MTI-850/1650A
Multiple Terminal Interface
Technical Manual

Revision: **B**
Document Number: **80-000098-5**
Revision Date: 1/31/86

Systech Corporation
6465 Nancy Ridge Drive
San Diego, CA 92121
(619) 453-8970
Telex: ITT 499-0507 "SYSTC"

MTI-850/1650A
Multiple Terminal Interface
Technical Manual

January 1986

Manual Part No. 80-000098-5

Date	Revision	By	Description	Pages Affected
Jan 1986	B	ELB	Corrections to 3.4.1.4, new 3.4.3.2; capacitors in appendix I identified. (note: change to 3-29 - 3-39, 3-41, and 3-44 - 3-71 consisted of page number only)	2, 3, 6, 3-24, 3-27 - 3-71, I-1 - I-7

MTI-850/1650A MULTIPLE TERMINAL INTERFACE

TECHNICAL MANUAL

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

Section 2: CONFIGURATION AND INSTALLATION (continued)

Section 3: PROGRAMMING

TABLE OF CONTENTS (continued)

Section 3: PROGRAMMING (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

MTI-850/1650A MULTIPLE TERMINAL INTERFACE

TECHNICAL MANUAL

LIST OF ILLUSTRATIONS

SECTION 1: INTRODUCTION

MTI-850/1650A TECHNICAL MANUAL

Section 1: Introduction

This section introduces you to the manual and gives a general description of Systech's MTI-850/1650A Multiple Terminal Interface, for computer systems using the Multibus* design.

1.1 MANUAL DESCRIPTION

1.1.1 Potential users

This manual is designed to serve a number of different types of needs. You will need it under one or more of the following circumstances:

(a) You have purchased an MTI-850/1650A and need to install it in a Multibus-based computer system.

(b) You are thinking about purchasing an MTI-850/1650A.

(c) You need an overview of the product to aid you in selling it or systems which include it, or to be able to answer questions about it.

(d) You need a functional description of the product's structure and behavior to aid in designing systems which include it.

(e) You must write computer software to work with the MTI-A series interfaces.

(f) An existing computer system which includes an MTI is malfunctioning and you must troubleshoot the interface.

1.1.2 Manual design

To meet the needs described, the **MTI-850/1650A Technical Manual** has been designed as follows (which sections apply to which needs are noted in parentheses.)

Section 1: Introduction (a, b, c, d, e, f)

Introduces the manual and the product. Describes the MTI-850/1650A physically, and gives an overview of its functioning in a computer system. Answers commonly asked questions.

* Multibus is a trademark of Intel Corporation.

Section 2: Installation and operation (a, d, e and f)

Describes unpacking the product and configuring it for the system it will be used with. Describes the location of the MTI-850/1650A's DIP switches, and how to set them. Explains installing the product in the Multibus and connecting cables, or removing and reinstalling, and describes normal operation.

Section 3: Programming (d, e and f)

Describes the various operating modes of the MTI's serial communications ports. Details the software interface with the product. Gives examples of programming approaches.

Section 4: Theory of operation (d and f)

Goes into more detail about the internal components and operation of the MTI-850/1650A.

Section 5: In case of trouble (a, c, and f)

Suggests possible sources of problems; explains what various patterns on the controller board's LEDs can mean; details Systech Corporation's warranty and repair policy.

Appendix A: Communications concepts

Gives an overview of serial communications in an effort to assist you in deciding which features of the MTI-850/1650A to implement.

Appendix B: Cable descriptions

Lists parts and gives pinouts for both internal and external cables for the MTI-850/1650A.

Appendix C: Current loop

Explains the current loop option, available from the factory.

Appendix D: Using the USARTS directly

Explains where in the manual you can find information for using synchronous communications or special applications where directly accessing the Universal Synchronous/Asynchronous Receiver/Transmitter (USART) chips may be necessary.

Appendix E: +12/-12 volts

Explains use of a jumper on the MTI controller board that makes +/-12vdc optionally available through the ribbon cable. Standard product is shipped with +/-12 volts disconnected.

Appendix F: FCC information

Contains information on Federal Communications Commission requirements regarding the use of shielded cables.

Appendix G: Using the SUN 68000 processor and derivatives

Explains so-called "byte-swapped" systems such as the SUN Microsystems 68000 design, and how this is handled on the MTI for users dealing with a reversed byte order.

Appendix H: Upgrade of MTI-850A to MTI-1650A.

Gives the steps involved in upgrading an MTI-850A to an MTI-1650A.

Appendix I Response control capacitors

Covers the use of optionally available capacitors for controlling the slew rate of the MTI driver, and high-frequency noise pulses. These capacitors may be necessary to meet RS-232 and CCITT, V.24 specifications.

Appendix J: Signetics 2661 (USART) specification

Contains a reprint of the technical specification for the Signetics 2661 Enhanced Programmable Communications Interface (EPCI), referred to as a USART chip.

Appendix K: Warranty

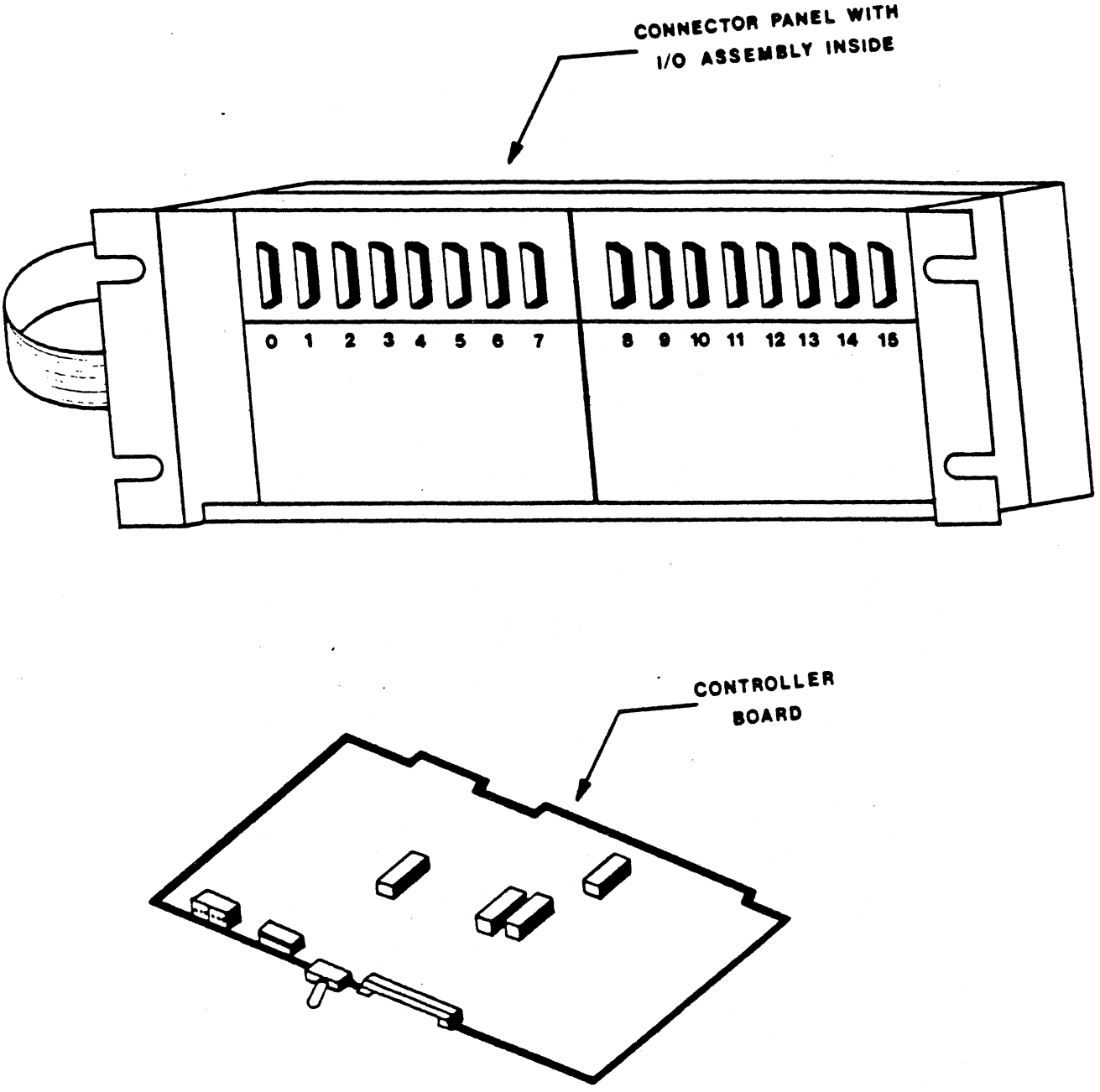


Figure 1-1 MTI-1650A (controller board and USART assembly)

1.2 PRODUCT DESCRIPTION

The MTI-850/1650A (Multiple Terminal Interface) is a two-piece serial communications subsystem. It consists of a controller board which slides into a Multibus, and a 19-inch chassis panel which mounts on the rack housing the Multibus, presenting a bank of eight or 16 connectors facing out from the back of the rack. The controller card connects to the chassis panel through a 50-conductor ribbon cable and a 5-conductor power cable. **Figure 1-1** shows the two parts of the product unmounted.

The MTI-850A has eight connectors and can communicate with eight devices simultaneously, while the MTI-1650A has 16 connectors and therefore can communicate with 16 devices at once. There are four banks of switches on the controller board, to be configured before the MTI-850/1650A is installed. These switches allow the user to specify the Multibus addresses which the MTI responds to, and other details of its software interface with the host computer.

Also on the controller board is a toggle switch for placing the device in a special diagnostic mode. In this mode, it can perform self-diagnosis of its components and identify problems by displaying a pattern on a bank of eight Light Emitting Diodes (LEDs).

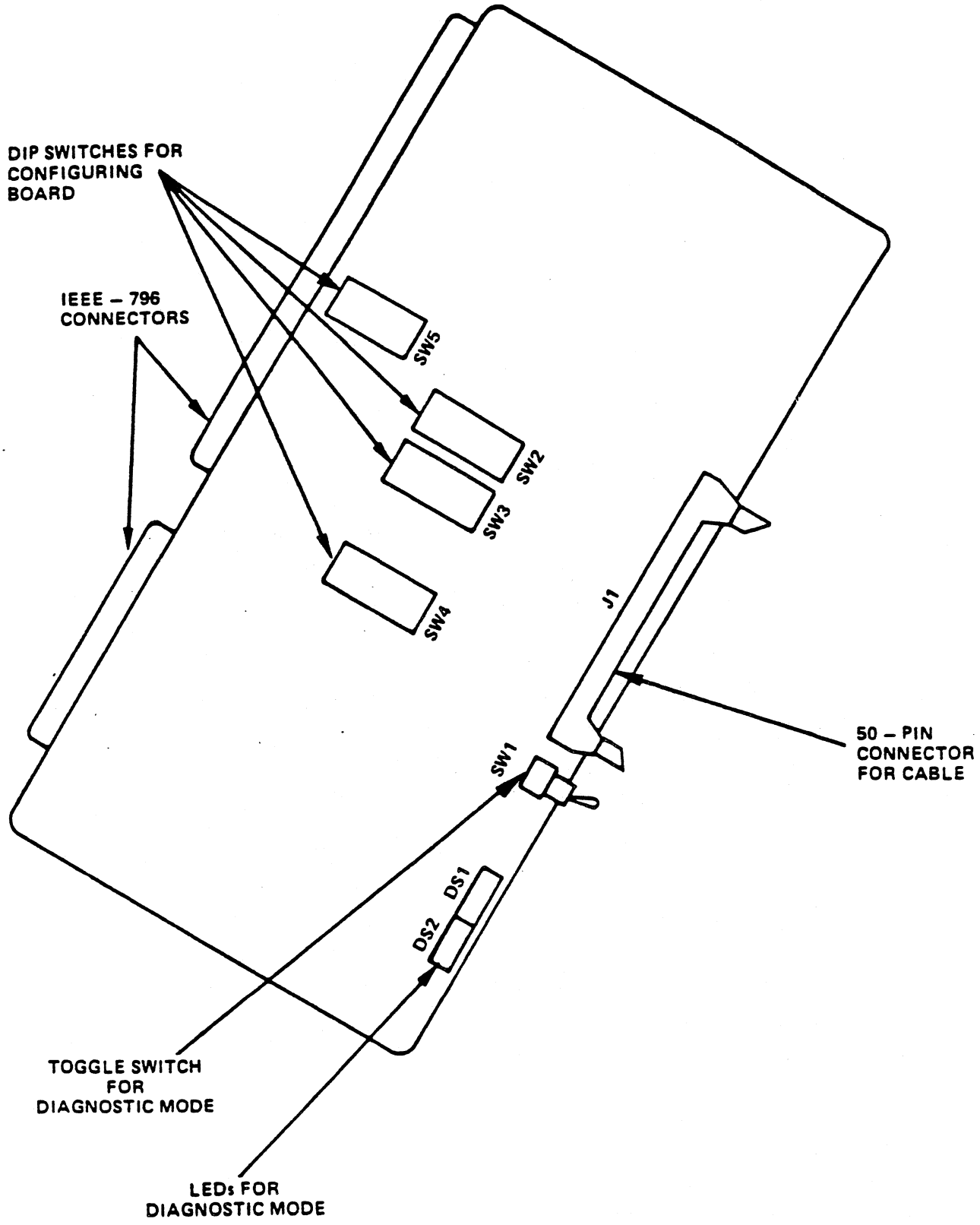


Figure 1-2 Controller board

1.3 FUNCTIONAL OVERVIEW

The purpose of the MTI-850/1650A is to permit a host computer's software to initiate transfers of data between the host and up to eight or 16 serial devices, while using a minimum amount of host computer time and a small number of Multibus addresses.

The MTI-850/1650A acts as a "multiplexer," appearing as only one interface to the host computer while offering eight or 16 interfaces to the serial devices. Without such a multiplexer, the host software would have to individually transfer characters to and from each device.

The MTI-850/1650A operates in two modes to transfer characters between serial devices and the host computer. In single character transfer mode, the MTI transfers one character at a time to or from the host computer's Central Processing Unit (CPU). In block transfer mode, the MTI transfers characters in blocks directly between the serial devices and the host computer's memory, bypassing the host CPU.

This shortcut is achieved through a special integrated circuit "chip" called a Direct Memory Access (DMA) controller, which is designed to work with the Multibus for high-speed transfers between memory and devices. Block transfer mode is not only faster, but also frees the host CPU to perform other computing tasks, thereby increasing overall system efficiency.

These two modes can be freely mixed. Not only may different devices connected to the MTI-850/1650A operate in different modes, but a single device may change modes or enable both modes at once, all under control of the host software.

The serial devices attached to the MTI-850/1650A may be of several types:

-- Asynchronous devices using the RS-232C standard, operating either as the modem or terminal end of the interface;

-- Asynchronous current loop devices, such as Teletype Model 33 (provided the current loop option is installed);

-- Synchronous devices. (If synchronous devices are used, all protocol handling must be done by the host software.)

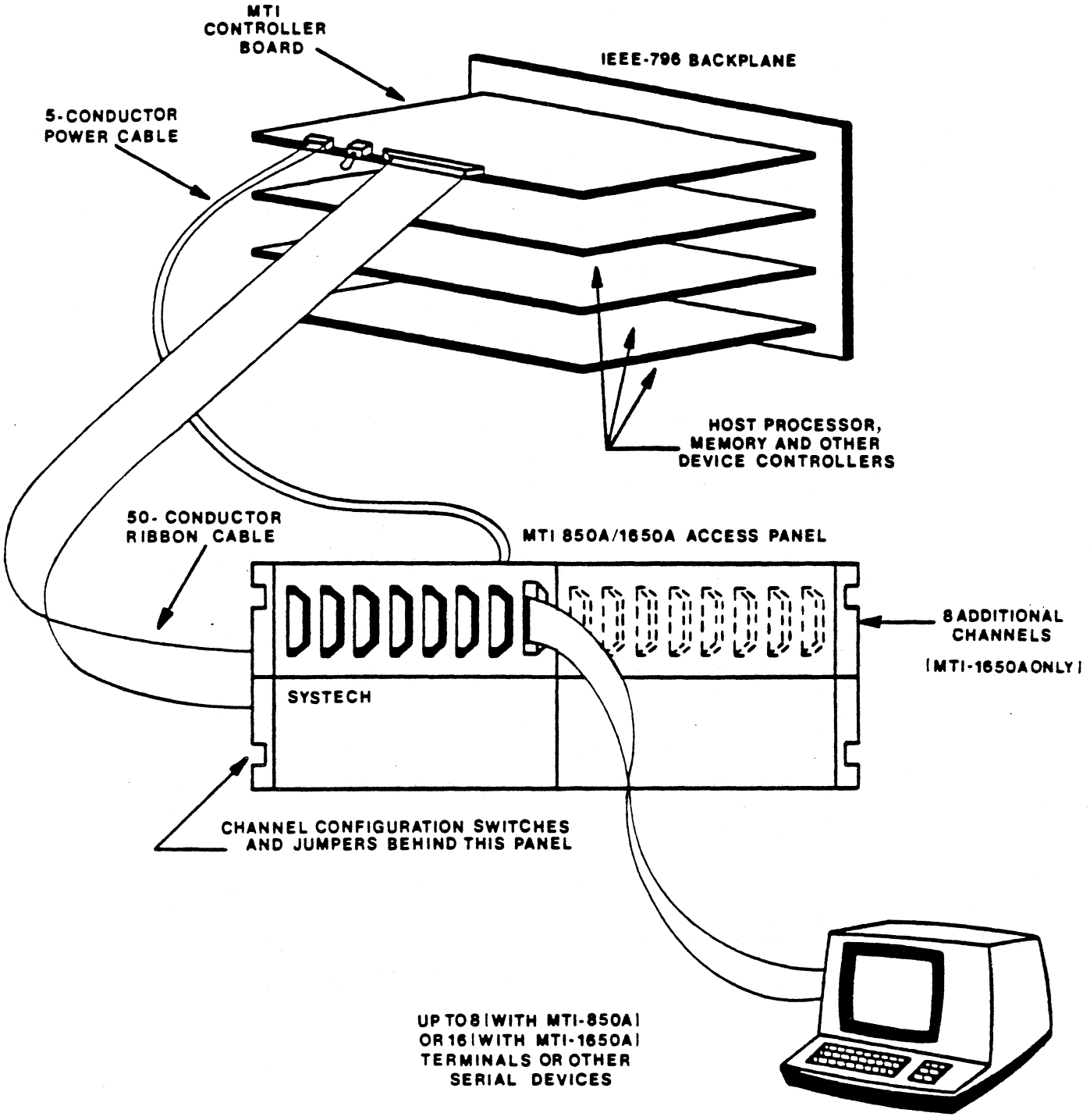


Figure 1-3 System overview

The host software may configure each channel separately to work with any of these serial device types, at any of these baud rates:

50, 75, 110, 134.5, 150, 300, 600, 1200, 1800
2000, 2400, 3600, 4800, 7200, 9600, and 19200.

The user can set switches on the MTI-850/1650A to select the default characteristics which all channels will initially have on power-up.

The MTI's functional components and their operations are described in more detail in the remainder of this section.

1.3.1 System components

The MTI-850/1650A occupies one slot in the physical chassis of the Multibus for the controller board, or main circuit board, shown in **Figure 1-2,** and responds to four Multibus addresses. When it is installed in a host computer's Multibus and connected to serial devices, the entire system can be represented by the following components, as shown in the diagram in **Figure 1-3.**

1.3.1.1 Host computer

The host computer consists of these components:

a) The host Central Processing Unit (CPU), sometimes called just "the host processor."

b) The host memory, which is read/write, high-speed Random Access Memory (RAM).

c) The Multibus, connecting the processor and RAM to all the other devices.

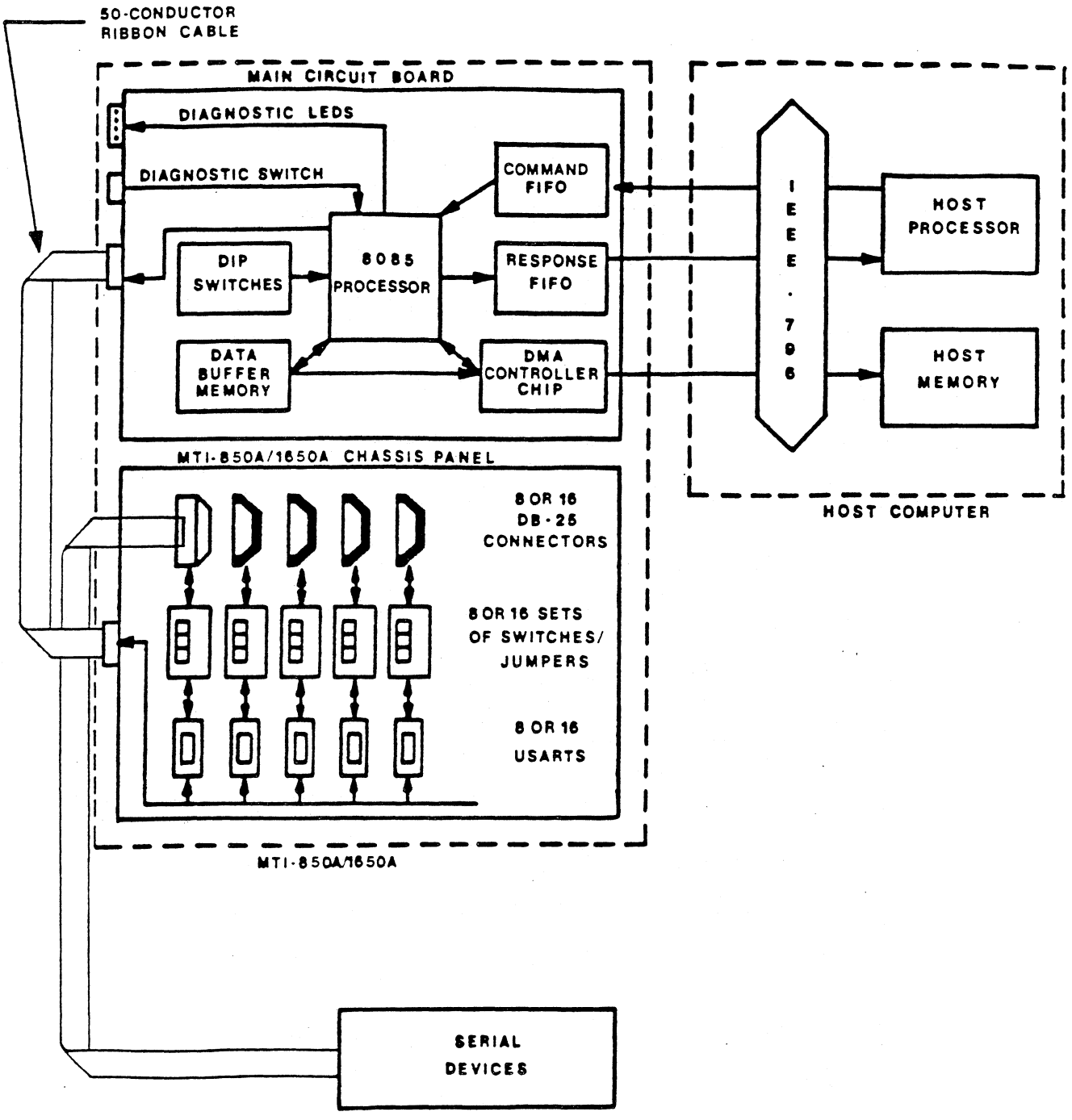


Figure 1-4 System components

1.3.1.2 MTI-850/1650A

The MTI-850/1650A contains these components:

a) The controller board, having:

1) An 8085 processor, which has its own high-speed, read/write Random Access Memory for "scratch pad" use, as well as its own software in high-speed Read Only Memory (ROM).

2) High-speed, read/write RAM for data buffering.

3) A "pipeline" for data, called a "First In First Out" (FIFO) buffer, for commands from the host processor to the MTI-8501650A.

4) Another FIFO for responses from the MTI-850/1650A to the host processor.

5) A special integrated circuit chip designed to directly access the host memory through the Multibus without involving the host processor in high-speed transfers. It is called a Direct Memory Access (DMA) controller chip (Intel 8237A).

6) Four banks of switches mounted in chip-like Dual Inline Packages (DIPs).

7) Eight Light Emitting Diodes (LEDs).

8) A toggle switch for selecting diagnostic mode.

9) A 50-pin cable connector and a 5-pin power cable connector.

b) A 50-wire ribbon cable connecting the controller board to the USART board.

c) The chassis panel, containing:

1) A 50-pin cable connector and 5-pin power cable connector.

2) Eight or 16 special high-speed serial communications chips, called Universal Synchronous/Asynchronous Receiver/Transmitters (USARTs).

3) Eight or 16 sets of chip-like sockets for placing jumper plugs to select port characteristics (there will be two or

three sockets per port).

4) Eight or 16 25-hole cable connectors for attaching cables to serial devices.

5) Optionally, eight or 16 banks of switches for configuring ports for current loop use (if the current loop option is installed).

1.3.1.3 Outside cables

A number of cables (between three and 25 conductors each) connecting the MTI-850/1650A to each serial device used.

1.3.1.4 Serial devices

A number of serial devices, such as operator terminals (keyboard and screen), letter quality printers, modems for telephone line communications, and paper tape readers/punches, might make up the remainder of the system.

1.3.2 Software capabilities

All elements of the system work together to ensure the orderly transmission of data between the host computer and its serial devices. Many of these elements operate simultaneously, so it is difficult to describe a linear sequence of data flow. However, all data transfers begin with actions taken by the host computer's software, so that is a logical place to begin.

When you turn on the power for your host computer system, the MTI-850/1650A comes to life as well. The 8085 processor, following its program in ROM, performs its self-test operation (see **Section 1.3.3**). It then configures the eight or 16 channels (by programming the USARTs and preparing other circuits on the MTI-850/1650A) as specified in the default channel configuration switches. The MTI then sits idle, waiting for a command from the host software.

1.3.2.1 Host direction

The host software can directly communicate with the MTI-850/1650A through any of four Multibus addresses. By reading from and writing to these addresses, the host software directs the circuitry of the MTI-850/1650A to perform various functions, such as:

-- Reset the entire MTI, returning all parameters to their default values and cancelling all data transfers in progress.

-- Read the status of the MTI, finding out if it has data or control information for the host software.

-- Specify which events cause the MTI to interrupt the host processor.

-- Write data or a command into the command FIFO, sending it on its way to the 8085.

-- Read data or a response from the response FIFO, which came from the 8085.

These last two actions involving the FIFO allow the host software to transfer data to and from serial devices, as well as commands to and responses from the 8085.

1.3.2.2 8085 commands

Commands to the 8085 fall into the following categories:

-- Channel configuration commands

These change any channel's characteristics, or other MTI-850/1650A characteristics. (Not required before beginning data transfers if the default characteristics selected in switches are adequate.)

-- Single character transfer commands

Read from or write to any serial device a character at a time.

-- Block transfer commands

Initiate transfer of data directly between the host memory and any serial device without further software action required (called Direct Memory Access, or DMA).

-- USART control commands

Directly program or interrogate a particular channel's USART chip.

-- Miscellaneous commands

Perform other occasionally needed functions.

After receiving any command through the command FIFO and executing it, the 8085 responds by setting a special bit in one of its status bytes, called the MTI READY bit. It may also provide an additional response to the host software through the response FIFO, depending on the type of command requested and whether an error occurs during execution.

1.3.3 Firmware execution

The MTI-850/1650A's on-board 8085 processor executes the firmware, which is software stored in Read Only Memory (ROM), so-called because it cannot change during execution. It uses the scratch pad memory to store pointers and variables that must change during execution.

The firmware provides the "intelligence" for the MTI-850/1650A, and coordinates the other circuits to perform the operations requested by the host software. The firmware performs the following operations.

1.3.3.1 Self-test

On power-up, or after a reset of the Multibus, the firmware executes a self-test operation on all of the MTI-850/1650A's circuitry. If it finds a problem, it displays an error code on the Light Emitting Diodes (LEDs) on the edge of the controller board.

If you flip on the diagnostic toggle switch, the firmware enters a special diagnostic mode, described in **Section 5** (Troubleshooting), to aid in identifying problems.

1.3.3.2 Default channel configuration

After a successful self-test, the firmware reads the DIP switches on the MTI-850/1650A which select the default channel characteristics, and uses their settings to configure all of the channels. To do this, the firmware programs the USARTs, which actually handle the details of each channel's communication with a serial device. (These details include the parity, character length, number of stop bits and baud rate.)

1.3.3.3 Memory partition

After it programs the USARTs, the firmware partitions the buffer memory on the MTI into 16 or 32 buffers, one for each channel's input and output. Each buffer acts as an independent FIFO, typically storing up to 512 characters in each buffer.

1.3.3.4 Commands

Once everything is set up, the firmware waits for commands from the host software to come through the command FIFO. When the host processor has stuffed a complete command into the command FIFO, it signals the MTI-850/1650A to execute the command, and the firmware decodes the command bytes and acts accordingly.

When a command requests data transfer, very different chains of events occur depending upon whether the transfer is single character or block mode.

1.3.3.5 Data transfers

If a single character transfer is requested, the firmware itself transfers the data between the command FIFO and the buffer memory. If, for example, the command is to write to a device, the 8085 takes the character from the command FIFO and puts it in the buffer memory. Meanwhile, independent of the host software, the firmware also moves data from the memory buffer to the appropriate USART whenever that USART signals that it is ready for more.

In contrast, when the host software requests a block transfer, the firmware sets up the DMA chip to handle the transfers between host memory and the MTI-850/1650A's buffer memory.

For example, a block write command causes the 8085 to pass the DMA controller chip the starting address of the data in the host memory, as well as the number of characters. The 8085 and the host processor can then each go on to other tasks. The DMA controller actually reads the data directly from host memory and loads it into the MTI-850/1650A's buffer memory (hence the name Direct Memory Access).

When the transfer is complete, the DMA controller interrupts the 8085. As in the case of a character write, the firmware also moves data from the memory buffer to the appropriate USART whenever that USART signals that it is ready for more. The firmware returns response or error codes to the host software through the response FIFO whenever it completes a command.

1.3.4 DMA controller operation

When instructed to by the firmware, the DMA controller directly accesses the host memory through the Multibus. For a block write operation, the DMA controller begins at the specified memory location and loads characters one after another into the appropriate buffer in the MTI-850/1650A's buffer memory, until it reaches the character count provided by the firmware.

When the firmware (commanded by the host software) enables block reading, the DMA controller takes characters from the appropriate memory buffer when they become available and loads them into the host memory beginning at the requested start location.

The DMA controller terminates the read operation when it has loaded the specified number of characters into the host memory, or when it encounters a termination character, whichever comes first. The host software chooses the termination character(s) for each block read.

1.3.5 Operation of USARTs

There are eight or 16 USART chips on the MTI-850/1650A's chassis panel, one for each channel. These circuits do much of the work of actually communicating with serial devices, including:

-- Serial-to-parallel and parallel-to-serial conversion of eight-bit characters (the MTI-850/1650A stores and manipulates characters internally as parallel bits).

-- Simultaneous data transmission and reception.

-- Generation of a baud rate clock, if needed.

-- Optional division of an external clock signal by one or 16.

-- Generation and stripping of start and stop bits for asynchronous transfers.

-- Generation and stripping of synchronization characters for synchronous transfers.

-- Parity generation and verification, if needed.

-- Protocol handling for RS-232C devices.

The firmware can program the USARTs as mentioned above, or the host software can program them directly for special applications.

1.4 COMMON QUESTIONS AND ANSWERS

Q: What is the MTI-850/1650A?

A: MTI stands for Multiple Terminal Interface: a controller board, or interface, which allows many terminals or other serial devices to be connected to one microcomputer.

There are actually two products: the MTI-850A (eight-channel version) and the MTI-1650A (16-channel version).

Q: What does it look like?

A: The MTI-850/1650A consists of two main pieces: the **controller board,** which is 6 3/4 by 12 inches and slides into any Multibus slot, and the **chassis panel,** which mounts on the back of a Multibus 19-inch rack and has the connectors where device cables attach. See **Figure 1-1** for an illustration of the complete product.

Q: What does the "A" mean?

