

User Manual

Tektronix

SP232
Serial Extended Function Module
(for 1502B/C, 1503B/C MTDRs)

071-0467-00

Copyright © Tektronix, Inc. 1998. All rights reserved.

Tektronix products are covered by U.S. and foreign patents, issued and pending. Information in this publication supercedes that in all previously published material. Specifications and price change privileges reserved.

Printed in the U.S.A.

Tektronix, Inc., P.O. Box 1000, Wilsonville, OR 97070-1000

TEKTRONIX, TEK, and SP232 are registered trademarks of Tektronix, Inc.

WARRANTY

Tektronix warrants that this product will be free from defects in materials and workmanship for a period of one (1) year from the date of shipment. If any such product proves defective during this warranty period, Tektronix, at its option, either will repair the defective product without charge for parts and labor, or will provide a replacement in exchange for the defective product.

In order to obtain service under this warranty, Customer must notify Tektronix of the defect before the expiration of the warranty period and make suitable arrangements for the performance of service. Customer shall be responsible for packaging and shipping the defective product to the service center designated by Tektronix, with shipping charges prepaid. Tektronix shall pay for the return of the product to Customer if the shipment is to a location within the country in which the Tektronix service center is located. Customer shall be responsible for paying all shipping charges, duties, taxes, and any other charges for products returned to any other locations.

This warranty shall not apply to any defect, failure, or damage caused by improper use or improper or inadequate maintenance and care. Tektronix shall not be obligated to furnish service under this warranty a) to repair damage resulting from attempts by personnel other than Tektronix representatives to install, repair, or service the product; b) to repair damage resulting from improper use or connection to incompatible equipment; or c) to service a product that has been modified or integrated with other products when the effect of such modification or integration increases the time or difficulty of servicing the product.

THIS WARRANTY IS GIVEN BY TEKTRONIX WITH RESPECT TO THIS PRODUCT IN LIEU OF ANY OTHER WARRANTIES, EXPRESSED OR IMPLIED. TEKTRONIX AND ITS VENDORS DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. TEKTRONIX' RESPONSIBILITY TO REPAIR OR REPLACE DEFECTIVE PRODUCTS IS THE SOLE AND EXCLUSIVE REMEDY PROVIDED TO THE CUSTOMER FOR BREACH OF THIS WARRANTY. TEKTRONIX AND ITS VENDORS WILL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IRRESPECTIVE OF WHETHER TEKTRONIX OR THE VENDOR HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Table of Contents

Safety Summary	iii
Operator Safety	iii
Service Safety	iv
Operating Instructions	
Overview	1-1
Applications	1-1
Serial & Parallel Communication	1-2
SP232 Description	1-2
SP232 Setup	1-4
SP232 Specifications	1-4
Options	1-5
SP232 Local Commands	1-5
Set Baud Rate	1-5
Set Response Mode	1-6
Reset MTDR Interface	1-6
Set Stop Bits	1-6
SP232 Serial Protocol	
Serial Hardware Protocol	2-1
Protocol Example	2-2
Serial Software Protocol	2-4
Basic Frame Descriptions	2-5
Fixed-Length Frame Arguments/Data Description	2-5
Variable-Length Frame Arguments/Data Description	2-5
Status Frames	2-6
150XB/C MTDR and SP232 Command Sets	
15XB/C MTDR's and SP232 Monitor Instructions	3-1
Instrument Setup	3-2
Acquisition Setup	3-3
Hardware Setup	3-3
Waveform	3-6
Cursor	3-6
Point 1	3-7
Diagnostic	3-7
Remote	3-8
Display	3-8
Acquisition	3-8
Get Byte	3-8
Delay	3-9

Table of Contents

150XB/CMTDR's and SP232 Remote Instructions	3-9
Queries	3-10
Commands	3-12
150XB/C MTDR and SP232 Instruction Sets	
Level 1 – Monitor	4-1
Level 2 – Remote	4-3
Miscellaneous Information	
150XB/C MTDR Software Version Via Computer	5-1
Gain/Offset Via Computer	5-2
EFMCMD Demonstration Program	5-5
Commands and Remote Operation	5-6
Sample Programs	5-6
EFMGO.BAS Program	5-6
EFMCMD Program	5-9
Parallel Protocol	
Overview	6-1
Setup	6-1
Parallel Interface Cable Pinouts	6-2
Hardware Protocol	6-3
Master/Slave Handshake	6-5
Accept Frame Sequence	6-6
Estimated Timing Intervals	6-6
Send Frame Sequence	6-7
Estimated Timing Intervals	6-7
Software Protocol	6-8
Basic Frame Description	6-9
FIO Demonstration Program	6-9
Commands and Remote Operation	6-11
Sample Program	6-11
C Version of Program	6-12
Pascal Version of Program	6-18
BASIC Version of Program	6-25



General Safety Summary

Operator Safety Summary

Power Source

The SP232 module is designed to operate from power supplied by the host instrument.

Grounding the Instrument

This product is grounded through the parent instrument. A protective ground connection is essential for safe operation.

Danger Arising From Loss of Ground

Upon loss of the protective-ground connection, all accessible conductive parts (including knobs and controls that appear to be insulating) can render an electric shock.

Use Proper Power Cord

Use only a power cord and connector specified for the instrument and in good condition. Refer cord and connector changes to qualified service personnel.

Use Proper Fuses

To avoid fire hazard, use only correct fuses with voltage and current ratings specified for the instrument.

Do Not Remove Covers or Panels

To avoid personal injury, do not remove the instrument's covers or panels nor operate the instrument without covers and panels in place.

Service Safety

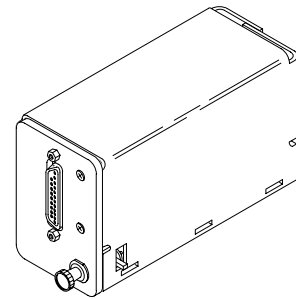
For Qualified Service Personnel Only

Disconnect the instrument from its power source before removing covers, and before soldering/desoldering components.



Operating Instructions

Operating Instructions



Overview

SP232 Extended Function Module

This manual describes how to use the Tektronix SP232 Serial Extended Function Module (EFM). The SP232 is a plug-in serial interface for Tektronix 1502B/C and 1503B/C Metallic Time Domain Reflectometers (MTDRs). *In this manual, all four MTDRs are collectively referred to as 150XB/C.*

The SP232 allows the 150XB/C to be controlled by an external host computer or other DTE (Data Terminal Equipment) device that uses RS232-C serial protocol. The SP232 acts as a DCE (Data Communications Equipment) device.

The SP232 module is powered by the 150XB/C and plugs into the Option Port on the front panel of the 150XB/C. This receptacle normally houses the Chart Recorder. The SP232 then connects to the serial port on the host computer. After the 150XB/C and computer are powered-up, use the codes described in this manual to write a program to control the 150XB/C through the computer.

NOTE. See *Options for ROM version and upgrade information.*

Applications

There are many applications where using the SP232 can result in productivity benefits. Some of these areas are: automated testing in a cable production facility, remote testing and measurement in a communication network environment, and automated calculation and data reduction.

In a cable production facility, cable testing can be completely automated. The PC can be programmed to set-up the 150XB/C, prompt the operator to attach a cable, invoke the measurement and transfer the resulting data to the PC. The PC then compares the data to a known standard and gives the operator a go/no-go decision.

In a communication network environment, the 150XB/C and SP232 can be connected remotely, via modem, to a controller. Periodic measurements are then made and transmitted to the controller. This way faults can be detected from a remote location.

Automation of soil moisture measurements can be done the same way as described in the cable testing example.

Serial and Parallel Communication

There are two protocol methods by which the 150XB/C can communicate with a host computer: Through RS232-C serial protocol by using the SP232 module; or through parallel protocol without the module. The actual information communicated, however, is the same.

The most common method is by using the SP232 module, and serial protocol which is compatible with most computer systems.

