will be transferred out of the processor into the selected R6522 register (write operation). If R/$\overline{W}$ is high and the chip is selected, data will be transferred out of the R6522 (read operation).

6.2.5 Data Bus (DB0 - DB7)

The eight bidirectional data bus lines are used to transfer data between the R6522 and the system processor. The internal drivers will remain in the high-impedance state except when the chip is selected (CS1 = 1, $\overline{CS2}$ = 0), Read/Write is high, and the Phase Two Clock is high. At this time, the contents of the selected register are placed on the data bus. When the chip is selected, with Read/Write low and $\phi 2$ = 1, the data on the data bus will be transferred into the selected R6522 register.

6.2.6 Reset ($\overline{RES}$)

The Reset input clears all internal registers to logic 0 (except T1, T2 and SR). This places all peripheral interface lines in the input state, disables the timers, shift register, etc. and disables interrupting from the chip.

6.2.7 Interrupt Request ($\overline{IRQ}$)

The Interrupt Request output goes low whenever an internal interrupt flag is set and the corresponding interrupt enable bit is a logic 1. This output is "open-drain" to allow the interrupt request signal to be "wire-OR'ed" with other equivalent signals in the system.

6.3 PERIPHERAL INTERFACE

This section contains a description of the buses and control lines which are used to drive peripheral devices under control of the internal R6522 registers. The operation of these peripheral interface lines is described in detail in subsequent sections.

6.3.1 Peripheral A Port (PA0 - PA7)

The Peripheral A port consists of eight lines which can be individually programmed to act as an input or an output under control of a Data Direction Register. The level of output pins is controlled by an Output Register and input data can be latched into an internal register under control of the CA1 line.

All of these modes of operation are controlled by the system processor through the internal control registers.

6.3.2 Peripheral A Control Lines (CA1, CA2)

The two peripheral A control lines act as interrupt inputs or as a handshake pair, one input and one output. Each line controls an internal interrupt flag with a corresponding interrupt enable bit. In addition, CA1 controls the latching of data on Peripheral A Port input lines. The various modes of operation are controlled by the system processor through the internal Control Registers.

6.3.3 Peripheral B Port (PB 0 - PB7)

The Peripheral B port consists of eight bidirectional lines which are controlled by an output register and a data direction register in the same manner as the PA port. These lines represent one standard TTL load in the input mode and will drive one standard TTL load in the output mode. In addition, they are capable of sourcing 1.0 ma at 1.5 VDC in the output mode to allow the outputs to directly drive Darlington transistor switches. In addition, the polarity of the PB7 output signal can be controlled by one of the interval timers while the second timer can be programmed to count pulses on the PB6 pin.

6.3.4 Peripheral B Control Lines (CB1, CB2)

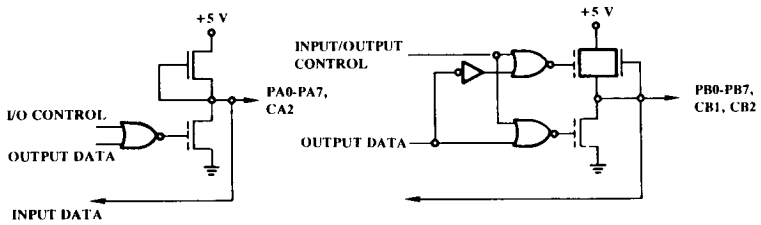
The Peripheral B control lines act as interrupt inputs or as a handshake pair, one input, and one output. As with CA1 and CA2, each line controls an interrupt flag with a corresponding interrupt enable bit. These lines represent one standard TTL load in the input mode and will drive one standard TTL load in the output mode. In addition, they are capable of sourcing 1.0 ma at 1.5 VDC in the output mode to allow the outputs to directly drive Darlington transistor switches. In addition, these lines act as a serial port under control of the Shift Register.

6.4 R6522 OPERATION

This section contains a discussion of the various blocks of logic shown in Figure 6-3. In addition, the internal operation of the R6522 is described in detail.

6.4.1 Data Bus Buffers (DB), Peripheral A Buffers (PA), Peripheral B Buffers (PB)

The characteristics of the buffers which provide the required voltage and current drive capability were discussed in the previous section.



Peripheral Output Buffers

FIGURE 6-4

6.4.2 Chip Access Control

The Chip Access Control contains the necessary logic to detect the chip select condition and to decode the Register Select inputs to allow accessing the desired internal register. In addition, the R/$\overline{W}$ and $\phi 2$ signals are utilized to control the direction and timing of data transfers. When writing into the R6522, data are first latched into a data input register during $\phi 2$. Data are then transferred into the desired internal register during Phase 2 Chip Select. This allows the peripheral I/O line to change without "glitching." When the processor reads the R6522, data are transferred from the desired internal register directly onto the Data Bus during Phase 2 high.

