2 7 0 0 1 1

M U L T I - 0 U S E R ' S M A N U A L

SERIAL
PARALLEL
IEEE-488
CLOCK/CALENDAR
12 BIT D-TO-A
8 CHANNEL 12 BIT A-TO-D

SUPPORT SOFTWARE

MAY, 1983 501-007-0 REV. B

COPYRIGHT NOTICE

Micro Technology Unlimited
2806 Hillsborough Street
P.O. Box 12106
Raleigh, NC 27605 USA
(919) 833-1458

DISCLAIMER OF ALL WARRANTIES AND LIABILITY

No warranties, either express or implied, are made by Micro Technology Unlimited with respect to this manual or the software described herein, its quality, performance, merchantability, or fitness for any particular application. This product is sold "as is". The buyer assumes all risk as to quality and performance. Under no circumstances will MTU be liable for direct, indirect, incidental or consequential damages resulting from any defect in the software, even if MTU has been advised of the possibility of such damages. Should the software prove defective following purchase, the buyer assumes the entire cost of all necessary servicing, repair or correction and any incidental, indirect, or consequential damages. Additional rights vary from state so some of the above exclusions and limitations may not apply to you.

NOTICE

Micro Technology Unlimited reserves the right to make changes to the product and specifications described in this manual at any time without notice.

TABLE OF CONTENTS

1. INTRODUCTION AND INSTALLATION

1.1 INTRODUCTION

MultI-O is an advanced multi-function input/output expansion board for the MTU-130 desktop computer. It provides the MTU-130 user with greatly expanded serial and parallel digital I/O capabilities, high accuracy analog I/O, a clock/calendar function, and an IEEE-488 bus (GPIB) interface. MultI-O is a contraction of the words "Multiple Input-Output" which is an apt name for a board with these many capabilities.

1.1.1 Standard Hardware Features

The MultI-O board provides two 8-bit parallel I/O ports supplemented with two "handshaking" lines each. These ports are typically used to interface printers, plotters, graphic tablets, and MTU accessories with a parallel interface. The parallel ports are driven by a 6522 Versatile Interface Adapter integrated circuit and terminate in a 36 pin connector. The pins in this connector; including +12, -12, and +5 volt power; are identical to the standard parallel I/O connector on every MTU-130 computer. Machine level programming of the parallel port is also identical except for different register addresses. Additionally, the 6522 VIA chip has two programmable interval timers and an 8-bit shift register which are completely available to the user.

The MultI-O board also provides two serial I/O ports. These are typically used to interface to modems, printers and plotters with serial interfaces, and to communicate directly with other computers. Each serial port is implemented with a 2651 synchronous/asychronous serial interface IC and each has its own programmable baud rate generator which is capable of speeds from 50 to 19,200 baud. Each serial port terminates in its own standard 25-pin D-type connector. Port 0 has a complete set of modem control signals while port 1 has only transmit and receive data implemented. Although machine level programming of the 2651 chip is slightly different from the MTU-130's standard 6551, it is generally more straightforward due to the former's greater number of control and status registers. Also, when combined with an appropriate modem and driver software, the chip's synchronous capability allows communicating with mainframe computers at high speed (2400 to 9600 bps) over telephone lines.

The IEEE-488 bus interface (also known as the General Purpose Interface Bus or GPIB) allows an MTU-130 equipped with a MultI-O board to communicate digitally with laboratory instruments having IEEE capability. This, coupled with the MultI-O's analog I/O capability, allows the MTU-130 to connect to a wide range of laboratory instrumentation. The IEEE interface is a full implementation using the TMS9914 integrated circuit and can function as a talker, listener, or controller under software control.

The clock/calendar function eliminates the need to enter the date whenever the system is "booted up" and allows the operating system and user programs to be constantly aware of the exact date and time to the second. The clock uses two standard, low-cost AA batteries mounted in the connector panel instead of expensive nickel-cadmium or silver-oxide watch batteries on-board.

Analog output is provided by a high-speed 12-bit digital-to-analog converter and amplifier with a range of -10 to +10, or 0 to +10 volts. Analog input may be either 8 single-ended or 4 fully differential overload protected channels which are serviced by a sample-and-hold amplifier and 50uS 12 bit analog-to-digital converter. Both the D-to-A and A-to-D converters are pre-adjusted to 12 bit accuracy and linearity and use low drift components including a temperature regulated 10 volt precision voltage reference.

1.1.2 Standard Software Features

Included on the MultI-0 Distribution diskette are many demonstration, utility, and diagnostic programs plus a variety of programming tools for both the BASIC and assembly language programmer. These programs and routines allow the MultI-0 user to tap much of the board's power with little or no programming effort. For advanced programmers, the included assembly source code files provide examples of proper MultI-0 programming.

Three demonstration programs are included to show the versatility of the analog I/0 and clock/calendar sections of the board and how the included programming tools are used in applications. The 8-channel digital voltmeter demonstration, which is less than a page of BASIC code, shows both a digital and a bar-graph readout of the voltage applied to each of the 8 analog inputs as well as the current date and time. The programmable voltage source demonstration, also in BASIC, shows how easily waveforms may be entered and edited with the light pen and then output at high speed through the digital-to-analog converter. The storage oscilloscope demonstration, which is written in 6502 assembly language, illustrates the high-speed performance capabilities of the MTU-130/MultI-0 combination.

A large number of utilities are provided for using the various board functions. Clock-calendar utilities include an interactive date/time setting program, an automatic date entry program that can be placed in your STARTUP.J file, and a utility for waiting until a specified date and time before returning. All of the standard printer drivers, both text and graphics, have been modified to use the MultI-0 parallel and serial ports and are included. In addition, the standard serial utilities (UPLOAD, SERLDR, etc.) including the AUTOTERM terminal emulator, have also been modified and included.

Software development tools include two BASIC libraries and packages of assembly language subroutines. The MIL BASIC library adds 19 commands and functions to MTU BASIC for operating the clock/calendar, analog I/0, and parallel I/0 functions of the MultI-0 board. A unique feature is its block analog I/0 functions which allow a BASIC program to handle analog data at rates up to 10K 12-bit samples per second without PEEKs, POKEs, or MCALLs. Fully commented assembly language subroutines are also included for operating the clock/calendar and the analog I/0 facilities.

The IEEEL library provides access to the IEEE-488 Bus, with the MTU-130 acting as the bus controller. Communication occurs via 10 IEEE "channels" which are assigned to devices on the IEEE bus. Commands are included for inputting and outputting data to the IEEE channels and polling the status of IEEE devices. The assembly language subroutines provide the necessary support to allow machine language programs to communicate with devices over the IEEE-488 bus.

1.1.3 Cautions

Although the MultI-0 board is a rugged device whose circuitry is protected to the greatest extent reasonable, it is also a precision device. Like all modern computer boards, the MultI-0 uses MOS integrated circuits that can be damaged by static discharge. While the analog inputs are protected against typical overloads up to 200 volts, direct discharge of thousands of volts of static to the analog inputs should be avoided. Note that when the MultI-0 board is installed into the MTU-130 that portions of the circuitry are still "live" even when the power is turned off because of the clock/calendar's battery. Therefore before repairing or modifying the board, it should be removed from the card file and the connecting cables disconnected. Finally, there are a number of programs on the MultI-0 Distribution Disk with the same name as standard MTU-130 distribution programs but which use I/O ports on the MultI-0 board. Please read section 2.1 for suggestions on how to avoid confusion.

1.2 INSTALLATION

Installation of the MultI-O involves the following major operations:

1. Removing the top cover of the MTU-130 keyboard unit.
2. Removing the RF interference shield.
3. Mounting the connector box to the rear of the '130 case.
4. Inserting the MultI-O board into the top bus slot.
5. Plugging the cables from the connector box onto the board.
6. Changing the jumper options, if desired.
7. Reassembling the keyboard unit.
8. Running the test programs supplied on the distribution disk.

Each of these operations will be described in a step-by-step manner in the following sections.

1.2.1 Keyboard Unit Disassembly

1. Unplug all cables from the MTU-130 keyboard unit and move it to an uncluttered, well lit work area.

2. Carefully stand the keyboard unit on its right end with the fan down.

3. Locate then remove two rear cover screws which are nearest the two back corners of the bottom plate and symmetrically placed. Also remove two front cover screws which are nearest the two front corners and go through long slotted holes in the bottom plate.

4. Lay the unit back down in its normal operating position but with the front overhanging the table by a couple of inches.

5. Remove the 4 screws along the front bottom surface that fasten the top cover to the bottom plate. Be sure to save the washers if present.

6. Slide the top cover forward about 1/2 inch then lift the front while keeping the back stationary so that it effectively "hinges" up about 40 degrees.

7. On the left side of the cover, locate the 2-wire cable going to the speaker and pull off the quick-disconnect lugs from the speaker terminals.

8. Lift the cover off completely and stand it upright at the right side of the base. The keyboard cable will be removed later. The fan wires may remain connected to the power supply board.

9. Remove the R-F shield which covers the front part of the boards. There are two screws on each edge of the shield.

10. Gently unplug the keyboard cable from its connector on the bottom PC board (use a small screwdriver to pry it up).

11. You should see the Monomeg CPU board (the large one in the bottom slot) and the Disk Controller board plugged into the card file. These need not be disturbed when installing the MultI-O board.

1.2.2 Mounting The Connector Box

1. Locate the connector box. It will have several cables hanging out of its back side.

2. Position the MTU-130 keyboard unit (or yourself) so that you are facing its rear panel. Remove the 4 screws holding the small vertical panel that is close to the power switch. The panel will not be used but you should save two of the screws.

3. Carefully remove the two screws holding the bus backplane that are closest to the parallel and serial connectors already present.

4. Obtain two AA size (penlite) batteries and install them into the battery holder in the connector box. Observe the polarity markings. Although no damage will result if the batteries are installed backwards, the clock will not keep time when power is off. Standard carbon-zinc types are more than adequate but should be fresh for maximum life (the batteries will last for their remaining shelf-life).

5. Thread the cables through the rectangular opening exposed in step 2 so that the connector box remains on the outside.

6. Position the connector box so that the batteries are uppermost and re-install the two screws removed in step 3 into the backplane standoffs.

7. Using two of the screws removed in step 2 and the two nuts supplied with the connector box, attach the other flange of the connector box to the '130's rear panel.

8. Route the cables between the power supply and the card cage to keep them out of the way.

1.2.3 Handling and Installing the MultI-0 Board

The MultI-0 board uses a number of MOS ICs in its construction. These are more sensitive to damage from static electricity than are TTL ICs which is why it is shipped in a black, conductive plastic bag. When handling the MultI-0 outside of this bag you should always hold it by the large metal shield plates. If it must be set down temporarily, sit it on top of its bag flattened out. If it is being set on a metal table or other highly conductive surface, touch the surface first with your hand to equalize the charge between your body (and the board since you are holding it) and the surface.

1. For the several cables from the connector box to easily fit onto the board, it should be installed in the top slot of the card file. If an expansion board already occupies the top slot, it should be moved down to the second slot. If both the top and second slots are occupied, it will be necessary to move the disk controller down to the fourth slot, immediately above the CPU board.

2. Carefully slide the MultI-0 board into the top slot until you feel it seat into its socket on the bus.

3. The MultI-0 board comes configured for bipolar analog input and output (-10 to +10 volts), twos-complement data representation, and 8 single-ended analog input channels. If you wish to change these settings, refer to section 1.3 and make the changes at this time. Note that the demonstration programs on the distribution disk may not operate properly if the jumpers are changed.

1.2.4 Installing the I/O Cables

There are 4 cables and connectors that route signals between the connector box and the MultI-0 board itself. Use the following procedure to connect them to the board. Since each connector is different and is keyed, there should be little chance of connecting them improperly.

1. Reposition yourself or the '130 case so that you are facing the rear edge of the MultI-0 board (the edge with all of the pin header connectors).

2. Extending from the right side of the MultI-0 board is a single flying lead with ringlug which will be connected directly to the MTU-130 system power supply. Place the ringlug over the screw holding the -12 volt regulator to the heatsink in the MTU-130 system power supply and secure it with the 4-40 nut supplied. Facing the front of the '130 chassis, the power supply is on the right and the -12 volt regulator is at the left of the power supply board and has a smaller heatsink than the two regulators at the front of the board.

3. The shortest cable from the connector box should be a flat ribbon type and is terminated in a 26 pin dual row female socket. This socket should be installed on J2 which is the rightmost header on the MultI-0 board. Be sure to plug the socket on so that the cable extends away from the rear of the board. This socket has two keying inserts installed in the opposite corners.

4. The next cable is made from two round cables terminated in the same single row 20 pin female socket. This should be plugged onto J3 which is 20 pins long. The connector can be plugged in only one way.

5. The third cable is also flat and very similar to the first cable except longer and not keyed. It should be plugged onto J5, again with the cable extending away from the board.

6. The last cable is a single round cable terminated in a single row 14 pin female socket. It should be plugged onto J4. When installed correctly, the wires will emerge downward from the connector.

7. All of the cables should be routed neatly along the rear edge of the board and then along the side of the card file so that they are out of the way when the RF shield and top cover are re-installed.

Reassembly Instructions

1. Find the top cover and re-attach the keyboard cable to the Monomeg CPU board.

2. Find the R-F shield and install it with its 4 screws. Insert each screw only part way until all 4 have been inserted. Then push the shield up and back as far as it will go and tighten the screws. Be careful not to pinch the flat cables.

3. Position the top cover in the "hinged up" position it was in step 6 of the disassembly instructions and reattach the two speaker lugs.

4. Lower the cover and slide it back until the top rear edge is flush with the back panel. The short tabs under the rear of the top panel should hook under the lip of the back panel. It may be necessary to press the top panel flat while doing this. Reposition the round cables if necessary to prevent them from being squeezed between the top cover and the card file edge.

5. While holding the two halves together with your hands, stand the unit on its right end as it was in step 2 of the disassembly instructions.

6. Carefully manipulate the cover as necessary to make the rear corner holes in the bottom plate match up with the internal cover mounting brackets and install the two rear corner screws (these were the ones that were removed in disassembly step 3). Also install the two front corner screws.

7. Set the unit back down with the front overhanging the tabletop edge and reinstall the 4 front cover screws that were removed in disassembly step 5.

8. Reconnect the keyboard unit to the disk drives, monitor, and power cord.

1.3 JUMPER OPTIONS

In order to accomodate a variety of applications, the analog and IEEE sections of the MultI-0 board have a number of jumper options. These have already been set by the factory to the most commonly desired configuration. The jumpers themselves consist of small black plugs that slip onto a pair of square pins. Each pair of pins is given an ID number (JPxx). Most of the analog section jumpers are exposed by removing two screws from the top shield plate and gently lifitng the shield plate up as the two voltage regulator leads bend. Refer to the parts layout drawing in section 11 for locations of the jumpers. If a jumper is removed, you may keep it from being lost by positioning it so as to engage only one of the square pins. The sections below describe each set of jumpers and how they may be changed if desired.

1.3.1 IEEE Interface Jumpers

Jumper JP14 determines whether the IEEE data lines (DI01 - DI08) are driven as open-collector or as tri-state lines. The default setting from the factory is JP14 installed which selects open-collector operation. Open-collector is necessary if parallel polls are to be used. Tri-state is normally used for very high speed applications (greater than 250KB/S) which are typically beyond the capability of the MultI-0/MTU-130 combination.

1.3.2 D-to-A Converter Jumpers

Jumpers JP7 and JP8 determine whether the D-to-A converter will produce voltages in the range of -10 to +10 volts or 0 to +10 volts. The default setting from the factory is JP7 installed and JP8 removed which gives a -10 to +10 volt range. If JP7 is removed and JP8 is installed, the 0 to +10 volt range is selected.

Jumpers JP3 and JP4 determine whether the DAC accepts offset-binary or twos-complement number coding. The default setting from the factory is JP3 installed and JP4 removed which gives twos-complement number coding ($800=-10 or 0; $000=0 or +5; $7FF=+10). This is most appropriate when the -10 to +10 volt range is selected (see above). If JP3 is removed and JP4 is installed, then offset-binary coding is selected ($000=-10 or 0; $800=0 or +5; $FFF=+10). This is most appropriate when the 0 to +10 volt range is selected.

1.3.3 A-to-D Converter Jumpers

Jumpers JP9 and JP10 determine whether the A-to-D converter will accept voltages in the range of -10 to +10 volts or 0 to +10 volts. The default setting from the factory is JP9 installed and JP10 removed which gives a -10 to +10 volt range. If JP9 is removed and JP10 is installed, the 0 to +10 volt range is selected.

Jumpers JP1 and JP2 determine whether the ADC produces offset-binary or twos-complement number coding. The default setting from the factory is JP2 installed and JP1 removed which gives twos-complement number coding ($800=-10 or 0; $000=0 or +5; $7FF=+10). This is most appropriate when the -10 to +10 volt range is selected (see above). If JP1 is installed and JP2 is removed, then offset-binary coding is selected ($000=-10 or 0; $800=0 or +5; $FFF=+10). This is most appropriate when the 0 to +10 volt range is selected.

Jumpers JP5, JP6, JP11, JP12, and JP13 determine whether the ADC has 8 single-ended input channels or 4 differential input channels. The default setting from the factory is JP5, JP11, and JP13 installed (JP5 and JP12 removed) which gives 8 single-ended input channels. If JP5, JP11, and JP13 are removed and JP6 and JP12 are installed, there will be 4 differential input channels. The non-inverting inputs will then be channel N+4 and the inverting inputs will be channel N where N is the differential channel number, either 1, 2, 3, or 4. In general, differential inputs are used when there is a considerable amount of "common mode" noise on the signals to be digitized. Refer to section 3.4 for additional discussion on the advantages and disadvantages of differential operation.

1.4 DIAGNOSTIC PROGRAMS

Two diagnostic programs are supplied on the MultI-0 distribution disk. These are similar to the ones used at the MTU factory except that they are simplified for use without the various loop-around test fixtures used at the factory. While they are not exhaustive, proper operation of the diagnostics gives a good indication that the board has not suffered shipping damage and has been installed properly. In addition to the diagnostic programs, the demonstration and utility programs can also be used to perform diagnostic functions.

1.4.1 MIOFTEST Diagnostic Program

This is a general functional test program which checks each major functional block of the board to the extent possible without loop-around cables. As each block and sub-circuit is checked, a pass-fail message is printed. If you have received a partial assembly of the MultI-0 (missing, for example, the analog I/O blocks), then a message indicating that the block under test is absent will be printed and the program will move on to the next functional block.

To run MIOFTEST, insert a copy of the MutlI-0 distribution disk into drive 0 and enter the command: MIOFTEST in response to a CODOS prompt. The program will be loaded from disk and immedaitely start running. Some of the tests require several seconds to complete. When testing is complete, it will print: FUNCTIONAL TEST COMPLETE and return to CODOS.

Most of the tests performed by MIOFTEST consist of checking the existence and proper functioning of the various registers in the I/O interface chips used. Since the actual external world I/O signals from these chips cannot be tested without loop-around cables, it is possible for a board to pass the MIOFTEST diagnostics yet still be faulty because of blown I/O buffers.

1.4.2 ADCERRPLOT Diagnostic Program

This program can be used to perform a quick visual check on the accuracy and linearity of the D-to-A and A-to-D converter. This program requires that the D-to-A converter output be connected to channel 1 of the A-to-D converter input. This can be accomplished by inserting a short piece of wire, paper clip, etc. between pins 2 and 8 of the analog I/O connector (15 pin D-type).

To run ADCERRPLOT, insert a copy of the MultI-0 distribution disk into drive 0 and enter the command: ADCERRPLOT in response to a CODOS prompt. The program will be loaded from disk and display a set of function key legends. One of two tests or a return to CODOS may be selected by pressing a function key.

The ALL VALS function key will begin a test sequence in which all 4096 possible voltage values are output to the D-to-A converter and then are read back from the A-to-D converter. The value read is subtracted from the value output to get an error value which is then plotted. The display consists of 10 rows of plotted information, each representing 2 volts of the 20 volts of range between -10 and +10 volts. The range represented by each row is indicated by labels at the left and right ends of the row. The solid straight horizontal line plotted in each row represents the values output to the DAC, one value for each pixel horizontally, 410 values per row. The A-to-D value read back is then plotted on the same axes. If the value read is identical to the value output, then it will plot on top of the DAC value and not be seen. If there is a difference, it will be plotted above or below the DAC value line according to the direction of the error. Each pixel vertically represents one least significant bit (about .004 volts) of deviation.

An ideal DAC and ADC adjusted perfectly would produce 10 unbroken horizontal lines one pixel in width. In practice, of course, there will be deviations due to imperfect zero and gain adjustments, linearity errors, and the effects of random noise. This is actually a very sensitive test since the display shows the sum of DAC and ADC errors rather than each individually. Properly adjusted units will typically show errors of + or - 2 units at various points in the range which is within specification when it is assumed that the DAC and ADC each contribute 1/2 of the total observed error.

The MAJOR CY function key performs a test that is a subset of the ALL VALS test. In this test, the 15 most error-prone voltage ranges are singled out and plotted much more rapidly than ALL VALS does. Each voltage range is represented by a short horizontal row which represents voltages from 8 counts (.04 volts) below the labeled voltage to 8 counts above. The plot in each row is constructed just like the ALL VALS plot and is interpreted in a similar manner. Typically the errors shown in these subranges should no larger than the errors over the entire range shown by ALL VALS.

2. DEMONSTRATION AND UTILITY PROGRAMS

This section describes the diskette that was included with your MultI-O board and how to use the demonstration and utility programs recorded on it. There are also programming tools on the disk which are described in section 4.

2.1 ABOUT THE MULTI-O DISTRIBUTION DISK

The MultI-O Distribution Disk contains a large quantity of software related to effective use of the MultI-O hardware board. This consists of standard operating system files, demonstration programs, programming tools, clock/calendar utilities, serial I/O utilities, and printer drivers. If you have not already done so, the distribution disk should be copied at this time to protect against damage or loss. On multi-drive systems this is most easily accomplished with the standard BACKUP utility which is on the distribution disk. Simply insert the distribution disk into drive zero, boot the system with MOD-RESET, enter BACKUP followed by a carriage return, and follow the instructions it displays. Single drive system users should use COPYF1DRIVE to copy all of the files as described in the MTU-130 System Notebook.

Below is a listing of the files on the MultI-O distribution disk by category:

SYSTEM	DEMO	CLOCK UTIL	SERIAL UTIL	PRINT DRIVERS
CODOS.Z	DVM.B	SETCLOCK.C	AUTOTERM.C	SYSGENPRINTR.C
COMDPROC.Z	DVS.B	SETCLOCK.A	UPLOAD.C	SYSGENPRINTR.A
SYSERRMSG.Z	SCOPE.C	AUTODATE.C	UPLOAD.A	SPRINTNEC.C
SVCPROC.Z	SCOPE.A	AUTODATE.A	DOWNLOAD.C	SPRINTNEC.A
STARTUP.J		WAITUNTIL.C	DOWNLOAD.A	SPRINTMX80.C
DIR.C		WAITUNTIL.A	SERLDR.C	SPRINTMX80.A
IODRIVER.Z			SERLDR.A	SPRINTPRISM.C
GRAPHDRIVER.Z			SERDMP.C	SPRINTPRISM.A
PRINTDRIVER.Z			SERDMP.A	SPRINTIDS440.C
COPYF.C	DIAGNOSTIC	TOOLS	SEND.C	SPRINTIDS440.A
COPYF1DRIVE.C			SEND.A	VMDUMPNEC.C
FORMAT.C	MIOFTEST.C	MIL.Z	RECEIVE.C	VMDUMPNEC.A
KILL.C	ADCERRPLOT.C	MIL.A	RECEIVE.A	VMDUMPMX80.C
EDIT.C		IEEEL.Z		VMDUMPMX80.A
BACKUP.C		IEEEL.A		VMDUMPPRISM.C
CPUID.C		DATETIMESUBS.A		VMDUMPPRISM.A
FILECRC.C		ADCDACSUBS.A		VMDUMPIDS440.C
MTU130EQU.A		IEEESUBS.A		VMDUMPIDS440.A
MACLIB6502.A				

After copying the distribution disk, you may wish to make one or more work disks with, for example, BASIC and its standard libraries included. This will be necessary to run the demonstration programs written in BASIC because the distribution disk as supplied is too full to receive a copy of BASIC.

Note that the files under Serial Utilities and Print Drivers have the same name as files you received on the MTU-130 distribution disk. These ARE NOT the same files however. In all cases they have been modified to use either the serial or parallel ports on the MultI-O board rather than the standard MTU-130 ports. To prevent possible confusion, only the serial utility and print driver files you need to use (for example to operate a printer or modem you have connected to the MultI-O board) should be copied onto working disks. You should also consider changing the name of any of these files as they are copied.

2.2 ANALOG I/O DEMONSTRATION PROGRAMS

Three demonstration programs using the analog I/O capabilities of the MultI-O board are included on the distribution disk. DVM.B is a BASIC program which displays a continuous digital and analog representation of the voltage level at each of the 8 analog inputs. DVS.B is a BASIC program provides three different methods for the user to control the output voltage of the D-to-A converter in real-time. SCOPE.C is an assembly language program that simulates the operation of a dual-trace oscilloscope on the MTU-130 screen. Detailed operation instructions for each program are given below.

2.2.1 DVM Digital Voltmeter Demonstration

DVM is a BASIC program that illustrates use of the MIL BASIC library for reading multi-channel input from the A-to-D converter. It also illustrates a graphical method for displaying slowly changing voltages using the IGL and VGL graphics libraries and a technique for printing numbers with a controlled number of decimal places. Follow these steps to load and run DVM:

1. Prepare a working disk from your backup MultI-O distribution disk which includes these files as a minimum: standard system files, BASIC, IGL.Z, VGL.Z, MIL.Z, DVM.B.

2. Start MTU BASIC in the usual way (either BASIC 1.0 or BASIC 1.5 may be used).

3. Enter: RUN "DVM"

At this point the DVM program will load in IGL, VGL, and MIL, clear the screen, and draw 8 "meter scales". Following this it will sample the voltage at each of the 8 analog inputs and display its digital equivalent in a column at the left of the screen and a corresponding bar-graph "meter pointer" immediately to the right. The pointer bar always starts from the center of the scale (zero) and either grows to the right for positive voltages or grows to the left for negative voltages. Note that if an analog input is not connected to anything, it will read wildly erratic values due to its extremely high input impedance responding to stray electrical fields. It is always a good idea to ground unused analog inputs either directly or through a 1meg resistor. The program will continue to sample the analog inputs indefinitely and redisplay the digital and graphical representation of any that change. Additionally, the current date is displayed in the upper left screen corner and the time in the upper right corner (see section 2.3.1 for setting instructions if the time and date are incorrect). DVM is stopped by pressing cntl/C on the keyboard.

2.2.2 DVS Digital Voltage Source Demonstration

DVS is a BASIC program that illustrates use of the MIL BASIC library for generating specific voltage values and waveforms at the D-to-A converter output. It also illustrates several methods for inputting constant and variable values to a BASIC program using the light pen functions of the IGL graphics library. Follow these steps to load and run DVS:

1. Prepare a working disk from your backup MultI-O distribution disk which includes these files as a minimum: standard system files, BASIC, IGL.Z, MIL.Z, DVS.B.

2. Start MTU BASIC in the usual way (either BASIC 1.0 or BASIC 1.5 may be used).

3. Enter: RUN "DVS"

At this time the DVS program will load in IGL and MIL, display 3 function key legends, and ask you to press a function key. The DIGITAL key will enter a very simple routine where you may enter any number between -10 and +10 and that voltage will be output by the D-to-A converter. This may be useful in checking the calibration of the D-to-A converter, or more likely, calibrating another piece of equipment. Entering 99 will exit this "digital input" mode and make the program responsive to the function keys again.

The ANALOG function key will enter a routine whereby the operator may continuously vary the DAC output voltage by moving the light pen up and down the white bar it displays. A small black "pointer" at the left edge of the bar, which is calibrated in volts, shows what the current output voltage is. Pointing the light pen to the box marked RETURN will make the program responsive to the function keys again.

The WAVE function key will enter a routine in which a waveform of voltage versus time may be drawn with the light pen and then output either as a single shot event or as a periodically repeating waveform. You will first be presented with a "sheet of graph paper" on which you may draw a waveform with the light pen. The background dot pattern provides a row for every 1.0 volts and a column for every 8 sample points. The zero axis is represented by a denser dotted line. Since a valid waveform must be a "single valued function" (i.e., no backtracking), the light pen entry routine is written to force this. Begin drawing your waveform at the left edge although you need not start at zero volts. As you slowly move to the right while drawing the waveform, the routine will interpolate a Y (voltage) point for every possible X (time) position. Don't try to draw too fast or you will get just a small number of straight lines connecting a few intermediate points. If you back up to the left, any part of the waveform to the right will be erased. When you are satisfied with the appearance of the waveform, point to the DONE box.

The program will now display "CONVERTING WAVEFORM ARRAY" for a few seconds and then display legends for several of the function keys. To play the stored waveform out through the D-to-A converter to an oscilloscope or other device once, press the ONE-SHOT function key. To start a repetitive output, press CONTNOUS. Press it again to stop the periodic output. The SET RATE key will ask for the time that each point on the waveform will dwell in the DAC before the next point is output. Allowable values are 70uS to 65535uS; the initial default value is 100uS. The SMOOTH 1 and SMOOTH 2 keys digitally filter the waveform you have drawn in order to make it smoother. SMOOTH 1 provides a moderate filtering effect which removes small irregularities while SMOOTH 2 provides heavy filtering which leaves only major waveform features intact. In either case the filtered waveform is displayed and either filter may be applied more than once. The EXIT key will return to the "main menu". Obviously, many improvements and features could be added to this simple one page demonstration program and the reader is encourged to do so.

2.2.3 SCOPE Oscilloscope Demonstration Program

SCOPE is an assembly language program that illustrates some of the speed and flexibility of the MTU-130/MultI-O combination. In essence it simulates a standard dual trace oscilloscope with the advantage of a long persistence screen with a "freeze" feature that can be used to freeze the waveform display at any time for study. SCOPE uses analog input 1 for channel A and input 2 for channel B.

To start SCOPE, simply insert a disk with the SCOPE.C program on it and enter the command: SCOPE followed by a carriage return. It will immediately clear the screen and then begin displaying channel A in the top portion of the screen at the fastest available sweep speed. Caution! SCOPE uses interrupts for timing the A-to-D conversions. It should be stopped only by using the EXIT function key (f8)!

The display area consists of a grid of dots with a row for every vertical divison and a column for every horizontal divison. Immediately above the function key legends are words and numbers that represent the current settings of the scope controls. The leftmost legend for example is labeled "V MODE" for the current vertical display mode. Immediately above is an indication of the current mode, either A, B, ALT, or CHOP. This is initially A but can be changed to the other possibilities by pressing f1 repeatedly until the desired mode is selected. Above f5 is the current sweep speed in milliseconds per divison. Like the vertical mode, f5 can be pressed to sequence through the available sweep speeds.

Function key f1 controls the vertical display mode. The "A" and "B" positions display the selected channel as a single trace in the upper and lower portion of the screen respectively. The "ALT." position alternately displays channel A at the top and channel B at the bottom to simulate the simultaneous display of two inputs. This is most useful at the faster sweep speeds. The "CHOP" position alternates between A and B on a point-by-point basis instead which gives a better illusion of simultaneous display. CHOP cannot be used on the fastest sweep speed however and there is a beep if an attempt is made to do so.

The f2 key controls the freeze function. When f2 is pressed, the display is stopped instantly and the FREEZE legend becomes backlighted. Since the display may be stoped in the middle of a sweep, there may be a visible break in the displayed waveform. The display to the left of the break represents the most recent part of the waveform while that to the right is left over from the previous sweep. To resume active waveform display, press f2 again.

The f3 key controls the amount of waveform displayed before the sweep was triggered. This allows viewing the leading edge of pulses or events leading up to the event which caused the trigger. The number displayed is the number of divisons (8 points per divison) of pretrigger waveform displayed from 0 to 20 divisone.

The f4 key controls the voltage level at which sweep triggering occurs. This is initially zero but can be varied from -9 to +9 volts in 1 volt steps by repeatedly pressing f4.

Sweep triggering is controlled by the TRIG key (f5). Triggering may either be from channel A or from channel B, regardless of whether that channel is being displayed or not. Positive triggering occurs when the trigger waveform passes through the trigger level from below (negative-to-positive). Negative triggering occurs on the transition from above to below the trigger legel.

The horizontal display mode is controlled by H MODE (f7). When the program is started, the horizontal sweep is free running and not influenced in any way by triggering. In the AUTO mode, there is a short pause at the end of each sweep during which the sweep is responsive to a trigger. If a trigger occurs before the end of this pause, a new sweep is started immediately. If no trigger occurs during the pause, a new sweep is started anyway. The TRIG mode is similar except that the wait is indefinite for a trigger at the end of a sweep. In the SINGLE mode, the freeze mode is automatically entered at the end of a sweep. Pressing FREEZE will then resume the wait for a trigger as in TRIG mode. Note that in any of the modes that require a trigger there will be some waveform data displayed before the actual trigger point according to the current value of PRETRIG.

The EXIT key (f8) is the only proper way to leave SCOPE and return to CODOS since interrupts are used. If you interrupt the program with the INT key, a system crash is likely when you attempt to run a different program. Pressing RESET after pressing INT can however preclude that possibility.

2.3 CLOCK/CALENDAR UTILITY PROGRAMS

Three utility programs are provided for convenient control of clock/calendar functions from the MTU-130 console or from a job file. SETCLOCK is used to display the current date and time, and if desired, set it. AUTODATE replaces CODOS's DATE function for totally automatic system startup. WAITUNTIL can be used to wait until a designated time or to continuously display the current date and time on the screen. Each is an assembly language program and the source for each is also included on the MultI-O distribution disk.

2.3.1 SETCLOCK Utility

The SETCLOCK utility is used to set the clock/calendar to the current date and time. This utility should be run after the MultI-O board is installed and also whenever the board is removed from the system since the backup battery is in the connector box rather than on the board. For safety, SETCLOCK is the only program provided that can set the clock.

To run SETCLOCK, insert a copy of the MultI-O distribution disk into drive 0, OPEN it, and enter the command: SETCLOCK and a carriage return. The screen will be cleared and the current contents of the date and time registers will be displayed. If the board has just been installed or has had its battery power interrupted, these registers will contain random data. To actually change the time, answer the question "DO YOU WISH TO CHANGE THE DATE AND TIME (Y/N)? - " with a Y. Then type in the new date and time in the indicated format (24 hour clock). Note that only the hours and minutes of the time are specified; the seconds will always be set to zero. The date and time registers are not actually set until you press return so you may edit the date and time line as much as desired. When return is pressed, the registers are written and then immediately read back and displayed. If the new date and time display is satisfactory, answer the question with an N to terminate the program. If an error is spotted, you can answer Y to repeat the setting sequence again.

The clock/calendar chip is set up by SETCLOCK for 24 hour mode and it is intelligent enough to assign the correct number of days to each month. It does not however handle leap years so you should reset the date on every February 29th to 29-FEB-XX at which point it will be good for another 4 years. Rated accuracy of the clock is + or - 30 seconds per month under normal indoor temperature conditions with the MTU-130 off the majority of the time. See section 6 for calibration instructions if the error is greater than this or you have the necessary precision instruments and wish to calibrate it more closely.

2.3.2 AUTODATE Utility

AUTODATE is intended to replace the DATE command of CODOS which is typically in the STARTUP.J file of every disk. AUTODATE can be used with or without the MultI-O board installed. In systems with the MultI-O, AUTODATE will read the date and time, display them, and then set CODOS's date stamp according to the date read. If for any reason the date displayed is not accurate, you can still use the DATE command to set CODOS's date stamp independently of the clock/calendar. If AUTODATE is run on a system without the MultI-O board installed, it simply reissues the standard DATE command which asks for manual input of the date.

2.3.3 WAITUNTIL Utility

WAITUNTIL is a dual function utility that can be used to wait until a specified date and time before returning and to continuously display the date and time. It is useful in job files to schedule the execution of a sequence of programs with each one beginning at a specified time.

To run WAITUNTIL, simply enter the command WAITUNTIL followed by the date and time in standard format (DD-MMM-YY HH:MM:SS) to wait until. The date may be omitted in which case CODOS's current date stamp is assumed. If neither the date nor the time is specified, the program will wait indefinitely. If the specified date and time have already passed, it will return immediately. While waiting, the program will continuously display the date and time to the second. The program may be aborted at any time by pressing cntl/C.

2.4 SERIAL I/O UTILITIES

All of the standard MTU-130 serial I/O utility programs have been recoded for the 2651 serial I/O chips on-board the MultI-0 board and included on the MultI-0 distribution disk. Except for the serial port used, they are identical to the standard utilities on the MTU-130 distribution disk. You should refer to the Utilities section of the MTU-130 System Notebook for operation instructions for these utilities. The assembly source code for each of these utilities (except AUTOTERM which is too large) is included on the MultI-0 distribution disk.

The following utilities have been recoded to use serial port 1 which is the one with full modem control signals:

AUTOTERM.C

These utilities have been recoded to use serial port 2 which has only data out and data in implemented:

UPLOAD.C DOWNLOAD.C SERLDR.C SERDMP.C SEND.C RECEIVE.C

2.5 PRINTER DRIVERS

All of the standard MTU-130 printer driver programs have been recoded for the parallel and serial I/O chips on the MultI-0 board. Except for the ports used, they are identical to the standard printer drivers on the MTU-130 distribution disk. The assembly source code for each of the printer drivers is included on the MultI-0 distribution disk.

SYSGENPRINTR is similar to the standard SYSGENPRINTR program. If parallel is selected, it will generate a Centronics style parallel printer driver that uses port B of the user parallel port on board the MultI-0 board. If serial is selected, it will generate an Anacom style serial printer driver that uses serial port 1 on the MultI-0 board. Except for these differences, operation is identical to the SYSGENPRINTR described in the MTU-130 System Notebook.

The following graphic printer drivers have all been recoded to use port B of the user parallel port on the MultI-0 board to drive a Centronics style parallel printer:

SPRINTNEC.A SPRINTMX80.A SPRINTPRISM.A SPRINTIDS440.A VMDUMPNEC.A VMDUMPMX80.A VMDUMPPRISM.A VMDUMPIDS440.A

3. CONNECTING TO EXTERNAL EQUIPMENT

Since the MultI-O board is, after all, an input/output board, proper connection to user devices is a central issue. The following sections will describe the connectors provided on the connector box and correct connection methods to a variety of devices.

3.1 PARALLEL PORTS

The two parallel ports terminate in a 36 pin female ribbon type of connector. The pin connections of this connector are identical to those of the standard MTU-130 parallel connector. They are listed below for convenience:

PIN	SIGNAL	PIN	SIGNAL
1	+5 VOLTS (100MA)	19	USER GROUND
2	N.C.	20	USER +12 VOLTS (100MA)
3	USER RESET	21	USER -12 VOLTS (50MA)
4	CA2	22	CA1
5	PA1	23	PA0
6	PA3	24	PA2
7	PA5	25	PA4
8	PA7	26	PA6
9	PB1	27	PB0
10	PB3	28	PB2
11	PB5	29	PB4
12	PB7	30	PB6
13	CB2	31	CB1
14	N.C.	32	N.C.
15	N.C.	33	N.C.
16	N.C.	34	N.C.
17	N.C.	35	N.C.
18	N.C.	36	N.C.

Outputs and ports programmed for output are guaranteed to drive one full TTL load (1.6MA in the zero state) and can typically drive as many as 5 (or one with a 1K resistor pullup to +5). All outputs can source at least .1MA at 2.4 volts while PB0-PB7 are further capable of sourcing at least 1MA at +1.5 volts for direct drive of darlington transistors. Inputs and ports programmed for input load the driving source with a maximum of 1.6MA in the zero state. Inputs have internal pullup resistors which means that they can be driven by simple switches to ground or open-collector sources without external added pullup resistors. Note that edge sensitive inputs (CA1, CA2, CB1, CB2, and PB6 when timer T2 is in pulse counting mode) respond reliably only to pulse widths greater than 2uS.

When driving longer cables (longer than approximately 5 feet) it is advisable to use twisted pairs for any signal that triggers something either in the external device (such as a strobe input to a printer) or on the MultI-O board (such as any of the handshake signals CA1, CA2, CB1, CB2, or PB6 when timer T2 is in pulse counting mode). One member of the pair should be connected to ground on both ends of the cable while the other member is the signal of interest. When driving very long cables (longer than 20 feet), all of the signals should use twisted pairs. These measures substantially reduce crosstalk and false triggering caused by rapid transitions of signal voltages.

The USER RESET signal will go low whenever the console Reset key is pressed. It will also remain low for approximately 1 second when power is applied to the system. This can be used to reset external equipment connected to the parallel port that does not have its own reset circuit.

The +5, +12, and -12 voltages are provided as a convenience for powering small interface circuits and small amounts of analog circuitry. The +5 and +12 voltages come from regulators on the MultI-O board while -12 comes directly from the MTU-130 power supply. Do not overload these supplies (see ratings in the pin connections table)! Overloading them may degrade the accuracy of the D-to-A and A-to-D converters, overheat the MultI-O board regulators, or even prevent the MTU-130 from operating properly.

3.2 SERIAL PORTS

The MultI-O board has two serial I/O ports. Each terminates in a standard 25 pin D-type female connector and is wired up as an RS-232 terminal. Port 1, which is the bottommost connector, has full modem controls implemented. Port 2 has only transmit and receive data implemented. The pin connections of each serial port connector are shown in the table below:

PORT 1 (LOWER)				PORT 2 (UPPER)			
PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL
1	FRAME GND	14	N.C.	1	FRAME GND	14	N.C.
2	TRANSMIT DATA	15	EXT TRANSMIT CLK	2	TRANSMIT DATA	15	N.C.
3	RECEIVE DATA	16	N.C.	3	RECEIVE DATA	16	N.C.
4	REQ. TO SEND	17	EXT RECEIVE CLK	4	N.C.	17	N.C.
5	CLEAR TO SEND	18	N.C.	5	N.C.	18	N.C.
6	DATA SET RDY	19	N.C.	6	N.C.	19	N.C.
7	SIGNAL GND	20	DATA TERM RDY	7	SIGNAL GND	20	N.C.
8	CARRIER DET	21	N.C.	8	N.C.	21	N.C.
9	N.C.	22	N.C.	9	N.C.	22	N.C.
10	N.C.	23	N.C.	10	N.C.	23	N.C.
11	N.C.	24	N.C.	11	N.C.	24	N.C.
12	N.C.	25	N.C.	12	N.C.	25	N.C.
13	N.C.			13	N.C.		

All signals on the serial connectors are standard EIA levels which are -12 volts for a mark (logic 1) and +12 volts for a space (logic zero). Output signals swing a full + and -11 volts and can deliver up to 10MA of current. The slew rate of output signals is internally limited to 13V/uS. Input signals are responsive signals as small as + and - 3 volts.

Input modem control signals (clear to send, data set ready, and carrier detect) have internal 10K pullup resistors to +12 volts. This makes these signals appear active when nothing is connected to them and thus allows operation of serial port 1 in simple data send/receive applications without special cables. For port 2, these signals are internally connected so as to always appear active to driving software. The external transmit and receive clock inputs are included for connection to synchronous modems. They should be left unconnected for asynchronous applications.

3.3 ANALOG OUTPUT

The analog output appears on the the analog I/O connector which is a 15 pin D-type female connector. See section 3.4 for pin connections for this connector. Note that the analog output has its own ground return which is isolated from the analog input ground by 100 ohms and is connected to digital ground at a single point. Devices being driven by the analog output should have their ground return connected directly to the analog output ground for best results.

The analog output is driven by an LM356 FET operational amplifier. This amplifier has an output impedance of a fraction of an ohm and is capable of driving up to 10MA into an external load. However in order to minimize self-heating and thus maximize accuracy, the load current should always be minimized. The settling time of the analog output is guaranteed to be less than 10uS and the slew rate is typically 13V/uS. Usually however the effective settling time is limited by the program that loads the D-to-A converter registers which is a two step operation and requires at least 4uS to accomplish. Capacitive loads up to 10,000pF may be driven without oscillation. For heavier capacitive loads than this, a 100 ohm resistor should be placed in series with the output at the MTU-130 end of the cable.

3.4 ANALOG INPUTS

The 8 analog inputs appear on the analog I/O connector which is a 15 pin D-type female connector. The pin connections for this connector are shown below:

PIN	SIGNAL	PIN	SIGNAL
1	ADC COMMON 1	9	ADC CH. 5
2	ADC CH. 1	10	ADC CH. 6
3	ADC CH. 2	11	ADC CH. 7
4	ADC CH. 3	12	ADC CH. 8
5	ADC CH. 4	13	ADC COMMON 2
6	SHIELD (DIG GND)	14	N.C.
7	N.C.	15	DAC COMMON
8	DAC OUT		

Note that the analog inputs have their OWN grounds (ADC COMMON 1 and ADC COMMON 2). These two grounds are equivalent and are isolated from the DAC ground (DAC COMMON) and system ground (SHIELD) by a 100 ohm resistor. This is to prevent circulating ground currents from introducing noise into the analog inputs.

The analog inputs enter an overvoltage protection circuit and an analog multiplexor (see the schematic diagram in section 12). The protection network consists of a 1K series resistor and diodes to the board's +12 and -12 volt power supplies. Thus any voltage above +12.6 or below -12.6 will be diverted into the power supplies. Momentary overvoltages up to + or - 200 volts can be withstood but a continuous overvoltage of that magnitude would burn out the 1K series resistor. You should note however that for input voltages greater than +10 volts that the channel will turn on whether it is selected or not and thus affect the ADC reading of a different channel. No damage is done but this effect could lead to unexpected measurement errors as long as the overvoltage persists.

Although the analog input impedance is very high (thousands of megohms), a small amount of bias current is drawn from each analog input. This current will cause an unconnected input to accumulate an unpredictable and possibly changing voltage that will certainly not be zero. Therefore it is important that any signal source have a DC return path to ground for draining off this bias current. Since the bias current magnitude is less than 5NA, a source resistance of 1 megohm or less is sufficient to keep bias current errors less than 1 least significant bit (5MV).

3.4.1 Single-Ended vs Differential

The A-to-D converter may be jumpered for either 8 single-ended inputs or 4 differential inputs; the normal configuration is single-ended. Single-ended means that one terminal of each analog source to be measured is connected to a common ground line (in this case ADC COMMON) and the other terminal is connected to an analog input channel. This configuration is normally somewhat susceptable to noise since any noise pickup on the ungrounded lead will be seen directly by the analog input and any noise pickup on the grounded lead will pass through the source and be seen by the input as well.

With differential inputs, the positive side of the source is connected to the non-inverting differential input (channel N+4) and the negative side is connected to the inverting differential input (channel N) where N is the channel number, either 1, 2, 3, or 4. Desired signal voltages produced by the source will be recognized since they tend drive the two inputs in opposite directions. External noise fields however will tend to induce voltages of the same magnitude and polarity on each of the signal leads resulting in no net difference noise voltage at the inputs. The noise is thus rejected by the differential inputs. Note that twisting the two input leads together will enhance the chance that the induced noise voltages will be equal. Although neither side of the signal source is connected to ground, there must be a DC path to ground of 1 megohm or less for bias currents to flow through.

In essence then, differential inputs have greater immunity to noise induced external to the signal source than do single-ended inputs. The difference is often dramatic; induced noise voltages in the low volt range may be rejected to the extent that measurement error is in the low millivolt range. The price paid however is a reduction in the number of analog channels from 8 to 4. The isolated ground feature of the MultI-O does however help reduce noise pickup in single-ended applications where all of the signal sources can share a common ground wire that is connected to ground only at the ADC COMMON pins on the analog I/O connector. This is because noise voltages induced into this single ground wire are subtracted from noise voltages induced into each signal and thereby at least partially cancel out.

3.4.2 Increasing the Analog Input Gain

The analog input differential amplifier comes from the factory set for unity gain. This means that the least significant bit of the A-to-D converter is about .005 volts in bipolar -10 to +10 mode or about .0025 volts in unipolar 0 to +10 mode. By installing a resistor on the board in the R28 position (component clips are provided for this purpose), the differential amplifier gain may be increased. The gain formula is: GAIN=1+(10000/Rg) where Rg is the user installed resistor's value in ohms. Following are values for some common gains: 10K=X2, 5K=X3, 2.5K=X5, 1.111K=X10, 526.3=X20, 204.1=X50, 101=100. The resistor used should be a stable metal film unit with .1% tolerance or better for a stable, predictable gain factor. When using internal gain, noise pickup considerations become much more critical. When possible, external amplifiers should be used in preference to increasing the internal gain.

3.5 DIGITAL I/O

Associated with the analog I/O subsystem is a digital I/O facility that is completely independent of the parallel I/O ports. This facility is implemented with four signals from the internal 6522 chip and provides for external triggering of events, pulse generation, pulse counting, and square wave generation of various frequencies in addition to regular TTL logic level sensing and generation. The digital I/O signals terminate in a 9 pin D-type female connector. The pin connections are shown below:

PIN	SIGNAL	PIN	SIGNAL
1	AIO PB6	6	AIO PB7
2	AIO CB2	7	AIO CB1
3	AIO +5 VOLTS (50MA)	8	AIO GROUND
4	N.C.	9	N.C.
5	N.C.		

The AIO prefix on the signal names indicates that they are associated with the analog I/O subsystem rather than the parallel ports. All signals are TTL compatible with .25 volt (0) and 3.5 volt (1) nominal logic levels. AIOPB6 can be used as a level sensing digital input, a digital output, or as an automatic pulse counting (2uS minimum pulse width) digital input. AIOPB7 can be used as a level sensing digital input, a digital output, or as a square wave output under the automatic control of timer 1. AIOCB1 is an edge sensing digital input with either edge selectable. AIOCB2 is an edge sensing input, pulse or level generating output, or a complex waveform output under the automatic control of timer 2. Additionally, AIOCB1 and AIOCB2 can be configured with the 6522's shift register for serial data transmission and reception using a variety offormats. Refer to section 5.2 for information on programming the 6522 I/O chip that is connected to these digital I/O lines. Keep in mind while reading section 5.2 that the internal 6522, which is addressed at $BF90-$BF9F, is used for the digital I/O.

A source of +5 volt power is provided on pin 3 for powering small amounts of digital logic. The current capacity of this output is 50MA if 5 volt power is also being drawn from the parallel outputs or 150MA if not. Do not overload this output or the MultI-0 5 volt regulator will overheat and affect the accuracy of the analog I/O circuits.

3.6 IEEE-488 BUS INTERFACE

The IEEE connector is a 24 pin ribbon type female connector. This should fit standard IEEE cables which are available from a variety of sources. The pinout is IEEE standard and is shown below:

PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL
1	DIO1	7	NRFD	13	DIO5	19	GND 7
2	DIO2	8	NDAC	14	DIO6	20	GND 8
3	DIO3	9	IFC	15	DIO7	21	GND 9
4	DIO4	10	SRQ	16	DIO8	22	GND 10
5	EOI	11	ATN	17	REN	23	GND 11
6	DAV	12	SHIELD	18	GND 6	24	LOGIC GND

4. USING THE SUPPLIED PROGRAMMING TOOLS

Because of the general nature of the MultI-O hardware, it is impossible to provide complete application programs for all or even a small portion of its potential applications. Consequently the software emphasis is on a good set of easy-to-use programming tools that allow the user to develop his or her own complete application programs with a minimum of effort. These tools take the form of two BASIC libraries for the BASIC programmer and two packages of subroutines for the assembly or compiler language programmer. Each of these four programming tools are described in the sections below.

4.1 MIL BASIC LIBRARY

MIL is the name of a library which adds 19 commands and functions to BASIC's repertoire for operating the clock/calendar, analog I/O, and parallel I/O ports on the MultI-O board. This library may be used in conjunction with other BASIC libraries such as CIL, IGL, and VGL without interference. MIL can be used with either BASIC 1.0 (standard MTU BASIC) or with BASIC 1.5 (uses the DataMover for greater speed and storage). Using MIL will typically subtract 2.25K from the available BASIC memory (just the program memory in BASIC 1.5). This section will describe each of the 19 MIL commands and give simple examples of their use. In addition, two of the MultI-O demonstration programs (DVM.B and DVS.B) are written in BASIC using the MIL library and can serve as more advanced examples.

4.1.1 Loading Instructions

To load the MIL library, simply include the name "MIL" as a file name in a LIB command. For example, you could use:

```
LIB "MIL"
```

The MIL library program is contained in the disk file MIL.Z. The ".Z" part of the name is assumed by the LIB command. If the MIL library is not on the default drive (usually 0), then you should specify the drive number as below:

```
LIB "MIL:1"
```

for example. Actually there are many copies of the program in the file, each assembled to run at a different location. This is done to give the LIB command a choice of where it may load the MIL library so that it will not conflict with libraries already in memory.

4.1.2 Clock and Calendar Functions

The clock and calendar functions allow a BASIC program to read the date and time as character strings and convert any given date and time into absolute time in seconds for comparison purposes. For safety, the MIL cannot set the date or time.

4.1.2.1 The DATE$ Function

PURPOSE: To read the current date as a 9 character string.

SYNTAX: DATE$

ARGUMENTS: None

DISCUSSION: The DATE$ function is used to read the current date from the clock/calendar in the format DD-MMM-YY. The MMM field may be one of the following: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC. The DATE$ function may be used anywhere a simple string variable may be used.

EXAMPLE: This BASIC staement: PRINT "The date is "+DATE$+"."
will print: The date is 04-JUL-76.
assuming that is the current date.