The less common parallel method requires a customer-supplied parallel interface cable that connects the 150XB/C to the host computer. And, the host computer must have a special parallel interface card installed. (See Chapter 6 for information about paralleled communication)

SP232 Description

***NOTE.** This manual assumes that you know how to make measurements using the 150XB/C MTDR. If necessary, refer to their Operator Manual for operator information. Also, you should have knowledge of personal computer operation and programming.*

The SP232 sends and receives data bytes with 1 start bit, 8 data bits and 1 stop bit. Transmit and receive baud rates are the same and may be set at 300, 600, 1200, 2400, 4800, 9600 or 19200 baud.

The factory-set baud rate is 1200, which is suitable for most computer systems. If necessary, the baud rate may be changed by using host computer software, which is the preferred method. To change the baud rate using software, see the Set Baud Rate instruction under SP232 Local Command Set.

The baud rate may also be changed by disassembling the SP232 and repositioning jumpers on its circuit board. This method is reserved for qualified service personnel because of the potential for mistakes and damage to instrument electronics.

SP232 signal levels conform to EIS RS232-C specs as follows:

AA (pin 1, protective or earth ground)

AB (pin 7, signal ground)

BA (pin 2, TXD – asynchronous data transmit, DTE to DCE)

BB (pin 3, RXD – asynchronous data received, DTE from DCE)

CA (pin 4, RTS – request to send signal originated by DTE)

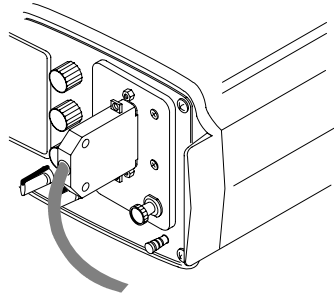
CB (pin 5, CTS – clear to send signal originated by DCE)

CC (pin 6, DSR – data set ready)

The SP232 serves as a data buffer that buffers and holds only the last computer frame and the last 150XB/C response.

The SP232 will operate under the 150XB/C battery power, but battery operating time will be reduced.

SP232 Set-Up



1. With the 150XB/C turned off, plug the SP232 into the Option Port on the front panel. Turn the locking latch clockwise to lock the SP232 in place.
2. Connect the SP232 to the serial port of the host computer using a standard RS232 cable. A female RS232 connector is located on the front panel of the SP232.
3. Power-up the 150XB/C and the host computer. Use the host computer as the controlling device as described in this and the remaining sections in this manual.

SP232 Specifications

Characteristic	Requirement	Supplemental Information
Front panel connector	RS232-C female 25 contact D type	SP232 is DCE
Data format	Asynchronous 8 bit ASCII with 1 start and 1 stop bit	Does not support XON/XOFF
Data rates	300, 600, 1200, 2400, 4800, 9600, 19200 baud	Selected by jumpers
Signal voltage levels	RS232-C compatible	Typically ± 10 V nominal
Battery operation of 150XB/C	SP232 works in battery operation	With reduced operating time per battery charge
EMI		Additional radiated emissions may exceed host instruments specifications
Environmental specs	Does not compromise host instrument	Will typically meet all host environmental specs except salt atmosphere/splash proof requirements

Options

In order for the SP232 to function properly, the host 150XB/C instrument must have ROM version 5.0 or higher. The ROM version is displayed when the 105XB/C is powered-up. The display will show [150XB/C ROM version XX], where XX is the ROM version.

If the host 150XB/C ROM version is below 5.0, order the proper ROM upgrade option from the table below. These ROM upgrades must be installed by qualified service personnel.

Instrument	Option
1502B – B021134 and below	Option 01
1503B – B022097 and below (without Ethernet, 06)	Option 02
1503B – B022097 and below (with Ethernet, 06)	Option 03

SP232 Local Commands

The following local commands configure the SP232 to accommodate host computer software.

In order to invoke the commands, the SP232 must be installed in the 150XB/C; connected via the RS232 cable; the host computer and 150XB/C must be powered-up. *The commands may be invoked whether the 150XB/C is making measurements or not.*

Local commands are designated with the Local frame-type (decimal 240), and are not relayed to the 150XB/C. They are executed by the SP232 only.

Set Baud Rate

<desc>:	Permits setting SP232 baud rate using host computer software
<cmd>:	<opcode><arg1>
<opcode>:	01h
<arg1>:	Divide actual baud rate by 100. Byte: 3 (300), 6 (600), 12 (1200), 24 (2400), 48 (4800), 96 (9600), 192 (19200)

Set Response Mode

<desc>:	<p>Lets user determine whether the SP232 will:</p> <ol style="list-style-type: none"> 1. wait for next request before sending 150XB/C responses to computer queries. 2. send responses immediately as received by the SP232. 3. wait until computer releases RS232-C Request To Send (RTS) line before sending responses.
<cmd>:	<opcode><arg1>
<opcode>:	03h
<arg1>:	<p>Single-Byte Argument</p> <p>00h – Wait for request (default). SP232 waits for ID byte (*) then sends directive before sending responses from 150XB/C to computer.</p> <p>01h – Respond immediately. SP232 sends responses as soon as received. This requires a serial interrupt driver or very fast polled program to catch the responses.</p> <p>02h – Respond with RTS line is released by computer. This permits the computer to signal for the data at the hardware level and avoids the extra frame handshake. The RTS line must be asserted again before the next ID byte (*) is sent.</p>

Reset 150XB/C Interface

<desc>:	Re-initializes the 150XB/C Option Port and sends a reset directive.
<cmd>:	<opcode>
<opcode>:	04h

Set Stop Bits

<desc>:	Sets number of stop bits per byte.
<cmd>:	<opcode><arg1>
<opcode>:	05h
<arg1>:	One byte containing 1 or 2 indicating one or two stop bits. Default is 1. Byte: 1, 2



SP232 Serial Protocol

SP232 Serial Protocol

Serial Hardware Protocol

***NOTE.** If you are writing an interrupt-driven or low-level driver for the SP232, you will need the information contained in the Serial Hardware Protocol subsection. If you are programming the SP232 from a high-level language, you probably will not need to know the details of hardware protocol.*

In most RS232 serial interfaces, hardware protocol is handled by low-level routines in the computer's operating system.

Serial hardware protocol is a subset of RS232-C which implements signals for Transmitted data (Tx), Received data (Rx), Request to Send (RTS), Clear to Send (CTS), Data Set Ready (DSR), and signal and chassis grounds.

The SP232 functions as a DCE device that asserts DSR and establishes communication with the 150XB/C upon power-up.

After asserting DSR, the SP232 waits for the host computer (or other DTE device) to assert the RTS signal line. Assertion means that the host computer wants to send data to the SP232.

In response to RTS asserted, the SP232 asserts the CTS signal line. The host computer never sends data to the SP232 until CTS is asserted, and stops sending as soon as CTS is no longer being asserted.

When the host computer senses the CTS signal, it begins sending data to the SP232. The SP232 looks for an asterisk character (*) as the ID byte at the beginning of each data frame from the host computer, and ignores all other characters until an asterisk has been received.

When the SP232 detects an asterisk character, it stops asserting CTS and sends a 1-byte directive back to the host computer.

The following example explains the possible values and meanings for the directive bytes.

Protocol Example

Serial software protocol is most easily understood by looking at an example exchange of data between the host computer and SP232. In this example, the SP232 is installed in the 150XB/C and the 150XB/C and host computer have just been powered-up.

Dialogue starts when you configure the computer's serial port to operate at 19.2 k-baud with 1 start bit, 1 stop bit, no parity check and no time-out. This starts low-level hardware handshake and leaves the SP232 ready for command or question from the computer.

Software handshake begins when you send an asterisk character through the computer to the SP232 and immediately read back a 1-byte directive from the SP232. The asterisk tells the SP232 that you want to start dialogue. The directive byte sent back defines the kind of dialogue.

There are three directive bytes that the SP232 may send back to the computer: reset directive (decimal 2), send-frame directive (decimal 6), or accept-frame directive (decimal 7).

Because the 150XB/C and the SP232 have just been powered-up, the first directive byte sent back will be the reset directive, which is ASCII character value 2.

Your program should reset itself when it receives this directive because the 150XB/C and SP232 have been reset (usually by power-up).

In this example, your program is just starting. There is no need to do anything except send another asterisk character which tells the SP232 that you are ready to continue.

In response, the SP232 will send another directive byte. The byte sent is the send-frame directive, which is ASCII character value 6. This byte tells your program that the SP232 is ready to accept either a command or query.