6.4.3 Port A Registers, Port B Registers

Three registers are used in accessing each of the 8-bit peripheral ports. Each port has a Data Direction Register (DDRA, DDRB) for specifying whether the peripheral pins are to act as inputs or outputs. A "0" in a bit of the Data Direction Register causes the corresponding peripheral pin to act as an input. A "1" causes the pin to act as an output.

When the pin is programmed to act as an output, the voltage on the pin is controlled by the corresponding bit of the Output Register (ORA, ORB). A "1" in the Output Register causes the pin to go high, and a "0" causes the pin to go low. Data written into Output Register bits corresponding to pins programmed to act as inputs will be unaffected.

Reading a peripheral port causes the contents of the Input Register (IRA, IRB) to be transferred onto the Data Bus. With input latching disabled, IRA will always reflect the data on the PA pins. With input latching enabled (ACR, bit 0), setting the CA1 Interrupt Flag (IFR1) by an active transition on CA1, will cause IRA to latch the contents of the Port A pins until the Interrupt Flag is cleared.

The IRB register operates in a similar manner. However, for output pins, the corresponding IRB bit will reflect the contents of the Output Register bit instead of the actual pin. This allows proper data to be read into the processor if the output pin is not allowed to go to full high voltage, e.g., driving transistors. If input latching is enabled on Port B, setting the CB1 Interrupt Flag will cause IRB to latch this combination of input data and ORB data until the Interrupt Flag is cleared.

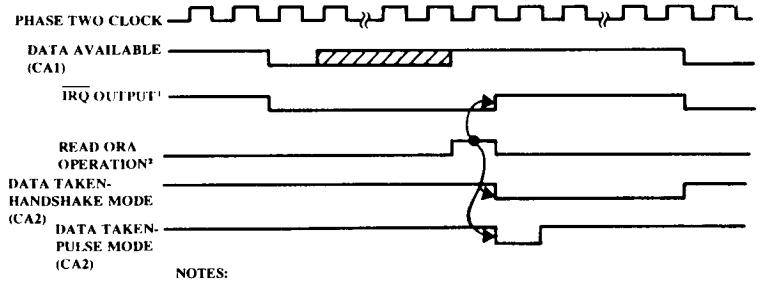
6.4.4 Handshake Control

The R6522 allows positive control of data transfers between the system processor and peripheral devices through the operation of "handshake" lines. Port A lines (CA1, CA2) handshake data on both a read and a write operation while the Port B lines (CB1, CB2) handshake on a write operation only.

READ HANDSHAKE

Positive control of data transfers from peripheral devices into the system processor can be accomplished effectively using "Read" handshaking. In this case, the peripheral device must generate "Data Ready" to signal the processor that valid data is present on the peripheral port. This signal normally interrupts the processor, which then reads the data, causing generation of a "Data Taken" signal. The peripheral device responds by making new data available. This process continues until the data transfer is complete.

In the R6522, automatic "Read" handshaking is possible on the Peripheral A port only. The CA1 interrupt input pin accepts the "Data Ready" signal and CA2 generates the "Data Taken" signal. The Data Ready signal will set an internal flag which may interrupt the processor or which can be polled under software control. The Data Taken signal can be either a pulse or a DC level which is set low by the system processor and is cleared by the Data Ready signal. These options are shown in Figure 6-5 which illustrates the normal Read Handshaking sequence.



NOTES:
1. SIGNALS "DATA AVAILABLE" TO THE SYSTEM PROCESSOR.
2. R/$\overline{W}$ = 1, $\overline{CS2}$ = 0, CS1 = 1, RS3 = 0, RS2 = 0, RS1 = 0, RS0 =1.

Read Handshake Timing Sequence

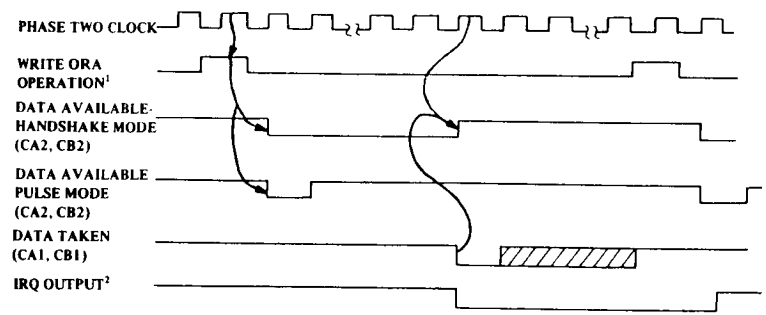
FIGURE 6-5

WRITE HANDSHAKE

The sequence of operations which allows handshaking data from the system processor to a peripheral device is very similar to that described in Section A for Read Handshaking. However, for "Write" handshaking, the processor must generate the "Data Ready" signal (through the R6522) and the peripheral device must respond with the "Data Taken" signal. This can be accomplished on both the PA port and the PB port on the R6522. CA2 or CB2 acts as a Data Ready output in either the DC level or pulse mode and CA1 or CB1 accepts the "Data Taken" signal from the peripheral device, setting the interrupt flag and clearing the "Data Ready" output. This sequence is shown in Figure 6-6.

