SPPU013

TMS9914A GPIB Controller User's Guide

System Interface Controllers

TEXAS INSTRUMENTS

6/88

TMS9914A GPIB Controller User's Guide

TEXAS INSTRUMENTS

Contents

Illustrations

Tables

1. Introduction

This user's guide bridges the gap between the background of the typical microprocessor design engineer/programmer and the other IEEE-488 and TMS9914A documentation. It is intended to be used as a tutorial rather than as a reference work.

This guide helps the reader who knows digital design and microprocessors to understand the complex technical manuals for the IEEE-488 bus. The basics of microprocessor design and logic design are not covered here as there are many good texts on the subject. This book introduces the IEEE-488 specification and the TMS9914A GPIB Data Manual (p/n MP033A/SPPS009), but it does not replace them. It is presumed that you are familiar with these two documents.

Since the IEEE-488 Bus is complex, it will be reviewed twice, starting with an overview of the bus and the TMS9914A, describing a simple instrumentation system. Next, a series of examples will be presented, each example describes a specific IEEE-488 bus mode.

Read the entire user's guide before referencing it. Section 3 through Section 5 present a basic understanding of the IEEE-488 bus. Section 3 makes the important distinctions between instrument and interface, commands and data, and remote and local. Section 4 describes a typical bus application to give a concrete frame of reference for the sections that follow. The TMS9914A is introduced in Section 5. It is important to understand its place in an IEEE-488 system, its registers and logic, and its interface requirements. These sections provide a base for the detailed descriptions found further in the text.

After a firm foundation has been provided in the first five sections of the book, Section 6 through Section 13 describe the main functions implemented on the IEEE-488 bus. First, a particular bus function is described and simple illustrations given, followed by the IEEE-488 state diagram for the function. Next, the implementation of the particular function on the TMS9914A is described. These sections progress from simple functions, such as listener, to complicated functions, such as transfer of control.

Section 14 describes the few details of TMS9914A hardware interfacing that are not easily accomplished by an experienced designer.

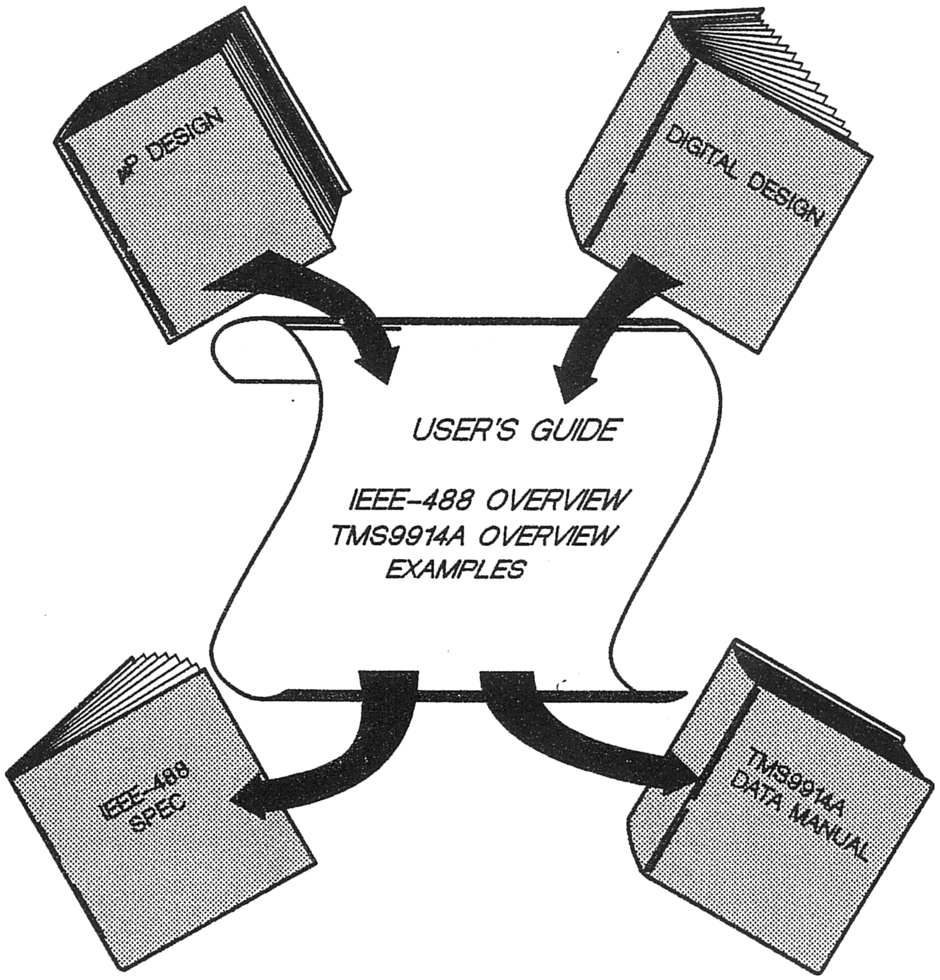


Figure 1-1. The TMS9914A User's Guide Bridges the Gap

2. Pinout and Signal Descriptions

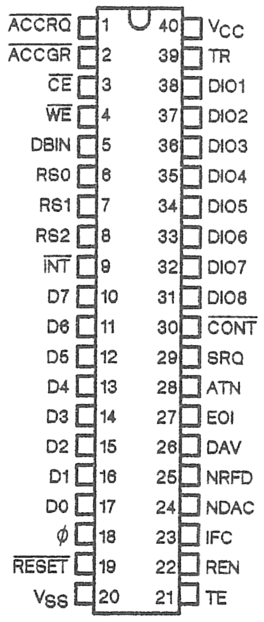


Figure 2-1. TMS9914A Pinout

Table 2-1. TMS9914A Pin Descriptions

SIGNAL	PIN	I/O (TYPE)†	DESCRIPTION
DIO8 DIO7 DIO6 DIO5 DIO4 DIO3 DIO2 DIO1	31 32 33 34 35 36 37 38	I/O(p/p) I/O(p/p) I/O(p/p) I/O(p/p) I/O(p/p) I/O(p/p) I/O(p/p) I/O(p/p)	DIO8 through DIO1 are the data input/output lines on the GPIB side. These pins connect to the IEEE-488 bus via non-inverting transceivers. DIO8 is MSB.
DAV	26	I/O(p/p)	DATA VALID: handshake line controlled by source to show acceptors when valid data is present to the bus.
NDAC	24	I/O(p/p)	NOT DATA ACCEPTED: handshake line. Acceptor sets this false (high) when it has latched the data from the I/O lines.
NRFD	25	I/O(p/p)	NOT READY FOR DATA: handshake line. Sent by acceptor to indicate readiness for the next byte.
ATN	28	I/O(p/p)	ATTENTION: sent by controller in charge. When true (low), interface commands are being sent over the DIO lines. When false (high), these lines carry data.
REN	22	I/O(o/d)	REMOTE ENABLE: sent by system controller to select control either from the front panel or from the IEEE bus.
IFC	23	I/O(o/d)	INTERFACE CLEAR: sent by the system controller to set the interface system into a known quiescent state. The system controller becomes the controller in charge.
SRQ	29	I/O(p/p)	SERVICE REQUEST: set true (low) by a device to indicate a need for service.
EOI	27	I/O(p/p)	END OR IDENTIFY: if ATN is false (high), this indicates the end of a message block. If ATN is true (low), the controller is requesting a parallel poll.
$\overline{\text{CONT}}$	30	O(p/p)	Indicates (low) 9914A is controller in charge. It is used to control direction of SRQ and ATN in pass control systems. Logically, it is (CIDS + CADS).
TE	21	O(p/p)	TALK ENABLE: controls the direction of the transfer of the line receivers. Logically, it is (CACS + TACS + EIO · ATN · (CIDS + CADS) · $\overline{\text{SWRST}}$)
D0 D1 D2 D3 D4 D5 D6 D7	17 16 15 14 13 12 11 10	I/O(p/p) I/O(p/p) I/O(p/p) I/O(p/p) I/O(p/p) I/O(p/p) I/O(p/p) I/O(p/p)	Data transfer lines on the MPU side of the device. D0 is MSB.

Table 2-1. TMS9914A Pin Descriptions (Concluded)

SIGNAL	PIN	I/O (TYPE)†	DESCRIPTION
RS0 RS1 RS2	6 7 8	I I I	REGISTER SELECT LINES: determine which register is addressed by the MPU during a read or write operation.
$\overline{\text{CE}}$	3	I	CHIP ENABLE: $\overline{\text{CE}}$ low allows access of read and write registers. If $\overline{\text{CE}}$ is high, D0-D7 are in high impedance unless $\overline{\text{ACCGR}}$ is low.
$\overline{\text{WE}}$	4	I	WRITE ENABLE: when active (low), indicates to the TMS9914A that data is being written to one of its registers.
DBIN	5	I	DATA BUS IN: an active (high) state indicates to the TMS9914A that a read is about to be carried out by the MPU.
$\overline{\text{INT}}$	9	O(o/d) (no pullup)	INTERRUPT: sent to the MPU to cause a branch to a service request.
$\overline{\text{ACCRQ}}$	1	O(p/p)	ACCESS REQUEST: this pin becomes active (low) to request a direct memory access.
$\overline{\text{ACCGR}}$	2	I	ACCESS GRANTED: when received from the direct memory access control logic, this enables the byte onto the data bus. $\overline{\text{ACCGR}}$ must be high when not participating in DMA transfer.
$\overline{\text{RESET}}$‡	19	I	INTIALIZES the TMS9914A at poweron.
TR	39	O(p/p)	TRIGGER: activated when the GET command is received over the interface or the fget command is given by the MPU.
Φ	18	I	CLOCK input: 500 kHz to 5 MHz. Need not be synchronous to system clock.
V_{SS}	20		Ground reference voltage.
V_{CC}	40		Supply voltage (+5V nominal).

† (p/p) = push/pull output
(o/d) = open drain output with internal pullup.
‡ The hardware $\overline{\text{RESET}}$ pin has the following effect on the TMS9914A:
- Serial and Parallel Poll registers cleared
- All clear/set auxiliary commands cleared except 'swrst'
- 'swrst' auxiliary command set. This holds the TMS9914A in known states.

3. IEEE-488 Overview

The IEEE-488 bus was developed to provide a standard interface for communication between instruments from diverse sources. This section discusses general information about using the interface between instruments.

3.1 The Standard

Before the IEEE-488 standard was introduced, each manufacturer, and sometimes each division of each manufacturer, had a separate standard for instrument communication. The designers of the IEEE-488 bus defined a standard that handles most communication between instruments. Conformity to this standard, although not easy, is very important.

3.2 The Bus

The IEEE-488 bus has twenty-four lines: eight for data, eight for control, and eight for ground and shielding. Data is sent a byte at a time on the eight data lines. The eight control lines are described in Section 8.

3.3 Instrument and Interface

Almost any instrument can be used with the IEEE-488 specification, because the specification says nothing about the function of the instrument itself, or about the form of the instrument's data. Instead the IEEE-488 defines a separate component, the interface, that can be added to the instrument. The signals passing into the interface from the IEEE-488 bus and from the instrument are defined in the standard. The instrument does not have complete control over the interface. Often the bus tells the interface what to do. The TMS9914A has been designed to make this interface easier to design and build. See Figure 3-1 for a conceptual illustration of the relationships between the instrument and the interface, and the interface and the bus.

3.4 Commands and Data

Commands and data are the types of information flowing through the interface. The commands are rigidly defined in the IEEE-488 specification. They are used to control the state of the interface, and to a small extent, the instrument.

Most of the information on the IEEE-488 bus is device-dependent data. For instance, if a voltmeter is sending voltage data to a printer, the data is device-dependent. The specification says nothing about the format of that data except that it must be transmitted eight bits at a time. The data can be sent in ASCII (preferred), EBCDIC, Baudot, BCD, or whatever code the printer can understand. The data rate can be from 1 byte to 1 megabyte per second.

3.5 Device Configuration Commands and Data

The IEEE-488 bus can be used to configure instruments. A few commands, such as Device Clear and Device Trigger, perform general configuration functions. The designer of each instrument defines the effect of these commands on that instrument. Most configurations are done with device-dependent data, however, rather than commands. For instance, no commands are defined to set the range on a voltmeter. The device-dependent data stream specified by the device's manufacturer must be sent. Even though this could be viewed as sending "commands" to the voltmeter, the IEEE-488 specification sees it as data. This allows the IEEE-488 bus to be flexible and adapt to new instruments easily. See Figure 3-1.

3.6 Remote and Local Messages

The IEEE-488 specification distinguishes between remote messages, which are those between the interface and the IEEE-488 bus, and local messages, which are those between the interface and the instrument (see Figure 3-1). The remote messages can be commands or data. Their form is rigidly defined. The local messages, on the other hand, can be logic levels on a line between the instrument and the interface. Or in the case of the TMS9914A, the local messages can be bits in a register. The instrument's designer defines the effect of a local message from the interface on the instrument. This allows flexibility in the design of the instrument.

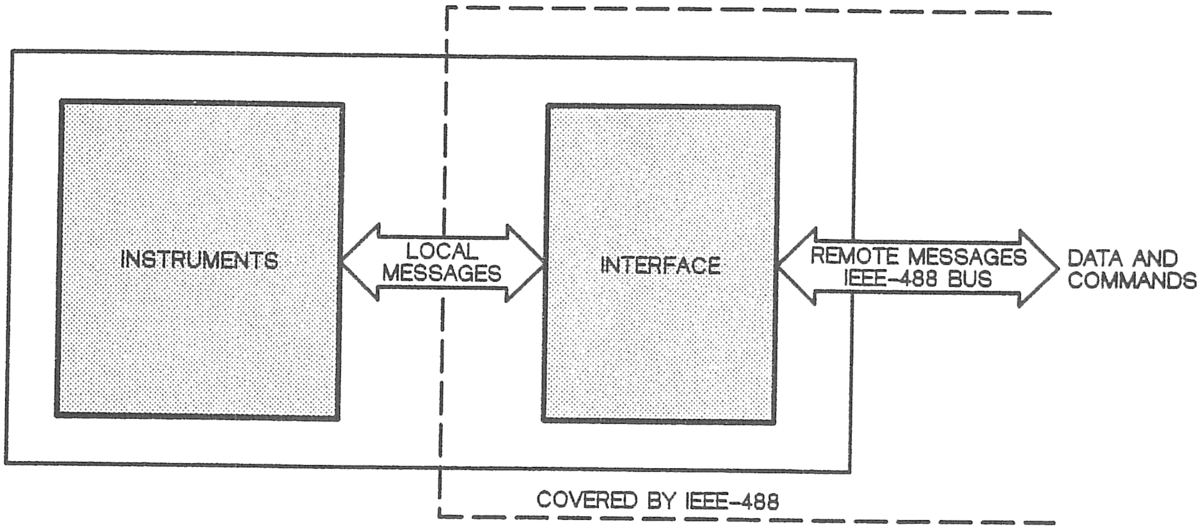


Figure 3-1. IEEE-488 Concepts: Instrument and Interface, Commands and Data, Remote and Local Messages

4. A Typical IEEE-488 System

An example will illustrate how the IEEE-488 bus works. A system with a computer, a voltmeter, and a printer will show all the basic features of the specification. The system can be used to take voltage readings at computer-controlled intervals and to print the results to evaluate or monitor the system.