In this example the waveform query and response is described. The first byte sent to the SP232 is the query frame type which is ASCII character value 32. The second byte sent is the opcode for the waveform query which is ASCII character value 130.

In addition to opcode, the waveform query needs three additional argument bytes that tell the 150XB/C which waveform to send back, the data format and how many data points to send back.

The first argument byte in this example is ASCII character value 0. This indicates that you want the current acquisition waveform sent with 1 byte per data point.

The second argument byte is ASCII character value 1. This tells the 150XB/C to start with the first data point in the 251-point waveform.

The third argument byte is ASCII character value 10. This tells the 150XB/C to end the response with the tenth data point in the 251-point waveform.

At this point in our example, the SP232 has sent the send-frame directive. And it's time for your program to send the waveform query to the SP232.

This is done by sending the frame type, opcode and three arguments 1 byte at a time to the computer, or by sending all five bytes in an ASCII string.

When the SP232 recognizes the waveform query frame, it releases hardware line (CTS) until it has sent the query to the 150XB/C. This prohibits your program from sending another frame of data until the 150XB/C receives the request and has responded.

Because this is a hardware function, your program will see this as only a slight delay before it requests response to the waveform query.

The next step is to tell the SP232 that you want a response to your query. This is done by sending another asterisk character to the SP232 through your computer's serial port.

Once again, the SP232 will send back a directive byte. This time the byte will be ASCII character value 7, which is the accept-frame directive. This means that the SP232 has a frame of data to send to your program.

In this example, your program should immediately read 15 bytes from the computer. These bytes comprise a frame of data from the 150XB/C which consists of a response-frame byte (ASCII 48), the waveform query opcode (ASCII 130), two frame-length bytes (ASCII 130), two frame-length bytes (ASCII 20), ten data points whose ASCII values are the first ten points of the 150XB/C

waveform, and a single CRC (Cyclical Redundancy Check) byte whose ASCII value is derived from the data bytes, and can be used to make sure that the data was not corrupted in transmission.

Your program then sends an asterisk character to tell the SP232 that you want to communicate. The SP232 then sent back a directive byte which told your program the kind of communication wanted by the SP232.

The example is not easy to understand immediately because it illustrates the most complex dialogue in the command set. Sections 3 through 7 explain in detail how data frames are constructed and list all commands in detail.

Serial Software Protocol

Software protocol is based upon a master/slave relationship that exists between the 150XB/C and host computer. Because data acquisition routines in the 150XB/C cannot be interrupted, the protocol provides service to the computer on a polled basis.

The polling frequency is determined by how much acquisition and display time is taken by the commands accepted for execution through the SP232. There are two levels of remote operation: monitor and remote.

Monitor Operation: The first level is simply monitoring the 150XB/C control settings and requesting data periodically. When monitoring, the 150XB/C operates normally (always acquiring) except for the time spent responding to queries from the SP232.

Monitor operation is useful for manually setting acquisition parameters and capturing data for archive or processing by the SP232. Monitor operation has the poorest SP232 response characteristic because most time is spent acquiring and displaying data in normal operation.

Remote Operation: Remote operation consists of taking remote control of the 150XB/C. In remote operation, the 150XB/C's front panel is locked out (however, front panel controls may still be monitored). The 150XB/C display may be turned off for faster response and nearly all CPU time is spent accepting and executing commands through the SP232.

In remote operation, the 150XB/C acquires data only when commanded through the SP232.

Basic Frame Descriptions

Software protocol is based on the assumption that the host computer is capable of parsing a serial byte stream into status, command and data according to the following description.

All data sent through the SP232 is composed into logical groups of bytes called frames. A frame may be from two to several thousand bytes, depending on frame type and amount of data conveyed.

The frame length is always known to the sender before the frame is sent. The majority of commands, queries and responses have fixed-length frames. Those frames that contain varying amounts of data will include a frame length field and be terminated by a CRC field to verify the integrity of the data sent.

Byte 1: Frame type in high nibble:
1 = command (values O and F are illegal)
2 = query (values 5-E are reserved)
3 = response
4 = status

Low nibble contents are ignored

Byte 2: Message byte containing specific command, query or status code.

Fix-Length Frame Arguments/Data Description

Most commands, queries or responses that require arguments or data are fixed length. Bytes 3 through the known length of the frame consist of arguments or data bytes in the order and format specified by the command, query or response indicated in frame byte 2.

Variable-Length Frame Arguments/Data Description

The current command sets use only one variable-length frame which is the waveform query response from the 150XB/C. Future additions to the command set will include other variable-length frames.


Those commands, queries, or responses that require variable-length arguments or data will contain a two-byte unsigned integer length of data argument (low byte first) in frame bytes 3 & 4. The length of data argument will be followed by the number of data or argument bytes indicated. The last byte sent (not counted in the length of data argument sent in bytes 3 & 4) is a 1-byte CRC computed by doubling a 1-byte CRC accumulator and adding the next data byte to it with the carry from the doubling operation.

All bytes after the frame length (bytes 5 onward) except the CRC byte are used to compute the CRC checksum.

Status Frames

The status frame is fixed length (2 bytes). This allows the SP232 to notify the host computer of a bad frame.

If you receive a status frame, it means that the last frame sent from the computer to the SP232 was not understood. If this happens, you may want to repeat the last frame to reset the 150XB/C and repeat the last frame. If you receive a status frame while waiting for a response to a query sent, you will have to send the query again.



150XB/C and SP232 Command Sets

150XB/C and SP232 Command Sets

150XB/C and SP232 Monitor Instructions

The following instrument control instructions can be executed through the SP232 interface when installed in a 150XB/C instrument.

In order to invoke the instructions, the SP232 must be installed in the 150XB/C; the SP232 must be connected via RS232 cable to the host computer; the host computer and the 150XB/C must be powered-up. *The instructions may be invoked whether the 150XB/C is making measurements or not.*

Currently, there are two levels of operation possible:

1. Level 1 – Monitor: The first level monitors instrument front panel controls and configuration.
2. Level 2 – Remote: The second level monitors operational mode changes, and can change the operational mode and reprogram software controls while blocking out the front panel.

Each instruction is listed by a description, frame type, opcode and arguments. The arguments are one of the following types:

Byte:	Standard 8-bit value noted in hexadecimal
Boolean:	Byte types with one of two values: TRUE = On = 0ffh False = Off = 00h
UI1:	Unsigned 1-byte integer
SI1:	Signed 1-byte integer (2s complement)
UI2:	Unsigned 2-byte integer
SI2:	Signed 2-byte integer (2s complement)
UI4:	Unsigned 4-byte integer
SI4:	Signed 4-byte integer (2s complement)

For both levels, any references to vertical and horizontal scale are dependent upon whether the instrument is configured for decibels or millirho and meters or feet.

The following show the differences when the unsigned integer is the value passed to/from the SP232:

Vertical Scale: $UI1 \times 4 = \text{DB}$; $10^{(UI1/80)} = \text{millirho}$
 (UI1 is an unsigned 1-byte integer)

Horizontal Scale: **1502B/C** (Metric): Distance = $UI4 \times 0.001$ meters; (English)
 Distance = $UI4 \times 0.004$ feet
 (UI4 is an unsigned 4-byte integer)

Horizontal Scale: **1503B/C** (Metric): Distance = $UI4 \times 0.01$ meters; (English)
 Distance = $UI4 \times 0.04$ feet
 (UI4 is an unsigned 4-byte integer)

Monitor Instructions

The monitoring level of operation consists of monitoring instrument control settings and requesting data.

Instrument Setup

<desc>:	Query instrument for current setups: instrument ID (150XB/C), vertical scale in decibels or millirho, horizontal scale in feet or meters, light on or off, battery or AC power.
<query>:	<opcode>
<opcode>:	00h
<reply>:	1502B/C <opcode><arg1><arg2><arg3><arg4><arg5><arg6> 1503B/C <opcode><arg1><arg2><arg3><arg4><arg5>
<arg1>:	Instrument ID – Byte: 01 = 1502X, 02 = 1503X
<arg2>:	Vertical Scale – Byte: 01 = decibels, 02 = millirho. Dependent on the instrument setup, replies will use as vertical scale: $UI1 \times 4 = \text{DB}$; $10^{(UI1/80)} = \text{millirho}$
<arg3>:	Horizontal Scale Byte: 01 = feet, 02 = meters. Dependent on the instrument setup, replies will use as horizontal scale: 1502B/C Metric: Distance – $UI4 \times 0.001$ meters English: Distance = $UI4 \times 0.004$ feet 1503B/C Metric: Distance – $UI4 \times 0.01$ meters English: Distance = $UI4 \times 0.04$ feet
<arg4>:	Light Boolean: True = light on, False = light off.