6.4.5 Timer 1

Interval Timer T1 consists of two 8-bit latches and a 16-bit counter. The latches are used to store data which are to be loaded into the counter. After loading, the counter decrements at system clock rate, i.e., under control of the clock applied to the Phase Two input pin. Upon reaching zero, an interrupt flag will be set, and $\overline{IRQ}$ will go low if enabled. The timer will then disable any further interrupts, or will automatically transfer the contents of the latches into the counter and will continue to decrement. In



NOTES:
1. R/$\overline{W}$ = 0, $\overline{CS2}$ = 0, CS1 = 1, RS3 = 0, RS2 = 0, RS1 = 0, RS0 = 1.
2. SIGNALS "DATA TAKEN" TO THE SYSTEM PROCESSOR.

Write Handshake Timing Sequence

FIGURE 6-6

addition, the timer can be instructed to invert the output signal on peripheral pin PB7 each time it "times-out." Each of these modes is discussed separately below.

WRITING THE TIMER 1 REGISTERS

The operations which take place when writing to each of the four Timer 1 addresses are as follows:

RS3	RS2	RS1	RS0	Operation (R/$\overline{W}$ = L)
L	H	L	L	Write into low-order latch.
L	H	L	H	Write into high-order latch Write into high-order counter. Transfer low-order latch into low order counter. Reset T1 interrupt flag.(IFR6)
L	H	H	L	Write low-order latch.
L	H	H	H	Write high-order latch. Reset T1 interrupt flag. (IFR6)

Note that the processor does not write directly into the low-order counter (T1C-L). Instead, this half of the counter is loaded automatically from the low-order latch when the processor writes into the high-order counter.

In fact, it may not be necessary to write to the low-order counter in some applications since the timing operation is triggered by writing to the high-order counter.

The second set of addresses allows the processor to write into the latch register without affecting the count-down in progress. This is discussed in detail below.

READING THE TIMER 1 REGISTERS

For reading the Timer 1 registers, the four addresses relate directly to the four registers as follows:

RS3	RS2	RS1	RS0	Operation (R/$\overline{W}$ = H)
L	H	L	L	Read T1 low-order counter. Reset T1 interrupt flag (IFR6)
L	H	L	H	Read T1 high-order counter.
L	H	H	L	Read T1 low-order latch.
L	H	H	H	Read T1 high-order latch.

TIMER 1 OPERATING MODES

Two bits are provided in the Auxiliary Control Register to allow selection of the T1 operating modes. These bits and the four possible modes are as follows:

ACR7 Output Enable	ACR6 "Free-Run" Enable	Mode
0	0	Generate a single time-out interrupt each time T1 is loaded. PB7 is disabled.
0	1	Generate continuous interrupts. PB7 is disabled.
1	0	Generate a single interrupt and an output pulse on PB7 for each T1 load operation.
1	1	Generate continuous interrupts and a square-wave output on PB7.

TIMER 1 ONE-SHOT MODE

The interval timer one-shot mode allows generation of a single interrupt for each timer load operation. As with any interval time, the delay between the "write T1C-H" operation and generation of the processor interrupt is a direct function of the data loaded into the timing counter. In addition to generating a single interrupt, Timer 1 can be programmed to produce a single negative pulse on the PB7 peripheral pin. With the output enabled (ACR7=1) a "write T1C-H" operation will cause PB7 to go low. PB7 will return high when Timer 1 times out. The result is a single programmable width pulse.

> **NOTE**
>
> The PB7 output enable function will over-ride bit 7 of the Data Direction Register B. PB7 will act as an output if DDRB7=1 or if ACR7=1.

In the one-shot mode, writing into the high-order latch has no effect on the operation of Timer 1. However, it will be necessary to assure that the low-order latch contains the proper data before initiating the countdown with a "write T1C-H" operation. When the processor writes into the high-order counter, T1L-H will also copy the data, the T1 interrupt flag will be cleared, the contents of the low-order latch will be transferred into the low-order counter, and the timer will begin to decrement at system clock rate. If the PB7 output is enabled, this signal will go low on the phase two following the write operation. When the counter reaches zero, the T1 interrupt flag will be set, the $\overline{IRQ}$ pin will go low (interrupt enabled), and the signal on PB7 will go high. At this time the counter will continue to decrement at system clock rate. This allows the system processor to read the contents of the counter to determine the time since interrupt. However, the T1 interrupt flag cannot be set again unless a "write T1C-H" operation has taken place.

Timing for the R6522 interval timer one-shot mode is shown in Figure 6-7.

TIMER 1 FREE-RUNNING MODE

The most important advantage associated with the latches in T1 is the ability to produce a continuous series of evenly spaced interrupts and