4.1 Clearing the Devices and Their Interfaces

When the system starts up, the computer is the controller. It sends commands out over the bus. First, it will put an Interface Clear (IFC) command on the bus, which tells the printer and voltmeter interfaces to stop receiving data (listening) and transmitting data (talking). This does not affect the instruments themselves, just their interfaces, as shown in Figure 4-1.

Note:

The IFC command must be reset after a minimum of 100 µs (IEEE minimum time), otherwise the interface will be held in a reset state until IFC is released.

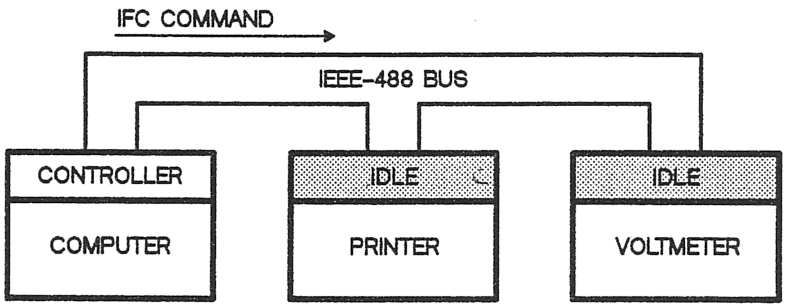


Figure 4-1. Clearing the Interfaces

The controller might then send out a Device Clear (DCL) command, shown in Figure 4-2. This command will cause each instrument to return to a default state defined by the instrument manufacturer. These two commands illustrate that a distinction is made by the IEEE-488 standard between the device (the instrument) and its interface.

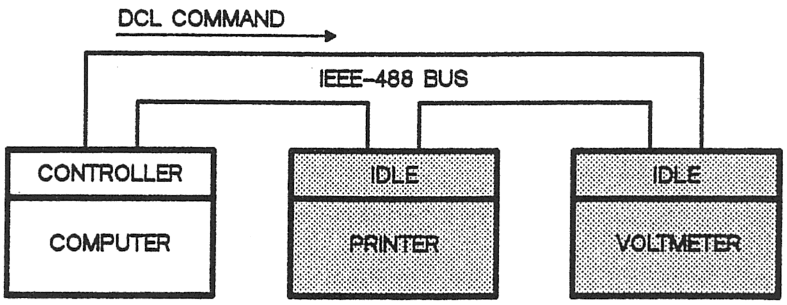


Figure 4-2. Clearing the Devices

4.2 Configuring the Devices

After the controller has cleared the devices and their interfaces, it will configure each device for the specific application. Each device has a unique address, usually set by switches or by software in the device. First, to configure the voltmeter, the controller makes the voltmeter a listener by sending out a listen command with the voltmeter's address. This is called a My Listen Address (MLA) command. See Figure 4-3.

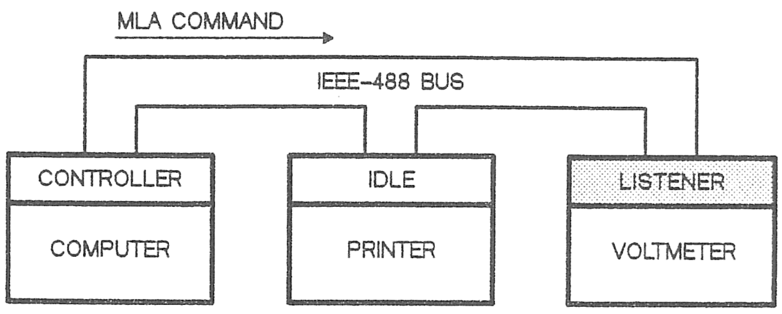


Figure 4-3. Addressing the Voltmeter

Next, the computer changes from a controller to a talker so it can send data rather than commands. Now the configuration data can be sent to the voltmeter, telling it, for example, what range to set itself to, as shown in Figure 4-4. The exact form of this data is defined by the manufacturer of the specific voltmeter, not the IEEE-488 specification. The data are device-dependent.

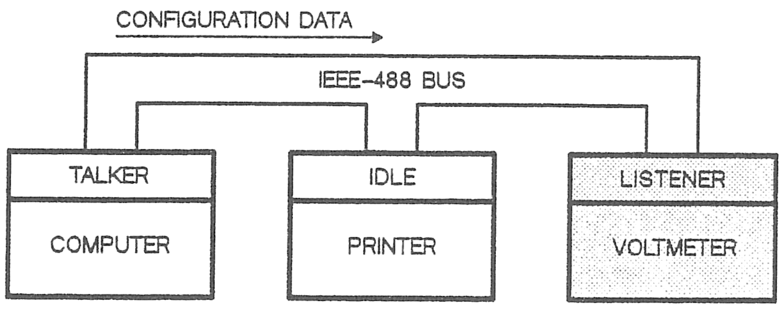


Figure 4-4. Initializing the Voltmeter

After configuring the voltmeter, the computer turns back into a controller from a talker, and send an Unlisten (UNL) command to the voltmeter. See Figure 4-5.

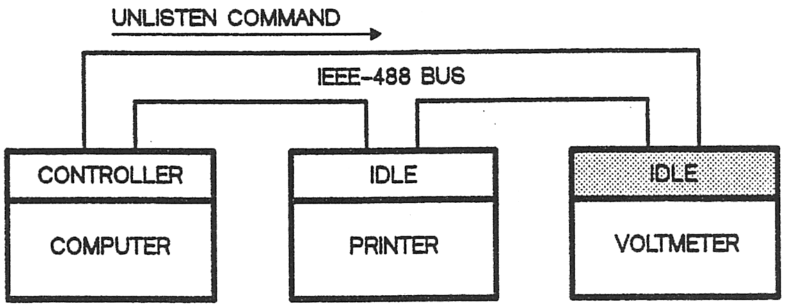


Figure 4-5. Unlistening the Voltmeter

The printer is initialized with the same sequence of steps previously defined for the voltmeter, as shown in Figure 4-6.

1) Send out the My Listen Address (MLA) command.
2) Change controller to talker.
3) Send configuration data to printer.
4) Change talker to controller.
5) Send an Unlisten command to printer.

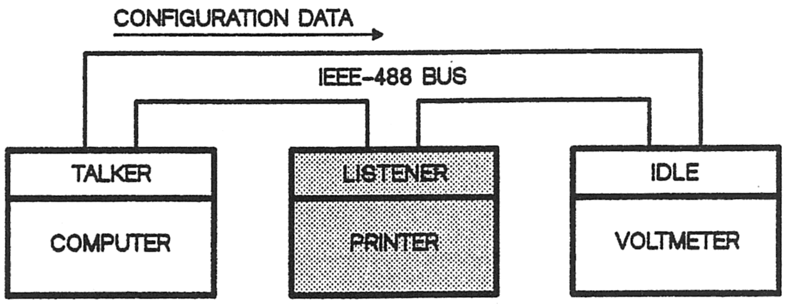


Figure 4-6. Initializing the Printer

4.3 Taking Data

Now that the system is initialized, readings may be taken. To insure precise timing between instruments, send a trigger command to the voltmeter. This is called a Group Execute Trigger (GET) by the IEEE-488 specification. The GET command works only on those devices that have their interfaces set up to listen, so first send an MLA (My Listen Address) command to the voltmeter. See Figure 4-7.

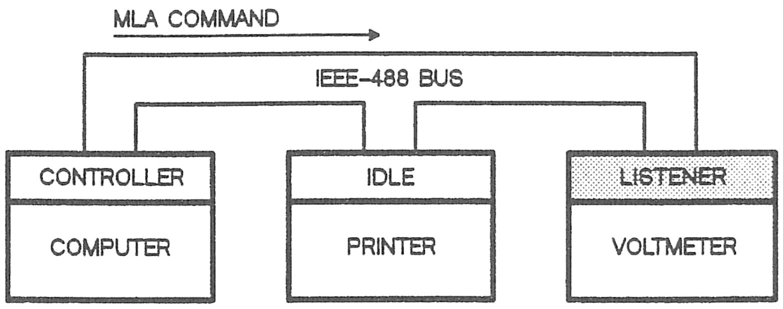


Figure 4-7. Addressing the Voltmeter Again

Now, to obtain a reading, program the computer to put a GET command on the bus, as shown in Figure 4-8. The voltmeter is designed by the manufacturer to take a reading upon receiving a GET command.

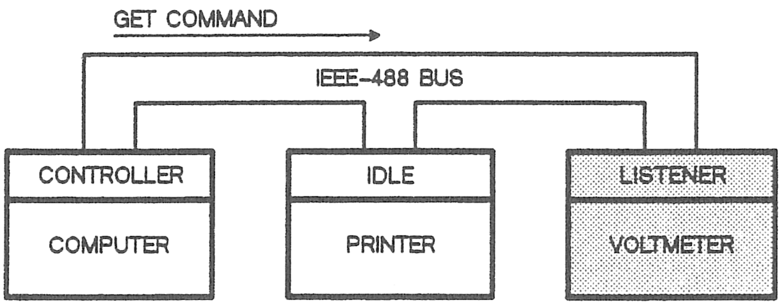


Figure 4-8. Triggering the Voltmeter

The computer puts an UNL (Unlisten) command on the bus, telling the voltmeter to stop listening. See Figure 4-9.

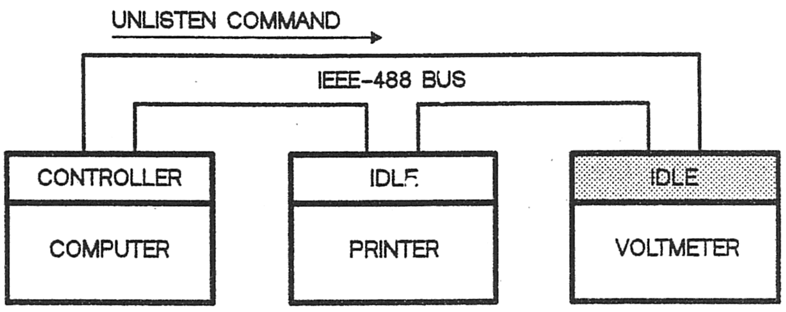


Figure 4-9. Unlistening the Voltmeter

Next, tell the printer to listen with an MLA command, as shown in Figure 4-10.

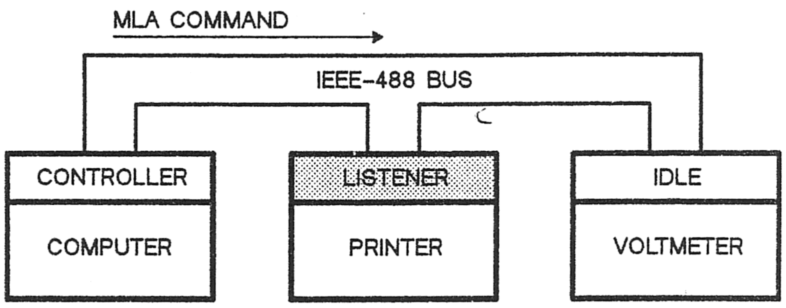


Figure 4-10. Addressing the Printer

Finally, the computer tells the voltmeter to talk by putting a My Talk Address (MTA) command on the bus. The MTA is similar to the My Listen Address (MLA). See Figure 4-11.

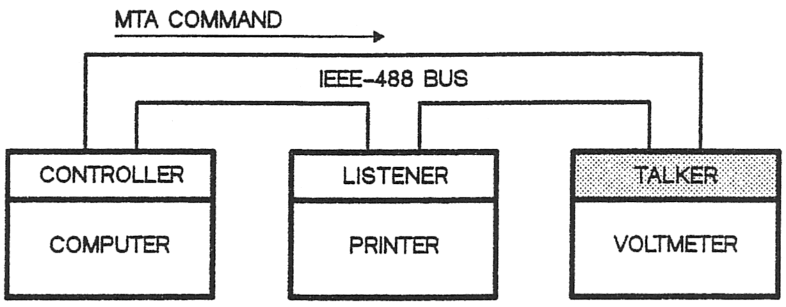


Figure 4-11. Addressing the Voltmeter as a Talker

The computer then goes into an idle state. This enables the voltmeter to talk and send data to the printer, which will print out the information. See Figure 4-12.

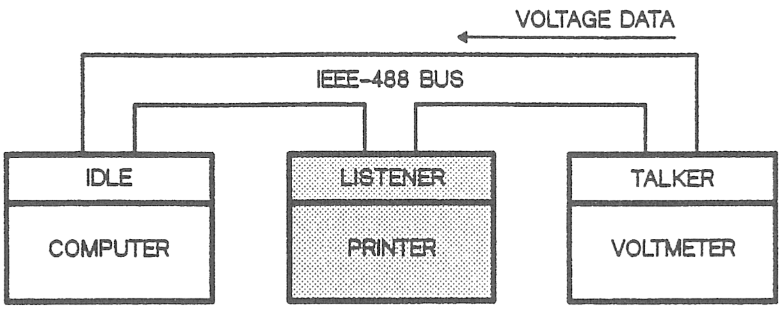


Figure 4-12. Printing the Voltage Data

Now an IFC (Interface Clear) can be sent and another reading taken by repeating each step, beginning with Addressing the Voltmeter (Section 4.2).

5. TMS9914A Overview

The hardware interface and registers of the TMS9914A will now be discussed in more detail.

5.1 Hardware Interface

The TMS9914A has a fairly conventional microprocessor interface. See Figure 5-1. The lines are:

- Eight Data lines, D0 to D7. D0 is the most significant bit.
- Three Register Select lines, RS0 to RS2. RS2 is MSB (Most Significant Bit).
- Chip Enable, $\overline{\text{CE}}$
- Write Enable, $\overline{\text{WE}}$
- Data Bus In or Read Enable, DBIN
- Clock, 0
- Interrupt Out, $\overline{\text{INT}}$
- Reset, $\overline{\text{RESET}}$
- Access Control for DMA, $\overline{\text{ACCRQ}}$ and $\overline{\text{ACCGR}}$

Note:

The TMS9914A interfaces to the CPU via an eight-bit bi-directional data bus which is labeled D0-D7 in the pin description in the data manual. In TI terminology, D0 is the MSB (Most Significant Bit) and D7 is the LSB (Least Significant Bit). Other manufacturers refer to these lines with the opposite terminology, thus referencing D0 as LSB and D7 as MSB. It is a common oversight to connect these data lines contrary to TI convention, resulting in the corruption of data.

Note:

"∧" will symbolize logical "and" in this manual, while "v" will symbolize logical "or."