Instrument Setup (continued)	
<arg5>:	Battery Byte: 00 = AC, 01 = battery; 02 = battery low
<arg6>:	1502B/C only Ohms at Cursor Boolean: True = ohms at cursor on, False = ohms at cursor off

Acquisition Setup

<desc>:	Query instrument for current acquisition setup: max hold, pulse and single sweep.
<query>:	<opcode>
<opcode>:	09h
<reply>:	<opcode><arg1><arg2><arg3>
<arg1>:	Max Hold Boolean: True = max hold on, False = max hold off
<arg2>:	Pulse Boolean: True = pulse disabled, False = pulse enabled
<arg3>:	Single Sweep Boolean: True = single sweep on, False = single sweep off

Hardware Setup

<desc>:	Query instrument for current hardware settings: This query will read the hardware latches but will not change the software settings.
<query>:	<opcode>
<opcode>:	01h
<reply>:	1502B/C <opcode><arg1><arg2>...<arg8>
<reply>:	1503B/C <opcode><arg1><arg2>...<arg10>
<arg1>:	Velocity of Propagation, Hundredths Digit Byte: 0 – 9
<arg2>:	Velocity of Propagation, Tenths Digit Byte: 3 – 9
<arg3>:	Distance Per Division Byte: 0 – 10 The following table shows distance-per-division representations based on instrument setup:

Dist/Div Argument		English (feet)		Metric (meters)	
1502X	1503X	1502X	1503X	1502X	1503X
00	00	0.1	1	0.025	0.25
01	01	0.2	2	0.05	0.5
02	02	0.5	5	0.1	1
03	03	1	10	0.25	2.5
04	04	2	20	0.5	5
05	05	5	50	1	10
06	06	10	100	2.5	25
07	07	20	200	5	50
08	08	50	500	10	100
09	09	100	1000	25	250
10	10	200	2000	50	500
	11		5000		1000

Hardware Setup (continued)

<arg4>:	<p>Button Status Byte: button status where a non-zero bit means button is pressed, zero means button is released: bit 0 – VIEW INPUT bit 1 – VIEW DIFF bit 2 – VIEW STORE bit 3 – STORE bits 4-7 – Cleared</p>
<arg5>:	<p>Horizontal Position SI1: Value returned represents the counts of change since last reading of cursor knob. Positive indicates clockwise rotation; negative indicates counterclockwise rotation. A value of zero indicates overflow or no change. Value 255 indicates underflow or negative 1.</p>
<arg6>:	<p>Vertical Scale SI1: Value returned represents counts of change since last reading of vertical scale knob. Positive indicates clockwise rotation, negative indicates counterclockwise rotation. A value of zero indicates overflow or no change. Value of 255 indicates under flow or negative 1.</p>

Hardware Setup (continued)

<arg7>:	<p>Noise Filter</p> <p>Byte: Noise filter (averaging) value where value returned represents:</p> <ul style="list-style-type: none"> 0 = Instrument in set ref mode, no averaging 1 = Instrument in set delta mode, no averaging 2 = Each waveform is reported or displayed with no averaging 3 = Two waveforms averaged before display or report 4 = Four waveforms averaged before display or report 5 = Eight waveforms averaged before display or report 6 = 16 waveforms averaged before display or report 7 = 32 waveforms averaged before display or report 8 = 64 waveforms averaged before display or report 9 = 128 waveforms averaged before display or report
<arg8>:	<p>Vertical Position</p> <p>SI1: Value returned represents the counts of change since last reading of vertical position knob. Positive indicates clockwise rotation; negative indicates counterclockwise rotation. A value of zero indicates overflow or no change. Value 255 indicates under flow or negative 1.</p> <p><i>Note: The values for horizontal position, vertical scale and vertical position change are read by the instrument when the front panel is enabled. Therefore, these values are not useful unless the front panel is disabled in remote control.</i></p>
<arg9>:	<p>1503B/C</p> <p>Pulsewidth</p> <p>1 Byte: The first byte is:</p> <ul style="list-style-type: none"> 0 – 2 nanosecond pulsewidth 1 – 10 nanosecond pulsewidth 2 – 100 nanosecond pulsewidth 3 – 1000 nanosecond pulsewidth 4 – auto pulsewidth
<arg10>:	<p>1503B/C</p> <p>Impedance</p> <p>Byte: the value sent means:</p> <ul style="list-style-type: none"> 0 – 50 Ohm 1 – 75 Ohm 2 – 93 Ohm 3 – 125 Ohm

Waveform

<desc>:	Request for waveform data. The binary waveform transmitted will be averaged based upon current value of noise filter. If averaging is not complete, the instrument will not send the waveform until averaging is complete. If an SP232 request is received while the instrument is waiting, it will override the previous query.
<query>:	<opcode><arg1><arg2><arg3>
<opcode>:	082h
<arg1>:	Data Type Byte: Flag requests the following information about the response: bit 0,1: 0 = Current waveform 1 = Stored waveform 2 = Difference waveform (8 bits only) 3 = Reserved bit 2: 0 = Screen data, 8 bits 1 = Acquired data, 13 bits bit 3-7: Cleared Binary waveforms are sent with no separators.
<arg2>:	Starting Data Point Location in Waveform UI1: Starting data point requested in waveform range is 1 – 251.
<arg3>:	Data points requested UI1: Range is N=1 - 251, where N ≤252 - starting data point. If N is too large, only those data points up to the 251st will be sent.
<reply>:	<opcode><length><arg1><arg2><arg3><...><arg251><crc>
<length>:	Number of data bytes transmitted 2 Bytes: 1 - 502. For averaged data, 1 byte per data point; for un-averaged data, 2 bytes per data point.
<arg1>:	First Data Point UI2 UI1: Size depends on type of waveform requested.
<argN>:	Nth data point UI2 UI1: Size depends on type of waveform requested.
<crc>:	CRC of variable-length data. UI1

Cursor

<desc>:	Query for distance to cursor.
<query>:	<opcode>

Cursor (continued)	
<opcode>:	03h
<reply>:	<opcode><arg1>
<arg1>:	Distance to Cursor UI4: Distance to the cursor is returned. Response is dependent on horizontal scale instrument setup.

Point 1

<desc>:	Query for distance to first point in the sweep (251 points).
<query>:	<opcode>
<opcode>:	04h
<reply>:	<opcode><arg1>
<arg1>:	Distance to First Data Sample UI4: Response is dependent on horizontal scale instrument setup.

Diagnostic

<desc>:	Query returns status of instrument diagnostics which is performed when the diagnostic query request is received.
<query>:	<opcode>
<opcode>:	05h
<reply>:	<opcode><arg1>
<arg1>:	Diagnostic Status Byte: Status is returned when a zero bit means self-test passed; a non-zero bit means self-test failed. bit 0: ROM0 self-test bit 1: ROM1 self-test (not currently implemented) bit 2: RAM self-test bit 3: NVRAM self-test (not currently implemented) bit 4: Display RAM self-test (not currently implemented) bits 5-7: Cleared

Remote

<desc>:	Query for remote status of instrument.
<query>:	<opcode>
<opcode>:	06h
<reply>:	<opcode><arg1>
<arg1>:	Remote State Boolean: TRUE – Remote on, FALSE – Remote off

Display

<desc>:	Query for front panel display status.
<query>:	<opcode>
<opcode>:	07h
<reply>:	<opcode><arg1>
<arg1>:	Display Status Boolean: TRUE = Display disabled, FALSE = Display enabled

Acquisition

<desc>:	Query for acquisition status.
<query>:	<opcode>
<opcode>:	0ah
<reply>:	<opcode><arg1>
<arg1>:	Acquisition Status Boolean: TRUE = Acquisition disabled, FALSE = Acquisition enabled

Get Byte

<desc>:	Query for contents of address.
<query>:	<opcode><arg1>
<opcode>:	08h
<arg1>:	Address to return byte contents UI2: 2-byte address, low byte first

Get Byte (continued)	
<reply>:	<opcode><arg1>
<arg1>:	Contents of Address UI1: byte

Delay

<desc>:	Query for delay value.
<query>:	<opcode>
<opcode>:	0bh
<reply>:	<opcode><arg1>
<arg1>:	Delay Value UI1: 1-byte value

150XB/C and SP232 Remote Instructions

The remote level of operation takes remote control of instrument by:

- Configuring instrument setup and front panel settings.
- Enabling/disabling access to front panel controls (though the front panel controls can still be monitored using the hardware setup query from the monitor mode command set).
- Enabling/disabling acquisitions.
- Enabling/disabling display refresh.