4.1.2.2 The TIME$ Function

PURPOSE: To read the current time as an 8 character string

SYNTAX: TIME$

ARGUMENTS: None

DISCUSSION: The TIME$ function is used to read the current time from the clock/calendar in the format HH-MM-SS. The HH field uses 24 hour format, i.e., 18:00:00 for 6:00 PM. The DATE$ function may be used anywhere a simple string variable may be used.

EXAMPLE: This BASIC statement: PRINT DATE$+" "+TIME$
will print: 04-JUL-76 19:27:32
assuming that is the current date and time.

4.1.2.3 The ABSTIME Function

PURPOSE: To convert a date and time string into absolute seconds since Jan 1, 1960.

SYNTAX: ABSTIME (stringvar)

ARGUMENTS: stringvar = A string of 17 or more bytes expressing the date and time. in the format "DD-MMM-YYHH:MM:SS"

DISCUSSION: This function is used to convert a date and time into a single number representing the number of seconds between 01-JAN-60 00:00:00 and the stated time. This is very useful when magnitude comparing two dates and times. Zero or more blanks may separate the 9 character date from the 8 character time. MTU BASIC's 32 bit floating point mantissa carries sufficient precision to represent the absolute time until the year 2028 without error.

EXAMPLE: The following program segment will print the number of days between an operator entered date and the present date:

```
10 INPUT "ENTER PROPOSED SELLING DATE (DD-MMM-YY) - ";D$
20 DY=(ABSTIME(D$+"00:00:00")-ABSTIME(DATE$+"00:00:00"))/(24*60*60)
30 PRINT "NUMBER OF ELAPSED DAYS =",DY
40 ...
```

4.1.3 Digital-to-Analog Converter Commands

The digital-to-analog converter commands allow a BASIC program to output either a single voltage value to the D-to-A converter for controlling something or an entire array of values for generating a varying control function or waveform.

4.1.3.1 The DACS Command

PURPOSE: To output a single signed voltage value to the D-to-A converter.

SYNTAX: DACS V

ARGUMENTS: V = Any numeric expression that evaluates to a value between -32768 and +32767.

DISCUSSION: This statement is used to update the voltage value being generated by the D-to-A converter. The new voltage value will persist until changed by another DACS, a DACU, a DACBLKS, or a DACBLKU command. This statement assumes that the DAC has been jumpered for -10 to +10 volts and twos-complement; see section 1.3 for details. The actual voltage produced by the DAC is: V/3276.8 volts. Note that since the DAC only has a 12 bit resolution that a change of 16 units + or - in the value of V is necessary to effect a non-zero output voltage change.

EXAMPLE: The BASIC statement: DACS -5*3276.8 will produce an output voltage of -5.000 volts.

4.1.3.2 The DACU Command

PURPOSE: To output a single unsigned voltage value to the D-to-A converter.

SYNTAX: DACU V

ARGUMENTS: V = Any numeric expression that evaluates to a value between 0 and +32767.

DISCUSSION: This command is used to update the voltage value being generated by the D-to-A converter. The new voltage value will persist until changed by another DACS, a DACU, a DACBLKS, or a DACBLKU command. This command assumes that the DAC has been jumpered for 0 to +10 volts and offset--binary; see section 1.3 for details. The actual voltage produced by the DAC is: V/3276.8 volts. Note that since the DAC only has a 12 bit resolution that a change of 8 units + or - in the value of V is necessary to effect a non-zero output voltage change.

EXAMPLE: The BASIC statement: DACU 6*3276.8 will produce an output voltage of +6.000 volts.

4.1.3.3 The DACBLKS Command

PURPOSE: To output an array of signed voltage values to the DAC at high speed.

SYNTAX: DACBLKS A%(I),N,SP

ARGUMENTS: A%(I) = An integer array containing values to output to the DAC. The first element output is I.
I = The subscript representing the first element of the A% array to output.
N = The number of elements of the A% array to output.
SP = The sample period in microseconds, see below for interpretation

DISCUSSION: This command will output elements of the specified integer array starting at the Ith element and continuing for N elements. SP specifies the source that will "trigger" the output of each element of the array. If SP is zero, the trigger will be a user supplied signal on the AIOCB1 pin in the analog I/O connector. If SP is positive, an internal clock will be used as the trigger. From 85 to 65535 microseconds may be specified. If SP is negative, no output will be performed until a pulse is seen on the AIOCB1 pin. Once the first pulse is seen, the timer will be used to output the rest of the array elements at the rate specified by the negative of SP. This command expects the DAC to be jumpered for -10 to +10 volts and twos-complement coding (see section 1.3 for details). The actual voltage values output are equal to the array values divided by 3276.8. If I+N-1 is greater than the dimension of the specified array, then garbage will be output.

EXAMPLE: The following program segment will output 10 cycles of a 250Hz tone burst when triggered by pulse on AIOCB1:

```
10 DIM SW%(200)
20 FOR I=1 TO 200: SW%(I)=32767*SIN(.314159*I): NEXT I
30 DACBLKS SW%(1),200,-100
40 GOTO 30
```

4.1.3.4 The DACBLKU Command

PURPOSE: To output an array of unsigned voltage values to the DAC at high speed.

SYNTAX: DACBLKU A%(I),N,SP

ARGUMENTS: A%(I) = An integer array containing values to output to the DAC. The first element output is I.
I = The subscript representing the first element of the A% array to output.
N = The number of elements of the A% array to output.
SP = The sample period in microseconds, see above for interpretation

DISCUSSION: This command is identical to DACBLKS except that the DAC is assumed to be configured for 0 to +10 volts and offset binary coding. The elements of A% must be in the range of 0 to +32767.

4.1.4 Analog-to-Digital Converter Commands

The analog-to-digital converter commands allow a BASIC program to input either a single voltage value from a designated channel of the A-to-D converter or an entire array of values for digitizing rapidly changing signals for later analysis.

4.1.4.1 The ADCS() Function

PURPOSE: To read a single signed voltage value from a specified A-to-D channel.

SYNTAX: ADCS(CH)

ARGUMENTS: CH = Any numeric expression that evaluates to a value between 1 and 8 which designates the channel to be read.

DISCUSSION: This function is used to read the current valtage level appearing at the specified channel of the A-to-D converter. The value returned will range between -32768 and +32752. The actual voltage level is equal to the value returned divided by 3276.8. This function assumes that the ADC has been jumpered for -10 to +10 volts and twos-complement; see section 1.3 for details. Note that since the ADC only has a 12 bit resolution that the values returned will always be divisible by 16. If the ADC has been configured for differential operation, the channel number should be constrained between 1 and 4.

EXAMPLE: The BASIC statement below will print the voltage appearing on channel C

```
PRINT "CHANNEL";C;" VOLTAGE IS";ADCS(C)/3276.8;"VOLTS"
```

4.1.4.2 The ADCU() Function

PURPOSE: To read a single unsigned voltage value from a specified A-to-D channel.

SYNTAX: ADCU(CH)

ARGUMENTS: CH = Any numeric expression that evaluates to a value between 1 and 8 which designates the channel to be read.

DISCUSSION: This function is used to read the current valtage level appearing at the specified channel of the A-to-D converter. The value returned will range between 0 and +32760. The actual voltage level is equal to the value returned divided by 3276.8. This function assumes that the ADC has been jumpered for 0 to +10 volts and offset-binary; see section 1.3 for details. Note that since the ADC only has a 12 bit resolution that the values returned will always be divisible by 8. If the ADC has been configured for differential operation, the channel number should be constrained between 1 and 4.

EXAMPLE: The following loop will double the voltage appearing on the specified input channel and output it through the DAC:

```
10 INPUT "ENTER CHANNEL # TO EXAMINE - ";CH
20 DACU 2*ADCU(CH): GOTO 20
```

4.1.4.3 The ADCBLKS Command

PURPOSE: To read a block of voltage values from the A-to-D converter into an array.

SYNTAX: ADCBLKS A%(I),M,SP,C%(J),N

ARGUMENTS: A%(I) = An integer array to receive the values read from the ADC.
I = The subscript representing the first element of the A% array to receive a value.
M = The total number of values that will be stored into A%.
SP = The sample period in microseconds, see below for interpretation
C%(J) = An integer array of channel numbers to read from, see below.
J = The subscript representing the first element of the C% array to read a channel number from.
N = The number of elements in the C% array that constitute one "scan" of the analog inputs.

DISCUSSION: This command will read a total of M signed values from the A-to-D converter into the specified integer array starting at subscript I and continuing for M elements. Each value read, regardless of the channel it is read from, is stored into successively higher elements of the A% array.

SP specifies the source that will "trigger" the reading of each value that is stored. If SP is zero, the trigger will be a user supplied signal on the AIOCB1 pin in the analog I/O connector. If SP is positive, an internal clock will be used as the trigger. From 110 to 65535 microseconds may be specified if N is greater than 1. If N=1, then the minimum allowable SP is 80. If SP is negative, no input will be performed until a pulse is seen on the AIOCB1 pin. Once the first pulse is seen, the timer will be used to input the rest of the array elements at the rate specified by the negative of SP.

Integer array C% contains a list of channel numbers to read from. C%(J) is the first element of the list and C%(J+N-1) is the last element of the list. After each sample is read and stored in A%, the next element of C% is accessed. If M is greater than N, the list is scanned through repeatedly until M samples have been taken. If C% contains positive channel numbers (+1 through +8), SP or the external pulse specifies the time between each select-read-store cycle, i.e., the readings are equally spaced in time. If C% contains negative channel numbers, the entire list of channels is read as rapidly as possible (burst mode) and SP or the external pulse specifies the time between bursts. If C% contains mixed positive and negative channel numbers, the burst will end after a positive channel number is read or when the end of the C% array is reached. See below for examples.

This command expects the ADC to be jumpered for -10 to +10 volts and twos-complement coding (see section 1.3 for details). The actual values stored are equal to the voltage values multiplied by 3276.8. Both the A% and C% arrays must already exist. In addition, A% must be dimensioned for at least I+M-1 elements; if it is dimensioned for fewer elements, other variables, strings, or even libraries may be overwritten in memory.

EXAMPLES: The program segement below will digitize the signal on channel 3 at a 5KHz rate for 1 second starting when a pulse is seen on AI0CB1:

```
10 DIM SI%(5000)
20 CH%(0)=3
30 ADCBLKS SI%(0),5000,-200,CH%(0),1
```

The program below will quickly sample 4 points 100 times per second for 10 seconds. It will then compute and print the overall average voltage that appeared on each of the channels.

```
10 DIM DA%(4000), CH%(4), AV(4)
20 INPUT "ENTER THE 4 CHANNELS TO READ FROM - ";C1,C2,C3,C4
30 CH%(1)=-C1: CH%(2)=-C2: CH%(3)=-C3: CH%(4)=-C4
40 ADCBLKS DA%(0),4000,10000,CH%(1),4
50 FOR I=1 TO 3999 STEP 4
60 FOR J=0 TO 3:  AV(J)=AV(J)+DA%(I+J):  NEXT J
70 NEXT I
80 PRINT "THE AVERAGES ARE:";
90 FOR I=0 TO 3: PRINT AV(I)/3276.8,: NEXT I
```

The program segment below will sample channel 1 at a 1Hz rate and channel 2 at a 50Hz rate for 1 minute. It will than print out the 60 channel 1 values and the min/max of channel 2 during that second.

```
10 DIM Z%(60*51), CH%(50)
15 CH%(0)=-1
20 FOR I=1 TO 50: CH%(I)=2: NEXT I
25 ADCBLK Z%(0),60*51,50000,CH%(0),51
30 FOR I=0 TO 60*51 STEP 51
35 MX=-100000: MN=100000
40 FOR J=1 TO 50
45 IF Z%(I+J)<MN THEN MN=Z%(I+J)
50 IF Z%(I+J)>MX THEN MX=Z%(I+J)
55 NEXT J
60 PRINT "CH1= ";Z%(I)/3276.8;"CH2 MIN= ";MN/3276.8,"MAX= ";MX/3276.8
65 NEXT I
```

4.1.4.4 The ADCBLKU Command

PURPOSE: To read a block of positive voltage values from the A-to-D converter into an array.

SYNTAX: ADCBLKU A%(I),M,SP,C%(J),N

ARGUMENTS:
- A%(I) = An integer array to receive the values read from the ADC.
- I = The subscript representing the first element of the A% array to receive a value.
- M = The total number of values that will be stored into A%.
- SP = The sample period in microseconds, see below for interpretation
- C%(J) = An integer array of channel numbers to read from, see below.
- J = The subscript representing the first element of the C% array to read a channel number from.
- N = The number of elements in the C% array that constitutes one "scan" of the analog inputs.

DISCUSSION: This command is identical to DACBLKS except that the ADC is assumed to be configured for 0 to +10 volts and offset binary coding. The elements of A% will be in the range of 0 to +32767.

4.1.5 Analog Bit I/O Commands and Functions

Associated with the analog inputs and outputs are four individual digital bits that may be used for a variety of trigger and sensing functions. These 4 signals are completely separate from the parallel I/O ports.

4.1.5.1 The AIOCB1 Function

PURPOSE: To determine if a pulse has been detected on the AIOCB1 digital input.

SYNTAX: AIOCB1

ARGUMENTS: None

DISCUSSION: This function will return a zero if no pulse has been seen on the AIOCB1 input since the last time the AIOCB1 function has been used. It will return a 1 if a pulse had been seen and then resets it. The input is actually sensitive to negative edges and is also reset by use of the AIOPBR, AIOPBW, ADCS, ADCU, ADCBLKS, and ADCBLKU functions and commands.

EXAMPLE: The BASIC statement:

```
100 IF AIOCB1=0 THEN GOTO 100
```

will wait until a pulse is seen on AIOCB1 and then continue with the next sequential statement.

4.1.5.2 The AIOCB2 Command

PURPOSE: To control the AIOCB2 digital output as either a level or a pulse.

SYNTAX: AIOCB2 M

ARGUMENTS: M = A numeric expression that evaluates to a value between 0 and 3 which which is interpreted as below:
0 = Generate logic low level
1 = Generate logic high level
2 = Generate a negative 50uS pulse
3 = Generate a positive 50uS pulse

DISCUSSION: This command is useful to turn equipment on or off or to trigger an event using the AIOCB2 output. AIOCB2 is set up as an output when the MIL library is linked in. Prior to initialization, it will default to an input and therefore appear as a logical high level.

EXAMPLE: The program segment below will produce a low-going 50uS pulse to trigger an event which is subsequently digitized using an ADCBLKS command:

```
500 PRINT "PRESS f1 WHEN READY TO START (f8 TO CANCEL)"
510 ON KEY GOTO 520 510 510 510 510 510 510 9000
511 GOTO 510
520 AIOCB2 2
600 ADCBLKS DA%(1000),2000,SR,CH%(0),8
9000 STOP
```

4.1.5.3 The AIOPBW Command

PURPOSE: To control the AIOPB6 or AIOPB7 digital I/O signals.

SYNTAX: AIOPBW M,N

ARGUMENTS: M = A numeric expression that evaluates to either 6 or 7 to select which AIO port B bit to control (either AIOPB6 or AIOPB7).
N = A numeric expression that evaluates to a value between 0 and 5 which is interpreted as below:
0 = Generate logic low level
1 = Generate logic high level
2 = Generate a negative 50uS pulse
3 = Generate a positive 50uS pulse
4 = Set up as an input (initially set when MIL is loaded)
5 = Set up as an output

DISCUSSION: This command is useful to sense an external digital input level or to turn equipment on or off or to trigger events using the AIOPB6 and AIOPB7 digital outputs. Note that a signal must be set up as an output using N=5 before the N=0 through N=3 functions will work. M is not checked for erroneous values between 0 and 5.

EXAMPLE: The program segment below will set AIOPB6 to a logic low, ask the operator for a parameter, set AIOPB7 to a logic high, wait for the specified time, and then begin taking data from analog input channel 5:

```
100 AIOPBW 6,5: AIOPBW 7,5: AIOPBW 6,1: AIOPBW 7,0: REM INITIALIZE
...
200 AIOPBW 6,0
300 INPUT "ENTER INITIAL DELAY IN SECONDS BEFORE TAKING DATA-";,D
310 FT=ABSTIME(DATE$+TIME$)+D
320 AIOPBW 7,1: REM START REACTION
330 IF ABSTIME(DATE$+TIME$)<FT THEN GOTO 330
600 ADCBLKS DA%(0),5000,50000,CH%(5),1
```

4.1.5.4 The AIOPBR() Function

PURPOSE: To read the logic state of the AIOPB6 or AIOPB7 digital I/O signals.

SYNTAX: AIOPBR (N)

ARGUMENTS: M = A numeric expression that evaluates to either 6 or 7 to select which AIO port B bit to read (either AIOPB6 or AIOPB7).

DISCUSSION: This function will return the digital level on AIOPB6 or AIOPB7. It returns 0 if the digital level is low and returns 1 if the digital level is high. This function is valid regardless of whether the bit checked is set up as an input or an output.

The BASIC statement below will wait intil AIOPB6 becomes a logic high:

```
100 IF AIOPBR(6)=0 THEN GOTO 100
```

4.1.6 Parallel I/O Bit I/O Commands and Functions

These commands and functions are intended for direct control of individual bits of the parallel ports on the MultI-O board. These are useful when the 4 digital I/O lines associated with the analog I/O port are not sufficient for the intended control function. Parallel ASCII or binary I/O on these ports should be performed with PEEK, POKE, or user-written assembly language routines rather than these bit oriented commands and functions.

4.1.6.1 The PIOPAW Command

PURPOSE: To control the Port A parallel I/O signals.

SYNTAX: PIOPAW M,N

ARGUMENTS: M = A numeric expression that evaluates to a value between 0 and 7 which selects which parallel I/O port A bit to control.
N = A numeric expression that evaluates to a value between 0 and 5 which is interpreted as below:
0 = Generate logic low level
1 = Generate logic high level
2 = Generate a negative 50uS pulse
3 = Generate a positive 50uS pulse
4 = Set up as an input (initially set by system reset)
5 = Set up as an output (output must be set first for 0-3 to work)

DISCUSSION: This command is similar to AIOPBW except that port A of the user 6522 connected to the parallel I/O ports is affected.

EXAMPLE: The program segment below will produce a binary control pattern on parallel I/O port a according to the contents of the B% array:

```
100 FOR I=0 TO 7: PIOPAW I,5: NEXT I: REM SET PORT A TO ALL OUTPUTS
200 FOR I=0 TO 7: PIOPAW I,B%(I): NEXT I: REM COPY B% TO PORT A
```

4.1.6.2 The PIOPAR() Function

PURPOSE: To read the logic state of the Port A parallel I/O signals.

SYNTAX: PIOPAR (M)

ARGUMENTS: M = A numeric expression that evaluates to a value between 0 and 7 which selects which parallel I/O port A bit to read.

DISCUSSION: This function will return the digital level on the selected parallel I/O port A bit. It returns 0 if the digital level is low and returns 1 if the digital level is high. This function is valid regardless of whether the bit checked is set up as an input or an output.

This BASIC statement below will wait until parallel I/O port A bit 3 becomes a logic high: `100 IF PIOPAR(3)=0 THEN GOTO 100`

4.1.6.3 The PIOPBW Command

PURPOSE: This command is identical to PIOPAW except that it controls Port B.

4.1.6.4 The PIOPBR() Function

PURPOSE: This function is identical to PIOPAR except that it reads Port B.

The IEEEL is a BASIC Library which allows BASIC programs to access the IEEE-488 bus with the MTU-130 acting as the bus controller. This bus is also known as the General Purpose Instrument Bus (GPIB). The "G" at the beginning of each of the commands in this library comes from the "G" in GPIB. The commands in this library provide a simple means of transferring data on the IEEE-488 bus. Also, there are commands and functions which implement some of the more advanced features of the IEEE-488 bus, such as serial and parallel polling. One of these is a command to allows you to send any interface message (i.e. command) on the bus to handle special situations.

The IEEEL supports 10 IEEE channels, numbered 0 through 9. These channels are implemented entirely within the IEEEL library, and are not associated with the CODOS channels in any way (except in concept). To communicate with an IEEE-488 device, you must assign an IEEE channel to the device, which is done with the GASSIGN command. Once the IEEE channel is assigned, the GGET# and GINPUT# commands may be used to input data from the associated device. The GPRINT# command may be used to send data to the device. A GEOI# command is provided to output a single byte with the EOI signal asserted.

Normally the GGET#, GINPUT#, and GPRINT# will each address the approriate device, transfer the data, and then unaddress the device. The overhead is incurred even if only one byte is transferred. If desired, this overhead may be eliminated by executing a GNOBRK command. After executing a GNOBRK command, the next GGET#, GINPUT#, or GPRINT# command will not unaddress the device when the command is completed, unless one of several events occurs. Afterwards, communication on that channel reverts back to the normal overhead. Refer to the discussion of the GNOBRK command for a description of the terminating events. A restriction on the use of the GNOBRK command is that operation must be returned to normal before any communications can occur over other IEEE channels.

A number of the commands set the reserved ST variable to indicate the completion status of the command. Also, these commands will set the ST variable in a consistent manner. The value is initially set to zero. It is incremented by 1 for each data argument sent or received, assuming the command transfers data. When outputing, the count is incremented only if all the data for an argument is successfully sent. When inputing, the count is incremented if an argument receives any data. 128 will be subtracted from the current value of ST, if the EOI signal is asserted while the final byte was being transferred. And finally, 64 is added to the value of ST if an error occurs while transfering a byte on the IEEE-488 bus. This typically will occur only when a particular device is not reponding, or isn't present on the IEEE-488 bus. Since an error adds 64, this implies that the commands which count data arguments should not have more than 63 arguments. This restriction should not present any hardship.

The IEEEL library does not support having the MTU-130 act as a device on the IEEE-488 bus responding to another controller. The library expects the MTU-130 to be the bus controller. Also, the IEEEL library does not support passing control to another controller on the IEEE-488 bus.

Below is a summary of the commands and functions available in the IEEEL library:

COMMANDS:

GASSIGN# - Assigns an IEEE channel to a device and optional secondary address.
GEOI# - Outputs a byte to an IEEE channel with the EOI signal asserted.
GFREE# - Frees an IEEE channel.
GGET# - Gets a byte from an IEEE channel.
GINIT - Causes MTU-130 to take control of, and initialize the IEEE-488 bus.

GINPUT# - Inputs data from an IEEE channel.
GNOBRK - Disables overhead during next communication on IEEE channel.
GNOLF# - Causes line feeds not to be expected or sent after carriage returns.
GOFF - Causes MTU-130 to release control of the IEEE-488 bus.
GPRINT# - Outputs data to an IEEE channel.
GCMD - Sends an interface message command byte-by-byte on the IEEE-488 bus.

FUNCTIONS:
GPADR() - Returns the primary device addrres associated with an IEEE channel.
GPPOLL() - Returns the result of a parallel poll of the IEEE-488 bus.
GSADR() - Returns the secondary address associated with an IEEE channel.
GSPOLL() - Returns result from a serial poll of an IEEE channel.
GSRQ() - Returns 1 if a service request is active on the IEEE-488 bus.

4.2.1 THE IEEEL COMMANDS

4.2.1.1 The GASSIGN Command

PURPOSE: To assign an IEEE channel to a primary address (i.e. device).

SYNTAX: GASSIGN <ieee chan>, <pri. addr> [, <sec. addr>] [, <string>]

ARGUMENTS: <ieee chan> = a numeric expression which evaluates to a number from 0 to 9. This number will be used as the IEEE channel to be assigned.
<pri. addr> = a numeric expression which evaluates to a number from 0 to 30. This number will be used as the primary address of the device with which the channel will communicate.
<sec. addr> = a numeric expression which evaluates to a number from 0 to 30. This number will be used as the secondary address within the device with which the channel will communicate.
<string> = a string to be sent to the specified primary and secondary addresses on the IEEE-488 bus.

DISCUSSION: The GASSIGN command is used to assign one of the ten IEEE channels to an IEEE device. These IEEE channels are implemented entirely within the IEEEL library, and are in no way associated with CODOS channels (except in principle). As a result, these IEEE channels may be accessed only by the commands in the IEEEL library.

The device with which to communicate is specified by its primary address. The secondary address is optional, and should be used only if the specified device supports secondary addresses. The string, if present, will be sent to the specified device. If no string is specified, then no action occurs on the IEEE-488 bus as a result of the GASSIGN command.

EXAMPLE: GASSIGN 4,8,2,"F1R1"

will assign IEEE channel 4 to IEEE device 8, secondary address 2 and send the string "F1R1" to this device.

NOTES: 1. The GASSIGN command will return the ST variable set appropriately. If no string is specified, then ST is returned set to 0. If a string is specified and is sent successfully, ST is returned set to 1. If a string is specified and the selected device does not respond, ST is returned set to 64.

4.2.1.2 The GEOI# Command

PURPOSE: To output a byte to an IEEE channel with the EOI signal asserted.

SYNTAX: GEOI # <ieee chan>, <byte>

ARGUMENTS: <ieee chan> = a numeric expression which evaluates to a number from 0 to 9. This number will be used as the IEEE channel to which the byte will be sent.
<byte> = a numeric expression which evaluates to a number from 0 to 255.

DISCUSSION: The GEOI# command is used to output a single byte to an IEEE channel. When the byte is sent to the assigned device, the EOI signal will be asserted. This command is intended to be used when the normal device addressing and unaddressing overhead has been suspended by a GNOBRK command. It allows the EOI signal to be sent explicitly with a particular byte. In addition to sending the byte, the GEOI# command will also unaddress the device and return the channel to normal addressing and unaddressing overhead.

EXAMPLE: GEOI# 4,12

will output the value 12 (hex $0C) to IEEE channel 4. The assigned device will receive the byte with the EOI signal asserted and the device will be unaddressed by an unlisten message on the IEEE-488 bus.

NOTES: 1. The GEOI# command will return the ST variable set to -127 (1 item minus 128 since EOI is sent) if the output occurs successfully. If the device does not respond, the ST variable is returned set to 64.

4.2.1.3 The GFREE Command

PURPOSE: To free an IEEE channel.

SYNTAX: GFREE <ieee chan>

ARGUMENTS: <ieee chan> = a numeric expression which evaluates to a number from 0 to 9. This number will be used as the IEEE channel to be freed.

DISCUSSION: The GFREE command is used to free the specified IEEE channel. Normally, executing this command does not cause any action to occur on the IEEE-488 bus. However, if the device is currently addressed as a result of executing a GNOBRK command, the GFREE command will unaddress the device to terminate the communication. In either case, the GFREE command will make the channel available for assignment to other devices.

EXAMPLE: GFREE 4

will free IEEE channel number 4.

NOTES: 1. If the device assigned to the IEEE channel is already unaddressed, or is successfully unaddressed by the GFREE command, ST is returned set to 0. If unsuccessful in unaddressing the device, ST is returned set to 64.

4.2.1.4 The GGET# Command

PURPOSE: To input a single character from an IEEE channel.

SYNTAX: GGET# <ieee chan>, <variable> [,<variable>] ...

ARGUMENTS: <ieee chan> = a numeric expression which evaluates to a number from 0 to 9. This number will be used as the IEEE channel from which the byte will be read.
<variable> = a variable which is to receive the input character.

DISCUSSION: The GGET# command reads data one character at a time from the specified IEEE channel. If the variable to receive the character is a string variable, then it is assigned to a string containing the one character. If it is a numeric variable, then the character is converted to a number before storing it in the variable. The command will read characters until each variable receives a character, or until an EOI signal is received. If an EOI signal is received, then any variables which haven't received a character will remain unchanged.

EXAMPLE: GGET# 8,A,B,G$

will input three bytes from the IEEE-488 bus assigned to IEEE channel 8. The first and second bytes will be stored as numbers in the variables A and B, respectively. The third byte will be stored as a one character string in the string variable G$.

NOTES: 1. If a GNOBRK command is in effect, the GGET# command will address the assigned device, if not already addressed. In addition, the GGET# command will not unaddress the device unless an EOI signal is received. Once unaddressed, normal IEEE channel operation resumes.

2. The GGET# command returns the ST variable set appropriately. The value will equal the number of arguments which received a character or value, minus 128 if EOI occurred. Since the EOI will terminate the GGET# command, the number of arguments which receive data may be less than the number present in the command. If no devices are present on the bus, ST will be returned set to 64.

3. There is no time limit within which a device must respond with its data. The IEEEL will wait forever for the device to respond, even if it isn't present in the system. If the device is failing to respond, or a non-existent device is accidentally being accessed, you may abort the GINPUT# command by pressing the BREAK key. The ST variable will be returned set to its current value at the time of the abort, plus 64.

4.2.1.5 The GINIT Command

PURPOSE: To reset the IEEE-488 bus and have the MTU-130 become the bus controller.

SYNTAX: GINIT

ARGUMENTS: none

DISCUSSION: The GINIT command is used to assume control of the IEEE-488 bus. It will cause the MTU-130 to assert the "Interface Clear" signal after which the '130 will assume control of the IEEE-488 bus. This command must be issued prior to executing any of the other commands in the IEEEL library.

EXAMPLE: GINIT

This statement will cause the IEEE-488 bus to be initialized and have the MTU-130 become the bus controller.

4.2.1.6 The GINPUT# Command

PURPOSE: To input data from an IEEE channel.

SYNTAX: GINPUT# <ieee chan>, <variable> [, <variable>] ...

ARGUMENTS: <ieee chan> = a numeric expression which evaluates to a number from 0 to 9. This number will be used as the IEEE channel from which data will be read.
<variable> = a variable which is to receive the input data.

DISCUSSION: The GINPUT# command is used to input data from the specified IEEE channel. Data will be input and stored in BASIC's input buffer until a comma, carriage return, or EOI signal is received. Also, the inputing will stop should the input buffer become full. Once the desired data has been input, it is converted as required and stored in the next variable. If there are additional variables specified, more data will be read and stored into these variables. Data will be read until all variables have received data, or an EOI signal is received. Any remaining variables which have not received data will be left unchanged.

Multiple items of data in the stream of characters received from the device may be separated by commas (,) or carriage returns. Each data item will be stored in a successive variables in the argument list of the GINPUT# command. When a carriage return is receive, the GINPUT# command will normally read one additional character, expecting it to be a line feed. If it is not a line feed, an error will occur. A GNOLF# command should be executed for the channel if the carriage returns will not be followed by line feeds in the input stream.

When inputing into a numeric variable, a string will be read until a separator is found. This string is then converted to a numeric value and stored in the variable. If there is any additional data in the string between the end of the number and the separator, that data will be lost.

EXAMPLE: GINPUT# 4,A$,B$,N

will input 3 items from the device assigned to IEEE channel 4. The first two items will be stored as strings in the variables A$ and B$. The third item is converted to a number and stored in the variable N.

NOTES: 1. If a GNOBRK command is in effect, the GINPUT# command will address the assigned device, if not already addressed. In addition, the GINPUT# command will not unaddress the device unless an EOI signal is received. Once unaddressed, normal IEEE channel operation resumes.

2. The GINPUT# command returns the ST variable set appropriately. The value will equal the number of arguments which received a character or value, minus 128 if EOI occurred. Since the EOI will terminate the GINPUT# command, the number of arguments which receive data may be less than the number present in the command. If no devices are present on the bus, ST will be returned set to 64.

3. There is no time limit within which a device must respond with its data. The IEEEL will wait forever for the device to respond, even if it isn't present in the system. If the device is failing to respond, or a non-existent device is accidentally being accessed, you may abort the GINPUT# command by pressing the BREAK key. The ST variable will be returned set to its current value at the time of the abort, plus 64.

4. The use of the comma as a separator may be disabled by POKEing 128 into location 1925 ($0785 hex). At this point only a carriage return will separate data items.

5. Unlike the standard BASIC INPUT statement, the GINPUT# command does not give any special treatment to double quotes (") or colons (:).

4.2.1.7 THE GNOBRK COMMAND

PURPOSE: To eliminate the normal overhead incurred by the GGET#, GINPUT#, and GPRINT# commands when they access the IEEE-488 bus.

SYNTAX: GNOBRK

ARGUMENTS: none.

DISCUSSION: The GNOBRK command is used to eliminate the normal overhead incurred by the GGET#, GINPUT#, and GPRINT# commands when they access the IEEE-488 bus. Normally each of these commands must first address the device assigned to the specified IEEE channel. Then after the data is sent or received, the device is unaddressed so that communications with other devices may occur. This means that each time one these commands is executed, a device will be addressed and unaddressed. In some cases, it may be desireable, or even necessary to eliminate the extra addressing and unaddressing (i.e. the overhead) that comes from executing the I/O commands multiple times during a transfer.

Executing a GNOBRK command for an IEEE channel will cause the next GGET#, GINPUT#, or GPRINT# command to address the assigned device, but not unaddress it after the data is sent or received. When a device is left addressed, communications with other devices may not occur until the device is unaddressed. Also, addressing a device also determines if the device is to send or receive data. This means that if a GINPUT# or GGET# command follows the GNOBRK, then the GPRINT# and GEOI# may not be used on that channel until it is restored to normal operation. Likewise, GINPUT# and GGET# may not be used if a GPRINT# follows the GNOBRK command.

Once a device has been left addressed as a result of a GNOBRK command, there four events which will unaddress the device and cause normal operation to be resumed. These events are:

1. Executing a GFREE command for the associated channel.
2. An EOI signal received during a GINPUT# or GGET# command.
3. Executing a GPRINT# command with no arguments specified.
4. Executing a GEOI# command.

Until one of these events occurs, the device will remain addressed. During this time, no other communications may occur on the IEEE-488 bus. However, it may be possible to perform a parallel poll without disturbing the channel.

EXAMPLE: GNOBRK:PRINT#4,"DATA"

will cause the normal addressing and unaddressing overhead to be suspended for IEEE channel 4.

NOTES: 1. The GNOBRK command itself will not cause the associated device to be addressed. On the next GGET#, GINPUT#, or GPRINT# command the device will be addressed, and remain addressed until normal operation is restored.

4.2.1.8 The GNOLF Command

PURPOSE: To specify if line feeds should be expected or sent after carriage returns when using the GINPUT# or GPRINT# commands, respectively.

SYNTAX: GNOLF# <ieee chan>

ARGUMENTS: <ieee chan> = a numeric expression which evaluates to a number from 0 to 9. This number will be used as the IEEE channel to be configured.

DISCUSSION: The GNOLF# command is used to specify whether line feeds should be expected or sent following carriage returns for a particular IEEE channel. This applies when inputing data with the GINPUT# command, or when outputing data with the

4.2.1.9 The GOFF Command

PURPOSE: To have the MTU-130 release control of the IEEE-488 bus.

SYNTAX: GOFF

ARGUMENTS: none

DISCUSSION: The GOFF command is used make the MTU-130 release control of the IEEE-488 bus. This command must be executed if you wish to allow some other device or computer to act as the IEEE-488 bus controller. As long as the MTU-130 thinks it is the controller, it will continue to drive certain signals on the bus. Executing the GOFF command will essentially disconnect the MTU-130 from the IEEE-488 bus, thus allowing some other device to be the controller. It is recommended that this command be executed prior to exiting BASIC if a GINIT command has been previously executed.

EXAMPLE: GOFF

This statement will cause the MTU-130 to release control of the IEEE-488 bus.

NOTES: 1. The MTU-130 will also release control of the IEEE-488 bus if a cold or warm reset of the 130 is performed.

2. The IEEEL library does not have the ability to pass control directly from the MTU-130 to another device capable of assuming control. You must execute a GOFF command first, then manually start the other controller.

4.2.1.10 The GPRINT# Command

PURPOSE: To output data to an IEEE channel.

SYNTAX: GPRINT# <ieee chan> [, <expression> [; [<expression>]] ...]

ARGUMENTS: <ieee chan> = a numeric expression which evaluates to a number from 0 to 9. This number will be used as the IEEE channel to which the output will be sent.

<expression> = an expression which evaluates to the data to be output.

DISCUSSION: The GPRINT# command is used to output data to an IEEE channel. Any numeric data specified will be converted to a string before being sent over the channel. After all the data has been sent, a terminating carriage return will be appended unless the GPRINT# command ends with a semicolon.

Normally a line feed will be sent immediately following each carriage return that occurs in the data sent by the GPRINT# command. This includes carriage returns which are explicitly embedded in the arguments (via CHR$(13)). These automatic line feeds may be inhibited by executing a GNOLF# command on the IEEE channel. Also, the EOI signal will be asserted during the transfer of the last byte for the command, unless a GNOBRK# command is in effect for the channel.

EXAMPLE: GPRINT# 5,D1$;XN;L$

will send the contents of the three data items to the device assigned to IEEE channel 5. The contents of D1$ will be immediately followed by the value in XN converted to a string, which is in turn followed by the contents of L$. A carriage return will be appended after the last character of L$. A line feed will follow the carriage return unless a GNOLF#5 command has been executed since opening IEEE channel.

NOTES: 1. If a GNOBRK command is in effect, the GPRINT# command will address the assigned device, if not already addressed. In addition, the GPRINT# will not unaddress the device after the data is sent.

2. The GPRINT# command returns the ST variable set appropriately. The value will equal the number of arguments output, minus 128 if EOI was sent. If the assigned device does not accept the data, 64 will be added to the current value of the ST variable and the GPRINT# command aborted.

4.1.1.11 The GCMD Command

PURPOSE: To send an interface message on the IEEE-488 bus.

SYNTAX: GCMD <byte> [, <byte>] ...

ARGUMENTS: <byte> = a numeric expression which evaluates to a number from 0 to 255.

DISCUSSION: The GCMD command is used to send interface messages (i.e. messages sent with the ATN line asserted) on the IEEE-488 bus. This allows BASIC programs to send commands such as group execute trigger (GET) or parallel poll configure (PPC), etc.

DISCUSSION: The GCMD will cause the ATN line to be asserted and then each byte to be sent on the bus while the ATN line remains asserted. When all the bytes have been sent, the ATN line will be released. All bytes specified will be sent to the bus without modification. It is the user's responsibility to make sure the bytes represent a valid message or command.

EXAMPLE: GCMD GPADR(4)+32,8,63

will send a GET command to the device assigned to IEEE channel 4. The first byte is the primary listen address of the device assigned to channel 4. The second byte is the group execute trigger (GET) message. The last byte is an unlisten (UNL) message to complete the interface message.

NOTES: 1. The GCMD command will set the ST variable to 0 if the iterface message is sent okay. It will be set to 64 if no acceptors are responding on the IEEE-488 bus.

4.2.2 THE IEEEL FUNCTIONS

4.2.2.1 The GPADR() Function

PURPOSE: To return the primary address of the device assigned to an IEEE channel.

SYNTAX: GPADR(<ieee chan>)

ARGUMENTS: <ieee chan> = a numeric expression which evaluates to a number from 0 to 9. This number will be used as the IEEE channel whose primary device address is returned.

DISCUSSION: The GPADR() function is used to obtain the primary device address of the device assigned to the specified IEEE channel. If the channel is not assigned, -1 is returned.

EXAMPLE: PA=GPADR(5)

will set the variable PA to the primary address of the device assigned to IEEE channel 5.

4.2.2.2 The GPPOLL() Function

PURPOSE: To return the result of a parallel poll on the IEEE-488 bus.

SYNTAX: GPPOLL(<dummy>)

ARGUMENTS: <dummy> = any numeric expression. This argument is not used in the execution of the function.

DISCUSSION: The GPPOLL() function is used to perform a parallel poll on the IEEE-488 bus. The value returned will be the results of the parallel poll. It may be necessary to use the GCMD command to send any necessary parallel poll configuration commands to may the poll operational.

EXAMPLE: PP=GPPOLL(0):IF PP>=128 THEN GOTO 5000:REM SERVICE SPECTROMETER

will perform a parallel poll and store the result in the variable PP. If the most significant bit in the byte is set, then execution jumps to line 5000.

NOTES: 1. To use the parallel poll feature, you must configure the data lines on the IEEE bus for open-collector output. Tri-state outputs must not be used. JP14 on the MULTI-0 board selects between open-collector and tri-state outputs. JP14 is normally insalled by the factory to select open-collector mode.

4.2.2.3 The GSADR() Function

PURPOSE: To return the secondary address of the device assigned to an IEEE channel.

SYNTAX: GSADR(<ieee chan>)

ARGUMENTS: <ieee chan> = a numeric expression which evaluates to a number from 0 to 9. This number will be used as the IEEE channel whose secondary device address is returned.

DISCUSSION: The GSADR() function is used to obtain the secondary device address of the device assigned to the specified IEEE channel. If the channel is not assigned or no secondary address is being used with that channel, -1 is returned.

EXAMPLE: SA=GSADR(5)

will set the variable SA to the secondary address of the device assigned to IEEE channel 5.

4.2.2.4 The GSPOLL Function

PURPOSE: To return the result of a serial poll of the specified device.

SYNTAX: GSPOLL(<pri. addr> [, <sec. addr>])

ARGUMENTS: <pri. addr> = a numeric expression which evaluates to a number from 0 to 30. This number will be used as the primary address of the device with which the channel will communicate.
<sec. addr> = a numeric expression which evaluates to a number from 0 to 30. This number will be used as the secondary address within the device with which the channel will communicate.

DISCUSSION: The GSPOLL() function is used to determine if a specific device is issuing a service request. The function will return the result of the serial poll of the specified device, and optional secondary address. An unlisten (UNL) command will be sent just prior to the serial poll to insure that no devices misinterpret the status bytes returned by the poll.

EXAMPLE: GSPOLL(2,15)

will return the result of a serial poll of IEEE device 2 and secondary address 15.

NOTES: 1. There is no time limit within which a device must respond with its status. The IEEEL will wait forever for the device to respond, even if it isn't present in the system. If the device is failing to respond, or a non-existent device is accidentally being accessed, you may abort the GSPOLL() function by pressing the BREAK key. The value returned by the function in this case will be 0.

4.2.2.5 The GSRQ() Function

PURPOSE: To indicate if a service request (SRQ) is being issued on the IEEE-488 bus.

SYNTAX: GSRQ(<dummy>)

ARGUMENTS: <dummy> = any numeric expression. This argument is not used in the execution of the function.

DISCUSSION: The GSRQ() function is used to determine if there is a device on the IEEE-488 bus which is requesting service via the SRQ line. This function returns a value of 1 if service is being requested, 0 if not.

EXAMPLE: IF GSRQ(0) THEN GOTO 5000:REM SERVICE SRQ REQUEST

will jump execution to line 5000 if service is being requested or pass execution through to the next line if not.

4.3 CLOCK/CALENDAR ASSEMBLY SUBROUTINE PACKAGE

The following subroutines may be found in the file called DATETIMESUBS.A on the MultI-O distribution disk. In order to use these subroutines, the source code should be copied from the DATETIMESUBS.A file into the user's program. The subroutines are completely self-contained and do not use any page-zero storage except for the pseudo registers as noted. They do expect however for the MTU130EQU.A file to have been .READ at the beginning of the user's program.

4.3.1 READDAT

PURPOSE: To read the 9 character date from the clock/calendar.

ARGUMENTS: U6 = Address of a buffer to receive the date.
Y = Displacement into the buffer to start storing the date.

ARGUMENTS RETURNED: 9 character date starting at (U6),Y in form of: DD-MMM-YY
CY set if date is from the calendar, clear if from CODOS.
Y is incremented by 9, A and X are preserved.

DESCRIPTION: READDAT converts the date in the clock/calendar registers into a 9 character string. This string is stored into what is typically the user's output line buffer in a format ready to print. Y is the index into the buffer and is incremented by 9 on return in preparation for additional output into the buffer. The DD field has a leading 0 if necessary and the MMM field has one of the following: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC. If the clock/calendar is not present, the date is returned from CODOS's 9 character date. Interrupts are disabled for approximately 500uS when this subroutine is called.

4.3.2 READTIM

PURPOSE: To read the 8 character time from the clock/calendar.

ARGUMENTS: U6 = Address of a buffer to receive the time.
Y = Displacement into the buffer to start storing the time.

ARGUMENTS RETURNED: 8 character time starting at (U6),Y in form of: HH:MM:SS
CY set if the clock is present, cleared if not.
Y is incremented by 8, A and X are preserved.

DESCRIPTION: READTIM converts the time in the clock/calendar registers into an 8 character string. This string is stored into what is typically the user's output line buffer in a format ready to print. Y is the index into the buffer and is incremented by 8 on return in preparation for additional output into the buffer. Each field has a leading 0 if necessary. If the clock is not present, the time is returned as 88:88:88 and the carry flag is cleared. Interrupts are disabled for approximately 500uS when this subroutine is called.

4.3.3 ABSTIME

PURPOSE: To convert a given date and time into seconds since 01-JAN-60 00:00:00.

ARGUMENTS: U4 = Address of date and time string formatted as: DD-MMM-YYHH:MM:SS

ARGUMENTS RETURNED: UO, U1 = 32 bit absolute time in seconds since 01-JAN-60, U1 is most significant.
CY set if conversion is successful, clear if not.
A, X, Y, UO-U3 U3 are used, U4 is preserved.

DESCRIPTION: This subroutine is useful for comparing two dates and times or for computing the difference between two dates and times. The input string must have both the date and time in the indicated format. There may be zero or more blanks separating the 9 character date from the 8 character time. The number fields should use leading zeros when the value is less than 10. The MMM field should use abbreviations from the following list: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC. The result will always be positive with 31 significant bits and will be valid for any date until the year 2028.

4.4 ANALOG I/O ASSEMBLY SUBROUTINE PACKAGE

The following subroutines may be found in the file called ADCDACSUBS.A on the MultI-O distribution disk. In order to use these subroutines, the source code should be copied from the ADCDACSUBS.A file into the user's program. Each subroutine is completely self-contained and does not use any page-zero storage except for the pseudo registers as noted. They do expect however for the MTU130EQU.A and MACLIB6502.A files to have been .READ at the beginning of the user's program and will require MACASM in order to assemble.

4.4.1 ADCINIT

PURPOSE: To set up the MultI-O internal 6522 prior to using the A-to-D converter.

ARGUMENTS: None

ARGUMENTS RETURNED: Uses A, preserves X and Y.

DESCRIPTION: This subroutine must be called prior to using the ADC. One call at the beginning of the user's program is sufficient. The subroutine simply sets the low 4 bits of port B of the internal 6522 to outputs.

4.4.2 READADC

PURPOSE: To read a single value from a specified ADC channel.

ARGUMENTS: A = channel number to read, 0-7

ARGUMENTS RETURNED: X = upper 8 bits of the ADC value
A = lower 4 bits of ADC value left justified
Y is preserved

DESCRIPTION: This subroutine reads the designated ADC channel and returns the 12 bit result in X (high) and A. Execution time is 93uS including call and return. ADCINIT must have been called prior to using READADC.

4.4.3 BLKADCX

PURPOSE: To read a block of multi-channel ADC data, external clock controlled.

ARGUMENTS: U1 = Address of a buffer to receive the data read
U2 = The total number of samples to read
U3 = Address of channel number vector (see description below)
A = Length of channel number vector

ARGUMENTS RETURNED: Data stored in buffer pointed to by U1
A, X, Y, U0-U2, low half of U4 used, U3 preserved

DESCRIPTION: This subroutine will read a block of multi-channel ADC data into a user supplied buffer under control of an external clock signal on AI0CB1. As each value is read, it is stored sequentially into the buffer pointed to by U1, high byte first. U2 gives the total number of samples to read before returning.

The channel number vector contains the sequence of ADC channels to read. The channel number occupies bits 1 (low) through 3, i.e., is multiplied by 2. If bit 7 of the channel number is zero, the routine will wait for a sample pulse before continuing after that channel is read. If bit 7 is a one, the routine will immediately advance to the next channel. The routine will always wait for a sample pulse when the vector is exhausted and wraparound to the beginnign occurs.

The time required to read a channel and advance to the next is 110uS. Therefore the time between sample pulses must be 110uS times the maximum number of channels read in a burst. A test loop is used to detect pulses on AI0CB1 therefore there is a 7uS uncertainty in the delay between the sample pulse and actual reading of the ADC. Both the data buffer and the channel number vector are in the current data bank.

4.4.4 BLKADCT

PURPOSE: To read a block of multi-channel ADC data, T1 timer controlled.

ARGUEMNTS: U0 = Time between samples in uS-2, max=$FFFF, see below for minimum.
U1 = Address of a buffer to receive the data read
U2 = The total number of samples to read
U3 = Address of channel number vector (see description below)
A = Length of channel number vector

ARGUMENTS RETURNED: Data stored in buffer pointed to by U1
A, X, Y, U0-U2, low half of U4 used, U3 preserved

DESCRIPTION: This subroutine is similar to BLKADCX described in section 4.4.3 except that an internal timer (internal 6522 T1) is used to generate the sample pulses. The time required to read a channel and advance to the next is 110uS. Therefore the minimum time between sample pulses must be 110uS times the maximum number of channels read in a burst. A test loop is used to detect timeouts therefore there is a 7uS uncertainty in the delay between timeout and actual reading of the ADC. The BLKADCL subroutine can be used for high-speed sampling where this jitter may be unacceptable. Both the data buffer and the channel number vector are in the current data bank.

4.4.5 BLKADCL

PURPOSE: To read a block of multi-channel ADC data, timed loop controlled.

ARGUEMNTS: Y = Time between samples, see description below for formula.
U1 = Address of a buffer to receive the data read
U2 = The total number of samples to read
U3 = Address of channel number vector (see description below)
A = Length of channel number vector

ARGUMENTS RETURNED: Data stored in buffer pointed to by U1
A, X, Y, U0-U2, low half of U4 used, U3 preserved

DESCRIPTION: This subroutine is similar to BLKADCT described in section 4.4.4 except that a timed loop is used to control the timing of the sample pulses. The time required to read a channel and advance to the next is 107uS. When all of the channels in a burst have been read, an additional wait of 5*YuS is performed before another burst is started. Therefore the timing is uniform only when the number of channels in each burst is constant. BLKADCT can be used when uniform timing of variable length bursts is required and where a 7uS jitter in the exact sample times is acceptable. Both the data buffer and the channel number vector are in the current data bank. If Y=0 then 256 is assumed.

4.4.6 BLKADCS

PURPOSE: To read a block of single-channel ADC data, timed loop controlled.

ARGUMENTS: A = Channel number to read from
Y = Time between samples, see description below for formula.
U1 = Address of a buffer to receive the data read
U2 = The total number of samples to read

ARGUMENTS RETURNED: Data stored in buffer pointed to by U1
A, X, Y, U0-U2 used

DESCRIPTION: This subroutine is similar to BLKADCL described in section 4.4.5 except that only a single channel is read. The time between samples from the designated channel is equal to 80+5*YuS. The advantage of this subroutine over BLKADCL is that a higher sampling rate is possible when only a single channel is being sampled. The data buffer is in the current data bank. If Y=0 then 256 is assumed.

4.4.7 WRITDAC

PURPOSE: To write a single value to the D-to-A converter.

ARGUMENTS: X = Upper 8 bits of 12 bit DAC word
A = Lower 4 bits of 12 bit DAC word left justified

ARGUMENTS RETURNED: None, all registers preserved

DESCRIPTION: This subroutine will store a single value into the DAC and return. Total execution time including call and return is 20uS. The lower 4 bits of A are ignored.

4.4.8 BLKDACX

PURPOSE: To output a block of data to the D-to-A converter with an external clock.

ARGUMENTS: U1 = Address of buffer to read the data from
U2 = Number of buffer words to output (non-zero)

ARGUMENTS RETURNED: None, A, X, Y, U1, and U2 are used

DESCRIPTION: This subroutine will output a sequence of voltage values taken from a user supplied buffer at the rate of one value for each pulse seen on AIOCB1. The data is organized in the buffer high byte first and the 12 bits are left justified. The buffer resides in the current data bank. The maximum input frequency is 18KHz (55uS period). A test loop is used to detect pulses on AIOCB1 which introduces a 7uS uncertainty in the delay between an AIOCB1 pulse and the exact update time of the DAC output voltage.

4.4.9 BLKDACT

PURPOSE: To output a block of data to the D-to-A converter with an internal clock.

ARGUMENTS: U0 = The update interval in microseconds-2, 55 - 65535
U1 = Address of buffer to read the data from
U2 = Number of buffer words to output (non-zero)

ARGUMENTS RETURNED: None, A, X, Y, U1, and U2 are used, U0 is preserved.

DESCRIPTION: This subroutine is similar to BLKDACX described in section 4.4.8 except that an internal timer (Internal 6522 T1) is used to control the output timing. A test loop is used to detect timeouts which introduces a 7uS uncertainty in the delay between an AIOCB1 pulse and the exact update time of the DAC output voltage. The BLKDACL subroutine can be used for high-speed output where this jitter may be unacceptable. The data buffer resides in the current data bank.

4.4.10 BLKDACL

PURPOSE: To output a block of data to the D-to-A converter, timed loop controlled.

ARGUMENTS: Y = The update interval = 5*Y+49uS
U1 = Address of buffer to read the data from
U2 = Number of buffer words to output (non-zero)

ARGUMENTS RETURNED: None, A, X, Y, U1, and U2 are used, U0 is preserved.

DESCRIPTION: This subroutine is similar to BLKDACT described in section 4.4.9 except that a timed loop is used to control the output timing. There is therefore no jitter in the output timing provided that interrupts are not occuring. For output rates slower than 750Hz (1329uS), BLKADCT may be used. The data buffer resides in the current data bank.

4.5 IEEE-488 ASSEMBLY SUBROUTINE PACKAGE

The file IEEESUBS.A found on the MultI-O Distribution disk contains an extensive set of subroutines to simplify the develompent of machine language programs which must access the IEEE-488 bus, either as a controller or a device. The subroutines in this package are divided into 3 catagories, low-level, mid-level, and high-level subroutines. Only the high-level subroutines are documented in this manual since these are subroutines which implement complete functions. Documentation for all the subroutines may be found in the source file. In fact, over 50% of the source file consists of descriptions and comments. In addition, the source file contains an example of both a controller and a device application. This application was used to exercise each subroutine to test for correct operation.