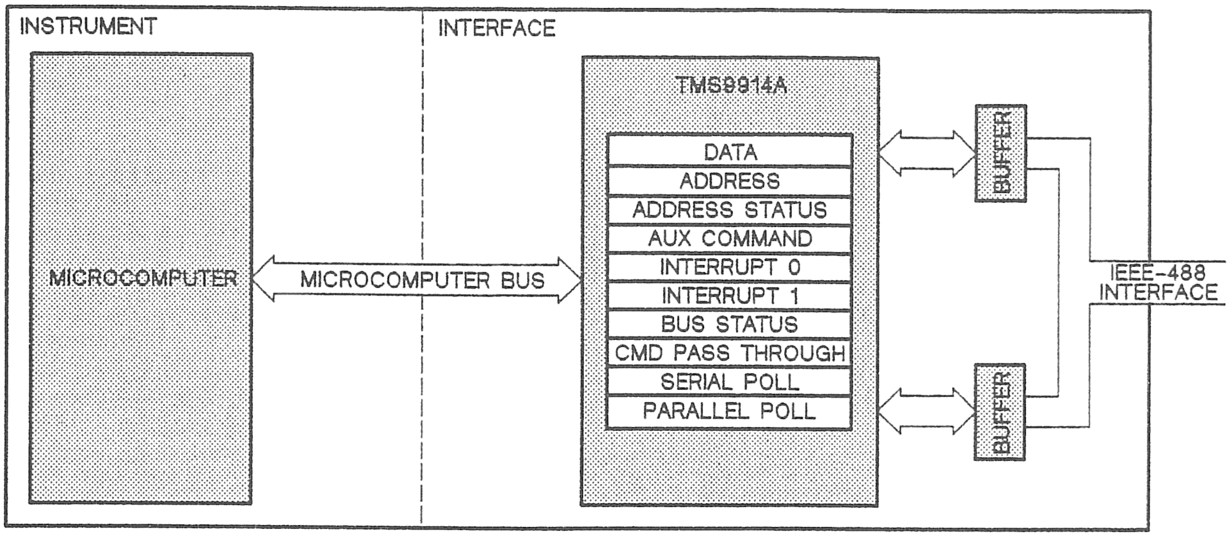


Figure 5-1. Block Diagram of an IEEE-488 System Using a TMS9914A

The TMS9914A can be easily interfaced to the IEEE-488 bus using the 75160A, 75161A, or 75162A Bus Transceiver/Driver chips. More detailed information concerning the use of the interface devices can be found in the TMS9914A Data Manual (p/n MP033A/SPOU002).

5.2 Registers

The registers of the TMS9914 are briefly described in the following list. For a more detailed discussion, see the TMS9914A Data Manual.

REGISTER	*FUNCTION*
Data	The microprocessor puts data bytes sent by the talker and command bytes sent by the controller here. The microprocessor also takes data bytes received by a listener from this register.
Address	The microprocessor puts the address of the device in this location.
Address Status	This register tells the microprocessor whether it has been addressed as a talker or a listener.
Auxiliary Command	The microprocessor sends commands to this register to tell the TMS9914A to change its state.
Interrupt Mask and Status	These registers show what events have occurred in the TMS9914A and control the generation of interrupts to the microprocessor. They also control

data flow on the IEEE-488 bus, dependent upon which bits are masked and which are unmasked.

Bus Status

This register gives the current state of the IEEE-488 control lines.

Command Pass Through

This gives the current state of the IEEE-488 bus data lines. The microprocessor reads certain commands directly from the IEEE-488 bus using this register.

Serial and Parallel Poll

The controller can poll the devices on the bus either serially or in parallel. The microprocessor puts the devices' responses to these polls in these registers, and the TMS9914A puts the responses on the IEEE-488 bus at the proper time.

6. Listener Mode

Understanding the IEEE-488 state diagrams is difficult, but it is essential to an understanding of the whole system. Since the listener state diagram is the simplest in the IEEE-488 specification, it will be discussed first. A listener is a device, such as the printer in our example system, that can accept data from a talker. The listener state diagram shows how to get our printer ready to accept data. Figure 6-1 is a simplified version of the IEEE-488 state diagram.

6.1 Listener Description

The circle on the left of Figure 6-1 represents the idle state. The printer is idle after power up and after receiving an IFC (Interface Clear) command. In this state, the printer remains inactive and any attempt to send data to it has no effect. The IFC command must be reset after at least 100 μs, or the interface will be held in an inactive state.

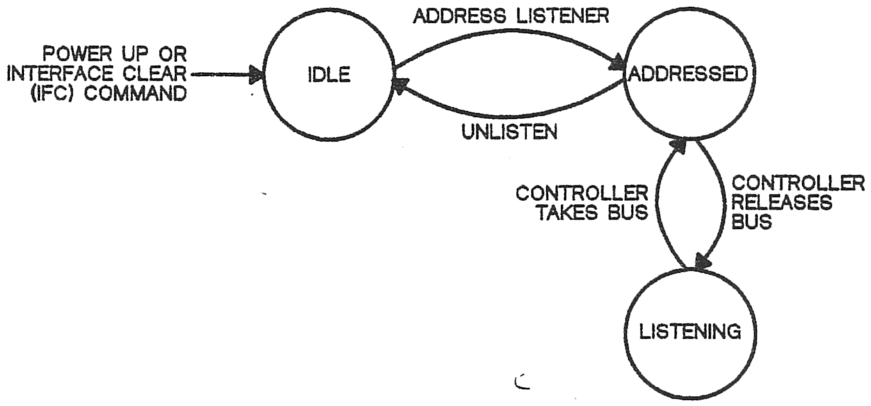


Figure 6-1. Simplified Listener State Diagram

The listener leaves the idle state by being addressed to listen, which puts it into the addressed state.

As soon as the controller releases the bus, the listener will go to the active state (listening) and will be ready to accept data.

When the controller takes control of the bus, the listener returns to the addressed state. The controller then tells the listener to stop listening, sending it back to the idle state.

6.2 State Diagrams

Listener is an "interface function." There are many other interface functions, each with their own state diagrams. Talker and controller are examples previously discussed, but they are not implemented in the printer in this example. The printer has an interface function called Acceptor Handshake (AH), which controls the acceptance of each byte of data coming into the printer. The printer is always someplace in the AH state diagram. At the same time, it is someplace in the listener (L) state diagram.

The computer in our sample system has controller, listener, talker, Acceptor Handshake, Source Handshake, Parallel Poll, and many other interface functions implemented. It is always at a defined point in the state diagram for each interface function. There are also cross-links between state diagrams.

6.3 IEEE-488 Listener State Diagram

Figure 6-2 is the IEEE-488 state diagram for the listener or L function. The circles and arrows are the same as in the simplified diagram. Table 6-1 shows LIDS means Listener Idle State, which is the left circle in Figure 6-2. All the other mnemonics are also in the table.

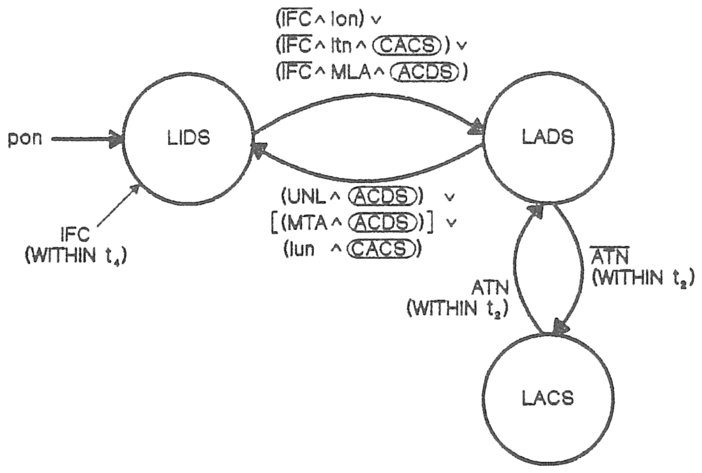


Figure 6-2. IEEE-488 Listener State Diagram

Table 6-1. IEEE-488 Listener Mnemonics

MESSAGES	INTERFACE STATES
pon = Power on	LIDS = Listener idle state
ltn = Listen	LADS = Listener addressed state
lun = Local unlisten	LACS = Listener active state
lon = Listen only	ACDS = Accept data state (AH function)
IFC = Interface clear	CACS = Controller active state (C function)
ATN = Attention	
UNL = Unlisten	
MLA = My listen address	
MTA = My talk address	

The "pon" to the left of the LIDS circle means power on. It is printed in lowercase, which indicates a local message. Remote messages are shown in uppercase. Local messages are between the instrument and the interface. The instrument tells the interface that power has been turned on. The "pon" has its own arrow, so regardless of location in the state diagram, "pon" leads to LIDS.

Just below "pon" is IFC (Interface clear). IFC is in uppercase, indicating it is a remote message, which is a message that comes in on the bus. All the messages are defined in Table 6-1.

Look at the top of the state diagram, where the simplified diagram says "address listener." The "v" means logical or. The IEEE-488 specification says "v" is evaluated last in the expression. If any of the three expressions separated by "v" goes true, the listener function will go from LIDS to LADS. The top expression is:

$$\overline{\text{IFC}} \wedge \text{lon}$$

The bar over IFC means "IFC not," and the "$\wedge$" means "and." So when IFC is false and "lon" is true, the listener function will go to the addressed state (LADS). The "lon" local message is used for systems without controllers. To use the voltmeter and printer without the computer, flip the "listen only" switch on the printer and the "talk only" switch on the voltmeter. This will probably cause the voltmeter to send voltage data, the printer to print, and paper to feed.

The next expression is:

$$\overline{\text{IFC}} \wedge \text{ltn} \wedge (\text{CACS})$$

$\overline{\text{IFC}}$ was discussed earlier, "ltn" is a local message, and CACS is a linkage from the controller state diagram. The oval indicates a linkage. This expression is used when the controller wants to be a listener when it releases control. For example, if the computer wants to listen to the voltage data itself, it would use the "ltn" local message to go to LADS. Then when it releases the bus, the computer will go to LACS. The "ltn" local message is valid only when the controller is in CACS state, because the controller is in control of the bus only in CACS.

The last expression for this arrow is:

$$\overline{\text{IFC}} \wedge \text{MLA} \wedge (\text{ACDS})$$

In this expression, MLA (My Listen Address) is needed. Although both IFC and MLA are commands, $\overline{\text{IFC}}$ is specified because commands go down the bus in two different ways. A few commands, like IFC, have a dedicated line, while most commands go along the same eight lines used for data. If an IFC and a MLA occur at the same time, the MLA is ignored. ACDS (Accept Data State) links the Acceptor Handshake state diagram to the listener state diagram. ACDS means that the data or command on the eight data lines has been accepted. For this example, the command is MLA. In other words, the controller sends a MLA command to the printer, the printer accepts the command, and goes into LADS state. This is how most devices get into listener addressed mode in most systems (such as the example system).

Go down the arrow from LADS to LACS. Table 6-1 shows ATN means attention. ATN is a command that has a line dedicated to it, like IFC. The ATN line tells whether the controller is using the data bus. Any byte that travels down the 8-bit bus when the ATN line is true is an interface command byte. If ATN is not true, that byte is a data byte. The MLA command used to get to LADS was sent with ATN true. The listener will stay in LADS until the controller is finished sending commands. When the controller is finished, it will release the ATN line and go to an idle state in its state diagram; then the listener can start listening. Going to LACS (Listener Active State), a maximum delay, t_2, is specified. The TMS9914A is fast enough to handle this delay.

Going back to LADS uses the opposite procedure. The controller takes the bus back by asserting the ATN line. If the controller only wants to send some commands and have the device go back to listening, it can release the bus again. The path between LACS and LADS will always be open if the controller will allow it.

When the controller wants a device to stop listening, it uses one of three possible methods:

1) For the UNL (Unlisten) command, the ATN line must be true, but this is implicit in the UNL. The ACDS means the command was accepted, and it was indicated with the handshake lines.

UNL ∧ (ACDS)

2) This term is in brackets because it is optional. The MTA (My Talk Address) means that if the device is addressed to talk, it ceases to listen.

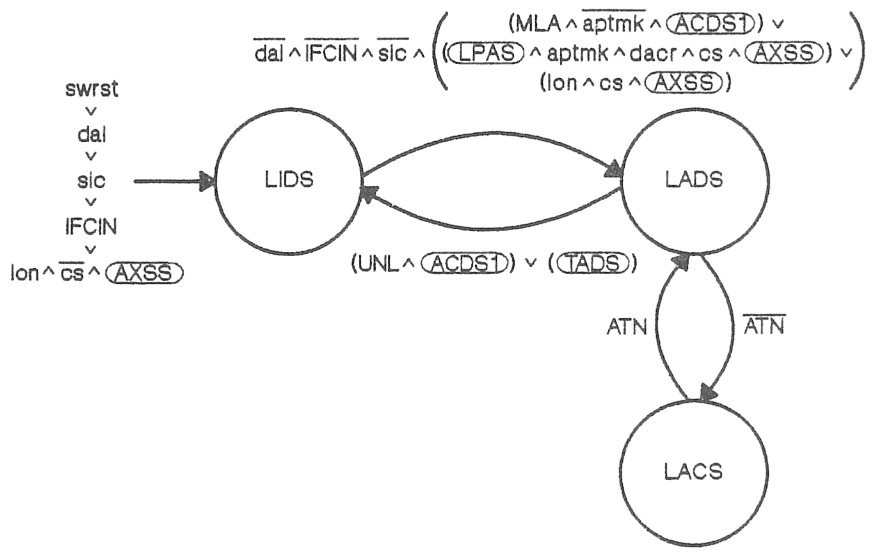
[MTA ∧ (ACDS)]

3) This again is the special case of the controller wanting to listen. Now the controller wants to stop listening. Because it is not allowed to listen to its own commands, sending an UNL to itself will not have an effect. It must take control (by asserting ATN), which puts it in the CACS state in the controller state diagram. When it gets a "lun" local message, its listener function will go to LIDS.

lun ∧ (CACS)

6.4 TMS9914A State Diagram

Figure 6-3 shows the TMS9914A implementation of the listener function. On the left of the diagram is an expression with an arrow going to LIDS. If any line of the expression becomes true, the listener function will go to LIDS.



$\overline{dal} \wedge \overline{IFCIN} \wedge \overline{sic} \wedge \left(\begin{array}{c} (MLA \wedge \overline{aptmk} \wedge (ACDS1)) \vee \\ ((LPAS) \wedge aptmk \wedge dacr \wedge cs \wedge (AXSS)) \vee \\ (lon \wedge cs \wedge (AXSS)) \end{array} \right)$
swrst ∨ dal ∨ sic ∨ IFCIN ∨ lon ∧ $\overline{cs}$ ∧ (AXSS)
LIDS
LADS
(UNL ∧ (ACDS1)) ∨ (TADS)
ATN
$\overline{ATN}$
LACS

Figure 6-3. TMS9914A Listener State Diagram

According to Figure 6-2, "swrst" means software reset. Software reset is a local message. It can be sent to the TMS9914A from the microcomputer using the "Auxiliary Command" register. A hex 80 will set the "swrst" state, and a hex 0 will clear it. A hardware reset of the TMS9914A will also set the "swrst" state. An "swrst" puts most of the state diagrams in an idle state.

Table 6-2. TMS9914A Listener Mnemonics

MESSAGES	INTERFACE STATES
aptmk = Address pass through interrupt mask bit	LACS = Listener active state
cs = Clear/set bit of the auxiliary command register	LADS = Listener addressed state
dacr = Release DAC Holdoff	LIDS = Listener idle state
dal = Disable listener	ACDS1 = Accept data state 1 (AH function)
lon = Listen only	AXSS = Auxiliary command strobe state (ACR function)
sic = Set interface clear	LPAS = Listener primary addressed state (LE function)
swrst = Software reset	TADS = Talker addressed state (T function)
ATN = Attention	
IFCIN = Internal IFC (a debounced signal, suppressed by sic)	
MLA = My listen address	
UNL = Unlisten	

A "dal" (disable listener) is also a local message, but it is not sent via the auxiliary register. Bit D1 of the address register in the TMS9914A is the "dal" bit. If it is set, listener mode is disabled (i.e., set permanently into the idle state).