Any remote-level commands will automatically turn remote control on whether or not the Remote command has been sent. For most commands, remote control ON implies that the front panel and acquisitions are disabled. The exceptions are Sweep and Resume commands (see their descriptions for further detail). When receiving remote-level commands (if instrument is not currently under remote control), the current instrument setup, front panel settings and stored waveform data are stored. When remote control is turned off, this data will be stored.

Queries

Software Setup	
<desc>:	Query the instrument for current software (programmable) settings. These settings may or may not reflect the front panel controls.
<query>:	<opcode>
<opcode>:	020h
<reply>:	1502B/C <opcode><arg1><arg2>...<arg8> 1503B/C <opcode><arg1><arg2>...<arg10>
<arg1>:	Velocity of Propagation, Hundredths Digit Byte: 0 – 9
<arg2>:	Velocity of Propagation, Tenths Digit Byte: 3 – 9
<arg3>:	Distance per Division Byte: 0 – 10 The following table shows the distance-per-division representations based on instrument setup:

Dist/Div Argument		English (feet)		Metric (meters)	
1502X	1503X	1502X	1503X	1502X	1503X
00	00	0.1	1	0.025	0.25
01	01	0.2	2	0.05	0.5
02	02	0.5	5	0.1	1
03	03	1	10	0.25	2.5
04	04	2	20	0.5	5
05	05	5	50	1	10
06	06	10	100	2.5	25
07	07	20	200	5	50
08	08	50	500	10	100
09	09	100	1000	25	250
10	10	200	2000	50	500
	11		5000		1000

Software Setup (continued)	
<arg4>:	<p>Button Status</p> <p>Byte: Button status where a non-zero bit means mode is enabled and button box is on; zero means mode is disabled and button box is off:</p> <p>bit 0 – VIEW INPUT</p> <p>bit 1 – VIEW DIFF</p> <p>bit2 – VIEW STORE</p> <p>bit 3 – STORE</p> <p>bits 4-7 – Cleared</p>
<arg5>:	<p>Horizontal Position</p> <p>UI1: Position of cursor on the display. Range is 0 – 250.</p>
<arg6>:	<p>Vertical Scale</p> <p>UI1: Vertical scale which is value of instrument gain applied to the incoming signal. Value is dependent on instrument setup. This value is the absolute vertical scale and does not reflect any vertical scale reference offset.</p>
<arg7>:	<p>Noise Filter</p> <p>Byte: Noise filter (averaging) value where value returned represents:</p> <p>2 = Each waveform is reported or displayed with no averaging</p> <p>3 = Two waveforms averaged before display or report</p> <p>4 = Four waveforms averaged before display or report</p> <p>5 = Eight waveforms averaged before display or report</p> <p>6 = 16 waveforms averaged before display or report</p> <p>7 = 32 waveforms averaged before display or report</p> <p>8 = 64 waveforms averaged before display or report</p> <p>9 = 128 waveforms averaged before display or report</p>
<arg8>:	<p>Vertical Position</p> <p>UI2: Vertical position of trace ranges from 0 – 16383 with 8192 being approximately the center. The 150XB/C's initial vertical position after power-up is set so that the top of the outgoing step is at center screen.</p>

Software Setup (continued)

<arg9>:	<p>1503B/C Pulsewidth 1 Byte: bits 0, 1: 0 – 2 nanosecond pulsewidth 1 – 10 nanosecond pulsewidth 2 – 100 nanosecond pulsewidth 3 – 1000 nanosecond pulsewidth bit 2: 0 – auto pulsewidth mode not selected 1 – auto pulsewidth mode selected</p>
<arg10>:	<p>1503B/C Impedance Byte: the value sent means: 0 – 50 Ohms 1 – 75 Ohms 2 – 93 Ohms 3 – 125 Ohms</p>

Commands

Remote

<desc>:	<p>Remote command enables or disables remote control of instrument. When Remote is OFF (false) the software settings and display are always consistent with the front panel settings. Acquisitions will be enabled. The instrument is back under manual control. When Remote is ON (true) any manual change to the front panel settings will not change the software control settings or affect the display unless the Resume command has been received. Acquisitions are disabled until either a Sweep, Remote Off or Resume command is received.</p>
<cmd>: 10	1502B/C <opcode>
<cmd>:	1503B/C <opcode><arg1>
<opcode>:	021h
<arg1>:	Remote On or Off Boolean: TRUE = Remote On, FALSE = Remote Off

Resume

<desc>:	Permits manual operation of instrument. Commands: vertical scale, vertical position, horizontal position, cursor, buttons and instrument and acquisition setups programmed from remote control will still apply to the measurements until changed manually. However, noise filter, dist/div, and Vp controls will be set immediately from the front panel hardware. Display refresh and acquisitions are enabled. Remote is turned off. However, the instrument setup, front panel settings and stored waveform data saved when remote was turned on are not restored. These settings are lost. This permits the SP232 to configure the instrument and then turn it over for manual control.
<cmd>:	<opcode>
<opcode>:	1502B/C 022h 1503B/C 023h

Sweep

<desc>:	Enables and starts acquisition(s). A single acquisition will occur if instrument is in single sweep mode, otherwise acquisitions will be continuous.
<cmd>:	<opcode>
<opcode>:	023h

Display

<desc>:	Enables or disables front panel. If the display is disabled there is faster SP232 response and acquisitions can be taken at a faster rate.
<cmd>:	<opcode><arg1>
<opcode>:	024h
<arg1>:	Display On or Off Boolean: TRUE = Display disabled, FALSE = Display enabled

Instrument Setup

<desc>:	Performs instrument setup for vertical scale, horizontal scale, light.
<cmd>:	1502B/C <opcode><arg1>...<arg4>
<cmd>:	1503B/C <opcode><arg1>...<arg3>
<opcode>:	02bh
<arg1>:	Vertical Scale Byte: 01 = decibels, 02 = millirho
<arg2>:	Horizontal Scale Byte: 01 = feet, 02 = meters

Instrument Setup (continued)

Command Set

<arg3>:	Light Boolean: True = light on, False = light off
<arg4>:	1502B/C Ohms at Cursor Boolean: True = Ohms at Cursor on, False = Ohms at Cursor off

Acquisition Setup

<desc>:	Performs acquisition setup for max hold, pulse and single sweep.
<cmd>:	<opcode><arg1>...<arg3>
<opcode>:	1502B/C 02ch 1503B/C 02c
<arg1>:	Max Hold Boolean: True = max hold on, False = max hold off
<arg2>:	Pulse Boolean: True = pulse disabled, False = pulse enabled
<arg3>:	Single Sweep Boolean: True = single sweep on, False = single sweep off

Software Setup

<desc>:	Set the 150XB/C current software (programmable) settings.
<cmd>:	1502B/C <opcode><arg1><arg2>...<arg8> 1503B/C <opcode><arg1><arg2>...<arg10>
<opcode>:	025h
<arg1>:	Velocity of Propagation, Hundredths Digit Byte: 0 – 9
<arg2>:	Velocity of Propagation, Tenths Digit Byte: 3 – 9
<arg3>:	Distance per Division Byte: 0 – 10

The following table shows the distance-per-division representations based on instrument setup:

Dist/Div Argument		English (feet)		Metric (meters)	
1502X	1503X	1502X	1503X	1502X	1503X
00	00	0.1	1	0.025	0.25
01	01	0.2	2	0.05	0.5
02	02	0.5	5	0.1	1
03	03	1	10	0.25	2.5
04	04	2	20	0.5	5
05	05	5	50	1	10
06	06	10	100	2.5	25
07	07	20	200	5	50
08	08	50	500	10	100
09	09	100	1000	25	250
10	10	200	2000	50	500
	11		5000		1000

Software Setup (continued)

<arg4>:	<p>Button Status</p> <p>Byte: Button status where a non-zero bit means mode is enabled and button box is on; zero bit means mode disabled and button box is off:</p> <p>bit 0 – VIEW INPUT</p> <p>bit 1 – VIEW DIFF</p> <p>bit2 – VIEW STORE</p> <p>bit 3 – STORE</p> <p>bits 4-7 – Cleared</p>
<arg5>:	<p>Horizontal Position</p> <p>UI1: Position of cursor on the display, range is 0 – 250.</p>
<arg6>:	<p>Vertical Scale</p> <p>UI1: Vertical scale which is value of instrument gain applied to incoming signal. Value is dependent on instrument setup. This value is the absolute vertical scale and does not reflect any vertical scale reference offset.</p>

Software Setup (continued)

<arg7>:	<p>Noise Filter Byte: Noise filter (averaging) value where the value sent represents: 0 = No averaging, instrument in Set Ref mode 1 = No averaging, instrument in Set Delta mode, 2 = Each waveform is reported or displayed with no averaging 3 = Two waveforms averaged before display or report 4 = Four waveforms averaged before display or report 5 = Eight waveforms averaged before display or report 6 = 16 waveforms averaged before display or report 7 = 32 waveforms averaged before display or report 8 = 64 waveforms averaged before display or report 9 = 128 waveforms averaged before display or report</p>
<arg8>:	<p>Vertical Position UI2: Vertical position of the trace ranges from 0 – 16383 with 8192 being approximately the center.</p>
<arg9>:	<p>1503B/C Pulsewidth 1 Byte: 0 – 2 nanosecond pulsewidth 1 – 10 nanosecond pulsewidth 2 – 100 nanosecond pulsewidth 3 – 1000 nanosecond pulsewidth 4 – auto pulsewidth mode not selected</p>
<arg10>:	<p>1503B/C Impedance Byte: the value sent means: 0 – 50 Ohms 1 – 75 Ohms 2 – 93 Ohms 3 – 125 Ohms</p>

Cursor

<desc>:	<p>Command to set the distance to cursor. This command changes horizontal position of cursor. A software setup query is recommended following this command to update current data. A sweep must occur following this command to accurately reflect the cursor change.</p>
<cmd>:	<p><opcode><arg1></p>
<opcode>:	<p>027h</p>
<arg1>:	<p>Distance Units to Cursor UI1: Distance units to cursor is sent. Dependent on horizontal scale instrument setup.</p>

Put Byte

<desc>:	Puts a byte of information at location specified.
<cmd>:	<opcode><arg1>
<opcode>:	02ah
<arg1>:	Address to Place Byte UI2: Address, low byte first
<arg2>:	Data UI1: Data

Delay

<desc>:	Sets delay for transmit/receive between instrument and user interface; default in power-up is 255.
<cmd>:	<opcode><arg1>
<opcode>:	1502B/C 02dh 1503B/C 02bh
<arg1>:	Value of New Delay Resume IP UI1: 1 – 255



150XB/C and SP232 Instruction Set

150XB/C and SP232 Instruction Sets

Level 1 – Monitor

Instruction	opcode	frame type	#args
Instrument Setup	00h	2	0
Instrument Setup 1502X	00h	3	6
Instrument Setup 1503X	00h	3	5
Instrument Id:	1 = 1502X	2 = 1503X	
Vertical Scale:	1 = db	2 = mrho	
Horizontal Scale:	1 = feet	2 = meters	
Light:	T = on	F = off	
Battery:	0 = ac	1 = battery	2 = batt low
Ohms at Cursor: 1502X	T = on	F = off	
Acquisition Setup	09h	2	0
Acquisition Setup	09h	3	3
Max Hold:	T = on	F = off	
Pulse:	T = off	F = on	
Single Sweep:	T = on	F = off	
Hardware Setup	01h	2	0
Hardware Setup 1502X	01h	3	8
Hardware Setup 1503X	01h	3	10
Vp, hundredths:	[0-9]		
Vp, tenths:	[3-9]		
Dist/div:	[0-10]		
Button:	bit 0,1,2,3		
Horizontal position:	1 byte		
Vertical scale:	1 byte		
Filter:	[0-9]		
Vertical position:	1 byte		
Pulse Width: 1503X	[0-4]		
Impedance: 1503X	[0-3]		

Instruction Sets

Instruction	opcode	frame type	#args
Waveform	082h (130)	2	3
Data Type:	0 = current, screen (8 bit) 1 = stored, screen (8 bit) 2 = diff, screen (8 bit) 4 = current, acq (13 bit) 5 = stored, acq (13 bit)		
Starting data pt:	[1-251]		
Num data pts:	[1-251]		
Waveform	082h	3	[1-251]
length:	2 bytes		
data pt <n>	1 byte/pt, screen or 2 bytes/ pt, acquired		
crc:	1 byte		
Cursor	03h	2	0
Cursor	03h	3	1 (4 bytes)
Point 1	04h	2	0
Point 1	04h	3	1 (4 bytes)
Diagnostic	05h	2	0
Diagnostic	05h	3	1
Rom0 bit 0:	0 = pass	1 = fail	
Rom1 bit 1:	0 = pass	1 = fail	
Ram bit 2:	0 = pass	1 = fail	
NvRam bit 3:	0 = pass	1 = fail	
DRam bit 4:	0 = pass	1 = fail	
Remote	06h	2	0
Remote	06h	3	1
state	T = on	F = off	

Instruction	opcode	frame type	#args
Display	07h	2	0
Display	07h	3	1
state:	T = off	F = on	
Get Byte	08h	2	1 (2 bytes)
address:	2 bytes; low, high		
Get Byte	08h	3	1
value:	byte at address		
Acq	0ah	2	0
Acq	0ah	3	1
state:	T = off	F = on	
Delay	0bh	2	0
Delay	0bh	3	1
value:	[1-255]		

Level 2 – Remote

Instruction	opcode	frame type	#args
SW Setup Query	020h (32)	2	0
SW Setup Query 1502X	020h	3	8 (9 bytes)
SW Setup Query 1503X	020h	3	10 (11 bytes)
Vp, hundredths:	[0-9]		
Vp, tenths:	[3-9]		
Dist/div: 1502X	[0-10]		
Dist/div: 1503X	[0-11]		
Button:	bit 0,1,2,3		
cursor pos:	[0-250]		
Vertical scale:	1 byte, 1 count per .25 db		
Filter:	[0-9]		

Instruction Sets

Instruction	opcode	frame type	#args
Vertical pos:	[0-16383]		
Pulse width: 1503X	bit 0,1 pw, bit 2 auto pw		
Impedance: 1503X	[0-3]		
Remote Command	021h (33)	1	1
state:	T = on	F = off	
Resume Command	022h (34)	1	0
Sweep Command	023h (35)	1	0
Display Command	024h (36)	1	1
state:	T = off	F = on	
Instru Setup Cmd 1502X	02bh (43)	1	4
Instru Setup Cmd 1503X	02bh (43)	1	3
Vertical Scale:	1 = db	2 = mrho	
Horizontal Scale:	1 = feet	2 = meters	
Light:	T = on	F = off	
Ohms at Cursor 1502X	T = on	F = off	
Acquisition Setup Cmd	02ch (44)	1	3
Max Hold:	T = on	F = off	
Pulse:	T = pulse off	F = pulse on	
Single Sweep:	T = on	F = off	
SW Setup Cmd 1502X	025h (37)	1	8* (9 bytes)
SW Setup Cmd 1503X	025h (37)	1	10 (11 bytes)
Vp, hundredths:	[0-9]		
Vp, tenths:	[3-9]		
Dist/div	[0-10]		
Button:	bit 0,1,2,3		
cursor pos:	[0-250]		
Vertical scale:	1 byte, 1 count per .25 db		
Filter:	[0-9]		

Instruction	opcode	frame type	#args
Vertical pos:	[0-16383]		
Pulse width: 1503X	[0-4]		
Impedance: 1503X	[0-3]		
Cursor Command	027h (39)	1	1 (4 byte)
Put Byte Command	02ah (42)	1	2 (3 bytes)
address:	2 bytes		
value:	1 byte		
Delay Command 1502X	02dh (45)	1	1
Delay Command 1503X	02bh (45)	1	1
value:	[1-255]		



Miscellaneous Information

Miscellaneous Information

This section contains miscellaneous software information and sample programs for serial communications using the SP232 module.