Most of the subroutines in the package may also be divided into subroutines which support controller functions and those which support device functions. The subroutines which support controller functions allow machine language programs to configure the MTU-130 as the IEEE-488 bus controller. The subroutines which support device functions allow the MTU-130 act as a device on the IEEE-488 bus. Naturally use of the routines from the two groups is mutually exclusive. The MTU-130 may be configured as the bus controller or a device, but not both simultaneously. Also, the approach to using these routines differs for the two groups. If you are using the controller support subroutines, you would write the main program which would call the appropriate subroutines in this package. If you are using the device support subroutines, you have to modify the subroutines in this package to call your routines at the appropriate times to make the device respond as required.

The 9914A IEEE-488 Interface chip will be configured to not generate any system interrupts when used with this subroutine package. However, the interrupt bits in these registers are used extensively to determine when certain events have occured. If an interrupt bit becomes set which is not expected, these subroutines will jump to the error handling code in the package. As implemented, an error number is generated and displayed along with the location of the JSR instruction to the high-level subroutine. Following this will be the contents of the two interrupt flag registers at the time of the error. Finally, the contents of the Addressed Status register in the 9914A will be displayed. At this point the program returns to CODOS. This process will be referred to as an "error abort" in the descriptions which follow.

The routines in this package make no use of page zero except for the pseudo registers as indicated. Access to these subroutines may be achieved by including the source code in your source file, or via the jump tables provided in the object file. This object file is obtained by simply assembling the source file without modification. If you choose to include the source in your assembly language source file, you may wish to delete some of unused subroutines to reduce the size of the file.

If you wish to use the jump table, the following is a summary of the entry points in this table:

ADDR	NAME	DESCRIPTION
$900	C_STRTUP	Startup controller operation.
$903	C_ASSIGN	Assign IEEE channel to device and secondary address.
$906	C_FREE	Free IEEE channel.
$909	C_OUM	Output inline message over IEEE channel.
$90C	C_INB	Input byte from IEEE channel.
$90F	C_OUB	Output byte over IEEE channel.
$912	C_INL	Input line from IEEE channel.
$915	C_OUL	Output line over IEEE channel.

ADDR	NAME	DESCRIPTION
$918	C_OUS	Output string over IEEE channel.
$91B	C_INR	Input record from IEEE channel.
$91E	C_OUR	Output record over IEEE channel.
$921	C_SERPOL	Perform serial poll of assigned IEEE channels.
$924	C_PARPOL	Perform parallel poll of IEEE bus.
$927	C_PPCONF	Perform parallel poll configure on IEEE channel.
$92A	C_OUIMB	Output interface message byte (i.e. command byte).
$980	D_STRTUP	Startup device operation.
$983	D_OUM	Output inline message to IEEE-488 bus.
$986	D_INB	Input byte from IEEE-488 bus.
$989	D_OUB	Output byte to IEEE-488 bus.
$98C	D_INL	Input line from IEEE-488 bus.
$98F	D_OUL	Output line to IEEE-488 bus.
$992	D_OUS	Output string to IEEE-488 bus.
$995	D_INR	Input record from IEEE-488 bus.
$998	D_OUR	Output record to IEEE-488 bus.
$99B	D_TALK	Handle device acting as talker.
$99E	D_LISTEN	Handle device acting as listener.
$9A1	D_LOOP	Main device handler loop.

4.5.2 The Controller Subroutines

The following subroutines are provided for applications where the MTU-130 is to act as the IEEE-488 bus controller. In this situation, you must write the main program which will call these subroutines to perform the appropriate functions. Specifying the source or destination on the IEEE-488 bus is done by assigning the desired device and optional secondary address to an IEEE channel. Once assigned, that device and secondary address may be referenced by passing the associated channel number in the X register, the same as in CODOS SVC operation. In fact, most of the subroutines perform a function similar to one of the CODOS SVC's. Please note that IEEE channels are distinct from CODOS channels except in concept.

There two flag locations which are used by these subroutine to specify two details of their operation. These are the LF.FLAG location and the EOI.FLAG location. Their effect is enabled by setting bit 7 of the associated location to 1. Also, their effect differs slightly depending whether an input or output operation is being performed. The input and output operations are described briefly below. The effect of LF.FLAG and EOI.FLAG locations are described in each of these.

An input operation involves addressing the desired device to be the "talker", inputing the data received from this device, then unaddressing the device once the last byte is received. The last byte is indicated by the presence of the EOI signal during the transfer of the byte. All controller subroutines which input data will automatically address the device to talk, if the device is not already addressed. (Note: An error abort will occur if some other device is already addressed, or the same device is currently addressed to listen.) These subroutines will then input the amount of data determined by their function. If EOI is encountered during the input, they will automatically unaddress the device and return CY=1 to indicate EOI was received. If EOI is not received before the desired amount of input is obtained, the device will be left addressed so that another input subroutine may be called to continue the input.

EOI.FLAG has no effect on the input subroutines. However, LF.FLAG will cause an extra character to be read following any carriage return which is read. If the following character is not a line feed, an error abort occurs. Otherwise, the line feed is discarded.

An output operation works virtually the same as the input operation except that data is sent rather than received. Since the MTU-130 is doing the sending, it will control the sending of the EOI signal with the last byte. All controller subroutines which output data will automatically address the device to listen, if the device is not already addressed. (Note: An error abort will occur if some other device is already addressed, or the same device is currently addressed to talk.) These subroutines then output the data as determined by their function. Whether or not EOI is sent during the last byte is controlled by the state of EOI.FLAG.

LF.FLAG during output operations will cause line feeds to be sent automatically after each carriage return output. EOI.FLAG determines if the EOI signal should be sent with the last byte sent by the subroutine. If the flag is set, EOI will be sent and the transmission terminated by sending an unlisten command. If not set, output is continued by calling additional output subroutines. Eventually, EOI.FLAG must be set prior to calling one of the output subroutines to terminate the transmission.

4.5.1.1 C_STRTUP

PURPOSE: To initialize MTU-130 as the IEEE-488 bus controller.

ARGUMENTS: None

ARGUMENTS RETURNED: CY = 1 if 9914A in active state.
SAV.INT0 and SAV.INT1 = last contents of ISTAT0 and ISTAT1.
Z = 1 if neither INT0 nor INT1 bits set in SAV.IST0.
N = 1 if INT0 bit set in SAV.IST0.
V = 1 if INT1 bit set in SAV.IST1.
A, X, and Y are preserved.

DESCRIPTION: The C_STRUP subroutine is used to startup the MTU-130 as the IEEE-488 bus controller. This involves resetting the 9914A, then configuring it to act as the bus controller. In this process, the IFC signal will be asserted for approximately 100 microseconds. The 9914A is then brought to the active state, which is an entry requirement for most of the other controller subroutines. If this last step is successful, the BO interrupt bit in the ISTAT0 register should become set. If this is occurs, the carry flag is returned set indicating successful transfer to the active state. The remaining flags indicate the status of the other interrupt bits stored in SAV.IST0 and SAV.IST1. If some other interrupt is present, it is up to your routines to process or ignore them as required.

4.5.1.2 C_ASSIGN

PURPOSE: To assign a device and optional secondary address to an IEEE channel.

ARGUMENTS: A = Device number.
Y = Secondary address. If > 30, secondary addressing disabled.
X = IEEE channel.

ARGUMENTS RETURNED: CY = 1 if illegal device number.
A, X, and Y registers preserved.

DESCRIPTION: The C_ASSIGN subroutine is used to associate a device and optional secondary address with an IEEE channel. This operations simply store the device number and secondary address into the DEV.TBL and SADR.TBL for future reference. No action will occur on the IEEE-488 bus. The number of available channels is defined by NCHAN constant in the source code. It is initially set to 64, but may increased if additional table space is provided. The main reasons for using the concept of IEEE channels is that it allows the device and secondary address to be specified with one passed argument, plus it allows the I/O subroutines to use the same passed arguments as the matching CODOS SVCs.

```
EXAMPLE:    LDA     #2        ;DEVICE 2
            LDY     #10       ;SECONDARY ADDRESS 10
            LDX     #0        ;IEEE CHANNEL 0
            JSR     C_ASSIGN  ;ASSIGN CHANNEL 0 TO DEVICE 2, SEC. ADDR. 10
```

This section of code will assign IEEE channel 0 to device 2 and secondary address 10.

4.5.1.3 C_FREE

PURPOSE: To free an IEEE channel.

ARGUMENTS: X = IEEE channel to free.

ARGUMENTS RETURNED: CY = 1 if channel was not assigned.
A, X, and Y registers preserved.

DESCRIPTION: The C_FREE subroutine is used to free an IEEE channel. This simply frees the associated entries in the DEV.TBL and SADR.TBL. No action will occur on the IEEE-488 bus. CY is returned set if the channel was not previously assigned.

```
EXAMPLE:    LDX     #0        ;IEEE CHANNEL 0
            JSR     C_FREE    ;FREE IEEE CHANNEL 0
```

This section of code will free IEEE channel 0.

4.5.1.4 C_OUM

PURPOSE: To output an inline message over an IEEE channel.

ARGUMENTS: None

ARGUMENTS RETURNED: A, X, and Y registers preserved.
U4 used as pointer.

DESCRIPTION: The C_OUM subroutine is used to output an inline message. This subroutine works the same as SVC #2. The first byte following the JSR C_OUM is used as the IEEE channel. The null terminated string of bytes following the channel number will be output over that channel. Execution of object code will resume following the null terminator. The effect of LF.FLAG and EOI.FLAG will be as described for subroutines which perform output.

```
EXAMPLE:    LDA     #$80
            STA     LF.FLAG  ;ENABLE AUTO LINE FEEDS
            STA     EOI.FLAG ;SEND EOI AT END OF MESSAGE.
            JSR     C_OUM
            .BYTE   2,"OUTPUT THIS INLINE MESSAGE",$OD,O
            ...
```

This section of code will output the message shown to IEEE channel 2. Since the LF.FLAG is set, a line feed ($0A) will be sent following the carriage return ($0D) at the end of the message. Since EOI.FLAG is set, the EOI signal will be sent with the last byte of the message, which would be the line feed.

4.5.1.5 C_INB

PURPOSE: To input a byte from an IEEE channel.

ARGUMENTS: X = IEEE channel.

ARGUMENTS RETURNED: A = byte received.
CY = 1 if EOI received with byte.
X and Y registers preserved.

DESCRIPTION: The C_INB subroutine is used to input a byte from an IEEE channel. This subroutine works similarly to SVC #3. The carry flag will be set if EOI is received with the byte that was input. The effect of LF.FLAG will be as described for subroutines which perform input.

```
EXAMPLE:    LDX     #1
            LDY     #$FF      ;INIT INDEX
      LOOP  INY               ;INCREMENT BUFFER INDEX
            JSR     C_INB     ;GET A BYTE FROM IEEE CHANNEL 1
            STA     BUFFER,Y  ;STORE BYTE IN BUFFER
            BCC     LOOP      ;BR UNTIL LAST BYTE
            ...
    BUFFER  *=*+      256
```

This section of code will input bytes from IEEE channel 1 and store them into a buffer. This will continue until until EOI is received. Note that this routine assumes that EOI will occur before the Y index wraps back to 0, i.e., that the message is less than 256 bytes long.

4.5.1.6 C_OUB

PURPOSE: To output a byte to an IEEE channel.

ARGUMENTS: A = Byte to output.
X = IEEE channel.

ARGUMENTS RETURNED: A, X, and Y registers preserved.

DESCRIPTION: The C_OUB subroutine is used to output a single byte to an IEEE channel. This subroutine works similarly to SVC #4. The effect of LF.FLAG and EOI.FLAG will be as described for subroutines which perform output.

```
EXAMPLE:    LSR     EOI.FLAG ;CLEAR EOI.FLAG
            LDA     #'A'     ;OUTPUT AN 'A'
            LDX     #1       ;IEEE CHANNEL 1
            JSR     C_OUB
```

This section of code will output an "A" to IEEE channel 1. Since EOI.FLAG is clear, EOI will not be sent with this byte and the device will be left addressed.

4.5.1.7 C_INL

PURPOSE: To input a line from an IEEE channel.

ARGUMENTS: U5 = Pointer to input buffer.
X = IEEE channel.

ARGUMENTS RETURNED: A = Number of bytes input (not including carriage return).
Y = 0.
CY = 1 if EOI received.

DESCRIPTION: The C_INL subroutine is used to input a line from an IEEE channel. This subroutine works similarly to SVC #5. Bytes will be read from the channel and stored in the buffer until a carriage return is encountered. The carriage return will be stored in the buffer, but not included in the count returned in the A register. There is a limit to the number of characters which may be stored in the input buffer. This limit is determined by the value at MAXLN which will contain 192 if not set otherwise. The effect of LF.FLAG will be as described for subroutines which perform input.

```
EXAMPLE:    SEC
            ROR     LF.FLAG  ;SET LF.FLAG
            LDA     #<INBUF  ;SET POINTER TO INPUT BUFFER
            STA     U5
            LDA     #>INBUF
            STA     U5+1
            LDX     #2       ;IEEE CHANNEL 2
            JSR     C_INL    ;INPUT A LINE
            ...
     INBUF  *=*+    192      ;THE INPUT BUFFER
```

This section of code will input a line from IEEE channel 1. The line will be stored in INBUF. Since LF.FLAG is set, a line feed will be expected following the carriage return. This line feed will not be stored in the buffer.

4.5.1.8 C_OUL

PURPOSE: To output a line of characters to an IEEE channel.

ARGUMENTS: U6 = Pointer to line of text.
Y = Number of characters in the line.
X = IEEE channel.

ARGUMENTS RETURNED: A, X, and Y registers preserved.
U6 is unchanged.

DESCRIPTION: The C_OUL subroutine is used to output a line to an IEEE channel. This subroutine works similarly to SVC #6. It will output the first N characters of the string pointed to by the contents of U6, where N is the value in the Y register. After these characters are output, a carriage return is automatically appended. The effect of LF.FLAG and EOI.FLAG will be as described for subroutines which perform output.

```
EXAMPLE:    SEC
            ROR       EOI.FLAG ;SET EOI.FLAG
            LSR       LF.FLAG  ;CLEAR LF.FLAG
            LDA       #<STRING ;SET POINTER TO STRING
            STA       U6
            LDA       #>STRING
            STA       U6+1
            LDY       #STRLEN  ;GET THE STRING LENGTH
            LDX       #2       ;IEEE CHANNEL 2
            JSR       C_OUL    ;OUTPUT THE LINE
            ...
    STRING .BYTE    'THIS IS A STRING TO BE OUTPUT'
    STRLEN = *-STRING
```

This section of code will output the string shown followed by a carriage return to IEEE channel 2. Since LF.FLAG is clear, no line feed is sent after the carriage return. EOI will be sent with the carriage return, since EOI.FLAG is set.

4.5.1.9 C_OUS

PURPOSE: To output a string to an IEEE channel.

ARGUMENTS: U6 = Pointer to line of text.
Y = Number of characters in the line.
X = IEEE channel.

ARGUMENTS RETURNED: A, X, and Y registers preserved.
U6 is unchanged.

DESCRIPTION: The C_OUS subroutine is used to output a string of characters to an IEEE channel. This subroutine is identical to the C_OUL subroutine except that only the characters are sent. No carriage return will added to the output following the string. There is no restriction on placing carriage returns within the string, however.

```
EXAMPLE:    SEC
            ROR       LF.FLAG  ;SET LF.FLAG
            LSR       EOI.FLAG ;CLEAR EOI.FLAG
            LDA       #<STRING ;SET POINTER TO STRING
            STA       U6
            LDA       #>STRING
            STA       U6+1
            LDY       #STRLEN  ;GET THE STRING LENGTH
            LDX       #4       ;IEEE CHANNEL 4
            JSR       C_OUS    ;OUTPUT THE STRING
            ...
    STRING .BYTE    'THIS IS A STRING',$OD,'TO BE OUTPUT',$OD
    STRLEN = *-STRING
```

This section of code will output the string shown to IEEE channel 4. Since the LF.FLAG is set, the embedded carriage returns will be followed by line feeds. EOI will not be sent with the second line feed since EOI.FLAG is clear.

4.5.1.10 C_INR

PURPOSE: To input a record from an IEEE channel.

ARGUMENTS: U1 = Starting address of buffer for storing the record.
U2 = Number of bytes to read.
X = IEEE channel.

ARGUMENTS RETURNED: U1 = Address of last byte read, plus one.
U2 = Actual number of bytes read.
CY = 1 if EOI occurred with last byte.
A, X, and Y registers preserved.

DESCRIPTION: The C_INR subroutine is used to input a record from an IEEE channel. This subroutine works similarly to SVC #15. Bytes will be read from the IEEE channel and stored in the buffer until the number of bytes requested have been read, or EOI is received with a byte. The effect of LF.FLAG will be as described for subroutines which perform input.

```
EXAMPLE:    LSR     LF.FLAG  ;CLEAR LF.FLAG
            LDA     #<BUF    ;SET POINTER TO INPUT BUFFER
            STA     U1
            LDA     #>BUF
            STA     U1+1
            LDA     #<500    ;READ 500 BYTES
            STA     U2
            LDA     #>500
            STA     U2+1
            LDX     #3       ;IEEE CHANNEL 3
            JSR     C_INR    ;INPUT THE RECORD
            ...
        BUF *=*+    500
```

This section of code will try to read 500 bytes into buffer BUF from IEEE channel 3. Since LF.FLAG is clear, any line feeds which follow carriage returns will be stored in the buffer.

4.5.1.11 C_OUR

PURPOSE: To output a record to an IEEE channel.

ARGUMENTS: U1 = Starting address of buffer which contains the record.
U2 = Number of bytes to write.
X = IEEE channel.

ARGUMENTS RETURNED: A, X, and Y registers preserved.
U1 and U2 unchanged.

DESCRIPTION: The C_OUR subroutine is used to output a record to an IEEE channel. This subroutine works similarly to SVC #16. Bytes will be written from the buffer to the IEEE channel until the number of bytes specified have been written. The effect of LF.FLAG and EOI.FLAG will be as described for subroutines which perform output.

```
EXAMPLE:    LDA     #$80
            STA     LF.FLAG  ;SET LF.FLAG
            STA     EOI.FLAG ;SET EOI.FLAG
            LDA     #<BUF    ;SET POINTER TO BUFFER
            STA     U1
            LDA     #>BUF
            STA     U1+1
            LDA     #<500    ;WRITE 500 BYTES
            STA     U2
            LDA     #>500
            STA     U2+1
            LDX     #3       ;IEEE CHANNEL 3
            JSR     C_OUR    ;OUTPUT THE RECORD
            ...
       BUF  *=*+    500
```

This section of code will output 500 bytes from the buffer BUF to IEEE channel 3. Since LF.FLAG is set, line feeds will be added after any carriage return which is output. EOI will be sent with the last byte, since EOI.FLAG is set.

4.5.1.12 C_SERPOL

PURPOSE: To perform a serial poll of assigned IEEE channels.

ARGUMENTS: None

ARGUMENTS RETURNED: CY = 1 if a requesting device is found.
A = Status byte read.
X = IEEE channel.

DESCRIPTION: The C_SERPOL subroutine will perform a serial poll of the assigned IEEE channels until a requesting device is found, or all assigned channels are tried. The channels are scanned in ascending order starting with channel 0. If CY is returned clear, then the requesting device is not currently assigned to an IEEE channel. This subroutine expects the 9914A to be in the active state (.i.e. no devices left addressed from EOI.FLAG being clear). If it is not, an error abort occurs. This subroutine does not provide for determining if service has been requested by the SRQ signal. This may be detected in your program by checking the SRQ bit in the bus status (BUSTAT) register. The other alternative is to modify the controller mid-level routines to act on SRQ interrupts when they are detected.

EXAMPLE:

```
   SRQ.BIT =    $04
            LDA     BUSTAT   ;GET BUS STATUS
            AND     #SRQ.BIT
            BEQ     NOSRQ    ;BR IF NO REQUEST
            JSR     C_SERPOL ;PERFORM SERIAL POLL
```

This section of code will check if the SRQ bit is the bus status register is high. If it is, a serial poll is performed.

4.5.1.13 C_PARPOL

PURPOSE: To perform a parallel poll of the IEEE-488 bus.

ARGUMENTS: None.

ARGUMENTS RETURNED: A = Results of parallel poll.
X and Y registers preserved.

DESCRIPTION: The C_PARPOL subroutine is used to perform a parallel poll of the IEEE-488 bus. This action does not involve IEEE channels in any way. In order to use parallel polling, the data lines on the IEEE-488 bus must be configured for open-collector outputs (this applies to the devices connected to the IEEE-488 bus).

```
EXAMPLE:    DEC       COUNT
            BNE       NOTYET    ;BR IF NOT TIME FOR PARALLEL POLL
            JSR       C_PARPOL  ;DO PARALLEL POLL
```

This section of code performs a parallel poll of the IEEE-488 bus provide the value in COUNT reaches zero.

4.5.1.14 C_PPCONF

PURPOSE: To send a parallel poll configuration byte (PPE) an IEEE channel.

ARGUMENTS: A = Configuration byte. Bit 3 = Sense, bits 2 - 0 = bit number.
X = IEEE channel.

ARGUMENTS RETURNED: A, X, and Y registers preserved.

DESCRIPTION: The C_PPCONF is used to send a parallel poll configure message to an IEEE channel. The lower four bits passed in the A register will be used to form the parallel poll enable (PPE) byte sent to the device. A parallel poll unconfigure command may be sent using the C_OUIMB subroutine.

```
EXAMPLE:    LDA       #$041     ;RESPONDE WITH BIT 1 HIGH WHEN REQUESTING
            LDX       #3        ;IEEE CHANNEL 3
            JSR       C_PPCONF  ;CONFIGURE DEVICE ASSIGNED TO CHANNEL 3
```

This section of code will configure the device assigned to IEEE channel 3 to respond to a parallel poll by setting bit 1 high when service is desired.

4.5.1.15 C_OUIMB

PURPOSE: To output an interface message byte (i.e. a command byte).

ARGUMENTS: A = Byte to output.

ARGUMENTS RETURNED: A, X, and Y registers preserved.

DESCRIPTION: The C_OUIMB subroutine is used to output an interface message byte. An interface message refers to bytes which are output while the ATN signal is true. These interface messages represent commands, such are Group Execute Trigger (GET) which are sent on the IEEE-488 bus to each device. Depending on circumstances, each device might or might not be required to act on the command. It may be desirable to combine the use of this subroutine with other mid-level subroutines to form useful functions. For details on the mid-level subroutines, refer to the source code. For proper operation, the 9914A must be in the controller active state when the C_OUIMB subroutine is called.

EXAMPLE:

```
LLO =       $11
    LDA     #LLO      ;GET LOCAL LOCKOUT COMMAND
    JSR     C_OUIMB   ;OUTPUT THE COMMAND
```

This section of code will set a local lockout interface message on the IEEE-488 bus. All devices attached to the bus should disable any front panel switch which would return device control to the front panel.

4.5.2 The Device Subroutines

The following subroutines are provided for applications where the MTU-130 is to act as an IEEE-488 device. This situation differs substantially from applications where the MTU-130 is the controller. In device applications, these subroutines must respond to events on the IEEE-488 bus rather that direct them. The result of this difference is that included among the device subroutines, there is a main program loop routine, called D_LOOP, to which you may interface your device dependent routines. For example, once the 9914A has been configured as a device, this main program loop will handle the overhead of detecting when the device has become addressed. It is then up to your routines to send or receive the appropriate data for the application. Control is then returned to the main loop to continue device operation.

To interface your routines, you will have to modify various high-level and possibly some mid-level subroutines to call your device dependent routines at the appropriate places. The most likely places where you might wish to call your routines are already designated. High-level input/output routines may be called by your device dependent routines to perform the actual transfer of data. In any event, implementing a device application will require a thorough study of the assembly language source code to fully understand what must be done.

The occurance of various events on the IEEE-488 bus are detected via interrupt bits in the ISTAT0 and ISTAT1 registers on the 9914A chip. These interrupts are used by each of the high-level routines to determine what step to take next. These interrupt bits are read strictly by software and are not used to generate any actual MTU-130 bus interrupts from the 9914A chip.

When a normal sequence of events occurs, with associated interrupts, execution will be stepped through the device subroutines to provide the proper responses. Provision is made for certain other interrupts to be detected which may not be part of a normal sequence. If the interrupt might occur during normal operation, special handling may be provided, or a place designated for the user to add his own handling. There may be other interrupts for which there is no defined proper action or for which handling is not provided. In these cases an error abort will occur. As implemented, the error abort will result in an error display which shows the contents of the ISTAT0 and ISTAT1 registers which contained the unexpected interrupt, along with the contents of the address status register (ADSTAT).

Of particular importance are the BO and BI interrupts. These indicate when the 9914A chip is ready to output or input the next byte, respectively. When outputing a data byte, these subroutines will aways wait for the BO interrupt to occur before proceeding. When inputting a data byte, these subroutines will normally wait for a BI interrupt to occur, then input the byte. In some cases, it may be necessary to detect the BI interrupt prior to executing the subroutine to input the byte. In this case, the input subroutine should not wait for the BI interrupt. Such a case is always indicated to the input subroutine by the CY flag being set. If the CY flag is clear, the input subroutine will wait to for a BI interrupt to occur.

The device subroutines also use the LF.FLAG and EOI.FLAG locations to specify two details of their operation. Their effect is enabled by setting bit 7 of the associated location to 1. Again, their effect differs slighly depending whether an input or output operation is being performed. On input, setting LF.FLAG will cause line feeds to be expected following carriage returns. An error abort occurs if the line feed is expected and some other character is received following a carriage return. On output, setting EOI.FLAG will cause line feeds to be sent automatically after any carriage return which is output. Also on output, setting LF.FLAG will cause the EOI signal to be sent true with the last byte sent by the output subroutine.

4.5.2.1 D_STRTUP

PURPOSE: To initialize MTU-130 as an IEEE-488 bus device.

ARGUMENTS: A = IEEE device address.
U1 = Pointer to table of secondary addresses.
High byte of U1 = 0 for no secondary address, otherwise
table is terminated by a byte > 30.

ARGUMENTS RETURNED: CY = 1 if device address is illegal.
A, X, and Y registers preserved.

DESCRIPTION: The D_STRUP subroutine is used to startup the MTU-130 as an IEEE-488 bus device. This involves resetting the 9914A, then configuring it to act as a bus device. In this process, the bus address for the device will be established by the value passed in the A register. If the high byte of pseudo register U1 is not zero, then it is expected to point to a table of secondary addresses. These secondary addresses will be stored in an internal table which will be used to recognize when this device becomes addressed. This table is terminated by a byte which is greater than 30. If the high byte of U1 is zero, no secondary addressing is enabled.

```
EXAMPLE:    LDA     #<SA_TBL ;SET POINTER TO TABLE
            STA     U1
            LDA     #>SA_TBL
            STA     U1+1
            LDA     #4       ;USE DEVICE ADDRESS 4
            JSR     D_STRTUP ;STARTUP THE DEVICE
            ...
   SA_TBL   .BYTE   1,10,3,$FF
```

This section of code will startup the MTU-130 as device number 4 and will respond to secondary addresses 1, 3, and 10.

4.5.2.2 D_OUM

PURPOSE: To output an inline message to the IEEE-488 bus.

ARGUMENTS: None

ARGUMENTS RETURNED: A, X, and Y registers preserved.
U4 used as pointer.

DESCRIPTION: The D_OUM subroutine is used to output an inline message to the IEEE-488 bus after the device has been addressed as the talker. This subroutine works similarly to SVC #2. There is one important difference, however. No output channel is required, so the inline message begins immediately following the JSR C_OUM. The inline message should consist of a string of characters terminated by a null character (a zero byte). Execution of object code will resume following the null terminator. The effect of LF.FLAG and EOI.FLAG will be as described for subroutines which perform output.

```
EXAMPLE:    LDA     #$80
            STA     LF.FLAG  ;ENABLE AUTO LINE FEEDS
            STA     EOI.FLAG ;SEND EOI AT END OF MESSAGE.
            JSR     D_OUM
            .BYTE   2,"OUTPUT THIS INLINE MESSAGE",$0D,0
```

This section of code will output the message shown. Since the LF.FLAG is set, a line feed ($0A) will be sent following the carriage return ($0D) at the end of the message. Since EOI.FLAG is set, the EOI signal will be sent will the last byte of the message, which would be the line feed.

4.5.2.3 D_INB

PURPOSE: To input a byte from the IEEE-488 bus.

ARGUMENTS: CY=1 if BI interrupt has been previously detected.

ARGUMENTS RETURNED: A = byte received.
CY = 1 if EOI received with byte.
X and Y registers preserved.

DESCRIPTION: The D_INB subroutine is used to input a byte from the IEEE-488 bus after the device has been addressed as a listener. This subroutine works similarly to SVC #3. Special attention must be given to the state of the CY flag before calling this subroutine. If it is in the wrong state, the program will hang, or an invalid byte input. The carry flag will be set on return if EOI is received with the byte that was input. The effect of LF.FLAG will be as described for subroutines which input.

```
EXAMPLE:
            LDY     #$FF      ;INIT INDEX
       LOOP INY               ;INCREMENT BUFFER INDEX
            JSR     D_INB     ;GET A BYTE FROM IEEE CHANNEL 1
            STA     BUFFER,Y  ;STORE BYTE IN BUFFER
            BCC     LOOP      ;BR UNTIL LAST BYTE, CY = 0 SO WAIT FOR NEW BI IRQ
            ...
     BUFFER *=*+    256
```

This section of code will input bytes the IEEE-488 bus and store them into a buffer. This will continue until until EOI is received. Note that this routine assumes that EOI will occur before the Y index wraps back to 0, i.e., that the message is less than 256 bytes long.

4.5.2.4 D_OUB

PURPOSE: To output a byte to the IEEE-488 bus.

ARGUMENTS: A = Byte to output.

ARGUMENTS RETURNED: A, X, and Y registers preserved.

DESCRIPTION: The D_OUB subroutine is used to output a single byte to the IEEE-488 bus after the device has been addressed as the talker. This subroutine works similarly to SVC #4. The device should already be addressed to talk before calling this routine. The effect of LF.FLAG and EOI.FLAG will be as described for subroutines which output.

```
EXAMPLE:    LSR     EOI.FLAG ;CLEAR EOI.FLAG
            LDA     #'A'     ;OUTPUT AN 'A'
            JSR     D_OUB
```

This section of code will output an "A" to the IEEE-488 bus. Since EOI.FLAG is clear, EOI will not be sent with this byte.

4.5.2.5 D_INL

PURPOSE: To input a line from the IEEE-488 bus.

ARGUMENTS: U5 = Pointer to input buffer.
CY = 1 if BI interrupt has been previously detected.

ARGUMENTS RETURNED: A = Number of bytes input (not including carriage return).
Y = 0.
CY = 1 if EOI received.

DESCRIPTION: The D_INL subroutine is used to input a line from the IEEE-488 bus after the device has been addressed to listen. This subroutine works similarly to SVC #5. Special attention must be given to the state of the CY flag before calling this subroutine. If it is in the wrong state, the program will hang, or an extra byte read at the beginning. The effect of the CY flag only applies to the first byte read. Bytes will be read from the IEEE-488 bus and stored in the buffer until a carriage return is encountered. The carriage return will be stored in the buffer, but not included in the count returned in the A register. There is a limit to the number of characters which may be stored in the input buffer. This limit is determined by the value at MAXLN which will contain 192 if not set otherwise. The effect of LF.FLAG will be as described for subroutines which perform input.

```
EXAMPLE:    LDA     #$80     ;NOTE: ASSUME CY SET IF BI ALREADY DETECTED
            STA     LF.FLAG  ;SET LF.FLAG
            LDA     #<INBUF  ;SET POINTER TO INPUT BUFFER
            STA     U5
            LDA     #>INBUF
            STA     U5+1
            JSR     D_INL    ;INPUT A LINE
```

```
          ...
INBUF     *=*+      192       ;THE INPUT BUFFER
```

This section of code will input a line from the IEEE-488 bus. The line will be stored in INBUF. Since LF.FLAG is set, a line feed will be expected following the carriage return. This line feed will not be stored in the buffer.

4.5.2.6 D_OUL

PURPOSE: To output a line of characters to the IEEE-488 bus.

ARGUMENTS: U6 = Pointer to line of text.
Y = Number of characters in the line.

ARGUMENTS RETURNED: A, X, and Y registers preserved.
U6 is unchanged.

DESCRIPTION: The C_OUL subroutine is used to output a line to the IEEE-488 bus after the device has been addressed as the talker. This subroutine works similarly to SVC #6. It will output the first N characters of the string pointed to by the contents of U6, where N is the value in the Y register. After these characters are output, a carriage return is automatically appended. The effect of LF.FLAG and EOI.FLAG will be as described for subroutines which output.

EXAMPLE:

```
          SEC
          ROR       EOI.FLAG ;SET EOI.FLAG
          LSR       LF.FLAG  ;CLEAR LF.FLAG
          LDA       #<STRING ;SET POINTER TO STRING
          STA       U6
          LDA       #>STRING
          STA       U6+1
          LDY       #STRLEN  ;GET THE STRING LENGTH
          JSR       D_OUL    ;OUTPUT THE LINE
          ...
   STRING .BYTE     'THIS IS A STRING TO BE OUTPUT'
   STRLEN =         *-STRING
```

This section of code will output the string shown followed by a carriage return to the IEEE-488 bus. Since LF.FLAG is clear, no line feed is sent after the carriage return. EOI will be sent with the carriage return, since EOI.FLAG is set.

4.5.2.7 D_OUS

PURPOSE: To output a string to the IEEE-488 bus.

ARGUMENTS: U6 = Pointer to line of text.
Y = Number of characters in the line.

ARGUMENTS RETURNED: A, X, and Y registers preserved.
U6 is unchanged.

DESCRIPTION: The D_OUS subroutine is used to output a string of characters to the IEEE-488 bus after the device has been addressed as the talker. This subroutine is identical to the D_OUL subroutine except that only the characters are sent. No carriage return will added to the output following the string. There is no restriction on placing carriage returns within the string, however.

```
EXAMPLE:    SEC
            ROR       LF.FLAG  ;SET LF.FLAG
            LSR       EOI.FLAG ;CLEAR EOI.FLAG
            LDA       #<STRING ;SET POINTER TO STRING
            STA       U6
            LDA       #>STRING
            STA       U6+1
            LDY       #STRLEN  ;GET THE STRING LENGTH
            JSR       D_OUS    ;OUTPUT THE STRING
            ...
       STRING .BYTE   'THIS IS A STRING',$OD,'TO BE OUTPUT',$OD
       STRLEN =       *-STRING
```

This section of code will output the string shown to the IEEE-488 bus. Since the LF.FLAG is set, the embedded carriage returns will be followed by line feeds. EOI will not be sent with the second line feed since EOI.FLAG is clear.

4.5.2.8 D_INR

PURPOSE: To input a record from the IEEE-488 bus.

ARGUMENTS: U1 = Starting address of buffer for storing the record.
U2 = Number of bytes to read.
CY = 1 if BI interrupt previously detected.

ARGUMENTS RETURNED: U1 = Address of last byte read, plus one.
U2 = Actual number of bytes read.
CY = 1 if EOI occurred with last byte.
A, X, and Y registers preserved.

DESCRIPTION: The D_INR subroutine is used to input a record from the IEEE-488 bus after the device has been addressed as a listener. This subroutine works similarly to SVC #15. Special attention must be given to the state of the CY flag before calling this subroutine. If it is in the wrong state, the program will hang, or an extra byte may be read at the beginning. The effect of the CY flag only applies to the first byte read. Bytes will be read from the IEEE channel and stored in the buffer until the number of bytes requested have been read, or EOI is received with a byte. The effect of LF.FLAG will be as described for subroutines which perform input.

```
EXAMPLE:    LDA       #0       ;NOTE: ASSUME CY SET IF BI ALREADY DETECTED
            STA       LF.FLAG  ;CLEAR LF.FLAG
            LDA       #<BUF    ;SET POINTER TO INPUT BUFFER
            STA       U1
            LDA       #>BUF
            STA       U1+1
            LDA       #<500    ;READ 500 BYTES
            STA       U2
            LDA       #>500
            STA       U2+1
```

```
        JSR     D_INR    ;INPUT THE RECORD
        ...
    BUF *=*+    500
```

This section of code will try to read 500 bytes into buffer BUF from the IEEE-488 bus. Since LF.FLAG is clear, any line feeds which follow carriage returns will be stored in the buffer.

4.5.2.9 D_OUR

PURPOSE: To output a record to the IEEE-488 bus.

ARGUMENTS: U1 = Starting address of buffer which contains the record.
U2 = Number of bytes to write.

ARGUMENTS RETURNED: A, X, and Y registers preserved.
U1 and U2 unchanged.

DESCRIPTION: The D_OUR subroutine is used to output a record to the IEEE-488 bus after the device has been addressed as the talker. This subroutine works similarly to SVC #16. Bytes will be written from the buffer to the IEEE channel until the number of bytes specified have been written. The effect of LF.FLAG and EOI.FLAG will be as described for subroutines which perform output.

```
EXAMPLE:    LDA     #$80
            STA     LF.FLAG  ;SET LF.FLAG
            STA     EOI.FLAG ;SET EOI.FLAG
            LDA     #<BUF    ;SET POINTER TO BUFFER
            STA     U1
            LDA     #>BUF
            STA     U1+1
            LDA     #<500    ;WRITE 500 BYTES
            STA     U2
            LDA     #>500
            STA     U2+1
            JSR     D_OUR    ;OUTPUT THE RECORD
            ...
        BUF *=*+    500
```

This section of code will output 500 bytes from the buffer BUF to the IEEE-488 bus. Since LF.FLAG is set, line feeds will be added after any carriage return which is output. EOI will be sent with the last byte, since EOI.FLAG is set.

4.5.2.10 D_TALK

PURPOSE: To handle operation when the device becomes addressed as the talker.

ARGUMENTS: BO interrupt not processed.

ARGUMENTS RETURNED: CY = 1 if output complete.
N, V, and Z flags reflect state of bits 6 & 7 in SAV.ISTO.

DESCRIPTION: The D_TALK subroutine is used to handle program operation when the device is addressed to talk. This subroutine is not intended to be called by the user. Instead it is the subroutine to which the device dependent output routines must be interfaced. Primarily this involves modifying the JSR to call your routine which outputs the required device dependent messages. Also, you may wish to modify this routine to handle the interrupts differently, depending on your application. You should refer to the source code for further details.

4.5.1.11 D_LISTEN

PURPOSE: To handle operation when the device becomes addressed as a listener.

ARGUMENTS: BI interrupt not processed.

ARGUMENTS RETURNED: CY = 1 if output complete.
N, V, and Z flags reflect state of bits 6 & 7 in SAV.ISTO.

DESCRIPTION: The D_LISTEN subroutine is used to handle program operation when the device is addressed to listen. This subroutine is not intended to be called by the user. Instead it is the subroutine to which the device dependent input routines must be interfaced. Primarily this involves modifying the JSR to call your routine which inputs the required device dependent messages. Also, you may wish to modify this routine to handle the interrupts differently, depending on your application. You should refer to the source code for further details.

4.5.1.12 D_LOOP

PURPOSE: To provide a main program loop device operation.

ARGUMENTS: None

ARGUMENTS RETURNED: This routine does not return.

DESCRIPTION: The D_LOOP subroutine is really not a subroutine. It is the main program loop to handle the normal overhead of responding to the IEEE-488 bus. You will typically call this routine after D_STRTUP has been executed and after device dependent initialization has been finished. Also, you may wish to modify this routine to handle the interrupts differently, depending on your application. Refer to the source code for further details.

5. DIRECT PROGRAMMING OF THE MULTI-0 HARDWARE

This section is intended for use by advanced programmers who wish to program one or more of the MultI-0's functions directly using assembly language. This may be desirable to achieve higher speeds or greater flexibility than the generalized standard programming tools provide. Nevertheless, while studying this section it is helpful to refer to the source code of these tools for examples of proper direct programming.

5.1 SUMMARY OF REGISTER ADDRESSES

Below is a summary of the 52 I/0 addresses assigned to the MultI-0 board. All 52 addresses are used and there are no duplicates or unused "holes". Individual register addresses may be found in the sections that detail each board function. The addresses are fixed in logic and can be changed only by modifying the board.

ADDRESSES	QUAN	USAGE
BFB3	1	Read: analog-to-digital converter, high 8 data bits Write: digital-to-analog converter, high 8 data bits
BFB2	1	Read: analog-to-digital converter, low 8 data bits Write: digital-to-analog converter, low 8 data bits
BFB1	1	Clock/calendar special operations register, write-only
BFB0	1	Read: Serial ports interrupt status register Write: Serial ports interrupt enable register
BFAC - BFAF	4	Serial port 2 (has only transmit and receive data)
BFA8 - BFAB	4	Serial port 1 (has full modem controls)
BFA0 - BFA7	8	IEEE-488 port
BF90 - BF9F	16	Internal 6522 (used by clock/calendar and analog I/0)
BF80 - BF8F	16	Parallel port 6522

5.2 PROGRAMMING THE PARALLEL PORTS

The 6522 VIA (Versatile Interface Adapter) chip used to implement the parallel ports provides 2 independent 8 bit ports plus 4 control lines for standard port-oriented I/0. In addition, it has two counter/timers and an 8 bit shift register. This section will describe how to use the parallel port for simple I/0 interfacing applications. The reader should refer to the 6522 data sheet in this section for detailed programming instructions for the control lines, counter/timers, and the shift register. Note that a second, internal 6522 is utilized for controlling the clock/calendar and analog input functions and this same data sheet is applicable to that chip also.

The 6522 VIA used for the parallel I/0 ports uses 16 addresses. The table below gives the function of each of these addresses:

ADDRESS	MNEMONIC	FUNCTION
BF80	PBDTA	Port B data register.
BF81	PADTA	Port A data regisrer (normal handshake operation).
BF82	PBDIR	Port B direction register.
BF83	PADIR	Port A direction register.
BF84	T1LC	Write timer 1 low latch. Read timer 1 low count and clear timer 1 interrupt flag.
BF85	T1HC	Write timer 1 high latch and high count, transfer low latch to low count. Read timer 1 high count.
BF86	T1LL	Write timer 1 low latch. Read timer 1 low count.
BF87	T1HL	Write timer 1 high latch. Read timer 1 high count.
BF88	T2LL	Write timer 2 low latch. Read timer 2 low count, clear timer 2 interrupt flag.
BF89	T2HC	Write timer 2 high count, transfer low latch to low count, clear timer 2 interrupt flag. Read high count.
BF8A	SR	Shift register.
BF8B	ACR	Auxiliary Control Register (counter and shift register mode).
BF8C	PCR	Peripheral Control Register (CA2, CA2, CB1, CB2 control).
BF8D	IFR	Interrupt Flag Register.
BF8E	IER	Interrupt Enable Register.
BF8F	PADTANH	Port A data register (no effect on handshake).

For simple port-oriented I/O, it is important that the Auxiliary Control Register (ACR), Peripheral Control Register (PCR), Interrupt Flag Register (IFR), and Interrupt Enable Register (IER) all be zeroes. Normally system reset can be counted upon to establish this condition. However if the I/O program must be completely self-initializing, the ACR and PCR registers can be initialized by simply writing zeroes into into them. The IER and IFR registers must be cleared by writing $7F into them.

The next step is to determine whether the port is to be an input port or an output port. For example, if Port A is to be input, then the program should write $00 into the Port A direction register. This is also accomplished by system reset which sets both direction registers to $00 and thus programs both data registers for inputs. To program a port for all outputs, $FF must be written into its corresponding direction register. One can also mix inputs and outputs on the same port if desired; simply set bits in the direction register to ones that correspond to those to be outputs in the data register.

Examination of the chart above will reveal that the Port A data register can be reached at two different addresses. With the Peripheral Control Register set to $00 there is little significant difference between the action of the two addresses. However when the Peripheral Control Register is set to trigger actions on CA1 and CA2 when port A is manipulated, using the $BF8F address will not trigger those actions.

Of the two timers, T1 is the most useful. For use as a simple single-shot timer, bits 6 and 7 of the ACR should be zero (default reset state). The low-order 8 bits of the time interval should first be written into the low latch using address $BF84 or $BF86 (no difference). The timer is actually started by writing the high-order 8 bits of the time interval into the high-order latch at $BF85. After the specified time interval has elapsed, bit 6 of the IFR will be set to a one. This bit is cleared when another timing interval is started or by writing $40 into the IFR. For use as a source of periodic timing marks, bits 6 and 7 of the ACR should be set to $01 and the timer started as above. After each timeout is recognized (either by testing IFR bit 6 or as a real interrupt), IFR bit 6 should be cleared by writing $40 into it. There is no cumulative timing error when in the periodic timing mark mode.

PA0-PA7 (Peripheral A Port)

The Peripheral A port consists of 8 lines which can be individually programmed to act as inputs or outputs under control of a Data Direction Register. The polarity of output pins is controlled by an Output Register and input data may 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. These lines represent one standard TTL load in the input mode and will drive one standard TTL load in the output mode. Figure 7 illustrates the output circuit.

CA1, CA2 (Peripheral A Control Lines)

The two Peripheral A control lines act as interrupt inputs or as handshake outputs. 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. CA1 is a high-impedance input only while CA2 represents one standard TTL load in the input mode. CA2 will drive one standard TTL load in the output mode.

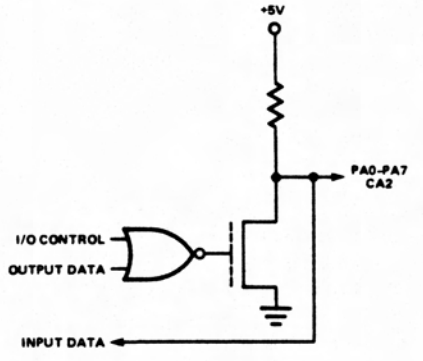


Figure 7. Peripheral A Port Output Circuit

PB0-PB7 (Peripheral B Port)

The Peripheral B port consists of eight bi-directional lines which are controlled by an output register and a data direction register in much the same manner as the PA port. In addition, 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. Peripheral B 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 1.0mA at 1.5VDC in the output mode to allow the outputs to directly drive Darlington transistor circuits. Figure 8 is the circuit schematic.

CB1, CB2 (Peripheral B Control Lines)

The Peripheral B control lines act as interrupt inputs or as handshake outputs. As with CA1 and CA2, each line controls an interrupt flag with a corresponding interrupt enable bit. In addition, these lines act as a serial port under control of the Shift Register. These lines represent one standard TTL load in the input mode and will drive one standard TTL load in the output mode. Unlike PB0-PB7, CB1 and CB2 cannot drive Darlington transistor circuits.

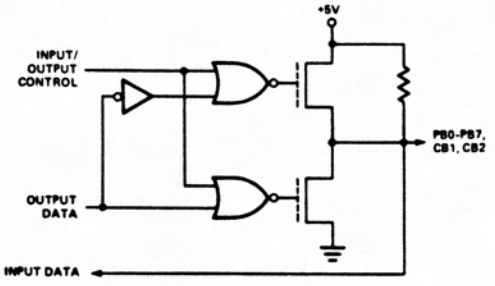


Figure 8. Peripheral B Port Output Circuit

FUNCTIONAL DESCRIPTION

Port A and Port B Operation

Each 8-bit peripheral 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 programmed as an output each peripheral pin is also controlled by a corresponding bit in the Output Register (ORA, ORB). A 1 in the Output Register causes the output to go high, and a "0" causes the output to go low. Data may be written into Output Register bits corresponding to pins which are pro-

grammed as inputs. In this case, however, the output signal is 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 levels on the PA pins. With input latching enabled and the selected active transition on CA1 having occurred, IRA will contain the data present on the PA lines at the time of the transition. Once IRA is read, however, it will appear transparent, reflecting the current state of the PA lines until the next "latching" transition.

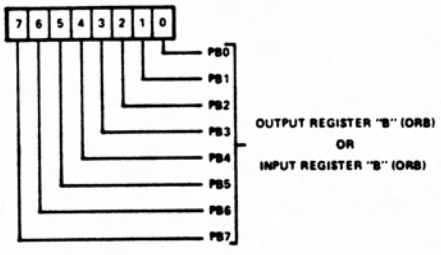
The IRB register operates similar to the IRA register. However, for pins programmed as outputs there is a difference. When reading IRA, the level on the pin determines whether a 0 or a 1 is sensed. When reading IRB, however, the bit stored in the output register, ORB, is the bit sensed. Thus, for outputs which have large loading effects and which pull an output "1" down or which pull an output "0" up, reading IRA may result in reading a "0" when a "1" was actually programmed, and reading a "1" when a "0" was programmed. Reading IRB, on the other hand, will read the "1" or "0" level actually programmed, no matter what the loading on the pin.

Figures 9, 10, and 11 illustrate the formats of the port registers. In addition, the input latching modes are selected by the Auxiliary Control Register (Figure 16.)

Handshake Control of Data Transfers

The SY6522 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.

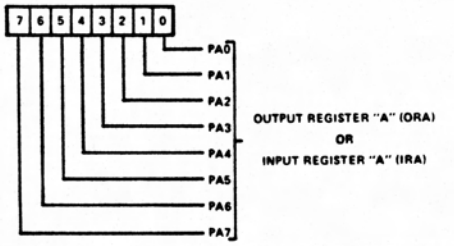
REG 0 – ORB/IRB



Pin Data Direction Selection	WRITE	READ
DDRB = "1" (OUTPUT)	MPU writes Output Level (ORB)	MPU reads output register bit in ORB. Pin level has no effect.
DDRB = "0" (INPUT) (Input latching disabled)	MPU writes into ORB, but no effect on pin level, until DDRB changed.	MPU reads input level on PB pin.
DDRB = "0" (INPUT) (Input latching enabled)		MPU reads IRB bit, which is the level of the PB pin at the time of the last CB1 active transition.

Figure 9. Output Register B (ORB), Input Register B (IRB)

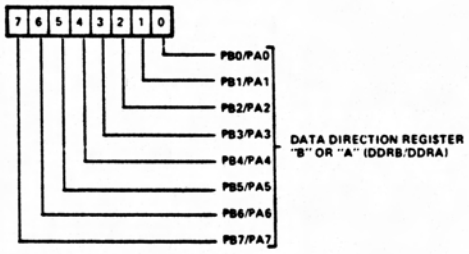
REG 1 – ORA/IRA



Pin Data Direction Selection	WRITE	READ
DDRA = "1" (OUTPUT) (Input latching disabled)	MPU writes Output Level (ORA).	MPU reads level on PA pin.
DDRA = "1" (OUTPUT) (Input latching enabled)		MPU reads IRA bit which is the level of the PA pin at the time of the last CA1 active transition.
DDRA = "0" (INPUT) (Input latching disabled)	MPU writes into ORA, but no effect on pin level, until DDRA changed.	MPU reads level on PA pin.
DDRA = "0" (INPUT) (Input latching enabled)		MPU reads IRA bit which is the level of the PA pin at the time of the last CA1 active transition.

Figure 10. Output Register A (ORA), Input Register A (IRA)

REG 2 (DDRB) AND REG 3 (DDRA)



"0" ASSOCIATED PB/PA PIN IS AN INPUT (HIGH-IMPEDANCE)

"1" ASSOCIATED PB/PA PIN IS AN OUTPUT WHOSE LEVEL IS DETERMINED BY ORB/ORA REGISTER BIT

Figure 11. Data Direction Registers (DDRB, DDRA)

Read Handshake

Positive control of data transfers from peripheral devices into the system processor can be accomplished very effectively using Read Handshaking. In this case, the peripheral device must generate the equivalent of a "Data Ready" signal to the processor signifying 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.

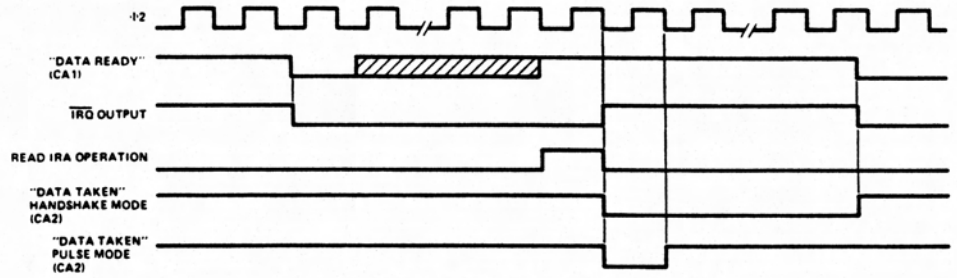


Figure 12. Read Handshake Timing (Port A, Only)

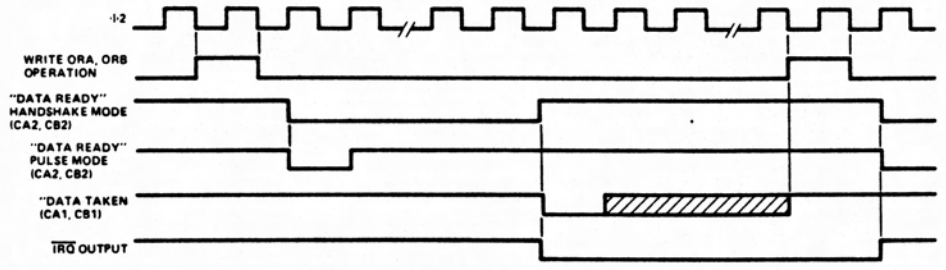


Figure 13. Write Handshake Timing

In the SY6522, 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 may be polled under program control. The "Data Taken" signal can either be a pulse or a level which is set low by the system processor and is cleared by the "Data Ready" signal. These options are shown in Figure 12 which illustrates the normal Read Handshaking sequence.