An "sic" (set interface clear) is an auxiliary command used to send out an Interface Clear when the TMS9914A is the system controller. When the TMS9914A is sending an Interface Clear, the TMS9914A does not listen to the Interface Clear itself. Thus, its listener function will not get to LIDS without the "sic" term in place.

"IFCIN" means an IFC has been received and debounced. A TMS9914A IFCIN works the same as an IEEE-488 IFC.

$$\text{lon} \wedge \overline{\text{cs}} \wedge \boxed{\text{AXSS}}$$

An "lon" (listen only) is a local message sent through the auxiliary command register. Some auxiliary commands have a clear/set bit associated with them, and "lon" is one of these. The $\overline{\text{cs}}$ (cs not) indicates the clear/set bit should be cleared (i.e., not listen only); the "cs" bit is the most significant bit. The "lon" is equal to a nine. This means a hex nine is "lon $\wedge$ $\overline{\text{cs}}$," and a hex 89 is "lon $\wedge$ cs." The AXSS is a state on the auxiliary command state diagram, which means "lon" is a pulsed command. AXSS goes true for five TMS9914A clock cycles, enough to put the TMS9914A into LIDS, and then goes false. The AXSS also means the device must wait five clock cycles before issuing another auxiliary command.

In the expression to get to LADS, this is not in the parentheses.

$$\overline{\text{dal}} \wedge \overline{\text{IFCIN}} \wedge \overline{\text{sic}}$$

As discussed, "dal" is disable listener, IFCIN is the same as IFC, and "sic" is set interface clear. These all must be false before the listener can be addressed.

$$\text{MLA} \wedge \overline{\text{aptmk}} \wedge \boxed{\text{ACDS1}}$$

MLA is My Listen Address, as it is on the IEEE-488 diagram. ACDS1 is the TMS9914A's version of Accept Data State, which is the state in the Acceptor Handshake state diagram where a byte, in this case the MLA command, is accepted. "aptmk" is the address pass-through interrupt mask. This is set to enable secondary addressing, a feature explained in Section 12. Since secondary addressing is not being used, "aptmk" is reset, so only MLA and ACDS1 need to change states.

$$\boxed{\text{LPAS}} \wedge \text{aptmk} \wedge \text{dacr} \wedge \text{cs} \wedge \boxed{\text{AXSS}}$$

This is how the secondary addressing gets into LADS. See Section 12.

$$\text{lon} \wedge \text{cs} \wedge \boxed{\text{AXSS}}$$

This equation is the same as the "lon" local message in the IEEE-488 state diagram, but includes the "cs" bit and the AXSS delay. The next equation was in the IEEE-488 state diagram, but it is not in the diagram for the TMS9914A.

$$\overline{\text{IFC}} \wedge \text{ltn} \wedge \boxed{\text{CACS}}$$

The "lon" local message puts the interface into LADS at any time. A second local message (ltn) is not needed to put the interface into LADS at a specific time; to do so would be a waste of silicon. To implement this expression on the TMS9914A, go into $\boxed{\text{CACS}}$, and issue an "lon."

Getting down to LACS (the active state where data was listened to) is the same as in the IEEE-488 diagram. $\overline{\text{ATN}}$ is used to go down, and ATN to go back up.

Going back from LADS to LIDS is easy. TADS means Talker Addressed State. Thus, if a device is addressed as a talker, it stops listening. This is the way the TMS9914A implements the MTA $\wedge$ $\boxed{\text{ACDS}}$ expression from the IEEE-488 diagram. UNL $\wedge$ $\boxed{\text{ACDS1}}$ is the same as the UNL $\wedge$ $\boxed{\text{ACDS}}$ from the IEEE-488 diagram. An expression in the IEEE-488 diagram is not in the TMS9914A diagram.

$$\text{lun} \wedge \boxed{\text{CACS}}$$

The "lun" command is the same as the multi-purpose TMS9914A "lon" message. This use of "lon" is shown on the left by LIDS (lon $\wedge$ $\overline{\text{cs}}$ $\wedge$ AXSS). Go into CACS, send the "lon" auxiliary command with the "cs" bit cleared, and wait five TMS9914A clock cycles. This gets one to LIDS.

6.5 Listener Implementation

Assuming starting with power on, the simplest implementation of the Listener function requires only four steps.

1) Set up the Address Register.

   a. Put the address into the five low bits of the TMS9914A's Address Register.

   b. Reset bit D1 (the "dal" bit) in the Address Register. The TMS9914A interfaces to the CPU via an eight-bit bi-directional data bus which is labeled D0-D7 in the pin description in the data manual. In TI terminology, D0 is the MSB and D7 is the LSB. Other manufacturers refer to these lines with the opposite terminology, thus referencing D0 as LSB and D7 as MSB. It is a common oversight to connect these data lines contrary to TI convention thus resulting in the corruption of data.

2) Clear both Interrupt Mask registers. Interrupts are not used for this example. Neither software nor hardware reset affects the Interrupt Mask Registers; they come up in a random state.

3) Clear the "swrst" auxiliary command. "Swrst" is set by hardware reset and must be cleared before the TMS9914A can start talking to the IEEE-488 bus. It is cleared by writing a 0 to the auxiliary command register.

4) Wait for the BI (Byte In) bit in Interrupt Register 0 to go high. BI indicates the TMS9914A has been addressed, has gone into LACS (Listener Active State), and has received a byte. One can read the byte from the Data register, wait for the BI bit to be set, read a byte, and so on. Either reading from Interrupt Register 0 or reading from the Data Register clears bit BI.

The TMS9914A handles detecting the address and switching between states. It also latches the bytes in.

7. Talker Mode

The listener function and the talker function are similar but have some differences. The same approach will be used to describe the talker function as to describe the listener function.

7.1 Talker Description

Figure 7-1 shows the simplified version of the talker state diagram. The talker state diagram has Idle, Addressed, and Active states, as in the listener diagram. There is also a fourth state, Serial Poll. Serial Poll is special because it has nothing to do with a talker sending data to a listener. It is discussed in Section 10.

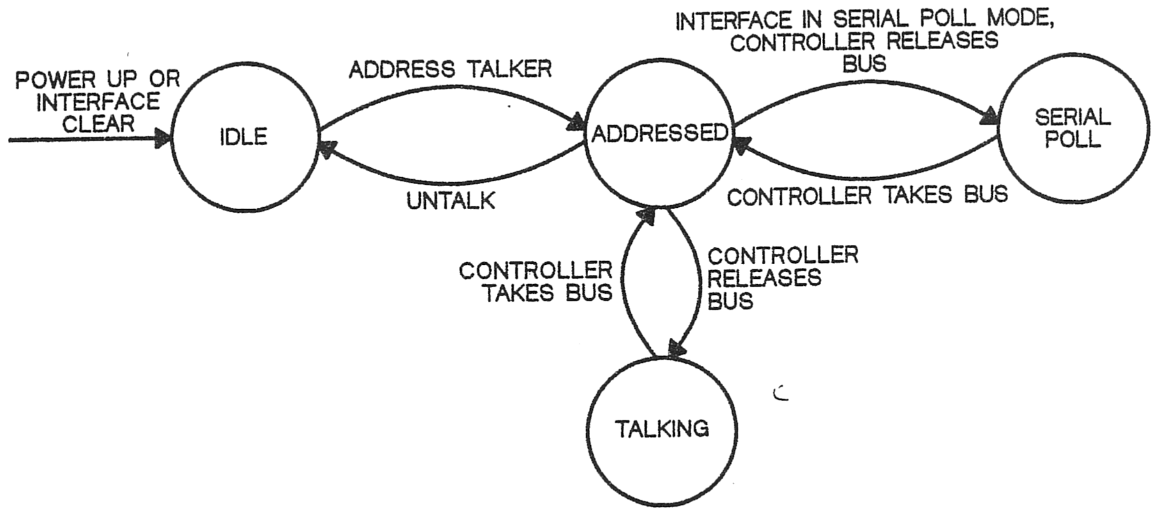


Figure 7-1. Simplified Talker State Diagram

7.2 IEEE-488 Talker State Diagram

Figure 7-2 and Table 7-1 show the IEEE-488 state diagram for the talker function. The "pon" and the IFC going into TIDS (Talker Idle State) are the same as the functions going to LIDS (Listener Idle State) for the listener function. When going from TIDS to TADS (Talker Addressed State), the "ton" (talk only) works the same as the listener's "lon" (listen only), and is intended for use in systems without controllers. The MTA (My Talk Address) works like the listener's MLA (My Listen Address). The controller puts an MTA on the bus. The Talker accepts it and goes into TADS.

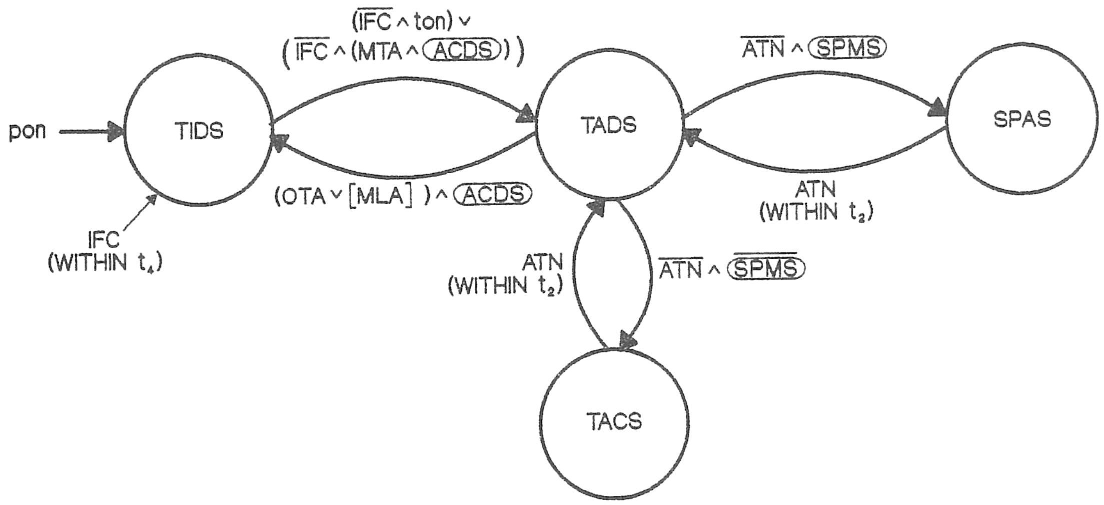


Figure 7-2. IEEE-488 Talker State Diagram

Table 7-1. IEEE-488 Talker Mnemonics

MESSAGES		INTERFACE STATES	
pon	= Power on	TIDS	= Talker idle state
ton	= Talk only	TADS	= Talker addressed state
IFC	= Interface clear	TACS	= Talker active state
ATN	= Attention	SPAS	= Serial poll active state
MTA	= My talk address	SPIS	= Serial poll idle state
SPE	= Serial poll enable	SPMS	= Serial poll mode state
SPD	= Serial poll disable	ACDS	= Accept data state (AH function)
OTA	= Other talk address		
MLA	= My listen address		

The transitions from TADS to TACS (Talker Active State) and back are exactly the same as the listener transitions, except the Serial Poll Mode, which will not be set except to do a Serial Poll.

The only difference in talker is the transition from TADS to TIDS:

$$(\text{OTA} \vee [\text{MLA}]) \wedge \boxed{\text{ACDS}}$$

This expression only looks different. It can be rewritten:

$$\text{OTA} \wedge \boxed{\text{ACDS}} \vee [\text{MLA} \wedge \boxed{\text{ACDS}}]$$

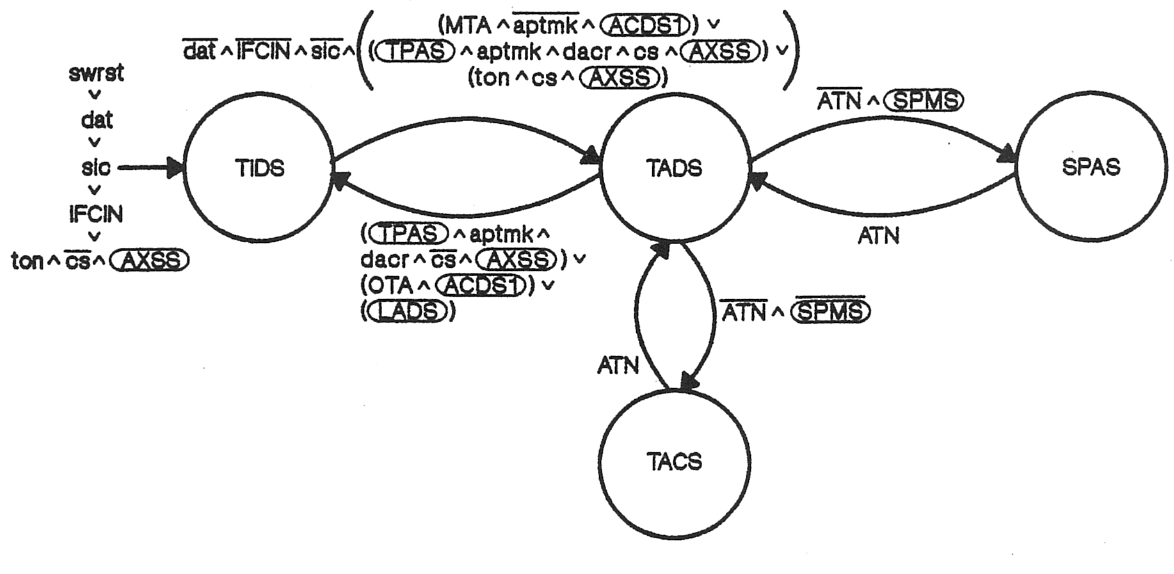
The OTA (Other Talk Address) tells the talker to go to an idle state because only one talker can be active at a time. When the talker sees another talker addressed, it is quiet. The MLA (My Listen Address) is an optional expression. It implies that if the talker is addressed to listen, it stops talking. As mentioned previously, ACDS is a state in another diagram that implies the data or command on the eight data lines has been accepted.

7.3 TMS9914A Talker State Diagram

Figure 7-3 and Table 7-2 cover the TMS9914A implementation of the talker function. They are similar to the listener diagram and table. Serial Poll has an extra state, and many terms have "l" replaced by "t," as in the IEEE-488 diagram.

Table 7-2. TMS9914A Talker Mnemonics

MESSAGES	INTERFACE STATES
aptmk = Address pass through interrupt mask bit	TACS = Talker active state
cs = Clear/set bit of the auxiliary command register	TADS = Talker addressed state
dacr = Release DAC Holdoff	TIDS = Talker idle state
dat = Disable talker	SPAS = Serial poll active state
sic = Set interface clear	ACDS1 = Accept data state 1 (AH function)
swrst = Software reset	AXSS = Auxiliary command strobe state (ACR function)
ton = Talk only	TPAS = Talker primary addressed state (TE function)
ATN = Attention	LADS = Listener addressed state (L function)
IFCIN = Internal IFC (a debounced signal, suppressed by sic)	
MTA = My talk address	
OTA = Other talk address	



$\overline{dat} \wedge \overline{IFCIN} \wedge \overline{sic} \wedge \left((MTA \wedge \overline{aptmk} \wedge ACDS1) \vee ((TPAS) \wedge aptmk \wedge dacr \wedge cs \wedge (AXSS)) \vee (ton \wedge cs \wedge (AXSS)) \right)$
swrst ∨ dat ∨ sic ∨ IFCIN ∨ ton ∧ $\overline{cs}$ ∧ (AXSS)
TIDS
TADS
SPAS
TACS
$\overline{ATN} \wedge \overline{SPMS}$
ATN
((TPAS) ∧ aptmk ∧ dacr ∧ $\overline{cs}$ ∧ (AXSS)) ∨ (OTA ∧ (ACDS1)) ∨ ((LADS))
$\overline{ATN} \wedge \overline{SPMS}$
ATN

Figure 7-3. TMS9914A Talker State Diagram

On the arrow leading into TIDS from the left, only two terms are different from those for listener. A "dat" (disable talker) has replaced a "dal." The "dat," like the "dal," is a bit in the Address Register. A "ton" (talk only) has replaced "lon" (listen only).