A: The MTI series includes an MTI-850/1650B, which features the same controller board. Instead of the 19-inch rack-mountable chassis, however, the MTI-850/1650B has a USART board with two or four 60-pin connectors on the board for cables to terminals.

Q: What microprocessors does the MTI-850/1650A work with?

A: Any eight-bit or 16-bit microprocessor designed into a system based on the Intel Multibus.

Q: What is the Multibus?

A: The Multibus is a standard, proposed by Intel Corporation and approved as the "IEEE-796," that defines three things: the physical shape of the chassis, the electrical characteristics of the circuits, and the protocols used for system components to communicate. A bus meeting these three criteria is referred to as a Multibus.

Q: What kind of terminals or other serial devices will it connect to?

A: Any serial device using an asynchronous RS-232C, asynchronous current loop, or synchronous interface. (The current loop option must be installed on the MTI-850/1650A for it to be used with current loop devices.) The host software must perform all protocol functions if synchronous devices are used. NOTE:" Host software must perform any protocol functions if synchronous modes are used. Jumpers are provided for synchronous communication, as detailed in **Section 2.**

Q: Can I configure channel characteristics from software?

A: Yes. By issuing commands to the MTI-850/1650A, the host software can change most channel characteristics, including baud rate, parity, and number of stop bits for asynchronous operation.

Q: Must I configure channel characteristics from software?

A: No. On power-up, the MTI-850/1650A configures all channels with default characteristics which the user can select with switches on the controller board.

Q: Does the MTI-850/1650A use single character transfers (programmed I/O) or block transfers (Direct Memory Access)?

A: Both. Users can freely mix transfer modes for maximum efficiency in any circumstance.

Q: Does the MTI-850/1650A use polling or interrupts to indicate status to the host computer?

A: It supports both methods.

1.5 MTI-850/1650A SPECIFICATIONS

A. Dimensions

Controller board

6.75 inches by 12 inches (17 cm by 31 cm).

I/O access panel

19 by 7 by 3 inches (48.4 by 17.6 by 7.9 cm).

B. Weight

Controller board

13.25 ounces (376 grams), not including cables.

I/O access panel

4 lbs. 9 ounces (2073 grams), including cables.

C. Logic

MSI/LSI LSTTL logic.

D. System slot requirements

One standard Multibus slot.

E. Environment

0 to 50 degrees Centigrade operating.

-10 to +70 degrees Centigrade storage.

10% to 90% humidity (non-condensing).

F. Data transfer method

Direct Memory Access (DMA) or programmed I/O 24-bit addressing.

G. I/O addressing

Eight or 16-bit selection, switch selectable.

H. Data transfer rate

9600 baud, 16 channels half-duplex at 100 percent loading.

I. I/O interface

RS-232C supported on all eight or 16 channels. Configurable as DCE or DTE with Systech-supplied jumpers.

J. Power requirements

MTI-850A:
+5 volts ± 5% at 3.7 amperes nominal.
+12 volts ± 5% at 250 mA nominal.
-12 volts ± 5% at 200 mA nominal.

MTI-1650A:
+5 volts ± 5% at 4.7 amperes nominal.
+12 volts ± 5% at 220 mA nominal.
-12 volts ± 5% at 200 mA nominal.

SECTION 2: CONFIGURATION & INSTALLATION

MTI-850/1650A TECHNICAL MANUAL

Section 2: Configuration and installation

This section explains determining the settings for the DIP switches, setting them, connecting any jumpers necessary, then installing the MTI-850/1650A in a Multibus-based computer system and checking for normal operation.

2.1 UNPACKING

The MTI-850/1650A arrives as two main pieces: the main circuit board, and the chassis panel. The chassis panel is fully assembled, and comes attached to the 50-conductor ribbon cable and the 5-conductor power cable.

You may want to make sure that the unit has arrived with the options that you ordered. Counting the number of DB-25 (standard RS-232) connectors on the chassis panel will tell if it is the eight-channel (MTI-850A) or 16-channel (MTI-1650A) version.

If you have ordered the current loop option, you can verify that it is in place by checking within the chassis itself. Remove the three Phillips head screws on the cover panel with the Systech logo (the one labeled "REMOVE THIS PANEL TO CONFIGURE"). If the current loop option is installed, there will be one Dual In-Line Package (DIP) switch pack of four switches below each connector, between the jumper blocks, for each channel. An illustration of this is shown in **Appendix C.** If the current loop option is not installed, these DIP switches will be absent.

CAUTION: IF THERE IS ANY EVIDENCE OF PHYSICAL DAMAGE
> > > DO NOT CONTINUE. DO NOT INSTALL A DAMAGED BOARD. < < <

2.2 CONFIGURATION

The DIP switch configuration of the MTI-850/1650A can be specified at the time of ordering. If it is not, you can configure the switches yourself with the following instructions.

Some of the options you will set require that you know some details of the hardware devices and of the host software's operations. You may need to talk to system designers and programmers, or refer to other system documentation. Consult the programmer who has written or modified the host software to work with the MTI; that person in turn will have to refer to **Section 3** of this manual to understand the product's software interface.

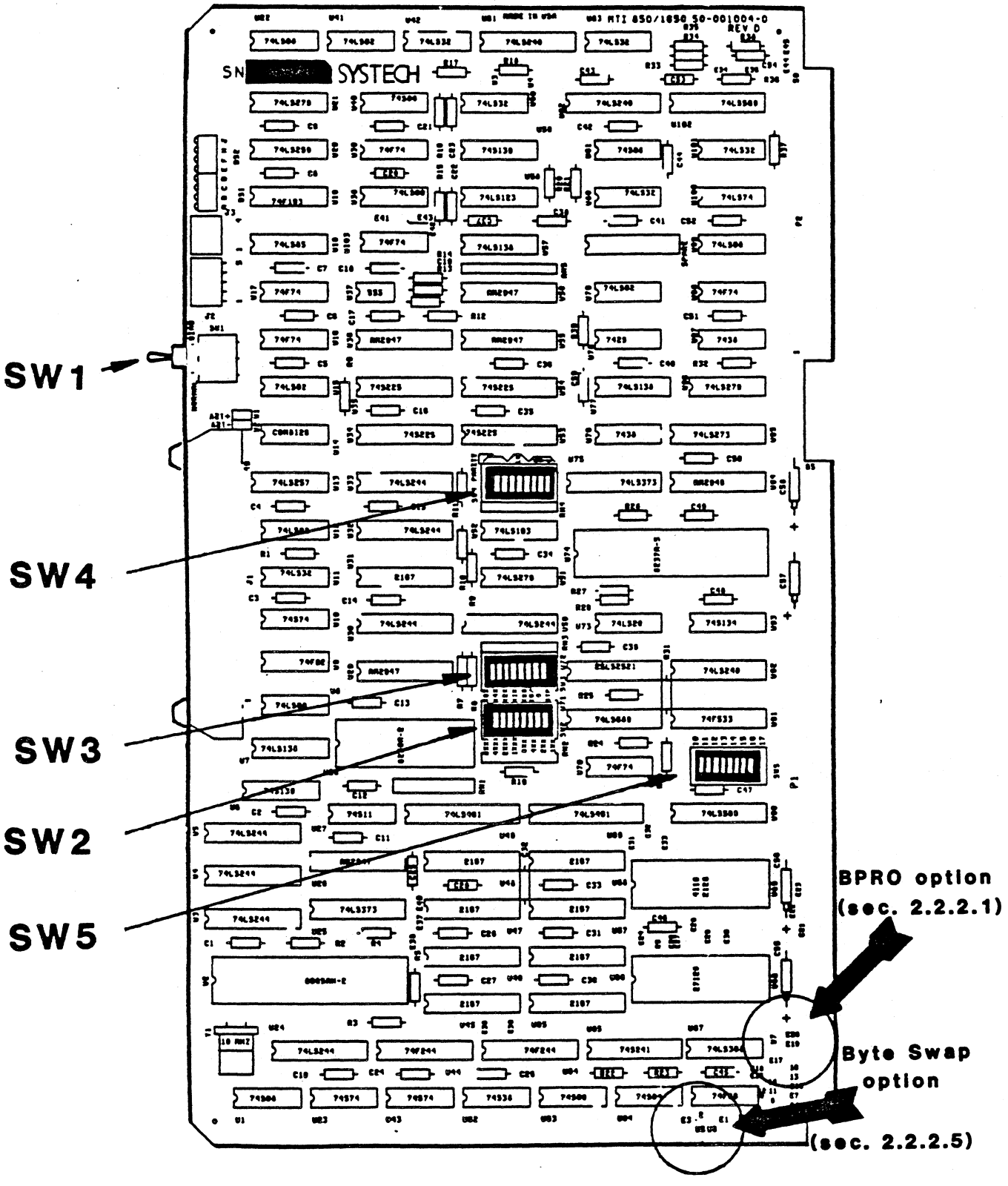


Figure 2-1 Controller board DIP switches

If you have questions regarding the options used by your system, you may want to refer to **Appendix A,** (Communications Concepts), which has been included to assist you in determining which of the MTI-850/1650A's features to employ. If you already know what options you want, you can continue immediately from here.

2.2.1 Bus arbitration

The MTI-850/1650A can operate as a bus master. This means that the board will request use of the IEEE-796 bus and, when given permission, move data directly to and from system memory. Since there can be more than one master controller on the Multibus, requests for use of the bus from different masters must be arbitrated.

Two methods are allowed by the IEEE-796 to arbitrate requests: either serial or parallel.

Parallel arbitration schemes allow an arbitrary number of boards to share the bus. Serial arbitration schemes, on the other hand, require that no more than three bus masters be installed in the IEEE-796 backplane. Bus masters are arranged in a row, or "daisy-chained", so that bus arbitration signals can be passed from one board to the next.

Your computer's backplane wiring or your application will determine if serial or parallel bus arbitration is to be used. If you do not know which method your system uses, check your manufacturer's documentation.

Since the MTI-850/1650A comes configured for serial arbitration, no change need be made if your system uses that method. If your system uses parallel arbitration, the two-pin jumper at W7 (between pads E19 and E20) must be **removed.** (Refer to **Figure 2-1.)**

Name	Switch location	Switch label	Meaning of setting
Number of stop bits	SW3-7 SW3-8	STP	Taken together: 7 8 --- 0 0 = invalid 0 1 = 1 stop bit 1 0 = 1.5 stop bits 1 1 = 2 stop bits
Parity type	SW4-1	E	0 = Odd 1 = Even
Parity control	SW4-2	E	0 = Disabled 1 = Enabled
Character length	SW4-3 SW4-4	LN	Taken together: 3 4 --- 0 0 = 5 bits 0 1 = 6 bits 1 0 = 7 bits 1 1 = 8 bits
Baud rate	SW4-5 SW4-6 SW4-7 SW4-8	R A T E	Taken together: 5 6 7 8 ------- 0 0 0 0 = 50 0 0 0 1 = 75 0 0 1 0 = 110 0 0 1 1 = 134.5 0 1 0 0 = 150 0 1 0 1 = 300 0 1 1 0 = 600 0 1 1 1 = 1200 1 0 0 0 = 1800 1 0 0 1 = 2000 1 0 1 0 = 2400 1 0 1 1 = 3600 1 1 0 0 = 4800 1 1 0 1 = 7200 1 1 1 0 = 9600 1 1 1 1 = 19200

Figure 2-2 DIP switch settings

2.2.2 Controller board DIP switches

As you hold the controller board of the MTI-850/1650A in front of you with the 50-pin cable header along one edge and the gold fingers that slide into the Multibus along the other, you will see the four banks of DIP switches near the center of the board, as shown in **Figure 2-1.** Notice that each bank of switches is labeled on the board with a number -- SW2, SW3, SW4 and SW5. (The diagnostic toggle switch is SW1.) These numbers will be used throughout the manual in reference to these sets of switches.

You will need to set the address switches on SW2 and part of SW3, the default channel configuration switches on part of SW3 and SW4, and the interrupt switches on SW5.

2.2.2.1 Address switches

As described in **Section 1.3.2,** the MTI-850/1650A responds to four device addresses on the Multibus. These addresses must begin, however, on an eight-address boundary. In other words, the three low-order bits of the address are provided by the MTI, so you must set only the 13 high-order bits. The switch labeled "8XXX" (SW2-1) is the highest-order bit; setting it to one adds 8000 hexadecimal to the device address value. The switch labeled "XX8" (SW3-5) is the lowest-order bit that can be set; setting it to one adds 0008 hexadecimal to the device address value.

For the address switches, OFF equals a binary ZERO; ON equals a binary ONE.

2.2.2.2 8- or 16-bit addressing

Set the address length with SW3-6, labeled "16/8". This switch informs the MTI-850/1650A whether the Multibus of your system is using 16-bit or eight-bit device addresses. Turn it OFF for 16-bit addresses, ON for eight-bit addresses.

2.2.2.3 Default channel configuration

The host software can reconfigure channel characteristics at any time. However, the default channel characteristic switches on SW3 and SW4 determine how the MTI-850/1650A will initially configure all eight or 16 channels on power-up. The switches and their meanings are shown in the list that follows; the meanings are also included on the worksheets.

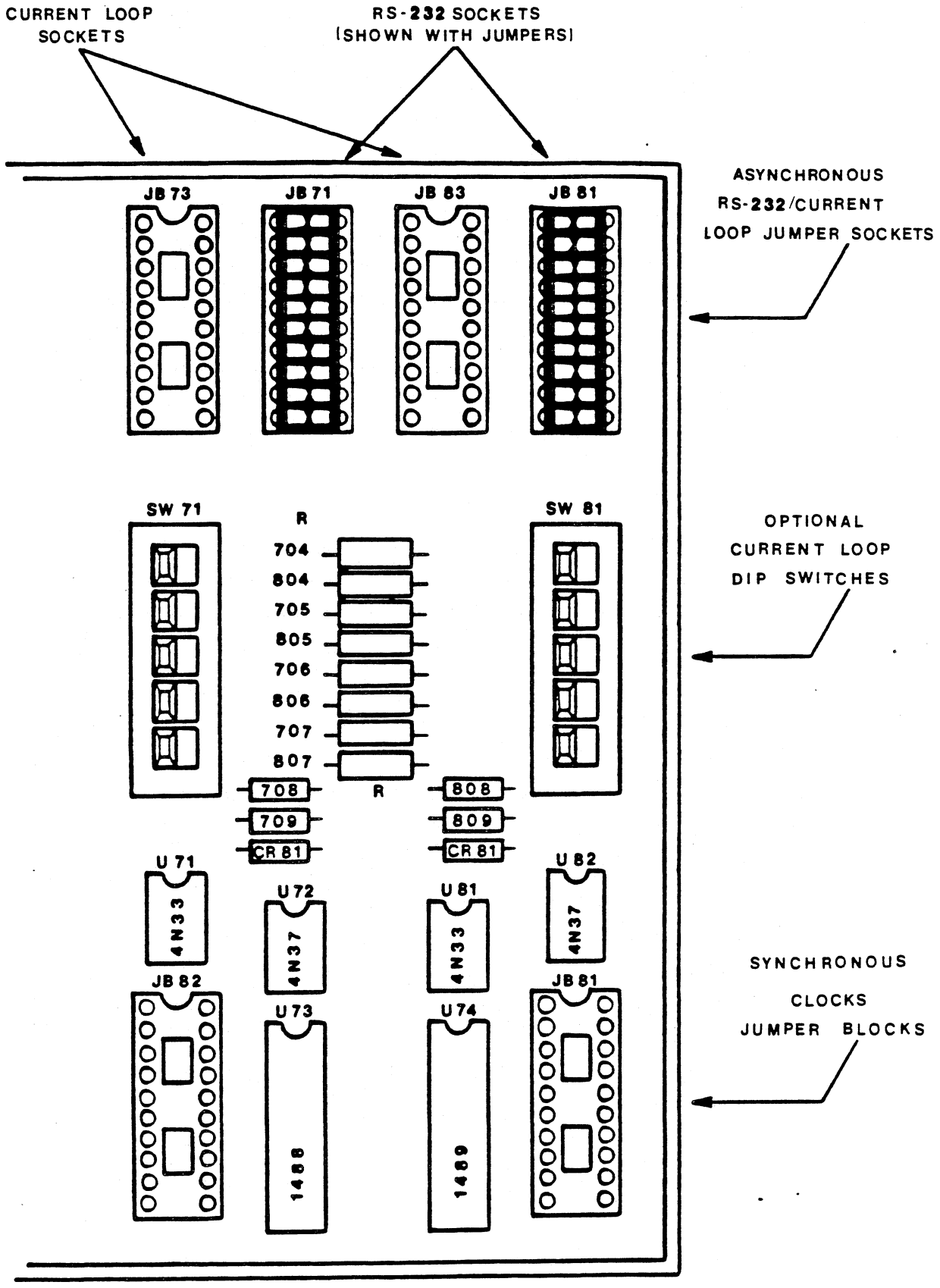


Figure 2-3 Jumper locations

2.2.2.4 Interrupt level switches

The switches at SW5 select the interrupt level of the MTI-850/1650A. Whoever programs the operating system will assign interrupt priority levels for each category of device. Determine which level the MTI-850/1650A should be at, and turn ON only one switch in the bank of eight switches on the DIP. CLOSED/ON is selected; OPEN/OFF is not selected. The switches are labeled I0 through I7 (on switches SW5-1 through SW5-8); I7 is the highest interrupt priority.

2.2.2.5 Byte order

The Multibus has a defined byte-within-word ordering, but some CPU's, particularly some 68000-based designs, use a "swapped" byte order instead (see **Appendix G.)** Two methods are available for dealing with swapped bytes. The system programmer can allow for it in the MTI driver program, or it can be accomodated with special circuitry on the controller board.

The first solution results in additional CPU overhead, which is unacceptable with high data-rate devices like the MTI-850/1650A. Instead, if your system has non-standard byte ordering, you can configure the MTI to accomodate it by moving a jumper. Refer to **Figure 2-1.** You will need to **remove the jumper at W6** (between pads E1 and E2), and **insert it at W5** (between pads E2 and E3). This jumper change will "correct" the memory addresses for DMA transfers, but the driver will still see the I/O registers as "twisted" and will need to correct for this. See Appendix H. Again, for normal operation, W6 is installed (not W5).

2.2.3 USART board jumpers

If any of the eight or 16 channels on the MTI-850/1650A needs a synchronous interface, you will need to install jumpers on that channel. Jumper pads are provided on the board to allow you to bring the external clock inputs/outputs from the USART chips to the 60-pin headers. Each double row of jumper pads is associated with a pair of channels. One pair is illustrated in **Figure 2-3.** To use external clocks, appropriate jumpers must be installed, and the USART chips must be programmed to generate (or accept) the clock signals on pins 9 and 25 of the USART chip (see **Appendix J** (Signetics 2661 Specifications). The wiring of the jumpers will be determined by the way the software uses lines from the USART board. See **Appendix D** (Using USARTs Directly) for more information.

If you do need to install jumpers, you will need to expose the jumper blocks and switches for each channel, which are inside the chassis panel cover. First remove the cover labeled "REMOVE THIS PANEL TO CONFIGURE" by removing the three Phillips head screws holding the panel cover in place. **Figure 2-3** shows how two adjacent channels will look. (Remember that the DIP switches shown will be in place only if you have the current loop option installed.)

For each of the eight or 16 channels, you must do the following:

If the channel needs an RS-232 interface, you must install a jumper block in the right-hand jumper socket of the two labeled "RS-232C/Current Loop jumper sockets" in **Figure 2-3.** There are three types of jumper blocks provided with the MTI-850/1650A, labeled "FOR MODEM," "FOR TERMINAL," and "FOR CURRENT LOOP." If this channel is to communicate with a modem (DCE), use a jumper block marked "FOR MODEM." (In other words, the MTI will emulate a terminal, or DTE, on this channel.) If this channel is to communicate with a terminal (DTE), use a jumper block marked "FOR TERMINAL." (The MTI will emulate a modem, or DCE, on this channel.)

If the channel needs a synchronous interface, you may need to install a custom jumper block in the socket labeled "synchronous jumper sockets" in **Figure 2-3.** The wiring will be determined by the way software uses lines from the USART. See **Appendix D** for more information.

2.2.2.1 Current loop

Current loop is a factory option that must be specially ordered. These instructions apply only if your board has been configured for current loop by the factory. Refer to **Appendix C** for more information.

If a channel needs a current loop interface, use a jumper block labeled "FOR CURRENT LOOP" on the left jumper block (labeled "RS-232C/Current Loop jumper sockets" in **Figure 2-3.)** If you are using current loop, you will also need to set the current loop switches. This is explained in **Appendix C.** In addition, a 20-pin shunt must be installed, as shown in **Appendix C.** Be sure that the RS-232 jumper plug is removed if you are using current loop on a channel.

2.3 CONFIGURATION INSTRUCTIONS

Two configuration worksheets are provided in **Figures 2-4a** and **2-4b.** A completed sample of the worksheet is shown in **Figure 2-5a** and **2-5b.** We suggest that you photocopy the blank sheets and use them to select the settings for your switches. Refer to **Section 2.2** for details on the choices shown.

The worksheet can be useful later in referring to DIP switch settings once the main circuit board is concealed inside the card cage.

On the first page of the worksheet, (Sheet A, DIP Switch Settings), the right-hand column shows a stylized representation of switch packs SW2 through SW5, so that you can record the switch positions required for your application, then use the diagram to set the switches on the board.

The second page of the worksheet (sheet B, Chassis Panel Configuration), is for you to use in assigning the interface types of your eight or 16 ports. Each port can be one of these types:

FT	RS-232 For Terminal (MTI emulates modem)
FM	RS-232 For Modem (MTI emulates terminal)
FC	Current Loop For Terminal

Check the appropriate box on the worksheet for each port, and refer to **Appendix C** for jumper installation instructions.

NOTE: If you use current loop, the current loop option must be installed on the board at the factory. You must also configure the current loop switches on the chassis panel. See **Appendix C.**

2.3.1 Example configuration

Suppose you have an MTI-850/1650A and want the following:

-- All of the channels but channel 0 are to be RS-232C, with the MTI-1600A talking to terminals.

-- Channel 0 will be a current loop interface.

-- The MTI will respond to 16-bit address 01F0 hexadecimal at interrupt level zero.

-- Default characteristics are:

-- 300 baud
-- No parity needed
-- Characters seven bits long with one stop bit.

To use the MTI-850/1650A as described above, you would configure your switches as shown in the sample worksheets. (Also, the Port 0 current loop switch pack would need to be set, using **Appendix C.)**

2.4 INSTALLATION INSTRUCTIONS

After you have set the DIP switches on the main circuit board, and installed jumpers on the USART board if needed, you are ready to install the board.

NOTE: Compliance with FCC regulations may require the use of shielded cable for the external serial cables. Refer to **Appendix F.**

2.4.1 Installing MTI-850/1650A

The installation process will be easiest if you follow the steps suggested below.

1) Attach ribbon cables to main circuit board

Slide the 50-pin socket on the ribbon cable onto the header on the main circuit board labeled J1. See **Figure 1-1** if necessary. The arrow on the cable must be aligned with the arrow on the socket. Slide the Molex connector on the 5-conductor power cable onto the 5-pin header, aligning it in the same manner.

The connectors will slide onto the headers in only one way. Connectors are "keyed" to prevent you from mounting them backward; the arrows on the header and the connector must be lined up. If the connector resists sliding onto the header, **STOP!** Turn the connector around (rotate it 180 degrees) and try again.

2) Seat main circuit board in Multibus

Slide the board, with 50-conductor ribbon cable and 5-conductor power cable attached, into its proper Multibus slot and seat it firmly.

Make sure that the diagnostic toggle switch on the edge of the board is in the NORMAL position.

3) Install chassis panel

Mount the chassis panel on the 19-inch rack with four machine screws (not provided), being careful not to pinch the internal ribbon and power cables in the process.

Connect your serial device cables to the eight or 16 DB-25 (standard RS-232C) connectors on the chassis panel, labeled port 0 through either Port 7 (MTI-850A) or Port 15 (MTI-1650A).

The board should now be firmly seated in the Multibus, all cables connected, and the chassis panel attached.

NOTE: The MTI-850/1650A is a master device, and as such uses the BPRN and BPRO signals of the Multibus to take control of the bus during DMA cycles.

This means that if other master devices are in the system, you must decide whether you are using a serial or parallel priority resolution technique, and mount all of the master devices with a properly wired backplane.

Refer to the "Intel Multibus Specification" (Order Number: 9800683-04) or the "Guide to Configuring Multibus-Based Systems" (Order Number: 144788-001). Both are available from:

Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

2.4.2 Removal and reinstallation

To remove the entire unit for any reason (i.e., reconfiguration, shipping or storage), first **TURN OFF POWER TO THE SYSTEM.**

Then slide the main circuit board out part way and disconnect the 50-conductor ribbon cable and the 5-conductor power cables from the board.

Then remove the chassis panel by unscrewing the four machine screws holding it onto the 19-inch rack.

NOTE: If there are master (DMA) devices below the MTI-850/1650A in the Multibus, you may have to do one of the following to operate the system with the MTI absent:

-- Install additional boards in the two vacant slots (or one vacant slot, if the USART board was not occupying a Multibus position);

-- Move all of the boards below up two slots each (or one slot if the USART board was externally mounted);

-- Rewire the Multibus backplane.

2.5 NORMAL OPERATION

On power-up, the MTI-850/1650A will perform a "self-test" operation, blinking the eight LEDs on the edge of the main circuit board in a variety of patterns.

During normal operation the LEDs will blink successively, two at a time, like "chaser" lights on a theater marquee, except that the pattern will move back and forth instead of in one direction only. (Be sure that the diagnostic toggle switch on the edge of the main circuit board is in the NORMAL position.)

The speed of the back-and-forth motion will indicate the device loading; the slower the lights, the greater the flow of data. If the lights should ever freeze in a single pattern other than a temporary halt in the back-and-forth pattern, this indicates a problem. Refer to **Section 5** (Troubleshooting) for assistance.

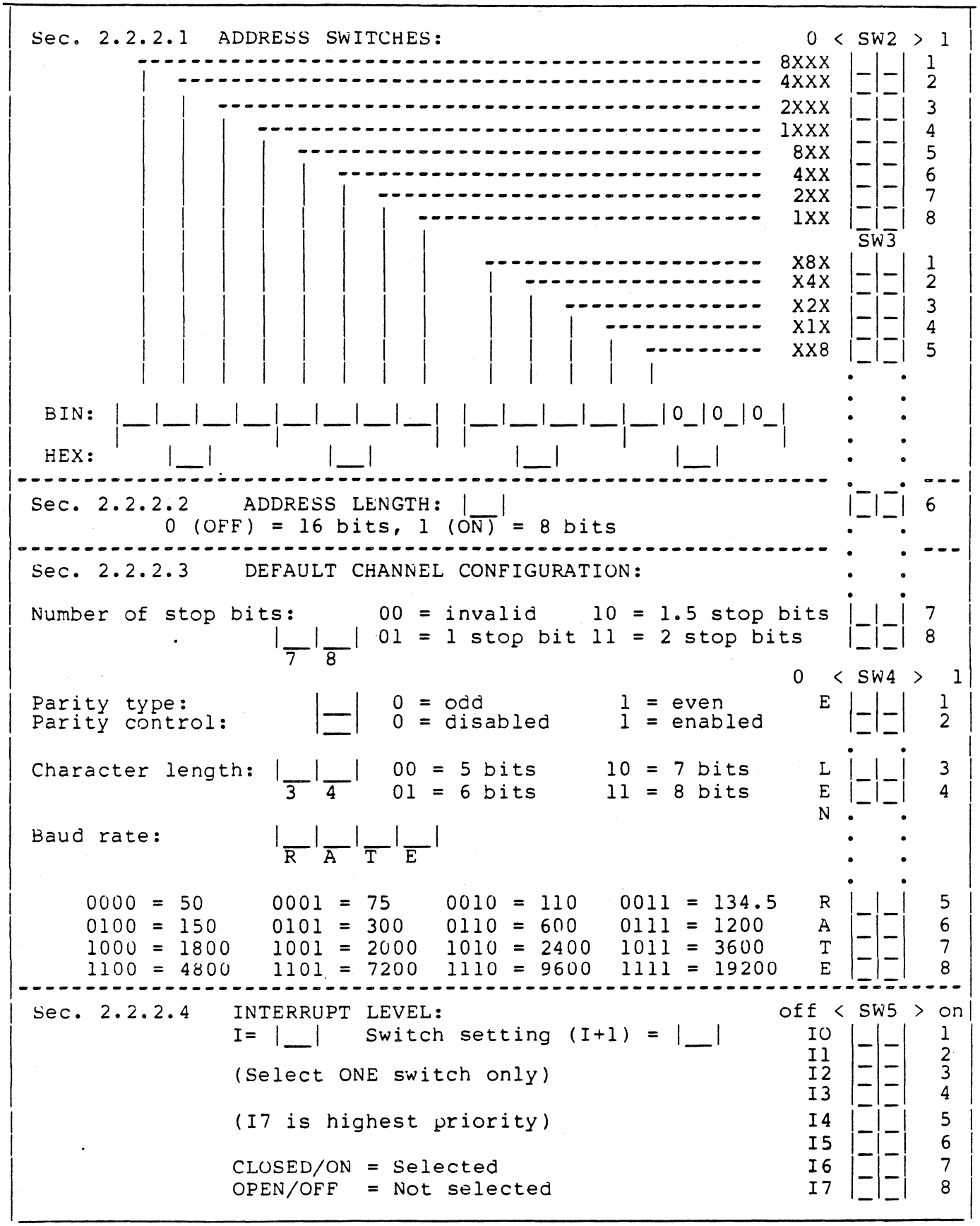


Figure 2-4a Configuration worksheet A - DIP switch settings

PORT 0	PORT 1	PORT 2	PORT 3	PORT 4	PORT 5	PORT 6	PORT 7
[ ] FT	[ ] FT	[ ] FT	[ ] FT	[ ] FT	[ ] FT	[ ] FT	[ ] FT
[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC
[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM

PORT 8	PORT 9	PORT 10	PORT 11	PORT 12	PORT 13	PORT 14	PORT 15
[ ] FT	[ ] FT	[ ] FT	[ ] FT	[ ] FT	[ ] FT	[ ] FT	[ ] FT
[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC
[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM

Check one box per port.

Each port can be one of the three types listed below.

Refer to text for further explanation.