Write Handshake

The sequence of operations which allows handshaking data from the system processor to a peripheral device is very similar to that described for Read Handshaking. However, for Write Handshaking, the SY6522 generates the "Data Ready" signal 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 SY6522. CA2 or CB2 act as a "Data Ready" output in either the handshake mode or pulse mode and CA1 or CB1 accept the "Data Taken" signal from the peripheral device, setting the interrupt flag and cleaning the "Data Ready" output. This sequence is shown in Figure 13.

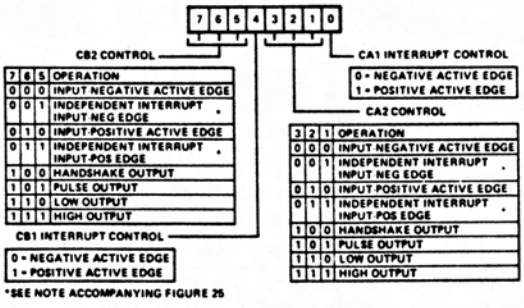
Selection of operating modes for CA1, CA2, CB1, and CB2 is accomplished by the Peripheral Control Register (Figure 14).

Timer Operation

Interval Timer T1 consists of two 8-bit latches and a 16-bit counter. The latches are used to store data which is to be loaded into the counter. After loading, the counter decrements at $\phi 2$ clock rate. Upon reaching zero, an interrupt flag will be set, and $\overline{IRQ}$ will go low if the interrupt is enabled. The timer will then disable any further interrupts, or (when programmed to) will automatically transfer the contents of the latches into the counter and begin to decrement again. In addition, the timer may be programmed to invert the output signal on a peripheral pin each time it "times-out". Each of these modes is discussed separately below.

The T1 counter is depicted in Figure 15 and the latches in Figure 16.

REG 12 – PERIPHERAL CONTROL REGISTER

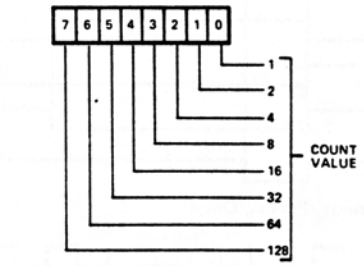


*SEE NOTE ACCOMPANYING FIGURE 25

Figure 14. CA1, CA2, CB1, CB2 Control

Two bits are provided in the Auxiliary Control Register (bits 6 and 7) to allow selection of the T1 operating modes. The four possible modes are depicted in Figure 17.

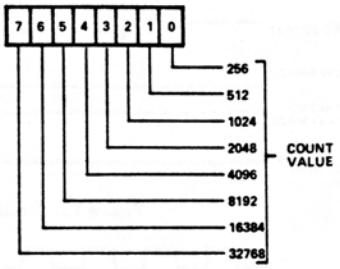
REG 4 – TIMER 1 LOW-ORDER COUNTER



WRITE – 8 BITS LOADED INTO T1 LOW-ORDER LATCHES. LATCH CONTENTS ARE TRANSFERRED INTO LOW-ORDER COUNTER AT THE TIME THE HIGH-ORDER COUNTER IS LOADED (REG 5).

READ – 8 BITS FROM T1 LOW-ORDER COUNTER TRANSFERRED TO MPU. IN ADDITION, T1 INTERRUPT FLAG IS RESET (BIT 6 IN INTERRUPT FLAG REGISTER).

REG 5 – TIMER 1 HIGH-ORDER COUNTER

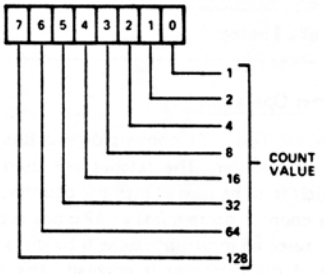


WRITE – 8 BITS LOADED INTO T1 HIGH-ORDER LATCHES. ALSO, AT THIS TIME BOTH HIGH AND LOW-ORDER LATCHES TRANSFERRED INTO T1 COUNTER, AND INITIATES COUNTDOWN. T1 INTERRUPT FLAG ALSO IS RESET.

READ – 8 BITS FROM T1 HIGH-ORDER COUNTER TRANSFERRED TO MPU.

Figure 15. T1 Counter Registers

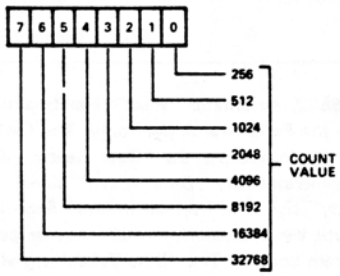
REG 6 – TIMER 1 LOW-ORDER LATCHES



WRITE – 8 BITS LOADED INTO T1 LOW-ORDER LATCHES. THIS OPERATION IS NO DIFFERENT THAT A WRITE INTO REG 4.

READ – 8 BITS FROM T1 LOW-ORDER LATCHES TRANSFERRED TO MPU. UNLIKE REG 4 OPERATION, THIS DOES NOT CAUSE RESET OF T1 INTERRUPT FLAG.

REG 7 – TIMER 1 HIGH-ORDER LATCHES



WRITE – 8 BITS LOADED INTO T1 HIGH-ORDER LATCHES. UNLIKE REG 4 OPERATION NO LATCH-TO-COUNTER TRANSFERS TAKE PLACE.

READ – 8 BITS FROM T1 HIGH-ORDER LATCHES TRANSFERRED TO MPU.

Figure 16. T1 Latch Registers

REG 11 – AUXILIARY CONTROL REGISTER

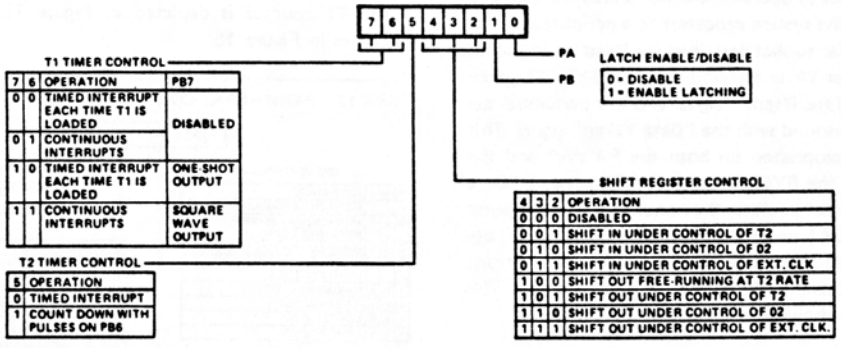


Figure 17. Auxiliary Control Register

Note: 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.

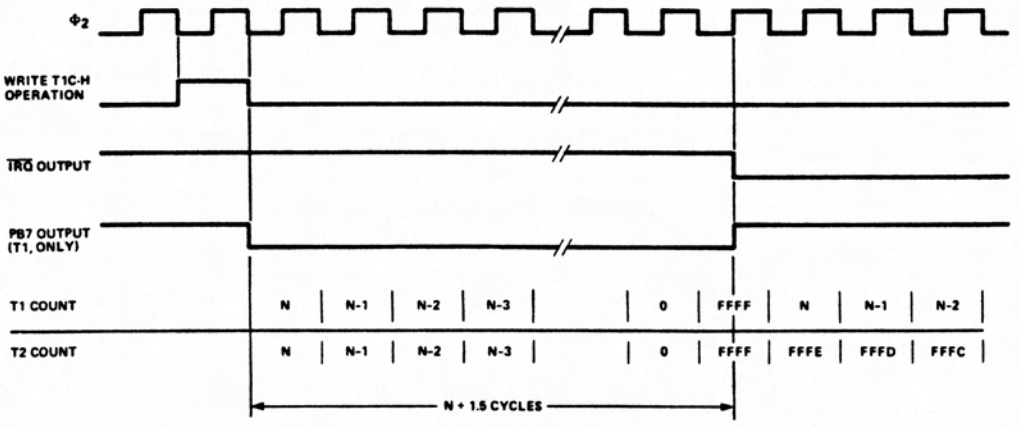


Figure 18. Timer 1 and Timer 2 One-Shot Mode Timing

Timer 1 One-Shot Mode

The interval timer one-shot mode allows generation of a single interrupt for each Timer load operation. In addition, Timer 1 can be programmed to produce a single negative pulse on PB7.

To generate a single interrupt ACR bits 6 and 7 must be 0 then either TIL-L or TIC-L must be written with the low-order count value. (A write to TIC-L is effectively a Write to TIL-L). Next the high-order count value is written to TIC-H, (the value is simultaneously written into TIL-H), and TIL-L is transferred to TIC-L. Countdown begins on the ϕ_2 following the write TIC-H and decrements at the ϕ_2 rate. T1 interrupt occurs when the counters reach 0. Generation of a negative pulse on PB7 is done in the same manner except ACR bit 7 must be a one. PB7 will go low after a Write TIC-H and go high again when the counters reach 0.

The T1 interrupt flag is reset by either writing TIC-H (starting a new count) or by reading TIC-L.

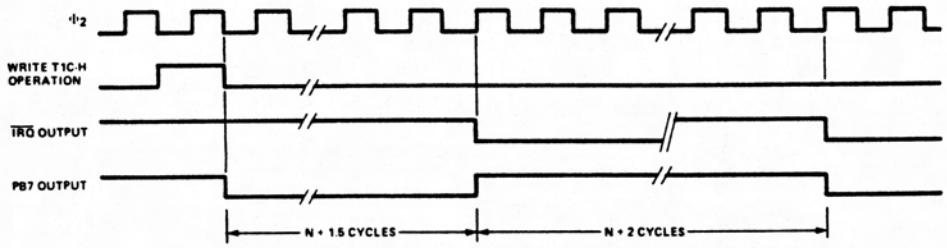
Timing for the one-shot mode is illustrated in Figure 18.

Timer 1 Free-Run Mode

The most important advantage associated with the latches in T1 is the ability to produce a continuous series of evenly spaced interrupts and the ability to produce a square wave on PB7 whose frequency is not affected by variations in the processor interrupt response time. This is accomplished in the "free-running" mode.

In the free-running mode, the interrupt flag is set and the signal on PB7 is inverted each time the counter reaches zero. However, instead of continuing to decrement from zero after a time-out, the timer automatically transfers the contents of the latch into the counter (16 bits) and continues to decrement from there. It is not necessary to rewrite the timer to enable setting the interrupt flag on the next time-out. The interrupt flag can be cleared by reading TIC-L, by writing directly into the flag as described later, or if a new count value is desired by a write to TIC-H.

All interval timers in the SY6522 are "re-triggerable". Rewriting the counter will always re-initialize the time-out period. In fact, the time-out can be prevented completely if the processor continues to rewrite the timer before it reaches zero. Timer 1 will operate in this manner if the processor writes into the high order counter (T1C-H). However, by loading the latches only, the processor can access the timer during each down-counting operation without affecting the time-out in process. Instead, the data loaded into the latches will determine the length of the next time-out period. This capability is particularly valuable in the free-running mode with the output enabled. In this mode, the signal on PB7 is inverted and the interrupt flag is set with each time-out. By responding to the interrupts with new data for the latches, the processor can determine the period of the next half cycle during each half cycle of the output signal on PB7. In this manner, very complex waveforms can be generated. Timing for the free-running mode is shown in Figure 19.



Note: A precaution to take in the use of PB7 as the timer output concerns the Data Direction Register contents for PB7. Both DDRB bit 7 and ACR bit 7 must be 1 for PB7 to function as the timer output. If either is a 0, then PB7 functions as a normal output pin, controlled by ORB bit 7.

Figure 19. Timer 1 Free-Run Mode Timing

Timer 2 Operation

Timer 2 operates as an interval timer (in the "one-slot" mode only), or as a counter for counting negative pulses on the PB6 peripheral pin. A single control bit is provided in the Auxiliary Control Register to select between these two modes. This timer is comprised of a "write-only" low-order latch (T2L-L), a "read-only" low-order counter and a read/write high order counter. The counter registers act as a 16-bit counter which decrements at Φ2 rate. Figure 20 illustrates the T2 Counter Registers.

Timer 2 One-Shot Mode

As an interval timer, T2 operates in the "one-shot" mode similar to Timer 1. In this mode, T2 provides a single interrupt for each "write T2C-H" operation. After timing out, (reading 0) the counters "roll-over" to all 1's ($FFFF_{16}$) and continue decrementing, allowing the user to read them and determine how long T2 interrupt has been set. However, setting of the interrupt flag will be disabled after initial time-out so that it will not be set by the counter continuing to decrement through zero. The processor must rewrite T2C-H to enable setting of the interrupt flag. The interrupt flag is cleared by reading T2C-L or by writing T2C-H. Timing for this operation is shown in Figure 18.

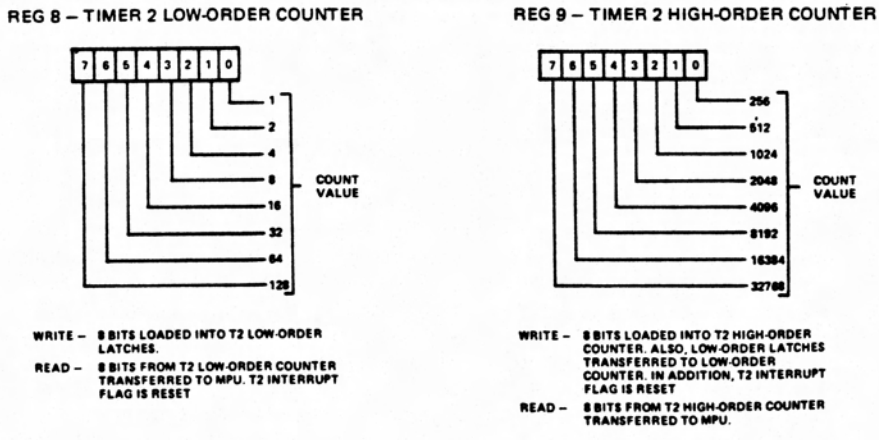


Figure 20. T2 Counter Registers

Timer 2 Pulse Counting Mode

In the pulse counting mode, T2 serves primarily to count a predetermined number of negative-going pulses on PB6. This is accomplished by first loading a number into T2. Writing into T2C-H clears the interrupt flag and allows the counter to decrement each time a pulse is applied to PB6. The interrupt flag will be set when T2 reaches zero. At this time the counter will continue to decrement with each pulse on PB6. However, it is necessary to rewrite T2C-H to allow the interrupt flag to set on subsequent down-counting operations. Timing for this mode is shown in Figure 21. The pulse must be low on the leading edge of Φ2.

Shift Register Operation

The Shift Register (SR) performs serial data transfers into and out of the CB2 pin under control of an internal modulo-8 counter. Shift pulses can be applied to the CB1 pin from an external source or, with the proper mode selection, shift pulses generated internally will appear on the CB1 pin for controlling external devices.

The control bits which select the various shift register operating modes are located in the Auxiliary Control Register. Figure 22 illustrates the configuration of the SR data bits and the SR control bits of the ACR.

Figures 23 and 24 illustrate the operation of the various shift register modes.

Interrupt Operation

Controlling interrupts within the SY6522 involves three principal operations. These are flagging the interrupts, enabling interrupts and signaling to the processor that an active interrupt exists within the chip. Interrupt flags are set by interrupting conditions which exist within the chip or on inputs to the chip. These flags normally remain set until the interrupt has been serviced. To determine the source of an interrupt, the microprocessor must examine these flags in order from highest to lowest priority. This is accomplished by reading the flag register into the processor accumulator, shifting this register either right or left and then using conditional branch instructions to detect an active interrupt.

Associated with each interrupt flag is an interrupt enable bit. This can be set or cleared by the processor to enable interrupting the processor from the corresponding interrupt flag. If an interrupt flag is set to a logic 1 by an interrupting condition, and the corresponding interrupt enable bit is set to a 1, the Interrupt Request Output ($\overline{IRQ}$) will go low. $\overline{IRQ}$ is an "open-collector" output which can be "wire-or'ed" with other devices in the system to interrupt the processor.

In the SY6522, all the interrupt flags are contained in one register. In addition, bit 7 of this register will be read as a logic 1 when an interrupt exists within the chip. This allows very convenient polling of several devices within a system to locate the source of an interrupt.

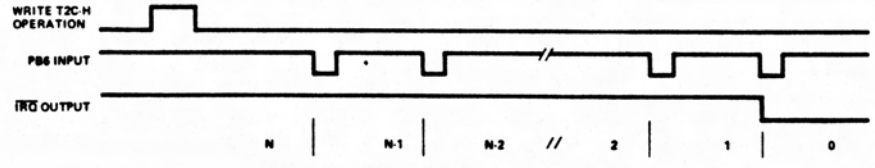
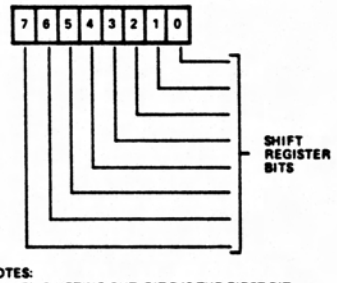


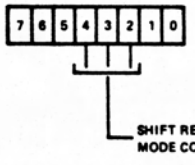
Figure 21. Timer 2 Pulse Counting Mode

REG 10 – SHIFT REGISTER



NOTES:
1. WHEN SHIFTING OUT, BIT 7 IS THE FIRST BIT OUT AND SIMULTANEOUSLY IS ROTATED BACK INTO BIT 0.
2. WHEN SHIFTING IN, BITS INITIALLY ENTER BIT 0 AND ARE SHIFTED TOWARDS BIT 7.

REG 11 – AUXILIARY CONTROL REGISTER



4	3	2	OPERATION
0	0	0	DISABLED
0	0	1	SHIFT IN UNDER CONTROL OF T2
0	1	0	SHIFT IN UNDER CONTROL OF Φ_2
0	1	1	SHIFT IN UNDER CONTROL OF EXT CLK
1	0	0	SHIFT OUT FREE-RUNNING AT T2 RATE
1	0	1	SHIFT OUT UNDER CONTROL OF T2
1	1	0	SHIFT OUT UNDER CONTROL OF Φ_2
1	1	1	SHIFT OUT UNDER CONTROL OF EXT CLK

Figure 22. SR and ACR Control Bits

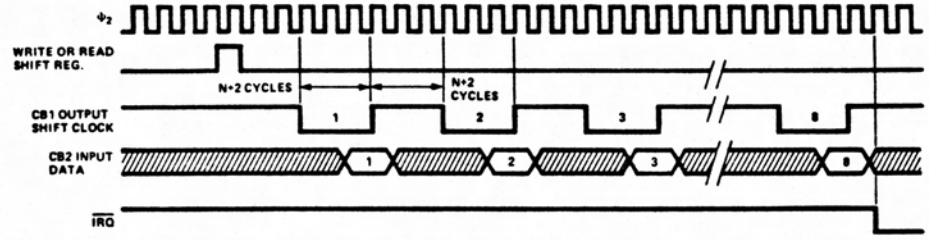
SR Disabled (000)

The 000 mode is used to disable the Shift Register. In this mode the microprocessor can write or read the SR, but the shifting operation is disabled and operation of CB1 and CB2 is controlled by the appropriate bits in the Peripheral Control Register (PCR). In this mode the SR Interrupt Flag is disabled (held to a logic 0).

Shift in Under Control of T2 (001)

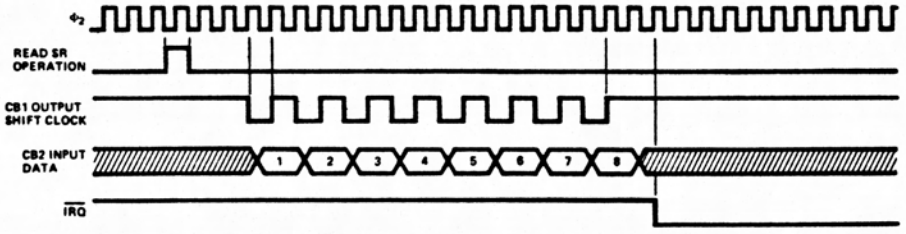
In the 001 mode the shifting rate is controlled by the low order 8 bits of T2. Shift pulses are generated on the CB1 pin to control shifting in external devices. The time between transitions of this output clock is a function of the system clock period and the contents of the low order T2 latch (N).

The shifting operation is triggered by writing or reading the shift register. Data is shifted first into the low order bit of SR and is then shifted into the next higher order bit of the shift register on the negative-going edge of each clock pulse. The input data should change before the positive-going edge of the CB1 clock pulse. This data is shifted into the shift register during the ϕ_2 clock cycle following the positive-going edge of the CB1 clock pulse. After 8 CB1 clock pulses, the shift register interrupt flag will be set and $\overline{IRQ}$ will go low.



Shift in Under Control of ϕ_2 (010)

In mode 010 the shift rate is a direct function of the system clock frequency. CB1 becomes an output which generates shift pulses for controlling external devices. Timer 2 operates as an independent interval timer and has no effect on SR. The shifting operation is triggered by reading or writing the Shift Register. Data is shifted first into bit 0 and is then shifted into the next higher order bit of the shift register on the trailing edge of each ϕ_2 clock pulse. After 8 clock pulses, the shift register interrupt flag will be set, and the output clock pulses on CB1 will stop.



Shift in Under Control of External CB1 Clock (011)

In mode 011 CB1 becomes an input. This allows an external device to load the shift register at its own pace. The shift register counter will interrupt the processor each time 8 bits have been shifted in. However, the shift register counter does not stop the shifting operation; it acts simply as a pulse counter. Reading or writing the Shift Register resets the Interrupt flag and initializes the SR counter to count another 8 pulses.

Note that the data is shifted during the first system clock cycle following the positive-going edge of the CB1 shift pulse. For this reason, data must be held stable during the first full cycle following CB1 going high.

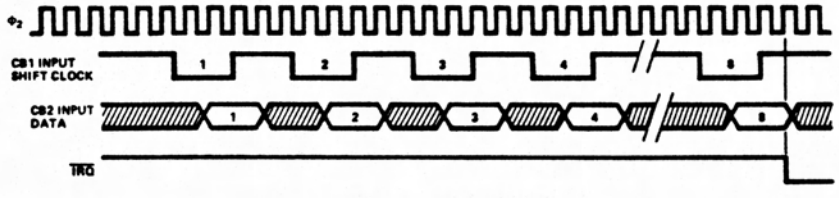
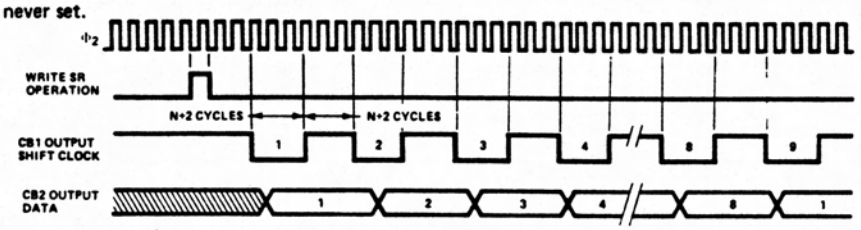


Figure 23. Shift Register Input Modes

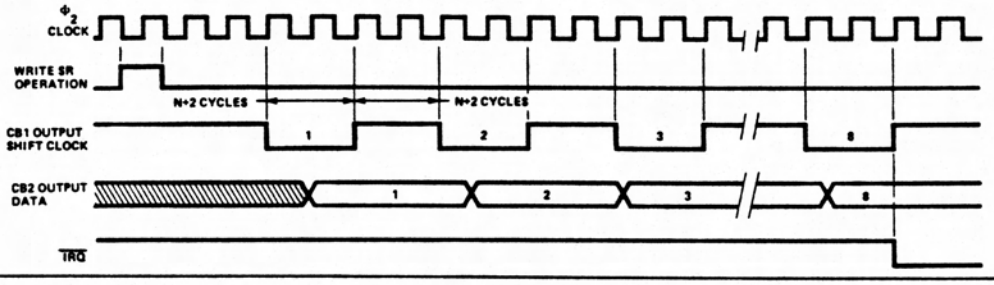
Shift Out Free-Running at T2 Rate (100)

Mode 100 is very similar to mode 101 in which the shifting rate is set by T2. However, in mode 100 the SR Counter does not stop the shifting operation. Since the Shift Register bit 7 (SR7) is recirculated back into bit 0, the 8 bits loaded into the shift register will be clocked onto CR2 repetitively. In this mode the shift register counter is disabled, and $\overline{IRQ}$ is never set.



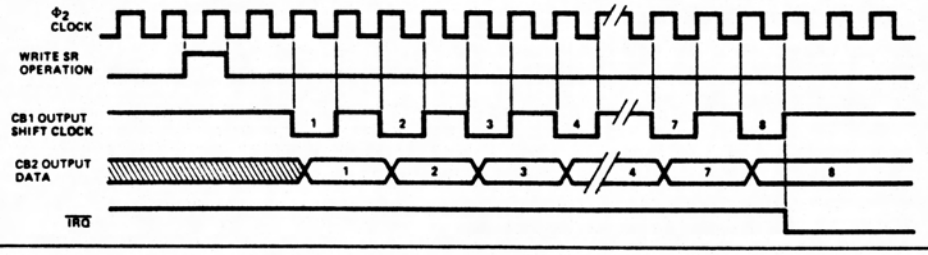
Shift Out Under Control of T2 (101)

In mode 101 the shift rate is controlled by T2 (as in the previous mode). However, with each read or write of the shift register the SR Counter is reset and 8 bits are shifted onto CB2. At the same time, 8 shift pulses are generated on CB1 to control shifting in external devices. After the 8 shift pulses, the shifting is disabled, the SR Interrupt Flag is set and CB2 remains at the last data level.



Shift Out Under Control of ϕ_2 (110)

In mode 110, the shift rate is controlled by the ϕ_2 system clock.



Shift Out Under Control of External CB1 Clock (111)

In mode 111 shifting is controlled by pulses applied to the CB1 pin by an external device. The SR counter sets the SR Interrupt flag each time it counts 8 pulses but it does not disable the shifting function. Each time the microprocessor writes or reads the shift register, the SR Interrupt flag is reset and the SR counter is initialized to begin counting the next 8 shift pulses on pin CB1. After 8 shift pulses, the interrupt flag is set. The microprocessor can then load the shift register with the next byte of data.

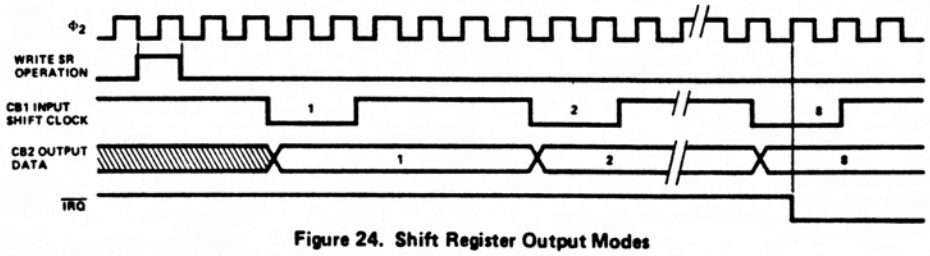


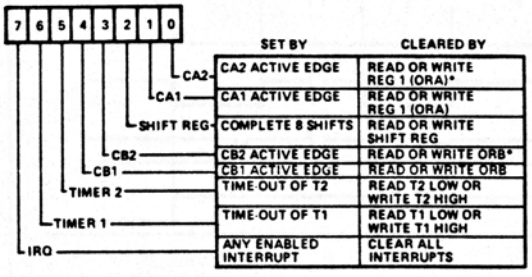
Figure 24. Shift Register Output Modes

The Interrupt Flag Register (IFR) and Interrupt Enable Register (IER) are depicted in Figures 25 and 26, respectively.

The IFR may be read directly by the processor. In addition, individual flag bits may be cleared by writing a "1" into the appropriate bit of the IFR. When the proper chip select and register signals are applied to the chip, the contents of this register are placed on the data bus. Bit 7 indicates the status of the IRQ output. This bit corresponds to the logic function: IRQ = IFR6 x IER6 + IFR5 x IER5 + IFR4 x IER4 + IFR3 x IER3 + IFR2 x IER2 + IFR1 x IER1 + IFR0 x IER0. Note: X = logic AND, + = Logic OR.

The IFR bit 7 is not a flag. Therefore, this bit is not directly cleared by writing a logic 1 into it. It can only be cleared by clearing all the flags in the register or by disabling all the active interrupts as discussed in the next section.

REG 13 – INTERRUPT FLAG REGISTER



* IF THE CA2/CB2 CONTROL IN THE PCR IS SELECTED AS "INDEPENDENT" INTERRUPT INPUT, THEN READING OR WRITING THE OUTPUT REGISTER ORA/ORB WILL NOT CLEAR THE FLAG BIT. INSTEAD, THE BIT MUST BE CLEARED BY WRITING INTO THE IFR, AS DESCRIBED PREVIOUSLY.

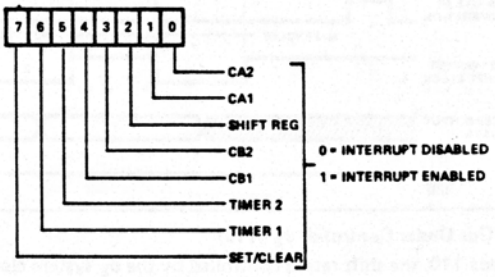
Figure 25. Interrupt Flag Register (IFR)

For each interrupt flag in IFR, there is a corresponding bit in the Interrupt Enable Register. The system processor can set or clear selected bits in this register to facilitate controlling individual interrupts without affecting others. This is accomplished by writing to address 1110 (IER address). If bit 7 of the data placed on the system data bus during this write operation is a 0, each 1 in bits 6 through 0 clears the corresponding bit in the Interrupt Enable Register. For each zero in bits 6 through 0, the corresponding bit is unaffected.

Setting selected bits in the Interrupt Enable Register is accomplished by writing to the same address with bit 7 in the data word set to a logic 1. In this case, each 1 in bits 6 through 0 will set the corresponding bit. For each zero, the corresponding bit will be unaffected. This individual control of the setting and clearing operations allows very convenient control of the interrupts during system operation.

In addition to setting and clearing IER bits, the processor can read the contents of this register by placing the proper address on the register select and chip select inputs with the R/$\overline{W}$ line high. Bit 7 will be read as a logic 1.

REG 14 – INTERRUPT ENABLE REGISTER



NOTES:
1. IF BIT 7 IS A "0", THEN EACH "1" IN BITS 0 - 6 DISABLES THE CORRESPONDING INTERRUPT.
2. IF BIT 7 IS A "1", THEN EACH "1" IN BITS 0 - 6 ENABLES THE CORRESPONDING INTERRUPT.
3. IF A READ OF THIS REGISTER IS DONE, BIT 7 WILL BE "1" AND ALL OTHER BITS WILL REFLECT THEIR ENABLE/DISABLE STATE.

Figure 26. Interrupt Enable Register (IER)

5.3 PROGRAMMING THE SERIAL PORTS

This section will describe programming of the serial ports in asynchronous mode. The intricacies of synchronous communications are beyond the scope of this manual although they are briefly described in the 2651 data sheet at the end of this section. The description will be geared to Port 1; port 2 is identical except that the various modem control signals are not implemented and all of the register addresses are 4 higher than those for Port 1.

5.3.1 Register Addresses

The 2651 serial I/O chip has 11 registers shared among 4 distinct register addresses. These register addresses are shown below for Port 1:

ADDRESS	MNEMONIC	READ/WRITE	FUNCTION
BFA8	MSIODTA	READ	Received data register
		WRITE	Transmit data register
BFA9	MSIOSTS	READ	Status register
	MSIOSYN	WRITE	Sync 1, sync 2, DLE registers
BFAA	MSIOMOD	R/W	Operation mode registers 1 and 2
BFAB	MSIOCMD	READ	Command register, reset mode & sync register counters
		WRITE	Command register

Two of the addresses actually access multiple registers under the control of an internal "distribution counter". Whenever address $BFAA is written into or read from, the register actually accessed is determined by a two-state counter. Immediately after a system reset or a read of the command register, this counter is set to state 1 and therefore selects mode register 1. After a read or write of the mode register address ($BFAA), the counter is incremented to state 2 and selects mode register 2 for the next read or write. When the second read or write does occur, the two state counter flips back to state 1. The MSIOSYN ($BFA9, write) register is similar except that its distribution counter has 3 states and the registers it selects are write-only. Reading $BFA9 will return the status register contents and will not affect the distribution counter.

5.3.2 Simple Asynchronous Mode Programming

The first step in using a MultI-0 serial port is to reset the distribution counters and then set up the various transmission parameters. Reset is accomplished by reading the command register at $BFAB and writing $00 to the interrupt enable register at $BFB0. Three registers are involved in specifying the transmission parameters and speed. The standard setting for mode register 1 at $BFAA is $4E which selects 1 stop bit, no parity, 8 data bits, and asynchronous mode at a speed of 1/16 the baud rate input clock. The standard setting for mode register 2 at $BFAA is $3X which selects the internal baud rate generator at the baud rate specified by X. The 16 possible rates are listed in the table below:

X	BAUD RATE	X	BAUD RATE	X	BAUD RATE	X	BAUD RATE
0	50	4	150	8	1800	C	4800
1	75	5	300	9	2000	D	7200
2	110	6	600	A	2400	E	9600
3	134.5	7	1200	B	3600	F	19200 *

* There is a 3.1% error at 19200 baud therefore it should not be used except between two MTU-130's, each equipped with a MultI-0 board.

Note that since the two mode registers share the same address, the mode 1 value should be written first (see section 5.3.1). The normal setting for the command register at $BFAB is $37 which clears any previous errors, selects normal operation (no echo), sets RTS and DTR outputs true, and enables both the transmitter and the receiver.

After resetting the chip and setting the mode and command registers, the serial port is ready for transmitting and receiving. The data register is at address $BFA8. When read its contents represent the character last received. When written to, its contents specify the next character to be sent. The chip is ready to transmit a character whenever bit 0 of the status register at address $BFA9 is a one. A character is transmitted simply by storing it at $BFA8 which then clears bit 0 until the chip is ready for another character. A character has been received when bit 1 of the status register is a one. Reading the character from the data register resets bit 1 to zero until another character is received. (Caution: resetting the chip does not clear the received data register therefore if the status register indicates that a character has been received immediately after reset, it should be read and discarded).

The Status Register also has a number of bits for error conditions and the modem control signals. Please refer to the 2651 chip data sheet for the location and meaning of these bits. Note that if nothing is connected to the CTS, DSR, and DCD input pins (this is always the case for port 2) that all three of these signals will appear to be active (true) because of pullup resistors on the MultI-0.

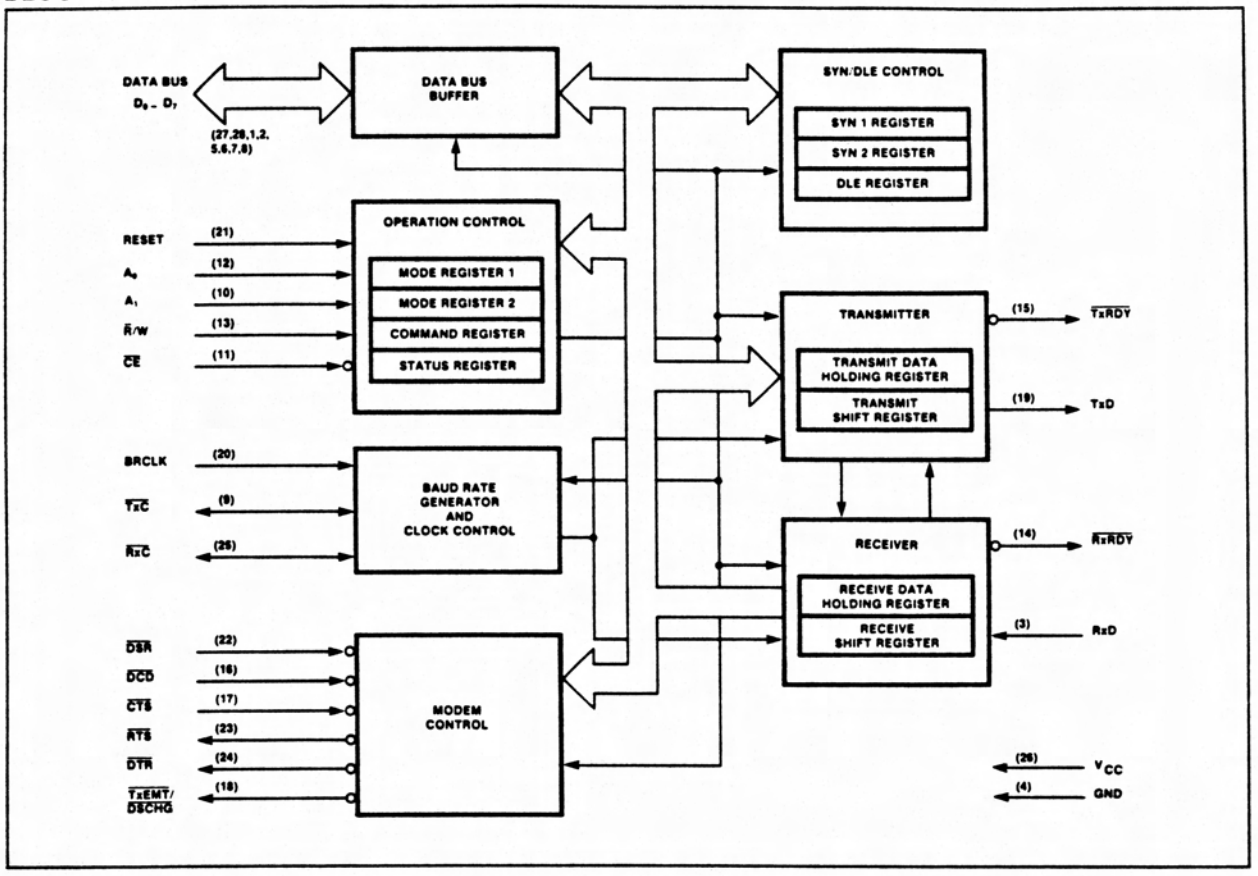
5.3.3 Using Interrupts

The 2651 serial I/0 chip does not have any interrupt control circuitry of its own. It does however have 3 outputs that represent major events and these are used by external circuitry on the MultI-0 board to provide interrupt functions. This external circuitry is accessed at address $BFB0 and controls interrupts from both serial I/0 ports. Bit assignments for the Interrupt Enable and Interrupt Status register are shown in the table below:

BIT #	READ FUNCTION	WRITE FUNCTION
7	1=Any enabled serial interrupt	Not used (don't care)
6	Port 2 TXRDY interrupt enable	Port 2 TXRDY interrupt enable
5	Port 2 RXRDY interrupt enable	Port 2 RXRDY interrupt enable
4	Port 2 DSCHG interrupt enable	Port 2 DSCHG interrupt enable
3	Not used (=0)	Not used (don't care)
2	Port 1 TXRDY interrupt enable	Port 1 TXRDY interrupt enable
1	Port 1 RXRDY interrupt enable	Port 1 RXRDY interrupt enable
0	Port 1 DSCHG interrupt enable	Port 1 DSCHG interrupt enable

To disable all serial port interrupts, simply write zeroes into the interrupt enable register at $BFB0. To enable an interrupt, set the corresponding bit in the interrupt enable register to a 1. An IRQ interrupt request will be generated when the condition corresponding to an enabled interrupt occurs. RXRDY is true when a character is being held in the received data register. It is reset by reading the data register. TXRDY is true when the transmit data register is empty. It is reset by writing the data register. DSCHG can be set either by a change in the status of the DSR or DCD modem status inputs or when both the transmit data register is empty and the previous character has been completely transmitted. It is reset by reading the status register. Note that TXRDY, RXRDY, and DSCHG reflect the state of status register bits 0-2 repectively. When more than one interrupt is enabled, the condition causing the interrupt is determined by reading the serial port status registers and testing these bits. Bit 7 of the interrupt status register is a one if any of the 6 possible serial interrupts is requesting and can be used for a quick poll in systems with many different interrupts enabled.

BLOCK DIAGRAM



BLOCK DIAGRAM

The PCI consists of six major sections. These are the transmitter, receiver, timing, operation control, modem control and SYN/DLE control. These sections communicate with each other via an internal data bus and an internal control bus. The internal data bus interfaces to the microprocessor data bus via a data bus buffer.

Operation Control

This functional block stores configuration and operation commands from the CPU and generates appropriate signals to various internal sections to control the overall device operation. It contains read and write circuits to permit communications with the microprocessor via the data bus and contains Mode Registers 1 and 2, the Command Register, and the Status Register. Details of register addressing and protocol are presented in the PCI Programming section of this data sheet.

Timing

The PCI contains a Baud Rate Generator (BRG) which is programmable to accept external transmit or receive clocks or to divide an external clock to perform data communications. The unit can generate 16 commonly used baud rates, any one of which can be selected for full duplex operation. See Table 1.

Receiver

The Receiver accepts serial data on the RxD pin, converts this serial input to parallel format, checks for bits or characters that are unique to the communication technique and sends an "assembled" character to the CPU.

Transmitter

The Transmitter accepts parallel data from the CPU, converts it to a serial bit stream, inserts the appropriate characters or bits (based on the communication technique) and outputs a composite serial stream of data on the TxD output pin.

Modem Control

The modem control section provides interfacing for three input signals and three output signals used for "handshaking" and status indication between the CPU and a modem.

SYN/DLE Control

This section contains control circuitry and three 8-bit registers storing the SYN1, SYN2, and DLE characters provided by the CPU. These registers are used in the synchronous mode of operation to provide the characters required for synchronization, idle fill and data transparency.

INTERFACE SIGNALS

The PCI interface signals can be grouped into two types: the CPU-related signals (shown in Table 2), which interface the 2651 to the microprocessor system, and the device-related signals (shown in Table 3), which are used to interface to the communications device or system.

PIN NAME	PIN NO.	INPUT/OUTPUT	FUNCTION
BRCLK	20	I	5.0688MHz clock input to the internal baud rate generator Not required if external receiver and transmitter clocks are used
$\overline{RxC}$	25	I/O	Receiver clock. If external receiver clock is programmed, this input controls the rate at which the character is to be received Its frequency is 1X, 16X or 64X the baud rate, as programmed by Mode Register 1 Data is sampled on the rising edge of the clock. If internal receiver clock is programmed, this pin becomes an output at 1X the programmed baud rate.
$\overline{TxC}$	9	I/O	Transmitter clock. If external transmitter clock is programmed, this input controls the rate at which the character is transmitted. Its frequency is 1X, 16X or 64X the baud rate, as programmed by Mode Register 1 The transmitted data changes on the falling edge of the clock If internal transmitter clock is programmed, this pin becomes an output at 1X the programmed baud rate.
RxD	3	I	Serial data input to the receiver "Mark" is high, "Space" is low
TxD	19	O	Serial data output from the transmitter. "Mark" is high, "Space" is low Held in Mark condition when the transmitter is disabled.
$\overline{DSR}$	22	I	General purpose input which can be used for Data Set Ready or Ring Indicator condition. Its complement appears as Status Register bit SR7. Causes a low output on $\overline{TxEMT/DSCHG}$ when its state changes.
$\overline{DCD}$	16	I	Data Carrier Detect input. Must be low in order for the receiver to operate Its complement appears as Status Register bit SR6. Causes a low output on $\overline{TxEMT/DSCHG}$ when its state changes.
$\overline{CTS}$	17	I	Clear to Send input. Must be low in order for the transmitter to operate.
$\overline{DTR}$	24	O	General purpose output which is the complement of Command Register bit CR1. Normally used to indicate Data Terminal Ready
$\overline{RTS}$	23	O	General purpose output which is the complement of Command Register bit CR5. Normally used to indicate Request to Send.

Table 3 DEVICE-RELATED SIGNALS

OPERATION

The functional operation of the 2651 is programmed by a set of control words supplied by the CPU These control words specify items such as synchronous or asynchronous mode, baud rate, number of bits per character, etc. The programming procedure is described in the PCI Programming section of this data sheet.

After programming, the PCI is ready to perform the desired communications functions. The receiver performs serial to parallel conversion of data received from a modem or equivalent device. The transmitter converts parallel data received from the CPU to a serial bit stream. These actions are accomplished within the framework specified by the control words.

Receiver

The 2651 is conditioned to receive data when the $\overline{DCD}$ input is low and the RxEN bit in the command register is true. In the asynchronous mode, the receiver looks for a high to low transition of the start bit on the RxD input line. If a transition is detected, the state of the RxD line is sampled again after a delay of one-half of a bit time. If RxD is now high, the search for a valid start bit is begun again. If RxD is still low, a valid start bit is assumed and the receiver continues to sample the input line at one bit time intervals until the proper number of data bits, the parity bit, and the stop bit(s) have been assembled. The data is then transferred to the Receive Data Holding Register, the RxRDY bit in the status register is set, and the $\overline{RxRDY}$ output is asserted. If the character length is less than 8 bits, the high order unused bits in the Holding Register are set to zero. The Parity Error, Framing Error, and Overrun Error status bits are set if required. If a break condition is detected (RxD is low for the entire character as well as the stop bit (s)), only one character consisting of all zeros (with the FE status bit set) will be transferred to the Holding Register. The RxD input must return to a high condition before a search for the next start bit begins.

When the PCI is initialized into the synchronous mode, the receiver first enters the hunt mode. In this mode, as data is shifted into the Receiver Shift Register a bit at a time, the contents of the register are compared to the contents of the SYN1 register. If the two are not equal, the next bit is shifted in and the comparison is repeated. When the two registers match, the hunt mode is terminated and character assembly mode begins. If single SYN operation is programmed, the SYN DETECT status bit is set. If double SYN operation is programmed, the first character assembled after SYN1 must be SYN2 in order for the SYN DETECT bit to be set. Otherwise, the PCI returns to the hunt mode. (Note that the sequence SYN1-SYN1-SYN2 will not achieve synchronization). When synchronization has been achieved, the PCI continues to assemble characters and transfer them to the Holding Register, setting the RxRDY status bit and asserting the $\overline{RxRDY}$ output each time a character is transferred. The PE and OE status bits are set as appropriate. Further receipt of the appropriate SYN sequence sets the SYN DETECT status bit. If the SYN stripping mode is commanded, SYN characters are not transferred to the Holding Register. Note that the SYN characters used to establish initial synchronization are not transferred to the Holding Register in any case.

Transmitter

The PCI is conditioned to transmit data when the $\overline{CTS}$ input is low and the TxEN command register bit is set. The 2651 indicates to the CPU that it can accept a character for transmission by setting the TxRDY status bit and asserting the $\overline{TxRDY}$ output. When the CPU writes a character into the Transmit Data Holding Register, these conditions are negated. Data is transferred from the Holding Register to the Transmit Shift Register when it is idle or has completed transmission of the previous character. The TxRDI conditions are then asserted

again. Thus, one full character time of buffering is provided.

In the asynchronous mode, the transmitter automatically sends a start bit followed by the programmed number of data bits, the least significant bit being sent first. It then appends an optional odd or even parity bit and the programmed number of stop bits. If, following transmission of the stop bits, a new character is not available in the Transmit Holding Register, the TxD output remains in the marking (high) condition and the $\overline{\text{TxEMT}}/\overline{\text{DSCHG}}$ output and its corresponding status bit are asserted Transmission resumes when the CPU loads a new character into the Holding Register The transmitter can be forced to output a continuous low (BREAK) condition by setting the Send Break command bit high.

In the synchronous mode, when the 2651 is initially conditioned to transmit, the TxD output remains high and the TxRDY condition is asserted until the first character to be transmitted (usually a SYN character) is loaded by the CPU Subsequent to this, a continuous stream of characters is transmitted. No extra bits (other than parity, if commanded) are generated by the PCI unless the CPU fails to send a new character to the PCI by the time the transmitter has completed sending the previous character Since synchronous communications does not allow gaps between characters, the PCI asserts TxEMT and automatically "fills" the gap by transmitting SYN1s, SYN1-SYN2 doublets, or DLE-SYN1 doublets, depending on the command mode. Normal transmission of the message resumes when a new character is available in the Transmit Data Holding Register If the SEND DLE bit in the command register is true, the DLE character is automatically transmitted prior to transmission of the message character

PCI PROGRAMMING

Prior to initiating data communications, the 2651 operational mode must be programmed by performing write operations to the mode and command registers. In addition, if synchronous operation is programmed, the appropriate SYN/DLE registers must be loaded. The PCI can be reconfigured at any time during program execution. However, the receiver and transmitter should be disabled if the change has an effect on the reception or transmission of a character A flowchart of the initialization process appears in Figure 1

The internal registers of the PCI are accessed by applying specific signals to the $\overline{\text{CE}}$, $\overline{\text{R}}/\text{W}$, A_1 and A_0 inputs. The conditions necessary to address each register are shown in Table 4.

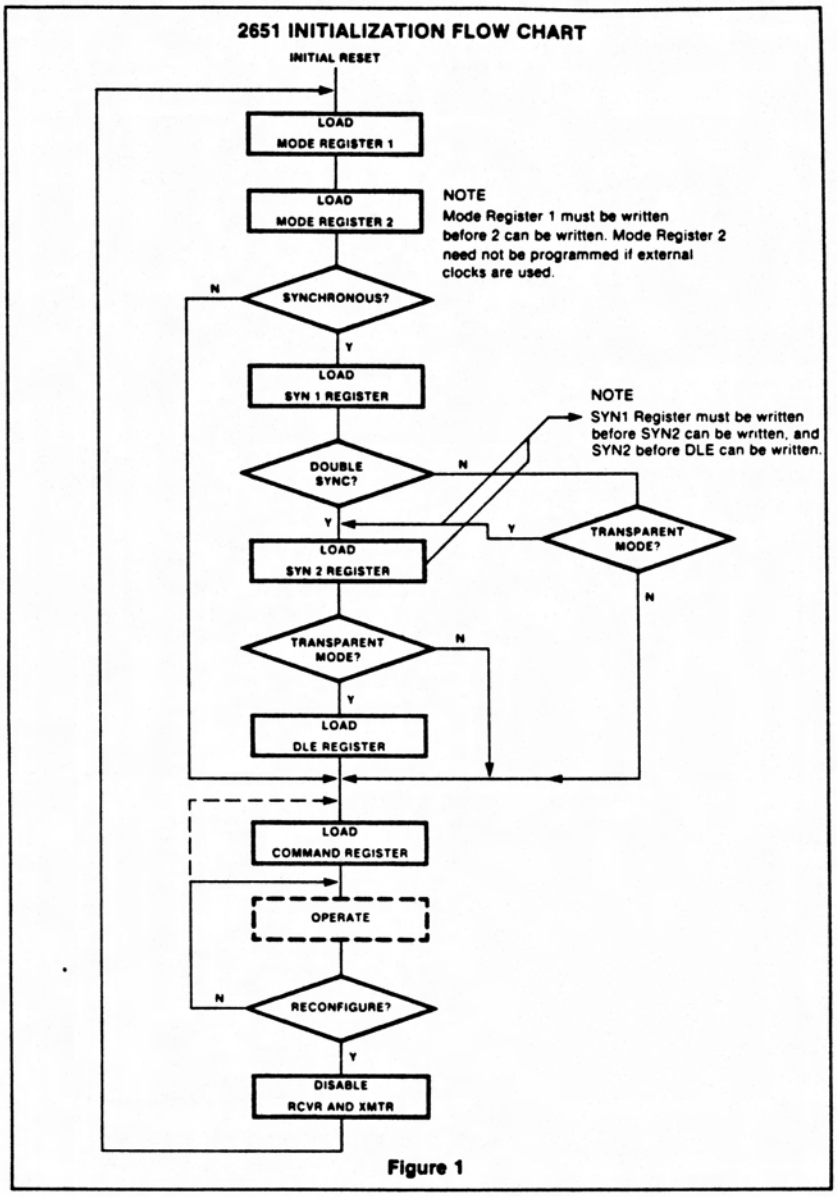


Figure 1

$\overline{\text{CE}}$	A_1	A_0	$\overline{\text{R}}/\text{W}$	FUNCTION
1	X	X	X	Tri-state data bus
0	0	0	0	Read receive holding register
0	0	0	1	Write transmit holding register
0	0	1	0	Read status register
0	0	1	1	Write SYN1/SYN2/DLE registers
0	1	0	0	Read mode registers 1/2
0	1	0	1	Write mode registers 1/2
0	1	1	0	Read command register
0	1	1	1	Write command register

Table 4 2651 REGISTER ADDRESSING

The SYN1, SYN2, and DLE registers are accessed by performing write operations with the conditions $A_1 = 0$, $A_0 = 1$, and $\overline{R}/W = 1$ The first operation loads the SYN1 register The next loads the SYN2 register, and the third loads the DLE register Reading or loading the mode registers is done in a similar manner. The first write (or read) operation addresses Mode Register 1, and a subsequent operation addresses Mode Register 2. If more than the required number of accesses are made, the internal sequencer recycles to point at the first register The pointers are reset to SYN1 Register and Mode Register 1 by a RESET input or by performing a "Read Command Register" operation, but are unaffected by any other read or write operation.

The 2651 register formats are summarized in Tables 5, 6, 7 and 8. Mode Registers 1 and 2 define the general operational characteristics of the PCI, while the Command Register controls the operation within this basic frame-work. The PCI indicates its status in the Status Register These registers are cleared when a RESET input is applied.

Mode Register 1 (MR1)

Table 5 illustrates Mode Register 1. Bits MR11 and MR10 select the communication format and baud rate multiplier. 00 specifies synchronous mode and 1X multiplier. 1X, 16X, and 64X multipliers are programmable for asynchronous format. However, the multiplier in asynchronous format applies only if the external clock input option is selected by MR24 or MR25.

MR13 and MR12 select a character length of 5, 6, 7, or 8 bits. The character length does not include the parity bit, if programmed, and does not include the start and stop bits in asynchronous mode.

MR14 controls parity generation. If enabled, a parity bit is added to the transmitted character and the receiver performs a parity check on incoming data. MR15 selects odd or even parity when parity is enabled by MR14.