The "ton," like the "lon," is an auxiliary command used for systems without controllers. A "ton" is also used when a controller wants to become a talker.

A "dat" and a "ton" also replace a "dal" and an "lon" in the expressions above the arrow from TIDS to TADS. MTA (My Talk Address) is used to address a talker, the same as MLA (My Listen Address) is used to address a listener.

The talker goes from being just addressed (TADS) to being active (TACS) at the same time as the listener, when the controller releases the ATN line. Only in TACS can the talker actually talk to a listener. When the talker is in TACS, and one or more listeners are in LACS, data can flow, which is the purpose of the IEEE-488 bus.

$$\boxed{\text{TPAS}} \wedge \text{aptmk} \wedge \text{dacr} \wedge \overline{\text{cs}} \wedge \boxed{\text{AXSS}}$$

The extra term, on the arrow between TADS and TIDS, is for secondary addressing. This is discussed in Section 12.

OTA acts like an "Untalk," as explained in the description of the IEEE-488 state diagram. TADS is analogous to the LADS at the same spot on the listener diagram. It implies that if the TMS9914A is addressed to listen, it will stop talking.

7.4 Talker Implementation

The actual implementation of the talker function is similar to the listener function. Here is the sequence, starting from power on.

1) Set up the Address Register

   a. Put the address into the five low bits of the TMS9914's Address Register.

   b. Reset bit D2 (the "dat" bit) in the Address Register. Note that the TMS9914A interfaces to the CPU via an eight-bit bi-directional data bus which is labeled D0-D7 in the pin description in the data manual. In TI terminology, D0 is the MSB and D7 is the LSB. Other manufacturers refer to these lines with the opposite terminology, thus referencing D0 as LSB and D7 as MSB. It is a common oversight to connect these data lines contrary to TI convention, thus resulting in the corruption of data.

2) Clear both Interrupt Mask registers. Interrupts are not used for this example. Neither software nor hardware reset affects the Interrupt Mask Registers; they come up in a random state.

3) Clear the "swrst" auxiliary command. The swrst is set by hardware reset and must be cleared before the TMS9914A can start talking to the IEEE-488 bus. It is cleared by writing a 0 to the Auxiliary Command Register.

4) Wait for the BO (Byte Out) bit in Interrupt Register 0 to go high. BO indicates the TMS9914A has been addressed, has gone into TACS (Talker Active State), and is waiting for a byte. Write the byte to the Data register, wait for the BO bit to be set, write a byte, etc. The BO bit is cleared by reading from Interrupt Register 0 and writing to the Data Register.

The TMS9914A handles detecting the address and switching between states. It also sends the bytes out.

8. Controller Mode

The controller state diagram is more complicated than the talker or listener diagrams. The controller has Idle, Addressed and Active states, like talker and listener. There is also a transfer control state, and many standby states. Parallel Poll has some states here, which are discussed in Section 11.

8.1 Controller Description

The translation of the controller state diagram requires Figure 8-1 and Figure 8-2. Figure 8-1 shows the addressing and unaddressing of the controller, which will be considered first.

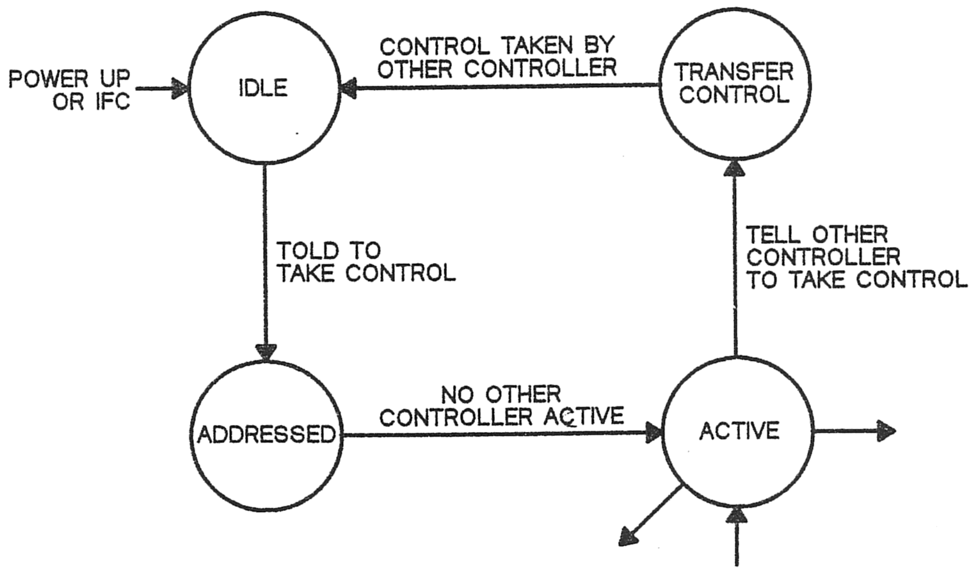


Figure 8-1. Simplified Controller State Diagram, Part 1

The controller reaches the idle state from power on or IFC, goes to the addressed state when it is told to, and goes to the active state when no controller is on the bus. This is the same as the talker and listener states, but the controller cannot retrace its steps back through addressed to idle as the previous two can. The controller normally returns to idle only if it transfers control to another controller. In systems with only one controller, the controller never returns to idle after power on.

To transfer control, the active controller tells the new controller to take control. This moves the active controller to transfer control state. When the next controller receives the message, it goes to addressed state, and acknowledges receipt of the message. The old controller receives the acknowledgment and goes to the idle state, releasing the bus. When the bus is released, the new controller goes to the active state. The arrows pointing out of the diagram around the active state are shown in Figure 8-2.

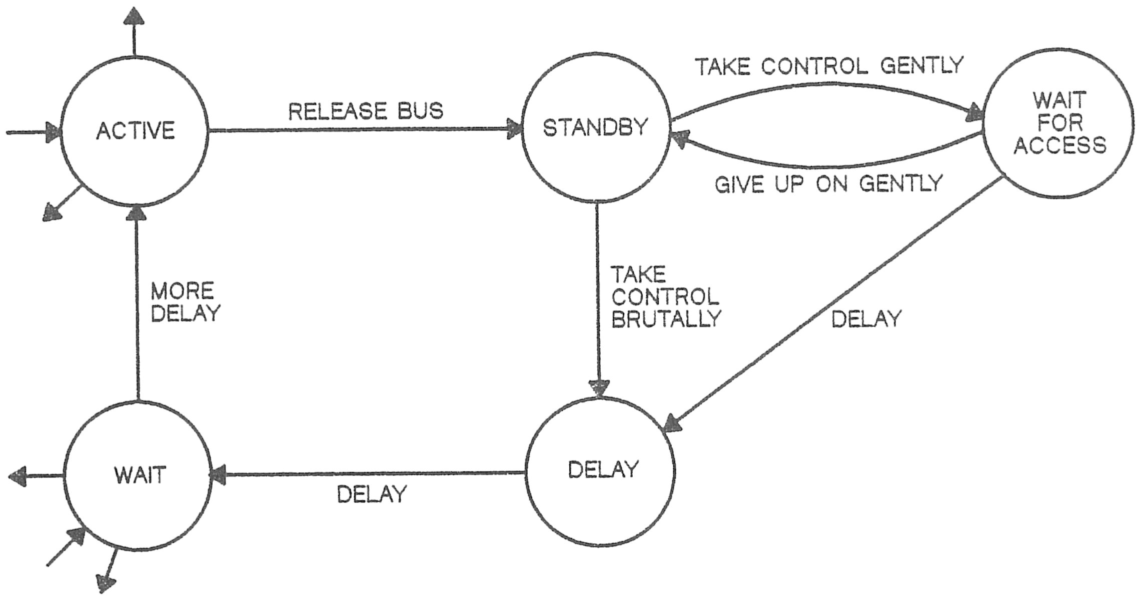


Figure 8-2. Simplified Controller State Diagram, Part 2

The active state in the lower right of Figure 8-1 is the same as the active state in the upper left of Figure 8-2. In a typical application, the controller spends most of its time in the active and standby states. In the active state, the controller is sending commands. After it has set everything up, it goes to standby and lets the talker and listener interact.

The controller can take control again in two different ways. It can wait for the time-slot after the listener has accepted a byte, before the listener is ready for the next byte, and easily take control. This method does not always work. Often the talker has sent all its data, the listener is ready for more, and the controller will never get control. The controller can also take control immediately, but this is risky because a data byte may look like a command.

Delays between taking and attaining full control prevent a data byte from being read as a command, and these delays give the talker time to get off the bus to eliminate bus conflicts. The arrows to the left of Figure 8-2 are for parallel poll.

8.2 IEEE-488 Controller State Diagram

Figure 8-3 shows the IEEE-488 state diagram for the controller function. Start at the idle state. A "pon" (power on) is there. IFC has another term with it.

$$\text{IFC} \wedge \overline{\text{(SACS)}}$$

SACS (System Control Active State) implies the interface is the system controller. A given system has only one system controller. The system controller does not have to be the active controller, but it is the only interface which can send the IFC and

REN (Remote Enable) commands. When the system controller sends the IFC, all other controllers are set to the idle state, but the system controller goes to the addressed state. This is shown by a term on the arrow from CIDS to CADS.

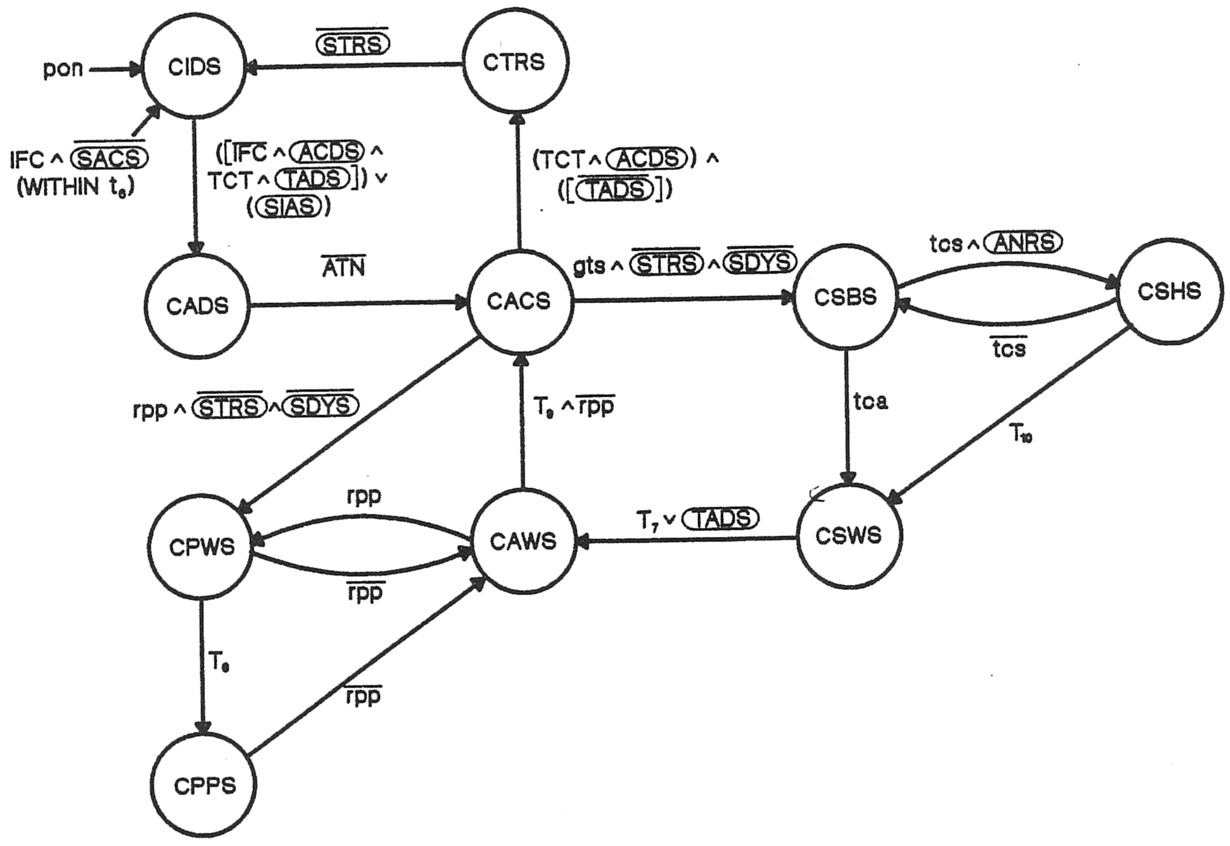


Figure 8-3. IEEE-488 Controller State Diagram

Note:

Figure 8-3 shows only the main controller diagram. Some of these mnemonics are for subsidiary diagrams. For a complete diagram, see the ANSI/IEEE Standard 488-1978.

Table 8-1. IEEE-488 Controller Mnemonics

MESSAGES	INTERFACE STATES
pon = Power on	CIDS = Controller idle state
rsc = Request system control	CADS = Controller addressed state
rpp = Request parallel poll	CTRS = Controller transfer state
gts = Go to standby	CACS = Controller active state
tca = Take control asynchronously	CPWS = Controller parallel poll wait state
tcs = Take control synchronously	CPPS = Controller parallel poll state
sic = Send interface clear	CSBS = Controller standby state
sre = Send remote enable	CSHS = Controller standby hold state
IFC = Interface clear	CAWS = Controller active wait state
ATN = Attention	CSWS = Controller synchronous wait state
TCT = Take control	CSRS = Controller service requested state
	CSNS = Controller service not requested state
	SNAS = System control not active state
	SACS = System control active state
	SRIS = System control remote enable idle state
	SRNS = System control remote enable not active state
	SRAS = System control remote enable active state
	SIIS = System control interface clear idle state
	SINS = System control interface clear not active state
	SIAS = System control interface clear active state
	ACDS = Accept data state (AH function)
	ANRS = Acceptor not ready state (AH function)
	SDYS = Source delay state (SH function)
	STRS = Source transfer state (SH function)
	TADS = Talker addressed state (T function)

SIAS (System Control Interface Clear Active State) implies the system controller is sending Interface Clear, which is normal on power up. The system controller sends an Interface Clear message, putting itself into CADS. Since ATN is not asserted, the system controller goes right from CADS to CACS. Another way for a controller to be addressed and active is:

$$[\overline{\text{IFC}} \wedge (\text{ACDS}) \wedge \text{TCT} \wedge (\text{TADS})]$$

In other words, if an IFC command is not on the bus, if the interface is addressed as a talker (TADS), and if the interface receives a TCT (Take Control) command, the interface will go to CADS. ACDS implies acceptance of the command. This term is used for transfer of control. The active controller addresses the new controller as a talker, then sends it a TCT command, causing it to go into CADS. When the active controller gets off the bus, the new controller goes to CACS.