FT = RS-232 for terminal (MTI emulates modem)
FM = RS-232 for modem (MTI emulates terminal)
FC = Current loop for terminal

Figure 2-4b Configuration worksheet B - Chassis panel settings

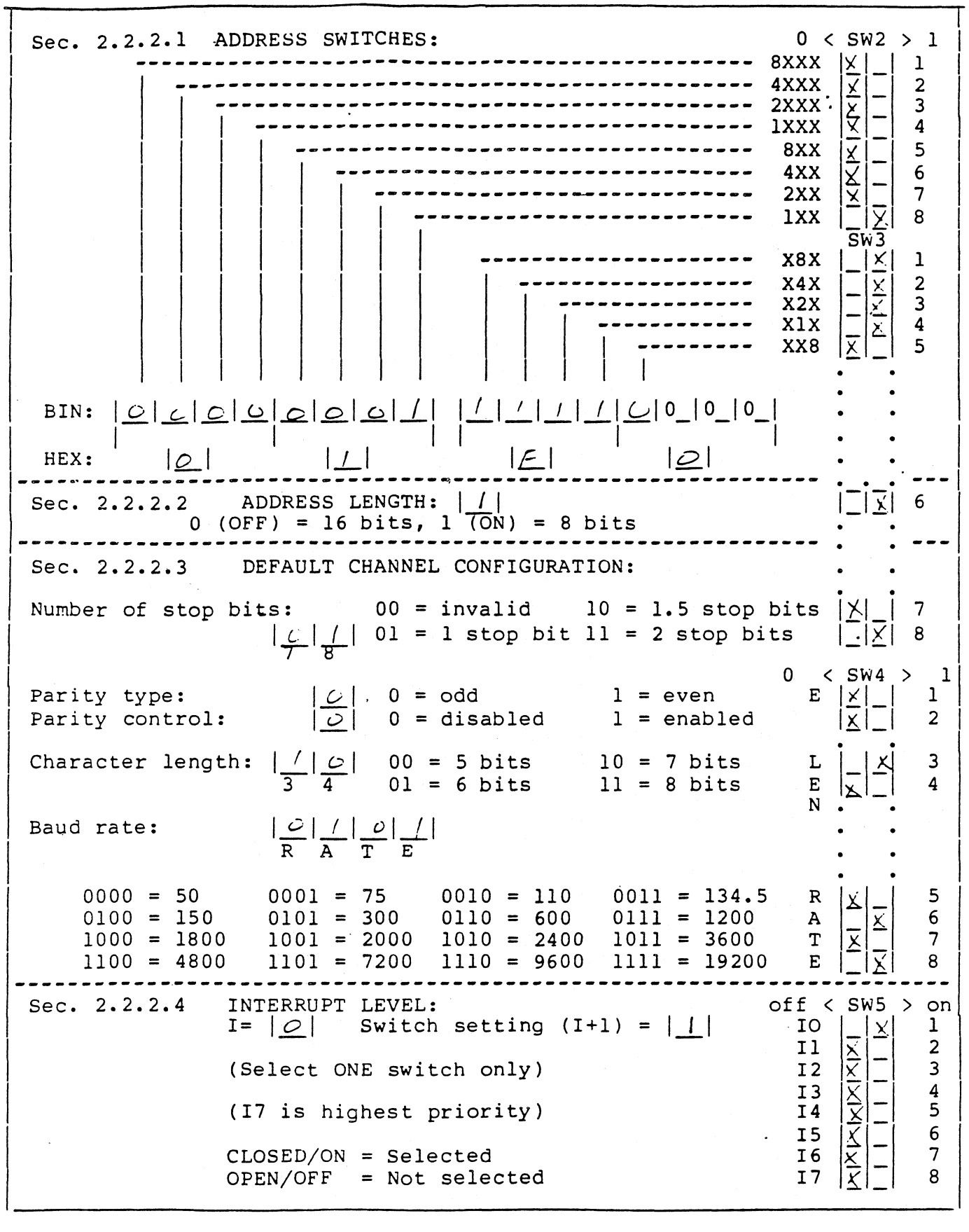


Figure 2-5a Sample worksheet A - DIP switch settings

PORT 0	PORT 1	PORT 2	PORT 3	PORT 4	PORT 5	PORT 6	PORT 7
[ ] FT	[✓] FT	[✓] FT	[ ] FT	[ ] FT	[ ] FT	[ ] FT	[ ] FT
[✓] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC
[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM

PORT 8	PORT 9	PORT 10	PORT 11	PORT 12	PORT 13	PORT 14	PORT 15
[ ] FT	[ ] FT	[ ] FT	[ ] FT	[ ] FT	[ ] FT	[ ] FT	[ ] FT
[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC	[ ] FC
[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM	[ ] FM

Check one box per port.
Each port can be one of the three types listed below.
Refer to text for further explanation.

FT = RS-232 for terminal (MTI emulates modem)
FM = RS-232 for modem (MTI emulates terminal)
FC = Current loop for terminal

Figure 2-5b Sample worksheet B - Chassis panel settings

SECTION 3: PROGRAMMING

Section 3: Programming

This section will be useful to you if you are writing "low-level" software to communicate with serial devices through the MTI-850/1650A, such as a "driver" program or a stand-alone utility. If this low-level software is correctly written, applications programs on the host computer will not need to be concerned with the details of the MTI-850/1650A's operations, but will instead be able to access any device connected to the MTI-850/1650A in a simple, "transparent" and well-defined manner.

NOTE: For some of the channel configuration options defined below, you may find it useful to review the communications concepts in **Appendix A.** If you are referring to this section to refresh your memory about the software commands, you may find **Section 3.7** (Software Reference Summary) useful.

In this section, we use the following terms in these precise ways:

Port refers to one of the eight (for the MTI-850A) or 16 (for the MTI-1650A) input/output pathways.

Channel refers to only the input or output portion of a port, so there are 16 (for the MTI-850A) or 32 (for the MTI-1650A) separately configurable channels.

3.1 HOST INTERFACE

Before we look at an overview of the flow of data through the system, we need to specify the direct interface the HOST has to the MTI-850/1650A.

3.1.1 MTI input/output addresses

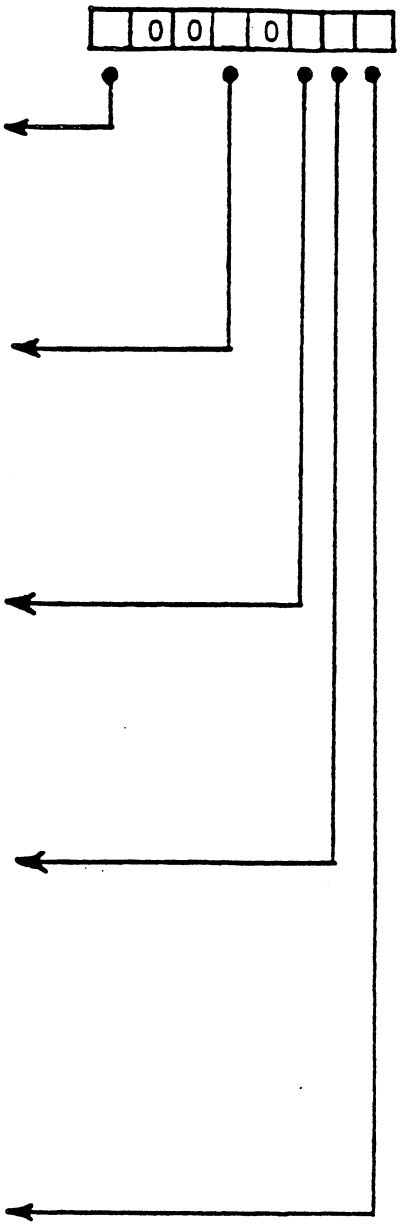
The MTI-850/1650A communicates with the HOST through four consecutive I/O addresses. The table that follows shows the uses of these addresses.

MTI input/output addresses

Address	Direction (from host CPU)	Purpose
base + 00	OUT	Command (and transmit data) FIFO
	IN	Response (and receive data) FIFO
base + 01	OUT	Interrupt enable byte
	IN	MTI status byte (see Figure 3-1)
base + 02	OUT	Set execute
	IN	Clear response available
base + 03	OUT	Reset board
	IN	Clear timer bit

You use address select switches on the MTI's main circuit board to map these addresses into the HOST bus address space at any 8-location boundary, so the MTI responds to eight consecutive addresses.

Address base +04 through base +07 are reserved by Systech for future additions to the MTI. The HOST should not access these addresses.



VALID DATA

The MTI will SET this bit whenever the Response FIFO contains at least one byte of data for the host.

TIMER

The MTI SETS this bit at each 'tick' of its interval timer. The MTI clears this bit when the HOST reads from the Clear Timer address.

COMMAND ERROR

The MTI will SET this bit if the HOST issues an invalid command. This MTI clears this bit after the HOST issues a Read Error Code command.

RESPONSE AVAILABLE

The MTI will SET this bit whenever it has loaded a complete response (which may consist of several bytes) into the Response FIFO for the HOST. The MTI clears this bit after the HOST reads from the Clear Response Available address.

MTI READY

The MTI will SET this bit when it is ready to accept a command from the HOST. The MTI clears this bit when the HOST writes to the Command FIFO address.

Figure 3-1 MTI status byte

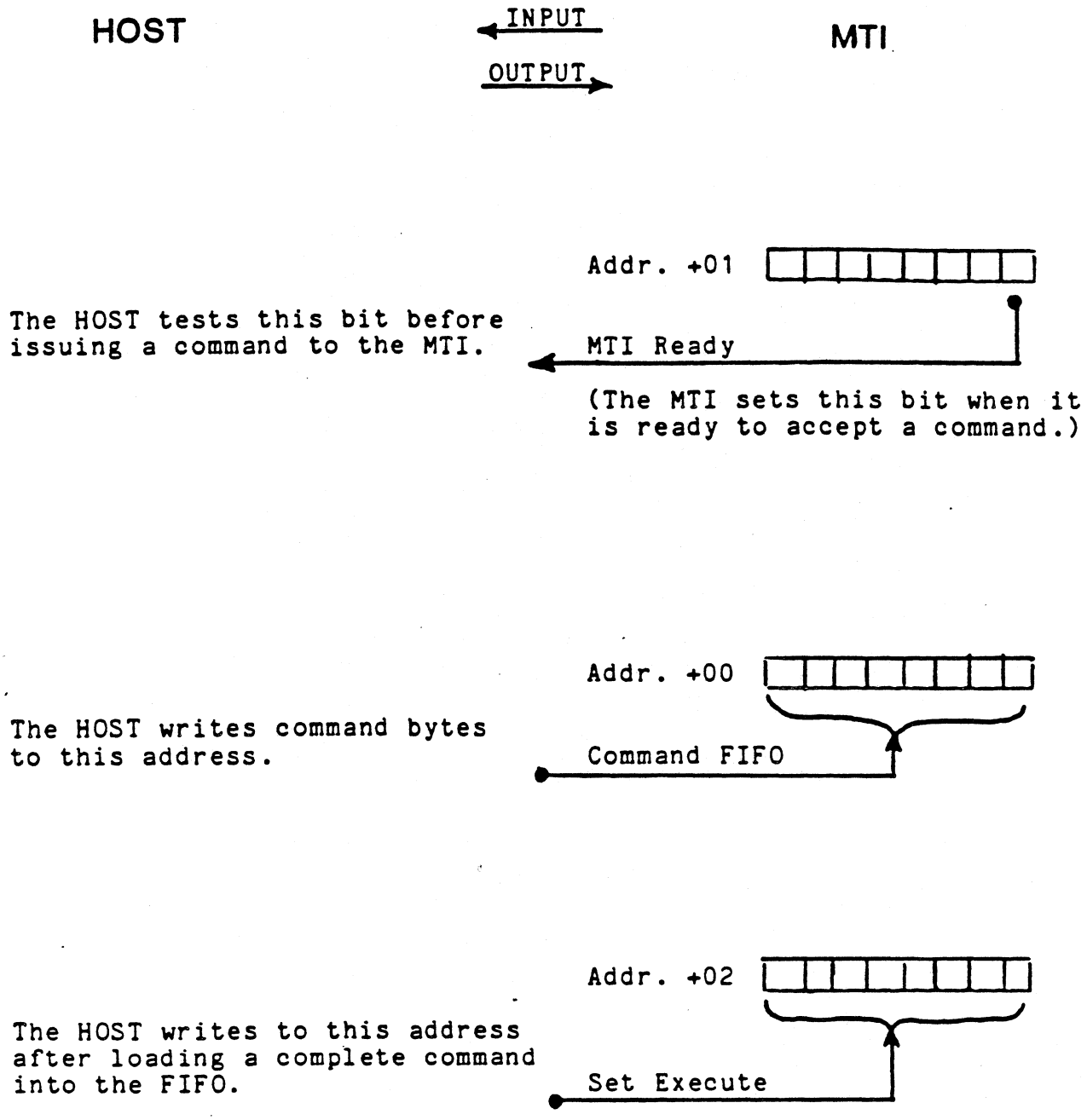


Figure 3-2 HOST to MTI command handshaking

3.1.2 Command handshaking

When the MTI is ready to accept a command from the HOST, it will set bit 0 (MTI Ready) of the MTI Status Byte, shown in **Figure 3-1.** When the host has assured itself that the MTI is able to accept a command by checking MTI Ready, it writes command bytes to the Command/Transmit data FIFO (address +00). (Writing to this address clears the MTI Ready status bit.)

In most cases, HOST commands to the MTI consist of several bytes, which the HOST loads sequentially into the FIFO. When it has written the entire command sequence to the FIFO, the HOST signals the MTI by writing to Set Execute (address +02). When the MTI is able to accept another HOST command, it will once again set MTI Ready, and the process can repeat (see **Figure 3-2**).

If the HOST should ever issue an invalid command to the MTI, the MTI will set bit 2 (Command Error) of the MTI Status Byte (address +01). This bit indicates that the MTI could not interpret the last command, and so ignored it, or else that an error condition occurred during command execution. The HOST must issue a Read Error Condition command (see **Section 3.4.5.5**) to clear this bit.

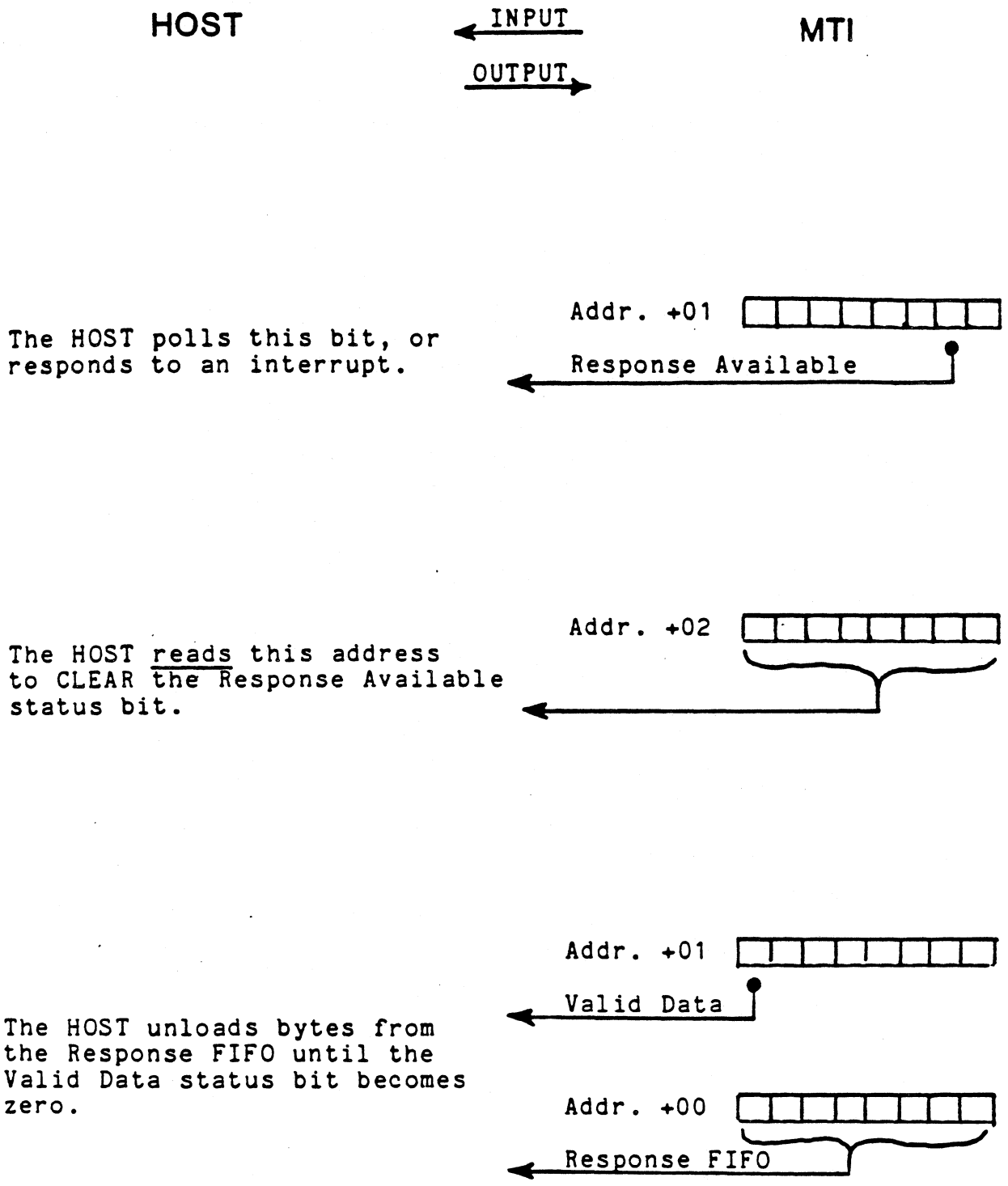


Figure 3-3 Response handshaking

3.1.3 Response handshaking

The MTI returns response or status information to the HOST through the Response FIFO. When the MTI has loaded a complete response message into the FIFO, the MTI will set the "Response Available" status bit in address +01. It is then the HOST's responsibility to unload the response message from the FIFO, and take appropriate action.

Before the HOST unloads each byte from the FIFO, it should check the Valid Data status bit. This bit, when set, indicates that another byte is available in the FIFO. If the Valid Data bit is zero, it indicates that the FIFO is empty, and must not be read by the HOST. The Valid Data bit will go zero after the HOST has unloaded the last byte from the FIFO.

In order to reduce the number of interrupts that must be processed by the host, the MTI returns responses in an asynchronous fashion. This means that the MTI may be loading the next response into the FIFO during the time that the host is reading a response from the FIFO. To avoid race conditions in response processing, the following algorithm is suggested. This algorithm assumes that the MTI is running in an interrupt-driven environment, and that the Response Available interrupt is un-masked.

```
 1   Enter response interrupt service routine
 2        "i" is a static integer variable
 3        "n" is a static integer variable
 4        "response" is a static array of 7 bytes

 5        Clear Response Available
 6        while (Valid Data is TRUE)
 7             response[i] = Response FIFO byte
 8             i = i + 1
 9             if (i is equal to 1)  /# We have the first byte      #/
10                  n = number of bytes in complete response

11             if (i is equal to n)  /# We have a complete response #/

12                  i = 0
13                  Process the response in "response"
14        end while
15   Exit response interrupt service routine
```

In the program fragment above, response bytes are accumulated in the array "response." When a complete response has been collected, code at line 13 takes some appropriate action, depending on the type of the response. Each type of response that the MTI

can generate contains a specific number of bytes. This number of bytes can be calculated as soon as the first byte of a response (which gives its type) has been unloaded from the response FIFO. Each time the first byte of a response is unloaded from the FIFO, the variable "n" is set to the number of bytes contained in the complete response (line 10). This can be done with a simple table lookup. The variable "i" counts bytes unloaded from the FIFO. When its value becomes equal to that of "n," one complete response has been collected in "response."

Note that the unloading of one response may be divided across two interrupt services. This will happen when the first few bytes of a response are read during the interrupt service associated with a previous response. It is important that the variables "i" and "n" and the array "response" be declared as static, so that their values will be maintained across calls to the interrupt service. When many responses are being generated by the MTI, several response packets can stack up in the response FIFO between host interrupt services. When this is the case, several responses will be handled on each MTI interrupt. This reduces the interrupt service overhead when system loading is heaviest.

3.1.4 Interrupts

You may enable four of the MTI's status bits to generate a hardware interrupt when they are set: MTI Ready, Response Available, Command Error, and Timer. The HOST enables any or all of these interrupts by setting bits in the Interrupt Enable Byte (address +01 OUT) which correspond to the status bits in the MTI Status Byte (address +01 IN). The HOST clears an interrupt by clearing its corresponding status bit. **(Figure 3-1** summarizes the operation of the status bits.)

3.1.5 Timer

The MTI includes an independent timer so that the HOST may program the MTI to generate periodic HOST interrupts. To use the timer, the HOST first selects an appropriate interval, using the Configure Timer command (see **Section 3.4.1.1**). It then sets the timer interrupt enable bit (bit 4) in the Interrupt Enable Byte (address +01). At each timer interval thereafter, the MTI will set the timer status bit and interrupt the HOST. The MTI clears the status bit and corresponding interrupt when the HOST reads from address +03 (Clear Timer Bit).

If an application does not use the timer, the HOST should disable timer interrupts to avoid having it generating spurious interrupts.

The MTI's timer is not crystal controlled, and is not intended for high accuracy applications.

3.2 OVERVIEW OF OPERATIONS

This section describes the movement of data between the HOST and serial devices, through the MTI. **Section 3.4** describes the commands that the HOST uses to control data movement.

3.2.1 General

When the MTI is powered up, it configures all of its channels as asynchronous, reading default parameters (baud rate, stop bits, etc.) from DIP switches on the MTI board. If the HOST wishes to change the channel configuration, it must issue an appropriate "Configure Channel" command (see **Sections 3.4.1.5** and **3.4.1.7**).

The MTI has its own on-board buffer memory (16K bytes). On power-up, the MTI allocates this buffer memory among the serial channels. In an 8-channel system, 1024 bytes will be allocated for each transmitter, and each receiver. In a 16-channel system, each will get 512 bytes. The HOST may change the buffer memory allocation with the Configure Buffer Sizes command (**Section 3.4.1.4**). If this command is used, it must be issued before any channel configuration commands are issued to the MTI.

3.2.2 Output

The MTI operates in two distinct output modes: single character and block. The HOST controls the output mode by issuing particular output commands. There are two output commands, one for character and one for block output. (See **Sections 3.4.2** and **3.4.3,** respectively). HOST software may freely mix output in the two modes.

If the HOST is going to use the block mode on a channel, it must issue a Configure Output Channel command (**Section 3.4.1.7**) for the channel, with the "Block Output Permitted" bit set. If the HOST attempts block output before issuing such a command, a command error will result.

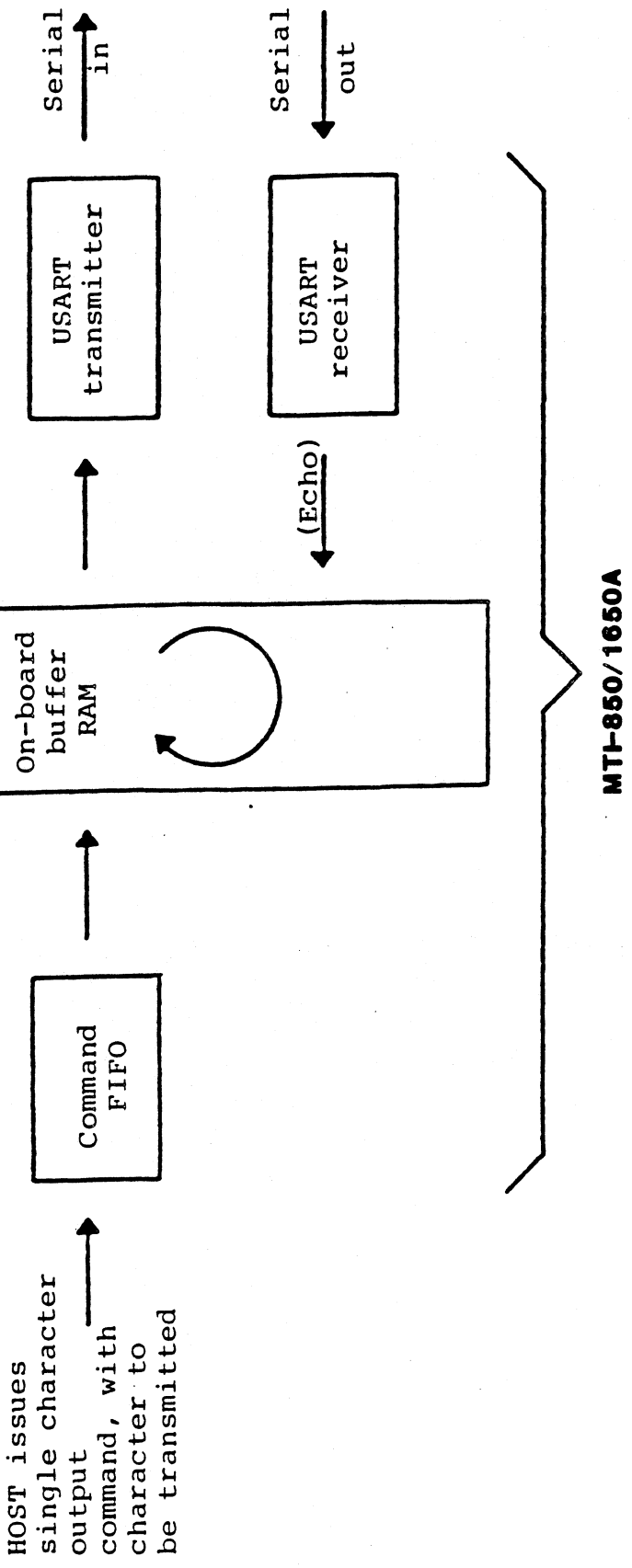


Figure 3-4 Single character output (one channel)

3.2.2.1 Single character output process

The single character output process is represented visually in **Figure 3-4.** This is the sequence of operations:

1) HOST waits for MTI Ready to be a "1", and then writes a Single Character Output command, with a character to transmit, into the Command FIFO.

2) When the MTI processes the HOST command:

a) If room exists in the channel's transmit buffer (on board the MTI), then:

i) The character is entered into the transmit buffer.

ii) The channel's transmitter service interrupt is enabled.

b) If no room exists in the channel's transmit buffer, then:

i) The "Command Error" status bit is set.

NOTE: The HOST can ensure that space exists in the transmit buffer by issuing a Read USART Status Register command to the MTI (see **Section 3.4.4.1**). The USART transmitter will never appear ready to the HOST, unless there is space in the MTI's transmit buffer.

3) When the MTI has completed processing the command:

a) It sets MTI Ready.

b) When the channel's USART transmitter is ready, the MTI's transmitter service routine will pull characters from the transmit buffer (if no block output is in progress), and send them out over the serial channel.

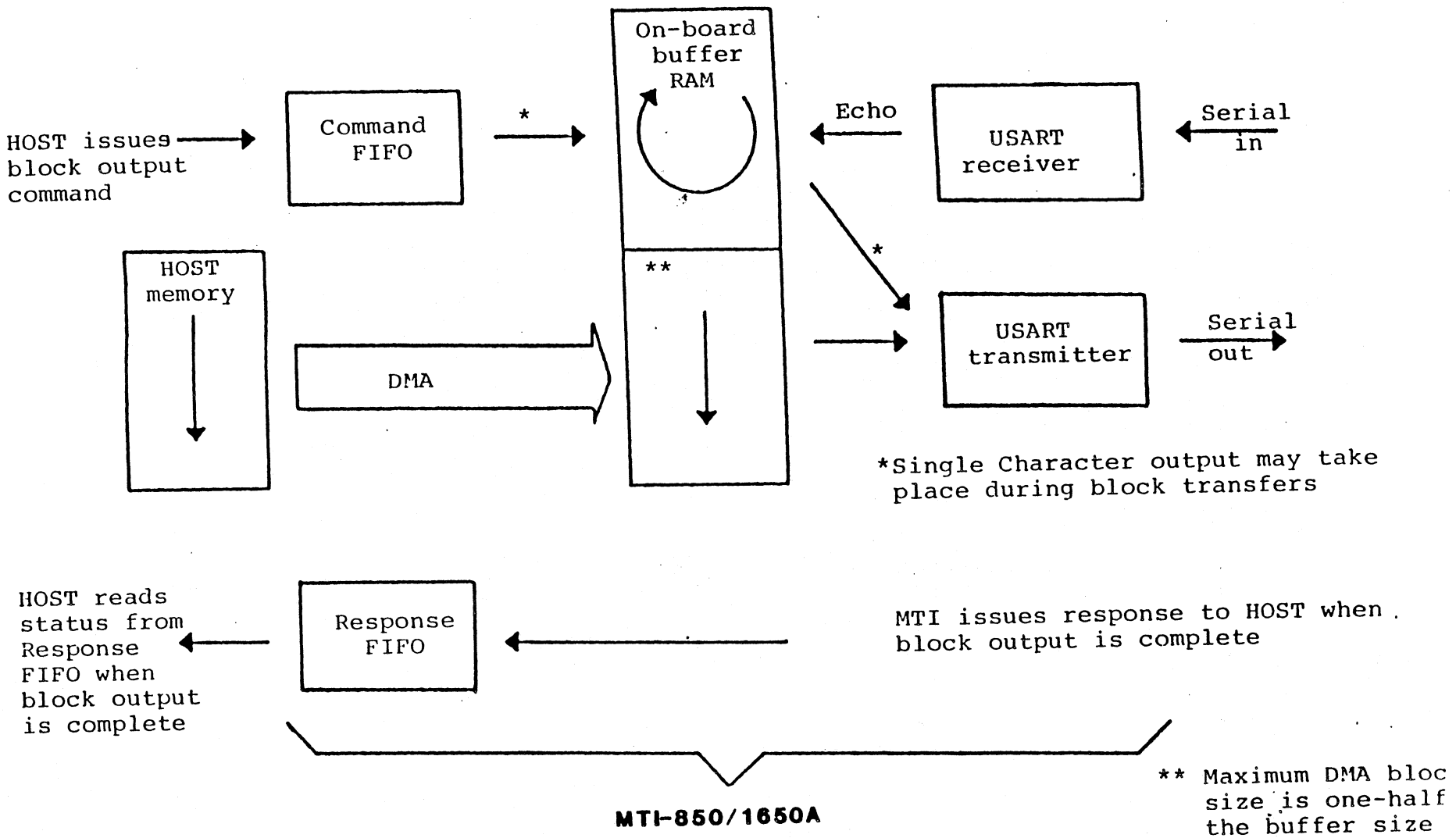


Figure 3-5 Block output (one channel)

3.2.2.2 Block output process

The block output process (shown in **Figure 3-5**) follows this sequence:

1) HOST waits for MTI Ready to be "1", and then writes a Block Output command into the Command FIFO.

2) When the MTI processes the command:

a) If no error conditions are present (below):

i) The data is transferred to the on-board buffer, via a Direct Memory Access (DMA) operation from HOST memory.

ii) The channel's transmitter interrupt service is enabled.

b) If any of the following conditions are present, the Command Error status bit is set:

i) The channel's Block Output Permitted bit has not been set.

ii) The specified character count exceeds one-half the configured buffer size.

iii) A block output is already in progress.

3) When the MTI has finished processing the command:

a) It sets MTI Ready

b) Characters are transmitted over the serial channel, as the USART transmitter becomes ready.

4) When the buffer is empty:

a) The MTI loads a termination message into the Response FIFO.

b) The MTI sets the Response Available bit in the MTI Status byte (bit 7 in address +01).

5) The HOST unloads the termination message, and takes whatever action is appropriate.

3.2.3 Input

The MTI operates in two distinct input modes: single character, and block. The input mode is established by the particular input commands issued by the HOST. There are two input commands, one for character and one for block input (see **Sections 3.4.2.1** and **3.4.3.1,** respectively). Although input in the two modes may be freely intermixed, normally any particular channel will use one mode or the other exclusively.

Before the HOST may use a channel for block input, it must configure that channel with a Configure Input Channel command (see **Section 3.4.1.5**), setting that channel's Block Input Permitted bit. Failure to do so will result in a command error if a Block Input command is issued.

Single character input need not be enabled at the time the input channel is configured. Single character input can be enabled and disabled explicitly by the commands Enable and Disable Single Character Input (see **Sections 3.4.2.1** and **3.4.2.2**). If both single character and block modes of input are enabled, any characters received on the channel will first be used to satisfy a pending block input command. If none is pending, then characters will be returned through the response FIFO as single character input.