In asychronous mode, MR17 and MR16 select character framing of 1, 1.5, or 2 stop bits. (If 1X baud rate is programmed, 1.5 stop bits defaults to 1 stop bits on transmit). In synchronous mode, MR17 controls the number of SYN characters used to establish synchronization and for character fill when the transmitter is idle. SYN1 alone is used if MR17 = 1, and SYN1-SYN2 is used when MR17 = 0. If the transparent mode is specified by MR16, DLE-SYN1 is used for character fill, but the normal synchronization sequence is used.

Mode Register 2 (MR2)

Table 6 illustrates Mode Register 2. MR23, MR22, MR21, and MR20 control the frequency of the internal baud rate generator (BRG) Sixteen rates are selectable. When driven by a 5.0688 MHz input at the BRCLK input (pin 20), the BRG output has zero error except at 134.5, 2000, and 19,200 baud, which have errors of +0.016%, +0.235%, and +3.125% respectively

MR25 and MR24 select either the BRG or the external inputs $\overline{TxC}$ and $\overline{RxC}$ as the clock source for the transmitter and receiver, respectively. If the BRG clock is selected, the baud rate factor in asynchronous mode is 16X regardless of the factor selected by MR11 and MR10. In addition, the corresponding clock pin provides an output at 1X the baud rate.

<table>
<tr><th>MR17</th><th>MR16</th><th>MR15</th><th>MR14</th><th>MR13</th><th>MR12</th><th>MR11</th><th>MR10</th></tr>
<tr><td colspan="2"></td><td>Parity Type</td><td>Parity Control</td><td colspan="2">Character Length</td><td colspan="2">Mode and Baud Rate Factor</td></tr>
<tr><td colspan="2">ASYNCH: STOP BIT LENGTH
00 = INVALID
01 = 1 STOP BIT
10 = 1½ STOP BITS
11 = 2 STOP BITS</td><td rowspan="2">0 = ODD
1 = EVEN</td><td rowspan="2">0 = DISABLED
1 = ENABLED</td><td colspan="2" rowspan="2">00 = 5 BITS
01 = 6 BITS
10 = 7 BITS
11 = 8 BITS</td><td colspan="2" rowspan="2">00 = SYNCHRONOUS 1X RATE
01 = ASYNCHRONOUS 1X RATE
10 = ASYNCHRONOUS 16X RATE
11 = ASYNCHRONOUS 64X RATE</td></tr>
<tr><td>SYNCH: NUMBER OF SYN CHAR
0 = DOUBLE SYN
1 = SINGLE SYN</td><td>SYNCH: TRANSPARENCY CONTROL
0 = NORMAL
1 = TRANSPARENT</td></tr>
</table>

NOTE
Baud rate factor in asynchronous applies only if external clock is selected. Factor is 16X if internal clock is selected.

Table 5 MODE REGISTER 1 (MR1)

<table>
<tr><th>MR27</th><th>MR26</th><th>MR25</th><th>MR24</th><th>MR23</th><th>MR22</th><th>MR21</th><th>MR20</th></tr>
<tr><td colspan="2"></td><td>Transmitter Clock</td><td>Receiver Clock</td><td colspan="4">Baud Rate Selection</td></tr>
<tr><td colspan="2">NOT USED</td><td>0 = EXTERNAL
1 = INTERNAL</td><td>0 = EXTERNAL
1 = INTERNAL</td><td colspan="4">0000 = 50 BAUD
0001 = 75
0010 = 110
0011 = 134.5
0100 = 150
0101 = 300
0110 = 600
0111 = 1200
1000 = 1800 BAUD
1001 = 2000
1010 = 2400
1011 = 3600
1100 = 4800
1101 = 7200
1110 = 9600
1111 = 19,200</td></tr>
</table>

Table 6 MODE REGISTER 2 (MR2)

Command Register (CR)

Table 7 illustrates Command Register Bits CR0 (TxEN) and CR2 (RxEN) enable or disable the transmitter and receiver respectively If the transmitter is disabled, it will complete the transmission of the character in the Transmit Shift Register (if any) prior to terminating operation. The TxD output will then remain in the marking state (high). If the receiver is disabled, it will terminate operation immediately. Any character being assembled will be neglected.

Bits CR1 (DTR) and CR5 (RTS) control the $\overline{DTR}$ and $\overline{RTS}$ outputs. Data at the outputs is the logical complement of the register data.

In asynchronous mode, setting CR3 will force and hold the TxD output low (spacing condition) at the end of the current transmitted character Normal operation resumes when CR3 is cleared. The TxD line will go high for a least one bit time before beginning transmission of the next character in the Transmit Data Holding Register. In synchronous mode, setting CR3 causes the transmission of the DLE register contents prior to sending the character in the Transmit Data Holding Register. CR3 should be reset in response to the next TxRDY

Setting CR4 causes the error flags in the Status Register (SR3, SR4, and SR5) to be cleared. This bit resets automatically

The PCI can operate in one of four submodes within each major mode (synchronous or asynchronous). The operational sub-mode is determined by CR7 and CR6. CR7-CR6 = 00 is the normal mode, with the transmitter and receiver operating independently in accordance with the Mode and Status Register instructions.

In asynchronous mode, CR7-CR6 = 01 places the PCI in the Automatic Echo mode. Clocked, regenerated received data is automatically directed to the TxD line while normal receiver operation continues. The receiver must be enabled (CR2 = 1), but the transmitter need not be enabled. CPU to receiver communications continues normally, but the CPU to transmitter link is disabled. Only the first character of a break condition is echoed. The TxD output will go high until the next valid start is detected. The following conditions are true while in Automatic Echo mode:

1 Data assembled by the receiver is automatically placed in the Transmit Holding Register and retransmitted by the transmitter on the TxD output.
2. Transmit clock = receive clock.
3. $\overline{TxRDY}$ output = 1
4. The $\overline{TxEMT/DSCHG}$ pin will reflect only the data set change condition.
5. The TxEN command (CR0) is ignored.

In synchronous mode, CR7-CR6 = 01 places the PCI in the Automatic SYN/DLE Stripping mode. The exact action taken depends on the setting of bits MR17 and MR16:

1 In the non-transparent, single SYN mode (MR17-MR16 = 10), characters in the data stream matching SYN1 are not transferred to the Receive Data Holding Register (RHR).
2. In the non-transparent, double SYN mode (MR17-MR16 = 00), characters in the data stream matching SYN1, or SYN2 if immediately preceded by SYN1, are not transferred to the RHR. However only the first SYN1 of an SYN1-SYN1 pair is stripped.
3. In transparent mode (MR16 =1), characters in the data stream matching DLE, or SYN1 if immediately preceded by DLE, are not transferred to the RHR. However, only the first DLE of a DLE-DLE pair is stripped.

Note that Automatic Stripping mode does not affect the setting of the DLE Detect and SYN Detect status bits (SR3 and SR5)

Two diagnostic sub-modes can also be configured. In Local Loop Back mode (CR7-CR6 = 10), the following loops are connected internally·

1 The transmitter output is connected to the receiver input.
2. $\overline{DTR}$ is connected to $\overline{DCD}$ and $\overline{RTS}$ is connected to $\overline{CTS}$.
3. Receive clock = transmit clock.
4. The $\overline{DTR}$, $\overline{RTS}$ and $\overline{TxD}$ outputs are held high.
5. The $\overline{CTS}$, $\overline{DCD}$, $\overline{DSR}$ and RxD inputs are ignored.

Additional requirements to operate in the Local Loop Back mode are that CR0 (TxEN), CR1 (DTR), and CR5 (RTS) must be set to 1. CR2 (RxEN) is ignored by the PCI.

The second diagnostic mode is the Remote Loop Back mode (CR7-CR6 = 11). In this mode:

1 Data assembled by the receiver is automatically placed in the Transmit Holding Register and retransmitted by the transmitter on the TxD output.
2. Transmit clock = receive clock.
3. No data is sent to the local CPU, but the error status conditions (PE, OE, FE) are set.
4. The $\overline{RxRDY}$, $\overline{TxRDY}$, and $\overline{TxEMT/DSCHG}$ outputs are held high.
5. CR1 (TxEN) is ignored.
6. All other signals operate normally

Status Register

The data contained in the Status Register (as shown in Table 8) indicate receiver and transmitter conditions and modem/data set status.

SR0 is the Transmitter Ready (TxRDY) status bit. It, and its corresponding output, are valid only when the transmitter is enabled. If equal to 0, it indicates that the Transmit Data Holding Register has been loaded by the CPU and the data has not been transferred to the Transmit Shift Register. If set equal to 1, it indicates that the Holding Register is ready to accept data from the CPU This bit is initially set when the Transmitter is enabled by CR0, unless a character has previously been loaded into the Holding Register. It is not set when the Automatic Echo or Remote Loop Back modes are programmed. When this bit is set, the $\overline{TxRDY}$ output pin is low. In the Automatic Echo and Remote Loop Back modes, the output is held high.

CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0
Operating Mode		Request to Send	Reset Error		Receive Control (RxEN)	Data Terminal Ready	Transmit Control (TxEN)
00 = NORMAL OPERATION 01 = ASYNCH: AUTOMATIC ECHO MODE SYNCH: SYN AND/OR DLE STRIPPING MODE 10 = LOCAL LOOP BACK 11 = REMOTE LOOP BACK		0 = FORCE $\overline{RTS}$ OUTPUT HIGH 1 = FORCE $\overline{RTS}$ OUTPUT LOW	0 = NORMAL 1 = RESET ERROR FLAG IN STATUS REG (FE, OE, PE/DLE DETECT)	**ASYNCH: FORCE BREAK** 0 = NORMAL 1 = FORCE BREAK **SYNCH: SEND DLE** 0 = NORMAL 1 = SEND DLE	0 = DISABLE 1 = ENABLE	0 = FORCE $\overline{DTR}$ OUTPUT HIGH 1 = FORCE $\overline{DTR}$ OUTPUT LOW	0 = DISABLE 1 = ENABLE

Table 7 COMMAND REGISTER (CR)

SR7	SR6	SR5	SR4	SR3	SR2	SR1	SR0
Data Set Ready	**Data Carrier Detect**	**FE/SYN Detect**	**Overrrun**	**PE/DLE Detect**	**TxEMT/DSCHG**	**RxRDY**	**TxRDY**
0 = $\overline{DSR}$ INPUT IS HIGH 1 = $\overline{DSR}$ INPUT IS LOW	0 = $\overline{DCD}$ INPUT IS HIGH 1 = $\overline{DCD}$ INPUT IS LOW	**ASYNCH:** 0 = NORMAL 1 = FRAMING ERROR **SYNCH:** 0 = NORMAL 1 = SYN CHAR DETECTED	0 = NORMAL 1 = OVERRUN ERROR	**ASYNCH:** 0 = NORMAL 1 = PARITY ERROR **SYNCH:** 0 = NORMAL 1 = PARITY ERROR OR DLE CHAR RECEIVED	0 = NORMAL 1 = CHANGE IN $\overline{DSR}$ OR $\overline{DCD}$, OR TRANSMIT SHIFT REGISTER IS EMPTY	0 = RECEIVE HOLDING REG EMPTY 1 = RECEIVE HOLDING REG HAS DATA	0 = TRANSMIT HOLDING REG BUSY 1 = TRANSMIT HOLDING REG EMPTY

Table 8 STATUS REGISTER (SR)

SR1, the Receiver Ready (RxRDY) status bit, indicates the condition of the Receive Data Holding Register If set, it indicates that a character has been loaded into the Holding Register from the Receive Shift Register and is ready to be read by the CPU. If equal to zero, there is no new character in the Holding Register. This bit is cleared when the CPU reads the Receive Data Holding Register or when the receiver is disabled by CR2. When set, the $\overline{RxRDY}$ output is low.

The TxEMT/DSCHG bit, SR2, when set, indicates either a change of state of the $\overline{DSR}$ or $\overline{DCD}$ inputs or that the Transmit Shift Register has completed transmission of a character and no new character has been loaded into the Transmit Data Holding Register. Note that in synchronous mode this bit will be set even though the appropriate "fill" character is transmitted. It is cleared when the transmitter is enabled by CR0 and does not indicate transmitter condition until at least one character is transmitted. It is also cleared when the Status Register is read by the CPU. When SR2 is set, the $\overline{TxEMT}$/$\overline{DSCHG}$ output is low.

SR3, when set, indicates a received parity error when parity is enabled by MR14. In synchronous transparent mode (MR16 = 1), with parity disabled, it indicates that a character matching the DLE Register has been received. However, only the first DLE of two successive DLEs will set SR3. This bit is cleared when the receiver is disabled and by the Reset Error command, CR4.

The Overrun Error status bit, SR4, indicates that the previous character loaded into the Receive Holding Register was not read by the CPU at the time a new received character was transferred into it. This bit is cleared when the receiver is disabled and by the Reset Error command, CR4.

In asynchronous mode, bit SR5 signifies that the received character was not framed by the programmed number of stop bits. (If 1.5 stop bits are programmed, only the first stop bit is checked.) In synchronous non-transparent mode (MR16 = 0), it indicates receipt of the SYN1 character is single SYN mode or the SYN1-SYN2 pair in double SYN mode. In synchronous transparent mode (MR16 = 1), this bit is set upon detection of the initial synchronizing characters (SYN1 or SYN1-SYN2) and, after synchronization has been achieved, when a DLE-SYN1 pair is received. The bit is reset when the receiver is disabled, when the Reset Error command is given in asynchronous mode, and when the Status Register is read by the CPU in the synchronous mode.

SR6 and SR7 reflect the conditions of the $\overline{DCD}$ and $\overline{DSR}$ inputs respectively. A low input sets its corresponding status bit and a high input clears it.

5.4 PROGRAMMING THE CLOCK/CALENDAR

Because of its slow, low-power CMOS circuitry, the clock/calendar is accessed primarily through port A of the internal 6522 parallel I/O interface chip. This chip is accessed at addresses $BF90-$BF9F with the primary addresses being $BF91 for the port A data register and $BF93 for the port A direction register. The discussion will assume familiarity with programming of the 6522; please refer to section 5.2.1 for its programming details. An additional special "trigger register" at address $BFB1 and two bits of port B are used for some functions.

5.4.1 Register Addresses

The clock/calendar uses 13 distinct 4-bit registers to hold the 13 binary coded decimal digits that represent the date and time. The lower 4 bits of port A (0-3) address the desired digit register while the upper 4 bits of port A are used to read or write the selected register. Thus the lower 4 bits of port A should always be outputs while the upper 4 bits should be inputs when reading the clock or outputs when setting the clock. The function of each of these registers is listed below:

ADDRESS	FUNCTION
0	Seconds, units, 0-9.
1	Seconds, tens, 0-5.
2	Minutes, units, 0-9.
3	Minutes, tens, 0-5.
4	Hours, units, 0-9.
5	Hours, tens, 0-2 in bits 0 and 1, bit 3=1 for 24 hour mode, bit 2=1 for PM when in 12 hour mode.
6	Day of week, 0-6.
7	Days, units, 0-9.
8	Days, tens, 0-3 in bits 0 and 1, bit 2=1 for 29 days in February.
9	Month, units, 0-9.
A	Month, tens, 0-1.
B	Year, units, 0-9.
C	Year, tens, 0-9.
D	-unused-
E	-unused-
F	Should be selected when idle to place timing pulses on data lines.

5.4.2 Procedure for Reading

The following procedure should be used to read the clock:

1. Zero the internal 6522 port A data register ($BF91).

2. Set the internal 6522 port A direction register (BF93) to $0F.

3. Set the CA2 line in the internal 6522 high to select read mode (set bits 1-3 at $BF9C to ones).

4. Write $10 to $BFB1 to temporarily stop the clock and disable clock interrupts. It will stop within 150uS and will remain stopped for 350uS during which time all of steps 5 and 6 must be performed (disable IRQ interrupts if necessary).

5. Read the port A data register ($BF91) and store it in a time array in memory.

6. Increment the port A data register and go to step 5. Repeat this 13 times to read all 13 significant time registers.

7. The clock will automatically restart. Port A should be restored to all inputs (write $00 to the direction register) so that clock interrupts can be used.

8. After copying the clock registers into memory, they may be converted into a standard time format as required.

Note that the clock must be given at least 1000uS after being read to "recover" before it can be read or written again. Insert a delay after step 8 above if necessary to insure this.

5.4.3 Procedure for Setting

The following procedure should be used for setting the clock:

1. Prepare a vector of 13 bytes in memory formatted with the register address in the lower 4 bits and the proper digit codes in the upper 4 bits. Note that the seconds digits ignore the data and are set to 0 when written.

2. Set the CA2 line on the internal 6522 low to select write mode (set bits 1-3 of address $BF9C to 110).

3. Set the internal 6522 port a direction register ($BF93) to $FF.

4. Write $10 to $BFB1 to temporarily stop the clock and disable clock interrupts. It will stop within 150uS and will remain stopped for 350uS during which time all of steps 5 -7 must be performed (disable IRQ interrupts if necessary).

5. Read a byte from the memory vector and store it in the port A data register ($BF91).

6. Write $20 to $BFB1 to actually trigger the write operation.

7. Increment the byte pointer and go to step 5. Repeat this 13 times to write all 13 significant time registers.

8. The clock will automatically restart. CA2 should be set high for maximum clock immunity to program crashes. Port A should be restored to all inputs (write $00 to the direction register) so that clock interrupts can be used.

Note that actual writing into the clock requires that CA2 be low, a valid register address be in port a bits 0-3, that $10 be written into $BFB1 within the last 500uS, and that $20 be written into $BFB1. Additionally the power supply voltage must be at least +4.5 volts and the bus reset line must be off. It is highly unlikely that these requirements would be met by accident. Also note that the clock must be given at least 1000uS after being written to "recover" before it can be read or written again. Insert a delay after step 8 above if necessary to insure this.

5.4.4 12/24 Hour Mode and Leap Years

Either 12 or 24 hour mode may be selected when the clock is set. If bit 3 of register 5 is written as a 1, then 24 hour mode is selected, otherwise 12 hour mode is selected. When reading register 5 (hours,tens) and 12 hour mode has been selected, bit 2 will be 0 for AM and 1 for PM.

When setting the date, if the current year is a leap year (as determined by the program doing the setting), then by setting bit 2 in register 8 (days, tens) to a one, the calendar will allow 29 days for February instead of 28. This bit will be automatically reset on March 1.

5.4.5 Clock Interrupts

The clock is capable of generating an interrupt every second or every hour or no interrupts at all. The CA1 input to the internal 6522 is used for clock interrupts. Therefore interrupts from CA1 must be enabled for clock interrupts to actually reach the MTU-130 bus. In addition, CA2 must be set to a one or input, the lower 4 bits of port A must be all ones or set to inputs, and the upper 4 bits of port A must be set to inputs for the interrupts to function properly. Bits 4 and 5 of port B of the internal 6522 select the interrupt frequency as follows: 00=no interrupts, 01=every hour, 10=every second, 11=every second. The interrupt occurs immediately after the selected register (second or hour) increments and is in the form of a 122uS positive-going pulse to CA1. Therefore the active edge selected for CA1 is not significant.

5.5 PROGRAMMING THE ANALOG OUTPUT

Two registers are associated with the 12-bit analog output D-to-A converter. The most significant 8 bits of the converter reside in a write-only register at $BFB3. The least significant 4 bits are left justified in a write-only register at $BFB2. The right 4 bits of this register are ignored. Thus it is best to regard the DAC input as a 16 bit binary fraction. After the DAC registers are written, its output voltage moves to the specified value and remains unchanged indefinitely until one or both registers are written into again. System reset does not change the DAC registers so the initial output voltage after power-on is unpredictable. Settling to .01% is guaranteed within 10uS of writing the most significant 8 bits and within 5uS of writing the least significant 4 bits. Thus overall settling time is minimized if the most significant 8 bits are written first. The DAC registers may be written to as often as desired.

Jumper options for the D-to-A converter can select between offset-binary and twos-complement and can select between -10 to +10 or 0 to +10 output ranges. The effect of each of the 4 possible combinations on the digital coding is shown in the table below:

	STANDARD			
DIGITAL CODE	2S-COMP,-10+10	2S-COMP,0+10	OFF-BIN,-10+10	OFF-BIN,0+10
000	0.000	+5.000	-10.000	0.000
100	+1.250	+5.625	-8.750	+0.625
700	+8.750	+9.375	-1.250	+4.375
7FF	+9.995	+9.998	-0.005	+4.998
800	-10.000	0.000	0.000	+5.000
900	-8.750	+0.625	+1.250	+5.625
F00	-1.250	+4.375	+8.750	+9.375
FFF	-0.005	+4.998	+9.995	+9.998

See section 1.3 for instructions for moving the jumpers.

The A-to-D converter uses read-only registers at $BFB2 and $BFB3 for the 12 bit data and port B bits 0-3 of the internal 6522 for selecting the input channel and triggering the analog-to-digital conversion. After a conversion, $BFB3 will contain the most significant 8 bits of the result. The lower 4 bits of the result will be found left justified at $BFB2. The right 4 bits at $BFB2 are undefined.

Bits 1-3 of port B of the internal 6522 select the input channel to be connected to the ADC. Each of the 8 combinations selects a different channel when the ADC is jumpered for single-ended operation. When jumpered for differential inputs, there are just 4 unique channels and bit 3 must be zero. See section 3.4 for details about differential operation.

Bit 0 of port B of the internal 6522 is used to control the input sample-and-hold (S&H) and the A-to-D converter itself. When bit 0 is zero, the S&H is in the sample mode and follows the selected channel's input voltage. When bit 0 switches to a one, the S&H is switched to the hold mode and the A-to-D converter is triggered to perform a conversion. 35uS later, the upper 8 bits of the result may be read at $BFB3 and 16uS after that, the lower 4 bits may be read at $BFB2. Alternatively, all 12 bits may be read in any order 51uS after the trigger. The data will persist in the A-to-D converter registers as long as bit 0 of port B remains a one.

Following conversion, another channel may be selected and the S&H placed back into the sample mode. 10uS is required after a channel change or entering the sample mode for the S&H to acquire the signal. More time is required if R28 is installed to increase the gain of the input differential amplifier. Note that a pair of INC instructions acting on port B for each channel will take bit 0 and bits 1-3 through the correct sequence for addressing and reading the input channels in sequence. With optimized software, it is possible to read all 8 channels and store their values in 544uS.

Jumper options for the A-to-D converter can select between offset-binary and twos-complement and can select between -10 to +10 or 0 to +10 input ranges. The effect of each of the 4 possible combinations on the digital coding is shown in the table below:

	STANDARD			
INPUT VOLTAGE	2S-COMP,-10+10	2S-COMP,0+10	OFF-BIN,-10+10	OFF-BIN,0+10
-10.000	800	800	000	000
-8.750	900	800	100	000
-1.250	F00	800	700	000
-0.005	FFF	800	7FF	000
0.000	000	800	800	000
+0.005	001	802	801	002
+1.250	100	A00	900	200
+8.750	700	600	F00	E00
+9.995	7FF	7FF	FFF	FFF

See section 1.3 for instructions for moving the jumpers.

5.7 PROGRAMMING THE IEEE INTERFACE

The task of programming the 9914A IEEE-488 bus controller can range from relatively easy to difficult depending on how thorough you intend to be on error checking and other such details. Unfortunately, understanding how to operate the 9914A requires that you be able to interpret the state diagrams shown in the following pages. This can be rather difficult if you are not already familiar with state diagrams.

One of the reasons this can be difficult is that there is very little information about how the various state diagrams work together to perform a desired function. By this is meant that the state diagrams show what must happen to move from one state to another, but not how to make it happen. To deal with this problem, you will probably need to famialarize yourself with all of the state diagrams. Then you may be able to piece together what must be done to acheive a desired function.

It is beyond the scope of this manual to discuss how to understand and make use of the state diagrams. You may find it helpful to examine the source code for the assembly language subroutine package. Each routine includes extensive comments to describe what the routine is intended to accomplish. In some cases, the comments include direct references to the state diagrams to show which state transistions they will cause.

Following is a short bibliography of IEEE-488 related articles:

1. An IEEE-488 Bus Tutorial, Microsystems, April, 1983

2. The Input/Output Primer, Part 3: The Parallel and HPIB (IEEE) Interfaces, BYTE, April, 1982.

There is one final thing you should keep in mind while examining the following data sheets exceprted from the Texas Instruments 9914A data book. Texas Instruments numbers their data bits <u>in reverse</u> of what MTU and other manufacturers do. This means that the least significant bit will be numbered D7 by TI and the most significant bit will be numbered D0. This applies only to pages 90 - 110 of this manual; comments in the IEEE oriented software on the distribution disk refer to data bits using the industry standard numbering sequence with bit 0 on the right.

1. INTRODUCTION

1.1 DESCRIPTION

The TMS9914A provides an interface between a Microprocessor System and the General Purpose Interface Bus (GPIB) specified in the IEEE-488 1975/78 standards and the IEEE-488A 1980 supplement. The device is controlled and configured through 8-bit memory mapped registers and enables all aspects of the standards to be implemented, including talker, listener and controller.

1.2 KEY FEATURES

- Handles all IEEE-488 1975/78 functions
- Compatible with IEEE-488A 1980 supplement
- Talker and listener function (T,TE,L,LE)
- Automatic source and acceptor handshakes (SH,AH)
- Controller with pass control
- System Controller capabilities
- Device trigger and device clear capabilities (DT,DC)
- Optional automatically cleared 'request service bit'
- Parallel and serial poll facilities (PP)
- Remote/local function with local lockout (RL)
- Single or dual primary addressing
- Secondary address capabilities
- Direct interface to SN75160/161/162 bus transceivers with no additional logic
- Compatible with most microprocessors
- Direct memory access facilities
- Memory-mapped microprocessor interface

1.3 RELATIONSHIP TO THE TMS9914

The TMS9914A is compatible with the TMS9914 and may replace it in any application without software alterations. New features are included on the TMS9914A which increase the flexibility of the device. These features are disabled at power-up and must be programmed by the user as needed.

TMS9914A VS. TMS9914:

1) Byte Output interrupt modification (see Section 2.1.1 and Appendix B)

2) Reduced bus settling time T1 (see Section 2.1.6 and Section 3.3)

3) Modification to the Service Request Function (see Section 3.5)

4) Addition of a second request service (rsv) bit which is automatically cleared (see Section 2.1.6 and Section 3.5)

1.4 INTRODUCTION TO THE IEEE-488 1975/78 INTERFACE BUS

The GPIB is designed to allow up to 15 instruments within a localized area to communicate with each other over a common bus. Each device has a unique address, read from external switches at power-on, to which it responds. Information is transmitted in byte serial bit parallel format and may consist of either device-dependent data or interface messages, commonly referred to as data or commands, respectively.

Device data may be sent by any one device (the talker) and received by a number of other devices (listeners). Instructions, such as select range, select function, or measurement data for processing or printout, may be sent in this way.

One of the devices on the bus, designated the Controller in charge (Controller), may send interface control messages. Devices can be assigned to the bus as listeners or talkers by sending their unique talk or listen addresses and may be switched between remote and local control.

The bus itself consists of a 24-wire shielded cable. Eight lines carry data; 8 are control lines; 8 are signal and system grounds. A diagram showing the IEEE bus configuration is given in Appendix A.

Three of the management lines operate as a three-line handshake between talker (or controller) and listeners. No new data is sent until each device addressed to listen has received the last byte and is ready for the next. This method of asynchronous communication ensures that the data rate is suited to the slowest active listener, as well as ensuring compatibility over a wide range of devices.

The remainder of this manual assumes working familiarity with the IEEE-488 1975/78 standards. Terminology and abbreviations defined within these standards are freely used throughout. The IEEE convention of lower case for local messages and upper case for remote messages (received via the interface) is used in all acronyms.

1.5 TYPICAL APPLICATIONS

The TMS9914A is used when an intelligent instrument is required to communicate with an IEEE-488 General Purpose Interface Bus (GPIB). It performs the interface function between the microprocessor and bus and relieves the processor of the task of maintaining the IEEE protocol. By utilizing the interrupt capabilities of the device, the bus does not have to be continually polled, and fast responses to changes in the interface configuration can be achieved.

A block diagram showing the TMS9914A in a typical application is given in Figure 1-1.

The GPIB input/output pins are connected to the IEEE-488 bus via bus transceivers. The direction of data flow is controlled by the TE and $\overline{\text{CONT}}$ outputs generated on the TMS9914A. The SN75160, 75161 and 75162 (see Appendix C) are designed specifically for use with a GPIB interface. The TE and $\overline{\text{CONT}}$ signals are routed within the devices so that the buffers on particular lines are controlled as required by the TMS9914A. Other buffers may be used, but they may require a small amount of external logic, particularly around the EOI line buffer.

Communication between the microprocessor and TMS9914A is carried out via memory-mapped registers. There are 13 registers within the TMS9914A, 6 of which are read and 7 write. These registers both pass control data to and get status information from the device.

The three least significant address lines from the MPU are connected to the register select lines RS0, RS1, and RS2 and determine the particular register selected. The high order address lines are decoded by external logic to cause the $\overline{\text{CE}}$ input to the TMS9914A to be pulled low when any one of eight consecutive addresses are selected. Thus the internal registers appear to be situated at eight consecutive locations within the MPU address space. Reading or writing to these locations transfers information between the TMS9914A and the microprocessor. Note that reading and writing to the same location will not access the same register within the TMS9914A since they are either read-only or write-only registers. For example, a read operation with RS2-RS0 = 011 gives the current status of the GPIB interface control lines, whereas a write to this location loads the auxiliary command register.

Each device on the bus interface is given a 5-bit address enabling it to be addressed as a talker or listener. This address is set on an external DIP switch (usually at the rear of an instrument) before power-on.

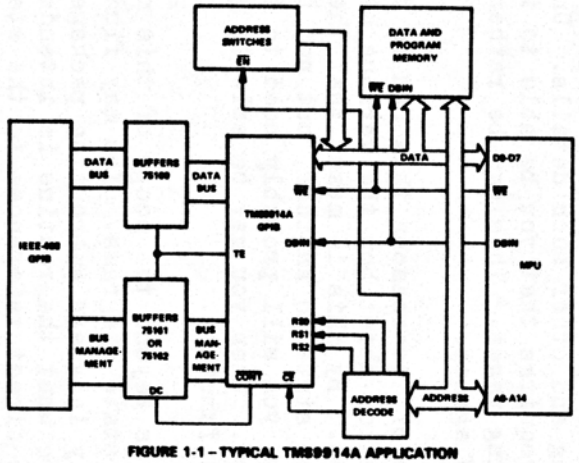


FIGURE 1-1 – TYPICAL TMS9914A APPLICATION

2. ARCHITECTURE

The block diagram of the internal architecture of the TMS9914A is given in Figure 2-1. As previously stated, there are 13 MPU accessible registers of which 6 are read and 7 are write. These registers handle all communication between the IEEE-488 1975/78 bus and microprocessor.

Each register is accessed by putting the relevant address on lines RS0, RS1 AND RS2 and performing a memory read ($\overline{WE}$ = 1 DBIN = 1) or memory write ($\overline{WE}$ = 0 DBIN = 0) operation. The register addresses and use of each bit is shown in Table 2-1 for the read registers and Table 2-2 for the write registers. A full description of each register is given in the following paragraphs.

Implementation of the functions described by the state diagrams of the IEEE-488 standard is carried out in the IEEE-488 state diagram block. Information is received from the IEEE bus and from the internal registers and is combined with the current status of the device (for example, Talker Active State, TACS) to produce the control signals to load registers or handle the handshake or bus management lines.

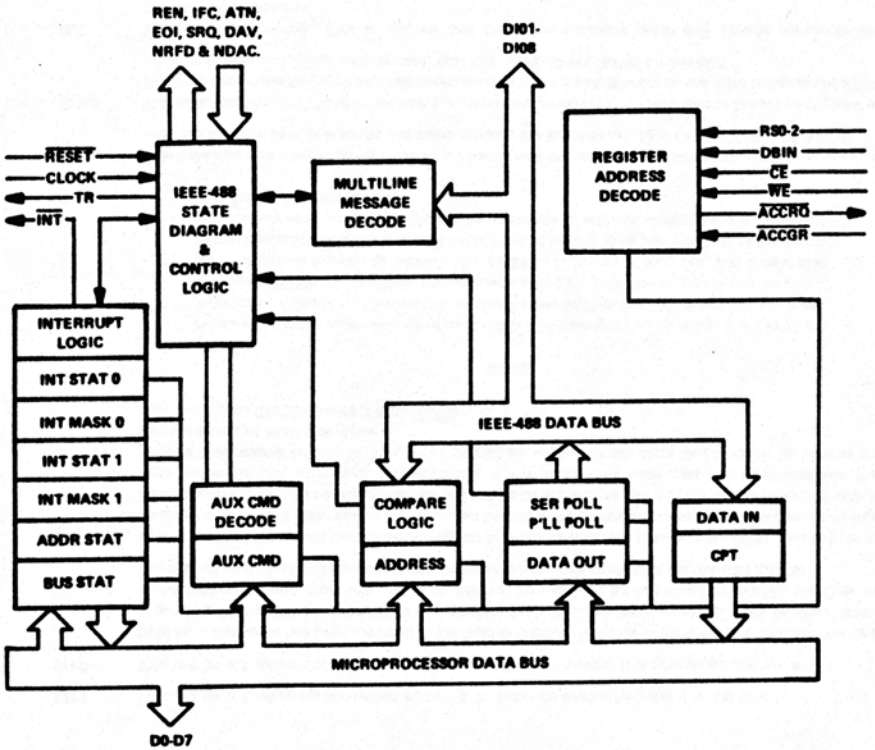


FIGURE 2-1 – SIMPLIFIED BLOCK DIAGRAM

TABLE 2-1 – TMS9914A READ REGISTERS

ADDRESS	REGISTER NAME	BIT ASSIGNMENT							
		D0	D1	D2	D3	D4	D5	D6	D7
BFA0	Int Status 0	INT0	INT1	BI	BO	END	SPAS	RLC	MAC
BFA1	Int Status 1	GET	ERR	UNC	APT	DCAS	MA	SRQ	IFC
BFA2	Address Status	REM	LLO	ATN	LPAS	TPAS	LADS	TADS	ulpa
BFA3	Bus Status	ATN	DAV	NDAC	NRFD	EOI	SRQ	IFC	REN
BFA4	*								
BFA5	*								
BFA6	Cmd Pass Thru	DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
BFA7	Data In	DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

*The TMS9914A host interface data lines will remain in the high impedence state when these register locations are addressed. An Address Switch Register may therefore be included in the address space of the device at these locations (see Section 1.5).

TABLE 2-2 – TMS9914A WRITE REGISTERS

ADDRESS	REGISTER NAME	BIT ASSIGNMENT							
		D0	D1	D2	D3	D4	D5	D6	D7
BFA0	Int Mask 0			BI	BO	END	SPAS	RLC	MAC
BFA1	Int Mask 1	GET	ERR	UNC	APT	DCAS	MA	SRQ	IFC
BFA2	*	xx	xx	xx	xx	xx	xx	xx	xx
BFA3	Auxiliary Cmd	cs	xx	xx	f4	f3	f2	f1	f0
BFA4	Address	edpa	dal	dat	A5	A4	A3	A2	A1
BFA5	Serial Poll	S8	rsvl	S6	S5	S4	S3	S2	S1
BFA6	Parallel Poll	PP8	PP7	PP6	PP5	PP4	PP3	PP2	PP1
BFA7	Data Out	DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1

*This address is not decoded by the TMS 9914A. A write to this location will have no effect on the device, as if a write had not occurred.

2.1 REGISTERS

2.1.1 Interrupt Mask and Status Registers 0

The Interrupt Mask and Interrupt Status registers operate independently of each other. The status bits will always be set when the appropriate events occur regardless of the state of the corresponding mask bit.

All interrupt bits, with the exception of INT0 and INT1 which are not storage bits, are edge triggered and are set when the appropriate condition becomes true. The storage bits are cleared immediately after the corresponding Interrupt Status Register is read by the host MPU. If an interrupt condition becomes true during this read operation, then the event is stored. The corresponding bit is set when the read operation ends, hence no interrupts are lost. In addition to being cleared by a read operation, the BO interrupt is also cleared by writing to the Data Out Register, and the BI interrupt is cleared by reading the Data In Register.

The interrupt status bits are cleared and held in the 0 condition while Software Reset (swrst) is set.

The corresponding bit of the Interrupt Mask register must be set to a 1 if an interrupt status bit is to cause an external interrupt ($\overline{INT}$ Low) when it is set (i.e., $\overline{INT}$ = $\overline{INT}$ STATUS.INT MASK). The mask register is not cleared by 'swrst' or the Hardware Reset pin ($\overline{RESET}$) and will power on in a random state. It must, therefore, be written to by the host MPU before 'swrst' is cleared to avoid extraneous interrupts (see Section 2.1.6 for operation of 'swrst').

The INT0 and INT1 bits of the Interrupt Status Register are not true status bits. INT1 will be true if there are any unmasked interrupt status bits set to a 1 in Interrupt Status Register 1. INT0 will be true if any of bits 2-7 of Interrupt Status Register 0 are unmasked and set to a 1. If either INT1 or INT0 is true, then the external interrupt pin ($\overline{INT}$) will be pulled low provided that the Disable All Interrupts feature (dal) has not been set.

The individual bits of Interrupt Status and Interrupt Mask Register 0 are described are in the following paragraphs. The conditions which set these bits, shown in parentheses, are given in terms of the state diagrams described in Section 3. Each bit is set on the rising edge of the condition shown.

INTERRUPT MASK/STATUS REGISTER 0

xx	xx	BI	BO	END	SPAS	RLC	MAC	INT	MASK 0
INT0	INT1	BI	BO	END	SPAS	RLC	MAC	INT	STATUS 0
D0	D1	D2	D3	D4	D5	D6	D7		MPU BUS

NOTE: A 0 masks and a 1 unmasks the bits in the interrupt mask registers.

INT1 This will be a 1 when an unmasked status bit in Interrupt Status Register 1 is set to a 1.

INT0 This will be a 1 when any of bits 2-7 of Interrupt Status Register 0 is unmasked and set to a 1.

BI Byte In. A data byte has been received in the Data In register. If the mask bit is not set, then no interrupt is generated but a RFD holdoff will still occur before the next data byte is accepted. If the Shadow Handshake feature is used, then this status bit will not be set. This bit is cleared by reading the Data In Register as well as after Interrupt Status Register 0 has been read. (Set On: ACDS1.LACS)

BO Byte Out. This is set when the Data Out Register is available to send a byte over the GPIB. This byte may be either a command if the device is a controller or data if the device is a talker. It is set when the device becomes an active talker or controller but will not occur if the Data Out register has been loaded with a byte which has not been sent. Subsequently, it will occur after each byte has been sent and the TMS9914A returns to SGNS. This bit is cleared by writing to the Data Out Register as well as by reading Interrupt Status Register 0.
(Set On: SGNS.CACS + SGNS.TACS.$\overline{SHFS}$)

NOTE

When a controller addresses itself as a talker and then goes to standby, there will be a momentary transition of the source handshake into SIDS before TACS becomes true and it reenters SGNS. Under these circumstances, the TMS9914A is guaranteed to give a BO interrupt on reentering 'SGNS'. The TMS9914, however, may not, and a controller going to standby as a talker should write the first byte of data into the Data Out Register immediately after writing the gts auxiliary command, without waiting for a BO interrupt (see Appendix B for details).

END This indicates that a byte just received by a listener was the last byte in a string, that is, it was received with the EOI line true. It is set at the same time as the BI interrupt. (Set On: (ACDS1.LACS.EOI)

SPAS This indicates that the TMS9914A has requested service via rsv1 or rsv2 (in the Serial Poll Register or Auxiliary Command Register) and has been polled in a serial poll. It is set on the false transition of STRS when the serial poll status byte is sent. (Set On: STRS.SPAS.(APRS1 + APRS2)

RLC Remote/Local Change. This is set by any transition between local and remote states in the Remote/Local function.
(Set On: (LOCS-REMS) + (REMS-LOCS) + (LWLS-RWLS) + (RWLS-LWLS)

MAC My Address Change. This indicates that a command has been received from the GPIB which has resulted in the addressed state of the TMS9914A to change. It will not occur if secondary addressing is being used, nor indicate that the TMS9914A has been readdressed on its other primary address.
(Set On: $ACDS1 \cdot (MTA \cdot \overline{TADS} + OTA \cdot TADS + MLA \cdot \overline{LADS} + UNL \cdot LADS)$

2.1.2 Interrupt Mask and Status Registers 1

The operation of Interrupt Mask and Status Register 1 is similar to that of Interrupt Mask and Status Register 0 except that all bits are true storage bits. The status bits are cleared only following the register being read and by 'swrst'.

There is one distinct group of interrupts in this register: GET, UNC, APT, DCAS, MA. These are all set in response to commands received over the bus and if unmasked, a Data Accepted (DAC) holdoff will occur when the interrupt in question is set. It may be released with a 'dacr' auxiliary command. This is further discussed in Section 3.2.

The mask bit of the APT interrupt is further used in the talker and listener functions. When the interrupt is unmasked, the talker and listener functions of the TMS9914A implement the extended talker and extended listener functions of IEEE-488. Otherwise these functions implement the talker and listener functions of IEEE-488.

The individual bits of Interrupt Status and Interrupt Mask Register 1 are described below. The conditions which set these bits, shown in parentheses, are given in terms of the state diagrams described in Section 3.

INTERRUPT MASK/STATUS REGISTER 1

GET	ERR	UNC	APT	DCAS	MA	SRQ	IFC	INT	MASK 1
GET	ERR	UNC	APT	DCAS	MA	SRQ	IFC	INT	STATUS 1
D0	D1	D3	D3	D4	D5	D6	D7	MPU	BUS

GET This is set if a Group Execute Trigger command is received. A DAC holdoff occurs if the interrupt is unmasked. The TR pin becomes high when this command is received and persists high for the duration of a DAC holdoff if one occurs. If the interrupt is masked, the TR pin becomes high for approximately five clock cycles.
(Set On: GET.LADS.ACDS1)

ERR Error. This is set if the source handshake becomes active and finds that the NDAC and NRFD lines are both high. This indicates that, for whatever reason, there are no acceptors on the bus.
(Set On: SERS)

UNC Unrecognized Command. This is set if a command has been received which has no meaning to the TMS9914A. Unrecognized addressed commands will only cause this interrupt if the device is LADS except for TCT which will only interrupt in TADS. Secondary commands will only cause this interrupt if the 'pts' auxiliary command has been set previously. A DAC holdoff will occur if this interrupt is unmasked which effectively enables the command pass through feature. Unrecognized commands may be inspected in the Command Pass Through Register before this holdoff is released.
(Set On: ACDS1.(UCG.$\overline{LLO.SPE.SPD.DCL}$ + ACG.$\overline{GET.GTL.SDC.TCT}$.LADS + TCT.TADS + SCG.pts))

APT Address Pass Through. Unmasking this interrupt enables secondary addressing. It is set if a secondary command is received provided that the last primary command received was a primary talk or listen address of the TMS9914A. A DAC holdoff will occur and the secondary address may be read from the Command Pass Through Register. The holdoff may be released by a 'dacr' auxiliary command and the 'cs' bit of the Auxiliary Command Register is used to indicate that a valid (cs = 1) or an invalid (cs = 0) secondary has been identified by the host MPU.
(Set On: ACDS1.SCG.(LPAS + TPAS))

DCAS Device Clear Active State. This is set when a device clear command (DCL) is received or when a selected device clear (SDC) is received with the TMS9914A in LADS. This will cause a DAC holdoff if unmasked.
(Set On: ACDS1.(DCL + SDC.LADS))

SRQ Service Request. This is provided for the benefit of the controller which should execute a serial poll in response to this interrupt. It is set when the SRQ line becomes true.
(Set On: SRQ.($\overline{CIDS + CADS}$))

MA My Address. This is set when the TMS9914A recognizes its primary talk or listen address. A DAC holdoff will occur if this is unmasked.
(Set On: (MLA + MTA).$\overline{SPMS}$.aptmk))

IFC Interface Clear. This is provided for the benefit of devices which are not the System Controller. It is set when the IFC line becomes true and indicates that the TMS9914A has been returned to an idle state. If the device is the System Controller, then the IFC interrupt is not set.
(Set On: IFCIN)

2.1.3 Address Status Register

REM	LLO	ATN	LPAS	TPAS	LADS	TADS	ulpa
D0	D1	D2	D3	D4	D5	D6	D7

MPU BUS

REM — The device is in the remote state
LLO — Local lockout is in operation
ATN — The attention line is low (true) on the bus
LPAS — TMS9914A is in the listener primary addressed state
TPAS — TMS9914A is the talker primary addressed state
LADS(or LACS) — The device is addressed to listen
TADS(or TACS) — The device is addressed to talk
ulpa — This bit shows the LSB of the last address recognized by the TMS9914A.

2.1.4 Address Register

edpa	dal	dat	A5	A4	A3	A2	A1
D0	D1	D2	D3	D4	D5	D6	D7

edpa — Enable dual primary addressing mode
dal — Disable listener function
dat — Disable talker function
A5-A1 — Primary address of the TMS9914A

Bits A5-A1 of this register contain the primary address of the device (denoted AAAAA in Table 3-15). IEEE-488 1975/78 does not allow a device to be assigned the value 11111 for bits A5-A1. When 'swrst' is true at power-up or if set by the host MPU, the TMS9914A is held in an idle state. During this time the host MPU may load the primary address of the device into these bits. Often this will be read from an Address Switch Register (see Section 1.5).

The 'edpa' bit is used to enable the dual addressing mode of the TMS9914A. It causes the LSB of the address to be ignored by the address comparator giving two consecutive primary addresses for the device. The address by which the TMS9914A was selected is indicated by the 'ulpa' bit of the Address Status Register.

The Address Register is not cleared by 'swrst' or hardware reset.

2.1.5 Auxiliary Command Register (see Section 3.1 for Auxiliary Command State Diagram)

cs	xx	xx	F4	F3	F2	F1	F0
D0	D1	D2	D3	D4	D5	D6	D7

f4-f0 — Auxiliary command select (see Table 2-3)
cs — Clear or set the feature (where applicable)

Auxiliary commands are used to enable and disable most of the selectable features of the TMS9914A and to initiate many of the actions of the device. The desired feature is selected by writing a byte to this register with the appropriate value in bits f4-f0. These values are given in Table 2-3.

The 'cs' bit is used in most cases when the feature selected by f4-f0 is of the clear/set type. The feature is enabled if 'cs' = '1' and disabled if 'cs' = '0'. The holdoff on all data (hdfa) feature is an example of such a feature. Other auxiliary commands initiate an action of the TMS9914A, such as release RFD holdoff (rhdf). In most cases, the 'cs' bit is unused and ignored by these commands.

All the clear/set auxiliary commands are cleared by the hardware RESET pin except 'swrst,' which is set true by RESET.

The force group execute trigger (fget) and return to local (rtl) auxiliary commands have a clear/set mode of operation and a pulsed mode of operation. They behave as normal clear/set features, but if they are written with 'cs' = '0' when they have not been previously set, then they will pulse true. Using the 'fget' command in this manner will produce a pulse of approximately 1 μs at the TR pin (with a 5 MHz clock). The 'rtl' command used in this way will cause a return to one of the local states (assuming local lockout is not in force) but the TMS9914A may reenter the remote state next time the listen address occurs (see Section 3.6).

TABLE 2-3 – AUXILIARY COMMANDS

c/s	f4	f3	f2	f1	f0	MNEMONIC	FEATURES
0/1	0	0	0	0	0	swrst	Software reset
0/1	0	0	0	0	1	decr	Release DAC holdoff
na	0	0	0	1	0	rhdf	Release RFD holdoff
0/1	0	0	0	1	1	hdfa	Holdoff on all data
0/1	0	0	1	0	0	hdfe	Holdoff on EOI only
na	0	0	1	0	1	nbaf	New byte available false
0/1	0	0	1	1	0	fget	Force group execute trigger
0/1	0	0	1	1	1	rtl	Return to local
na	0	1	0	0	0	feoi	Send EOI with next byte
0/1	0	1	0	0	1	lon	Listen only
0/1	0	1	0	1	0	ton	Talk only
na	0	1	0	1	1	gts	Go to standby
na	0	1	1	0	0	tca	Take control asynchronously
na	0	1	1	0	1	tcs	Take control synchronously
0/1	0	1	1	1	0	rpp	Request parallel poll
0/1	0	1	1	1	1	sic	Send interface clear
0/1	1	0	0	0	0	sre	Send remote enable
na	1	0	0	0	1	rqc	Request control
na	1	0	0	1	0	rlc	Release control
0/1	1	0	0	1	1	dai	Disable all interrupts
na	1	0	1	0	0	pts	Pass through next secondary
0/1	1	0	1	0	1	stdl	Short T1 settling time
0/1	1	0	1	1	0	shdw	Shadow handshake
0/1	1	0	1	1	1	vstdl	Very short T1 delay
0/1	1	1	0	0	0	rsv2	Request Service Bit 2

2.1.6 Description of Auxiliary Commands

Software Reset (swrst) 0/1xx00000

Setting this command causes the TMS9914A to be returned to a known idle state during which it will not take part in any activity on the GPIB. This auxiliary command is set by the power-on RESET and the chip should be configured while 'swrst' is set. Configuration should include writing the address of the device into the Address Register, writing mask values into the Interrupt Mask Registers and selecting the desired features in the Auxiliary Command Register and Address Register. After this, 'swrst' may be cleared at which point the device becomes logically existent on the GPIB. The Serial Poll Register and Parallel Poll Registers may also be written in this period but this is not necessary if there is no status to report as both of these are cleared by the power-on RESET pin. Table 2-4 lists the various states and other conditions forced by 'swrst'.

TABLE 2-4 – SOFTWARE RESET CONDITIONS

MNEMONIC	DESCRIPTION
SIDS	Source idle state
AIDS	Acceptor idle state
TIDS	Talker idle state
TPAS	Talker primary idle state
LIDS	Listener idle state
LPAS	Listener primary state
NPRS	Negative poll response state
LOCS	Local state
CIDS	Controller idle state
SPIS	Serial poll idle state
PPSS	Parallel poll standby state
ADHS	DAC holdoff state
AEHS	RFD holdoff on end state
SHFS	Source holdoff state
ENIS	END idle state

NOTES: 1. See Section 3 for definition of above.
2. All interrupt status bits are held in a 0 state, but interrupt mask bits are not affected.

Release DAC Holdoff (dacr)0/1xx00001

The Data Accepted (DAC) holdoff allows time for the host microprocessor to respond to unrecognized commands, secondary addresses, and device trigger or device clear commands. The holdoff is released by the MPU when the required action has been taken. Normally the command is loaded with the clear/set bit at zero; however, when used with the address pass through feature CS is set to one if the secondary address was valid or to zero if invalid (see APT interrupt in Section 2.1.2).

Release RFD Holdoff (rhdf)nexx00010

Any Ready For Data (RFD) holdoff caused by a 'hdfa' or 'hdfe' is released.

Holdoff on All Data (hdfa) 0/1xx00011

A Ready For Data (RFD) holdoff is caused on every data byte until the command is loaded with CS set to zero. The handshake must be completed after each byte has been received by the MPU using the 'rhdf' command.

Holdoff on End (hdfe)0/1xx00100

A RFD holdoff will occur when an end of data string message (EOI true with ATN false) is received over the interface. This holdoff must be released using 'rhdf'.

Set New Byte Available False (nbaf)nexx00101

If a talker is interrupted before the byte just stored in the data out register is sent over the interface, this byte will normally be transmitted as soon as the ATN line returns to the false state. If, as a result of the interrupt, this byte is no longer required, its transmission may be suppressed using the 'nbaf' command.

Force Group Execute Trigger (fget)0/1xx00110

The state of the TR output from the TMS9914A is affected when this command is executed. If the CS bit is zero, the line is pulsed high for approximately 5 clock cycles (1 μs at 5 MHz). If CS is one, the TR line goes high until 'fget' is sent with CS equal to zero. No interrupts or handshakes are initiated.

Return to Local (rtl)0/1xx00111

Provided the local lockout (LLO) has not been enabled, the remote/local status bit is reset, and an interrupt is generated (if enabled) to inform the host microprocessor that it should respond to the front panel controls. If the CS bit is set to one the 'rtl' command must be cleared (CS = 0) before the device is able to return to remote control. If CS is set to zero, the device may return to remote without first clearing 'rtl'.

Force End or Identify (feoi) nexx01000

This command causes the EOI message to be sent with the next data byte. The EOI line is then reset.

Listen Only (lon) 0/1xx01001

The listener state is activated until the command is sent with CS set to 0 or until deactivated by a bus command.

Talk Only (ton) 0/1xx01010

The talker state is activated until the command is sent with CS set to 0 or until deactivated by a bus command.

NOTE

'ton' and 'lon' are included for use in systems without a controller. However, where the TMS9914A is being used as a controller, it utilizes the 'lon' and 'ton' functions to set itself up as a listener or talker, respectively. Care must therefore be taken to ensure these functions are reset if sending UNL or OTA.

Go to Standby (gts)nexx01011

Issued by the controller in charge to set the ATN line false.

Take Control Synchronously (tcs)nexx01101

Control is again taken by the controller in charge, and ATN is asserted. If the controller is not a true listener, the shadow handshake command must be used to monitor the handshake lines so that the TMS9914A is synchronous with the talker/ listeners and only sends ATN true at the end of byte transfer. This ensures that no data is lost or corrupted.

Request Parallel Poll (rpp)0/1xx01110

This is executed by the controller in charge to send the parallel poll command over the interface (the TMS9914A must be in the Controller Active State so that the Attention line is asserted). The poll is completed by reading the Command Pass Through Register to obtain the status bits, then sending 'rpp' with the CS bit at zero.