The arrow from CACS to CTRS shows the other half of the process. The expression should have been written:

$$\text{TCT} \wedge (\text{ACDS}) \wedge [\overline{(\text{TADS})}]$$

The TCT ∧ ACDS represents the active Controller receiving its own TCT command. The optional $\overline{\text{TADS}}$ is a hold-over from the early TTL implementations of the standard. It has no other significance. Now the active controller goes to CIDS, using:

$$\overline{(\text{STRS})}$$

STRS is the state on the Source Handshake state diagram where the data byte is transferred. In this case, the data byte is really the TCT command, and the active controller is waiting until the byte is completely transferred. When the TCT is transferred, the active controller goes to CIDS, releasing the ATN line. This causes the new controller to go from CADS to CACS and assert the ATN line itself.

Transfer of control does not happen often. A controller will generally be in CACS issuing commands, or in CSBS (Controller Standby State), letting a talker talk to a listener. One way to go from CACS to CSBS is:

$$\text{gts} \wedge \overline{(\text{STRS})} \wedge \overline{(\text{SDYS})}$$

A "gts" is a local message that means "go to standby." STRS and SDYS are states from the Source Handshake diagram that ensure that the last command is accepted before the controller goes to standby. The controller will normally sit in CSBS, waiting for the data transfer between the talker and listener to complete. When it is ready to go back to CACS, it has two routes to choose from. The route to CSHS is:

$$\text{tcs} \wedge (\text{ANRS})$$

The "tcs" is a local message meaning "take control synchronously" and ANRS is used for synchronization. ANRS (Acceptor Not Ready State) is an Acceptor Handshake state. The Acceptor is in this state after it has accepted a data byte, and before it is ready for the next byte. This is the one time the controller can take control of the bus with no chance of data corruption, because no data are on the bus.

Another way to take control of the bus is:

$$\text{tca}$$

If a "tca" (take control asynchronously) local command is issued while in CSBS, the controller goes to CSWS. If a data byte is being transferred, it may be interpreted as a command instead. The command "tca" is used when the data transmission has stopped. The talker has sent its last byte, the listener has accepted it and is ready for additional data. In this case, a "tcs" will lock up. The listener will wait forever for the next data byte to be transferred. In our sample system, the voltmeter sends the voltage to the printer and then stops sending. A "tcs" would not work for taking control after the transmission was done.

In CSWS, the controller asserts the ATN line. A delay from CSHS to CSWS helps prevent data corruption. Going from CSWS to CAWS, a delay occurs if the controller is not the talker. This delay allows the talker to recognize the ATN and get off the bus. The delay between CAWS and CACS prevents a false indication of Parallel Poll. The TMS9914A takes care of all these delays.

CPWS and CPPS are Parallel Poll states, which are discussed in Section 11.

8.3 TMS9914A State Diagram

The TMS9914A controller diagram, shown in Figure 8-4, is simpler than the IEEE-488 controller diagram. The full IEEE-488 controller function can be implemented on the TMS9914A with the proper software.

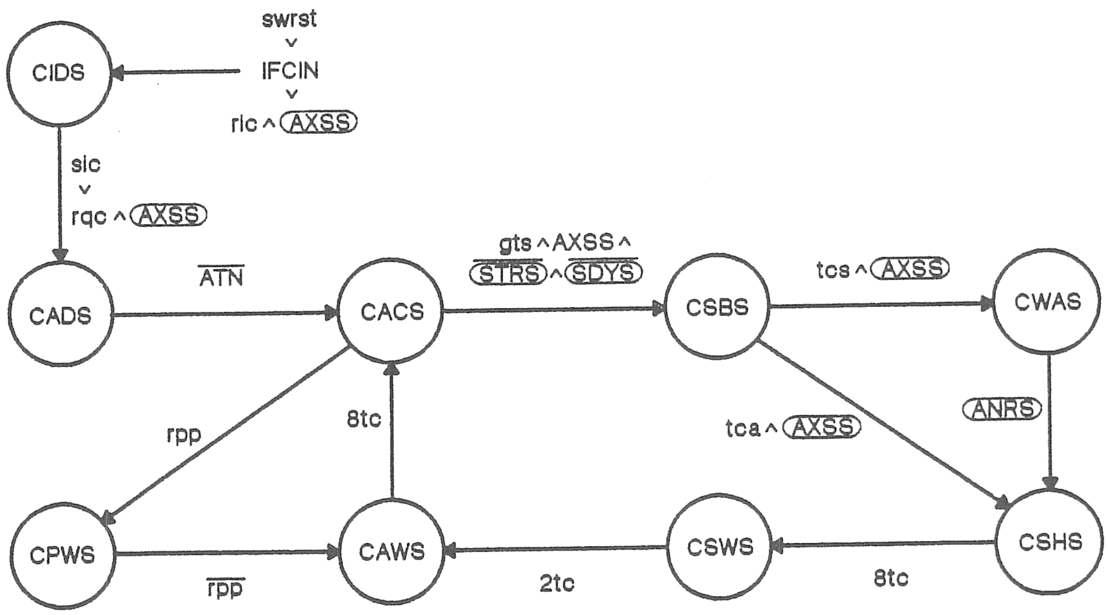


Figure 8-4. TMS9914A Controller State Diagram

Table 8-2. TMS9914A Controller Mnemonics

MESSAGES	INTERFACE STATES
gts = Go to standby	CACS = Controller active state
rlc = Release control	CADS = Controller addressed state
rpp = Request parallel poll	CAWS = Controller active wait state
rqc = Request control	CIDS = Controller idle state
sic = Set interface clear	CPWS = Controller parallel poll wait state
swrst = Software reset	CSBS = Controller standby state
tca = Take control asynchronously	CSHS = Controller standby hold state
	CSWS = Controller synchronous wait state
tcs = Take control synchronously	CWAS = Controller wait for ANRS state
ATN = Attention	ANRS = Acceptor not ready state
IFCIN = Internal IFC	AXSS = Auxiliary command strobe state
	SDYS = Source delay state
	STRS = Source transfer state

Consider the idle state, CIDS, in the upper left corner of Figure 8-4. IFCIN and "swrst" should be familiar. A "swrst" is an auxiliary command that occurs at power up, so it is similiar to the IEEE-488 "pon." IFCIN is almost the same as IFC. Another way to get to CIDS is:

$$rlc \wedge (\text{AXSS})$$

An "rlc" (release control) is an auxiliary command used during transfer of control. The active controller waits until the new controller has acknowledged receipt of the TCT command, and then issues an "rlc." The new controller should detect that TCT has been sent to it and issue an "rqc" (request control). Then it will acknowledge the TCT command (see Section 8.4). The "rqc" command is on the arrow between CIDS and CADS.

An "sic" is also on the arrow from CIDS to CADS. The system controller uses "sic" (set interface clear) to send out an IFC and to take control of the bus. Normally this is done on power up.

The remainder of the diagram is like the IEEE-488 diagram. The "gts" auxiliary command gets the controller to standby, and the "tca" and "tcs" auxiliary commands work like the "tca" and "tcs" remote messages from the IEEE-488 diagram. There are many delays before getting to CACS again. The organization of the states is not quite the same, but the delays between states are as long as the minimum required. CPWS is used for Parallel Poll.

8.4 Controller Implementation

The controller function implementation is more complicated and critical than the functions discussed so far.

First consider the system controller, starting from power up. When the TMS9914A is given a hardware reset, it enables "swrst," and puts the controller function into CIDS.

Before releasing "swrst," initialize the TMS9914A. The interrupt registers are not initialized by reset, so they need to be initialized. Because interrupts are not used for the example, write zeroes to both Interrupt Registers. The Address Register must be initialized, though it has no effect on the controller function. The talker and listener functions still need to be instructed what to do. See Section 6 and Section 7 on listener and talker. Also send the "sic" command to the TMS9914A before releasing "swrst."

To begin, send an "swrst" command to the TMS9914A with the "cs" bit cleared. This releases "swrst," and gets the TMS9914A to CADS. Since no controller is on the bus, the TMS9914A goes straight to CACS. The BO bit will go high when entering CACS, but bytes cannot be sent immediately. A 100-µs delay is necessary here, because this is the minimum width allowed for IFC. While delaying, issue an "sre" (send remote enable) auxiliary command. This will turn on the REN (Remote Enable) line of the IEEE-488 bus. Some listeners will not use any data sent without Remote Enable. It is customary just to turn Remote Enable on at power up and leave it on at all times.

After the 100-µs delay, send an "sic" command to the TMS9914A with the "cs" bit cleared. This turns IFC off. BO has been indicated, so the first command byte can be sent out. Commands go out through the data register, just like data. On each successive BO, another command can be sent.

To become a listener when control is released, an "lon" auxiliary command is issued while still in CACS (for a talker, send "ton"). With talker, also issue a MTA command with our address. This will not affect the interface, but is necessary to meet the IEEE-488 specification, and to ensure that all other talkers are off the bus. After sending the last command down the bus, (and getting the next BO), issue a "gts" (go to standby) auxiliary command, causing the TMS9914A to go to CSBS.

When going into talker or listener, the appropriate bit (BO or BI) will be set to tell when one can start talking or listening.

Issue the "tca" (take control asynchronously) auxiliary command to take control. If the talker issues the "tca," it must wait for a BO after the last byte sent. If the listener issues the "tca," it must wait for the last byte to come in. Note that a talker or listener going into CACS will be the same on leaving CACS, unless a "lon" or a "ton" is issued with the "cs" bit cleared.

A "tcs" (take control synchronously) is more complicated. Controllers acting as talkers cannot use "tcs" (and do not need to). Only controllers acting as listeners can issue a "tcs" because only listeners are synchronized to the handshake. The only proper use for "tcs" is the interruption of a string of data. If "tcs" is needed, consider these points:

1) Before going to standby, issue a "lon." If the listener does not need to read the data on the bus, issue a "shdw" (shadow handshake) command after the "lon." A "shdw" will handshake each data byte immediately, without waiting for it to be read.

2) To interrupt the data stream, issue a "tcs." If a BO is not received after a reasonable amount of time, try a "tca." A "reasonable amount" is determined by the speeds of the talker and listener that are communicating. When BO is true, the state is CACS, and one can begin sending commands.

The procedure for transfer of control is relatively easy. The active controller should start in CACS, and the new controller in CIDS. The new controller must have the "unc" bit set in its Mask Register 1. The active controller must address the new controller as a talker (MTA). After the BO bit goes high, the active controller should send out a TCT (take control) command.

This will cause the "unc" (unrecognized command) bit in the new controller's interrupt register 1 to go high. The "unc" also causes the TMS9914A to hold the DAC (Data Accepted) signal false, telling the active controller that the byte has not been read yet. The new controller will read this bit and then read the TCT from its command pass-through register. It will then verify that its TADS (Talker Addressed State) bit in the Address Status Register is set, and issue a "rqc" (request control) auxiliary command. This will move the new controller to the CADS state. The new controller then must issue a "dacr" (release DAC holdoff) auxiliary command, to signal that it has accepted the TCT command.

When the new controller releases its DAC holdoff, the BO bit in the active controller will go high. At this point, the active controller issues an "rlc" auxiliary command, sending itself to CIDS and allowing the new controller to go to CACS.

9. Command Implementation

Now the physical implementation of commands will be discussed in more detail. There are two types of commands on the IEEE-488 bus. There are eight control lines for the bus, each of which can be used for a command or a request. In one case, two of them are used together as a command; most commands, however, are transmitted on the eight data lines of the bus. These commands are distinguished from data by the ATN line. The following paragraphs describe the commands.

9.1 Negative Logic

The lines on the IEEE bus use negative logic. This implies that a true state is a low voltage, and a false state is a high voltage. Since driver chips 75160A, 75151A, and 75162A for use with the TMS9914A are non-inverting, the lines coming out of the TMS9914A are negative logic.

9.2 Control Lines

The current state of the control lines can be read from the bus Status Register on the TMS9914A, which is useful primarily for debugging. Use it with care, because the lines can change while being read.

9.2.1 IFC

IFC (Interface Clear) is a line that can be asserted only by the system controller. Everything else on the bus must go idle when IFC is asserted.

9.2.2 REN

REN (Remote Enable) is used to tell the systems receiving data that they can actually use the data, and that their front panel controls are locked out. It also is asserted only by the system controller.

9.2.3 ATN

The Attention line is asserted by the currently active controller. When asserted, it means that the byte on the eight data lines is an interface command. If ATN is not asserted, the byte is device-dependent data.

9.2.4 EOI

EOI (End or Identify) can be used by a talker to indicate the end of a multiple byte data string. The talker issues the "feoi" (Forced End or Identify) auxiliary command just before the last byte is sent. Sending the last byte sets the EOI line. The listener will set the end bit in Status Register 0, which will indicate EOI has been received. When EOI is asserted with the ATN line, it indicates the controller is conducting a parallel poll.

9.2.5 SRQ

SRQ (Service Request) can be asserted by any device requiring service. It is like an interrupt line on a microprocessor bus. The controller can respond to a SRQ in different ways or not at all. Service request is discussed further in the Section 10.

9.2.6 RFD, DAV, and DAC

Negative logic causes some problems with nomenclature. The commands are called RFD and DAC, but the physical lines are called NRFD and NDAC. A low voltage on the NRFD line (a true in negative logic) implies "not ready for data." RFD and DAC are the only two commands with this problem. Fortunately, the TMS9914A always handles the handshake without any intervention.

These commands are used for data and command handshaking. NRFD (Not Ready For Data) and NDAC (Not Data Accepted) are lines controlled by the listeners. They are driven by open collector devices, so that all listeners must respond before the line can go high. DAV (Data Valid) is controlled by the talker. A typical data handshake has these characteristics:

1) NDAC goes false after all the listeners have accepted the last byte.

2) The talker puts the new data byte on the bus.

3) NRFD goes false after all the listeners are ready for the next byte.

4) The talker asserts DAV.

5) The listeners read in the data byte and then allow NDAC to go false. When all the listeners have done so, NDAC goes false.

6) The talker puts the next byte on the bus.

7) Repeat the sequence.

9.3 Command Bytes

Command bytes use the low seven bits of the data lines. The most significant bit is ignored by the interfaces receiving the command. All numbers in the following section are hexadecimal. Table 9-1 lists the command bytes that a controller can send out.

Devices can have addresses from 0 to 1E. MLA commands go from 20 to 3E. A 20 is used to tell device 0 to listen, 21 for device 1, and so on. MTA is similar; it uses 40 to 5E to tell devices 0 to 1E to talk. OTA (Other Talk Address) refers to the same codes. Device 0 is addressed to talk by a 40, so for device 0, 40 is an MTA. For all other device addresses, 40 is an OTA.

SCG (Secondary Command Group) goes from 60 to 7E. These commands follow an earlier command that gives them meaning. If they follow an MTA or MLA, they extend addressing; if they follow a PPC (Parallel Poll Configure), they enable or disable a parallel poll response. The details of both these topics will be discussed later.