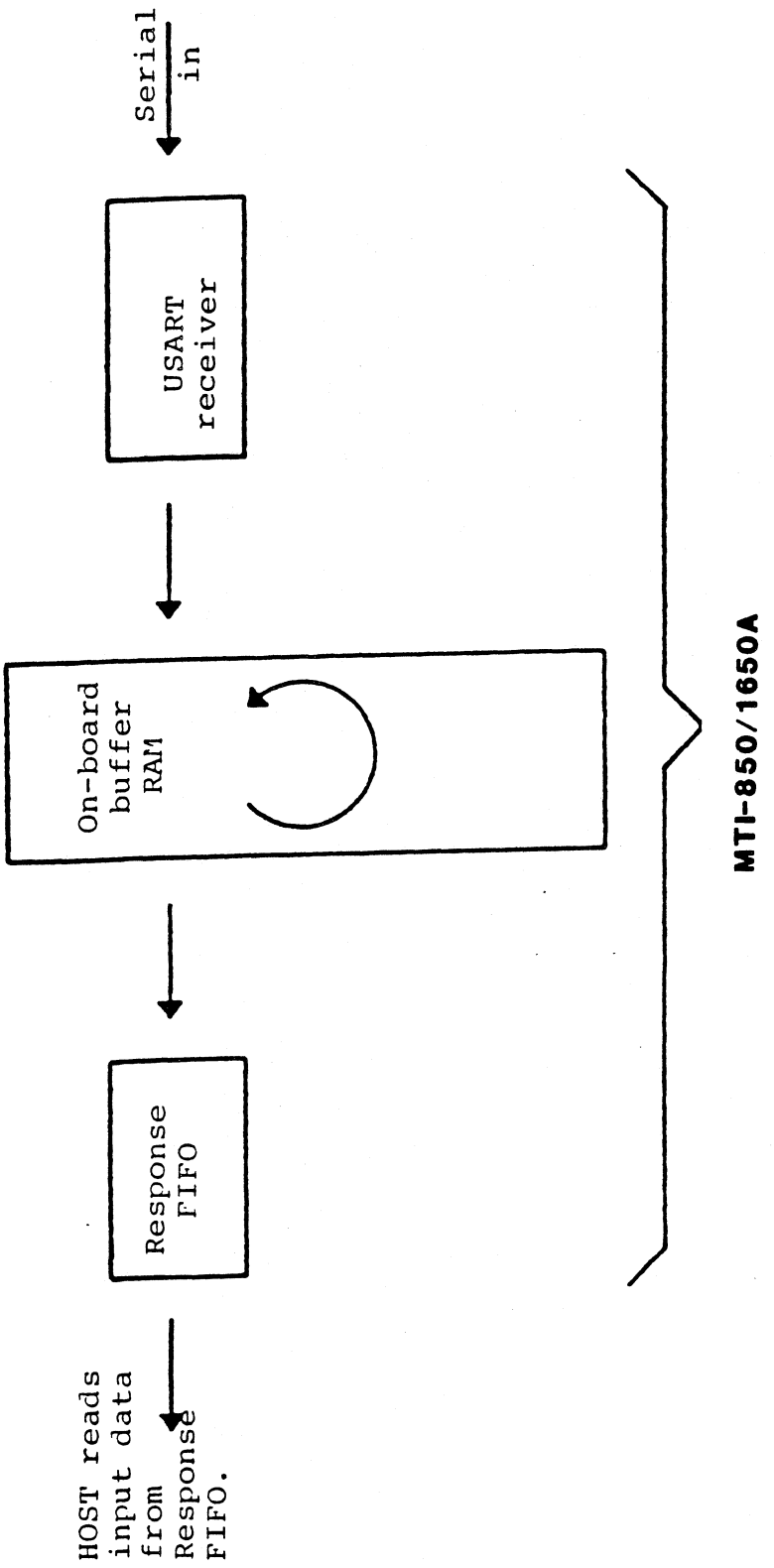


Figure 3-6 Single character input (one channel)

3.2.3.1 Single character input process

The sequence of actions for single character input (shown in **Figure 3-6**) is:

1) HOST enables single character input

2) Character arrives at USART receiver

3) Character and status are entered into MTI receiver buffer

4) When space is available in the Response FIFO:

   a) The MTI loads the character and channel status into the FIFO

   b) The MTI sets the Response Available status bit

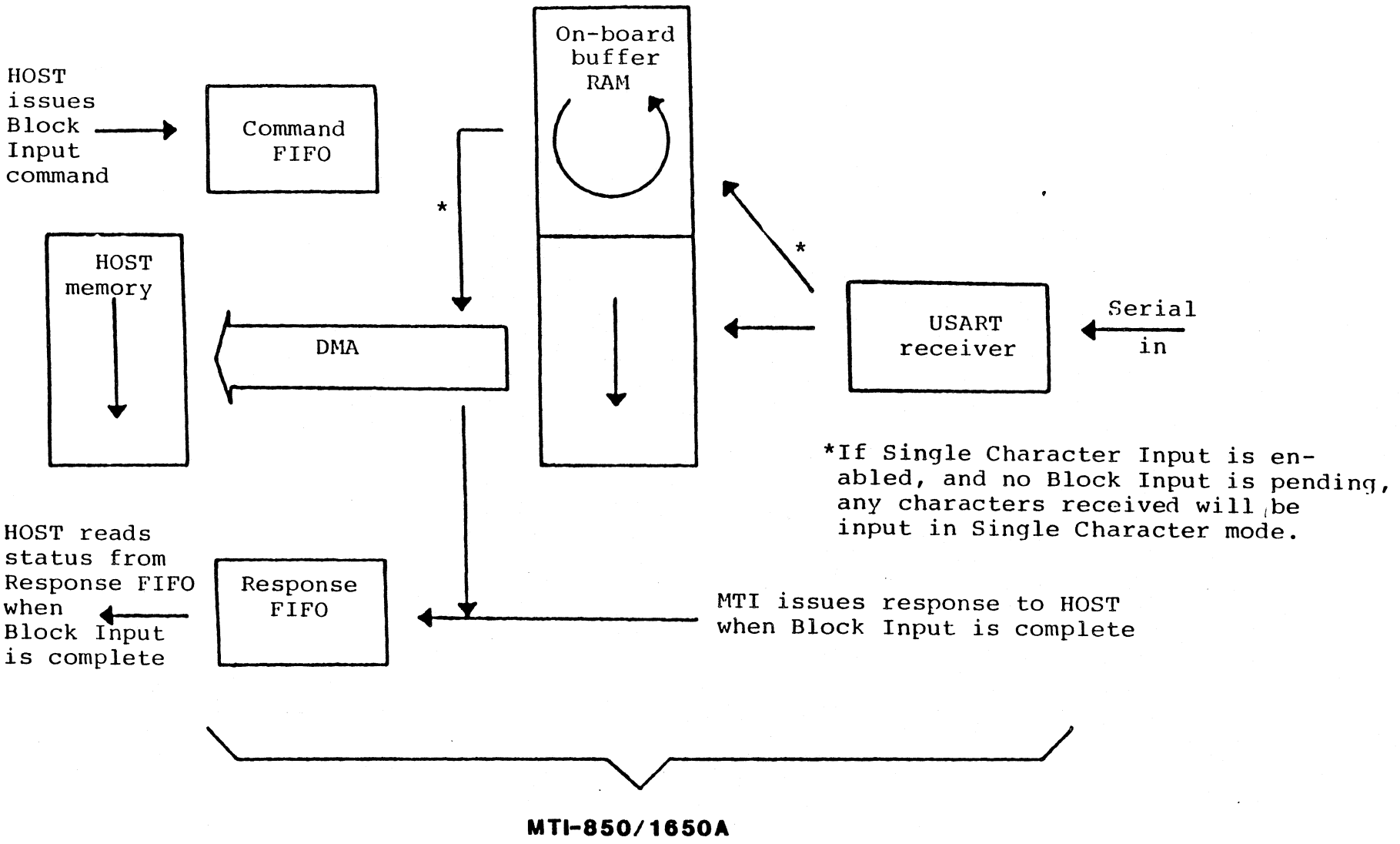


Figure 3-7 Block input (one channel)

3.2.3.2 Block input process

The block input process shown in **Figure 3-7** consists of these steps:

1) The HOST waits for MTI Ready status bit to be SET, and then issues a Block Input command to the MTI.

2) When the MTI processes the command:

a) If no error conditions (below) exist:

i) The MTI sets up internal tables for the channel.

b) If any of the following errors exist, the MTI sets the Command Error bit:

i) The Block Input Permitted bit for this channel is not set.

ii) The specified byte count exceeds one-half the configured buffer size.

iii) A Block Input command is already in progress.

3) The MTI sets the MTI Ready status bit.

4) When a character arrives at the receiver:

a) The character is loaded into the on-board buffer.

b) Termination conditions are checked.

c) If echo has been specified, the character is copied into the channel's transmit buffer.

5) When termination conditions are met:

a) The on-board buffer is copied, via DMA, to HOST memory.

b) When space is available in the Response FIFO, the MTI loads a response message into the FIFO.

c) The MTI sets the Response Available status bit.

3.3 PERFORMANCE FACTORS

In determining whether to use block or single character transfers for any given channel, take these factors into account:

1. For input and output, the MTI-850/1650A operates most efficiently in Block Mode.

2. You can minimize HOST operating system overhead if a request from an applications program to "input a line" or "output a message" can be passed directly to the MTI as a block I/O request.

3. Some applications may operate more efficiently with Character Mode transfers if they require immediate responses to many different "control" characters, such as editing command characters.

3.4 MTI COMMANDS

This section specifies the sequences of bytes that the HOST must load into the MTI Command FIFO. The commands are divided into five groups:

GROUP:	FUNCTION:
1	Configuration commands
2	Single characters transfer commands
3	Block transfer commands
4	USART control commands
5	Miscellaneous commands

NOTE: In the command formats below, all values are in binary unless otherwise indicated. A command summary reference chart appears at the end of this section.

3.4.1 Configuration commands

The HOST may use the configuration commands to configure the MTI before any data transfers take place. There are eight configuration commands.

NOTE: The Set Buffer Sizes and Configure Timer Interrupts commands are unique in that they apply to the MTI board as a whole, rather than to a specific channel.

3.4.1.1 Configure Timer command

This command sets the rate at which the MTI's interval timer generates HOST interrupts. If the timer is not to be used at all, the HOST should keep the timer's interrupt enable bit (in the Interrupt Enable byte) OFF at all times. The interval timer is not crystal controlled, and is NOT INTENDED AS A HIGH ACCURACY DEVICE.

Command

Byte 0: 0 1 1 1 r r r r Command byte

rrrr = nominal timer rate (in milliseconds):

0000	= 530	1000	= 14.72
0001	= 353	1001	= 13.22
0010	= 241	1010	= 11.04
0011	= 197	1011	= 7.36
0100	= 177	1100	= 5.52
0101	= 88.3	1101	= 3.68
0110	= 44.2	1110	= 2.76
0111	= 22.1	1111	= 1.34

Byte 1: 1 0 0 1 0 0 0 0 Second Command byte

3.4.1.2 Configure a Port Asynchronous command

This command sets up a channel for asynchronous communication. Bytes 2-4 of the command sequence program the mode and command registers of the USART associated with the given channel. The meanings of the possible bit values for these registers appear in **Figure 3-8, a and b.** Additional information on the USARTs is given in **Appendix J.** The bit fields applicable to asynchronous operation are shown below.

Command		
Byte 0:	0 1 1 1 n n n n	First command byte: nnnn is the port number
Byte 1:	0 0 0 0 0 0 0 0	Second command byte
Byte 2:	s s e p 1 1 c c	USART mode register 1: ss number of stop bits e even/odd parity p parity/no parity 11 character length cc clock rate divisor
Byte 3:	m m m m r r r r	USART mode register 2: mmmm Tx/Rx clock operation rrrr Baud rate
Byte 4:	e e R r b c D t	USART command register: ee echo/loop back control R RTS signal control r reset error flags b break control c enable receiver D DTR signal control t enable transmitter

3.4.1.3 Configure a Port Synchronous command

This command configures a port for synchronous communication. Bytes 2-7 are sent to the mode and command registers of the USART associated with the specified port. See **Figure 3-8, a and b** for the meanings of these registers. Additional information on the USARTs appears in **Appendix D.**

Command		
Byte 0:	0 1 1 1 n n n n	First command byte: nnnn is the port number
Byte 1:	0 0 0 1 0 0 0 0	Second command byte
Byte 2:	s t e p 1 1 0 0	USART mode register 1: s single/double SYN t transparent/normal SYN e even/odd parity p parity/no parity 11 character length
Byte 3:	m m m m r r r r	USART mode register 2: mmmm Tx/Rx clock operation rrrr baud rate
Byte 4:	s s R r b c D t	USART command register: ss SYN & DLE strip/normal R RTS signal control r reset error flags b break control c enable receiver D DTR signal control t enable transmitter
Byte 5:	c c c c c c c c	SYN1 character
Byte 6:	c c c c c c c c	SYN2 character
Byte 7:	c c c c c c c c	DLE character

3.4.1.4 Configure Buffer Sizes command

The MTI has 16K bytes of on-board buffer memory to allocate among the eight or 16 I/O ports. Normally, 1/2K (512 bytes) will be allocated to each input and output channel on an MTI-1650, or 1K (1024 bytes) on an MTI-850. The HOST may change this allocation with the Configure Buffer Sizes command. The HOST passes the address of a 64-byte table of buffer sizes to the MTI. This table contains a two-byte buffer length for each input and output channel. The order for the bytes is:

```
low  byte of Port 0 input  channel buffer length
high byte of Port 0 input  channel buffer length
low  byte of Port 0 output channel buffer length
high byte of Port 0 output channel buffer length
low  byte of Port 1 input  channel buffer length
  etc....
```

NOTE: There are constraints on the 32-buffer lengths specified in the table; the MTI will generate a command error if any of these constraints are violated:

1. The sum of all the buffer lengths must not exceed 16K (16,384 bytes).

2. Each buffer size must be a multiple of eight.

3. Each buffer must be at least 32 bytes long.

A special 32-byte buffer may be reserved for DMA operations associated with the Configure Termination Mask command by setting a bit in the second command byte (see section 3.4.1.6). When this feature is not used, the sizes of the remaining input/output buffers must not total more than 16,352 bytes (16K less 32).

NOTE: If this command is used, it must be issued after power-up, or a Master Reset, and prior to issuing any Configure Input Channel, Configure Output Channel, Enable Single Character Input, or Single Character Output commands. Failure to do so will result in a command error.

Command		
Byte 0:	0 1 1 1 0 0 0 0	First command byte
Byte 1:	1 0 0 0 0 0 0 R	Second command byte ("R" indicates "reserved buffer bit")
Byte 2:	1 1 1 1 1 1 1 1	Low byte
Byte 3:	m m m m m m m m	Medium byte
Byte 4:	h h h h h h h h	High byte of address of Buffer Length Table in HOST memory.

Note that if the "reserved buffer bit" is set, the sum of all buffer lengths must not exceed 16K less 32 (16352).

3.4.1.5 Configure Input Channel command

When a channel is to be used for input, the HOST must tell the MTI four things about how data transfers will work:

-- Shall block input be enabled?

-- Shall single character input be enabled?

-- Under what conditions shall the MTI end Block Input commands?

-- Shall the MTI automatically echo non-terminating input characters?

The HOST may use this command to enable single character input on the channel. If single character input is enabled, characters will be passed to the HOST through the Response FIFO. Note that single character input can also be controlled with separate commands (see **Sections 3.4.2.1** and **3.4.2.2**).

The MTI can end Block Input on any of three events. The MTI will always end input on "terminal count". (That is, when the number of characters received by the MTI reaches the number requested by the HOST). Input will also end if a transmission error is detected, or if the HOST issues an abort command. In addition, the MTI can be told to end input when it receives certain "termination" characters. Termination characters can be programmed as any non-printing ASCII character, or as a specified set of "enter" or "end-of-line" characters.

If both options are selected, any character which is either a non-printing ASCII, or in the specified list, will end the Block Input. Termination character sets may also be programmed with the Configure Termination Mask command (**Section 3.4.1.6).**

Finally, the HOST may specify that all non-terminating characters received be automatically echoed to the output. This allows most of the interaction with a user terminal to be handled without HOST overhead.

There are two additonal features in MTI input processing. The first is the "Continuous Block Input" mode. This input mode allows input blocks to be buffered by the MTI, even when no Block Input command is pending. The second feature is available in the normal block input mode. In normal block input, the host may specify a "trailer byte" count. The use of the trailer byte count is explained below.

If desired, both block input and single character input may be enabled simultaneously. If continuous block input is selected, then single character may not be enabled. When the continuous block input mode is selected, data received on the channel will be processed by the MTI as with normal block input. Input blocks will be held in the MTI's input buffer until a Block Input command has been issued by the host. At that time, one block will be transferred to the host (if a block is available). Additional input will continue to be processed and buffered by the MTI. As Block Input commands are issued by the host, blocks will be moved into the host's memory.

The MTI can end input blocks on any of three events. The MTI will always end input on "terminal count". (That is, when the number of characters received by the MTI reaches the number requested by the HOST). An input block will also end if a transmission error is detected, or if the HOST issues an abort command. In addition, the MTI can be told to end input when it receives certain "termination" characters. Termination characters can be programmed as any non-printing ASCII character, or as a specified set of "enter" or "end-of-line" characters.

If both options are selected, any character which is either a non-printing ASCII, or in the specified list, will end the input blocks. Termination character sets may also be programmed with the Configure Termination Mask command.

When the normal block input mode has been selected, an additional option is available when blocks are terminated. Normally, the termination character is not included in the block of data moved into the host's buffer (although it is returned in the "Block Input Complete" response). If desired, a "trailer byte" count may be specified in the Configure Input command. This count gives the number of bytes past the normal end of the block to include in the data block. If a trailer count of one is given, the termination character itself will be included in the data block. If a count of two is given, the termination character, and the first following character will be included. The maximum trailer count is seven. Note that Block Input commands should specify large enough buffers to include the trailing bytes if this feature is used.

Finally, the HOST may specify that all non-terminating characters received be automatically echoed to the output. This allows most of the interaction with a user terminal to be handled without HOST overhead. Note that the auto-echo mode selected with this command will echo characters received as single-character input, as well as non-terminating characters received as normal or continuous block input.

Configure Input Channel:

Command		
Byte 0:	0 1 1 1 n n n n	First command byte: nnnn is the port number
Byte 1:	0 1 0 0 C t t t	Second command byte: C Set to 1 to select continuous block input mode ttt Trailer byte count
Byte 2:	A T T T e s E B	Transfer modes: A set to 1 to end input blocks on ASCII non-printables (controls) TTT termination character list length (000 if no list) e set to 1 to automatically echo non-termination characters s set to 1 to implement control-S/control-Q protocol E set to 1 to enable single character input B set to 1 to permit block input. (If 'C' above is set, then this bit must be set as well)
Byte 3:	c c c c c c c c	Termination character list: This list should contain as many characters as specified by 'ttt' above. If 'ttt' is zero, then these bytes need not be sent by the HOST.
Byte 4:	c c c c c c c c	
Byte 5:	c c c c c c c c	
Byte 6:	c c c c c c c c	
Byte 7:	c c c c c c c c	
Byte 8:	c c c c c c c c	
Byte 9:	c c c c c c c c	

3.4.1.6 Configure Termination Mask command

As an alternative to specifying Block Input termination either as all non-printables or as a list of up to seven specific characters, input termination characters can be specified in a 32-byte mask. Each of the 256 bits in this 32-byte mask corresponds to a possible 8-bit character received during Block Input. A zero bit indicates that the corresponding character is non-terminating. A one bit indicates a termination character.

When this command is executed by the MTI, the 32-byte termination mask is moved into the MTI's local memory (via DMA) for processing. If a buffer has been reserved for this purpose with the Configure Buffer Sizes command (see 3.4.1.4), then the reserved buffer will be used for this DMA operation. In this case, there is no restriction on when a Configure Termination Mask command may be issued. If a DMA buffer was not reserved, then the channel's block input buffer must be used to hold the termination during configuration. When this is the case, the Configure Termination Mask command must not be issued while a Block Input is in progress.

Normally, no response is generated by this command. If desired, however, the HOST may request that a response be returned after completion of the command, by setting a bit in the second command byte.

NOTE: Termination characters specified by this command override any specified in the "Configure Input Channel" command. The 32-byte mask is loaded from HOST memory. Bit 7 (MSB) of the first byte corresponds to ASCII NULL (00h). Bit 0 (LSB) of the last byte corresponds to value FFh.

Command

Byte 0:	0 1 1 1 n n n n	First command byte nnnn is the port number
Byte 1:	0 1 0 1 0 0 0 r	Second command byte r Set to 1 to request response Set to 0 for no response
Byte 2:	1 1 1 1 1 1 1 1	Low byte
Byte 3:	m m m m m m m m	Medium byte
Byte 4:	h h h h h h h h	High byte of address of 32 byte mask in HOST memory

Response

Byte 0:	0 1 0 1 n n n n	Response code: nnnn is the port number

3.4.1.7 Configure Output Channel command

Before a channel may be used for block output operations, the HOST must tell the MTI whether or not to permit "Block Output" commands.

Command		
Byte 0:	0 1 1 1 n n n n	First command byte: nnnn is the port number
Byte 1:	0 1 1 0 0 0 0 0	Second command byte
Byte 2:	0 0 0 0 0 0 0 B	Transfer Modes: B set to 1 to permit Block Output commands

---

3.4.1.8 Configure Modem Status command

The HOST may monitor changes in the modem signals (DSR and DCD) by enabling modem status. When enabled, a response will be returned to the HOST if either the DSR or DCD changes state. The response message will identify the channel affected, and show the new status for that channel.

Command

Byte 0:	0 1 1 1 n n n n	First command byte: nnnn is the port number
Byte 1:	0 1 1 1 0 0 0 0	Second command byte
Byte 2:	0 0 0 0 0 0 0 n	Third command byte: n = 1 notify of changes in modem status, via a response message n = 0 do not notify

Response (returned when modem signals change, if 'n' = 1)

Byte 0:	0 1 1 1 n n n n	Response code: nnnn is the port number
Byte 1:	D C F o p c r t	Channel status (See Figure 3-8-b)

3.4.1.9 Configure Modes Command

The Configure Modes command offers a way to easily change a channel's operating modes, without issuing a Configure Channel command. Auto-echo, and CTRL-S/CTRL-Q flow control may be controlled with this command. Note that finer control over auto-echo is available through this command than with the Configure Input command. Echo can be enabled on single character input only, block input only, or both.

Command		
Byte 0:	0 1 1 1 n n n n	First command byte: nnnn is the port number
Byte 1:	0 0 1 0 0 f b e	Second command byte

b Set to 1 to echo block input characters only.

e Set to 1 to echo both single-character input and block input characters. (Same as Configure Input command "e" bit)

Set both to zero to disable all auto-echo.

f Set to 0 to turn CTRL S/CTRL Q flow control off.

Set to 1 to turn CTRL S/CTRL Q

3.4.2 Single Character Transfers

The MTI operates most efficiently when it can transfer blocks of data to and from HOST memory. If necessary, however, single character transfers can be made through the MTI.

3.4.2.1 Enable Single Character Input command

This command requests that the MTI return characters from the specified channel through the Response FIFO. Channel status will be returned with each character. Any time a Block Input command is pending, and Single Character Input is enabled, characters received on the channel will be first used to satisfy the Block Input.

Command

Byte 0:	0 0 0 0 n n n n	Command byte: nnnn is the port number

Response (issued by the MTI when character is received
-------- on the channel)

Byte 0:	0 0 0 0 n n n n	Response code: nnnn is the port number
Byte 1:	D C f o p c r t	Channel status (see Figure 3-8-b)
Byte 2:	d d d d d d d d	Input data

3.4.2.2 Disable Single Character Input command

This command will "turn off" single character input from the specified channel. It has no effect on block input.

Command		
Byte 0:	0 1 1 0 n n n n	Command byte: nnnn is the port number

3.4.2.3 Single Character Output command

This command requests that a single character be sent to a channel. The HOST passes the character to be sent to the MTI along with the command.

Command		
Byte 0:	0 1 0 0 n n n n	Command byte: nnnn is the port number
Byte 1:	d d d d d d d d	Data byte: This byte is sent out on port nnnn

3.4.3 Block Transfers

These commands are used to specify "DMA" transfers of blocks of data to or from HOST memory. The HOST tells the MTI where the data is, and how many bytes are to be transferred. The MTI will later notify the HOST when the transfer is complete.

3.4.3.1 Block Input command

This command requests the MTI to accept input data on a channel, and buffer it until one of four things happens:

-- Terminal Count is reached

-- A termination character is encountered (see **Sections 3.4.1.5** and **3.4.1.6).**

-- An Abort command is issued by HOST (See **Section 3.4.5).**

-- A transmission error occurs

When a Block Input command is terminated for any of these reasons, the HOST will be notified. A response will be returned to the HOST indicating the reason for the termination.

The Block Input command's characteristics are somewhat different when it is used with the Continuous Block Input mode. When the channel has been configured for continuous block input, one or more input blocks may already be buffered by the MTI at the time the Block Input command is issued. If this is the case, an input block will be moved into host memory immediately, and a response will be returned to the host. If no input blocks are present in the MTI's buffers when the Block Input command is issued, then the MTI will wait for a complete input block to be buffered.

Command

Byte 0:	1 0 0 0 n n n n	Command byte: nnnn is the port number
Byte 1:	l l l l l l l l	Low byte
Byte 2:	m m m m m m m m	Medium byte
Byte 3:	h h h h h h h h	High byte of data address in HOST memory. The MTI will load data into the HOST memory starting at this address.
Byte 4:	l l l l l l l l	Byte count: The MTI will accept at most this many bytes from the channel. This is the "terminal count"
Byte 5:	h h h h h h h h	

Note that the byte count must be less than or equal to the input buffer size less 4.

You can use the default input buffer sizes of:

- 512 for 16 channel controllers (MTI-1650s)
- 1024 for 8 channel controllers (MTI-850s)

or you can set the buffer size by using the Configure Buffer Size command (refer to section 3.4.1.4).

Block Input (continued):

Response (returned when input has ended)

Byte 0:	1 0 0 0 n n n n	Response code: nnnn is the port number
Byte 1:	D C f o p c r t	Channel status: (see Figure 3-8-b)
Byte 2:	0 0 T e 0 n c a	Termination cause: (mutually exclusive) T termination character encountered e a transmission error (framing, parity, overrun) occurred. n DMA specified from non-existent memory c terminal count reached a HOST aborted command
Byte 3:	l l l l l l l l	Low byte
Byte 4:	h h h h h h h h	High byte of number of characters received (exclusive of termination character, if any)
Byte 5:	d d d d d d d d	Termination character: If 'T' is set, then this byte contains the character that caused the termination.

3.4.3.2 Return Buffered Data command

This command requests that a pending Block Input be terminated. Any data currently buffered will be moved to the HOST's DMA buffer, and a Block Input response will be generated. The Block Input response will indicate that the Block Input command was aborted.

Note: The input channel must be configured for Continuous Block Input for this command to work. In normal block input, the same effect may be achieved with the Abort Input command (refer to section 3.4.6.1).

Command	
Byte 0:	0 0 1 1 n n n n

3.4.3.3 Block Output command

This command requests that a block of data be transmitted over the specified channel. When the transfer has ended, the MTI will send a response to the HOST, indicating how the transfer ended. A command error will be generated if Block Output is not enabled.

Command		
Byte 0:	1 1 0 0 n n n n	Command byte: nnnn is the port number
Byte 1:	l l l l l l l l	Low byte
Byte 2:	m m m m m m m m	Medium byte
Byte 3:	h h h h h h h h	High byte of data address in HOST memory. Data starting at this address will be transmitted.
Byte 4:	l l l l l l l l	Low byte
Byte 5:	h h h h h h h h	High byte of Byte Count. This many bytes will be transmitted.

Note that the byte count must be less than or equal to the output buffer size divided by two.

You can use the default input buffer sizes of

- 512 for 16 channel controllers (MTI-1650s)
- 1024 for 8 channel controllers (MTI-850s)

or you can set the buffer size by using the Configure Buffer Size command (refer to section 3.4.1.4).

Response (returned when output has ended)

Byte 0:	1 1 0 0 n n n n	Response code: nnnn is the port number
Byte 1:	D C f o p c r t	Channel status: (see Figure 3-8-b)
Byte 2:	0 0 0 0 0 n c a	Termination cause: (mutually exclusive) n DMA specified from non-existent memory c terminal count (successful transfer) a Host aborted command
Byte 3:	l l l l l l l l	Low byte
Byte 4:	h h h h h h h h	High byte of number of characters transferred.

MODE REGISTER 1 (MR 1)

MR17 MR16	MR15	MR14	MR13 MR12	MR11 MR10
Sync/Async	Parity Type	Parity Control	Character Length	Mode and Baud Rate Factor
Async: Stop Bit Length 00 = Invalid 01 = 1 stop bit 10 = 1½ stop bits 11 = 2 stop bits	0 = Odd 1 = Even	0 = Disabled 1 = Enabled	00 = 5 bits 01 = 6 bits 10 = 7 bits 11 = 8 bits	00 = Synchronous 1X rate 01 = Asynchronous 1X rate 10 = Asynchronous 16X rate 11 = Asynchronous 64X rate
Sync: Number of SYN char 0 = Double SYN 1 = Single SYN Sync: Transparency Control 0 = Normal 1 = Transparent				

NOTE
Baud rate factor in asynchronous applies only if external clock is selected. Factor is 16X if internal clock is selected. Mode must be selected (MR11, MR10) in any case.

MODE REGISTER 2 (MR2)

MR27–MR24											MR23–MR20
	TxC	RxC	Pin 9	Pin 25		TxC	RxC	Pin 9	Pin 25	Mode	Baud Rate Selection
0000	E	E	TxC	RxC	1000	E	E	XSYNC[1]	RxC/TxC	sync	
0001	E	I	TxC	1X	1001	E	I	TxC	BKDET	async	
0010	I	E	1X	RxC	1010	I	E	XSYNC[1]	RxC	sync	
0011	I	I	1X	1X	1011	I	I	1X	BKDET	async	See baud rates in table 1
0100	E	E	TxC	RxC	1100	E	E	XSYNC[1]	RxC/TxC	sync	
0101	E	I	TxC	16X	1101	E	I	TxC	BKDET	async	
0110	I	E	16X	RxC	1110	I	E	XSYNC[1]	RxC	sync	
0111	I	I	16X	16X	1111	I	I	16X	BKDET	async	

NOTES
1. When pin 9 is programmed as XSYNC input, SYN1, SYN1-SYN2, and DLE-SYN1 detection is disabled.

E = External clock
I = Internal clock (BRG)
1X and 16X are clock outputs

2661-3 (BRCLK = 5.0688MHz)

MR23-20	BAUD RATE	ACTUAL FREQUENCY 16X CLOCK	PERCENT ERROR	DIVISOR
0000	50	0.8kHz	-	6336
0001	75	1.2	-	4224
0010	110	1.76	-	2880
0011	134.5	2.1523	0.016	2355
0100	150	2.4	-	2112
0101	300	4.8	-	1056
0110	600	9.6	-	528
0111	1200	19.2	-	264
1000	1800	28.8	-	176
1001	2000	32.081	0.253	158
1010	2400	38.4	-	132
1011	3600	57.6	-	88
1100	4800	76.8	-	66
1101	7200	115.2	-	44
1110	9600	153.6	-	33
1111	19200	316.8	3.125	16

NOTE
16X clock is used in asynchronous mode. In synchronous mode, clock multiplier is 1X and BRG can be used only for TxC.

Figure 3-8a. USART registers

3.4.4 USART control commands

The MTI offers the HOST the ability to read and program the USARTs. Note that the MTI returns the state of the USART status register when I/O transfer commands end, as well as with the Read Status Register command. The HOST may set the USART command register outputs with channel configuration commands as well as with the Write Command Register command.

3.4.4.1 Read USART Status Register command

Command

Byte 0:	0 0 0 1 n n n n	Command byte: nnnn is the port number

Response

Byte 0:	0 0 0 1 n n n n	Response code: nnnn is the port number
Byte 1:	D C f o p c r t	Channel status: D DSR signal status C DCD signal status f framing error or SYN detected o overrun error p parity error or DLE received c transmitter empty or change in DSR or DCD r receiver has data (this will be zero, if input has not been enabled) t transmitter ready (this will be zero if output has been suspended)

See **Figure 3-8-b** and **Appendix J** for more information on the USART Status Register.

COMMAND REGISTER (CR)

CR7 CR6	CR5	CR4	CR3	CR2	CR1	CR0
Operating Mode	Request To Send	Reset Error	Sync/Async	Receive Control (RxEN)	Data Terminal Ready	Transmit Control (TxEN)
00 = Normal operation 01 = Async: Automatic echo mode Sync: SYN and/or DLE stripping mode 10 = Local loop back 11 = Remote loop back	0 = Force $\overline{RTS}$ output high one clock time after TxSR serialization 1 = Force $\overline{RTS}$ output low	0 = Normal 1 = Reset error flags in status register (FE, OE, PE/DLE detect)	Async: Force break 0 = Normal 1 = Force break	0 = Disable 1 = Enable	0 = Force $\overline{DTR}$ output high 1 = Force $\overline{DTR}$ output low	0 = Disable 1 = Enable
			Sync: Send DLE 0 = Normal 1 = Send DLE			

STATUS REGISTER (SR)

SR7	SR6	SR5	SR4	SR3	SR2	SR1	SR0
Data Set Ready	Data Carrier Detect	FE/SYN Detect	Overrun	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	Async: 0 = Normal 1 = Framing Error	0 = Normal 1 = Overrun Error	Async: 0 = Normal 1 = Parity error	0 = Normal 1 = Change in $\overline{DSR}$, or $\overline{DCD}$, or transmit shift register is empty	0 = Receive holding register empty 1 = Receive holding register has data	0 = Transmit holding register busy 1 = Transmit holding register empty
		Sync: 0 = Normal 1 = SYN detected		Sync: 0 = Normal 1 = Parity error or DLE received			

Figure 3-8b. USART registers (continued)

3.4.4.2 Write USART Command Register command

This command may be used to set modem control outputs (RTS, DTR), or to reset error flags.