Take Control Asynchronously (tca)nexx01100

This command is used by the controller in charge to set the attention line true and to gain control of the interface. The command is executed immediately and data corruption or loss may occur if a talker/listener is in the process of transferring a data byte.

Send Interface Clear (sic)0/1xx01111

The IFC line is set true when this command is sent with CS set to one. This must only be sent by the system controller and should be reset (CS = 0) after the IEEE minimum time for IFC has elapsed (100 μs). The system controller is put into the controller active state.

Send Remote Enable (sre)0/1xx10000

Issued by the system controller to set the REN line true and send the remote enable message over the interface, REN is set false by sending 'sre' with CS at zero.

Request Control (rqc)nexx10001

When the TCT command has been recognized via the unidentified command pass through, this command is sent by the MPU. The TMS9914A waits for the ATN line to go false and then enters the controller active state (CACS).

Release Control (ric)nexx10010

This command is used after TCT has been sent and handshake completed to release the ATN line and pass control to another device.

Disable All Interrupts (dai)0/1xx10011

The $\overline{\text{INT}}$ line is disabled, but the interrupt registers and any holdoffs selected are not affected.

Pass Through Next Secondary (pts)nexx10100

This feature may be used to carry out a remote configuration of a parallel poll. The parallel poll configure command (PPC) is passed through the TMS9914A as an unrecognized addressed command and is identified by the MPU. The 'pts' command is loaded, and the next byte received by the TMS9914A is passed through via the Command Pass Through Register. This would be the parallel poll enable (PPE), which is read by the microprocessor.

Set T1 Delay (std1)1xx10101

The T1 delay time can be set to 6 clock cycles (1.2 μs at 5 MHz) if this command is sent with the CS bit at one. The TI delay time is 11 clock cycles (2.2 μs at 5 MHz) following a power-on reset or if the command is sent with CS set to zero.

Shadow Handshake (shdw)0/1xx10110

This feature enables the controller in charge to carry out the listener handshake without participating in a data transfer. The Data Accepted line (DAC) is pulled true a maximum of 3 clock cycles after Data Valid (DAV) is received, and Not Ready For Data (NRFD) is allowed to go false as soon as DAV is removed.

The shadow handshake function allows the 'tcs' command to be synchronized with the Acceptor Not Ready State (ANRS) so that ATN can be re-asserted without causing the loss or corruption of data byte. The END interrupt can also be received and causes a RFD holdoff to be generated.

Very Short T1 Delay (vstd1)0/1xx10111

If this feature is enabled, the GPIB settling time (T1) will be reduced to 3 clock cycles (600 ns at 5 MHz) on the second and subsequent data bytes when ATN is false. Otherwise, the GPIB settling time is determined by the std1 feature.

Request Service Bit 2 (rsv2)0/1xx11000

The rsv2 bit performs the same function as the rsv1 bit (see Section 2.1.8) but provides a means of requesting service which is independent of the Serial Poll Register.

This allows minor updates to be made to the Serial Poll Register without affecting the state of the request service.

In addition, rsv2 is cleared when the serial poll status byte is sent to the controller during a serial poll. It is therefore used in situations where a service request is simply a request from an instrument for the controller to poll its status. As soon as this happens, rsv2 is cleared since the reason for requesting service has been satisfied. This eliminates the burden of clearing the bit from the host MPU but also guarantees that rsv2 is cleared before another serial poll can occur. If this were not so, there would be a possibility of a second status byte being sent with the RQS message true, which could result in confusion for the controller. (rsv2 is cleared on: SPAS.(APRS1 + APRS2).STRS). It should be noted that the vstd1 and rsv2 features were not present on the TMS9914.

2.1.7 Bus Status Register

ATN	DAV	NDAC	NRFD	EOI	SRQ	IFC	REN	
D0	D1	D2	D3	D4	D5	D6	D7	MPU BUS

The host MPU may examine the status of the GPIB management lines at the time of reading.

The IFC bit of this register does not indicate a true value if the device is a system controller using the 'sic' auxiliary command.

2.1.8 Serial Poll Register

S8	rsv1	S6	S5	S4	S3	S2	S1	
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1	GPIB
D0	D1	D2	D3	D4	D5	D6	D7	MPU BUS

S8, S6-S0 Device status
rsv1 Request service bit 1

Bits S8, S6-S1 of this register are sent out over the GPIB when the device is addressed during a serial poll. They are cleared by a hardware reset but not by 'swrst' and may therefore be set up during configuration of the chip. These bits are fully double buffered and if the register is written to while the device is addressed during a serial poll (serial poll active state, SPAS), the value written is saved, and these bits are updated when SPAS is terminated.

The rsv1 bit provides an input to the service request function of the TMS9914A and is used to instruct this to request that the controller service the device. When rsv1 is set true, the SRQ line is pulled true on the GPIB, and the controller typically responds by setting up a serial poll to obtain the status of all instruments on the bus that may require service. When the TMS9914A is addressed to send its status byte, SRQ is set false, and the status byte is sent with the RQS message true on DIO7. The rsv1 bit must then be cleared and set true again if service is to be requested a second time. The SPAS interrupt is set immediately following the status byte being sent.

The rsv1 bit is also cleared by the hardware reset pin but not by 'swrst'. It is not double-buffered but the service request function comprehends changes in the state of rsv1 while the device is in SPAS. The Serial Poll Register may therefore be written to any time. Section 3.5 contains more information on the service request function. Note that the rsv1 bit of the TMS9914 was simply referred to as 'rsv' since this device did not have an rsv2 auxiliary command.

2.1.9 Command Pass Through Register

DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1	GPIB
D0	D1	D2	D3	D4	D5	D6	D7	MPU BUS

This provides a means of directly inspecting the GPIB data lines (DIO(8-1)). It has no storage and should only be used when the data lines are known to be in a steady state such as will occur during a DAC holdoff or in CPWS during a parallel poll. It is used to read unrecognized commands and secondaries following a UNC interrupt or to read secondary addresses following an APT interrupt. In addition, an active controller uses this register to read the results of a parallel poll at least 2μs after setting the 'rpp' auxiliary command.

2.1.10 Parallel Poll Register

PP8	PP7	PP6	PP5	PP4	PP3	PP2	PP1	
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1	GPIB
D0	D1	D2	D3	D4	D5	D6	D7	MPU BUS

When a controller initiates a parallel poll, the contents of this register are presented to the GPIB data lines. If all bits of the register are cleared, then none of the lines DIO(8-1) will be pulled low during a parallel poll which corresponds to the Parallel Poll Idle State (PPIS) of IEEE-488. If it is desired to participate in a parallel poll, then the bit corresponding to the desired parallel poll response is set to a 1.

The Parallel Poll Register is fully double buffered. If it is written to during a parallel poll, the new value is held until the parallel poll ends, at which point the register is updated. This permits the host MPU to update the parallel poll response completely asynchronously to the GPIB.

If this register is cleared by the hardware RESET pin but not by 'swrst,' it may be loaded while the chip is being configured with 'swrst' set.

2.1.11 Data In Register

DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1	GPIB
D0	D1	D2	D3	D4	D5	D6	D7	MPU BUS

This register is used to hold data received by the TMS9914A when it is a listener. It is loaded during Accept Data State (ACDS1) and, following this, an RFD holdoff will occur. This will normally be released when the byte is read by the host MPU, but if the Holdoff On All Data (hdfa) feature is selected, this holdoff must be released by the 'rhdf' auxiliary command.

If the Holdoff On End (hdfe) feature is selected, the RFD holdoff will be released by reading the Data In Register. But if the EOI line is true when the byte is received, reading the data byte will not release the holdoff and rhdf must be used.

As the Data In Register is loaded, the BI interrupt is set. The END interrupt is set simultaneously if the byte is accompanied by a true EOI line.

2.1.12 Data Out Register

DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1	GPIB
D0	D1	D2	D3	D4	D5	D6	D7	MPU BUS

The Data Out register is used by a controller or talker for sending interface messages and device dependent messages. When the TMS9914A enters the Talker Active State (TACS) or the Controller Active State (CACS), the contents of the Data Out Register are presented to the GPIB data lines (DIO(8-1)), and the byte is sent over the bus under the control of the Source Handshake. Each time a byte is written, the source handshake is enabled, and the byte is sent. If the handshake is interrupted before the byte can be sent, then it will be sent next time the Source Handshake becomes active unless a new byte available false (nbaf) auxiliary command is written. This has the effect of clearing an unsent byte from the Data Out Register, and although the register itself is not cleared, the TMS9914A behaves as if it had not been loaded.

Each time the source handshake becomes active and there is no unsent byte in the Data Out Register, a BO interrupt will occur informing the host MPU that the Data Out Register is available for use.

The Data In Register and Data Out Register operate independently. The Data Out Register is not double buffered, and its contents are output directly to the data lines of the GPIB.

2.2 DIRECT MEMORY ACCESS

The TMS9914A can operate in DMA using the $\overline{\text{ACCRQ}}$ (DMA request) and $\overline{\text{ACCGR}}$ (DMA grant) DMA handshake lines. The operation is automatic within the TMS9914A and needs no 'mpu' configuration.

The $\overline{\text{ACCRQ}}$ signal is set by (BO.$\overline{\text{CACS}}$ + BI) and can therefore not be used by a controller while ATN is asserted. It is reset by 'swrst' readin data in register, writing to the data out register and $\overline{\text{ACCGR}}$. It is not cleared by reading interrupt status register 0.

If using DMA, the internal CE and addressing is disabled by the $\overline{\text{ACCGR}}$ signal going low and $\overline{\text{ACCGR}}$ will automatically address either the data in register (DBIN = 0) or the data out register (DBIN = 1).

NOTE

The sense of DBIN is inverted for DMA operation.

At the end of a DMA read from memory sequence, the $\overline{\text{ACCRQ}}$ will be left low (also BO bit set). It may be necessary for the 'mpu' to clear this in some circumstances, e.g., starting DMA write to memory sequence.

In DMA it is recommended that the MA interrupt be unmasked to prevent errors due to interrupted data streams.

If DMA is not being utilized, the $\overline{\text{ACCGR}}$ signal must be held high. In this case, the $\overline{\text{ACCRQ}}$ signal can be used as a seperate interrupt line for BO and BI. This allows faster 'mpu' transfers to take place as it is not necessary to read the interrupt register to find the cause of the interrupt. Figure 2-2 shows a typical DMA configuration.

2.3 TERMINAL ASSIGNMENTS AND FUNCTIONS

The IEEE-488 standard uses the negative logic convention for the GPIB lines. The FALSE state (0) is represented by a high voltage (> 2.0 V); the TRUE state (1) is represented by a low voltage (> 0.8 V). The GPIB terminations of the TMS9914A are in agreement with this convention. For example, if Data Valid is true (1), the DAV line is pulled low by the device. These terminations are connected to the bus via noninverting buffers to obtain the correct signal polarity.

Note that the terminations on the microprocessor side of the device are in positive logic (true state (1) = high voltage : false state (0) = low voltage). This is in agreement with the logic convention used by most microprocessors. Thus if:

D0(MSB)							D7(LSB)
0	1	1	0	1	0	0	1

is written into the data out register, it will appear as:

DIO8(MSB)							DIO1(LSB)
HIGH	LOW	LOW	HIGH	LOW	HIGH	HIGH	LOW

on the IEEE-488 D10 lines.

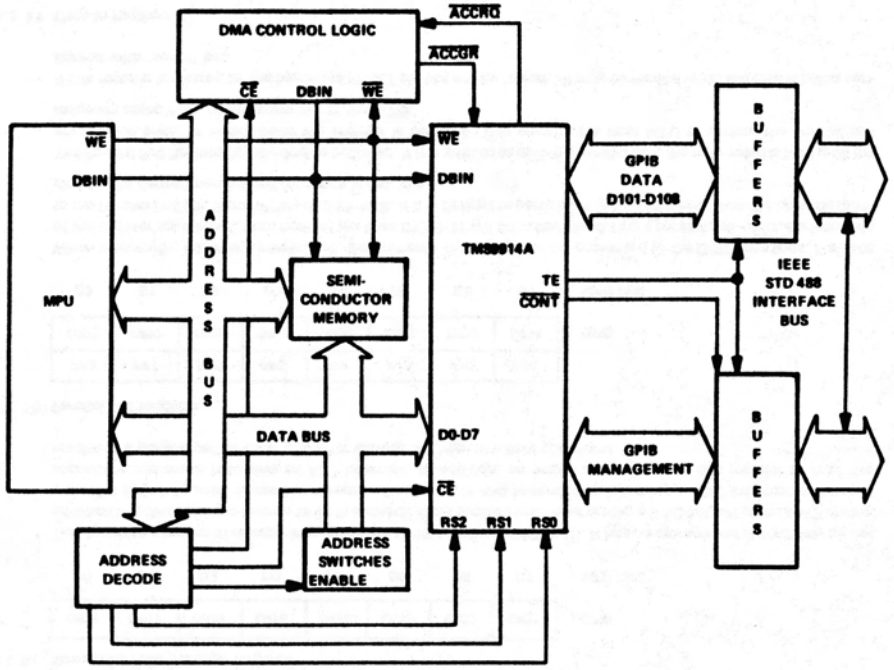


FIGURE 2-2 – DMA CONFIGURATION

3. STATE DIAGRAM IMPLEMENTATION

This section presents the state diagrams for the TMS9914A.

Where equivalent, the names of TMS9914A states are the same as those of IEEE-488. In some cases, IEEE-488 states have been divided, for example, ACDS of the IEEE-488 has been split into ACDS1 and ACDS2. The convention of lower case characters for local messages and upper case for remote messages and interface states is retained.

State diagrams with remote message outputs are supplemented with tables. T is used to represent a true output and F a false output. Parentheses denote a passive output; otherwise, it is active. The outputs shown are the values presented to the bus and assume the use of the SN75160 and SN75161 or SN75162 transceivers or their logical equivalents. The symbol (NUL) associated with DIO(1-8) indicates that each of these lines is sent passive false by the function in question.

NOTE

An arrow into a state with no state as its origin represents a transition from every other state on the diagram. Note, however, that this does not imply that all exit conditions from the destination state are overridden. If such an entry condition is true and, simultaneously, an exit condition is true then this represents an illegal situation and should be avoided. Such situations will not occur in normal operation of the device.

No maximum timings are discussed. The TMS9914A with its recommended transceivers meets all IEEE-488 maximum timing requirements as may be determined from Section 4.4 and Appendix C. If the TMS9914A is used with other transceivers, then it must be ensured that these requirements are still met.

3.1 AUXILIARY COMMANDS

There are two basic types of commands implemented in the auxiliary command register: immediate execute and clear/set.

The clear/set commands are used to enable and disable the various features of the TMS9914A. The particular feature is selected by the code on f0-f4 (see Section 2.1.6) and it is set or cleared according to the value on the cs bit. For the purposes of the state diagrams, the mnemonic of a clear/set command simply represents its current state.

The immediate execute auxiliary commands remain active for the duration of a strobe signal after the auxiliary command register has been written to. This is represented in the form of a state diagram in Figure 3-1. Note that writes to the auxiliary command register must be spaced by at least five clock cycles. For the purposes of the remaining state diagrams, the immediate execute commands are represented as the mnemonic gated by the auxiliary command strobe state (AXSS).

The clear/set bit of the auxiliary command register is used by several of the immediate execute commands, for example, 'decr' uses it to differentiate between valid and not valid secondary addresses when releasing a DAC holdoff on a secondary address. The 'lon' and 'ton' auxiliary commands are also considered immediate execute, as described in Section 3.4.

The 'fget' and 'rtl' auxiliary commands are both immediate execute and clear/set. They may be cleared or set in the normal way, but if they are cleared when they are already in the false state, they will pulse true for the duration of AXSS. In the following state diagrams, however, these are simply included in their clear/set form.

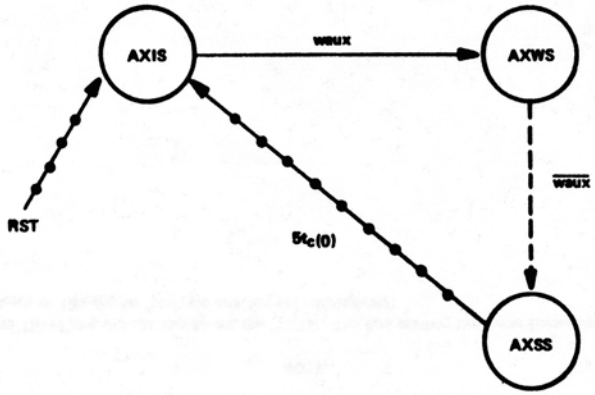


FIGURE 3-1 – TMS9914A AUXILIARY COMMAND STATE DIAGRAM

TABLE 3-1 – AUXILIARY COMMAND STATE DIAGRAM MNEMONICS

MESSAGES			STATES		
waux	=	write to auxiliary command register	AXIS	=	auxiliary command register idle state
$t_{c(0)}$	=	clock cycle time	AXWS	=	auxiliary command write state
			AXSS	=	auxiliary command strobe state

3.2 ACCEPTOR HANDSHAKE

The TMS9914A acceptor handshake is shown in Figure 3-2. The main variation from IEEE-488 to note is that the device remains in AIDS while the controller function is in CACS. The TMS9914A, therefore, does not monitor the commands which it sends over the bus and this places some restrictions on the user which are outlined in Section 3.8.

The accept data state of IEEE-488 (ACDS) is divided into two states. The first, (ACDS1) is used to strobe data into the Data In Register or to sequence the decoding of commands from the bus. All interrupts generated by the acceptor handshake (GET, MA, MAC, DCAS, APT, UCG, BI, and END) are generated by this state. The second (ACDS2) is used as a holding state where the device will remain in the event of a DAC holdoff.

As discussed in Section 2.1.2, certain of the commands will cause interrupts in ACDS1 and, if the interrupts are unmasked, a DAC holdoff will occur. The interrupts concerned are GET, MA, DCAS, UCG, and APT. This is represented in the state diagram by the signal SAHF which becomes true when one of the above interrupts is set if it is unmasked. It persists for the duration of ACDS1. This event is stored by causing the ADHS to become active which inhibits the transition from ACDS2 to AWNS. ADHS is cleared by 'decr'. Table 3-15 shows the response of the TMS9914A to the various bus commands.

If a GET command is received in ACDS1, then the TR pin will be set high. This high condition persists throughout ACDS1 and ACDS2, which means that if a DAC holdoff occurs, the TR pin will remain high until the holdoff is released by a 'decr' auxiliary command.

Two additional state diagrams are included to record the type of data received in ACDS1 when ATN is false. ANHS indicates that a data byte has been received and that an RFD holdoff should be caused before the next data byte is accepted. The holdoff may be released by reading the Data In Register unless the 'hdfa' feature is enabled in which case 'rhdf' must be used. AEHS shows that the last data byte was accepted with the EOI message true and the 'hdfe' feature set. This will cause an RFD holdoff which can only be released by 'rhdf'.

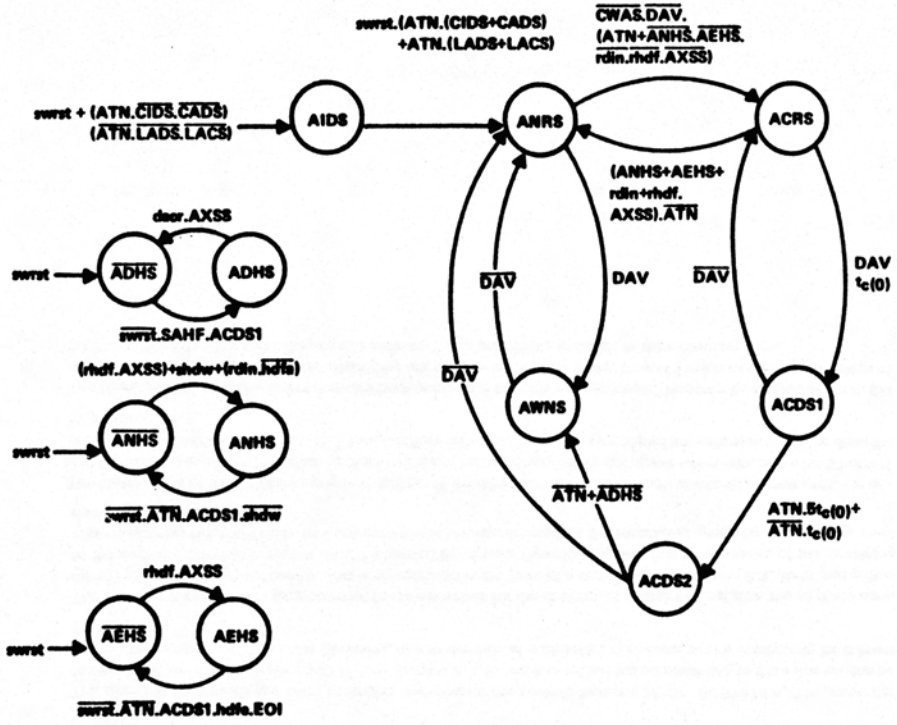


FIGURE 3-2 – TMS9914A ACCEPTOR HANDSHAKE STATE DIAGRAM

TABLE 3-2 – ACCEPTOR HANDSHAKE MNEMONICS

MESSAGES			STATES		
swrst	=	software reset	AIDS	=	acceptor idle state
dacr	=	DAC release	ANRS	=	acceptor not ready state
rhdf	=	release RFD holdoff	ACRS	=	acceptor ready state
shdw	=	shadow handshake	ACDS1	=	accept data state 1
rdin	=	read data in register	ACDS2	=	accept data state 2
hdfe	=	enable RFD holdoff after END messages received	AWNS	=	acceptor wait for new cycle state
hdfa	=	enable RFD holdoff on all data	ADHS	=	accept data holdoff state
ATN	=	attention	ANHS	=	acceptor not ready holdoff state
DAV	=	data valid	AEHS	=	acceptor not ready holdoff after 'END'
EOI	=	end or identify state	CWAS	=	controller wait for ANRS state (controller function)
RFD	=	ready for data	AXSS	=	auxiliary command strobe state (auxiliary command register)
DAC	=	data accepted	LADS	=	listener addressed state (listener function)
SAHF	=	set accept data holdoff state	LACS	=	listener active state (listener function)
$t_{c(0)}$	=	clock cycle time	CIDS	=	controller idle state (controller function)
			CADS	=	controller addressed state (controller function)

TABLE 3-3 – ACCEPTOR HANDSHAKE MESSAGE OUTPUTS

STATE	REMOTE MESSAGES SENT		OTHER ACTIONS	
	RFD	DAC		
AIDS	(T)	(T)		
ANRS	F	F		
ACRS	(T)	F		
ACDS1	F	F	ATN False:	– data entered into Data In Register – BI interrupt generated – end interrupt generated if EOI is true.
			ATN true:	– commands decoded – command related interrupts set – sahf set if command requires a DAC holdoff – TR pin set true if GET message is received – 'pts' feature cleared after UNC interrupt set
ACDS2	F	F	TR	pin set true if GET command was received in ACDS1
AWNS	F	(T)		

3.3 SOURCE HANDSHAKE

The TMS9914A source handshake state diagram is shown in Figure 3-3. IEEE-488 states SIWS and SWNS have been removed. These record the false then true transition of 'nba' (new byte available) as the old data byte is removed and a new data byte is made ready. Instead the TMS9914A uses a separate state (SHFS) to record the availability of a data byte in the Data Out Register. This state is exited when a byte is written to the Data Out Register which enables the transition from SGNS to SDYS and the subsequent transmission of the byte. The SHFS is reentered as the byte is sent in STRS, but if the handshake is interrupted before this, then the fact that the byte has not been sent is recorded until the source handshake again becomes active. If, however, the byte in the data out register is to be disregarded, then 'nbaf' may be used to return the device to SHFS.

The status byte in the Serial Poll Register is continually available. The transition from SGNS to SDYS is not dependent on SHFS during a serial poll, that is, while SPAS is active. By separately recording the availability of a byte in the Data Out Register, a talker sending data may be interrupted for a serial poll without risk of a byte being lost.

The additional state SERS is included to detect an error condition on the bus. This will be entered when the source handshake tries to send a byte but finds both the NRFD and NDAC lines false at the same time. This condition will normally indicate for a controller that there are no devices powered up on the bus, or for a talker that there are no devices addressed to listen on the bus.

The state VSTS will be entered after the first data byte of a talker has been sent if the 'vstdl' feature is enabled. This enables a very short bus settling time ($4t_{c(0)}$) for all subsequent bytes until ATN next becomes true. The TMS9914A will not use the short bus settling time when it is an active controller.

NOTE

The TMS9914 did not implement the 'VSTS'. The bus settling time was therefore either $8t_{c(0)}$ or $12t_{c(0)}$ for 'stdl' set and not set respectively.

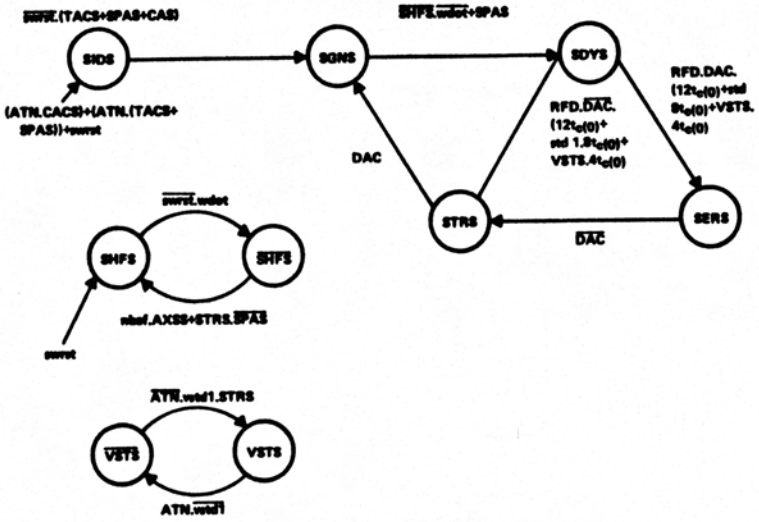


FIGURE 3-3 – TMS9914A SOURCE HANDSHAKE STATE DIAGRAM

TABLE 3-4 – SOURCE HANDSHAKE MNEMONICS

MESSAGES			STATES		
swrst	=	software reset	SIDS	=	source idle state
nbaf	=	new byte available false	SGNS	=	source generate state
wdot	=	write to the data out register	SDYS	=	source delay state
stdl	=	enable short bus settling time	SERS	=	source error state
vstdl	=	enable very short bus settling time	STRS	=	source transfer state
ATN	=	attention	SHFS	=	source holdoff state
RFD	=	ready for data	VSTS	=	very short bus settling time state
DAC	=	data accepted	TACS	=	talker active state (talker function)
$t_{c(0)}$	=	clock cycle time	CACS	=	controller active state (controller function)
			SPAS	=	serial poll active state (talker function)
			AXSS	=	auxiliary command strobe state (auxiliary command register)

TABLE 3-5 – SOURCE HANDSHAKE MESSAGE OUTPUTS

STATE	REMOTE MESSAGES SENT DAV	OTHER ACTIONS
SIDS	(F)	BO interrupt and ACCRQ set
SGNS	F	true if SHFS is false and SPAS is not true
SDYS	F	
SERS	F	ERR interrupt set true
STRS	T	

3.4 TALKER AND LISTENER FUNCTIONS

Figures 3-4 and 3-5 show the TMS9914A listener and talker state diagrams, which serve the purpose of the listener and talker or extended listener and extended talker functions of IEEE-488, depending on the state of the APT interrupt mask bit.

The TMS9914A does not recognize secondary addresses on-chip and these must be passed through to the host MPU for verification. Secondary addressing is enabled by unmasking the APT interrupt. A secondary address will cause this interrupt if the last primary command received was a primary address of the device, that is, it is in TPAS or LPAS. A DAC holdoff will also occur. The host MPU must respond to the interrupt by reading the secondary from the Command Pass Through Register and identifying it as being valid or not valid. The holdoff may then be released with a 'decr' auxiliary command, the sense of the 'cs' bit being used to indicate a valid (cs = 1) or not valid (cs = 0) secondary. If a valid secondary address is indicated then the TMS9914A will enter TADS or LADS depending on whether it is in TPAS or LPAS.

The 'lon' and 'ton' auxiliary commands together with the clear/set bit (cs) have a direct influence on the appropriate state diagrams. Therefore, although they appear as ordinary clear/set auxiliary commands, they can be effectively cleared by other bus events. For example, if a TMS9914A addresses itself as a listener via the 'lon' command it may be returned to LIDS by an UNL command from the bus at a later time.

The 'lon' and 'ton' auxiliary commands are used to implement two features of IEEE-488. First, talk only and listen only are used in situations where there is no active controller on the bus. Note that the 'lon' and 'ton' commands are linked with these features to indicate to the user that these commands are not enabled by CAS as are 'ltn' and 'lun' of IEEE-488.

Second, the 'lon' and 'ton' auxiliary commands are used by an active controller to address itself. IEEE-488 provides for a controller to address itself to listen via the 'ltn' and 'lun' message but there is no corresponding message for the talker. Hence, when a controller addresses itself to talk via 'ton,' it must send its talk address over the bus and similarly, if it sends another talk address over the bus then it must un-address itself by writing 'ton' false.

When the TMS9914A enters SPAS, the contents of the serial poll register are sampled and presented on DIO(8-1). These will remain unchanged until SPAS is exited. The source handshake will, however, send this status byte as many times as the controller will accept it.

The internal IFC signal of the TMS9914A (IFCIN) is suppressed when the device itself is sending IFC in order to simplify implementation of the controller function (see Section 3.8.3). Therefore, the send interface clear (sic) auxiliary command is included with IFCIN to return the talker and listener functions to their idle states and allow a system controller to clear its own interface.

A separate state diagram is included to control the sending of the END message of IEEE-488. If the 'feoi' auxiliary command is written followed by loading a byte into the Data Out Register, the TMS9914A will enter ERAS, and the EOI line will be asserted as 'DIO(8-1)' begin to change. The function will enter ENAS as soon as the source handshake begins to send this byte, and EOI will be released when the Data Out Register is next loaded. If it is desired to send EOI true with the next byte as well, then 'feoi' may be written before the Data Out Register returns the device to ERAS.

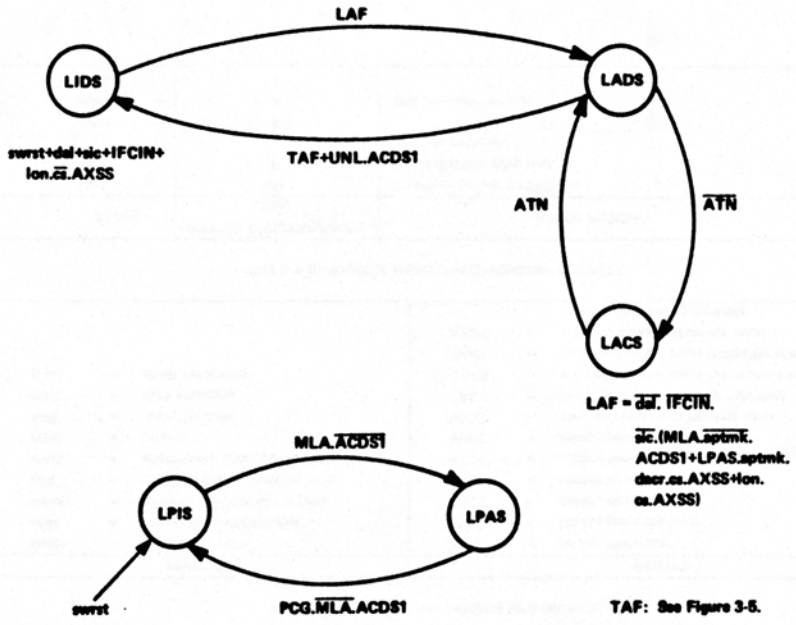


FIGURE 3-4 – TMS9914A LISTENER STATE DIAGRAM

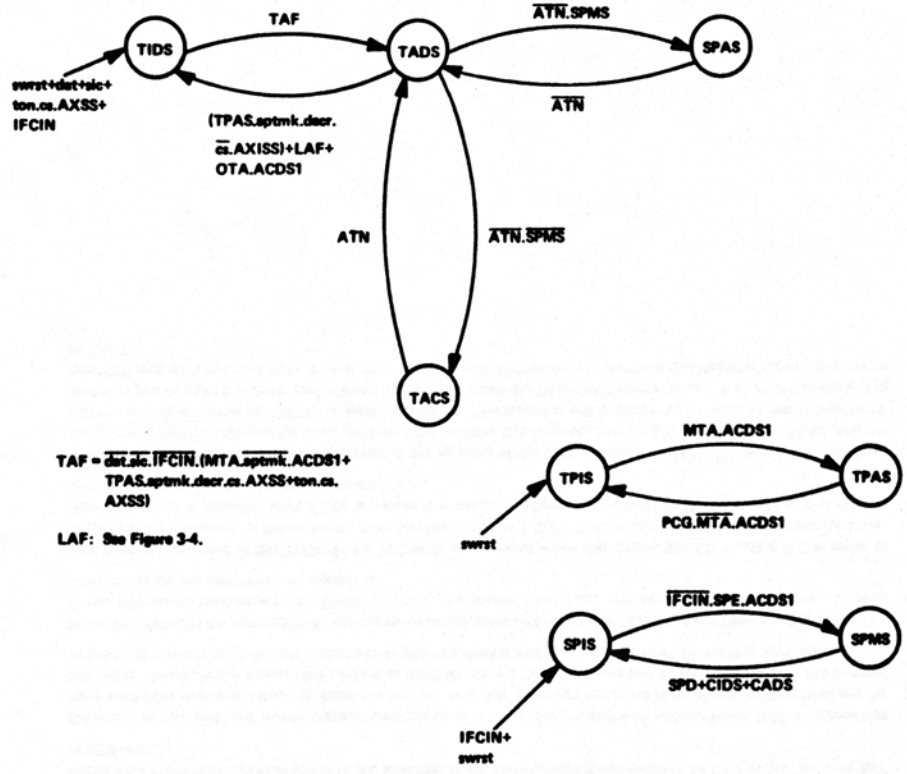


FIGURE 3-5 – TMS9914A TALKER STATE DIAGRAM

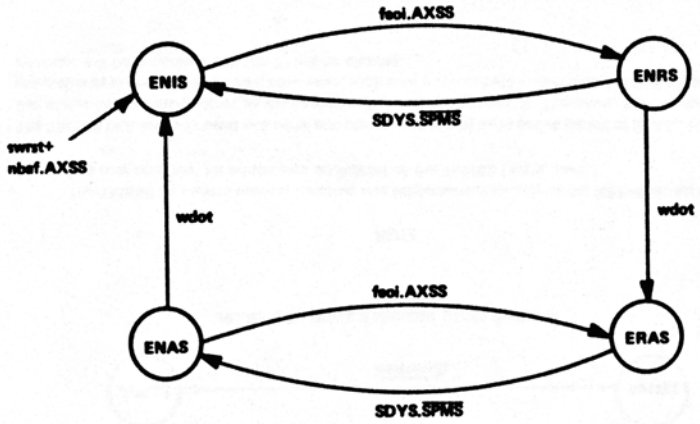


FIGURE 3-6 – TMS9914A TALKER STATE DIAGRAM (Continued)

TABLE 3-6 – TALKER AND LISTENER MNEMONICS

MESSAGES			STATES		
swrst	=	software reset	LIDS	=	listener idle state
dal	=	disable listener	LADS	=	listener addressed state
dat	=	disable talker	LACS	=	listener active state
sic	=	send interface clear	LPIS	=	listener primary idle state
lon	=	listen only	LPAS	=	listener primary addressed state
ton	=	talk only	TIDS	=	talker idle state
cs	=	clear/set bit of the auxiliary command register	TADS	=	talker addressed state
decr	=	release 'DAC' holdoff	TACS	=	talker active state
aptmk	=	address pass through interrupt mask	SPAS	=	serial poll active state
nbaf	=	new byte available false	SPIS	=	serial poll idle state
feoi	=	force 'EOI'	SPMS	=	serial poll mode state
wdot	=	write to the Data Out Register	TPIS	=	talker primary idle state
ATN	=	attention	TPAS	=	talker primary addressed state
IFCIN	=	internal interface clear message (a debounced signal, suppressed by 'sic')	ENIS	=	end idle state
EOI	=	end or identify	ENRS	=	end ready state
PCG	=	primary command group	ERAS	=	end ready and active state
MLA	=	my listen address	ENAS	=	end active state
MTA	=	my talk address	SDYS	=	source delay state (source handshake)
OTA	=	other talk address	CIDS	=	controller idle state (controller function)
SPE	=	serial poll enable	CADS	=	controller addressed state (controller function)
SPD	=	serial poll disable	ACDS1	=	accept data state 1 (acceptor handshake)
UNL	=	unlisten	AXSS	=	auxiliary command strobe state (auxiliary command register)
PCG	=	primary command group			

TABLE 3-7 – TALKER FUNCTION MESSAGE OUTPUTS

STATE	QUALIFIER	REMOTE MESSAGES SENT		OTHER ACTIONS
		RQS	EOI	DIO(8-1)
TIDS		(F)	(F)	(NUL)
TADS		(F)	(F)	(NUL)
TACS	ENIS.ENRS	(F)	F	DATA OUT REG
TACS	ENAS.ERAS	(F)	T	DATA OUT REG
SPAS	NPRS.SRQS	F	F	SERIAL POLL REG
SPAS	APRS1.APRS2	T	F	SERIAL POLL REG

3.5 SERVICE REQUEST FUNCTION

Figure 3-6 shows the state diagram for the TMS9914A service request function. The device has two means of implementing the request service (rsv) local message of IEEE-488: the first, 'rsv1', is bit 7 of the Serial Poll Register; the second is the auxiliary command 'rsv2'. These are simply ORed together to provide an input to the service request function, and, in any particular application, only one would normally be used, the other being left in its hardware reset state.

The affirmative poll response state (APRS) of IEEE-488 is split into two states on the TMS9914A for the following reason: Consider the case where a device has requested service, has been serial polled, and then wishes to request service again. The host MPU must clear the 'rsv' message and then set it true again. Now suppose this temporary false condition happens within one occurrence of SPAS. If the service request function has been implemented exactly as per IEEE-488, it will not be recognized, and SRQ will not be asserted a second time. Therefore, 'rsv' may only be cleared when the device is known not to be in SPAS, which can only happen if it is cleared as a consequence of some pre-arranged action of the controller. This action would normally be a part of the service routine executed by the controller as a response to the request for service. For example, if service was requested by an instrument which had some data to send for processing or to a printing device then 'rsv' could be cleared when it is addressed to talk and send its data over the bus.

For many applications, the fact that the device has been serial polled after requesting service is considered sufficient response from the controller. The 'rsv' local message therefore simply becomes a request for the controller to read its serial poll status byte. It is then desirable to be able to clear and reassert 'rsv' at any time after the serial poll status byte has been polled and the SPAS interrupt set. The TMS9914A is able to record a false transition of 'rsv1' or 'rsv2' by moving from APRS1 to APRS2 even if the device is in SPAS. This makes the above approach to serial polling possible.

To further support this approach, the 'rsv2' auxiliary command is automatically cleared when the serial poll status byte is polled, ensuring that "rsv2' is cleared before a second serial poll can occur. If this were not the case, then the same status byte might be polled twice by the controller with the RQS bit true, which may indicate that two reasons for requiring service have arisen.

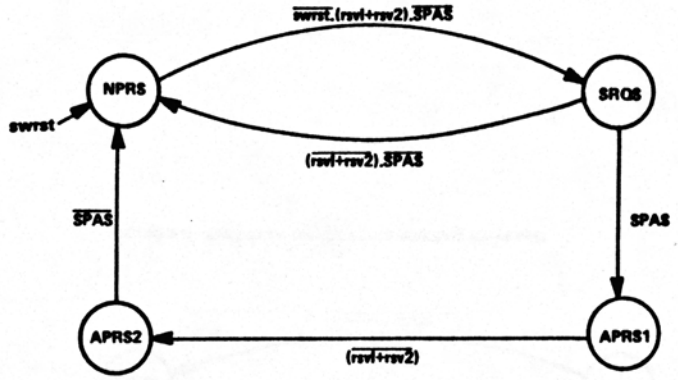


FIGURE 3-6 – SERVICE REQUEST STATE DIAGRAM

NOTE

The TMS9914 service request function was implemented exactly as per IEEE-488. Also, it had only one 'rsv' bit which was equivalent of the TMS9914A's 'rsv1.'

The TMS9914A will only send one serial poll status byte during each active period of SPAS., However, it will send this status byte as many times as the controller is prepared to accept it. Therefore, the controller should only read the status byte once per serial poll; otherwise, each time a status byte is sent with the RQS message true, the SPAS interrupt will be generated and 'rsv2' will be cleared.

TABLE 3-8 – SERVICE REQUEST MNEMONICS

MESSAGES			STATES		
swrst	=	software reset	NPRS	=	negative poll response state
srv1	=	request service 1 (bit 7 of serial poll register)	SRQS	=	service request state
rsv2	=	request service 2 (auxiliary command register)	APRS1	=	affirmative poll state 1
			APRS2	=	affirmative poll state 2
			SPAS	=	serial poll active state (talker function)

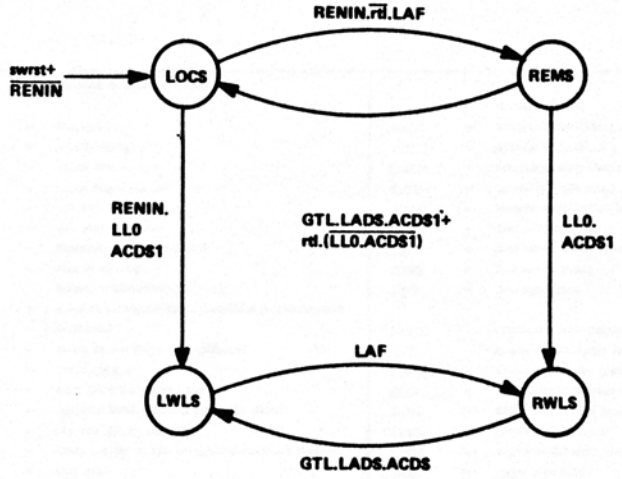
TABLE 3-9 – SERVICE REQUEST MESSAGE OUTPUTS

STATE	REMOTE MESSAGES SENT SRQ	OTHER ACTIONS
NPRS	(F)	
SRQS	T	
APRS1	(F)	– rsv2 cleared if in SPAS and STRS – SPAS interrupt set if in SPAS when STRS is exited
APRS2	(F)	– same as APRS1

3.6 REMOTE/LOCAL FUNCTION

The TMS9914A remote local state diagram is shown in Figure 3-7. It differs little from that of IEEE-488.

The complete listener function (LAF) is used to effect the transition from LOCS to REMS or from LWLS to RWLS. This means that if the APT interrupt is masked, the device will enter one of the remote states in response to its listen address, but if secondary addressing is enabled, then this will not happen until 'decr' is written with 'cs' true in response to a valid secondary address. In addition, the transition to one of the remote states will occur if 'lon' is used to address the device to listen.



LAF: See Figure 3-4

FIGURE 3-7 – TMS9914A REMOTE LOCAL STATE DIAGRAM

TABLE 3-10 – REMOTE/LOCAL MNEMONICS

MESSAGES			STATES		
swrst	=	software reset	LOCS	=	local state
rtl	=	return to local	REMS	=	remote state
RENIN	=	internal remote enable message (debounced)	RWLS	=	remote with lockout state
GTL	=	go to local	LWLS	=	local with lockout state
LLO	=	local lockout	LADS	=	listener addressed state (listener function)
			ACDS1	=	accept data state 1 (acceptor handshake)

3.7 PARALLEL POLL FUNCTION

The parallel poll function of the TMS9914A only nominally supports logically-configured parallel poll. With a suitable software package, remotely-configured parallel poll may also be easily implemented. The state diagram is shown in Figure 3-8.

When the EOI and ATN lines become true simultaneously (the Identify message), the contents of the Parallel Poll Register are output to DIO(8-1). If parallel poll is to be used in a particular bus environment, then the Pull-Up Enable (PE) input of the SN75160 must be held low so that the DIO(8-1) are driven by open collector buffers. Parallel Poll, occurring when the Parallel Poll Register is in the hardware reset condition of all zeros, will result in none of DIO(8-1) being pulled low. This corresponds to the parallel poll idle state (PPIS). If it is desired to participate in a parallel poll, then the bit corresponding to the desired parallel poll response is set true. This implements the parallel poll standby state (PPSS), and, when the Identify message becomes true, the appropriate line of DIO(8-1) is pulled low. This is equivalent to the parallel poll active state (PPAS). Only one bit of the parallel Poll Register should be set true at once.

3.7.1 Remotely Configured Parallel Poll

The parallel poll configure command (PPC) is treated by the TMS9914A as an unrecognized addressed command. It is passed through when the TMS9914A is in LADS. If an instrument is to be remotely configured for parallel poll, then the pass through next secondary (pts) auxiliary command should be written before releasing the DAC holdoff. This will cause the next command received to also set a UNC interrupt if it is a secondary command. The secondary command will be either the parallel poll enable command (PPE) or the parallel poll disable command (PPD) and should be read from the Command Pass Through Register and identified. If it is the PPE command, then the attendant bits (S, P1, P2, P3) should be extracted and stored by the host MPU (see Section 2.9.3 of IEEE-488 1978). The S bit should then be matched against the individual status of the instrument (represented by 'ist'), and if they are the same, the bit corresponding to the parallel poll response, specified by P1, P2, P3, should be set true in the Parallel Poll Register. If this is not the case, then the Parallel Poll Register should be cleared if it is not already clear. After this, each time the individual status of the device changes, the 'ist' should again be matched against the S bit and the Parallel Poll Register updated accordingly until PPD or PPU is received.

If a PPD command is passed through after the 'pts' feature has been written, the Parallel Poll Register should be cleared before the DAC holdoff is released. The PPC command that precedes PPD is an address command; it is a means of eliminating individual members of a parallel poll. The parallel unconfigure command is treated by the TMS9914A as an unrecognized universal command. When it is passed through, the host MPU should clear its Parallel Poll Register before releasing the DAC holdoff. This command will clear all members of a parallel poll.

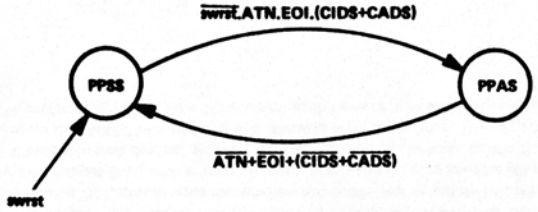


FIGURE 3-8 – TMS9914A PARALLEL POLL STATE DIAGRAM

TABLE 3-11 – PARALLEL POLL MNEMONICS

MESSAGES			STATES		
swrst	=	software reset	PPSS	=	parallel poll standby state
ATN	=	attention	PPAS	=	parallel poll active state
EOI	=	end or identify	CIDS	=	controller idle state (controller function)
			CADS	=	controller addressed state (controller function)

TABLE 3-12 – PARALLEL POLL MESSAGE OUTPUTS

STATE	REMOTE MESSAGES SENT DIO(8-1)	OTHER ACTIONS
PPSS	(NUL)	
PPSS	PARALLEL POLL REG*	

* If there is a true bit in the Parallel Poll Register, it must be sent active; any false bit must be sent passive.

3.8 CONTROLLER FUNCTION

The controller function of the TMS9914A is greatly simplified compared with that of IEEE-488. It relies heavily on software support but, with suitable software, it enables all subsets of the controller function to be implemented. With this approach the controller logic is reduced to a small proportion of the chip area which means that the device may be economically used in situations where a talker/listener only is required.

Figure 3-9 shows the controller function state diagram. With suitable software, it will perform the full controller function, as described in the IEEE-488A 1980 supplement to the IEEE-488 1978. It therefore includes the additional state CSHS, which allows time for DAV to be recognized false by all devices on the bus before ATN is asserted. The 'tca' local message is implemented by an immediate execute auxiliary command. The state CWAS is therefore added to record the occurrence of this command until the acceptor handshake enters ANRS and the device can enter CSHS. The 'tca' auxiliary command also causes entry into CSHS although IEEE-488A 1980 allows it to move directly from CSBS to CSWS. This is done for convenience of implementation and results in the 'tca' auxiliary command taking an extra 1.6 microseconds to assert ATN.

The delay between CSWS and CAWS is slightly less than specified in IEEE-488A 1980 but the total time taken in moving from CSWS to CACS is still greater than the specified minimum.

The Controller Parallel Poll State (CPPS) is not included on the TMS9914A. To conduct a parallel poll, a TMS9914A based controller must set the 'rpp' clear/set auxiliary command true when it is in CACS, moving it to CPWS which sends EOI true. The host MPU must then wait 2 microseconds before reading back the parallel poll responses via the Command Pass Through Register. The 'rpp' auxiliary command can then be cleared, EOI will go false, and the parallel poll is complete. The host MPU will receive a BO interrupt as soon as the TMS9914A reenters CACS and the source handshake becomes active.

3.8.1 Controller Self Addressing

As discussed in Section 3.2, the acceptor handshake does not operate when the controller is active. This means commands being sent are not monitored, and special precautions are required as a consequence of this when addressing devices and when passing control.

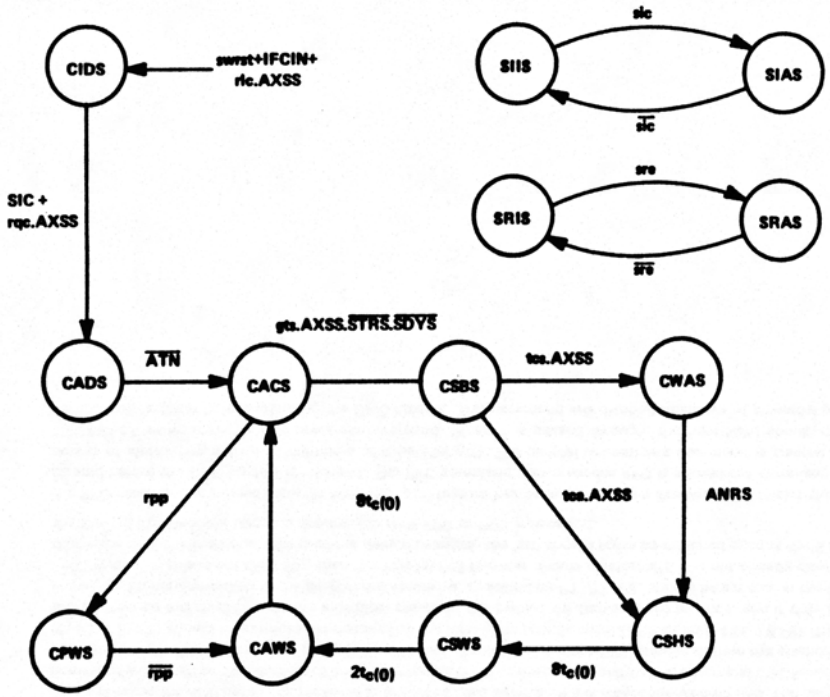


FIGURE 3-9 – TMS9914A CONTROLLER STATE DIAGRAMS

TABLE 3-13 – CONTROLLER FUNCTION MNEMONICS

MESSAGES			STATES		
swrst	=	software reset	CIDS	=	controller idle state
sic	=	send interface clear	CADS	=	controller addressed state
sre	=	send remote enable	CACS	=	controller active state
rqc	=	request control	CSBS	=	controller standby state
rlc	=	release control	CSHS	=	controller standby hold state
gts	=	go to standby	CSWS	=	controller synchronous wait state
tcs	=	take control synchronously	CAWS	=	controller active wait state
tca	=	take control asynchronously	CPWS	=	controller parallel poll wait state
rpp	=	request parallel poll	ANRS	=	acceptor not ready state (acceptor handshake)
IFCIN	=	internal interface clear message (a debounced signal which is suppressed if 'sic' is true)	SDYS	=	source delay state (source handshake)
ATN	=	attention	STRS	=	source transfer state (source handshake)
$t_{c(0)}$	=	clock cycle time	AXSS	=	auxiliary command strobe state (auxiliary command register)
			CWAS	=	controller wait for ANRS state

TABLE 3-14 – CONTROLLER FUNCTION MESSAGE OUTPUTS

STATE	REMOTE MESSAGE SENT			OTHER ACTIONS
	ATN	EOI	DIO(8-1)	
CIDS	(F)	(F)	(NUL)	
CADS	(F)	(F)	(NUL)	
CACS	T	F	DATA OUT REG	Data Out Reg. may contain any of the commands in Table 3-15
CSBS	F	(F)	(NUL)	
CWAS	F	(F)	(NUL)	
CSHS	F	(F)	(NUL)	
CSWS	T	F	(NUL)	
CAWS	T	F	(NUL)	
CPWS	T	T	(NUL)	DIO(8-1) may be read via the Command Pass Through Register

STATE	REMOTE MESSAGES SENT	OTHER ACTIONS
	IFC	
SIIS*	(F)	Internal interface
SIIS	F	clear message IF-CIN
SIAS	T	is held false

STATE	REMOTE MESSAGES SENT	OTHER ACTIONS
	REN	
SRIS*	(F)	
SRIS	F	
SRAS	T	

* Buffers not configured for a system controller; otherwise, buffers are configured for system controller.

When the controller is active, it uses 'ton' or 'lon' to address and unaddress itself. IEEE-488 provides for the controller to locally address itself to listen, but there is no corresponding local message for the talker. The TMS9914A should always accompany a 'ton' auxiliary command with 'cs' true with its own talk address or an UNT command sent over the bus. Similarly, if the TMS9914A sends the talk address of another device over the bus, it should ensure that it is in TIDS by writing the 'ton' auxiliary command false.

Appendix B shows some typical sequences of events when the controller addresses itself, goes to standby, takes control again, etc.