GTL (Go To Local) enables local control. Local control is control of an instrument from its front panel. LLO (Local Lockout) makes it possible to prevent local control so that the bus can completely control the device. See the sections on remote and local in the TMS9914A Data Manual (p/n MP033A/SPOU002) and the 1978 IEEE-488 specification for more information. The SDC (Selected Device Clear) command is similar to DCL but works only on devices addressed as listeners.

Table 9-1. Command Bytes

HEX VALUE	COMMAND	COMMAND MEANING	SECTION
1	GTL	Go to local	--
4	SDC	Selected device clear	--
5	PPC	Parallel poll configure	11
8	GET	Group execute trigger	4
9	TCT	Take control	8
11	LLO	Local lockout	--
14	DCL	Device clear	4
15	PPU	Parallel poll unconfigure	11
18	SPE	Serial poll enable	10
19	SPD	Serial poll disable	10
20 - 3E	MLA	My listen address	6
3F	UNL	Unlisten	6
40 - 5E	MTA	My talk address	7
	OTA	Other talk address	7
5F	UNT	Untalk	7
60 - 7E	SCG	Secondary command group	9
	MSA	My secondary address	12
	PPD	Parallel poll disable	11
	PPE	Parallel poll enable	11

10. Service Request/Serial Poll

Service Request and Serial Poll functions interact closely. Normally, when a controller receives a service request, it does a serial poll to find out who is requesting service. Special features in the TMS9914A help with this process.

This part of the book dispenses with the previously-used presentation of the state diagrams in three parts (the simplified diagram, the IEEE diagram, and the TMS9914A diagram) and shows only the TMS9914A diagrams.

10.1 Requesting Service

SRQ (Service Request) is one of the control lines on the IEEE-488. Any device requiring service can pull it low. On the TMS9914A, there are two ways to request service.

rsv1 To use "rsv1," set bit 1 of the serial poll register. After the TMS9914A has been serviced, reset bit 1. Remember, bit 0 is the most significant bit on the TMS9914A.

rsv2 To use "rsv2," issue the "rsv2" auxiliary command. An "rsv2" will be automatically cleared after the serial poll, eliminating the need to reset it.

To receive an SRQ, the controller should monitor the SRQ bit in interrupt register 1. If this bit is set, the controller should execute a serial poll.

10.2 Serial Polling

The functions interacting here are controller, talker, and service request. Refer to the state diagrams at the appropriate times. Figure 10-1 shows the Service Request Diagram.

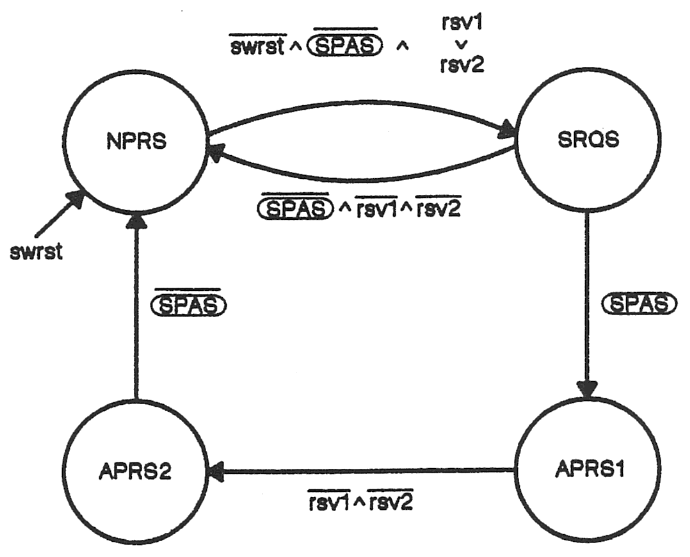

$\overline{swrst} \wedge \overline{(SPAS)} \wedge (rsv1 \vee rsv2)$
NPRS
SRQS
$\overline{(SPAS)} \wedge \overline{rsv1} \wedge \overline{rsv2}$
swrst
$\overline{(SPAS)}$
(SPAS)
APRS2
APRS1
$\overline{rsv1} \wedge \overline{rsv2}$

Figure 10-1. TMS9914A Service Request State Diagram

Table 10-1. TMS9914A Service Request Mnemonics

MESSAGES	INTERFACE STATES
rsv1 = request service 1	APRS1 = affirmative poll response state 1
rsv2 = request service 2	APRS2 = affirmative poll response state 2
swrst = software reset	NPRS = negative poll response state
	SPAS = serial poll active state

The list below describes what the controller must do to execute a serial poll.

1. **Take control** The Controller must be active to poll, so if control has not been taken, take it first. See Section 8.

2. **Send an SPE** An SPE (Serial Poll Enable), will put the device into SPMS (Serial Poll Mode State).

3. **Send its MTA** MTA (My Talk Address) will send the device being polled into TADS (Talker Addressed State). See the talker state diagram in Section 7 for details.

4. **Issue an "lon"** The controller must listen to the Serial Poll response, and the "lon" sets it up to do this.

5. **Issue a "gts"** The "gts" (go to standby) auxiliary command will release control of the bus, causing ATN to go false. Since SPMS is true, the talker will go from TADS to SPAS. In SPAS, it will put its serial poll register on the bus whenever another device wants to read the register.

6. **Wait for a BI** Wait for the device to send its Serial Poll byte.

7. **Issue a "tcs"** The TMS9914A has latched the byte in, so control can be taken again.

8. **Read data** Now that control has been taken, read the data. If the data was read before issuing "tcs," the talker would have sent the byte again, and the TMS9914A would have latched it again. The data would then wait there and be confusing later. If this device is requesting service, bit 1 of the byte will be set. Whatever service it requires may be performed. If bit 1 is not set, the controller polls each of the next devices until the one that requested service is found.

9. **Send a SPD** SPD (Serial Poll Disable) tells all the talkers to leave SPMS (Serial Poll Mode State). This way, the next time they are addressed to talk, they will not send their Serial Poll byte, but device-dependent data.

10. **Handle request** The instrument manufacturer defines the nature of the request and the service required. The other seven bits of the Serial Poll byte can be used as status bits, indicating the type of service required. On the TMS9914A, the Serial Poll Register is automatically put on the bus whenever the device is polled. Remember that the controller is currently ready to listen and the device requiring service is ready to talk. It may be necessary to change these states before handling the request.

10.3 Turning Service Request Off

The actions of the device requesting service are described here.

Figure 10-1 on page 10-2 shows the TMS9914A state diagram for Service Request. In NPRS (Negative Poll Response State), the SRQ line is not pulled low, nor will bit 1 be sent true for a serial poll. When either "rsv1" or "rsv2" is set, the state moves to SRQS, where SRQ line will be pulled low. When polled (SPAS), the device goes to APRS1, the SRQ line is released, and bit 1 of the Serial Poll register is sent as true. If "rsv2" is being used, as soon as the controller accepts the status byte, "rsv2" is cleared, moving the device to APRS2. At the same time, the SPAS bit in Interrupt Register 0 will be set to indicate this has happened. But an "rsv1" is not automatically cleared, so the TMS9914A stays in APRS1 until "rsv1" is cleared, then goes to APRS2.

When exiting SPAS, the TMS9914A goes from APRS2 to NPRS, where it started from. The "rsv2" was developed to solve a potential problem with "rsv1." If the controller polls and services one device, but that device does not clear "rsv1" immediately, it will still be in APRS1, ready to send its status byte with bit 1 high. If a second device then requests service, the controller will read from the status register of the first device, and will think it requires service again. Use "rsv2" to avoid this situation.

11. Parallel Poll

Parallel Poll is used when the controller needs a fast report on the status of the devices on the bus. Up to eight devices can be programmed to set a bit on the IEEE-488 bus in response to a Parallel Poll. When the controller performs a Parallel Poll, it reads in the data byte, looks at the bits, and immediately identifies which devices need attention. The devices can be told which bit to set either locally with switches, or remotely with commands.

11.1 Parallel Poll Configuration

The TMS9914A puts the contents of its Parallel Poll register on the bus when a Parallel Poll is started by the controller. For a local configuration within the instrument, the microprocessor can read the number of the desired bit from the switches and set that bit in the Parallel Poll register when the device needs attention.

Remote configuration is more complicated. First, remote configuration will be shown from the controller's end. Then it will be shown from the point of view of the devices being polled. Figure 11-1 shows the IEEE-488 state diagram for Parallel Poll configuration.

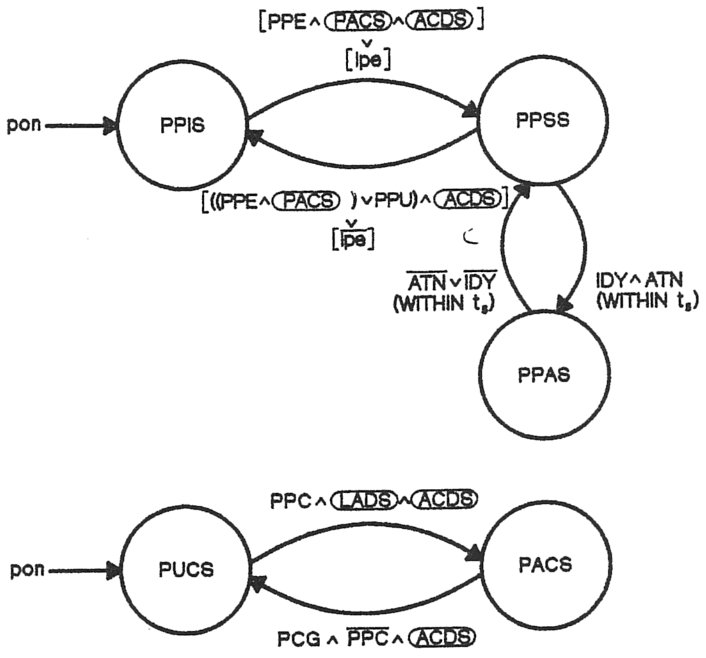


Figure 11-1. IEEE-488 Parallel Poll State Diagram

Table 11-1. IEEE-488 Parallel Poll Mnemonics

MESSAGES	INTERFACE STATES
pon = Power on	PPIS = Parallel poll idle state
ist = Individual status (Table 25)	PPSS = Parallel poll standby state
lpe = Local poll enabled	PPAS = Parallel poll active state
ATN = Attention	PUCS = Parallel poll unaddressed configure state
IDY = Identify	PACS = Parallel poll addressed to configure state
PPE = Parallel poll enabled	ACDS = Accept data state (AH function)
PPD = Parallel poll disable	LADS = Listener addressed state (L function)
PPC = Parallel poll configure	
PCG = Primary command group	
PPU = Parallel poll unconfigure	

Here is a description of the controller actions required for Parallel Poll configuration.

1. Take control This is necessary to send commands. Take control of the bus with a "tca" or "tcs." (See Section 8 for a review.) Note that it is necessary to wait for the BO bit to go true before sending each command.

2. Send an MLA The Parallel Poll configuration commands are "addressed" commands, which implies they affect only those devices addressed as listeners. MLA (My Listen Address) tells the device being configured to listen.

3. Send a PPC PPC (Parallel Poll Configure) tells the listener the configuration data will follow.

4. Send a PPE or PPD A PPE (Parallel Poll Enable) tells the device being configured which bit to set for its parallel poll response. The three least significant bits of the PPE tell which bit to set. The fourth bit indicates whether the bit should be set or reset to signal that the device needs attention. A PPD tells the device not to respond to a parallel poll. PPE and PPD are both "secondary commands." Their meaning is determined by the command that precedes them. When preceded by a PPC, they are PPE and PPD. Section 12 describes what happens when the secondary commands are preceded by MTA or MLA.

5. Send a UNL A UNL (Unlisten) tells the device being configured to stop listening so that reconfiguration will not occur with the next PPE.

6. Send a new MLA Now a new device can be configured. First it must be made to listen with MLA, and then it can be sent a PPC, PPE, and UNL, as with the first device. Now continue through the devices.

A PPU (Parallel Poll Unconfigure) command can be issued by the controller, telling all devices not to respond to parallel poll. It is an unaddressed command, implying that it affects all devices.

Now the parallel poll configuration will be discussed from the point of view of the device being configured. This function does not have a TMS9914A state diagram; it requires support from device software. Here is a typical sequence of events:

1. **Set the UNC bit** — The UNC bit in Interrupt Mask Register 1 must be set by the microprocessor to enable the holdoff discussed below.

2. **Detect UNC bit set** — Receipt of the PPC (Parallel Poll Configure) command will cause the UNC (Unrecognized Command) bit in the TMS9914A Interrupt Status Register 1 to be set. The UNC bit will be set only if the TMS9914A is in LADS, which implies the TMS9914A has been addressed to listen. The PPC will also cause a DAC (Data Accepted) holdoff if the UNC bit in the Interrupt Register has been unmasked, keeping the PPC command on the bus.

3. **Read in PPC** — The device being configured should read the command pass-through register of the TMS9914A, to determine that the command is a PPC.

4. **Issue "pts"** — The "pts" (pass through secondary) auxiliary command will cause the next command to set the UNC bit if the next command is a secondary command (a PPE or PPD).

5. **Issue "dacr"** — The "dacr" (DAC holdoff release) auxiliary command releases the bus so the controller can send the next byte.

6. **Wait for UNC**

7. **Read PPE or PPD** — After the UNC, it is necessary to read the command pass-through register again, to determine the secondary command.

8. **Set up PP register** — If the secondary command is a PPE or PPD, set the bits in the Parallel Poll register accordingly. For a PPD, all the bits in the PP register should be cleared. A PPE is more complicated. A bit in the Parallel Poll register should be set or reset. The bit number is given by the three least significant bits in the PPE, and the fourth bit is used to indicate the encoding of the bit. For explanation, use the example with the voltmeter. The voltmeter responds affirmatively to parallel poll if it has taken a voltage reading and not put it out over the bus yet. If the voltmeter gets a PPE with the fourth bit <u>set</u> and the low three bits set to the value of two, it will <u>set</u> bit two in its Parallel Poll register when it has taken a voltage reading and <u>reset</u> bit two when the reading has been sent out on the bus. However, if the fourth bit was <u>reset</u> in the last PPE, it will <u>reset</u> bit two when it has voltage data and <u>set</u> it when the data have been sent.

9. **Issue "dacr"** — Once the Parallel Poll register is set up properly, the DAC holdoff must be released so that the bus can start running again.

10. Update PP register	Now that the device has been configured for parallel poll, the state of the bit in the Parallel Poll register must be changed each time the state of the instrument changes. When the TMS9914A is parallel polled, it automatically puts the contents of the Parallel Poll register on the bus.

11.2 Parallel Polling

Now that a device has been configured correctly, a parallel poll can be performed. The steps are:

1) The controller must take control of the bus and issue an "rpp" (request parallel poll) auxiliary command. Wait for the BO (Byte Out) bit to be set to be assured the bus is available.

2) Issue the "rpp" with the CS bit set and read the command pass-through register to get the parallel poll response from the other devices on the bus.

3) To exit parallel poll mode, issue an rpp with the CS bit cleared.

11.3 Changing Parallel Poll Response During Parallel Poll

The parallel poll response of the TMS9914A is double-buffered. This means that if a parallel poll is in process and a different value is written to the Parallel Poll register, the value will not be put on the bus until the next parallel poll. Some controllers initiate a parallel poll and then wait for one of the bits to change. The only way to meet this requirement with the TMS9914A is to:

1) Issue the "swrst" auxiliary command.

2) Re-initialize the TMS9914A to its previous state.

3) Write the new value to the Parallel Poll register.