Command

Byte 0:	0 1 0 1 n n n n	Command byte: nnnn is the port number
Byte 1:	e e R r b c D t	USART command register: ee operating mode R RTS signal control r reset error flags b break control c enable receiver D DTR signal control t enable transmitter

See **Figure 3-8-b** and **Appendix J** for more information on the USART Command Register.

3.4.5 BISYNC SUPPORT

Two new commands support Binary Synchronous Communications (BSC) protocols. These protocols were developed by IBM for use with its 2780 and 3780 data transmission terminals. The MTI BSC support functions follow the BSC protocols as used by those machines.

The "Configure BSC" command executes all configuration and initialization required for BSC operation. After a "Configure BSC" command has been issued, subsequent "BSC Input" and conventional "Block Output" commands may be issued to receive and send BSC messages.

3.4.5.1 Configuration Details

Many operational details of the BSC protocols have been made user-configurable. Where appropriate in the discussion below, we give the configurations for conventional 2780/3780 BSC.

a) Half/Full Duplex Operation

BSC is a half-duplex protocol; data never moves both ways on the communication channel simultaneously. When the underlying communications channel is half-duplex as well, the channel is electrically switched between send and receive modes by turning the RTS modem control signal on and off. This line turn-around can take an appreciable amount of time. Line turn-around delays can be avoided by using a full-duplex communications channel. In this case, the RTS signal remains on continuously, during receive as well as transmit time.

The MTI board can automatically handle the RTS signal, but it must be told whether the communications channel is half or full duplex. If a half-duplex channel is used, set the "H" bit (bit 0) in byte 4 of the "Configure BSC" command. When the channel is configured in this mode, the RTS signal will only be turned on when a "Block Output" command is issued to the MTI. If the "H" bit is not set, RTS will be turned on at the time the "Configure BSC" command is issued, and left on subsequently.

b) Character Length

Character length is programmed by the "ll" bits in byte 2 of the "Configure BSC" command. When using the EBCDIC code set, or ASCII without parity, an eight-bit character length should be selected (ll=11). When using ASCII with parity, a seven-bit character length should be selected (ll=10).

c) Parity

If the "p" bit (bit 4) of byte 2 of the "Configure BSC" command is set, then the parity of each character received will be checked, and a parity bit will be generated for each character transmitted. If a parity error is detected during input, the host will be notified immediately, with an appropriate response.

Conventional BSC protocols (2780/3780) use parity (odd) only with the ASCII character set. When the EBCDIC character set is used, parity is turned off ("p" bit not set).

As an option, the user may select (non-conventional) even parity by setting the "e" bit (bit 5), along with the "p" bit, in byte 2 of the "Configure BSC" command.

3.4.5.2 Block Check Sequences

The 2780/3780 BSC protocols use one of three different methods, alone or in combination, to detect errors in data transmission. When using the ASCII character set without transparent mode, a one-byte LRC (Longitudinal Redundancy Check) is appended to data blocks, and odd parity is generated for each character. (See "Parity," above). When the EBCDIC character set is used, or when ASCII is used in transparent mode, a two-byte CRC (Cyclic Redundancy Check) is appended to data blocks.

Parity generation and checking is described above, under "Parity." To enable either LRC or CRC checking, set the "C" bit (bit 7) in byte 4 of the "Configure BSC" command. Select the appropriate block check by setting the "PPP" bits (bits 4-6) in byte 4 of the "Configure BSC" command, as shown.

Byte	Name	ASCII hex code	EBCDIC hex code
00	SOH	01	01
01	STX	02	02
02	ETX	03	03
03	ETB	17	26
04	IUS	1F	1F
05	SYN	16	32
06	NAK	15	3D
07	DLE	10	10
08	ENQ	05	2D
09	EOT	04	37
0A		00	00
0B	ACK0	30	70
0C	ACK1	31	61
0D	RVI	3C	7C
0E	WACK	3B	6B
0F	PAD	FF	FF

Figure 3-9. Control code table

3.4.5.3 Control Code Sets

The 2780/3780 BSC protocols use a set of ten primary control characters, plus a set of five auxiliary control characters. The binary codes for these characters vary somewhat, depending on the character set (ASCII or EBCDIC) used. Before issuing the "Configure BSC" command to the MTI, the host must build a table in Multibus memory, defining the binary values for the control code sets. The address of this 32-byte code table is passed to the MTI in bytes 3-5 of the "Configure BSC" command.

Control code sets for ASCII and EBCDIC are given in **Figure 3-1.** The control code table in host memory must be in the format shown.

3.4.5.4 Pad Characters

BSC message blocks are transmitted with preceding and trailing "pad" characters. The preceding pad characters consist of two (or more) SYN (synchronous idle) characters. These leading pad characters are used by the receiver to establish byte-synchronization with the data stream. Trailing pad characters consist of all "1" bits. These characters must be present on many systems to insure proper accumulation of block check sequences. The MTI will add pad characters before and after each message block it transmits. The number of leading and trailing pads is specified in byte 5 of the "Configure BSC" command. The binary codes for leading and trailing pad characters (SYN and PAD, respectively) are defined in the control code table, described above.

3.4.5.5 Operation of the Protocols

The 2780/3780 BSC protocols are "master/slave" protocols. This means that on any BSC channel, there is one and only one "master" station, which controls all data transfers on the channel. One or more "slave" stations communicate under the control of the master station.

The master station initiates an exchange by sending a "poll" or "select" message to a slave. The addressed slave responds with an "acknowledge," or other message. The exchange continues, with the master and slave alternately sending and receiving messages, one at a time, until the exchange is terminated.

3.4.5.6 Using 'Delayed Receiver Enable'

Suppose that the MTI is playing the part of "master" on a BSC channel. When a "select" message is sent to a slave station, we expect that the slave will respond with an "acknowledge" message in a short time. In order to insure that the slave's answering message is received properly, we issue a "BSC Input" command, with the "Delayed Receiver Enable" bit (bit 1) in byte 1 SET. We THEN issue the "Block Output" associated with the "select." The "Delayed Receiver Enable" bit tells the MTI not to enable the receiver until after the next block output is complete. The select/acknowledge sequence looks like the following:

Host:	MTI:
Issue "BSC Input" Delayed Enable SET	
	Set up input operation. Do NOT enable receiver.
Issue "Block Output" with "select" message	
	Transmit "select"
	When "select" output is complete, return response to host. Enable receiver, in anticipation of slave's acknowledgement.
	When acknowledgement is received, return response to host.
Repeat process, with next message.	

If the MTI is playing the part of the slave, the situation is somewhat different, since the slave must start out "listening:"

Host:	MTI:
Issue "BSC Input" Delayed Enable NOT SET	
	Set up input operation. Enable receiver.
	When a message is received, return "BSC Input" response to host.
Issue "BSC Input". If we are going to transmit a message, SET Delayed Enable. (In this case, subsequent interaction looks like that shown above for the "master".)	
If the received message requires no response (e.g., a select for another station), DO NOT set Delayed enable.	

Use of the Delayed Receiver Enable feature is optional. However, if it is not used, the host must be able to guarantee that a "BSC Input" command will be posted to the MTI in time to pick up an incoming message.

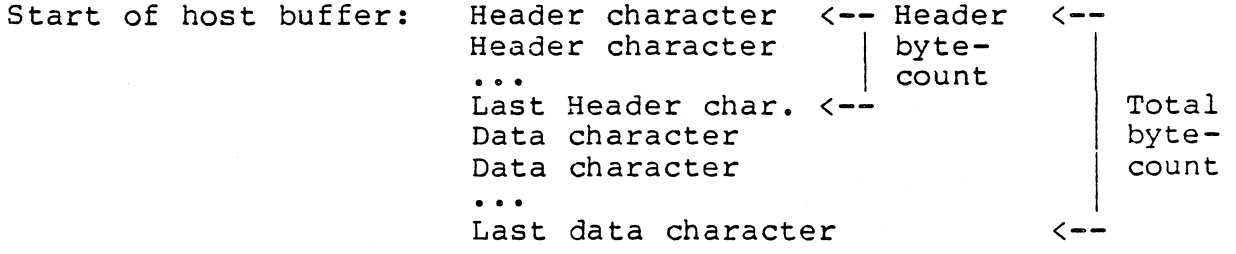
3.4.5.7 Using 'Wait for Poll'

Normally, when the MTI receives an EOT character during a "BSC Input" operation, the input will be terminated, and the host will be informed that an EOT message was received. Very often, however, an EOT will be immediately followed by a poll or select sequence. The host software controlling the MTI can tell the MTI to wait for such a poll or select by setting the "W" bit (bit 0) in byte 1 of the "BSC Input" command. When this bit is set, the "BSC Input" will not be terminated when an EOT is received. The MTI will wait for an additional message (presumably a poll or select), and then terminate. The "BSC Input" response will indicate the message class of this second message; the EOT is effectively ignored.

3.4.5.8 Headers

When a "BSC Input" command terminates, two byte-counts are passed back to the host in the "BSC Input" response. The first byte count (byte 3 and 4 of the response) gives the total number of bytes moved into host memory. The second byte count (bytes 5 and 6 of the response) gives the number of bytes of "header" data moved into host memory. Note that the header byte-count is included in the total byte-count.

Headers may be prepended to data blocks (between SOH and STX characters). Address bytes in poll/select messages are also treated as header information. Header and data bytes are placed into host memory as shown below:



Note that there may not be any header characters in some BSC messages. In this case, data characters will be stored beginning at the start of the host buffer, and the header byte-count will be zero. If there are neither header nor data characters, then no bytes will be moved into host memory, and both byte counts will be zero. If the message contains only header bytes, the two byte counts will be equal.

3.4.5.9 Multiple Blocks

BSC data messages may contain several logical records, separated by IUS characters, and block-check sequences. When this is the case, the IUS characters will be included in the data returned to the host, but the block check bytes will not.

a) Output

2780/3780 BSC output is straightforward. The host builds a message to be sent, including all control characters, headers, and block-check sequences in Multibus memory. A "Block Output" command is then issued to the MTI, specifying the buffer address and byte count.

After the MTI has moved the data block into buffer memory, the RTS signal will be turned on (if the "H" bit was set during configuration), and the specified number of SYN characters will be transmitted, followed by the host data. The specified number of PAD characters will be appended to the end of the block.

After the last byte has been shifted out of the USART, the RTS signal will be turned off (if the "H" bit was set during configuration). If a "Delayed Enable" "BSC Input" command is pending, the receiver will be enabled, putting it into sync hunt mode. A "Block Output" response will be returned to the host.

b) Input

The host initiates input by issuing a "BSC Input" command to the MTI. The "Wait for Poll" and "Delayed Receiver Enable" bits should be set or cleared as appropriate. When a valid message block is received, or a recognizable error is detected, any data associated with the message will be transmitted via DMA to the host buffer. A response will be returned to the host, indicating the type of message received. The USART receiver will be disabled after the last byte of the message has been received.

3.4.5.10 Configure BSC command

This command executes all configuration and initialization required for BSC operation.

Command		
Byte 0:	0 1 1 1 n n n n	Command byte nnnn = channel number
Byte 1:	0 0 1 1 0 0 0 0	Second Command byte
Byte 2:	0 0 e p 1 1 0 0	USART Mode Register 1: e Set for even parity p Set to enable even or odd parity 11 Select character length (See Figure 3-8-a, "Mode Register 1")
Byte 3:	m m m m r r r r	USART Mode Register 2: mmmm Tx/Rx clock operation rrrr Baud rate (See Figure 3-8-a, "Mode Register 2")
Byte 4:	C P P 0 0 0 0 H	Command options byte: C Set to enable LRC/CRC PP Select LRC/CRC: 00 LRC 01 CRC16 H Set for auto RTS control on half-duplex channel.
Byte 5:	L L L L T T T T	Pad characters: LLLL Number of leading SYNs to prepend to messages (0-15) TTTT Number of trailing PADs to append to messages (0-15)
Byte 6:	1 1 1 1 1 1 1 1	Low byte,
Byte 7:	m m m m m m m m	Medium byte,
Byte 8:	h h h h h h h h	High byte of address of 16-byte control codes table

3.4.5.11 BSC Input command

This command tells the MTI to input a BSC message. When a message has been received, any data associated with the message will be transmitted via DMA into host memory, at the location specified in the BSC Input command. A response will be returned to the host, indicating that input is complete, and giving the message type received.

Command		
Byte 0:	1 0 0 1 n n n n	Command byte: nnnn is the port number
Byte 1:	0 0 0 0 0 0 d W	Options byte: d Set to delay receiver enable until after completion of next Block Output W Set to Wait for Poll/Select
Byte 2:	l l l l l l l l	Low byte,
Byte 3:	m m m m m m m m	Medium byte,
Byte 4:	h h h h h h h h	High byte of data address in HOST memory. The MTI will load data into the HOST memory starting at this address.
Byte 5:	l l l l l l l l	Byte count:
Byte 6:	h h h h h h h h	The MTI will transfer at most this many bytes into host memory.

Response (returned when input has ended)

Byte 0:	1 0 0 1 n n n n	Response code: nnnn is the port number
Byte 1:	t X X M M M M M	Message type: t Set if message contained transparent mode data MMMMM = 00000 Good data - a data block, possibly with a header, was received, ETB terminated. 00001 Good data - a data block, possibly with a header, was received, ETX terminated. 00010 ACK0 - Any address information returned as "header" 00011 ACK1 00100 ENQ - Any address information returned as "header" 00101 EOT 00110 NAK 00111 RVI 01000 TTD - Any partial block received will be returned 01001 WACK 10000 Error - LRC or CRC error detected. 10001 Error - VRC (Parity) error detected 10010 Error - Buffer overrun 10011 Error - Host aborted input
Byte 2: Byte 3:	1 1 1 1 1 1 1 1 h h h h h h h h	Low byte, High byte of number of characters in message.
Byte 4: Byte 5:	1 1 1 1 1 1 1 1 h h h h h h h h	Low byte, High byte of number of characters in header.

3.4.6 Miscellaneous commands

The HOST may use this set of commands to control the operation of channels that are transferring data.

3.4.6.1 Abort Input command

This command will abort any input in progress. If a Block Input command is in progress, the MTI will return a response to the HOST indicating that the command has been aborted. Note that if a partial input block has been buffered at the time the Abort Input command is issued, the partial block will be moved into host memory. The byte count returned with the Block Input response will reflect the number of bytes in the partial block.

If the channel is in Continuous Block Input mode, this command clears all input blocks from the MTI buffers. Input received following the Abort Input command will be buffered normally. Note that if a Block Input command is pending at the time the Abort Input is issued, a Block Input response will be generated. To clear the input buffers, and stop further input buffering, a new Configure Input command should be issued with all input modes disabled.

Command

Byte 0:	1 0 1 0 n n n n	Command byte: nnnn is the port number

3.4.6.2 Abort Output command

This command will abort any output in progress. If a Block Output command is in progress, the MTI will return a response to the HOST indicating that the command has been aborted.

Command

Byte 0:	1 1 1 0 n n n n	Command byte: nnnn is the port number

3.4.6.3 Suspend Output command

The HOST can use this command to temporarily halt data transfer during Output commands. When the operation is resumed, output will begin at the point where it was suspended. If output is already suspended, this command has no effect. Output remains suspended until it is resumed (with a Resume Output command or a CTRL-Q from the terminal, if CTRL-S/CTRL-Q protocol is in effect for this channel) or else aborted.

Command		
Byte 0:	1 0 1 1 n n n n	Command byte: nnnn is the port number

3.4.6.4 Resume Output command

The HOST uses this command to resume output that is temporarily halted by either a Suspend Output command, or reception of a CTRL S (assuming the CTRL-S/CTRL-Q protocol is in effect for this channel). Output will begin at the point where it was suspended. If no output is in progress on the indicated channel, or if output is not suspended, this command has no effect.

Command		
Byte 0:	1 1 0 1 n n n n	Command byte: nnnn is the port number

3.4.6.5 Read Error Code command

When the Command Error bit is set, (port +01, bit 2), the HOST must issue this command to clear the bit. A response code will be returned to the HOST, specifying the error that has occurred. See **Section 3.5** (Error Handling), for an explanation of possible error codes.

Command

Byte 0:	0 0 1 0 0 0 0 0	Command byte

Response

Byte 0:	0 0 1 0 n n n n	Response Code nnnn is the port number, only when applicable (see Section 3-5)
Byte 1:	e e e e e e e e	Error Code

3.5 ERROR HANDLING

The MTI-850/1650A signals the HOST that an error condition has occurred by setting the Command Error Bit (bit two) of the Status Port (+01 IN). The HOST software must issue a Read Error Code command to clear this byte.

These are the possible error conditions which the MTI-850/1650A can return; each uses two bytes in the Response FIFO. Some of the responses will include a port number in the right nibble of the first byte; this is shown as an "n" below:

Hex	Code	Error Condition Reported
20	00	Undefined Command Error
20	01	Undefined Command Nibble
2n	02	Port Offline (Self-test Error)
2n	03	Command Contains Too Few Bytes *
20	04	Bad Host Memory Address Given
2n	10	Block Input Not Permitted (on this channel)
2n	11	Block Input Already In Progress (on this channel)
2n	13	Input Request Too Big
2n	20	Block Output Not Permitted (on this channel)
2n	21	Block Output Already In Progress (on this channel)
2n	22	Output Buffer Full
2n	23	Output Request Too Big
2n	30	Block Input In Progress When Attempting to Reconfigure Termination Mask -- Input Must Be Quiescent
2n	31	Single Character Input In Progress When Attempting to Reconfigure Termination Mask -- Input Must Be Quiescent
20	40	Buffer Length Table Bad (>16K)
20	41	Channels Already Configured
20	42	Buffer Length Not a Multiple of 4
20	43	Buffer Length Not At Least 32 Bytes
20	FF	Command Contains Too Many Bytes (NOTE: Error code FF is only a warning; the command is executed anyway.)

See the Software Reference Summary in **Section 3.7** for a list of which errors can occur in response to which commands.

* If the MTI received at least one byte, then it will return the port number. If no bytes were in the Command FIFO, n will be 0.

3.6 EXAMPLE APPLICATIONS

Programming the MTI-850/1650A requires some care and planning, especially in the response handling routine, because so many things can be happening at once and because it is not necessary for the HOST to read from the Response FIFO after every command it writes to the Command FIFO.

To aid you in sorting out all of the MTI's programming requirements, we discuss three possible applications below. In the first, we look at how you would use the MTI without reconfiguring its channels, defaulting to the channel characteristics set on the DIP switches on the main circuit board. In the second, we examine reconfiguring the MTI for more complex single character transfers. In the third example we use block transfer mode.

3.6.1 Default characteristics example

Let's assume that you have an MTI-1650A and wish to use it with one terminal on port 0 which is configured to operate with these default channel characteristics set on the DIP switches (the settings which MTI's are normally shipped with): 9600 Baud; 8 data bits; 1 stop bit; no parity.

3.6.1.1 Execute command subroutine

Since the HOST software doesn't need to reconfigure any channels, it can begin communicating with the terminal right away. The software will need to repeatedly issue commands to the MTI, so we will construct an Execute Command Subroutine to use as a module. It will do the following:

	Action	Function
1.	Input a byte from +01	
2.	AND the byte with 01	Wait for MTI Ready
3.	If result zero then jump to 1.	
4.	Output command byte(s) to +00	Load Command into FIFO
5.	Output anything to +02	Strobe Execute Bit
6.	Input a byte from +01	
7.	AND the byte with 01	Wait for MTI Ready
8.	If result zero then jump to 6.	(will occur as soon as
9.	Input a byte from +01	MTI validates command)
10.	AND the byte with 04	
11.	If result nonzero then jump to 13.	Check for Error Bit
12.	Return from subroutine	Normal return
13.	Notify calling routine of error	
14.	return from subroutine	Error return

3.6.1.2 Retrieve response subroutine

Next we need a module to read responses back from the Response FIFO, called the Retrieve Response Subroutine, which does this:

	Action	Function
1.	Input a byte from +01.	Check Response Available
2.	AND the byte with 02.	bit in status byte. If
3.	If result zero, exit.	no response, then exit.
4.	Input a byte from +02.	Clear Response Available bit.
5.	Input a byte from +01.	Get status byte.
6.	AND byte with 80h.	Test the Valid Data bit.
7.	If result zero, exit.	Exit if no more response bits.
8.	Input response byte from +00.	
9.	Store response byte in buffer.	
10.	If this is the first byte of a response, then set a counter to the number of additional bytes required to complete response. If this is not the first byte, decrement counter.	
11.	If counter is not zero, jump to 5.	
12.	If counter is zero, return with complete response in buffer.	

3.6.1.3 Character output routine

Now we can use these subroutines to construct the routines for single character input and output. First we'll do the Character Output Routine:

	Action	Function
1.	Call Command Execute Subr with bytes: 40,cc	Output single character cc to port 0
2.	If no error then return.	Process complete
3.	Call Common Execute subr with byte 20	Read Error Code
4.	Call Retrieve Response Subr	Get error code
5.	If code=20,02 then jump to 10.	Port off line!
6.	If code=20,23 then jump to 7.	Output buffer full
7.	Increment temporary variable R	Retry count
8.	If R>RMax then jump to 10.	Too many retries
9.	Wait	
10.	Jump to 1. with same value of cc	Try again
11.	Report error	(Inform operator)

3.6.1.4 Single character input routine

Here is the comparable Single Character Input Routine:

	Action	Function
1.	Call Command Execute Subr with byte: 00 (unless done previously)	Enable single char. input from port 0
2.	If no error returned, then jump to 5.	
3.	Call Retrieve Response Subr	Get error code
4.	Report error and exit	(Must be Port Off Line!)
5.	Input a byte from +01	Get status
6.	AND the byte with 02	Check Response Available
7.	If result is zero then jump to 5.	(This is wait loop polling; not very efficient. Using the Timer or Interrupts or both would save CPU time.)
8.	Call Retrieve Response Subr	Get single char. input response (3 bytes)
9.	Verify first byte is 00	
10.	Check next byte (USART status)	Look for Framing Error (20) or Over-run (10)
11.	If error then report & exit	
12.	Put 3rd byte in HOST input buffer	Third byte is the input character.
13.	Jump to 5.	Go get another one

Though not the most efficient use of the MTI-850/1650A's capabilities, this approach is simple and workable for a quick routine to input and output, and you can improve upon its basic structure.

3.6.2 Reconfigured character mode

In this next example, we'll take the same MTI-1650A with factory switch settings, and reconfigure it from software to work with four 4800-baud terminals. We'll want each terminal to have as much buffer space as possible in the MTI's on-board 16K buffer, so we'll reconfigure buffer sizes. Also we'll look for modem signal changes, use the CNTL-S/CNTL-Q feature to allow the users to start and stop output to their screens, and use timer interrupts to minimize HOST overhead servicing these single-character operations.

Again, we will use the Execute Command and Retrieve Response subroutines defined in **Section 3.6.1** above.

Before we begin coding, we need to calculate the buffer sizes for the four ports (therefore eight channels) we will use. (We'll do the calculations in decimal notation.) There's 16K, or 16,384 bytes total available. Each unused channel (of which there are 24) needs a 32-byte buffer. This leaves:

16,384 - (24 * 32) = 16,384 - 768 = 15,616 bytes

to split between eight channels, or 1952 bytes per channel. (This is 07A0 hexadecimal.) Coincidentally, this is an even multiple of 4, which each buffer size must be.

The first code sequence initializes the MTI's ports to the desired configurations. It must begin with the Configure Buffer Sizes command. Following this, commands must be issued to reconfigure each of the first four ports (from 0 to 3) with the Reconfigure a Port Asynchronous command to change the baud rates in each USART's Mode Register 2.

Since this command also changes a USART's Command Register and other parameters in the Mode Registers (which we don't want to affect), we must rewrite them with the default values:

Mode Register 1: ss=01, e=0, p=0, ll=11, and cc=01,
which forms the byte 4D Hex,

Mode Register 2: mmmm=0011, rrrr=Baud rate (1100 for 4800 Baud)
which forms the byte 3C Hex,

Command Register: ee=00, R=1, r=1, b=0, C=1, D=1 and T=1,
which forms the byte 37 Hex.

Finally, the initialization routine must set the timer rate with the Configure Timer command, set up to respond to modem signal changes with the Configure Modem Status command, turn on the CNTL-S/CNTL-Q protocol with the Configure Input Channel command, and write to the Interrupt Enable byte to enable timer interrupts. This is how the code will go:

	Action	Function
1.	Set up the Buffer Length Table in the HOST memory: A0,07,A0,07,A0,07,A0,07, A0,07,A0,07,A0,07,A0,07, 20,00,20,00,20,00,20,00,...	Defines 07A0 hex bytes per channel for first four channels, 0020 hex bytes each for remainder
2.	Call Command Execute subr with bytes 70,80,ll,mm,hh	Configure Buffer Sizes ('hhmmll' is Buffer Length Table address in HOST mem.)
3.	If error return then report and exit	(If HOST software works no errors should appear here)
4.	For n = 0 to 3 do 5. through 10:	n = Port #
5.	Call Command Execute subr with bytes: 7n,00,4D,3C,37	Configure a Port Asynch. (default parameters except new Baud rate)
6.	If error returned, then report and exit	Error 02, Port Off-line, could occur here
7.	Call Command Execute subr with bytes 7n,40,04	Configure Input Channel for CTRL-S/CTRL-Q protocol
8.	If error returned, then report and exit	Error 02, Port Off-line, could occur here
9.	Call Command Execute subr with bytes 7n, 41	Configure Modem Status to notify of changes
10.	If error returned, then report and exit	Error 02, Port Off-line, could occur here
11.	Call Command Execute subr with bytes 73,90	Configure Timer (200 mSec. interval)
12.	If error returned, then report and exit	No errors should appear here
13.	Output byte 10 to +01	Set interrupt enable for timer interrupts

---

Now the HOST can begin performing single character I/O on all these reconfigured channels. Since we're programming an interrupt-driven program, we need to break it into two portions: a module to handle the interrupts, and a module to move characters in and out of a buffer in HOST memory serviced by the interrupt module. Let's look at the interrupt handler first. It "comes to life" when the MTI interrupts the HOST, and will do the following:

	Action	Function
1.	Input a byte from +01	MTI Status Byte
2.	AND the byte with 10	Check Timer bit
3.	If nonzero result then jump to 13.	Timer interrupt
4.	AND the byte with 01	Check MTI Ready
5.	If nonzero then jump to 7	MTI Ready Interrupt
6.	Report error and exit	Unexpected interrupt
7.	Output 00 to +01	Clear MTI Ready
8.	If HOST buffer not empty then jump to 11.	More data to go out?
9.	Output 10 to +01	Enable timer int. only
10.	Return	
11.	Get a "command packet" (bytes forming a command) from the HOST buffer and call Command Execute subr	
12.	Return	
13.	Input a byte from +01	Timer Interrupt
14.	AND the byte with 20	Check for Response Avail.
15.	If zero result then jump to 18.	
16.	Call Retrieve Response subr	Get responses
17.	Read a byte from +02	Clear Response Available
18.	Read a byte from +03	Clear timer bit
19.	Return	

Note that the response handling module, not shown here, must be able to sort out character inputs from each of the four channels, error messages from any command, notification of changes in modem status.

The output buffering module works with the interrupt handler, doing this part of the job:

1. Some task in the HOST system wants to send a character to a terminal
2. If output buffer full then jump to 6.
3. Put command bytes into output buffer
4. Write 01 to +01 (Enable MTI Ready interrupts)
5. Return
6. Handle buffer overflow
7. Return

3.6.3 Block mode

Having already explored the intricacies of reconfiguring channels, handling interrupts and error recovery, we will not complicate the block mode example with these details. Consider the task of programming an intelligent data-entry terminal that can perform on-screen editing independent of the HOST software.

Such a device is ideal for block transfers, because the user can interact with the terminal in entering and editing data, and then notify the HOST that a block is ready to go by pressing one of the keys set by the HOST in the MTI's termination mask. Commonly, the End of Text character (ETX, ASCII number 03) is used for this purpose. Here is what the HOST software could do to set up to communicate this way:

	Action	Function
1.	Call the Command Execute subr with bytes 7n, 40, 11, 03	Configure Input Channel n command for Block input with termination char. 03
2.	Call the Command Execute subr with bytes 7n, 60, 01	Configure Output Channel 'n' for Block output

To input a block:

1.	Create buffer in HOST memory	Up to 256 bytes needed
2.	Call the Command Execute subr	Data in to HOST memory buffer with bytes 8n, 11, mm, hh, 11, hh starting at 'hhmm11' for up to 'hh11' bytes
3.	Input a byte from +01	MTI Status Byte
4.	AND the byte with 01	Check for MTI Ready
5.	If nonzero then jump to 4.	Wait for it
6.	Done for now...	

On interrupt:

	Action	Function
7.	Call Retrieve Response subr	Get response
8.	If no errors, buffer has data!	

To output data:

	Action	Function
1.	Create a buffer in HOST memory	Data in buffer to start
2.	Call Command Execute subr with bytes Cn, ll, mm, hh, ll, hh	Send 'hhll' bytes out from buffer at 'hhmmll'
3.	Input a byte from +01	MTI Status Byte
4.	AND the byte with 01	Check for MTI Ready
5.	If nonzero then jump to 4.	Wait for it
6.	Data has been moved to MTI. Buffer is free.	Host memory may be re-allocated.

On interrupt:

	Action	Function
6.	Call Retrieve Response subr	Get response
7.	If no errors, data went out.	

As you can see, block mode is simple to use and offers much power to the HOST software. Again, when communicating with many terminals make sure the response handling routine has sufficient intelligence to handle non-synchronous and sometimes unexpected messages in the Response FIFO.

3.7 SOFTWARE REFERENCE SUMMARY

In this chart, the "#" column shows the number of bytes in a normal response, and the "Errs." column indicates which errors can result from the command. In addition to the errors noted, every command can receive error codes 03 (too few bytes) and FF (too many bytes).

Binary	Command Name	#	Errs.	Section
0000 nnnn	Enable Single Character Input	3	02	3.4.2.1
0001 nnnn	Read USART Status Register	2		3.4.4.1
0010 0000	Read Error Code	2	02	3.4.5.5
0100 nnnn	Single Character Output	0	02,22	3.4.2.3
0101 nnnn	Write USART Command Register	0	02	3.4.4.2
0110 nnnn	Disable Single Character Input	0	02	3.4.2.2
0111 nnnn	Configuration Command -- second byte follows:			
	0000 0000 Config. Asynchronous	0	02	3.4.1.2
	0001 0000 Configure Synchronous	0	02	3.4.1.3
	0100 0000 Configure Input	0		3.4.1.5
	0101 0000 Termination Mask	0	30,31	3.4.1.6
	0110 0000 Configure Output	0		3.4.1.7
	0111 000n Modem Status	2	02	3.4.1.8
	1000 0000 Buffer Sizes	0	40,41, 42,43	3.4.1.4
	1001 0000 Configure Timer	0		3.4.1.1
1000 nnnn	Block Input	6	02,10, 11,13	3.4.3.1
1010 nnnn	Abort Input	0	02	3.4.5.1
1011 nnnn	Suspend Output	0	02	3.4.5.3
1100 nnnn	Block Output	5	02,20, 21,23	3.4.3.2
1101 nnnn	Resume Output	0	02	3.4.5.4
1110 nnnn	Abort Output	0	02	3.4.5.2

SECTION 4: THEORY OF OPERATION

This section will not attempt to explain the workings of the MTI-850/1650A down to the integrated circuit level, but will describe the way the main elements function. This discussion is mainly intended for the curious, and is not required reading for the installation, programming or normal use of the MTI-850/1650A.

4.1 OVERVIEW

A simpler overview of the MTI-850/1650A's elements and their operations is given in **Section 1.3.** This section will cover those concepts in more detail.