3.8.2 Passing Control

As Figure 3-9 shows, the controller transfer state (CTRS) of IEEE-488 is not present, and all transitions associated with the TCT command have been removed. Instead, two immediate execute auxiliary commands are included. Request control (rqc) will cause a transition from CIDS to CADS, and the release control command (rlc) will return the function to CIDS. The TCT command is treated similarly to an unrecognized addressed command but will cause a UNC interrupt if the device is in TADS.

Figure 3-10 is a representation of the sequence of events involved in passing control from one TMS9914A based device to another. The device passing control must initially ensure that it is not in TADS; then it should send out the talk address of the device to receive control. The receiving device will enter TADS, and after any DAC holdoff has been released, the host MPU of the device passing control will set a BO interrupt indicating that it may then send the TCT command. The TCT command will cause a UNC interrupt to the host MPU of the receiving device, and also a DAC holdoff will occur. The host MPU of the receiving device must examine its Command Pass Through Register, and upon identifying TCT, should write the auxiliary command 'rqc' to put its TMS9914A into CADS. The receiving device may then release DAC with a 'dacr' auxiliary command causing another BO interrupt at the device passing control. This indicates that the 'rlc' auxiliary command may then be used by the host MPU of the device passing control to return its TMS9914A to CIDS and allowing ATN to go false. The receiving device then enters CACS, asserts ATN, and its host MPU gets a BO interrupt as the source handshake becomes active. The passing of control is complete.

3.8.3 System Controller

The TMS9914A has no on-chip means of determining whether or not it is the system controller. Instead, this is determined by the software and by the configuration of the buffers to the IEEE-488 bus.

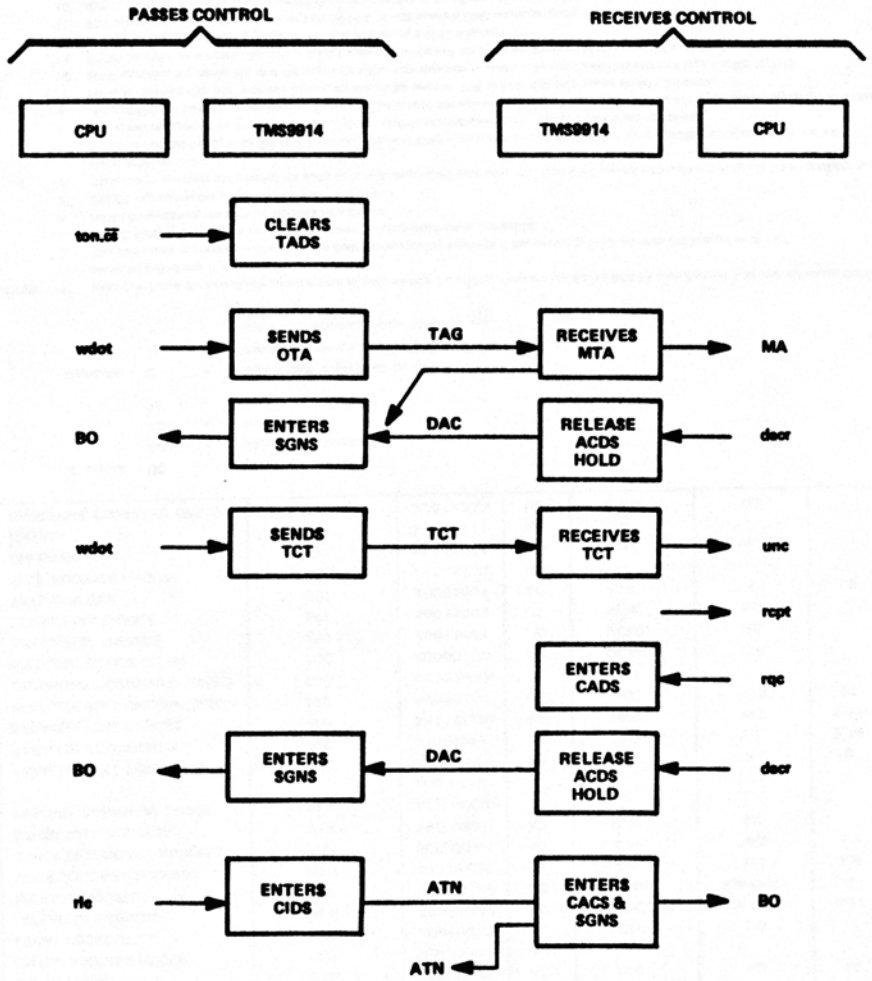
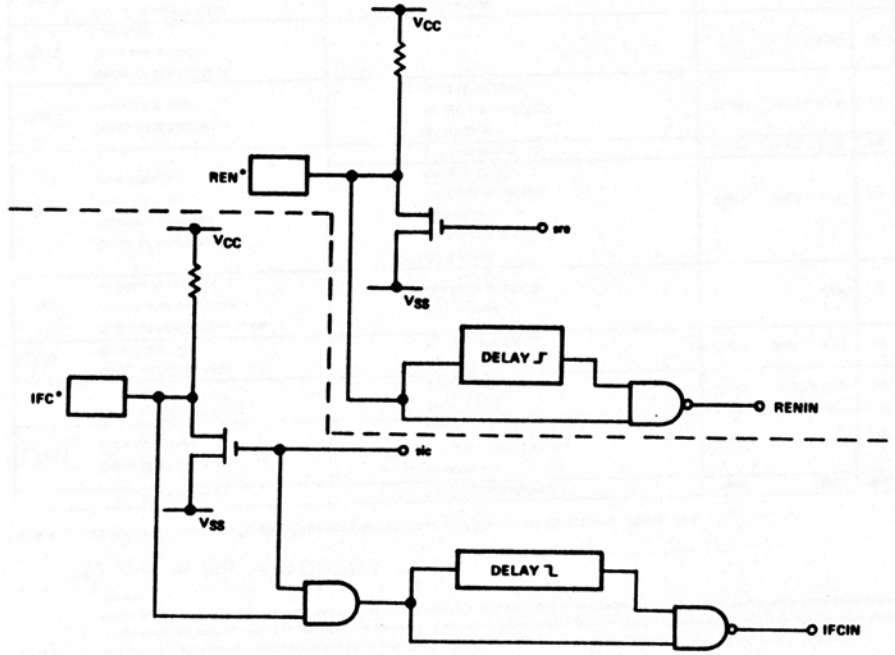


FIGURE 3-10 – PASSING CONTROL BETWEEN TMS9914s

The REN and IFC outputs of the TMS9914A are controlled by the auxiliary commands 'sre' and 'sic'. These should never be used by the host MPU of a device unless it is the system controller. As may be seen from Figure 3-11, the REN and IFC outputs of the TMS9914A are open drains with internal pull-ups. This means that the outputs are capable of driving the inputs of the buffers if the device is a system controller. If not, the buffers will drive into the REN and IFC pins and override the pull-ups. Hence, no direction control is required.

The false transition of REN and the true transition of IFC are both debounced to prevent noise on these lines from causing permanent state changes on the TMS9914A. In addition, the internal interface clear signal (IFCIN) is held false if the TMS9914A is sending IFC. Figure 3-9 shows the reason for this. If the device is not a system controller, then the occurence of IFC will return the controller function to CIDS. If, however, the device is a system controller, when it asserts IFC and is in CIDS, the 'sic' auxiliary command will cause it to enter CADS. As IFCIN is suppressed, it will not be forced back into CIDS, and there will be no conflict.



* The REN and IFC signals are at the pins of the TMS9914A and are therefore negative logic signals. The remaining signals are conventional positive logic signals.

FIGURE 3-11 – IFC AND REN PINS

TABLE 3-15 – MULTILINE INTERFACE MESSAGES

COMMAND	SYMBOL	DIO 8 – 1	CLASS	INTERRUPT (1,2)	DAC (3) HOLDOFF	NOTE
ADDRESSED COMMAND GROUP	ACG	000XXXX	AC	–	–	
DEVICE CLEAR	DCL	X0010100	UC	DCAS	YES	
GROUP EXECUTE TRIGGER	GET	X0001000	AC	GET	YES	
GO TO LOCAL	GTL	X0000001	AC	RLC	NO	14
LISTEN ADDRESS GROUP	LAG	X01XXXXX	AD	–	–	
LOCAL LOCKOUT	LLO	X0010001	UC	NONE	NO	
MY LISTEN ADDRESS	MLA	X01AAAAA	AD	MA,MAC,RLC	MA ONLY	4,14
MY TALK ADDRESS	MTA	X10AAAAA	AD	MA,MAC	MA ONLY	4
MY SECONDARY ADDRESS	MSA	X11SSSSS	SE	APT	YES	5,6
OTHER SECONDARY ADDRESS	OSA	SCG.MSA-	SE	APT	YES	6,7
OTHER TALK ADDRESS	OTA	TAG.MTA-	AD	MAC	NO	
PRIMARY COMMAND GROUP	PCG	ACG+UCG+ LAG+TAG	–	–		
PARALLEL POLL CONFIGURE	PPC	X0000101	AC	UNC	YES	8
PARALLEL POLL ENABLE	PPE	X110SPPP	SE	UNC	YES	9,10
PARALLEL POLL DISABLE	PPD	X111DDDD	SE	UNC	YES	9,11
PARALLEL POLL UNCONFIGURE	PPU	X0010101	UC	UNC	YES	12
SECONDARY COMMAND GROUP	SCG	X11XXXXX	SE	–	–	
SELECTED DEVICE CLEAR	SDC	X0000100	AC	DCAS	YES	
SERIAL POLL DISABLE	SPD	X0011001	UC	NONE	NO	
SERIAL POLL ENABLE	SPE	X0011000	UC	NONE	NO	
TAKE CONTROL	TCT	X0001001	AC	UNC	YES	13
TALK ADDRESS GROUP	TAG	X10XXXXX	AD	–	–	
UNLISTEN	UNL	X0111111	AD	MAC	NO	
UNTALK	UNT	X1011111	AD	–	–	
UNIVERSAL COMMAND GROUP	UCG	X001XXXX	UC	NONE	NO	

Classes: UC – universal command
AC – addressed command
AD – address
SE – secondary command

Symbols: 0 – logical zero (high level on GPIB)
1 – logical one (low level on GPIB)
x – don't care (received message)

NOTES:
1. Interrupts listed are as a direct consequence of the command received. They are set during ACDS1 (see Section 3.2) and will cause the INT pin to be pulled low if unmasked.
2. The addressed commands will only cause their corresponding interrupt if the device is in LADS with the exception of TCT.
3. A DAC holdoff will only be caused if the corresponding interrupt is unmasked.
4. AAAAA represents the primary address of a device.
5. SSSSS represents the secondary address of a device.
6. Secondary addresses are handled via address pass through (APT interrupt). The host MPU should respond by writing the 'dacr' auxiliary command with 'cs' false.
7. If OSA is passed through via the APT interrupt, the host MPU should respond by writing the 'dacr' auxiliary command with 'cs' false.
8. PPC is not recognized by the TMS9914A and is therefore treated as an unrecognized addressed command.
9. PPE and PPD are secondary commands. These may be passed through to the host MPU using the 'pts' auxiliary command. When the PPC command is received the 'pts' auxiliary command should be written. PPE or PPD will then cause an APT interrupt.
10. SPPP specifies the sense bit, and the desired parallel poll response is a remotely configured parallel poll (see Section 3.7.1).
11. DDDD specifies don't care bits which must be sent as zeros but need not be decoded by the host MPU of the receiving devices.
12. PPU is not recognized by the TMS9914A and will cause a UNC interrupt.
13. TCT is not recognized directly by the TMS9914A. It will cause a UNC interrupt when the device is in TADS.
14. RLC is set if MLA or GTL causes an appropriate transition in the Remote/Local function.

4.4.3 Source Handshake Timing Characteristics Over Full Range of Operating Conditions (see Note 1)

PARAMETER		TEST CONDITIONS	MIN	MAX	UNIT
t_{d1}	Delay of DAV true from end of write operation to data out register	Normal T_1 (see Note 2)	12(ϕ)!	12(ϕ)! + 310	ns
		Short T_1 (see Note 2)	8(ϕ)!	8(ϕ)! + 310	ns
		Very short T_1 (see Note 2)	4(ϕ)!	4(ϕ)! + 310	ns
t_{d2}	Delay of valid GPIB data lines from end of write cycle			140	ns
t_{d3}	Delay of BO interrupt from DAC true	BO interrupt unmasked		300	ns
t_{d4}	Delay of $\overline{\text{ACCRQ}}$ DAC true			300	ns
t_{d5}	Delay of DAV false from DAC true			160	ns

NOTES:
1. The timing of the source handshake is the same whether ATN is true or false, i.e., whether the device is in TACS, CACS, or SPAS.
2. A very short bus settling time (T_1) occurs on the second and subsequent data byte sent when ATN is false if the 'vstd1' feature is set. A slightly longer bus settling time takes place if 'std1' is set unless there is a very short bus settling time. In all other instances, a normal bus settling time occurs.

3. Tc = To = 1000NS

4.4.4 Acceptor Handshake Timing Characteristics Over Full Range of Operating Conditions

PARAMETER		TEST CONDITIONS	MIN	MAX	UNIT
t_{d6}	Delay of BI interrupt from DAV true	BI interrupt unmarked ATN = false device is in LACS	2(ϕ)!	2(ϕ)! + 415	ns
t_{d7}	Delay of $\overline{\text{ACCRQ}}$ from DAV true	ATN = false device is in LACS	2(ϕ)!	2(ϕ)! + 290	ns
t_{d8}	Delay of NDAC false from DAV true	ATN = false device is in LACS	3(ϕ)!	3(ϕ)! + 445	ns
t_{d9}	Delay of NRFD false from end of read operation of Data In register	ATN = false device is in LACS		220	ns
t_{d10}	Delay of interface message interrupt from DAV true (see Note 3)	ATN = true device not in CACS all interface message interrupts (except UNO)	2(ϕ)!	2(ϕ)! + 415	ns
		UNO interrupt only	5(ϕ)!	5(ϕ)! + 415	ns
t_{d11}	Delay of NDAC false from DAV true	ATN = true device not in CACS no DAC holdoff	7(ϕ)!	7(ϕ)! + 415	ns
t_{d12}	Delay of NDAC false from end of write operation			230	ns
t_{d13}	Delay of NRFD false from DAV false	ATN = true device not in CACS		180	ns

NOTE 3: The interrupts generated by interface messages are shown in Table 4-1.

4.4.5 ATN, EOI, and IFC Timing Characteristics Over Full Range of Operating Conditions

PARAMETER		TEST CONDITIONS	MIN	MAX	UNIT
t_{d14}	Delay of NDAC true from ATN true	Device is not in CACS		195	ns
t_{d15}	Delay of TE high from EOI true	Device is not in CACS		125	ns
t_{d16}	Delay of valid data from EOI true	Device is not in CACS		140	ns
t_{d17}	Delay of TE low from EOI false	Device is not in CACS		125	ns
t_{d18}	Delay of NRFD true from ATN false	Device is in LADS/LACS		140	ns
t_{d19}	Response time to IFC		$16t_{c(0)}$	$30t_{c(0)}$	ns

4.4.6 Controller Timing Characteristics Over Full Range of Operating Conditions

	PARAMETER	TEST CONDITIONS	MIN	MAX	UNIT
t_{d20}	Delay of ATN true from end of t_{ca} aux command		$8t_{c(0)}$	$10(\phi)1+220$	ns
t_{d21}	Delay of BO interrupt from end of t_{ca} aux command		$18t_{c(0)}$	$22(\phi)1+415$	ns
t_{d22}	Delay of ATN true from end of t_{ca} aux command	BO unmasked device is in ANRS	$8t_{c(0)}$	$10(\phi)1+220$	ns
t_{d23}	Delay of BO interrupt from end of t_{ca} aux command	BO unmasked device is in ANRS	$18t_{c(0)}$	$22(\phi)1+415$	ns
t_{d24}	Delay of EOI true from r_{pp} aux command set			230	ns
t_{d25}	Delay of EOI false from r_{pp} aux command cleared			230	ns
t_{d26}	Delay of EOI from r_{pp} aux command cleared	BO unmasked	$8t_{c(0)}$	$10(\phi)1+415$	ns
t_{d27}	Delay of ATN false from s_{tg} aux command	Device is not in SDYS or STRS		210	ns

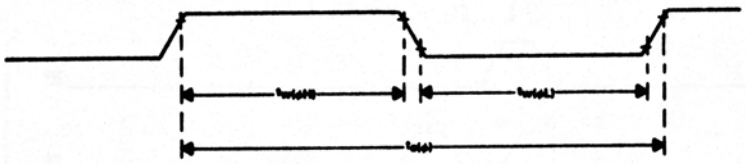
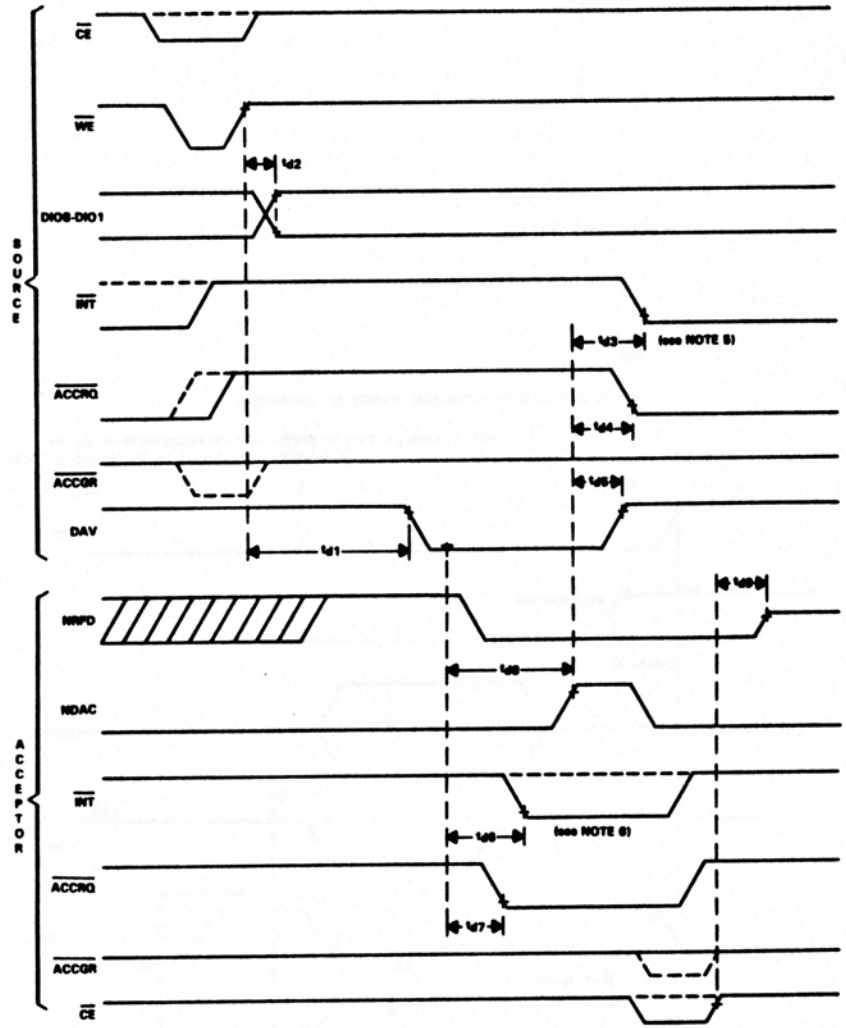
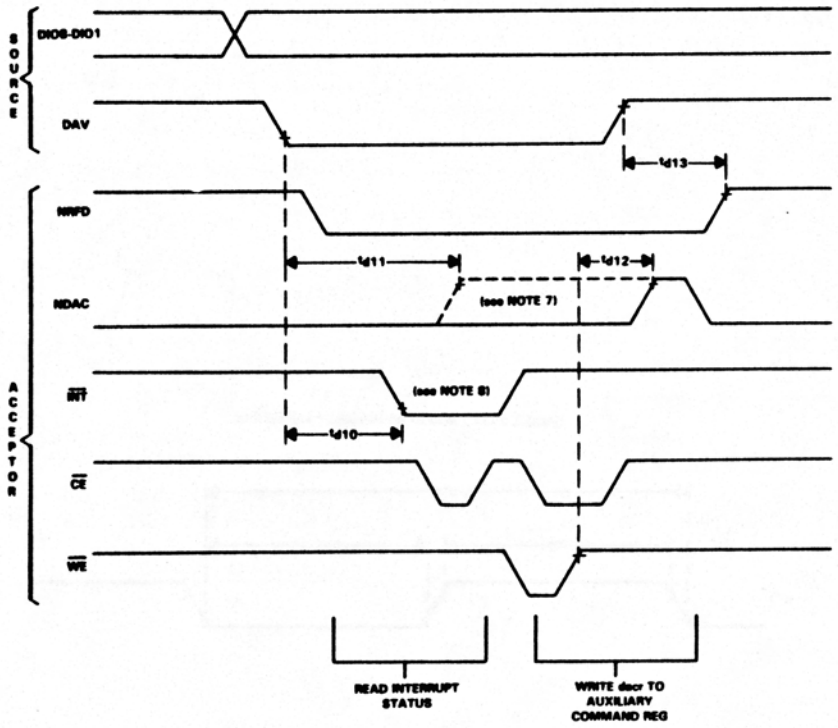


FIGURE 4-1 – TMS9914A CLOCK CYCLE TIMING



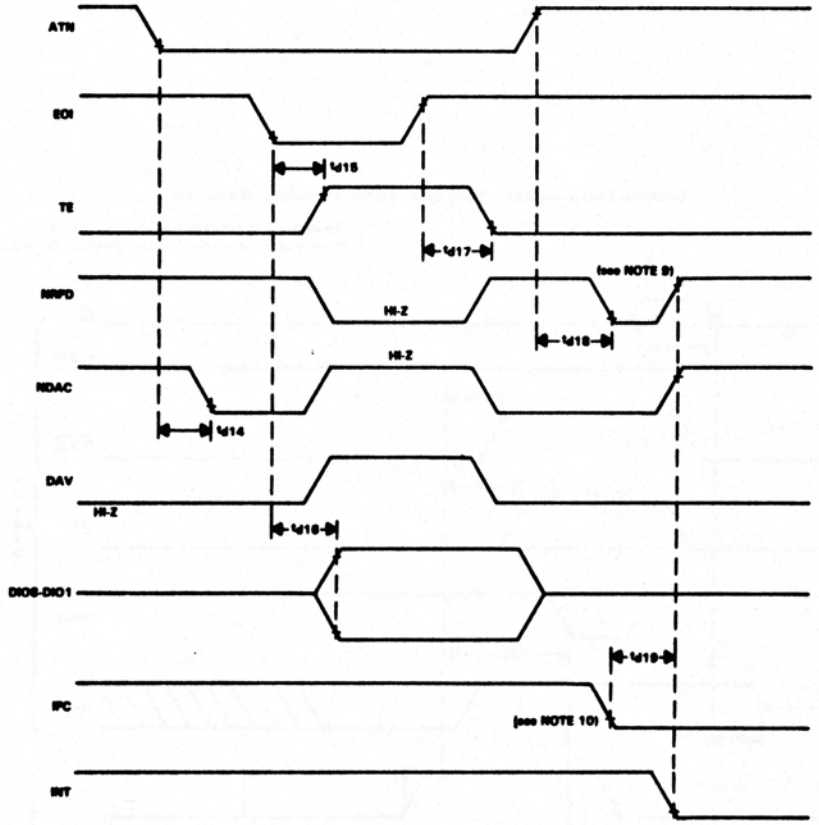
NOTES: 5. The interrupt line is taken low by a BO interrupt.
6. The interrupt line is taken low by a BI interrupt.

FIGURE 4-6 – TMS9914A SOURCE AND ACCEPTOR HANDSHAKE TIMING(S)



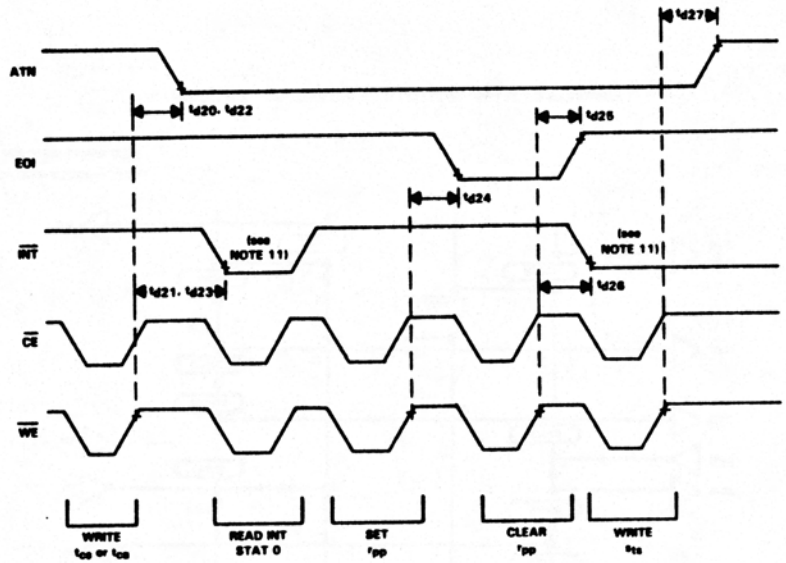
NOTES: 7. The broken line shows the waveform if there is no DAC holdoff. The solid lines assume there is a DAC holdoff.
8. The interrupts generated by interface messages are shown in Table 4-1.

FIGURE 4-7 – TMS9914A ACCEPTOR HANDSHAKE TIMING "ATN" TRUE



NOTES: 9. This assumes that an RFD holdoff occurs.
10. IFC causes the TMS9918A to be unaddressed and an IFC interrupt occurs.

FIGURE 4-8 – TMS9914A RESPONSE TO 'ATN' AND 'EOI'



NOTE 11: A BD interrupt occurs as the TMS9914A enters CACS.

FIGURE 4-9 – TMS9914A CONTROLLER TIMING

TYPICAL SEQUENCES OF EVENTS FOR THE CONTROLLER

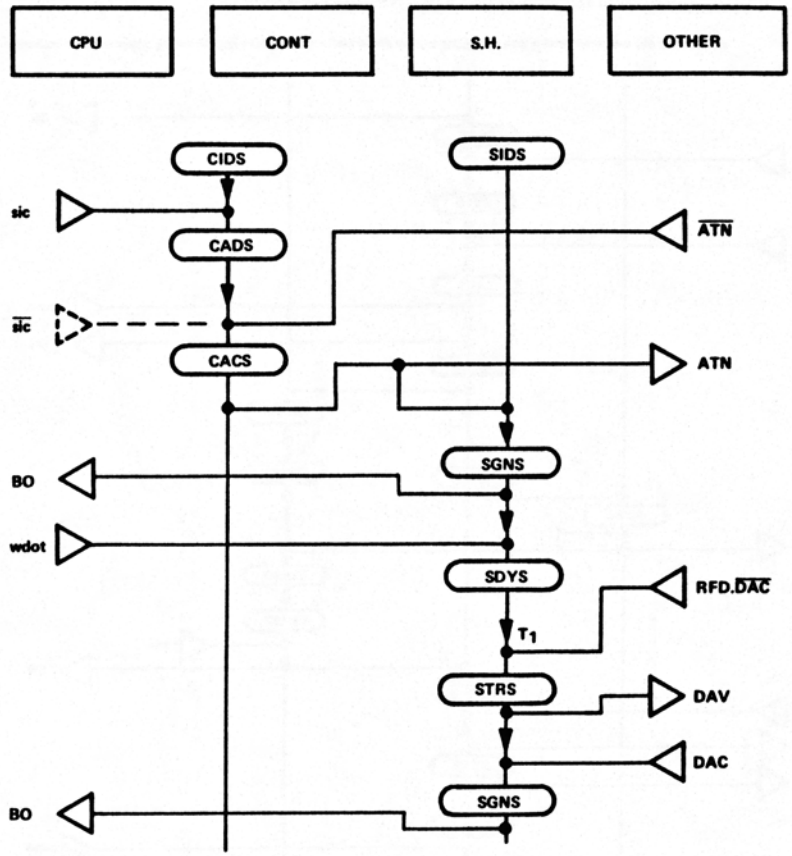
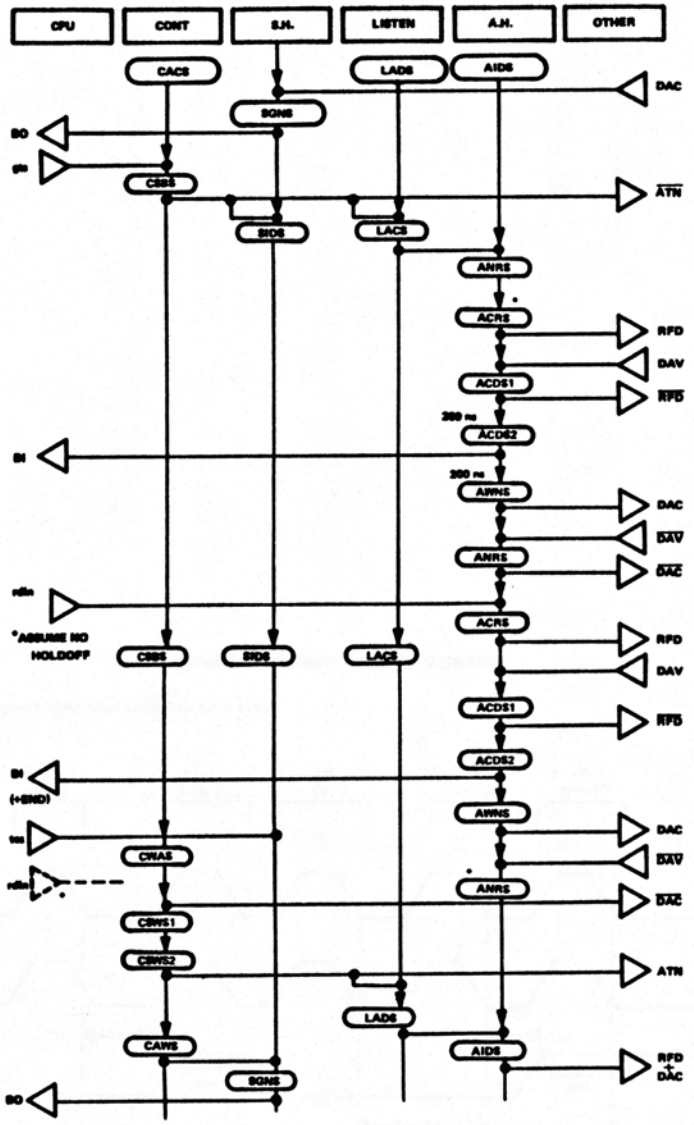
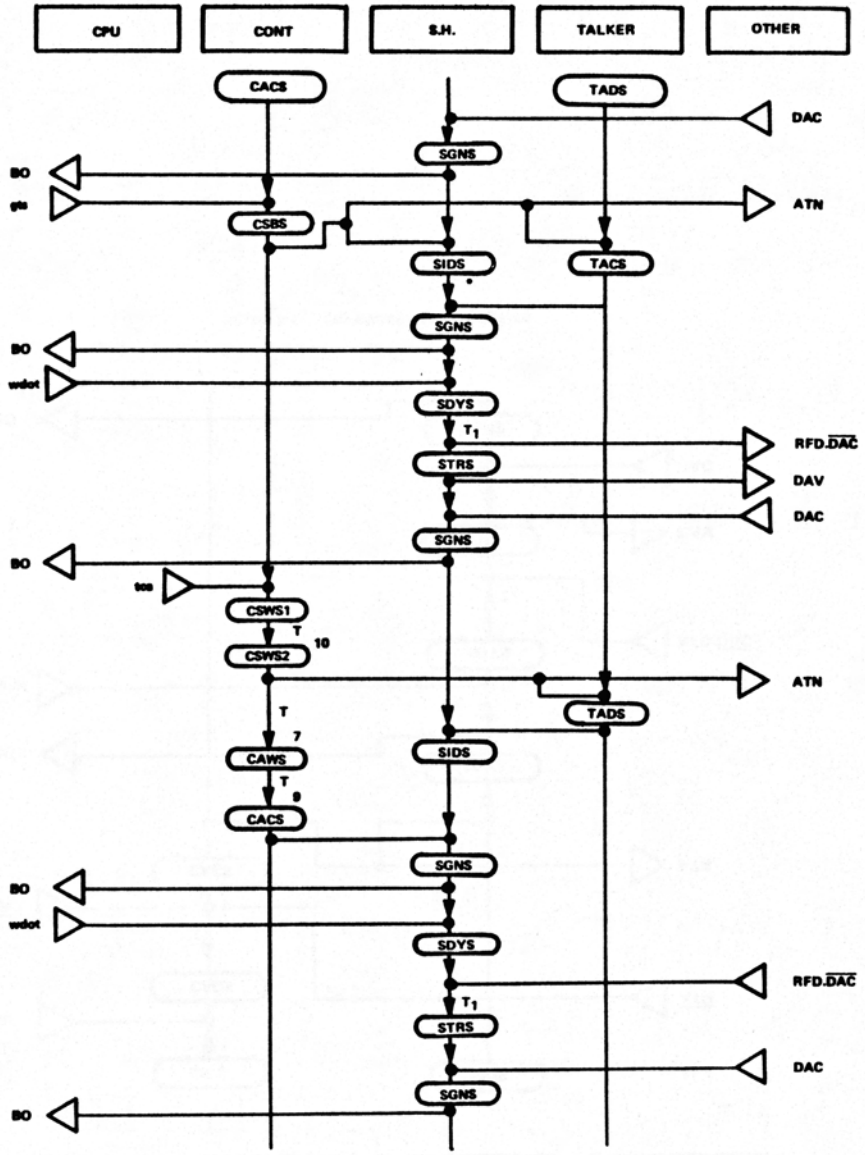


FIGURE 8-1 – CONTROLLER TAKING CONTROL



*CWAS inhibits ANRS → ACRS, so rdin can occur before ATN is set.

FIGURE 8-2 – CONTROLLER AS A LISTENER (GOING TO STANDBY)



*Momentary transition following BO interrupt may not occur on the TMS9914 but is guaranteed on the TMS9914A.

FIGURE 8-3 – CONTROLLER AS A TALKER (GOING TO STANDBY)

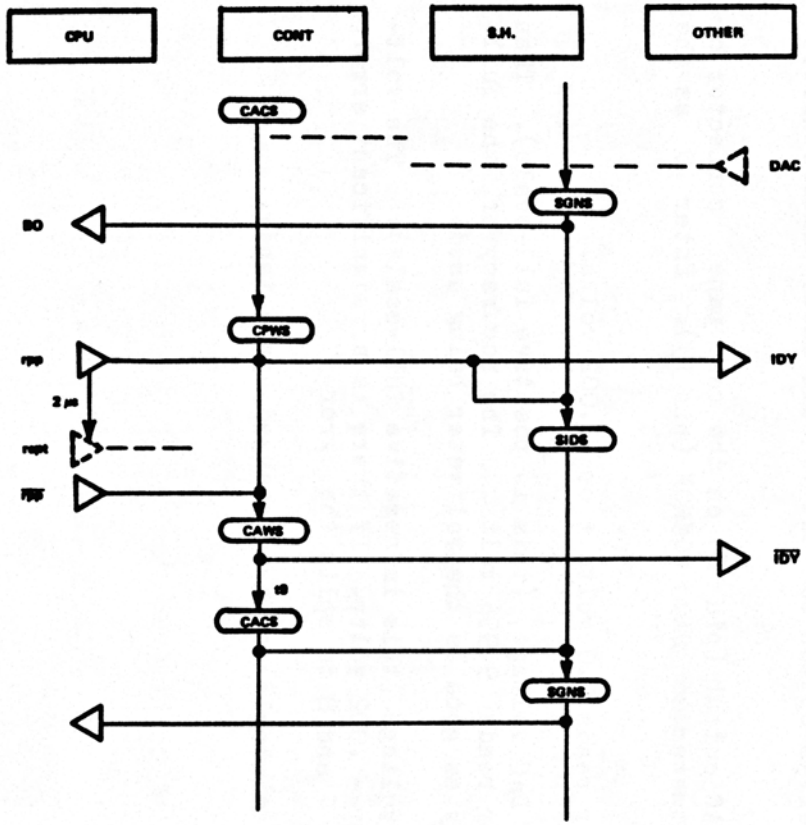


FIGURE 5-4 – CONTROLLER PARALLEL POLLING

6. CALIBRATION PROCEDURES

From the factory, the analog section of the MultI-O board comes calibrated to + or - .01% for full scale range and + or - .002 volts for zero. Whenever any component in the analog section is replaced, particularly op-amps, the calibration should be checked. It is also a good idea to check the calibration yearly when the application demands high absolute accuracy. Relative accuracy (i.e., linearity and repeatibility), is substantially independent of component characteristics and time. It is also non-adjustable.

The following steps will lead you through a complete recalibration from scratch (this is the procedure used at the MTU factory). If you are just checking the calibration or recalibrating after a component change, some of these steps will not be applicable. In this case, read all of the steps first and then only perform the ones that apply in your particular case.

6.1 CALIBRATING THE D-TO-A CONVERTER

1. Remove U8 which is the D-to-A converter chip and then insert a jumper between pins 1 and 2 of the socket (If a DIP clip is available, it is safe to short these pins with U8 in the socket).

2. Connect a voltmeter or scope to pin 6 of U47 or pin 18 of U8. This point will usually be approximately either +14 or -14 volts.

3. Adjust R44 until the meter or scope connected in step 2 jumps to the opposite polarity. Try to set the pot as close to the transition point as possible. If you succeed in getting the voltage to hover between -14 and +14 volts, it is normal for it to fluctuate wildly and eventually drift to + or - 14 volts. This step nulls the offset voltage of U47 which is necessary for optimum differential linearity of the D-to-A converter.

4. Re-install U8, turn on the system, and run the DVS program as described in section 2.2.2. Select the DIGITAL option so that you can enter numbers directly into the DAC.

5. Connect a voltmeter to the DAC output (pin 8) on the rear panel connector or U48 pin 6. Ground the voltmeter to DAC COMMON (pin 15). Enter 0 as the desired DAC voltage.

6. Adjust R45 until the voltmeter reads 0.0 volts + or - .002 volts.

7. Enter 9.996 as the desired DAC voltage (this is positive full-scale). Then adjust R53 until the voltmeter reads 9.996 volts. The accuracy of the full-scale calibration will be only as good as the voltmeter being used.

8. Enter -10 as the desired DAC voltage (this is negative full-scale). The voltmeter should read -10.000 + or - .002 volts. If there is a significant error, you may want to iterate steps 7 and 8 to split the error.

9. Recheck the zero adjustment, it should not have changed appreciably.

6.2 CALIBRATING THE A-TO-D CONVERTER

1. Remove U15 which is the D-to-A converter chip and then insert a jumper between pins 1 and 2 of the socket (If a DIP clip is available, it is safe to short these pins with U15 in the socket).

2. Connect a voltmeter or scope to pin 6 of U45 or pin 18 of U15. This point will usually be either +14 or -14 volts.

3. Adjust R43 until the meter or scope connected in step 2 jumps to the opposite polarity. Try to set the pot as close to the transition point as possible. This step nulls the offset voltage of U45 which is necessary for optimum differential linearity of the A-to-D converter.

4. Re-install U15, turn on the system, and run the DVM program as described in section 2.2.

5. Ground the channel 1 ADC input on the rear panel connector (pin 2) to ADC COMMON 1 (pin 1). During the next step, ignore the readout from the other channels.

6. Adjust R38 until channel 1 reads 0.0. Note that the reading will change in steps of .004 or .005 volts and will probably change slightly from reading to reading.

7. Try grounding each of the other 7 channels. Each one should give a reading of zero + or - .005 volts.

8. Connect an accurate voltage source of +9.996 volts to channel 1. The negative lead of the source should go to ADC COMMON. If a precision voltage source is not available, use the DVS program to set the DAC to 9.996 volts, connect the DAC output to the channel 1 ADC input, and then re-run DVM.

9. Adjust R54 until the channel 1 display just changes from +9.991 to +9.996 volts or alternates between these two values.

10. Use a source of -10 volts and check that channel 1 reads -10.000. If there is a significant difference between + and - full scale, you will have to iterate steps 9 and 10 to split the error.

11. Verify that the other 7 channels produce substantially the same readings as channel 1 did in steps 8 and 10.

12. Ground channel 1 and recheck the zero adjustment, it should not have changed appreciably.

6.3 CALIBRATING THE CLOCK/CALENDAR

1. Calibration of the clock/calendar requires the use of a frequency counter with a period measurement mode and a rated accuracy of .001% (10PPM) to obtain 30 second per month accuracy.

2. With the system power on and waiting to load CODOS from disk (i.e., no program running), connect the frequency counter probe to U20 pin 12. This is a pulse with a 1 second period.

3. Adjust C13 for a period reading of 1000000uS.

7. PRINCIPLES OF OPERATION

This section gives a complete circuit level description of the MultI-0 hardware. Understanding of this material is not required to successfully program or connect the MultI-0 although it may be useful in resolving detail operation questions that may arise. The description is intended to be sufficiently detailed so that an engineer or experienced technician can readily understand the MultI-0's operation for maintenance purposes.

Reference should also be made to sections 8-12 while studying this section. In particular the discussion is keyed to the schematic diagrams in section 12. The four schematic pages will be referred to by "schematic page numbers" which can be found in the lower right corner of the schematic pages. Conventional logic symbols are used in all cases. Signal names are referred to in all caps and inverse signals (active-low) are designated with overbars. Input signals generally enter a page from the left and output signals leave from the right. The physical schematic location of the source for input signals and all destinations of output signals is designated with 3 characters next to the signal name. The first character is the schematic page number, the second character gives the vertical location on the page, and the last character gives the horizontal location. Such "zone numbers" are also used to identify the location of circuitry geing discussed. J1 is the board edge fingers that plug into the MTU-130 bus. J2 - J5 connect to cables which terminate in the off-board connector box. Pin assignments for all connectors is given in section 9.

7.1 BUS BUFFERS

Buffer circuitry between the MTU-130 bus and the internal on-board bus is shown in the left half of schematic page 1. U2 in zone D4, which is a 74LS245 8 bit transceiver, buffers the data bus in both directions. It is conditioned to drive out (board-to-bus) during read cycles and drive in (bus-to-board) during write cycles by a buffered version of the MTU-130's READ/$\overline{\text{WRITE}}$ bus signal. However the chip is actually enabled to drive only when something on the board has actually been addressed. Gate U16-6 also insures that such driving only occurs during Phase 2 when data is or needs to be valid.

Four of the address bus lines (A0 - A3), PH2, and R/$\overline{\text{W}}$ are buffered by U3 in zone C5. These signals are unidirectional (driven only by the MTU-130's Monomeg processor) and always valid so the non-inverting 74LS367 hex buffer used is always enabled. The $\overline{\text{BUS RESET}}$ line is buffered by a simple inverter in zone B6 of schematic page 2 and is distributed throughout the board in both polarities. Since the other address and control lines only go to one destination on the board, they do not require buffering.

7.2 ADDRESS DECODING

Because of the large number of addressable chips and registers on the MultI-0 board and the desire to avoid "wasting" addresses, the actual address decoding logic which is in the left half of schematic page 1, is fairly involved. The address decoder must actually perform two functions; the generation of enable signals for the various I/O chips and registers, and a "board addressed" signal for the data bus buffer. Logically the latter is simply the OR of the former.

As can be seen in the register address map in section 5.1, the MultI-0 board occupies exactly 52 addresses between $BF80 and $BFB3. For decoding purposes these are broken down into 3 groups of 16 and a group of 4. One of the 16 groups is in turn divided into 4 groups of 4. In the following paragraphs, each group will be described in detail.

The first level of decoding takes place in U9-1 which is in zone A5. This one-of-4 decoder is enabled for all 64 addresses between $BF80 and $BFBF by nand gate U1-6 which detects addresses in this range by looking at the AB6-AB8 and I/O ENAB bus signals. Each of the decoder's 4 outputs corresponds to a block of 16 addresses. Two of these, U9-4 and 5, corresponding to $BF80-$BF8F and $BF90-$BF9F, are used directly by the two 6522s on-board each of which has 16 addressable registers. The third output, U9-6, corresponding to $BFA0-$BFAF, is passed to another one-of-4 decoder, U9-15, where it is split into 4 groups of 4. The last output, U9-7, corresponding to $BFB0-$BFBF, is further gated against AB2 and AB3 by U16-12 so that only the first 4 addresses, $BFB0-$BFB3, are actually used.

One-of-4 decoder U9-15 splits up addresses $BFA0-$BFAF into 4 groups of 4 each. Nand gate U16-8 enables the decoder only during Phase 2 as required by the I/O chips it drives which are not 6502/6800 family members (they were intended for 8080/Z-80 systems). The first two groups, $BFA0-$BFA3 and $BFA4-$BFA7, are used directly by the two serial chips which require 4 addresses each. The second two groups, $BFA8-$BFAB and $BFAC-$BFAF, are ored together in U35-3 to form a single group of 8 addresses for use by the 9914A IEEE-488 interface chip.

One-of-8 decoder U4 splits the remaining 4 addresses, $BFB0-$BFB3, into 4 single addresses with separate outputs for read cycles and write cycles. These enable or strobe the various individual TTL registers associated with the A-to-D converter, D-to-A converter, clock/calendar, and serial chip interrupt control. This decoder is enabled during the last 400NS portion of Phase 2 through active-high E1 (pin 6) and active-low E2 (pin 5). E2 is the inverse of Phase 2 which is applied to E1 except that it is delayed 100NS by R62 and C68. Thus the logical-AND function of these enable inputs delays enabling U4 by 100NS at the beginning of Phase 2 but disables it immediately at the end of Phase 2. This leading edge delay is necessary to insure that the data bus is stable before the devices driven by U4 are enabled.

Gate U1-8 is used to logically OR the 52 addresses detected by the address decoder into a single "board addressed" signal that can be used to enable the data bus buffer. This is accomplished with just 4 inputs by combining 3 of the first level groups of 16 with the last group of 4 enable signal used by U4.

7.3 USER 6522

The User 6522 is shown in zone D5 of schematic page 3 and is completely dedicated to the user parallel ports. All 20 I/O pins, two 8 bit bidirectional ports and 2 sets of 2 handshake lines connect directly to the parallel port connector, J5. Additional lines on J5 connect to internal power supply voltages for use by external circuitry. The 6522 receives directly 4 address lines from the bus buffer to address its 16 internal registers as well as buffered PH2, R/$\overline{W}$, and $\overline{RESET}$ signals. The active-low chip select is driven for addresses $BF80-$BF8F by the address decoder described in section 7.2. The active-high chip select is tied to +5.

7.4 INTERNAL 6522

The internal 6522 in zone B5 of schematic page 5 is used internally to control the clock/calendar and the A-to-D converter. Port A is used by the clock while Port B is used by the A-to-D converter. Detaile of the port usage may be found in sections 7.7 and 7.9-12 respectively. Four signals from Port B however (PB6, PB7, CB1, and CB2) connect to J3 for external use independent of the user parallel port. The 6522 itself receives directly 4 address lines from the bus buffer to address its 16 internal registers as well as buffered PH2, R/$\overline{W}$, and $\overline{RESET}$ signals. The active-low chip select is driven for addresses $BF90-$BF9F by the address decoder described in section 7.2. The active-high chip select is tied to +5.

The two serial ports are shown in the left half of schematic page 2. These consist of two type 2651 Universal Synchronous/Asynchronous Receiver/Transmitter serial interface chips (USARTs). The 2651 was chosen for its synchronous capability without sacrificing a programmable on-chip baud rate generator. The 8 data lines, two address lines, R/W line, and RESET line are connected directly to the corresponding internal buffered signals. The single CE signal is connected to the address decoder. Since there is no Phase 2 input (the chips were designed for 8080/Z-80 systems), the CE signal must be anded with Phase 2 externally which is accomplished in the address decoder.

To drive the on-chip baud rate generator, an accurate source of 5.0688MHz is needed. Since this bears no simple relationship to the 1MHz Phase 2 available from the bus, a separate crystal oscillator built from inverters U19-8 and U19-6 is used. The crystal is a standard AT series resonant type. The oscillator output drives the BRCLK input of both 2651s.

Serial port signals normally use EIA logic levels which are approximately -12 volts for a logic one and +12 volts for a logic zero. Logic level shifters are therefore necessary to convert from these levels to the normal 0 and +3 TTL logic levels used by the 2651. For converting from EIA to TTL levels, type 1489 ICs are used in positions U37 and U38. Pullup resistors to +12 are used on some modem control inputs to make them appear active when not driven. For converting from TTL to EIA, a quad operational amplifier (type TL084) is used. This is preferable to the usual 1488 because of its tightly controlled output slew rate (13V/uS), current limiting, and lower power consumption. To function as level shifters, the op-amps are used as comparators. Resistors R9 and R11 establish a +2.5 volt level on the non-inverting inputs and the TTL logic signal is applied to the inverting inputs. Thus a high logic level (approx +3 volts) will cause the amplifier's output to swing to about -11 volts and a low level will produce about +11 volts. Resistors (330 ohms) are placed in series with the outputs to satisfy the EIA source impedance specs.

The top 2651, U29, is connected so that all of its modem control signals are available for use. The bottom 2651 however has only serial data out and serial data in implemented in order to conserve board and connector real estate. According to how the 2651 is programmed, the TXC (transmit clock) and RXC (receive clock) pins may be either inputs or outputs. When using the internal baud rate generator, they are outputs and when using an external clock they are inputs. Normally the internal clocks are used but for synchronous applications, the modem must supply the transmit and receive clocks. Series resistors R10 and R12 allow these pins to be driven by U38-11 and U37-8 when needed without shorting them out when the internal baud rate generator is selected.

One feature the 2651 lacks however is on-chip control of interrupts. The circuitry in the bottom left portion of the page performs this function. Each 2651 has 3 active-low open-drain interrupt outputs that cannot be disabled. Pullup resistors RP1 and inverters U25 convert these to 6 active-high interrupt requests. Hex register U17 is the serial I/O interrupt enable register and each bit corresponds to one of the 6 requests. Nand gates U18 and U24 gate each interrupt request with the corresponding enable and then through a wire-or connection and two inversions connect to the IRQ bus line. Thus if any enabled interrupt is requesting, the IRQ line will be pulled down to request a 6502 IRQ interrupt. Octal buffer U10 provides a readback of the state of the interrupt enable register (so that individual bits can be programmed) and a bit that indicates if any of the 6 serial interrupts are active. Registers in the 2651s must then be polled to determine which of the 6 is responsible. The interrupt enable register is connected to BUS RESET so that all serial interrupts are initially disabled.

7.6 IEEE PORT

The IEEE port circuitry is in the right half of schematic page 2. A TMS9914A IEEE-488 controller chip in conjunction with specialized buffers U33 (a 75160) and U34 (a 75161A) perform all of the electrical interfacing necessary to support the IEEE-488 bus as a talker, listener, or controller. The 8 data lines, 3 address lines, and RES line of the TMS9914A are connected directly to the corresponding buffered MTU-130 bus lines. The CE signal is connected to the address decoder described in section 7.2. Since the 9914A was designed for 8080 bus protocol, the DBIN and WE signals require additional gating. DBIN is really a Read Enable input which is driven high during Phase 2 by U35-6 whenever the MTU-130 bus executes a read cycle. WE is driven low during Phase 2 by U20-3 whenever the MTU-130 bus executes a write cycle. The 9914A's CLK input is not used for chip access control but does drive the internal sequential circuitry. The 1MHz Phase 2 signal that is readily available is used to drive this input.

7.7 CLOCK/CALENDAR

The clock/calendar circuitry is in the right half of schematic page 3. An MSM5832 clock chip which is free of bugs and has been available for several years is used. This very low power CMOS IC is too slow to respond to bus signals directly therefore it is driven from one of the internal parallel ports.

Functionally the MSM5832 is organized as 13 registers of 4 bits each. Each register holds a BCD digit of the date and time. By reading and writing these registers, the date and time may be determined or set to the second. Four address lines are used to address the 13 registers and these are connected to the lower 4 bits of Port A of the internal 6522. The 4 data lines are connected to the upper 4 bits of Port A. These 4 bits must be set up as outputs when it is desired to write into the clock, otherwise they must be set up as inputs.

Because the clock must not be disturbed by power up/down cycles or typical software crashes, the circuitry for enabling, reading, and writing the 5832 is somewhat more complex than normal. The CS input is tied to on-board +5 power through a silicon diode so that the chip is enabled for access only when full +5 power is available. This input has an internal "pull-down" resistor connected to ground to assure a solid zero level when power is off. The READ signal is simply driven by the CA2 output of the internal 6522 since spurious reading is not harmful.

The two sensitive inputs are HOLD and WRITE. The HOLD input will stop the clock as long as it is held high for accurate reading or setting of the registers. If the hold duration is less than a second, no time is actually lost. Gate U26-6 and the single-shot formed from U12-2 and U12-4 insures that the hold duration is suitably short regardless of unintended program action. U26-6 locks out hold operation while $\overline{\text{LOC RESET}}$ is active and requires a simultaneous $\overline{\text{CLOCK OP}}$, data bus bit 4=1, and data bus bit 5=0 to trigger the single-shot. The CMOS gates form a single-shot that cannot be triggered by the application of loss of power. Once the single-shot is triggered, its pulse width is 500uS which is sufficient time to read all 13 registers. After timing out, at least 1000uS must be allowed before it can be triggered again.

The WRITE input is also driven by a single-shot that cannot be triggered by power fluctuations. It can be triggered only when gate U26-8 detects a simultaneous $\overline{\text{CLOCK OP}}$, data bus bit 5=1, no $\overline{\text{LOC RESET}}$, and the HOLD single-shot has been triggered. These conditions are unlikely to be met by accident.