150XB/C Software Version via Computer

To determine the software revision level of the 150XB/C when under control of the host computer, use the Get Byte instruction to directly access locations in the instrument's EPROM.

The 150XB/C uses a bank-switched EPROM. The highest location of both banks is 7FFFh. The ROM checksum and ROM part number are stored in the EPROM (both banks) in the highest 10 addressed bytes.

The top 10 bytes are:

7FFFh	High byte of checksum
7FFEh	Low byte of checksum
7FFDh	Complement of lowest byte of ROM part number
7FFCh	Highest byte of ROM part number
7FFBh	Middle byte of ROM part number
7FFAh	Lowest byte of ROM part number
7FF9h	High byte of bank switch address
7FF8h	Low byte of bank switch address
7FF7h	Bank ID byte: either 0 or 1
7FF6h	Instrument ID byte: either 01 or 02

The Tektronix part numbers for programmed EPROMS are 9 digit numbers. The first three numbers (prefix) indicates programmed ROM. The middle four numbers are the base part numbers, and the last two digits are the revision control numbers.

The byte address 7FFFCh contains the ROM version number in packed BCD format (i.e., 160-4411-03 will have 03h at address 7FFFCh). The last 2 digits of the ROM base part number are in packed

BCD format at 7FFBh, and the first two digits of the ROM base part number are at address 7FFAh. The byte at address 7FF6h will contain 01h for a 1502X and 02h for a 1503X.

Gain/Offset via Computer

This describes how the gain and offset parameters in the software setup command work.

Of the 13 bits acquired by the A/D converter, only the top 7 are displayed by the 150XB/C. The display is mapped so that 1 LCD pixel is equal to 64 A/D counts. This is a constant relationship that does not change with gain or offset.

Offset is set by a 14-bit D/A converter. The values in the software Query, Response, and Command are 14 bit unsigned integers. The top two bits are ignored in the Response and do not need to be set in the Command.

Offset is added to the sampled signal before the gain stages and before the A/D. At unity gain (0 dB), two counts of offset equal one A/D count. At higher gains, each count of offset has a greater effect at the A/D.

The gain stages amplify about mid-point on the A/D converter. A signal that is 15 pixels (one division) above the centerline on the LCD at unity gain will be 30 pixels (two divisions) above centerline at a gain of two (6 dB).

The offset D/A is not centered exactly with the A/D and gain stages. In order to position a waveform accurately at any gain, the true system zero offset point must be found.

Before describing how to determine the zero offset point, it helps to examine the relationship between counts of offset, voltage gain, A/D and LCD pixels. The following table illustrates these relationships:

Counts Offset	Volts and Gain (dB)	A/D Counts	Pixels Change
128	1 (0 dB)	64	1
64	2 (6 dB)	64	1
32	4 (12 dB)	64	1

Counts Offset	Volts and Gain (dB)	A/D Counts	Pixels Change
8	16 (24 dB)	64	1
1	128 (42 dB)	64	1
1	256 (48 dB)	128	2
1	512 (54 dB)	256	4
1	1029 (60.25 dB)	515	8
1	1540 (63.75 dB)	770	12

Gain has been rounded to the nearest quarter dB and pixels to the nearest integer value. At gains greater than 128 (42 dB), each count of offset will move more than one pixel on the display.

In normal operation, the amount of screen movement is held proportionate to the amount of control knob rotation with a scaling algorithm. Even so, 12-pixel jumps at very high gain is noticeable.

The following equality expresses the relationship between gain, offset and change in position on the LCD in pixels:

$$(\text{gain_volts})(\text{delta_offset}) / (\text{delta_pixels}) = 128$$

where:

$$\text{delta_pixels} = \text{pixels} - 64 \text{ (waveform query w/ 8-bit values)}$$

$$\text{delta_offset} = \text{offset} - \text{zero_offset}$$

$$\text{gain_volts} = \text{actual voltage gain (not in dB units)}$$

Note: Delta refers to differences from system zero points, not changes made by the operator or programmer.

When the goal is to keep the waveform at the same location on the LCD, the only requisite is to keep the gain delta_offset product constant. In order to do this, find the system zero_offset value.

A method for finding the system zero_offset value at power-up is:

1. Power-up instrument.
2. Set pulse to OFF with Acquisition Setup command.
3. Send Sweep command.
4. Read where trace is with Waveform query (8-bit values from current).

5. Compute a first offset correction. $\text{Correction} = (64 - \text{waveform value}) \times 128$.
6. Add offset correction to 8192 counts (power-up default for 150XB/C).
7. Set gain to 16 (96 quarter-dB counts) and offset to corrected value with the software Setup command.
8. Send Sweep command.
9. Read where trace is with Waveform query.
10. Compute new offset correction: $\text{Correction} = (64 - \text{waveform value}) \times 8$.
11. Add correct to offset value.
12. Set gain to 128 (168 quarter-dB counts) and offset to new value with software Setup command (use 16 averages also).
13. Send Sweep command.
14. Read where trace is with Waveform query.
15. Compute final offset correction: $\text{Correction} = 64 - \text{waveform value}$.
16. Add correction to last offset value to get zero_offset value.
17. Release instrument using Remote OFF command.

This procedure provides a zero_offset value that is useful even at very high gain settings.

The conversion from quarter-dB counts to voltage gain is:

$$\text{voltage_gain} = 10^{\wedge} (\text{quarter_dB_count}/80)$$

where: \wedge indicates exponentiation

The conversion from voltage gain to quarter-dB counts is:

$$\text{quarter_dB_counts} = 80 \log (\text{voltage_gain})$$

where: log is the common or base 10 logarithm

Finally, though the zero_offset value may vary slightly from instrument to instrument, it is constant for every instrument and can be computed only once when a 150XB/C computer system is first set up.

EFMCMD Demonstration Program

The EFMCMD demonstration program is an interactive way to test commands when in serial communication mode.

When you invoke the EFMCMD program, the first prompt asks if you want to enter a command, request service from the 150XB/C or quit the EFMCMD program.

Before requesting service, enter the codes for a command using the Edit function. Example:

- Run the EFMCMD program and type 'e' to enter a new command.

In this example you are asked for the first 10 data points from the current 150XB/C waveform in 8-bit format for each data point. Enter all numbers in decimal.

- Enter '32' to query the 150XB/C for data.
- Enter '130' for the opcode (waveform query).
- Enter '3' for command length.
- Enter '0' for the first command argument data byte.
- Enter '1' for the second command argument data byte.
- Enter '10' for the third command argument data byte.

You will now be returned to the Main Menu. Send the just-entered commands to the 150XB/C.

- Enter 'r' to request 150XB/C service.

The EFMCMD program will send frametype, opcode and data. Or print a message stating that a reset directive was received from the 150XB/C. *The 150XB/C always sends a reset directive the first time it responds to a service request after power-up.*

If the 150XB/C responded with a reset directive, request service again and the frame information should be sent.

The 150XB/C has now accepted a request for data. To request data:

- Enter 'r' to request service.

The EFMCMD program will show you frametype 6 accepted and the 15 bytes of data sent in response to your query.

The waveform response is a variable-length data frame containing the number of data bytes terminated by a 1-byte CRC:

- The first byte is 48 which is the response frametype byte.
- The second byte is 130 which is the waveform query opcode being responded to.
- The next two bytes are 10 and 0 which is the number of data bytes in a 16-bit integer. The least significant byte of the integer is sent first. *These bytes are not present in fixed-length data frames.*
- The next ten bytes are the actual waveform data points in 8-bit integers. These will change if you move the vertical position of the waveform and repeat the query.
- The last byte is the cyclical redundancy check (CRC) byte used to validate the data. *This byte is not present in fixed-length data frames.*

Commands and Remote Operation

When sending a command to the 150XB/C, use directive value 16 and you will not have to request service a second time.