4.1.1 Block diagram

The MTI-850/1650A consists of a number of separate circuit elements, connected by several address and data buses, as illustrated in **Figure 4-1.** Bidirectional switching elements (transceivers) cause these buses to connect the circuit elements in different ways during different phases of the MTI-850/1650A's operation. We have numbered the transceivers 1 through 7 for easy identification.

Each transceiver can be OFF, passing no signals, or ON, passing signals in one direction at time. In **Figure 4-1** some of them are oriented to appear to operate "up" and "down," others "left" and "right." For purposes of this explanation, a transceiver is described as being ON LEFT when it is passing signals from right to left, and so on.

The buses shown are multipurpose; some are used by different components of the MTI-850/1650A during different phases of its operation. To help clarify the bus functions, a shading convention has been used: bus lines with white interiors are for address information; bus lines with half-tone grey interiors are for data.

The type of edge on a bus line indicates the main device using it: edges are evenly dashed for the 8085 processor, unevenly dashed for the DMA controller chip, and hash-marked for the Multibus.

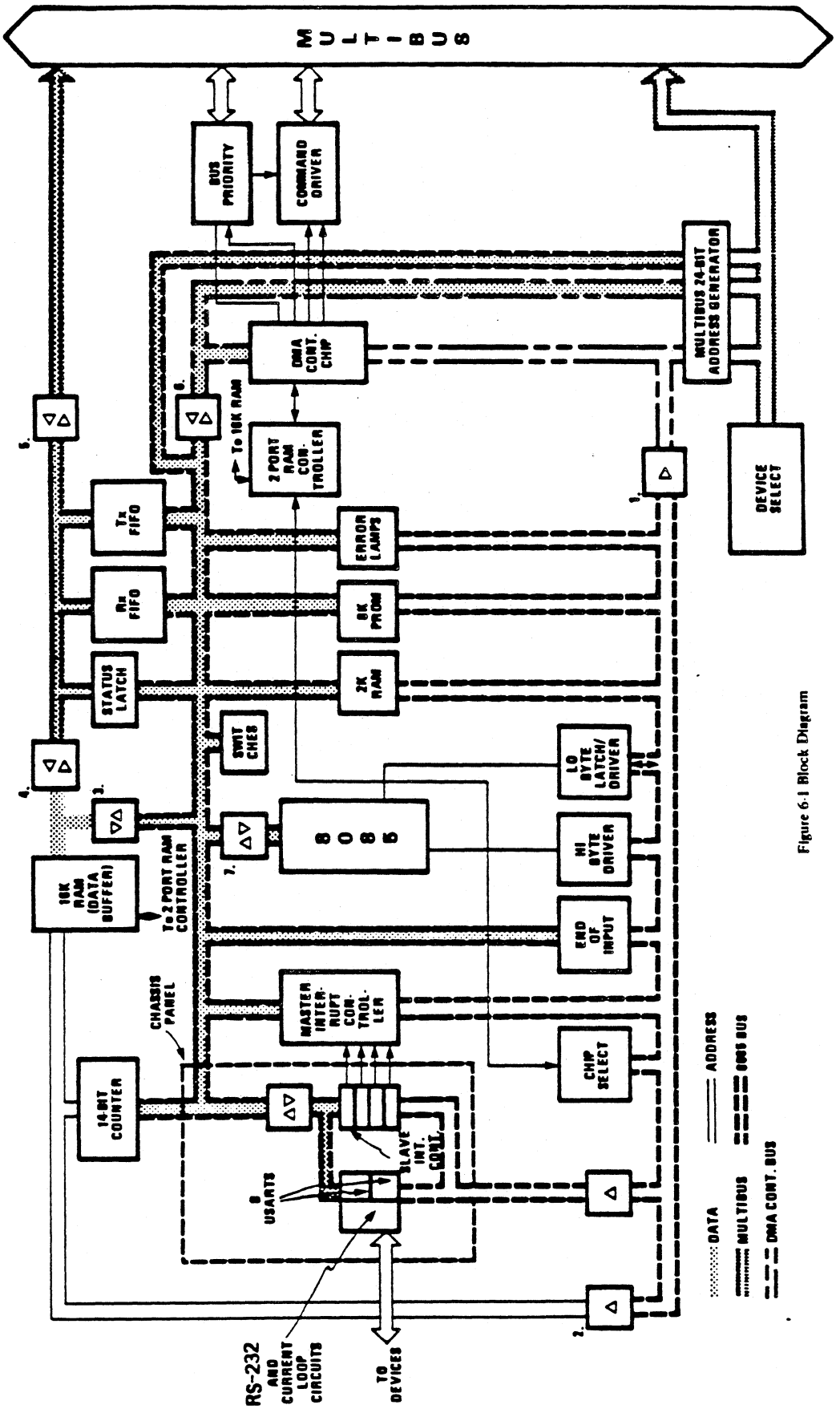


Figure 6-1 Block Diagram

Figure 4-1 Block diagram

4.1.2 Main circuit elements

The main circuit elements are:

-- 8085 microprocessor.

-- 8K Programmable Read Only Memory (PROM) containing the 8085's firmware.

-- 2K scratchpad Random Access Memory (RAM) used by the 8085.

-- Address bus interface for the 8085 (represented by two boxes labeled "HI BYTE DRIVER" and "LOW BYTE LATCH/DRIVER").

-- An address latch to hold the low address byte generated by the 8085.

-- DIP switches and the diagnostic switch read by the 8085.

-- Status LEDs controlled by the 8085.

-- DMA controller chip.

-- 16K data buffer RAM used by the 8085 and the DMA controller.

-- A 2-port RAM controller to resolve contentions between the 8085 and the DMA controller for access to the 16K RAM.

-- 14-bit counter for generating data buffer RAM addresses during DMA operations.

-- 24-bit Multibus address generator.

-- Multibus interface circuitry for the DMA controller (consisting of the two boxes labeled "BUS PRIORITY" and "COMMAND DRIVER").

-- Command and Response FIFOs (labeled "Tx FIFO" and "Rx FIFO").

-- Device select circuitry which decodes Multibus operations for the MTI.

-- Chip select circuitry which decodes 8085 memory and I/O operations.

-- End of input detect circuitry.

-- Master interrupt controller that generates interrupts for the 8085.

-- Four slave interrupt controllers that feed interrupt requests to the master interrupt controller.

-- Sixteen USARTs that handle serial communications with devices and feed interrupt requests to the slave interrupt controllers.

-- RS-232 and current loop driving circuits controlled by the USARTs.

4.1.3 Phases of operation

Under the direction of the 8085 firmware, the elements listed above act together in different ways during different phases of the MTI-850/1650A's operation. These phases are:

1) Multibus Slave Mode (under host software control)

2) Multibus Master Mode (MTI-850/1650A controls Multibus)

3) MTI-850/1650A internal operations

NOTE: Many events in the MTI-850/1650A occur simultaneously or in a rapid succession that appears to be tandem operation.

4.2 MULTIBUS ACCESS

The MTI-850/1650A interacts with the Multibus in one of two ways. In slave mode, the unit is just another peripheral device in the bus, following orders from the host software. However, when directed by the host to perform block transfers, the MTI-850/1650A enters master mode and takes control of the Multibus to directly access the host's memory without involving the host CPU.

4.2.1 Slave mode

In slave mode, the MTI-850/1650A is programmed by the host software. When the host issues a command to access one of the four bus addresses assigned to the MTI-850/1650A, the device select circuitry recognizes this by decoding the address and producing one of eight possible outputs.

The possible outputs produced by the device select circuitry are:

```
 Read +0 -- Response FIFO
      +1 -- Read Board Status
      +2 -- Clear Response Available
      +3 -- Clear Timer Bit

Write +0 -- Command FIFO
      +1 -- Set Interrupt Mask
      +2 -- Set Execute
      +3 -- Reset
```

In slave mode, the MTI-850/1650A has Transceiver 5 OPEN LEFT during write operations (the host writing to the board) and OPEN RIGHT during read operations. Depending upon which of the above is requested by the host, the MTI may use its FIFOs, Status Latch, or Interrupt Mask (not shown).

A reset operation initializes the MTI board to its power-up condition. Writing to the Set Execute address causes the Master Interrupt Controller to interrupt the 8085 processor, informing it that a command may be in the Command FIFO.

4.2.2 Master mode

In master mode, the MTI-850/1650A takes control of the Multibus, and the DMA controller chip reads or writes host memory. Transceivers 4 and 5 are OPEN LEFT when the DMA chip is reading data out of host memory into the MTI's 16K buffer RAM, and the DMA chip generates the necessary Multibus address with the help of the Multibus 24-bit address generator circuitry.

When the chip is writing into the host memory, the address lines work the same, but the data lines flow in the opposite direction, with 4 and 5 OPEN RIGHT, allowing data to move from the 16K RAM to the host memory. The chip uses the Bus Priority and Command Driver circuits to control the bus. The 14-bit counter generates the sequential addresses for the chip, to assist the chip in moving data to and from the 16K RAM smoothly.

4.3 8085 OPERATION

The 8085 processor has the USARTs, the End of Input circuitry, the Status Latch, its scratchpad RAM and ROM, and the switches and LEDs mapped into its address space. The 8085 shares the 16K buffer RAM with the DMA controller chip, through the 2 Port RAM Controller.

When the 8085 reads the 16K RAM, transceivers 3 and 7 are OPEN DOWN; when it writes to the 16K RAM, these are OPEN UP. The 8085 communicates with the DMA chip with transceiver 1 OPEN RIGHT and transceiver 6 OPEN LEFT or RIGHT depending on whether the chip is being programmed or interrogated.

When the 8085 communicates with the USARTs, Chip Select decodes that the USARTs are selected and transceiver 7 is ON in the appropriate direction.

4.3.1 8085 memory and I/O map

Here is the basic structure of the 8085's memory and I/O map:

		READ	WRITE
CS0	0000 - 1FFF	ROM	
	2000 - 3FFF		
CS1	4000 - 43FF	RAM Indexed	RAM Indexed
	4400 - 47FF	RAM Normal	RAM Normal
	4800 - 7FFF		
CS2	8000 - BFFF	Buffer RAM	Buffer RAM
CS3	C000 - FFFF	Memory Mapped I/O	Memory Mapped I/O

4.4 USARTs OPERATIONS

The USARTs handle all of the actual protocol (for RS-232) and timing for communicating with serial devices. Each of the up to eight or 16 USARTs can separately interrupt the Master Interrupt Controller, which in turn interrupts the 8085, so that a USART can be serviced quickly to maintain high-speed data transfers.

SECTION 5: IN CASE OF TROUBLE

Section 5: In case of trouble

This section is an aid to troubleshooting problems in the communications subsystem. Typically, when a problem appears it is not obvious where on the board it may be. This section can help you trace the problem to its source.

5.1 PROBLEM ISOLATION

This section assumes that you have read **Sections 1** and **2** of this manual, and are familiar with the configuration, installation and operation of the MTI-850/1650A. If you require more in-depth information, refer to **Section 4,** (Theory of Operation).

5.1.1 Status LEDs

To help you determine whether the MTI-850/1650A is the source of trouble, the board includes an automatic self-test feature, as well as a special diagnostic mode, utilizing the on-board intelligence of the 8085's firmware to check the integrity of the MTI-850/1650A.

Eight status LEDs are located on the edge of the MTI-850/1650A's main circuit board, on the left side as you look into the Multibus enclosure at a normally mounted unit (refer to **Figure 1-3** if necessary). The automatic self-test facility and the diagnostic mode both use these LEDs to communicate status information to the user. Each lamp represents one bit in an eight-bit pattern. In the list that follows, each pattern is identified by its actual appearance (an "x" indicates a lamp is ON, "." indicates off), and its hexadecimal equivalent.

5.2 SELF-TEST

Whenever you power up your Multibus or reset it with the reset button, the MTI-850/1650A firmware performs its automatic self-test operation. While this is occurring, you will see a rapidly changing sequence of patterns on the status LEDs.

The first procedure is a "checksum" test of the firmware itself, guaranteeing its data integrity. The firmware then proceeds to test other components of the unit. If the MTI-850/1650A passes all tests, the LEDs will begin displaying their normal operating pattern. This is a distinctive appearance of left and right motion caused by the lamps blinking successively, two at a time, something like the "chaser" lights on a theater marquee.

If the self-test detects an error, it will halt, displaying one of the patterns that follow.

5.2.1 LED self-test error codes

Pattern	HEX	Error Condition
xxxx ...x	F1	ROM Checksum Failure
xxxx ..x.	F2	Scratchpad Memory Address Lines Failure
xxxx ..xx	F3	Scratchpad Memory Data Lines Failure
xxxx .x..	F4	Data Buffer Memory Address Lines Failure
xxxx .x.x	F5	Data Buffer Memory Data Lines Failure
xxxx .xx.	F6	Indexer Failure
xxxx x...	F8	Master Interrupt Controller Failure
xxxx x..x	F9	Missing USART Assembly
xxxx x.x.	FA	Bad USART Second Eight USARTs
xxxx x.xx	FB	All USARTs Failed Loopback Test

Tests are performed in the order listed, so a failure at any point means that all prior tests passed.

5.2.2 Self-test checkpoint patterns

As an aid to identifying problems with the MTI's 8085 microprocessor, the status LEDs briefly flash a "checkpoint" pattern after each test successfully completes. Normally these patterns do not remain on the LEDs long enough to be visible. However, should the 8085 halt abnormally during self-test, the LEDs will show the last checkpoint the 8085 successfully reached. Here are the checkpoint patterns:

Pattern	Hex	Meaning
x..x	90	Start
x..x ...x	91	Stack Works
x..x ..x.	92	ROM OK
x..x ..xx	93	Scratch Pad OK
x..x .x..	94	Data Buffer Memory OK
x..x .x.x	95	Indexer OK
x..x .xx.	96	Interrupt Controllers OK
x..x .xxx	97	USARTs Initialized
x..x x...	98	DMA Controller Initialized
x..x x..x	99	Comparator Initialized
x..x xxxx	9F	Initialization Complete

5.2.3 LED panic codes

After the self-test completes, and the MTI-850/1650A is performing normal communications between the host and serial devices, the LEDs may display an error pattern if the 8085 encounters an unrecoverable error, or "panic" condition.

Pattern	Hex	Meaning
x.xx ...x	B1	Bad 8085 Interrupt
x.xx ..x.	B2	Bad DMA Data
x.xx ..xx	B3	DMA Busy On Level 6
x.xx .x..	B4	VRFIFO packet length too big
x.xx .x.x	B5	VRFIFO packet contains incomplete response
x.xx .xx.	B6	Needs Service bit set, but service not possible
x.xx .xxx	B7	Needs Service bit cleared during test
x.xx x...	B8	VRFIFO full after response FIFO service

If one of these conditions occurs, call Systech for further instructions. See **Section 5.4** for instructions on contacting the customer service department.

5.3 DIAGNOSTIC MODE

To more thoroughly test the MTI-850/1650A's operation, you can use its special diagnostic mode. Activate the diagnostic mode by flipping the toggle switch on the edge of the controller board from its NORMAL position to its DIAGNOSTIC position. (Refer to **Figure 1-2** for the position of this switch on the board.) When the switch is flipped, the MTI-850/1650A's firmware will stop responding to the Multibus (and therefore be out of communication with the host computer). It will then make its diagnostic capabilities available to you through one of its communications ports.

5.3.1 Automatic port selection

The first thing the firmware does in diagnostic mode is to test the USART chips associated with each port and locate the lowest number port that is operational. The firmware acknowledges that it is operating in diagnostic mode by displaying a blinking hexadecimal "D" (the pattern xx.x) on the left bank of LEDs. On the right bank appears the number of the port selected:

.... = 0	.x.. = 4	x... = 8	xx.. = 12
...x = 1	.x.x = 5	x..x = 9	xx.x = 13
..x. = 2	.xx. = 6	x.x. = 10	xxx. = 14
..xx = 3	.xxx = 7	x.xx = 11	xxxx = 15

Plug a terminal into the indicated port to continue in diagnostic mode. This terminal becomes the diagnostic operator's terminal.

5.3.2 Diagnostic options

At this point you will see the diagnostic menu on your screen:

```
Systech Corporation MTI-850/1650
On-board diagnostic monitor V1.0H

E - Echo test
P - Pattern test
X - Exit diagnostic monitor

ENTER (E, P, OR X):
```

If you enter any key besides those listed, the menu will automatically redisplay.

5.3.3 Echo Test

When you select the echo test, the diagnostic mode program attempts to "echo" the input from each port back as its output, except for the port connected to the diagnostic operators console. This means you can plug terminals into any one of the other ports and press keys, and the diagnostic program will attempt to "echo" each terminal's keyboard activity to its screen.

All of the standard printable characters echo as is. Control characters appear as two characters: an "up arrow" followed by the letter or symbol which you held down, along with the control key. For example, a "control A" character (ASCII code 01) will echo as a "©A" sequence. The delete key will echo as "[DEL]" and the Break key as "[BREAK]".

If the echo test program encounters a parity error, it displays it as "[P ERROR]"; if it encounters an over-run condition occurs, it displays "[OVER-RUN]".

To exit the echo test, press any key on the diagnostic operator's console and it will return to the diagnostic menu.

5.3.4 Pattern test

When you select the pattern test, the diagnostic program begins attempting to send a test pattern to every port except the one used by the diagnostic operator's console port. This pattern is a sequence of lines that appears something like a barber pole, each with a complete set of printable characters.

If the program finds that any port is not responding (in other words, is not connected to a terminal), it stops sending the pattern to that port. Otherwise, the pattern output continues until you hit any key on the diagnostic operator's console.

5.3.5 Interpreting diagnostic results

If both tests fail on a given terminal, it could mean a break in a cable line, particularly the Transmit Data (TxD) or Signal Ground (SG) line. Alternately, it can mean that your device requires the Request To Send (RTS) line from the MTI-850/1650A to be jumpered directly to the Clear To Send (CTS) line to the MTI. (This latter problem is described in **Appendix A.)**

If the Pattern Test passes but the Echo Test fails, this can indicate that the Received Data (RxD) line is not making it from the terminal to the MTI-850/1650A.

It is rare that the Pattern Test fails but the Echo Test passes. Such an event could be due to a noise problem at higher data rates.

If both tests pass but you cannot communicate with terminals through the host software, this can be due to host software problems, Multibus or host computer hardware problems, or a failure of the MTI-850/1650A's bus circuitry.

5.4 WARRANTY AND REPAIR

If the corrective procedures described in **Section 5.2** and **5.3** fail to resolve the problem, and it appears that the MTI-850/1650A is the source of the difficulty, assemble the information noted below, call Systech Customer Service at the telephone number shown and someone will assist you. Customers outside the United States should contact their Authorized Distributor for service and warranty procedure assistance.

5.4.1 Calling customer service

Before calling the customer service number, you should have the following information available:

a) A detailed description of the problem.

b) Date you received the product.

c) Part number and serial number of the board. These numbers are found on the component side of the circuit board.

d) Your shipping and billing address, should the product need to be returned.

e) If your Systech warranty has expired, you will also need a purchase order number for billing purposes.

5.4.2 Returning product

Should the product need to be returned for repair, the Systech Customer Service person will give you a Return of Material Authorization (RMA) number. The RMA number is valid for 30 days.

PLEASE NOTE: All returns to factory (warranty, non-warranty, and exchanges) require a Return of Material Authorization (RMA) number.

A parcel received without an RMA number noted on the outside of the box will be **refused** by the Receiving Department at Systech and will be **returned** freight collect.

Unless otherwise requested, all Systech shipments within the continental U.S. (except California) are sent U.P.S. Blue. All charges are prepaid and billed.

All Systech shipments within California are sent U.P.S. Ground.

For shipment to international customers, please specify the air carrier and forwarder of your choice. All charges are collect and all shipments are insured unless otherwise requested. For international shipments, we **do not** recommend Air Parcel Post. The FOB (Freight On Board) point is always factory/San Diego.

SYSTECH CUSTOMER SERVICE NUMBER:

(619) 453-8970

For warranty provisions, refer to **Appendix K** at the end of this manual. Boards can be returned for repair (warranty or non-warranty) following the standard or emergency (loaner) procedure detailed below.

5.4.2.1 Factory board repair procedure

a) An RMA number, as described above, is required.

b) Unit must be returned freight and insurance prepaid. See RMA marking requirements in **Section 5.4.2.**

c) Unit will be returned, repaired or exchanged within 10 working days.

d) The return freight will be paid by Systech.

5.4.2.2 Loaner procedure

a) An RMA number, as described above, is required.

b) Unit must be returned immediately, freight and insurance prepaid. See RMA marking requirements in **Section 5.4.2.**

c) A loaner board will be shipped air freight prepaid within one business day.

d) Customer will be invoiced for the loaner board at list price plus a loaner charge of $150. The list price charged for the loaner board will be credited when the loaner is returned to Systech.

APPENDIX A: Communications concepts

APPENDIX A

Communications concepts

This appendix has been included to assist you in determining which of the MTI-850/1650A's many options to employ. If you are installing the product as outlined in **Section 2,** but have questions regarding some of its communications features, this explanation may be of help.

Communications concepts

To help you decide which of the MTI-850/1650A's features to select, it will be helpful to review some fundamentals of serial communications.

Serial communications are so named, of course, because the data travels serially, one bit at a time. The communications may be asynchronous, meaning that there is no fixed interval between characters, and each end of the channel has its own clock for timing the length of bits. Or communications may be synchronous, meaning that the sender and receiver are locked to the same clock, and transmission occurs continually; the sender sends either data or special synchronization characters, so that the receiver always has a timing signal to follow.

Early Teletype machines were the original asynchronous serial communications devices. They used the current loop method of data transfer. The MTI-850/1650A does support this method (with the current loop option installed). However, the current loop method has many options that are not well-defined or standardized, so over time confusion has increased. Also, current loop is inadequate for communications between a terminal and a modem for transmission of data over public telephone or leased lines to remote stations. So beginning in 1963, the Electronic Industry Association (EIA) began issuing standards for communications. The RS-232C standard, defined in 1969, is the most recent and widely used, and is supported by the standard MTI-850/1650A.

Asynchronous current loop

Current loop began as an electromechanical method for Teletype and similar terminals to communicate. Each terminal had a rotary switch that could generate or respond to character patterns, and units were often connected in a loop to operate in tandem. Pressing a key on any one unit in the loop caused them all to print that character. The circuit connecting them all stayed in an "idle" state (usually a binary one condition) until character transfer began. A character consisted of a start bit,

followed by a parity bit, and terminated by 1, 1 1/2 or 2 stop bits, after which the line returned to its idle state. This was all simple and straightforward, except for inconsistencies in the electrical characteristics of the current loop circuit. Refer to Appendix C for details of the MTI-850/1650A's support of current loop options.

The RS-232C standard

The RS-232C standard defines the electrical characteristics and protocol for communications between Data Terminal Equipment (DTE), usually a terminal or computer, and Data Communications Equipment (DCE), meaning a modem. Since no comparable widely used standard exists for connecting a computer directly to a terminal, the RS-232C standard is often used for this purpose as well; in this case one end of the channel has to pretend to be the modem.

Actually, the RS-232C standard defines several possible interfaces between a DTE and a DCE. In any application, devices use only a subset of the complete standard, depending on their capabilities and requirements. For example, all devices require these circuits:

Signal Name	Abbr.	DCE	DTE
Transmitted Data	(TxD)	IN	OUT
Received Data	(RxD)	OUT	IN
Signal Ground	(SG)	-- common --	

which carry the actual transfer of data. Most devices also require:

Request To Send	(RTS)	IN	OUT
Clear To Send	(CTS)	OUT	IN

and some require:

Data Set Ready	(DSR)	OUT	IN
Data Terminal Ready	(DTR)	IN	OUT

Only when a modem is actually being used does the channel need:

Signal Name	Abbr.	DCE	DTE
Data Carrier Detect	(DCD)	OUT	IN

---

and possibly:

Received Clock	(RxC)	OUT	IN
Transmitter Clock	(TxC)	OUT	IN
		IN	OUT

There are also secondary communications lines needed by some error-reporting code schemes, but the USART chips do not support these lines, so neither does the MTI-850/1650A and other recent RS-232C devices which use the USARTs.

When connecting two devices which are RS-232, it is necessary to know which subset lines each requires. For maximum versatility, the MTI-850/1650A provides all the lines listed above, but requires only these inputs:

In DTE Emulation Mode (connected to modem)	In DCE Emulation Mode (connected to terminal)
Received Data (RxD)	Transmitted Data (TxD)
Signal Ground (SG)	Signal Ground (SG)
Clear To Send (CTS)	Request To Send (RTS)

The CTS signal is used by some printers and other hard-copy devices which operate as DCEs. This allows the printer to stop the flow of data from the DTE device when needed (for example, during a carriage return).

NOTE: Some DCE devices do not provide a CTS signal. If you connect such a device to a channel of the MTI-850/1650A (operating as a DTE), you must specially wire the cable from the device to the MTI-850/1650A. The RTS signal out from the MTI-850/1650A must "loop back" to its CTS connection in.

Synchronous communications

In synchronous communications, the transmitter and receiver share a clock, and characters always begin at regular intervals triggered by the clock. If the sender has no characters to send, it maintains synchronization by sending special synchronization characters.

APPENDIX A

Communications concepts

There are many protocols for synchronous communications. The MTI-850/1650A does not provide any specific synchronous protocol handling. In order to use the MTI-850/1650A with synchronous devices, the host software must do all protocol handling. The user may need to install special jumpers on the USART board as well. This is explained in **Section 2;** additional information is in **Appendix D.**

Appendix B: Cable construction

APPENDIX B

Cable diagrams

This appendix is to assist you in building or rewiring any of four types of cables used with the MTI-850/1650A: the external 25-pin RS-232 cable connecting the MTI to external devices (below); the internal 50-pin ribbon cable connecting the controller board to the I/O device (page B-4); the 50-pin ribbon cable wired for current loop (page B-5); and the 5-pin power cable (page B-5 also).

RS-232 cable instructions

NOTE: Because the chassis panel has jumper blocks that allow you to "swap" terminal (DTE) and modem (DCE) signals, you can wire all serial device cables the same way without knowing ahead of time what you will connect to a given port.

Pin #		Signal Name	Direction: When MTI is DCE	Direction: When MTI is DTE
1	(AA)	Frame Ground (FG)	n/a	n/a
2	(BA)	Transmitted Data (TxD)	IN	OUT
3	(BB)	Received Data (RxD)	OUT	IN
4	(CA)	Request To Send (RTS)	IN	OUT
5	(CB)	Clear To Send (CTS)	OUT	IN
6	(CC)	Data Set Ready (DSR)	OUT	IN
7	(AB)	Signal Ground (SG)	n/a	n/a
8	(CF)	Data Carrier Detect (DCD)	OUT	IN
9		not used		
10		not used		
11		not used		
12	(SCF)	not used		
13	(SCB)	not used		
14	(SBA)	not used		
15	(DB)	Transmitter Clock (TxC)	OUT*	IN*
16	(SBB)	not used		
17	(DD)	Receiver Clock (RxC)	OUT*	IN*
18		not used		
19	(SCA)	not used		
20	(CD)	Data Terminal Ready (DTR)	IN	OUT
21	(CG)	not used		
22	(CE)	not used		
23		not used		
24	(CH)	not used		
25	(DA)	not used		

APPENDIX B

Cable diagrams

---

* The directions of these signals are independently configurable with jumper plugs. The 266ls must also be specifically programmed to use these signals (see Appendix J, Table 6).

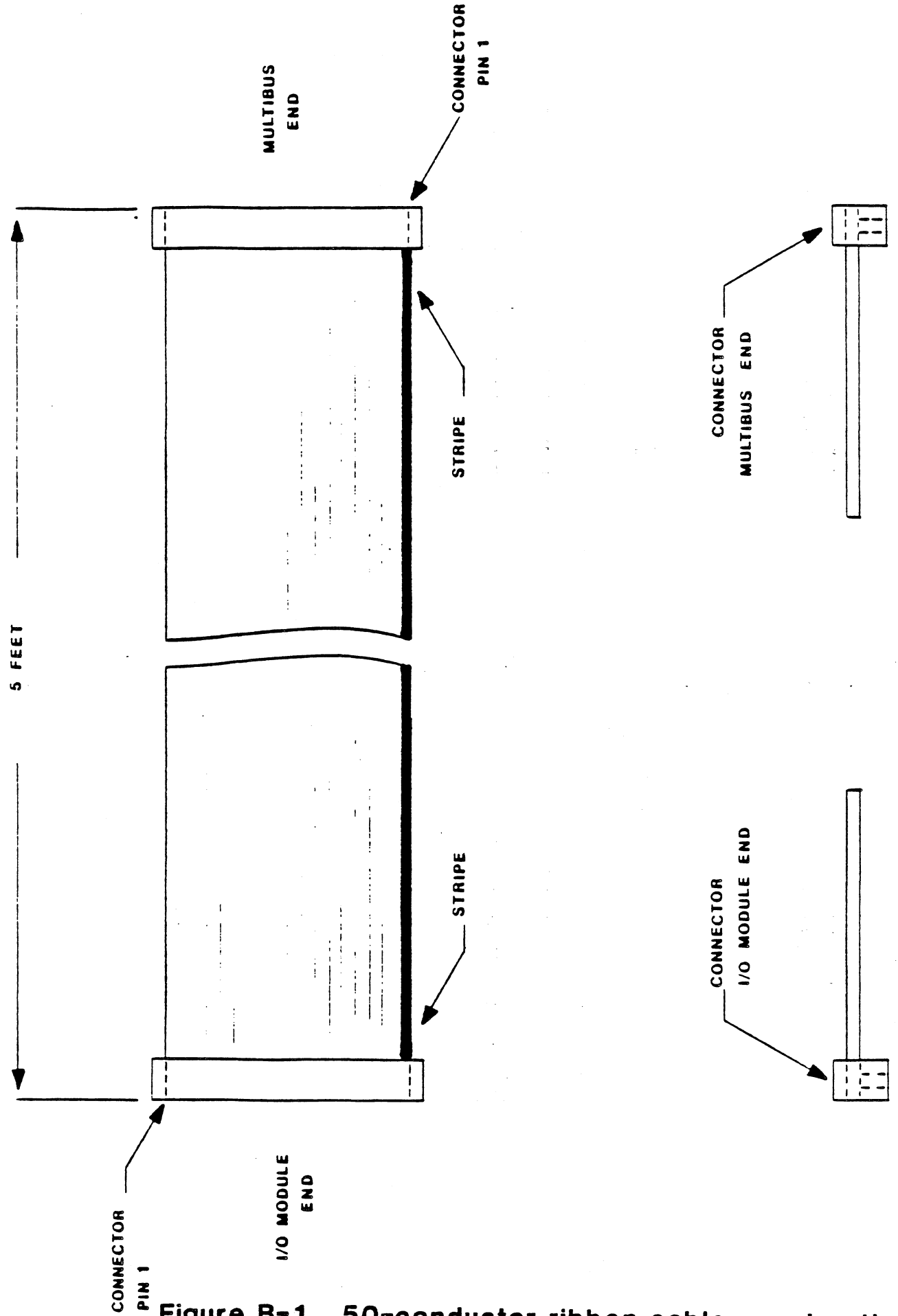


Figure B-1 50-conductor ribbon cable construction

APPENDIX B

Cable diagrams

Current loop cable instructions

NOTE: The chart on this page shows the signals associated with current loop operation. When constructing cables, refer to this information **and** to the specifications for your peripheral hardware, since manufacturers vary in their connector pinouts. Be sure to use wire specified for 20 mA of current. If you use external voltages, the wire for the +12 and -12 volts must support 40 mA of current. The parts list for constructing current loop cables is the same as the standard RS-232 cable.