The timebase for the clock is a 32.768KHz crystal tuned by fixed capacitor C12 and variable capabitor C13. This crystal is a sub-miniature watch type that is smaller than a 1/4 watt resistor. The MSM5832 has an on-chip oscillator that has been carefully temperature and voltage compensated for accurate timekeeping. If frequency readjustment is necessary, neither of the crystal pins should be probed because the extra capacitance will affect the frequency slightly. For tuning, the clock should be programmed for idle mode (read mode and digit address=$F) and a frequency counter with a period mode should be connected to pin 12 which supplies pulses at 1 second intervals.

Backup power is supplied from an off-board 3 volt battery through J3-20. Diodes D6 and D7 gate power from either the battery or the on-board power supply to the clock depending on which has the higher voltage. Germanium diodes are used to minimize voltage drops which are around 100mV at the low currents needed by the clock. The clock's keep-alive voltage is 2.3 volts and its low voltage current consumption is less than 30uA which allows the two AA size cells to last for their shelf life.

When idle, the clock provides pulses every millisecond, second, minute, and hour on its data lines. Two of these, the second and hour pulses, are gated through U20-6, 8, and 11 into the CA1 input of the internal 6522. This CA1 input can in turn be programmed to generate an interrupt when the selected line pulses. Port B bits 4 and 5 are used to select neither, either, or both of these pulses for application to CA1.

7.8 DIGITAL-TO-ANALOG CONVERTER

The digital-to-analog converter is in the upper right corner of schematic page 4. It consists of a 12 bit holding register, 12 bit current output D-to-A converter chip, and output amplifiers. The holding register has an 8 bit portion (U7, 74LS374) which holds the most significant 8 DAC bits, and a 4 bit portion (U6, 74LS174) that holds the low-order 4 bits. The data inputs to these registers come from the buffered MTU-130 data bus. The clock inputs to these registers are connected to the DAC HI WRITE ($BFB3) and DAC LO WRITE ($BFB2) outputs of the address decoder. Since the 74LS374 does not have a clear input, the 74LS174's clear input is tied up. Thus resetting the system has no effect on the register contents. The low 2 bits of U6 latch data from bits 2 and 3 of the bus but are not used for anything.

The D-to-A converter proper is U8 which is an AD7541 or equivalent 12 bit unit with 12 bit linearity guaranteed. The accuracy is determined by external components but can never be better than the linearity. Except for D11, the 12 bit inputs are driven directly from the holding register. D11 is either the true or complement version of register bit 11 for offset binary or twos complement coding respectively as determined by jumper JP4. The normal setting is complement which gives twos complement coding for bipolar (+ and -) output voltages. The reference input, which determines the full-scale accuracy, is nominally +10.000 volts from the reference source described in section 7.13.

Output current from pin 1 of the converter (U8) is applied to amplifier U47 to be converted into a voltage ranging from -10 to 0 volts. A resistor inside U8 from pin 18 to pin 1 provides feedback and is matched with the converter's ladder resistors. C19 is paralleled accross this resistor to compensate the considerable output capacitance of the converter and thus minimize the settling time of the amplifier's output. R44 is used to null the offset voltage of U47 which is required to achieve rated linearity.

Amplifier U48 is used for final preparation of the DAC's output. When jumper JP7 is installed, the +10 volt reference is combined with the 0 to -10 output of U47 to produce a bipolar output from -10 to +10. This can be understood by noting that the feedback resistor, R48, is 10K so that the gain from the reference voltage through R56, 10K, is -1 and the gain from the DAC through R46, 5k, is -2. When jumper JP8 is installed instead, the output of U47 is simply inverted to provide a range of 0 to +10 volts. Note that resistors R46, R48, and R56 are .1% precision units. Pot R45 and resistor R47 are used to adjust the DAC's zero setting and cancel any offset of U48. Full scale is determined by the +10 reference adjustment.

7.9 ANALOG INPUT MULTIPLEXOR

To achieve 8 analog input channels at a reasonable cost, an analog multiplexor is used to select the channels one-at-a-time and present them to the A-to-D converter. This circuitry is in zone A5 of schematic page 4 and zone A4 of page 3. The multiplexor itself consists of two quad analog switches, U31 and U32. Each switch is a JFET type for reasonable immunity to static discharge and is controlled by its own independent enable. Each of the 8 analog inputs are isolated and protected with a 1K series resistor and diodes to the +12 and -12 power supplies. Any transient voltage greater than +12 or less than -12 volts will be shunted into the power supplies. A severe sustained overload (such as connection to 115vac) will fuse the 1K resistor.

The 8 switches are controlled by dual 1-of-4 decoder U22 which is driven by Port B bits 1-3 of the internal 6522. When JP5, JP11, and JP13 are installed, the two decoders act as a single 1-of-8 decoder which will cause only one of the 8 analog switches to be closed and pass its signal to the non-inverting input of the differential amplifier. If JP4 and JP12 are installed instead, both decoders act in parallel and close <u>two</u> switches, one on each input of the differential amplifier.

7.10 DIFFERENTIAL AMPLIFIER

Because the least significant bit of a 12 bit analog input system is as little as 2.5 millivolts, the effects of ground noise must be controlled for the 12 bit resolution to be meaningful. This is normally accomplished with a differential amplifier, also known as an instrumentation amplifier. U41, U42, and U43 are connected into a classical unity gain differential amplifier. U41 and U42 are actually just unity gain buffers which gives the overall amplifier a very high input impedance. U41-3 is the non-inverting input and U42-3 is the inverting input. R32-R35 are precision matched resistors which are necessary for good common-mode rejection of the overall amplifier. Normally, R29 and R31 are not significant in the operation of the amplifier because R28 is not installed. However if R28 is installed, the amplifier's gain will increase above unity. For example, if R28 is 10K, the gain will be 2. Small values for R28 can be used for substantial input gain and therefore a smaller input range for tha A-to-D converter. See section 3.4 for additional discussion about increasing the differential amplifier gain.

7.11 SAMPLE-AND-HOLD

Since the A-to-D converter used is a successive approximation type, a sample-and-hold circuit must be used to hold the ADC input steady while a conversion is taking place. An LF398 (U44) is used for this purpose. Its sample/hold mode is controlled by Port B bit 0 of the internal 6522. Hold capacitor, C20, is a polystyrene type to minimize the effects of dielectric absorption. Its 4700pF value is a balance between fast acquisition time and freedom from hold step and droop errors.

The analog-to-digital converter proper is in the upper left corner of schematic page 4. It consists of a 12 bit D-to-A converter, amplifiers, a comparator, a 12 bit successive approximation register, and a set of 12 tri-state gates. This arrangement was found to be much more cost-effective than a 12 bit ADC module.

U5 in zone D6 is the successive approximation register. This CMOS device is adequately fast for the application and requires very little operating power. A clock frequency of 250KHz is used which is derived from the 1MHz Phase 2 signal by a divide-by-4 circuit. This frequency gives a conversion speed of 4uS per bit for a total of 48uS for the entire 12 bit conversion. The START signal is driven by the same source (Port B bit 0) as the sample-and-hold is and is phased such that the conversion starts immediately when the S&H enters its hold mode. Note that the conversion does not start until the next negative edge of the clock after the START signal. There is therefore a delay of 0 to 3uS before conversion starts bringing the worst case conversion time up to 51uS. There is no end-of-conversion output; the using program simply waits at least 51uS for the conversion to complete.

U13 and U14 are wired up as an 8 bit buffer and a 4 bit buffer to gate the successive approximation register's output to the on-board data bus. These are enabled by outputs from the address decoder. Note that it is permissable to read the upper 8 bits 35uS after the conversion is initiated since they have settled by that time. Thus if the upper 8 bits are read first, greater throughput than implied by the 51uS conversion time is possible.

The D-to-A converter is U15 and is driven by the digital output of the successive approximation register. Amplifier U45 converts the DAC's output current into a voltage ranging from 0 to -10 volts. Like the D-to-A converter described in section 7.9, R43 is used to null the offset of U45 as required to achieve U15's rated linearity. C18 damps the amplifier's ringing which is much more critical here since the output voltage must settle to .01% is less than 4uS.

U46 is the comparator in the A-to-D converter. Its job is to compare the DAC's output with the analog input from the sample-and-hold and generate a less-than or greater-than input to the successive approximation register. An LM311 is used which provides the necessary 2mV sensitivity along with a 200NS response time. The DAC's 0 to -10 volt output is summed with the analog input and +10 volt reference through R39, R40, and R42 and appears on the non-inverting input of the comparator. R37 and R38 inject a small amount of current into the summing junction to cancel offsets in the analog input system. Since the inverting input is tied to ground, the comparator reports whether the DAC's output is less than or greater than the analog input. If jumper JP-9 is installed, the reference voltage is also summed in resulting in a -10 to +10 volt range of analog inputs. If JP10 is installed instead, the reference voltage is not included and the analog input sensitivity is doubled (R40 now parallels R39) thus giving a 0 to +10 range. Diodes D27 and D28 prevent the comparator from becoming overly saturated and exhibiting sluggish response. Since the comparator has an open-collector output, R35 serves as a pullup resistor to +5 volts.

7.13 VOLTAGE REFERENCE

The short and long term accuracy of the A-to-D and D-to-A converters is determined primarily by the +10 volt reference voltage supplied to them. Temperature stability is particularly important since the MultI-O board is mounted in a confined area near other heat generating boards and thus experiences a significant temperature excursion while the system is warming up. To achieve the necessary stability, an LM3999 temperature regulated zener diode, Q6, is used in the reference circuit in the lower right corner of schematic page 4. This unremarkable looking IC (it looks like just another plastic transistor) actually contains an oven, a heater, a temperature regulator, and a "synthesized" zener diode inside. The heater and regulator keeps the zener at at a constant temperature regardless of its environment and the synthesized zener exhibits a sharper breakdown knee and better stability than conventional zener diodes. Connection is simple: +12 volts is applied to the regulator/heater and a series resistor biases the zener diode. On the MultI-O board, a styrofoam bead or plastic jacket covers U6 to minimize heat loss and therefore minimize power consumption by the regulator and heater.

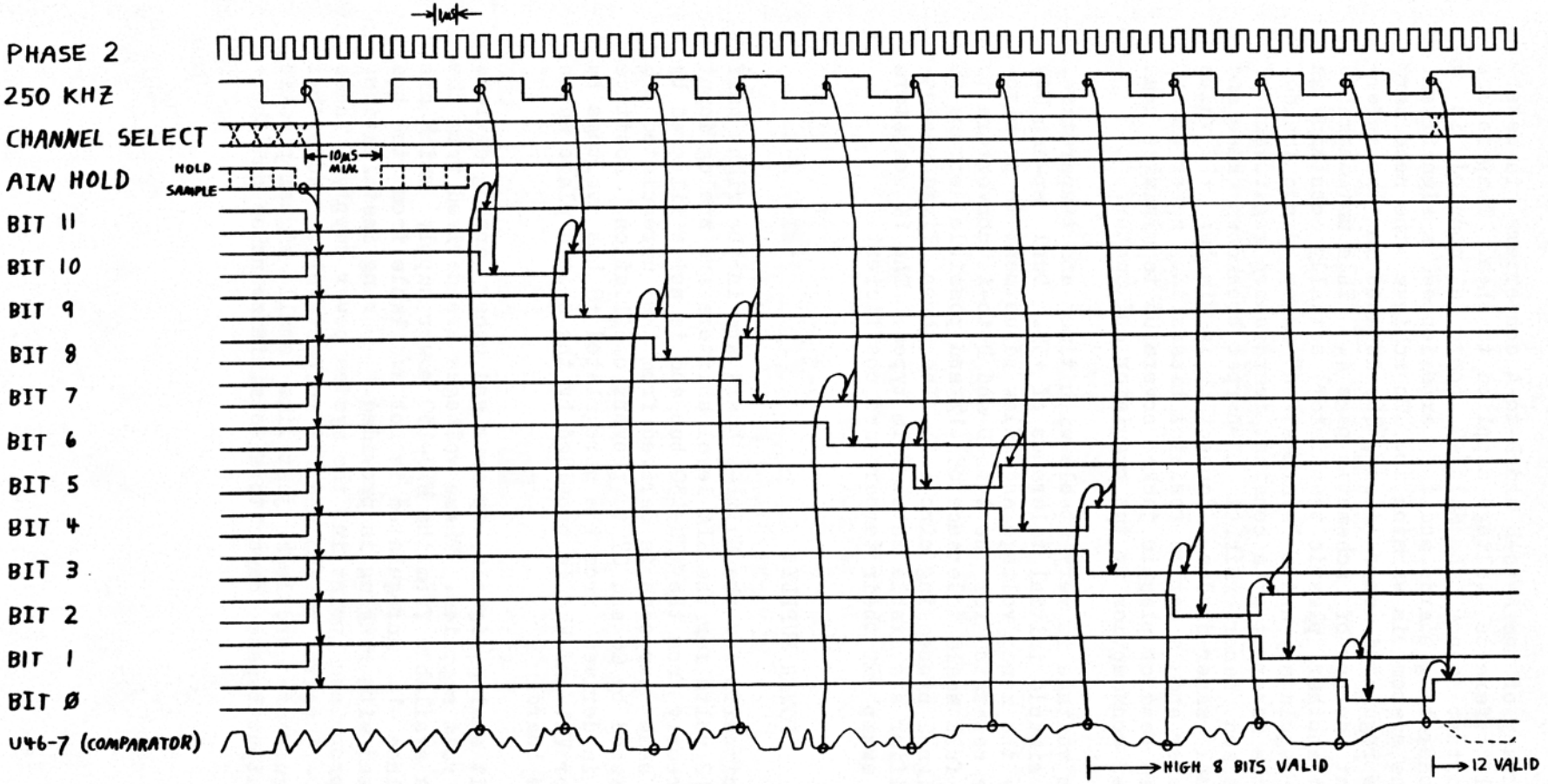
Although the diode breakdown voltage is very stable with time and temperature, it is only 6.95 volts and has a sizable initial tolerance of ±5%. Dual op-amp U49 and controls R49 and R50 amplify the zener voltage and allow adjustment to precisely +10 volts. The adjustment is separate for the A-to-D and D-to-A converters to allow compensating the AD8541's full scale tolerance of .1% and possible errors in the .1% resistors used in the signal processing circuits. The three fixed resistors used in each reference amplifier are stable metal film types. The 1K resistors accross the 25K adjustment pots swamp out their temperature coefficient.

7.14 POWER SUPPLY

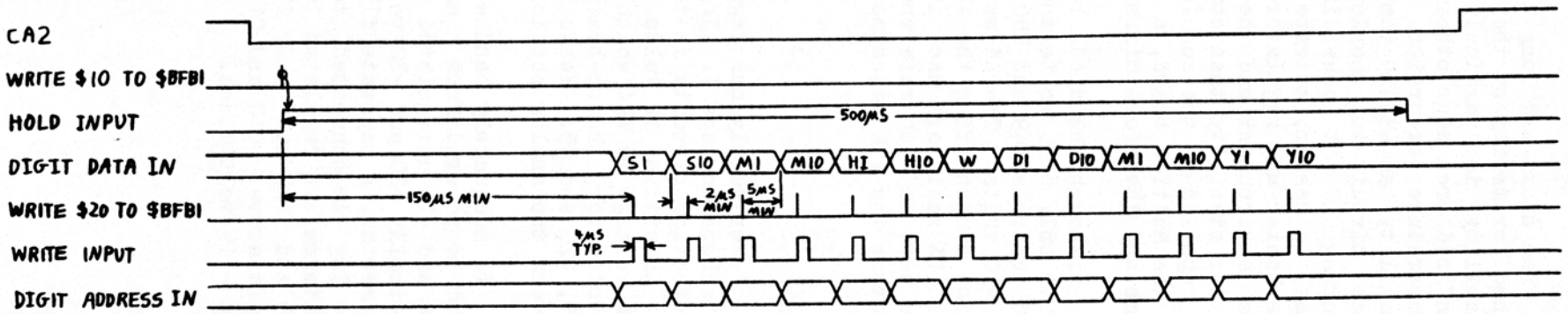
Power supply and regulator circuits for the MultI-O board are in the right half of schematic page 1. Negative 12 volts for the EIA level shifters and analog input protection circuits is taken directly from the MTU-130 bus and is noise filtered by C1. Positive 12 volts for the same purposes is derived from the unregulated +16 bus voltage by VR4 which is bypassed by C6 and C7 to prevent oscillation. Positive 5 volts for the on-board logic is derived from the unregulated +8 bus voltage by VR1, C2, and C3. Heatsinking for VR1 and VR4 is provided by the shield plate which covers the analog portion of the board.

To adequately handle ±10 volt analog signals, op-amps and other analog devices must be powered from + and - 15 volt supplies. These voltages are derived from the +16 and -16 unregulated voltages available from the MTU-130 power supply which are typically +20 and -20 volts. Since -16 unregulated is not available from the bus connector, a separate flying lead with ringlug is provided. During installation, the lug is connected to an appropriate point in the system power supply. These voltages are regulated to + and -15 volts by VR3 and VR2 respectively. Note that VR3 and VR2 are only 100MA regulators and look just like small signal plastic transistors. C10 and C11 are output bypass capacitors to minimize noise on the + and - 15 volt lines.

ANALOG-TO-DIGITAL CONVERTER



SETTING THE CLOCK/CALENDAR



9. CONNECTOR PIN CONNECTIONS

9.1 REAR PANEL CONNECTORS

9.1.1 Parallel Ports

PIN	SIGNAL	PIN	SIGNAL
1	+5 VOLTS (100MA)	19	USER GROUND
2	N.C.	20	USER +12 VOLTS (100MA)
3	USER RESET	21	USER -12 VOLTS (50MA)
4	CA2	22	CA1
5	PA1	23	PA0
6	PA3	24	PA2
7	PA5	25	PA4
8	PA7	26	PA6
9	PB1	27	PB0
10	PB3	28	PB2
11	PB5	29	PB4
12	PB7	30	PB6
13	CB2	31	CB1
14	N.C.	32	N.C.
15	N.C.	33	N.C.
16	N.C.	34	N.C.
17	N.C.	35	N.C.
18	N.C.	36	N.C.

9.1.2 Serial Ports

PORT 0 (LOWER)

PIN	SIGNAL	PIN	SIGNAL
1	FRAME GND	14	N.C.
2	TRANSMIT DATA	15	EXT TRANSMIT CLK
3	RECEIVE DATA	16	N.C.
4	REQ. TO SEND	17	EXT RECEIVE CLK
5	CLEAR TO SEND	18	N.C.
6	DATA SET RDY	19	N.C.
7	SIGNAL GND	20	DATA TERM RDY
8	CARRIER DET	21	N.C.
9	N.C.	22	N.C.
10	N.C.	23	N.C.
11	N.C.	24	N.C.
12	N.C.	25	N.C.
13	N.C.		

PORT 1 (UPPER)

PIN	SIGNAL	PIN	SIGNAL
1	FRAME GND	14	N.C.
2	TRANSMIT DATA	15	N.C.
3	RECEIVE DATA	16	N.C.
4	N.C.	17	N.C.
5	N.C.	18	N.C.
6	N.C.	19	N.C.
7	SIGNAL GND	20	N.C.
8	N.C.	21	N.C.
9	N.C.	22	N.C.
10	N.C.	23	N.C.
11	N.C.	24	N.C.
12	N.C.	25	N.C.
13	N.C.		

9.1.3 Analog I/O

PIN	SIGNAL	PIN	SIGNAL
1	ADC COMMON 1	9	ADC CH. 5
2	ADC CH. 1	10	ADC CH. 6
3	ADC CH. 2	11	ADC CH. 7
4	ADC CH. 3	12	ADC CH. 8
5	ADC CH. 4	13	ADC COMMON 2
6	SHIELD (DIG GND)	14	N.C.
7	N.C.	15	DAC COMMON
8	DAC OUT		

9.1.4 Digital I/O

PIN	SIGNAL	PIN	SIGNAL
1	AIO PB6	6	AIO PB7
2	AIO CB2	7	AIO CB1
3	AIO +5 VOLTS (50MA)	8	AIO GROUND
4	N.C.	9	N.C.
5	N.C.		

9.1.5 IEEE Interface

PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL
1	DIO1	7	NRFD	13	DIO5	19	GND 7
2	DIO2	8	NDAC	14	DIO6	20	GND 8
3	DIO3	9	IFC	15	DIO7	21	GND 9
4	DIO4	10	SRQ	16	DIO8	22	GND 10
5	EOI	11	ATN	17	REN	23	GND 11
6	DAV	12	SHIELD	18	GND 6	24	LOGIC GND

9.2 ON-BOARD CONNECTORS

9.2.1 MTU-130 Bus Connector (J1)

PINS	SIGNALS	PINS	SIGNALS
1	I/O SELECT	A	ADDRESS BUS 0
2	--NC--	B	ADDRESS BUS 1
3	--NC--	C	ADDRESS BUS 2
4	IRQ	D	ADDRESS BUS 3
5	--NC--	E	ADDRESS BUS 4
6	--NC--	F	ADDRESS BUS 5
7	RESET	H	ADDRESS BUS 6
8	DATA BUS 7	J	ADDRESS BUS 7
9	DATA BUS 6	K	ADDRESS BUS 8
10	DATA BUS 5	L	--NC--
11	DATA BUS 4	M	--NC--
12	DATA BUS 3	N	--NC--
13	DATA BUS 2	P	--NC--
14	DATA BUS 1	R	--NC--
15	DATA BUS 0	S	--NC--
16	-12 VOLTS REGULATED	T	--NC--
17	--NC--	U	PHASE 2
18	+8 VOLTS UNREGULATED	V	READ/WRITE
19	--NC--	W	--NC--
20	--NC--	X	+16 VOLTS UNREG
21	--NC--	Y	--NC--
22	GROUND	Z	--NC--

9.2.2 IEEE Connector (J2)

PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL
1	--KEY--	8	DIO4	15	GND	22	ATN
2	DIO1	9	DIO8	16	NDAC	23	GND
3	DIO5	10	EOI	17	GND	24	GND
4	DIO2	11	REN	18	IFC	25	GND
5	DIO6	12	DAV	19	GND	26	--KEY--
6	DIO3	13	GND	20	SRQ		
7	DIO7	14	NRFD	21	GND		

9.2.3 Miscellaneous Connector (J3)

PIN	SIGNAL	PIN	SIGNAL
1	SERIAL 0 DATA TERM READY	11	AIO +5 VOLTS
2	SERIAL 0 TRANSMIT DATA	12	SERIAL 0 TRANSMIT CLOCK
3	SERIAL 1 TRANSMIT DATA	13	SERIAL 0 CARRIER DETECT
4	SERIAL 1 GROUND	14	AIO CB2
5	SERIAL 1 RECEIVE DATA	15	AIO CB1
6	SERIAL 0 RECEIVE CLOCK	16	AIO GROUND
7	SERIAL 0 RECEIVE DATA	17	SERIAL 0 GROUND
8	SERIAL 0 DATA SET READY	18	AIO PB6
9	SERIAL 0 REQ TO SEND	19	AIO PB7
10	SERIAL 0 CLR TO SEND	20	CLOCK BATTERY, +3 VOLTS

9.2.4 Analog I/O Connector (J4)

PIN	SIGNAL	PIN	SIGNAL
1	ADC CH 1	8	ADC CH 8
2	ADC CH 2	9	--NC--
3	ADC CH 3	10	--NC--
4	ADC CH 4	11	ADC COMMON
5	ADC CH 5	12	DAC OUTPUT
6	ADC CH 6	13	DAC COMMON
7	ADC CH 7	14	SYSTEM (DIGITAL) GROUND

9.2.5 Parallel I/O Connector (J5)

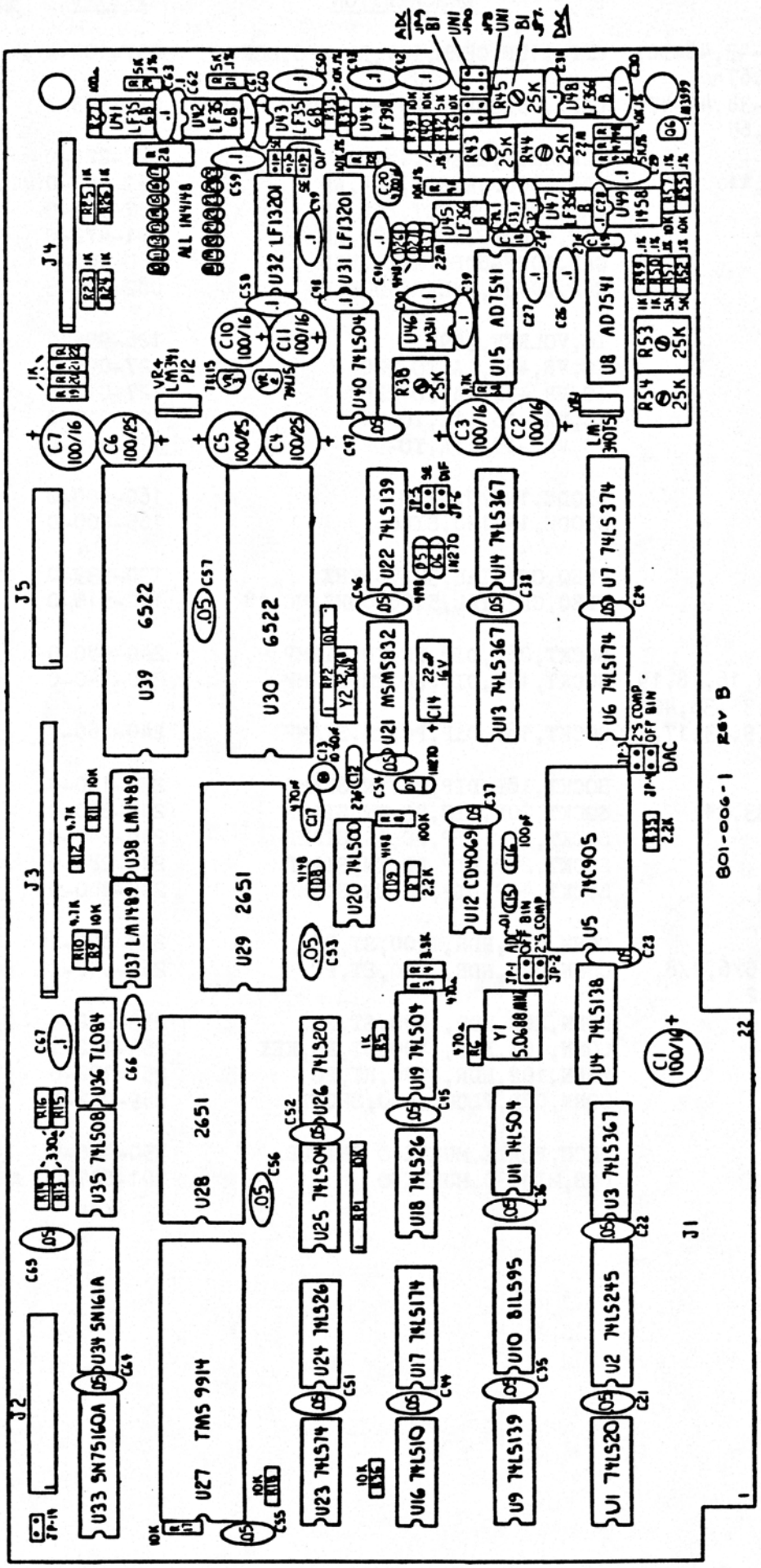
PIN	SIGNAL	PIN	SIGNAL
1	+5	14	PA4
2	GROUND	15	PA7
3	--NC--	16	PA6
4	+12	17	PB1
5	RESET	18	PB0
6	-12	19	PB3
7	CA2	20	PB2
8	CA1	21	PB5
9	PA1	22	PB4
10	PA0	23	PB7
11	PA3	24	PB6
12	PA2	25	CB2
13	PA5	26	CB1

10. PARTS LIST

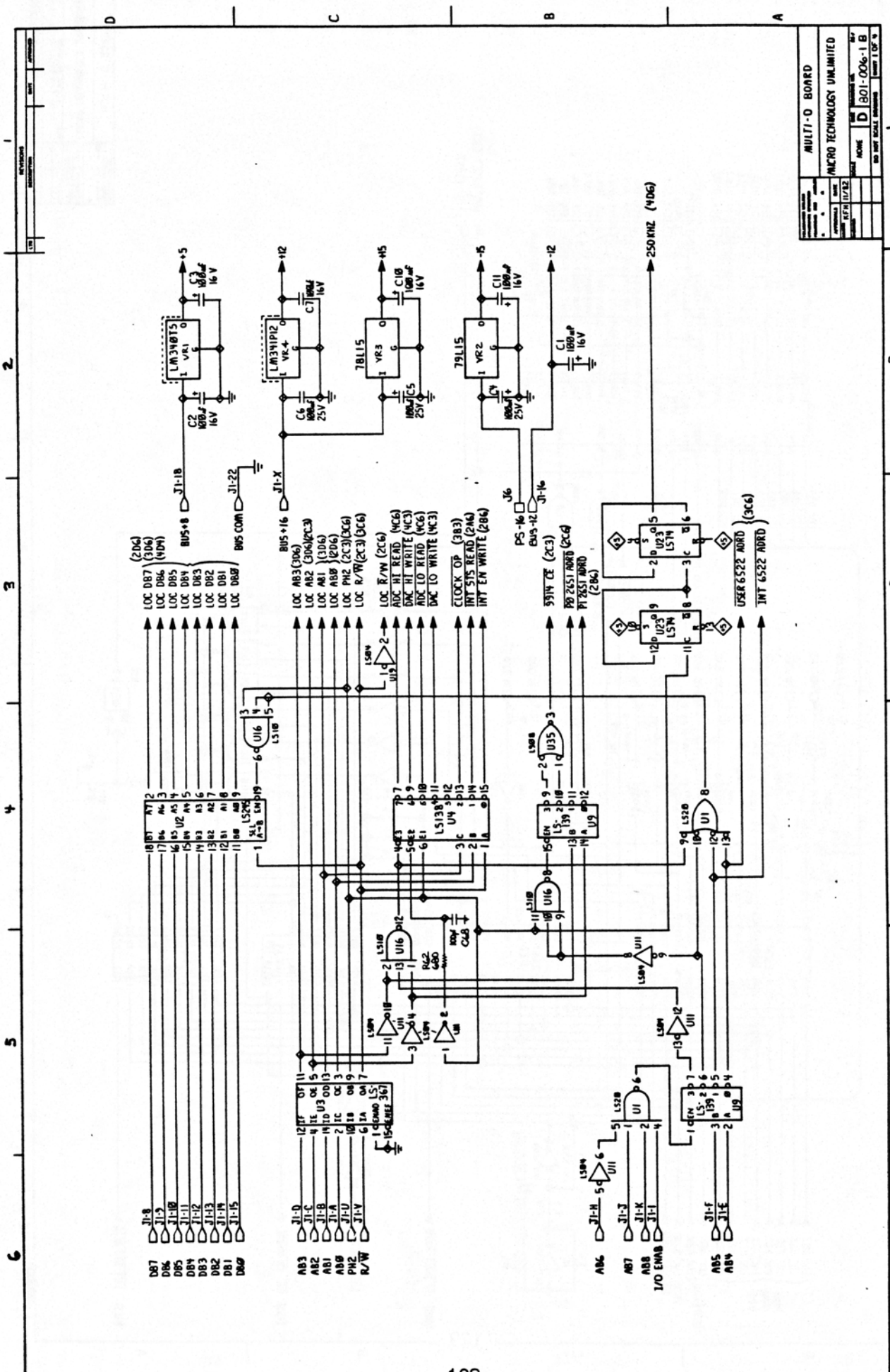
ID NUMBER	DESCRIPTION	PART #	REV	QTY
U5	IC,74C905	103-905-0		1
U20	IC,74LS00	108-000-0		1
U11,19,25,40	IC,74LS04	108-004-0		4
U35	IC,74LS08	108-008-0		1
U16	IC,74LS10	108-010-0		1
U1,26	IC,74LS20	108-020-0		2
U18,24	IC,74LS26	108-026-0		2
U23	IC,74LS74	108-074-0		1
U4	IC,74LS138	108-138-0		1
U9,22	IC,74LS139	108-139-0		2
U6,17	IC,74LS174	108-174-0		2
U2	IC,74LS245	108-245-0		1
U3,13,14	IC,74LS367	108-367-0		3
U7	IC,74LS374	108-374-0		1
U30,39	IC,6522	110-522-0		2
U33	IC,75160A,INTF,IEEE-488	111-160-0		1
U34	IC,75161A,INTF,IEEE-488	111-161-0		1
U37,38	IC,1489,INTF,RCVR,RS232	111-489-0		2
U28,29	IC,2651,MOS,USART	113-651-0		2
U21	IC,5832,MOS,CLOCK/CAL	113-832-0		1
U27	IC,9914,MOS,IEEE-488 CNTLR	113-914-0		1
U10	IC,81LS95	114-095-0		1
U12	IC,4069	115-069-0		1
U31,32	IC,LIN,ANLGSW,13201,4SW,JFET	121-201-0		2
U8,15	IC,LIN,DAC,12-BIT,1218,MULTPLY	122-218-0		2
U36	IC,LIN,OPAMP,084,BIFET,QUAD	123-084-0		1
U41-43,45,47,48	IC,LIN,OPAMP,356B,JFET INPUT	123-356-0		6
U49	IC,LIN,OPAMP,1458,DUAL	123-458-0		1
U44	IC,LIN,SAMHLD,398	124-398-0		1
U46	IC,LIN,VOLCMP,311	125-311-0		1
R27	RES,100,5%,CF,.25W	000-101-0		1
R5,19-26	RES,1K,5%,CF,.25W	000-102-0		9
R9,11,18,36,17,59-61	RES,10K,5%,CF,.25W	000-103-0		8
R8	RES,100K,5%,CF,.25W	000-104-0		1
R7,35	RES,2.2K,5%,CF,.25W	000-222-0		2
R37,47	RES,2.2M,5%,CF,.25W	000-225-0		2
R13-16	RES,330,5%,CF,.25W	000-331-0		4
R4	RES,3.3K,5%,CF,.25W	000-332-0		1
R3,6	RES,470,5%,CF,.25W	000-471-0		2
R10,12,58	RES,4.7K,5%,CF,.25W	000-472-0		3
R62	RES,680,5%,CF,.25W	000-681-0		1
R49,50	RES,1K,1%,MF,.25W	005-102-0		2
R29,31,42,46,51,52	RES,5K,.1%,MF,.25W	009-502-0		6
R32-34,39-41,48, 55-57	RES,10K,.1%,MF,.25W	009-103-0		10
RP1,RP2	RES,NETWK,10K,7RES,S1P,8P1N,5%	011-103-0		2
R38,43-45,53,54	RES,VAR,25K,CERM,SQ,VT ADJ	016-253-0		6
C16,68	CAP,100PF,CERM,12V,RD,NPO	035-101-0		2
C12,18,19	CAP,27PF,CERM,12V,RD,NPO	035-270-0		3
C17	CAP,470PF,CERM,12V,RD,NPO	035-471-0		1

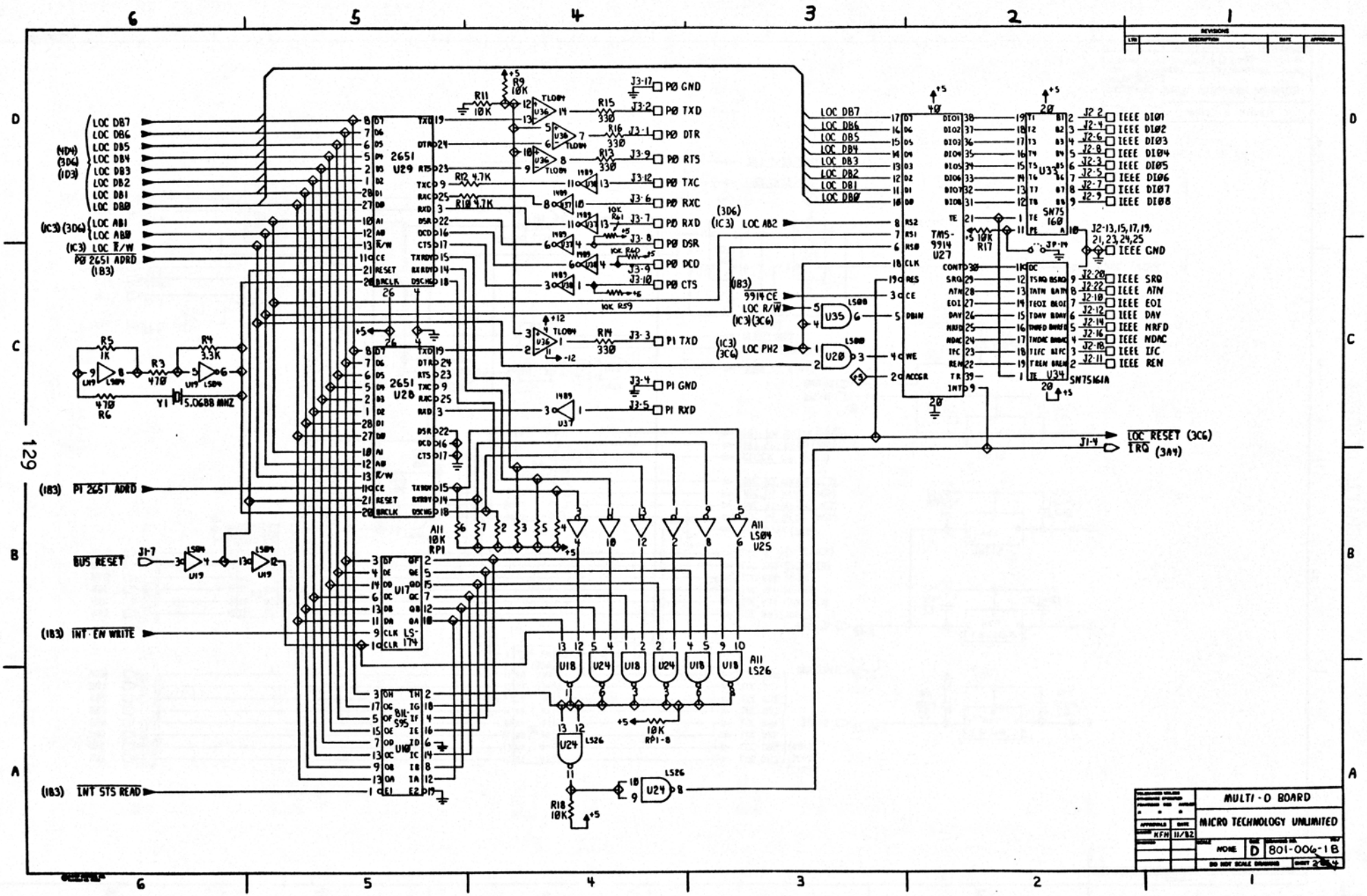
ID NUMBER	DESCRIPTION	PART #	REV	QTY
C26-34,39-43,48-50, 58-63,66,67	CAP,.1UF,CERM,50V,RD,NPO,MONO	037-104-4		25
C21-24,35-38,44-47, 51-57,64,65	CAP,.05,CERM,12V,RD,Z5U	037-503-0		21
C14	CAP,22UF,ELECT,16V,AX	042-226-0		1
C1-3,7,10,11	CAP,100UF,ELECT,16V,RD	051-107-0		6
C4-6	CAP,100UF,ELECT,25V,RD	052-107-0		3
C20	CAP,4700PF,POLY,50V,RD	071-472-0		1
C13	CAP,10PF-50PT,MICA,VAR	080-100-0		1
C15	CAP,.01UF,MYLAR,100V,RD	082-103-5		1
Q6	IC,VOLREF,3999	126-999-0		1
VR1	IC,VR,+5V,1A,TO-220	127-005-3		1
VR4	IC,VR,+12,1A,TO-220	127-012-3		1
VR3	IC,VR,+15,.1A,TO-92	127-015-0		1
VR2	IC,VR,-15,.1A,TO-92	127-115-0		1
D6,7	DIODE.1N270,GERM	160-000-0		2
D5,8-27	DIODE,1N4148,SILC	165-000-0		21
Y2	FREQ,CRYSTAL,32.768KHZ	190-333-0		1
Y1	FREQ,CRYSTAL,5.0688MHZ,HC-18	190-515-0		1
XU41-49	SOCKT,08P,DIP,PC,TN,STAMP	240-080-0		9
XU11,12,14,16,18,19, 20,23-26,35-38,40	SOCKT,14P,DIP,PC,TN,STAMP	240-140-0		16
XU1,3,4,6,9,13,17, 22,31,31	SOCKT,16P,DIP,PC,TN,STAMP	240-160-0		10
XU8,15,21	SOCKT,18P,DIP,PC,TN,STAMP	240-180-0		3
XU2,7,10,33,34	SOCKT,20P,DIP,PC,TN,STAMP	240-200-0		5
XU5	SOCKT,24P,DIP,PC,TN,STAMP	240-240-0		1
XU28,29	SOCKT,28P,DIP,PC,TN,STAMP	240-280-0		2
XU27,30,39	SOCKT,40P,DIP,PC,TN,STAMP	240-400-0		3
JP13,14	CONN,02D,HDR,.100,ST,PC	253-022-1		2
JP1/2,3/4,5/6,7/8, 9/10,11/12	CONN,04D,HDR,.100,ST,PC	253-042-1		6
J2,5	CONN,26D,HDR,.100,ST,PC	253-262-1		2
J3	CONN,10S,HDR,.100,ST,PC,KEY	254-102-1		2
J4	CONN,10S,HDR,.100,RT,PC	254-102-6		2
	CONN,02D,PLUG,.100,SHUNT	259-022-0		9
	MECH,PLATE,MULTI-0 SHIELD	750-000-0		2
	PCB,MTU130,MULTI-0	701-007-0	A	1

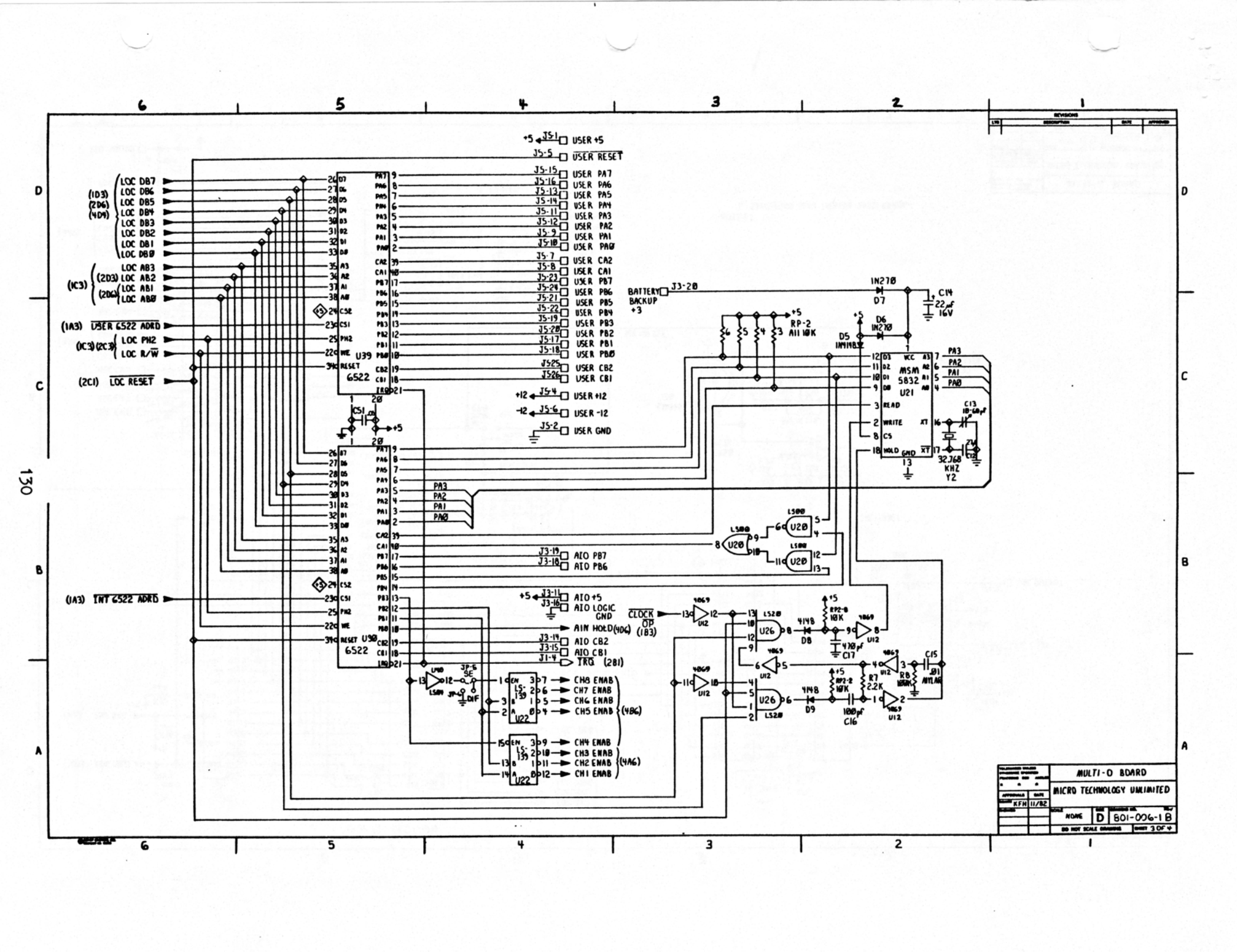
PARTS LAYOUT

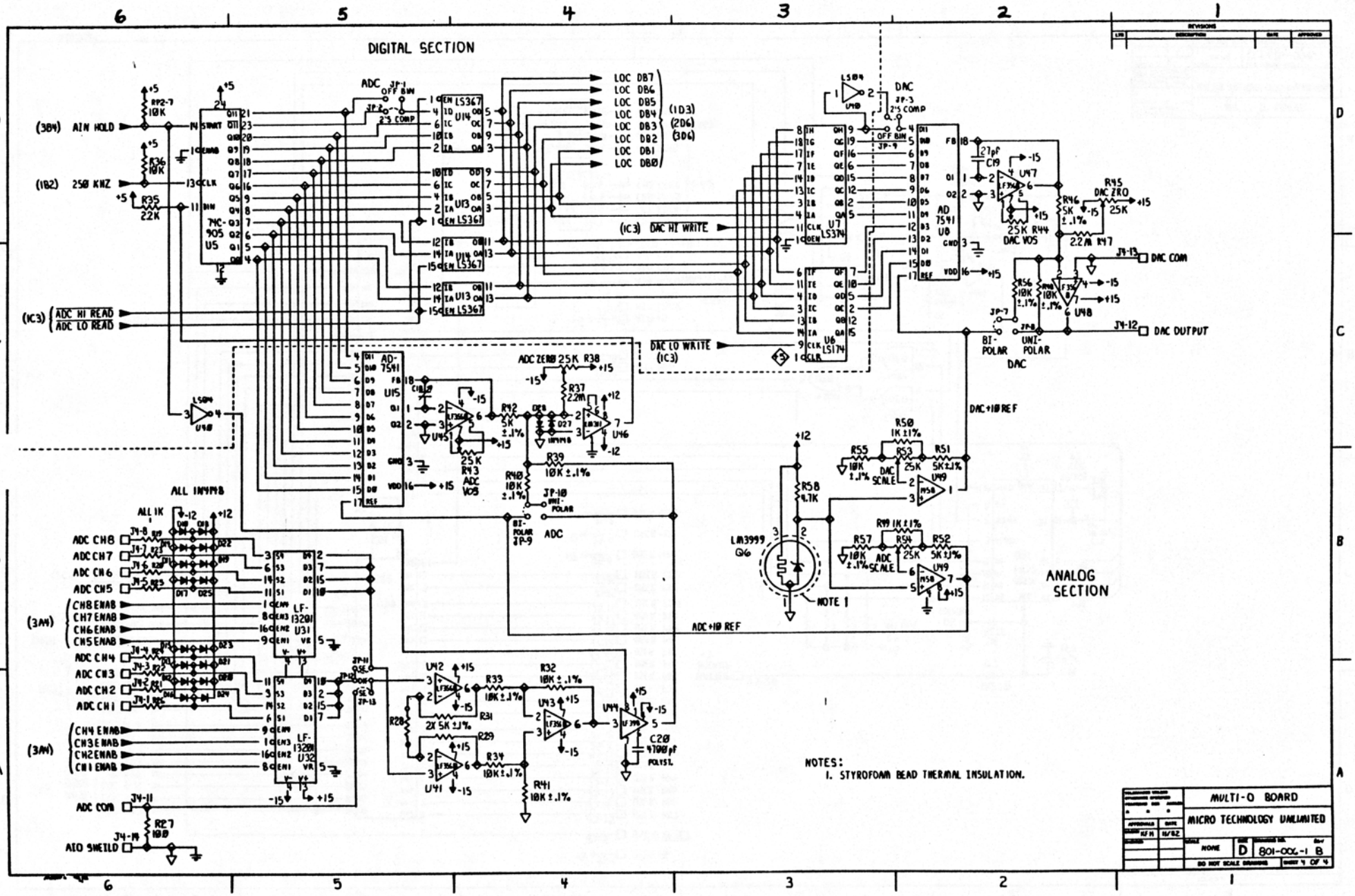


SCHEMATIC DIAGRAM




MULTI-O BOARD
MICRO TECHNOLOGY UNLIMITED
801-0006-1B
2651 U29
2651 U28
TMS-9914 U27
SN75161A
SN75160
BH-595 U16
LS-174 U17
All LS04 U25
All LS26
All LS126
10K RP1
10K RP1-8
5.0688 MHZ
P0 GND
P0 TXD
P0 DTR
P0 RTS
P0 TXC
P0 RXC
P0 RXD
P0 DSR
P0 DCD
P0 CTS
P1 TXD
P1 GND
P1 RXD
IEEE DIØ1
IEEE DIØ2
IEEE DIØ3
IEEE DIØ4
IEEE DIØ5
IEEE DIØ6
IEEE DIØ7
IEEE DIØ8
IEEE GND
IEEE SRQ
IEEE ATN
IEEE EOI
IEEE DAV
IEEE NRFD
IEEE NDAC
IEEE IFC
IEEE REN
LOC RESET (3C6)
IRQ (3A4)
BUS RESET
INT EN WRITE
INT STS READ
P1 2651 ADRD
P0 2651 ADRD
LOC R/W
LOC PW2
9914 CE
LOC AB2
LOC AB1
LOC AB0
LOC DB7
LOC DB6
LOC DB5
LOC DB4
LOC DB3
LOC DB2
LOC DB1
LOC DB0


MULTI-O BOARD
MICRO TECHNOLOGY UNLIMITED
NONE
D
801-006-18
SHEET 3 OF 4
DO NOT SCALE DRAWING
130
U39 6522
U30 6522
U22
U21
U20
U26
U12
Y2
32.768 KHZ
USER +5
USER RESET
USER PA7
USER PA6
USER PA5
USER PA4
USER PA3
USER PA2
USER PA1
USER PA0
USER CA2
USER CA1
USER PB7
USER PB6
USER PB5
USER PB4
USER PB3
USER PB2
USER PB1
USER PB0
USER CB2
USER CB1
USER +12
USER -12
USER GND
BATTERY BACKUP +3
AIO PB7
AIO PB6
AIO +5
AIO LOGIC GND
AIN HOLD(404)
AIO CB2
AIO CB1
TRQ (281)
CLOCK OP (183)
CH8 ENAB
CH7 ENAB
CH6 ENAB
CH5 ENAB
CH4 ENAB
CH3 ENAB
CH2 ENAB
CH1 ENAB
LOC DB7
LOC DB6
LOC DB5
LOC DB4
LOC DB3
LOC DB2
LOC DB1
LOC DB0
LOC AB3
LOC AB2
LOC AB1
LOC AB0
USER 6522 ADRD
LOC PH2
LOC R/W
LOC RESET
INT 6522 ADRD
IN270
MSM 5832
RP-2 All 10K
RP2-8 10K
RP2-2 10K
R7 2.2K
R8 100K
C13 10-60pF
C14 22µF 16V
C15 .01 MYLAR
C16 100pF
C17 470pF
D5 IN4148
D6 IN270
D7 IN270
D8 4148
D9 4148
JP-5 SE
JP-4 DIF


DIGITAL SECTION
ANALOG SECTION
ADC
DAC
(3B4) AIN HOLD
(1B2) 250 KHZ
(IC3) ADC HI READ
(IC3) ADC LO READ
(IC3) DAC HI WRITE
DAC LO WRITE (IC3)
LOC DB7
LOC DB6
LOC DB5
LOC DB4
LOC DB3
LOC DB2
LOC DB1
LOC DB0
(1D3)
(2D6)
(3D4)
ADC ZERO 25K R38
R43 ADC VOS
R93 25K
R44 DAC VOS
R95 DAC ZERO 25K
R46 5K ±.1%
2.2M R47
DAC COM
DAC OUTPUT
BI-POLAR
UNI-POLAR
DAC +10 REF
ADC +10 REF
ALL 1K
ALL IN4148
ADC CH8
ADC CH7
ADC CH6
ADC CH5
ADC CH4
ADC CH3
ADC CH2
ADC CH1
CH8 ENAB
CH7 ENAB
CH6 ENAB
CH5 ENAB
CH4 ENAB
CH3 ENAB
CH2 ENAB
CH1 ENAB
(3A4)
(3A4)
ADC COM
AIO SHEILD
R27 10Ω
LM3999 Q6
NOTE 1
C20 4700pf POLYST.
NOTES:
1. STYROFOAM BEAD THERMAL INSULATION.
MULTI-O BOARD
MICRO TECHNOLOGY UNLIMITED
801-006-1 B
SCALE NONE
SIZE D
DO NOT SCALE DRAWING