4) Send the "swrst" auxiliary command again, with the clear/set bit cleared.

12. Extended Addressing

Extended addressing is a simple concept that is used when more than five bits of address are required. The MLA and MTA commands are used as "primary" addresses but must be followed by the MSA (My Secondary Address) command to identify the particular function. MSA can be any of the 32 secondary commands.

The TMS9914A does not handle extended addressing automatically. The APT bit in the Interrupt Mask Register must be set. If the TMS9914A sees an MTA or MLA with its address, followed by an MSA, it will:

1) Set the APT bit in the Interrupt Status Register
2) Generate an interrupt
3) Hold off the DAC (Data Accepted) handshake

Then the MSA command must be read from the command pass-through register, and the address judged valid or not. If the address is valid, send the "dacr" auxiliary command to the TMS9914A with the clear/set bit set. If the MSA address is not valid, the command should be issued with the bit cleared. After helping the TMS9914A, it will be ready to listen or talk, requiring the user only to move bytes between memory and the TMS9914A.

13. Direct Memory Access

The TMS9914A supports DMA (Direct Memory Access) data transfers for talker and listener, but not for controller. The first part of this section describes a simple approach to the software for DMA, while the second part gives pointers for calculating throughput with DMA.

13.1 DMA Software

The MA (My Address) bit in the TMS9914A interrupt register 1 is intended for use with DMA. If the MA bit is set in the interrupt mask register, an interrupt will be generated whenever the TMS9914A is addressed to talk or to listen. A DAC holdoff will also be generated. To service an MA interrupt, read from the command pass-through register to see whether the command is MTA or MLA, and initialize the DMA system appropriately. After setting up, issue a "dacr" (DAC holdoff release) auxiliary command to let the transfer begin.

After the DMA transfer is done, it is necessary to clear a line on the TMS9914A. When the TMS9914A is being used with DMA, there are two hardware lines that come into play. When the TMS9914A needs a new byte, it pulls ACCRQ (Access Request) low. When the DMA hardware is ready to send a byte to the TMS9914A, it pulls the ACCGR (Access Grant) line low and sends the byte. When the data bytes have all been transferred, the DMA hardware stops the transfer by not responding to ACCRQ with an ACCGR. This leaves the ACCRQ (Access Request) line low, causing problems at the start of the next transfer. To avoid these problems, read from the Data In register at the end of a DMA data transfer. This permits the ACCRQ line to go high again. Normally the byte in the data in register can be discarded. If the data may be valid, check the BI (Byte In) bit to see if a valid byte is waiting.

13.2 DMA Timing

It is important to be able to determine the system throughput for DMA. This section provides a framework for this, giving the timing for the TMS9914A and the place to add in the timing for the rest of the system. See Section 4 of the TMS9914A Data Manual for more details, particularly the timing diagrams.

The timing from DAV (Data Valid) going low on the talker, to the listener returning with NRFD (Not Ready For Data) high, is shown in Table 13-1.

Table 13-1. Listener Timing

DELAY TIME DESCRIPTION	TIME
Propagation delay for the SN75161A bus interface chip used to buffer the TMS9914A (See Appendix C in the TMS9914A Data Manual -- p/n MP033A/SPOU002 -- for details.)	25 ns
The delay from the receipt of DAV low at theTMS9914A to the falling edge of ACCRQ, which is "t_{d7}" in the data manual. Note that all timings are worst case, and assume a 200-ns clock cycle for the TMS9914A.	690 ns
The delay between ACCRQ going low and ACCGR going high. It is dependent on the system, but must be at least 150 ns to meet TMS9914A requirements.	sda (150 ns minimum)
The delay from the rising edge of ACCGR to the rising edge of NRFD, or "T_{d9}" in the TMS9914A Data Manual.	220 ns
The propogation delay for SN75161A on NRFD.	25 ns

These delays total 960 ns, plus "sda." This is the time between the appearance of a valid byte and the TMS9914A's indication of its readiness for another byte.

The talker timing starts with the rising edge of NDAC (Not Data Accepted), signaling that the listener has accepted the data. Table 13-2 below describes the timing.

Table 13-2. TMS9914A Talker Timing

DELAY TIME DESCRIPTION	TIME
The propagation delay of the SN75161A	25 ns
The delay between NDAC and ACCRQ ("t_{d4}" in the TMS9914A Data Manual)	300 ns
The delay for the DMA system between ACCRQ and the rising edge of ACCGR. It must be at least 100 ns to meet TMS9914A requirements	sds (100 ns minimum)
The delay from the rising edge of ACCGR to the falling edge of DAV. This delay allows for bus settling time. The timing assumes a high speed configuration of the bus. Depending on configuration, the number could be as high as 2.7 microseconds ("t_{d1}" in the TMS9914A Data Manual).	1110 ns
The propogation delay of the SN75161A.	25 ns

The total of these numbers is 1460 ns, plus "sds." The talker function on the TMS9914A is slower than the listener. Until now, the timing has been presented in halves, so the timing can be added for whatever listener or talker is on the other end.

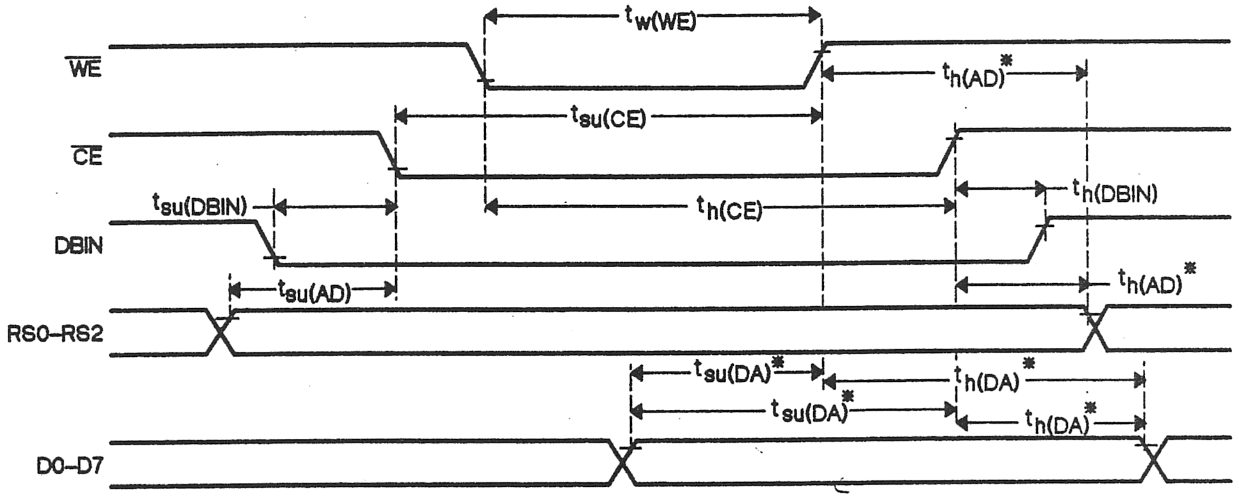
With two TMS9914As together, consider the falling edge of DAV, coming out of the SN75161A on the talker. The talker is assumed to be the limiting factor, rather than the listener. The listener still participates in the delays, though, because it must release NDAC in response to DAV. The TMS9914A needs 1045 ns to do this ("t_{d8}" in the TMS9914A Data Manual) plus 25 ns for the DAV signal to propogate through the SN75161A, and 25 ns for the NDAC signal to propogate back out. This is a total of 1095 ns. As discussed previously, the time from NDAC to DAV for the TMS9914A talker is 1460 ns plus "sds."

The total time per byte is 2555 ns plus "sds." If the DMA hardware is fast, "sds" could be as low as 200 ns, so that the total time is 2755 ns. This leads to a data rate of 360 kilobytes per second, which is worst case. Typical data rates with this configuration could be higher.

TMS9914A Hardware Interface

14.1 Write Timing

Figure 14-1 is a corrected timing diagram for write timing. Two inaccuracies occur in the TMS9914A Data Manual (MP033A/SPOU002). The parameter $t_{su(CE)}$ is measured from the falling edge of CE to the rising edge of WE. The time shown as $t_{w(CE)}$ in the TMS9914A Data Manual is actually $t_{h(CE)}$.



* $t_{su(DA)}$, $t_{h(DA)}$, and $t_{h(AD)}$ are only applicable to the first signal to become inactive, whether it is $\overline{WE}$ or $\overline{CE}$.

Figure 14-1. TMS9914A Write Cycle Timing

14.2 Interrupts

The INT line on the TMS9914A goes low to indicate an interrupt. It goes high when all interrupt conditions have been cleared. A new interrupt condition can occur while the previous interrupt condition is still being processed. Although the first interrupt condition is cleared, the INT line will still be low. The interrupt hardware must be designed to handle this possibility. It should respond to the level of the INT line, not to a falling edge alone.

14.3 Unintentional Clearing of Status Bits

The interrupt status bits are cleared by reading them. The hardware designer using the TMS9914A should make sure these registers are not read from inadvertently. Some processors do a read before each write. With these processors, writing to the Interrupt Mask registers will also clear the interrupt status bits, since they have the same address. This causes no problems if the interrupt masks are modified only during initialization. If the masks are changed during the operation of the device, solution, which is to use two different addresses for the TMS9914A: one for reading, and one for writing. Otherwise, the programmer will have to read the register and save the state of the bits before each operation that will read that register.

14.4 Power-Up Self Test

A useful power-up self test is not possible on the TMS9914A. Since what is out on the IEEE-488 bus is not known, the bus cannot be used to perform a test. All other devices put between the SN7516X devices and the TMS9914A to facilitate a self-test will slow the process. The TMS9914A is guaranteed to be fast enough to meet the IEEE-488 specification when used with SN7516X devices, but not with anything else.

A. Glossary

ACCGR: Access Grant; a pin on the TMS9914A.

ACCRQ: Access Request; a pin on the TMS9914A.

ACDS: Accept Data State; state in Acceptor Handshake state diagram.

AH: Acceptor Handshake; listener handshake function.

ANRS: Acceptor Not Ready State; an Acceptor Handshake state. The Acceptor is in this state after it has accepted a data byte and before it is ready for the next byte.

ATN: Attention line; when asserted, indicates that the byte on the eight data lines is a command. If ATN is not asserted, the byte is device-dependent data.

BI: Byte In; bit in Interrupt register 0.

BO: Byte Out; bit in Interrupt register 0.

CACS: Controller Active State; state on controller state diagram in which commands can be sent.

CADS: Controller Addressed State; state on controller state diagram.

CIDS: Controller Idle State; state on controller state diagram.

CSBS: Controller Standby State; state on controller state diagram.

DAC: Data Accepted; IEE-488 bus signal used for data and command handshaking; is controlled by the listeners.

dacr: DAC holdoff release; auxiliary command.

dal: disable listener; local message which is bit D1 of the address register.

dat: disable talker; local message which is bit D2 of the address register.

DAV: Data Valid; IEEE-488 bus signal used for data and command handshaking; controlled by the talker or controller.

DCL: Device Clear; IEEE-488 bus command.

DMA: Direct Memory Access

EOI: End or Identify; IEEE-488 bus signal that can be used by a talker to indicate the end of a multiple- byte data string.

GET: Group Execute Trigger; IEEE-488 command.

GTL: Go To Local; IEEE-488 command enables local control.

gts: go to standby; auxiliary command.

IFC: Interface Clear; IEEE-488 bus line that can be asserted only by the system controller. Everything else on the bus must go idle when IFC is asserted.

LACS: Listener Active State; active state in the listener state diagram where data can be listened to.

LADS: Listener Addressed State; state in the listener state diagram.

LLO: Local Lockout; command that permits local control to be locked out.

LSB: Least Significant Bit; the LSB of the TMS9914A is D7.

LIDS: Listener Idle State; state in the listener state diagram.

Listening: Receiving data

Local control: Control of an instrument from its front panel.

lon: listen only; local message sent through the auxiliary command register. It has a clear/set bit associated with it.

MA: My Address; bit in Interrupt register 1.

MSA: My Secondary Address; command that can be any of the 32 secondary commands.

MSB: Most Significant Bit; the MSB of the TMS9914A is D0.

MLA: My Listen Address command.

MTA: My Talk Address command.

NDAC: Not Data Accepted; IEEE-488 line.

NPRS: Negative Poll Response State; state in the TMS9914A service request state diagram.

NRFD: Not Ready For Data; IEEE-488 line.

OTA: Other Talk Address; tells the talker to go to an idle state because only one talker can be active at a time.

pon: power on.

PPC: Parallel Poll Configure; command that tells all devices addressed as listeners to expect a PPE or PPD.

PPD: Parallel Poll Disable; IEEE-488 command that tells the device being configured to disable parallel poll.

PPE: Parallel Poll Enable; IEEE-488 command that tells the device being configured which bit to set for its parallel poll response.

PPU: Parallel Poll Unconfigure; an unaddressed command that can be issued by the controller, telling all devices not to respond to parallel poll.

pts: pass through secondary; auxiliary command that causes the next command to set the UNC bit if the next command is a secondary command (a PPE, PPD, or MSA).

rlc: release control; IEEE-488 bus line auxiliary command.

REN: Remote Enable; command used to tell the systems receiving data that they can actually use the data and that their front panel controls are locked out.

RFD: Ready for Data; line used for data and command handshaking, controlled by the listeners.

rpp: request parallel poll; auxiliary command.

rqc: request control; used for transfer of control.

SCAS: System Control Active State; state in the system control state diagram, indicating the interface is the system controller.

SCG: Secondary Command Group; commands from 60 to 7E. These commands follow an earlier command that gives them meaning.

SDC: Selected Device Clear; command that can clear a device (like DCL) but works only on devices addressed as Listeners.

shdw: shadow handshake; command that will handshake each data byte immediately, without waiting for it to be read. Used with "tcs."

SIAS: System Control Interface Clear Active State; implies the system controller is sending Interface Clear.

sic: set interface clear; auxiliary command used to send out an Interface Clear.

SPAS: Serial Poll Active State; state on talker state diagram.

SPD: Serial Poll Disable; IEEE-488 command that tells all talkers to leave SPMS.

SPE: Serial Poll Enable; IEEE-488 command that will put a device into SPMS.

SPMS: Serial Poll Mode State; state on the talker state diagram.

SRQ: Service Request; IEEE-488 bus line that can be asserted by any device requiring service.

SRQS: Service Request State; state on service request state diagram

sre: send remote enable; auxiliary command that turns on the REN line.

STRS: Source Transfer State;state on the Source Handshake state diagram where the data byte is transferred.

TACS: Talker Active State; data can be sent by a device in this state.

TADS: Talker Addressed State; state in which a device is waiting to talk.

tca: take control asynchronously; an auxiliary command used to take control after a data transfer is complete.

tcs: take control synchronously; another auxiliary command, used in the interruption of a string of data.

TCT: Take Control; command used to tell a new controller to take control.

TIDS: Talker Idle State; state in talker state diagram

Talking: Transmitting data

ton: talk only; auxiliary command used to make a device talk when an MTA command is not possible.

unc: unrecognized command; a bit in interrupt register 1 that tells the microprocessor that there is a command it needs to read in the Command Pass Through Register.

UNL: Unlisten; IEEE-488 bus command that tells all devices on the bus to stop listening.

UNT: Untalk; IEEE-488 bus command that tells all devices on the bus to stop talking.

Index