SINGLE LOOP PINOUTS

DB-25 PIN NO.	EXTERNAL CURRENT SOURCE	EXTERNAL VOLTAGE SOURCE	INTERNAL CURRENT SOURCE
9	IN (+)		
10	OUT (-)		
11		+12 V	
12			
13		OUT (+)	OUT (+)
23		-12 V	
24		IN (-)	IN (-)
25			

DOUBLE LOOP PINOUTS

DB-25 pin No.	External current source	External voltage source	Internal current source
9	TxD IN (+)		
10	RxD OUT (-)		
11		+12 V	
12		RxD OUT (+)	RxD OUT (+)
13	TxD OUT (-)	TxD OUT (+)	TxD OUT (+)
23		-12 V	
24	RxD IN (+)	RxD IN (-)	RxD IN (-)
25		TxD IN (-)	TxD IN (-)

APPENDIX B

Cable diagrams

Internal ribbon cable instructions (J1 connector)

PIN	CONNECT	PIN	CONNECT
1	+5 volts	26	BA1
2	BA5	27	-12 volts
3	+5 volts	28	BA2
4	BA6	29	BA3
5	+5 volts	30	BA4
6	BA7	31	CS3A*RD+WR /
7	GND	32	CS3B*RD+WR /
8	BA0	33	BINTA/
9	GND	34	BRD/
10	BD7	35	GND
11	GND	36	BWR/
12	BD6	37	INT3
13	GND	38	CS3F/
14	BD5	39	INT2
15	GND	40	CS3E/
16	BD4	41	INT1
17	GND	42	CS3D/
18	BD3	43	INT0
19	GND	44	CS3C/
20	BD2	45	GND
21	+12 volts	46	CAS2
22	BD1	47	GND
23	+12 volts	48	CAS1
24	BD0	49	GND
25	-12 volts	50	CAS0

J2 connector (5-pin power cable)

1	GND
2	GND
3	+5
4	-12
5	+12

Appendix C: Using the current loop option

APPENDIX C

Using the current loop option

The standard MTI-850/1650A is configured to operate with RS-232C serial devices. As an option, a 20 ma **current loop** configuration is offered for customers who will be using current loop devices.

The basic current loop circuit consists of a receive isolator, a transmit isolator, and a current source. An assembly that has the current loop option can be identified by the row of switches on the assembly. Each set of switches corresponds to an I/O channel. You will use these switches to configure the current loop to your specific needs.

Figure C-1 shows a USART assembly with the current loop option installed. The box labeled "1" is the I/O port connector; "2" and "3" are the jumper blocks used to select RS-232 or current loop transmission for that port. The assembly comes with jumper plugs for insertion into the jumper blocks. If the jumper plug is in the left jumper block ("2"), then the assembly is configured for current loop operation. If the jumper plug is in the right jumper block ("3"), the port is configured for RS-232C transmission. The box labeled "4" represents the switches used to configure the "style" of current loop operation for the port.

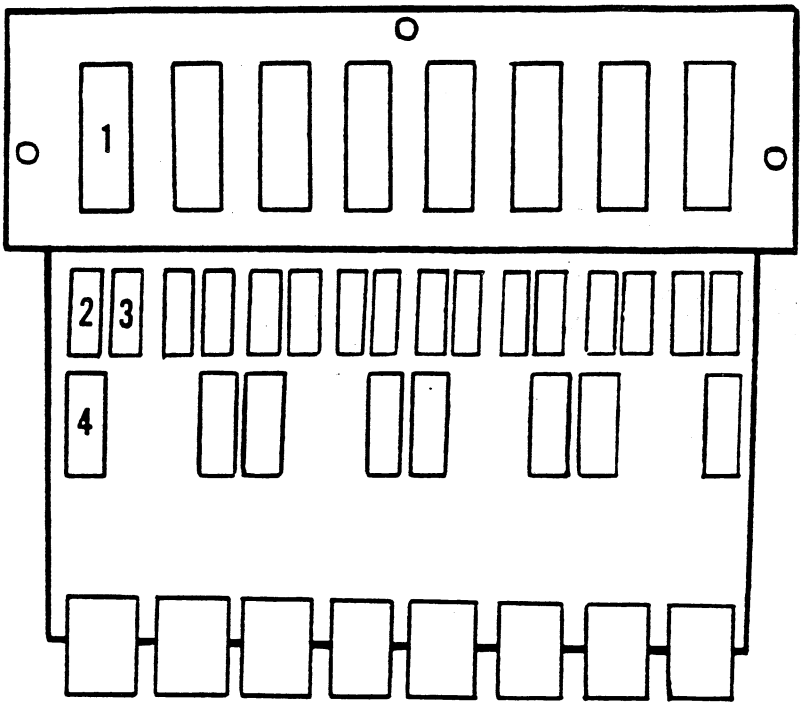


Figure C-1 USART assembly with current loop

APPENDIX C

Using the current loop option

Current loop can be configured in six different styles, as shown in the table on this page. It can use either a single or double loop method. The single loop can have an internal current source (Figure C-2), an external current source (Figure C-3), or an external voltage source (Figure C-4). The double loop configuration, like the single loop, may have an internal current source (Figure C-5), an external current source (Figure C-6) or an external voltage source (Figure C-7).

The MTI-850/1650A can be configured in any of the six ways. You should configure your assembly to conform to the type of loop configuration used by the peripheral you will be communicating with. Configure the I/O assembly by using the diagrams in Figures C-2 through C-7. Configure the MTI assembly to the proper loop style by using the switch settings shown on the following page.

Current loop configurations

I. Single loop configurations

A. Internal current source	Figure C-2
B. External source	
1. External current source	Figure C-3
2. External voltage source	Figure C-4

II. Double loop configurations

A. Internal current source	Figure C-5
B. External source	
1. External current source	Figure C-6
2. External voltage source	Figure C-7

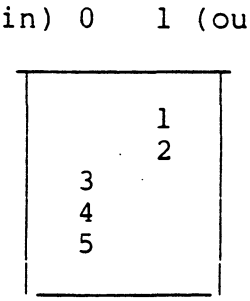
APPENDIX C

Using the current loop option

Current loop switch settings

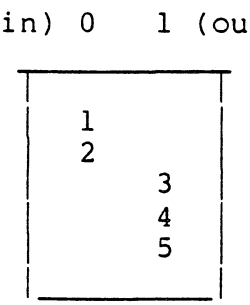
If you have the current loop option installed, these are the switch settings you use on the DIP switches mounted on the chassis panel, one per channel:

External voltage source (+12V on pin 11, -12V on pin 23, uses internal resistors):



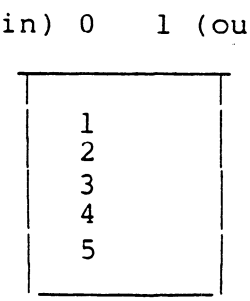
IN	OUT
24	13

External current source (doesn't use internal resistors):



IN	OUT
9	10

Internal current source:



IN	OUT
24	13

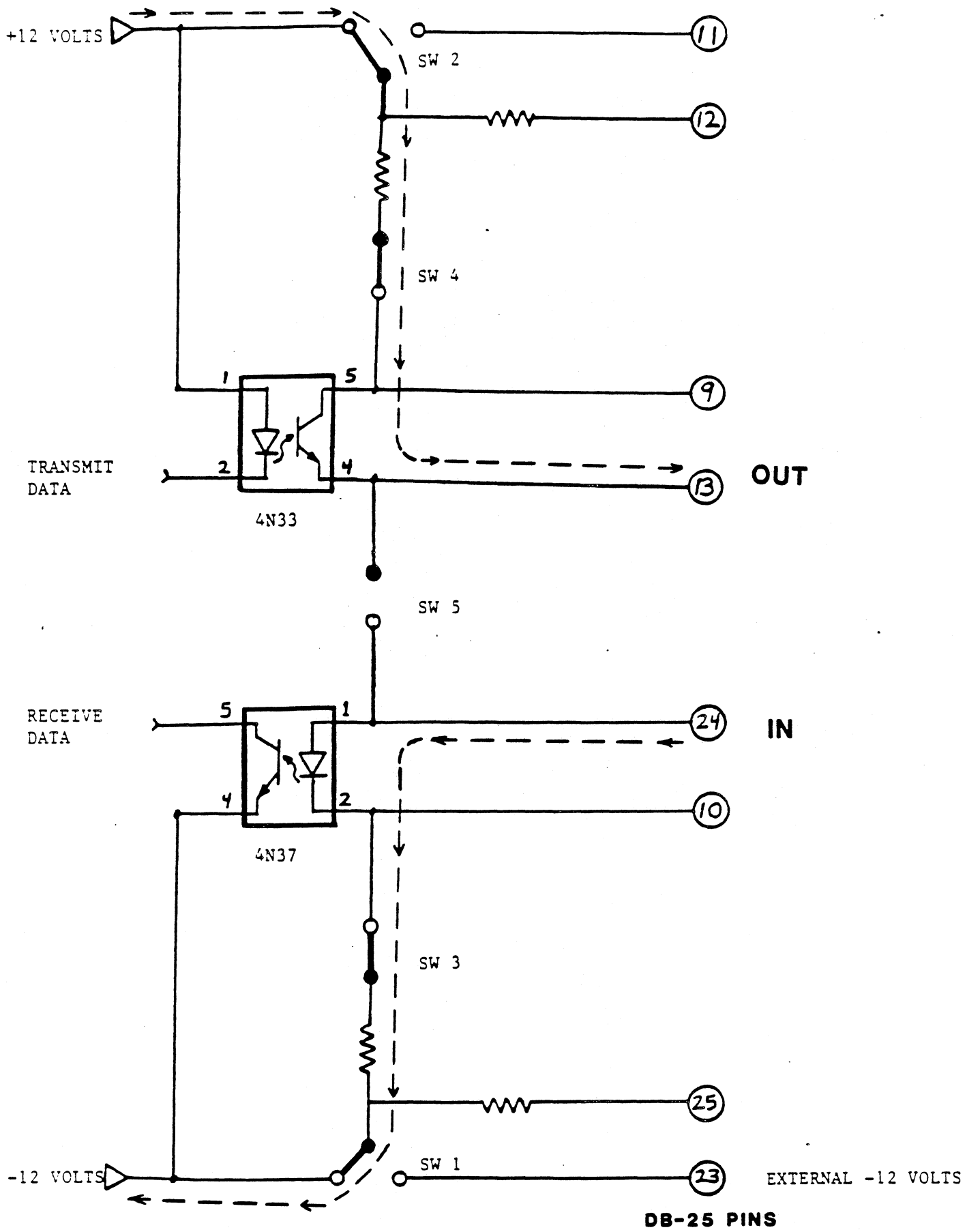


Figure C-2 Single loop internal current source

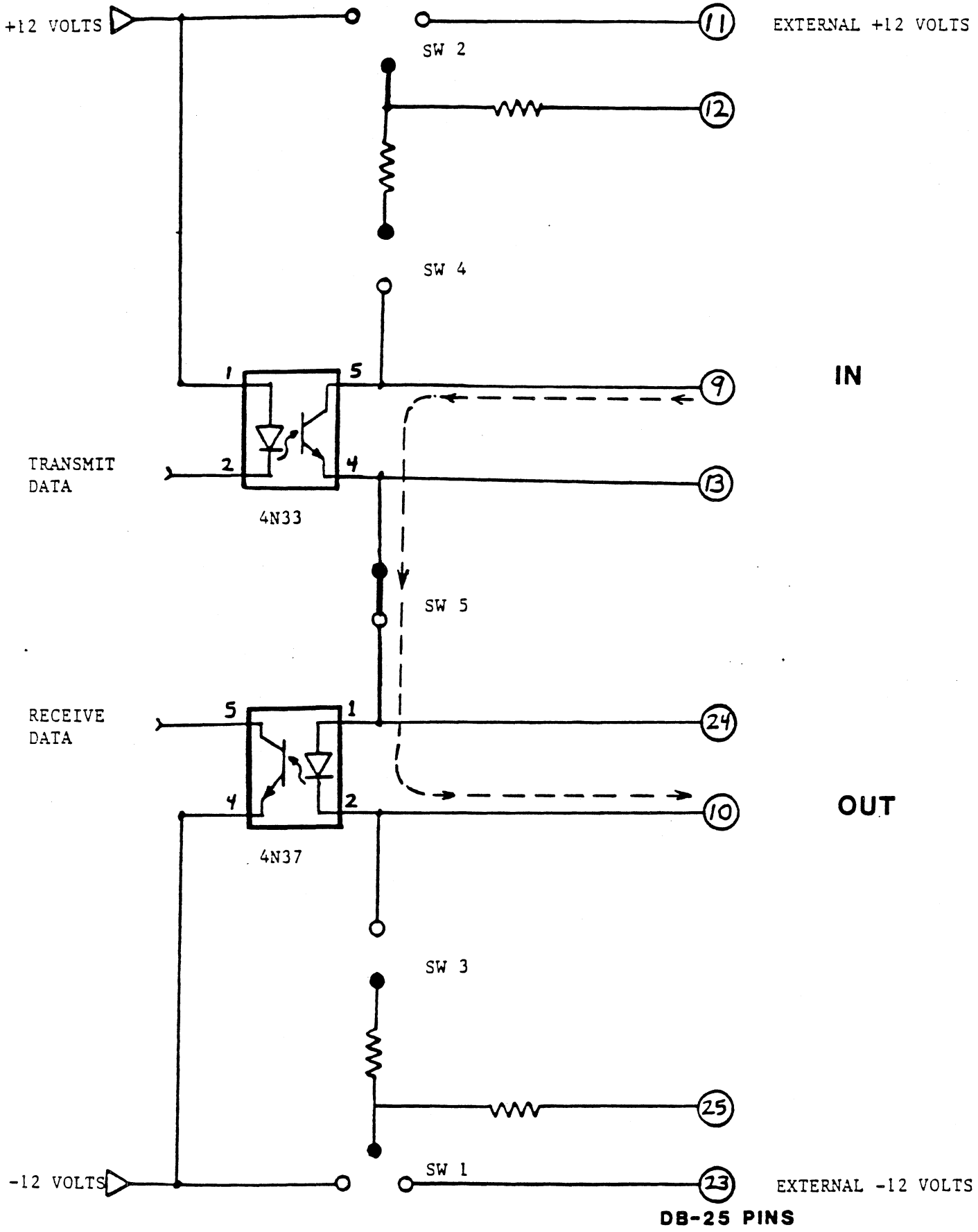


Figure C-3 Single loop external current source

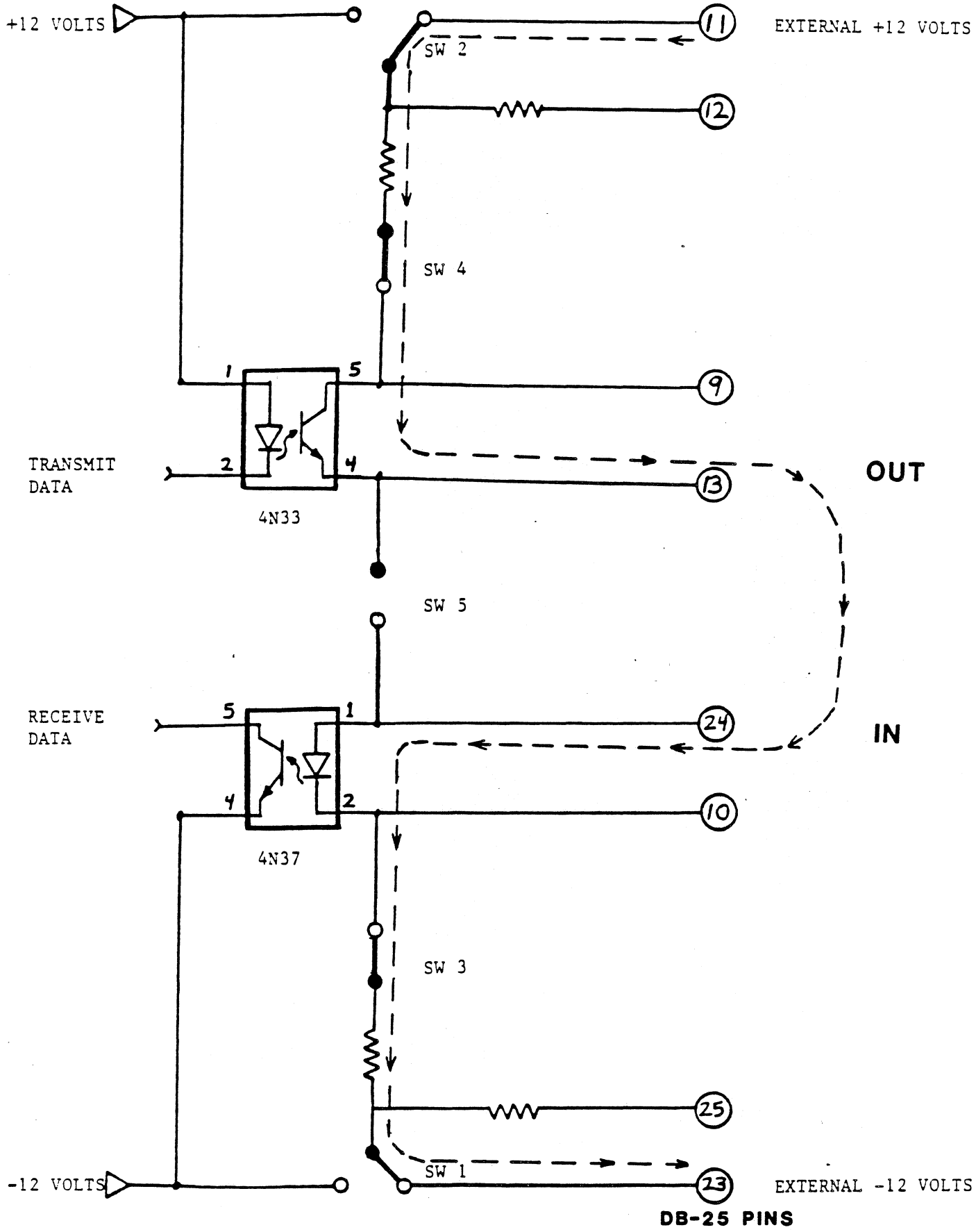


Figure C-4 Single loop external voltage source

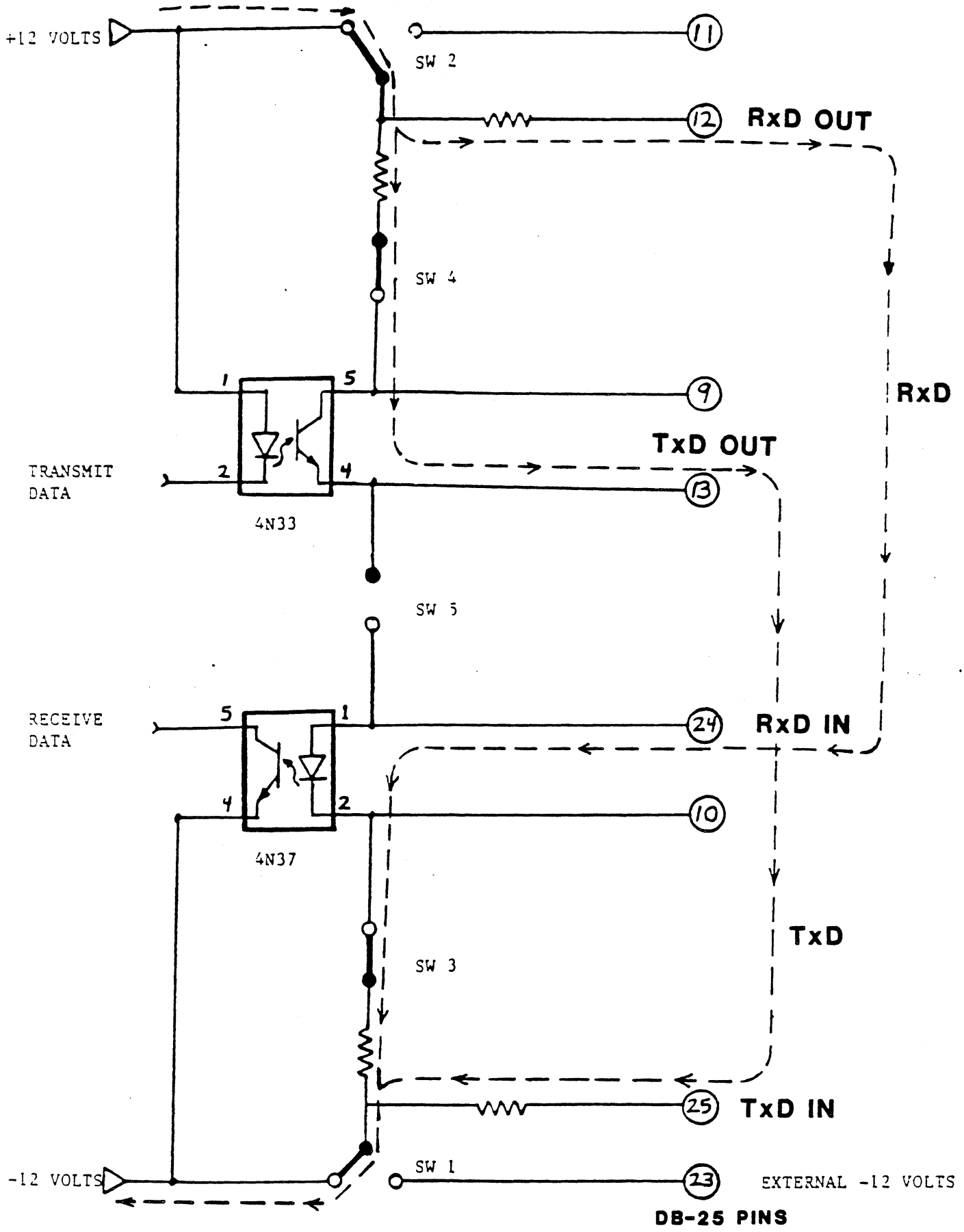


Figure C-5 Double loop internal voltage/current source

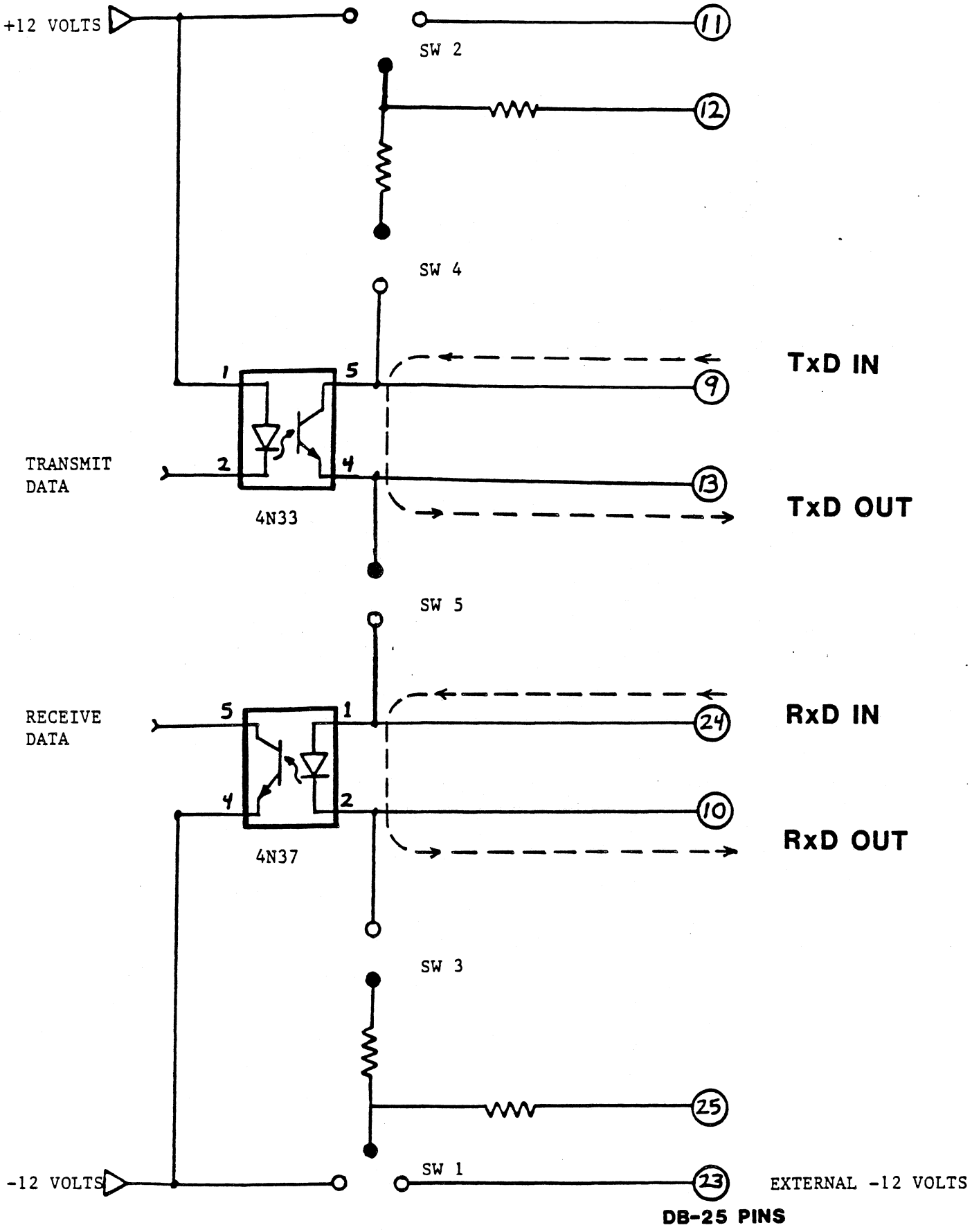


Figure C-6 Double loop external current source

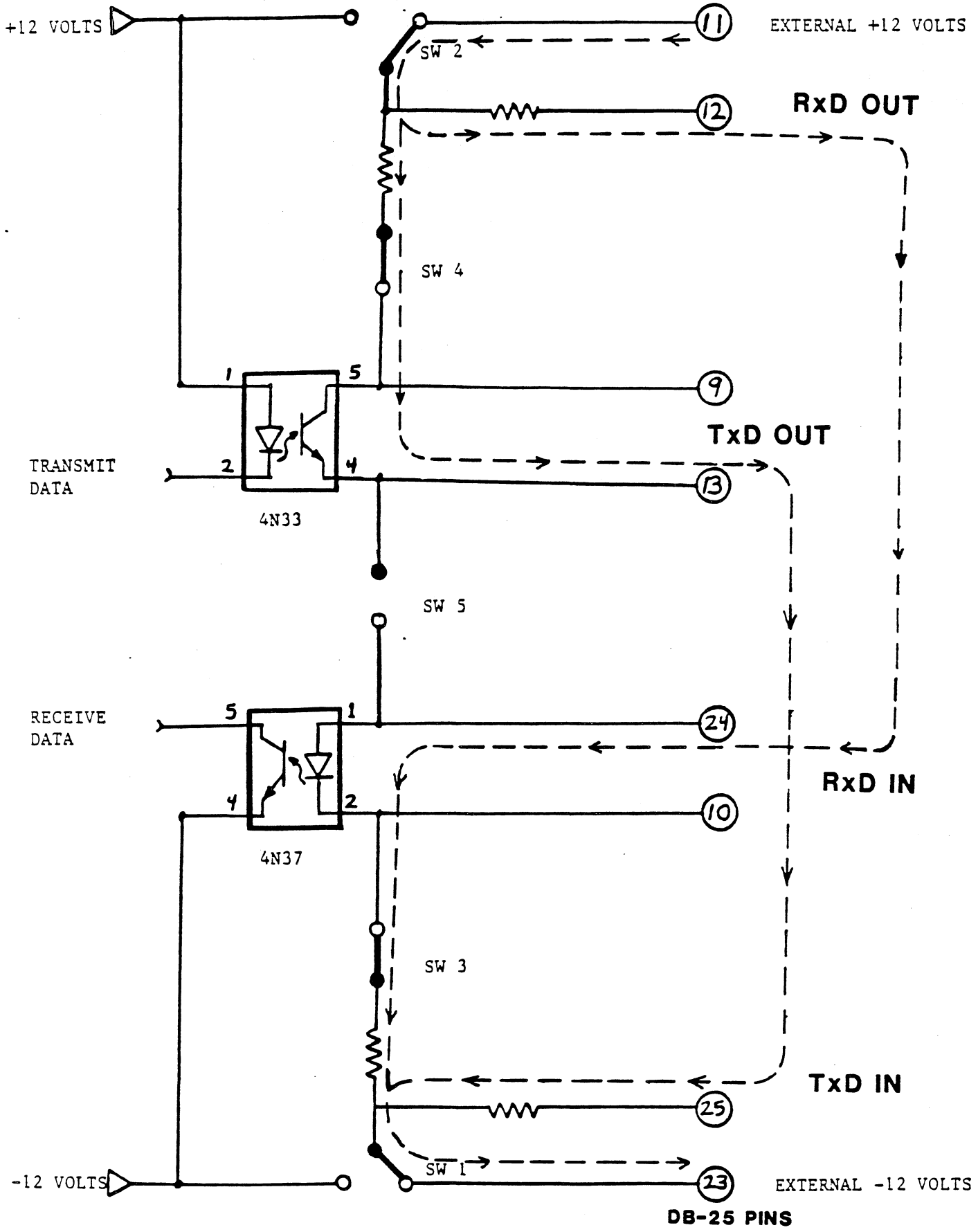


Figure C-7 Double loop external voltage source

Appendix D: Accessing the USARTs directly

APPENDIX D

Accessing the USARTs directly

For synchronous communications or special applications, you may want to have the host software directly program the Universal Synchronous/Asynchronous Receiver/Transmitter circuits. **Section 3** (Programming) gives the software commands for reading from and writing to the USARTs. **Appendix F** contains the Signetics specification for the 2661 chip used in the MTI-850/1650A. You may also, however, need a way to get certain signals from a USARTs to a cable conductor.

For synchronous operation, jumpers need to be installed for enabling and setting the direction of the transmit and receive clocks. For this purpose, we provide jumper sockets on the USART board. Refer to **Section 2.2.2** for information on setting these jumpers.

Section 2 (Configuration and Installation) has more detailed information on locating these jumper blocks.

NOTE: When programming the USARTs directly from host software, you need to know that every time the MTI-850/1650A firmware reads a character from a USART, it rewrites the complete command register to clear any possible errors, which will put the USART back in the state the firmware last configured it to, and possibly affect the state of Request to Send (RTS).

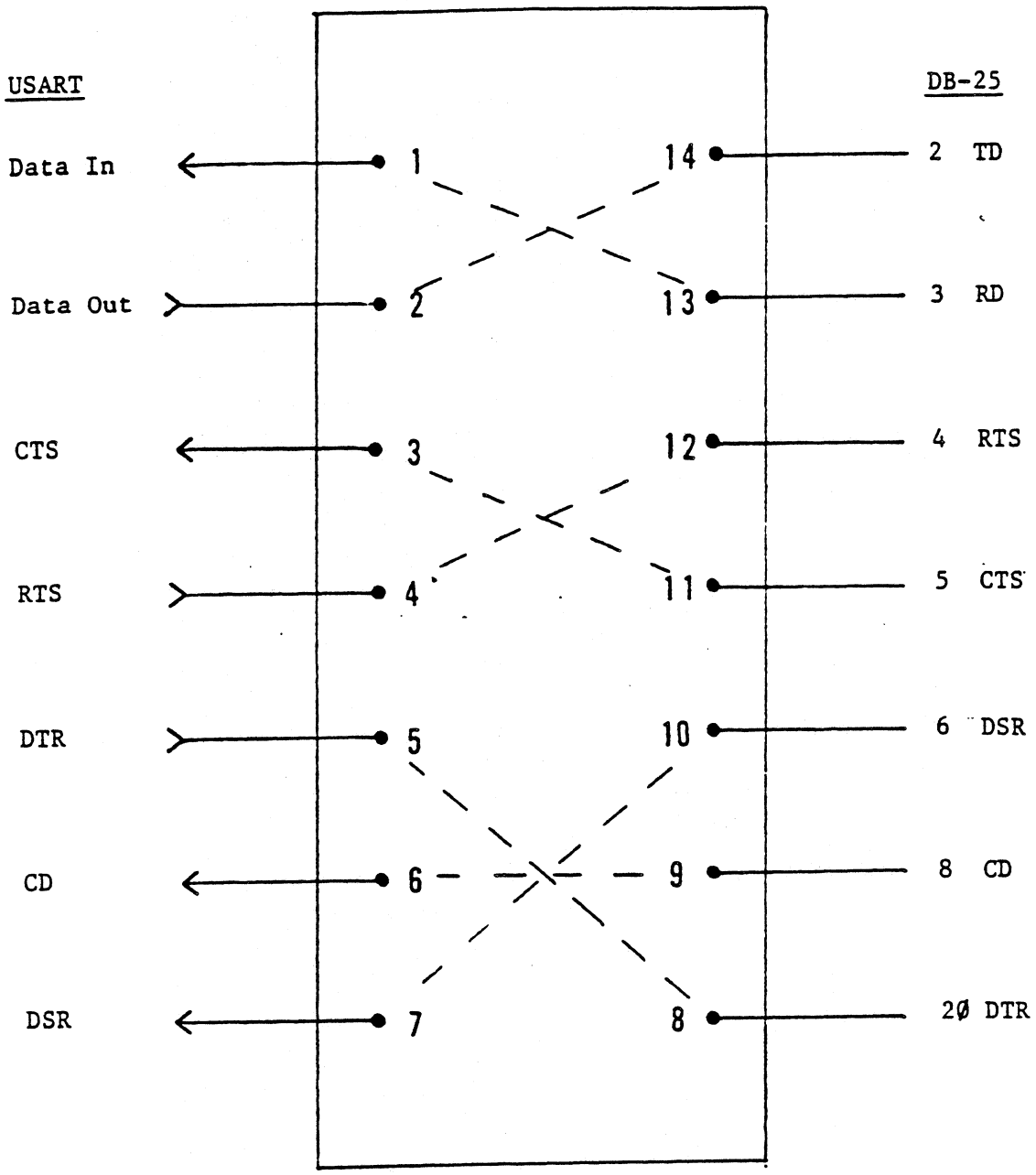


Figure D-1 TO MODEM (RS-232) jumper plug

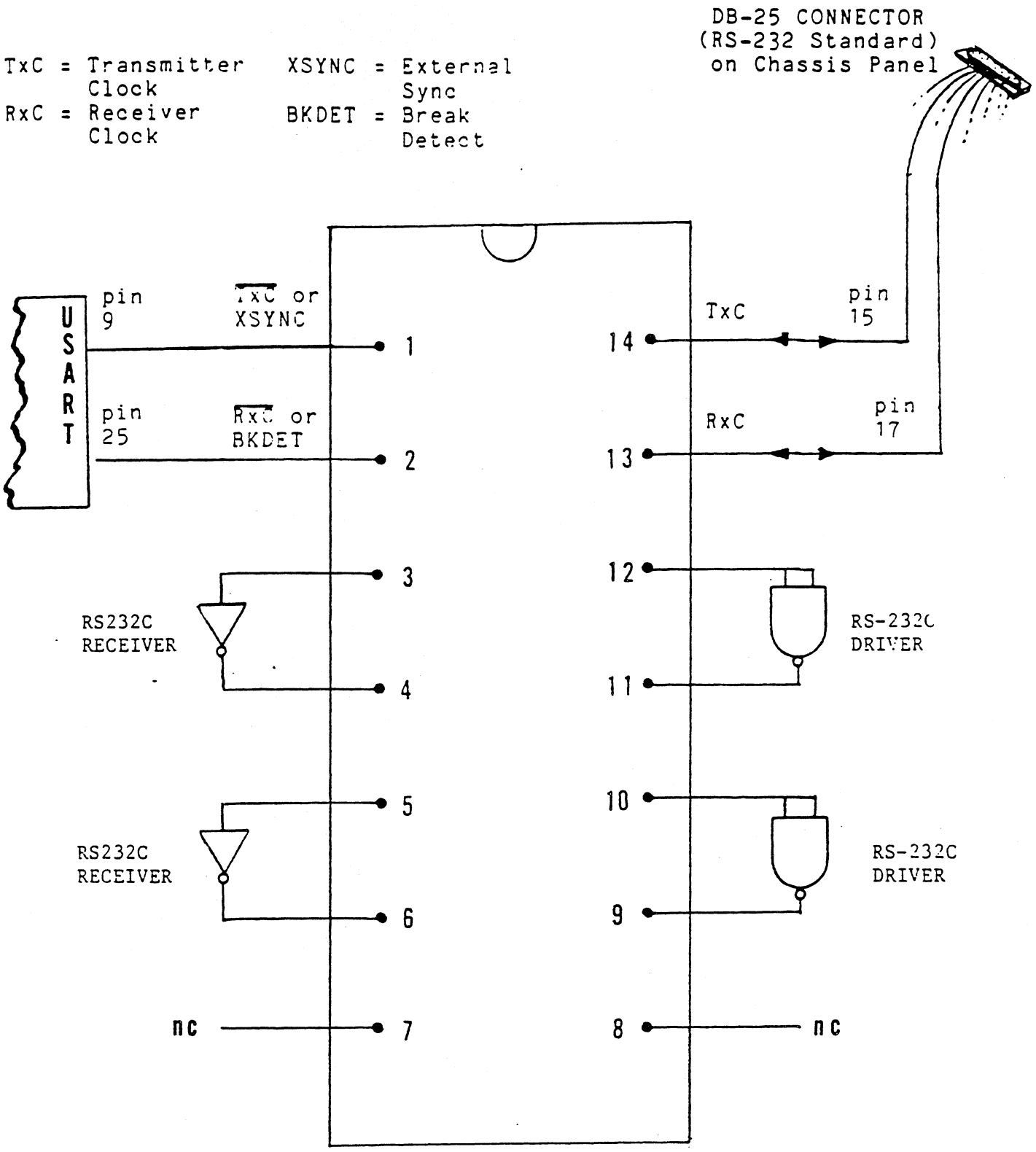


Figure D-2 Synchronous signals jumper block

Appendix E: Jumpers for +12/-12 volts

APPENDIX E

Jumpers for +12/-12 volts

To accommodate those users who need +12vdc to be available through the MTI-850/1650A's ribbon cable, Systech has included a +12/-12 volt jumper-selectable option. The jumper pads, shown in **Figure E-1,** allow you to install jumpers at locations W1 and W2 (near connector J1 on the board). Installing these jumpers configures the board to supply 12 volts of power through the 50-pin ribbon cable.

The standard MTI-850/1650A is shipped with this option disconnected.

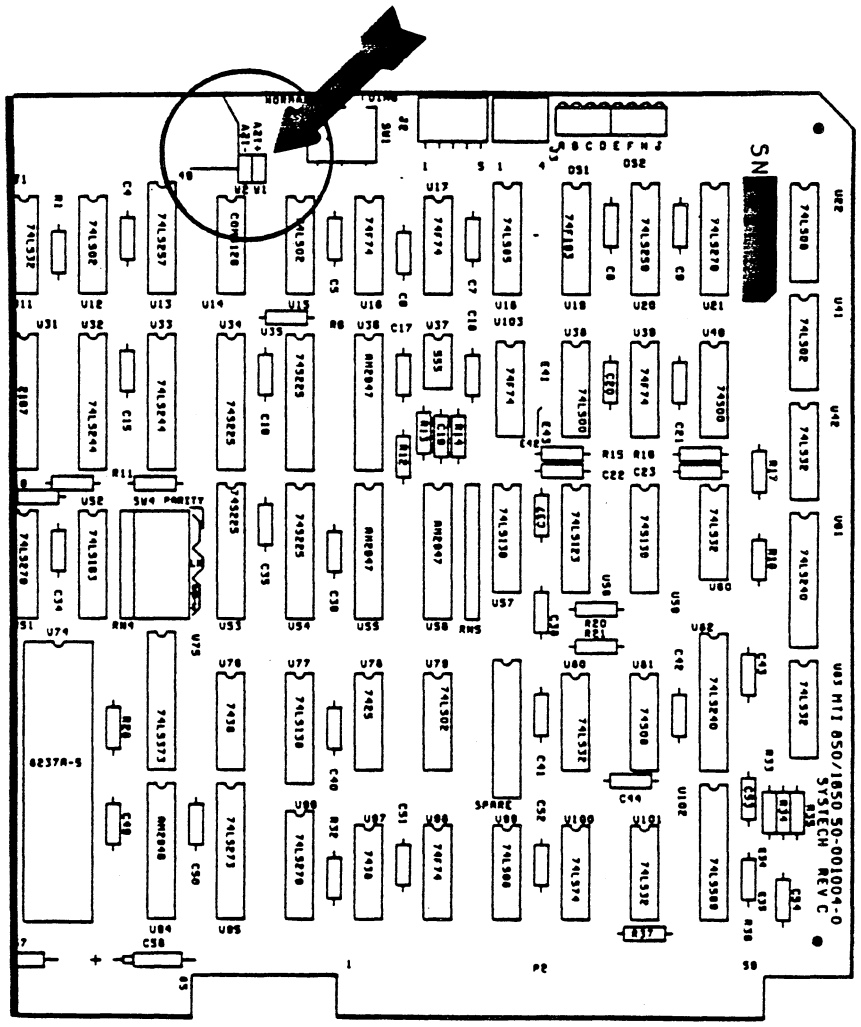


Figure E-1 Portion of controller board showing location of +12/-12 volt jumpers

Appendix F: FCC information

APPENDIX F

FCC Information

The Federal Communications Commission has adopted rules (47 CFR 15, SUBPART J) which impose limits on EMI (Electromagnetic Interference) and RFI (Radio Frequency Interference) emanating from digital electronic equipment. Individual circuit cards such as Multibus PCB's are considered to be a subassembly of a completed mainframe system.

Compliance with these regulations is the responsibility of the mainframe system manufacturer or system integrator. To enable compliance with these rules, this product should be installed in a properly shielded enclosure and interconnected to its intended peripherals with properly shielded (and terminated) cables.

APPENDIX G: Notes for SUN 68000 processor

APPENDIX G

Notes for SUN 68000 processor, and derivatives

Two areas of processor design impact Multibus controllers and their driver software. These are considerations with any processor, but in Systech's experience, they come up primarily with boards based on the SUN Microsystems 68000 design.

Byte-swap

Intel Corp., designing the Multibus specification, specified the following byte-within-word ordering:

word 0: MSB !__byte 1_____byte 0__! LSB

Motorola Inc., designing the 68000 microprocessor, specified this byte-within-word ordering:

word 0: MSB !__byte 0_____byte 1__! LSB

SUN Microsystems, designing their 68000 processor board, chose to maintain the 68000 byte order in Multibus memory. This has two results. First, the memory-mapped I/O addresses of MTI-850/1650A registers must be "twisted," relative to the addresses shown in the manual. Thus, address 00 becomes 01; 01 becomes 00; 02 becomes 03; 03 becomes 02; etc. The second result, which may be more serious, is that bytes sent to the printer/plotter will be written in "twisted" fashion. Thus, if a memory buffer has been defined as

```
char buf[512] = "ABCDEF . . .";
```

the bytes sent to the printer/plotter (when raw output is used) will be

B A D C F E . . .

To eliminate programming overhead associated with these problems, Systech has included a "byte-swap" option on the MTI-850/1650A to accomodate byte-swapped CPUs. If your processor does not conform to standard Multibus byte ordering, then you will have to cut the etch between pads E1 and E2, and connect a jumper between pads E2 and E3. (Refer to **Section 2.2.2.5** and **Figure 2-1** for details.) This will adjust both the mapping of the MTI-850/1650A registers into your processor's memory space, and also the ordering by which the MTI-850/1650A will access data from your memory.

APPENDIX G

Notes for SUN 68000 processor, and derivatives

Memory access restrictions

Typically, CPU manufacturers have included significant amounts of memory on their processor cards. On some designs, this memory is not accessible to devices (like the MTI-850/1650A) on the Multibus. On these systems, special buffer areas have to be set aside in Multibus memory. Data must be shuffled through these buffers when its ultimate source or destination is in the CPU's on-board memory. Thus, driver routines must be aware of what memory is not "DMA-able," and invoke appropriate shuffling when required.

Appendix H: Upgrade instructions - MTI-850A to MTI-1650A

APPENDIX H

Instructions for field upgrade of MTI-850A to MTI-1650A

NOTE: The following upgrade instructions apply to MTI-850/1650A connector boards for Rev. A and later ONLY.

Turn off power to the system and remove cables from the MTI controller board (see MTI-850/1650A Technical Manual, Sec. 2.4.2). Free the cables from any cable clamps or ties, and remove the chassis panel from the rack it is mounted in by removing the screws holding it in place.

Place the assembly face up on a bench or desk. Referring to **Figure H-1** for guidance:

1) Remove the four screws labeled **C, D, G** and **H.**

2) Carefully lift the top of the chassis panel box, and remove the power cable from the board inside.

3) Remove screws **A, B, E** and **F,** and remove the cover panels, exposing the inside board.

Now, referring to **Figure H-2** for guidance:

4) Remove SIP from location RP4 and discard.

5) Install the replacement SIP supplied by Systech in its place, so that the orange dot is oriented away from the "RP4" silkscreen label.

6) Slide the new circuit assembly supplied by Systech through the front, flex jumpers first.

7) Plug the free end of the short cable into the old card connector, and install screws **A** and **B.**

Now re-assemble the chassis panel:

8) Install the lower panel cover ("PORT 8" through "PORT 15"), and install screws **E** and **F.**

9) Align assembly over bottom and re-install

---

power cable, observing polarity.

10) Install screws **C, D, G** and **H.**

11) Mount I/O assembly as before, and with power off, reconnect cable ends at controller.

When all is complete, run self-test on the new channels to see that they are functioning properly.

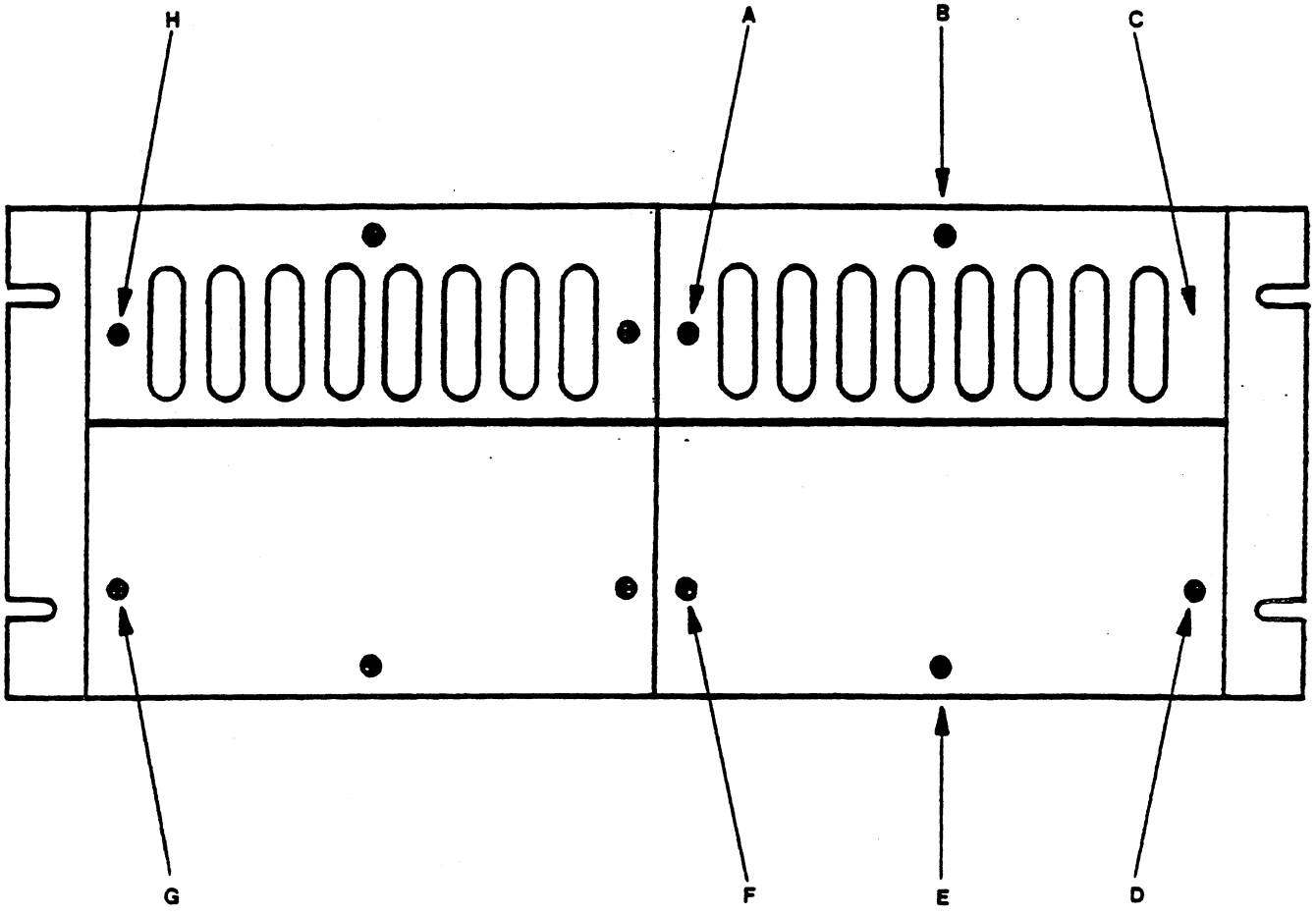


Figure H-1 Cover panel to I/O assembly

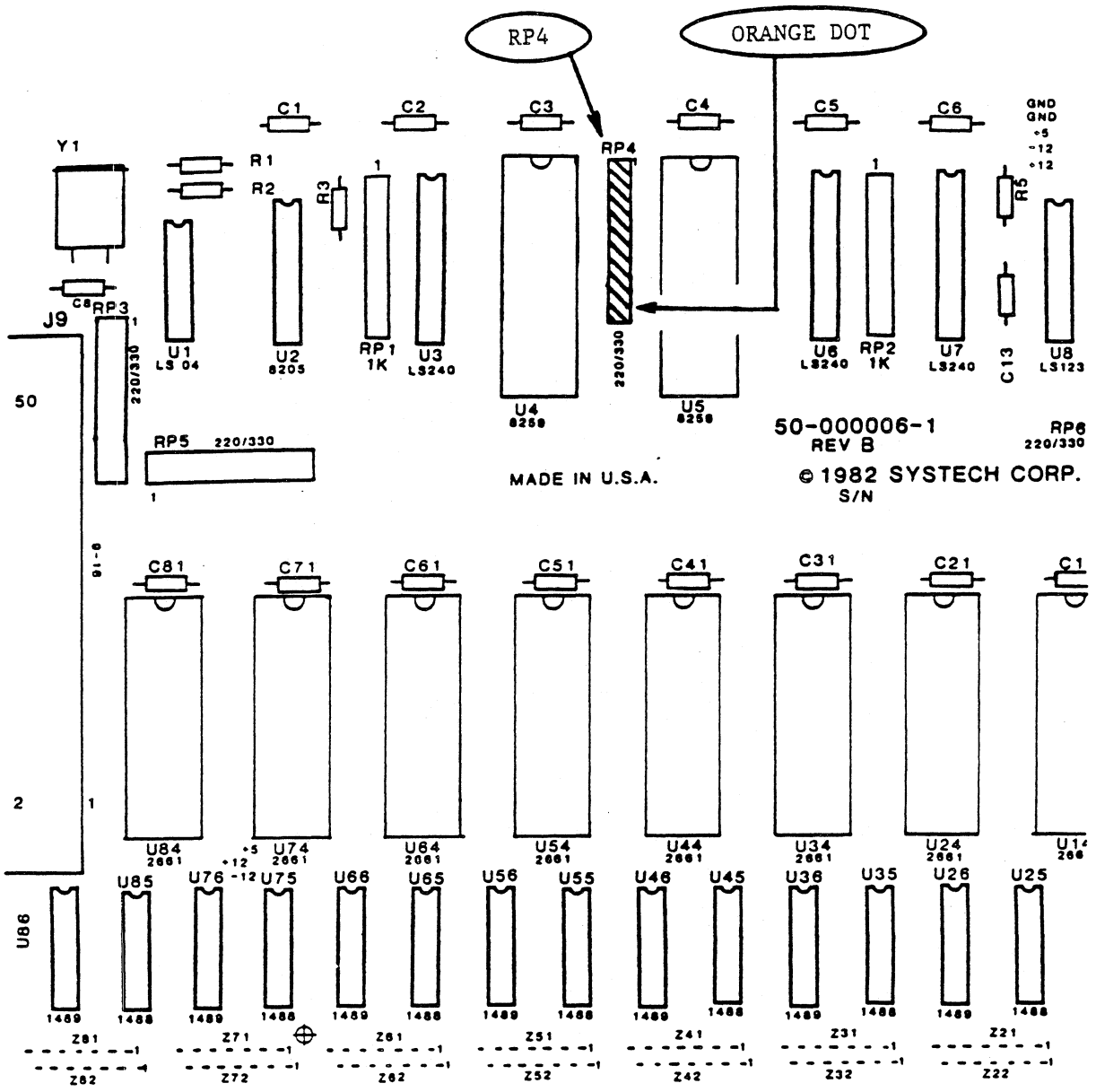


Figure H-2 Connector board showing SIP exchange

Appendix I: Response control capacitors

Appendix I

Response Control Capacitors

This appendix describes adding capacitors to the MTI-850/1650A to control the slew rate of the driver, as may be required to meet RS-232 and CCITT, V.24 specifications. It also describes adding capacitors to filter high frequency noise pulses.

Overview

The MTI-850/1650A has been designed so that capacitors may be added to control the slew rate of the driver in order to meet RS-232 and CCITT recommendations. The capacitors, in the form of discrete components, are added to the outputs of the 1488s. Since the selection of these capacitors is influenced by the length and type of serial cable used, the value of the capacitors needs to be selected by the customer. Capacitors can also be added to the response control lines of the 1489s to filter any potential incoming high frequency noise pulses.

With the advent of the more modern controllers in use today, these filter capacitors are not normally used. However, they can be installed to satisfy particular applications.

Appendix I

Response Control Capacitors

MC1488

QUAD MDTL LINE DRIVER RS-232C

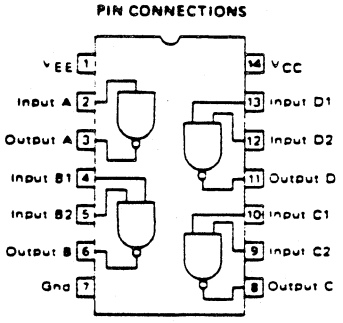


Figure I-1

Controlling the slew rate

Recommendations for RS-232C state that during transitions, the driver output slew rate should not exceed 30 volts per microsecond. The inherent slew rate of an unloaded 1488 driver (see **Figure I-1** is much faster, and if needed the rate can be controlled by connecting a capacitor to the output of each driver. Refer to **Table I-1** and the controller board schematics as necessary when adding capacitors to control the slew rate (the schematics are available as a separate manual, Systech part number 80-000100-9).

The required capacitance can be determined by using the following equation:

Capacitance =
Isc (short cicuit current) * Change In Time / Change In Voltage

Appendix I

Response Control Capacitors

Table I-1. Slew Rate Capacitors

Channel	Connector Assembly P/N 65-201005-9			
	DXOUT	RTSOUT	DTROUT	Synchronous Clocks Slew Control/Filter Capacitors
0	C002	C001	C003	C005, C008
1	C102	C101	C103	C113, C115
2	C202	C201	C203	C205, C208
3	C302	C301	C303	C313, C315
4	C402	C401	C403	C405, C408
5	C502	C501	C503	C513, C515
6	C602	C601	C603	C605, C608
7	C702	C701	C703	C713, C715

Appendix I

Response Control Capacitors

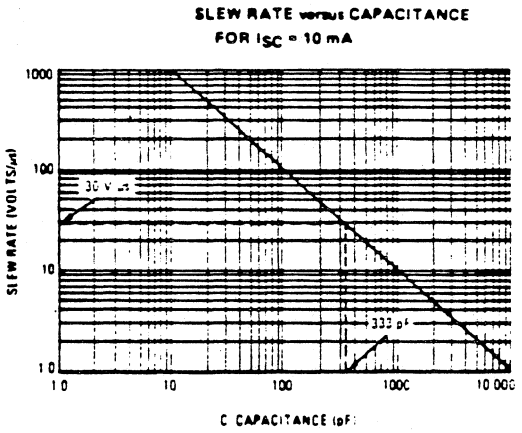


Figure I-2

Figure I-2 shows the slew rate with 10 mA as the current. The chart shows that a 330-pF capacitor could be used to achieve the desired slew rate.

Appendix I

Response Control Capacitors

MC1489L
MC1489AL

QUAD MDTL
LINE RECEIVERS
RS-232C

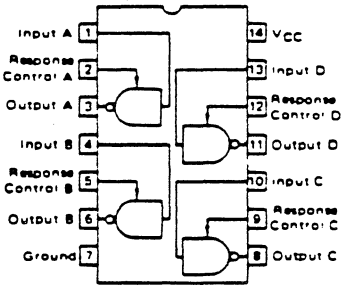


Figure I-3

Filtering noise pulses

A 1489 receiver IC (see **Figure I-3)** contains inputs called "response control". These pins can be used to filter high frequency noise pulses. **Figures I-4** and **I-5** show typical noise rejection for capacitors of various sizes. The circuit schematics of the 1489 and the 1489A are shown in **Figure I-6.** Refer to **Table I-2** and the controller board schematics as necessary when adding capacitors (the schematics are available as a separate manual, Systech part number 80-000100-9).

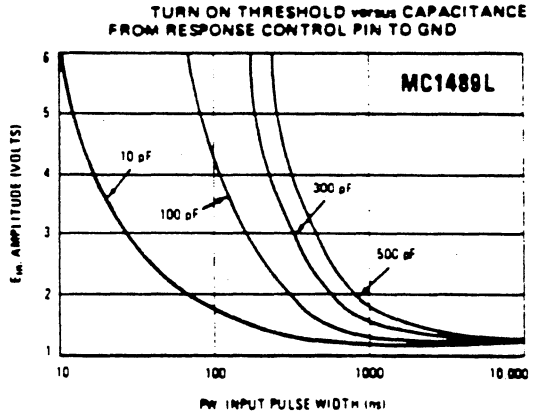


Figure I-4

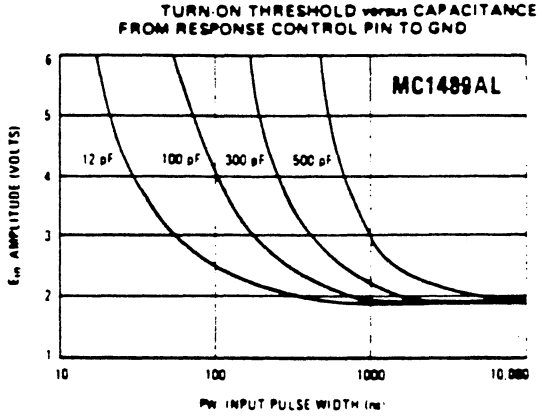


Figure I-5

Appendix I

Response Control Capacitors

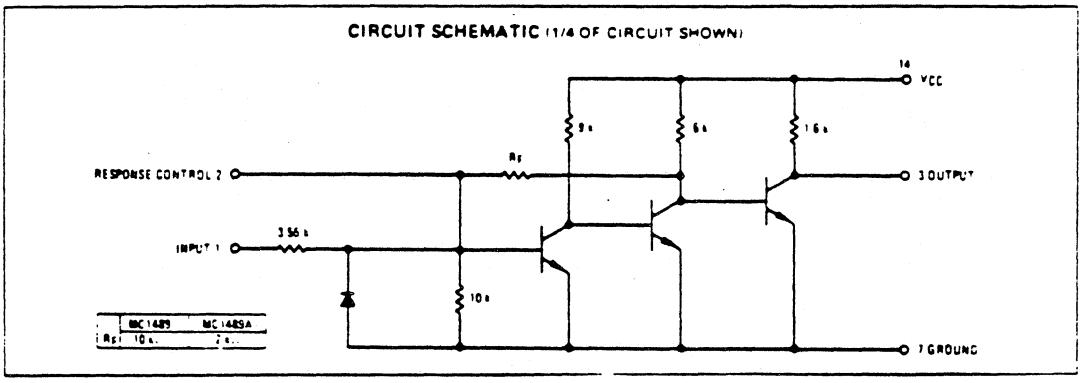


Figure I-6

Appendix I

Response Control Capacitors

Table I-2. Response Control Capacitors

Channel	USART A Board Assembly 65-201006-0				Connector Assembly 65-201005-9
	DXIN	CTSIN	DCDIN	DSRIN	Synchronous Clocks Response Control Capacitors
0	C69	C36	C61	C37	C011, C014
1	C68	C49	C60	C35	C116, C117
2	C67	C33	C59	C34	C211, C214
3	C66	C45	C58	C32	C316, C317
4	C65	C30	C57	C31	C411, C414
5	C64	C41	C56	C29	C516, C517
6	C63	C27	C55	C28	C611, C614
7	C62	C53	C54	C52	C716, C717

Appendix J: Warranty

SYSTECH Corporation Warranty

1. **WARRANTY:** SYSTECH Corporation ("Company") warrants that hardware products manufactured by it are free from defects in materials and workmanship under normal conditions of usage and service for a period of one year from the date the product is shipped from Company's factory. This warranty shall not apply to any product which was repaired or altered outside of Company's factory or authorized service stations, nor which has been subject to misuse, negligence or accident, or use not in accord with instructions furnished by Company.

2. **RETURN OF DEFECTIVE PRODUCT:** Purported defective products shall be properly packaged by Buyer and shipped, insured, at Buyer's cost to one of Company's authorized service stations for verification of defects.

3. **REPAIR OR REPLACEMENT:** If after examination, Company determines that a product is defective, Company shall, at its option, replace or repair the product at no cost to the Buyer. Return of the repaired or replaced defective product to Buyer shall be at Company's expense. Company warrants any repair or replacement for 30 days or remainder of original warranty, whichever period is longer.

4. **PRODUCTS NOT SUBJECT TO WARRANTY:** Buyer shall pay all transportation charges plus Company's established price for replacement or repair of products sent to Company by Buyer for replacement or repair outside the scope of the warranty.

5. **DISCLAIMER AND LIABILITY RESTRICTION:** THERE ARE NO WARRANTIES EXPRESS, IMPLIED, OR ARISING BY OPERATION OF LAW, EXCEPT THOSE SET FORTH HEREIN, AND THE LIMITATION OF OBLIGATION HEREIN DESCRIBES THE SOLE AND EXCLUSIVE DUTIES, OBLIGATIONS AND RIGHTS OF COMPANY, BUYER, AND THEIR SUCCESSORS.

   IN NO EVENT, BE IT DUE TO A BREACH OF ANY WARRANTY HEREUNDER OR ANY OTHER CAUSE WHATSOEVER, SHALL COMPANY BE LIABLE FOR OR OBLIGED IN ANY MANNER TO PAY CONSEQUENTIAL OR INDIRECT DAMAGES, INCLUDING, BUT NOT LIMITED TO, LOSS OF PROFITS, PLANT DOWNTIME, PROPERTY DAMAGE, OR PERSONAL INJURY DAMAGE ASSERTEDLY SUFFERED BY BUYER OR THIRD PARTIES WHETHER SUCH CLAIM IS BASED ON CONTRACT OR TORT.

   The warranty contained herein, including the Disclaimer and Liability Restriction, constitutes the entire agreement of Company and Buyer with respect to such matters, supersedes and is a merger of all prior negotiations and shall be controlling over any conflicting terms and conditions of any contracts, purchase orders, invoices, or the like, which are or may be executed in connection with the attached contract. If this warranty is voided by any breach of Condition, the Disclaimer and Liability Restriction shall remain fully effective.

   No agent, employee or representative of Company has any authority to bind Company to any affir - mation, representation or warranty concerning any goods, services or processes other than set forth herein.

6. **ASSIGNMENT:** This warranty may not be assigned or otherwise transferred by Buyer without the prior written consent of Company; provided, if Buyer advises Company in writing at the time of purchase that the product has been bought for resale, Buyer may assign this warranty to its customer.

7. **LIABILITY LIMITATIONS:** If at any time there shall be any liability asserted against Company for goods, services or processes furnished under this Agreement, notwithstanding the foregoing limitations and waivers, Buyer agrees to indemnify Company, hold it harmless and defend it against all liabilities from loss, damage or injury to persons or property by reason of the use of any goods, services or processes provided under this Agreement, regardless of cause or responsibility for negligence.

8. **LIMITATIONS OF DAMAGES OR REMEDIES:** In the event the provisions disclaiming warranties relieving Company from liability for its negligence should, for any reason, be held ineffective, it is expressly agreed that replacement and repair shall be the sole remedy of Buyer with respect to any other remedy available by applicable law.