Once you start sending commands to change 150XB/C settings, you will automatically put the 150XB/C into remote operating mode. This means that you must request a new data sweep to see the effects of your changes.

Sample Programs

The following programs, written in Basic, demonstrate the SP232 to a knowledgeable programmer.

EFMGO.BAS Program

This program demonstrates the waveform query command

```
,*****
```

```
GOSUB into ' Show user instructions  
GOSUB setup ' Setup variables & display
```

```

'=====
main: ' Stays in while loop <.> key is pressed to end program
'=====

WHILE x$ <> "."
CLOSE #2
OPEN "com1:1200,n,8,1" FOR RANDOM AS #2
PRINT #2, "***";
d$ = INPUT$(1, #2)
IF d$ = r$ THEN PRINT "Reset!!!"
IF d$ = a$ THEN GOSUB getwave
IF d$ = s$ THEN PRINT #2, q$;
x$ = INKEY$
IF x$ = "s" THEN GOSUB suspend
WEND

'=====
finish: ' end of program
'=====

SCREEN 0: CLS
CLOSE
END

'=====
intro:
'=====

CLS : PRINT : PRINT
PRINT "Short Basic Program to show SP232 serial interface at work."
PRINT : PRINT
PRINT "Press the period key <.> to end program."
PRINT : PRINT
PRINT "Press <s> to stop at current waveform & <g> to go again."
PRINT : PRINT
INPUT "Press <ENTER> to begin": x$
RETURN

'=====
setup: ' Initialize string constants & display window
'=====

' q$ contains the waveform query command
' r$ contains the reset directive string
' s$ contains the send frame directive string
' a$ contains the accept frame directive string

```

Miscellaneous Information

```
'q$ = CHR$(32) + CHR$(130) + CHR$(0) + CHR$(1) + CHR$(251)
r$ = CHR$(2)
s$ = CHR$(6)
a$ = CHR$(7)
SCREEN 2: WINDOW (0, 0)-(251, 255); CLS
RETURN
```

```
'=====
getwave: ' get waveform from 150XB/C
'=====
```

```
' This routine draws waveform as it gets each point from 150XB/C
' It would be just as easy to put the data into an array...
```

```
c$ = INPUT$(4, #2) ' gets 4 bytes before first data point in frame
GOSUB graticule
PSET (0, 2 * ASC(INPUT$(1, #2))) ' draw first point
FOR I = 2 TO 251
LINE -(1, 2 * ASC(INPUT$(1, #2))) ' draw to next point.
NEXT I
```

```
' This routine does not check CRC to make sure data is all valid...
```

```
c$ = INPUT$(1, #2) ' this gets the CRC byte at the end of the frame
RETURN
```

```
'=====
suspend: ' wait until g or period key is pressed
'=====
```

```
BEEP
WHILE (x$ <> "g") AND (x$ <> ".")
x$ = INKEY$
WEND
BEEP
RETURN
.BP
```

```
'=====
graticule: 'clear screen & draw graticule
'=====
```

```
CLS
LINE (0, 0)-(251, 255), 1, B ' draw border

FOR x = 0 TO 251 STEP 251 / 10 ' draw grid marks
FOR y = 0 TO 255 STEP 255 / 40
```

```

PSET (x, y)
NEXT y
NEXT x

FOR x = 0 TO 251 STEP 251 / 50
FOR y = 0 TO 255 STEP 255 / 8
PSET (x, y)
NEXT y
NEXT x

RETURN

```

```

,*****

```

EFMCMD Program

This program interactively teaches and tests 150XB/C Serial Protocol & Commands.

```

'=====
setup:
'=====

sb$ = "1"           ' default to one stopbits
br$ = "1200"       ' default to 1200 baud
sp$ = "com1:" + br$ + " ,n,8," + sb$

r$ = CHR$(2): s$ = CHR$(6): a$ = CHR$(7)
q$ = CHR$(32) + CHR$(0) 'Instrument Setup Query for default
CLS
PRINT : PRINT
PRINT "This is a short interactive Basic program to let the user"
PRINT "experiment with the 150XB/C computer command set."
PRINT
PRINT "All numbers must be entered in decimal/letters in lower"
PRINT "case."
PRINT
PRINT "See the software protocol description and command set sections"
PRINT "of the SP232 manual for explanations of data formats &"
PRINT "specific command syntax."
PRINT

'=====
main: 'Main Loop of Program
'=====

```

```
GOSUB prompt
WHILE o$ <> "q"
CLS
IF o$ = "e" THEN GOSUB incommand
IF o$ = "r" THEN GOSUB request
IF o$ = "s" THEN GOSUB setserial
IF o$ = "?" THEN PRINT : PRINT "Using..."; sp$
GOSUB prompt
WEND
CLOSE
CLS
END

'=====
prompt: "Prompt user for next choice
'=====

PRINT
PRINT "Enter your choice (one letter) & press <enter> key:"
PRINT
INPUT "e-nter command, r-quest service, s-et serial port, q-uit program"; o$
RETURN

'=====
setserial: 'demonstrates changing SP232 Baud rates with Local command
'=====

' t$ is used to temporarily store command string while baud command in q$

CLS
PRINT : PRINT "Current SetUp is: "; sp$: PRINT
INPUT "Enter baudrate: "; br$
INPUT "Enter stopbits: "; sb$

t$ = q$
q$ = CHR(240) + CHR$(1) + CHR$(VAL(br$) / 100)
ld$ = a$
WHILE ld$ <> s$
GOSUB request
WEND
q$ = t$

sp$ = "com1:" + br$ + ",n,8," + sb$
PRINT : PRINT "New SetUp is: "; sp$
PRINT
RETURN
```



```

=====
incommand: 'get command to send
=====

PRINT : PRINT "Frametypes are: command = 16, query = 32, local = 240"
PRINT : INPUT "Enter frametype"; c: q$ = CHR$(c)
PRINT : PRINT "Command OpCodes are summarized in SP232 the manual."
PRINT : INPUT "Command OpCode"; c: q$ = q$ + CHR$(c)
PRINT : PRINT "Command length is # of bytes needed for command arguments."
PRINT : INPUT "Command Length"; cl
FOR i = 1 TO cl
PRINT "command argument byte #"; i;
INPUT c
q$ = q$ + CHR$(c)
NEXT i
RETURN

=====
request: 'Request Service from 150XB/C
=====

CLOSE #2
OPEN sp$ FOR RANDOM AS #2

PRINT #2, "***";
d$ = INPUT$(1, #2)
PRINT "Directive from 150XB/C is: "; ASC(d$)
ld = d$

IF d$ = r$ THEN GOSUB init
IF d$ = s$ THEN GOSUB send
IF d$ = a$ THEN GOSUB accept
IF d$ <>"ok" THEN PRINT "Illegal Directive Received!!!"
RETURN

=====
send: 'Send Command to 150XB/C
=====

PRINT #2, q$;
PRINT
PRINT "Command Sent...";
FOR i = 1 TO LEN(q$): PRINT "["; ASC(MID$(q$, i, 1)); "]" : NEXT i
PRINT
d$ = "ok"
RETURN

```

Miscellaneous Information

```
'=====
accept: 'accept whatever data the 150XB/C has sent back
'=====

PRINT
INPUT "<ENTER> for response"; d$
PRINT
d$ = INPUT$(LOC(2), #2)
PRINT "Frametype = "; ASC(MID$(d$, 1, 1))
PRINT "OpCode = "; ASC(MID$(d$, 2, 1))
FOR i = 3 TO LEN(d$)
  PRINT "Byte"; i - 2;" = "; ASC(MID$(d$, i,1))
NEXT i
d$ = "ok"
PRINT
RETURN

'=====
init: 'Reset your program code would go here...
'=====

PRINT
PRINT "Received Reset Directive!!!"
d$ = "ok"
BEEP
RETURN

'=====
```



Parallel Protocol

Parallel Protocol

Overview

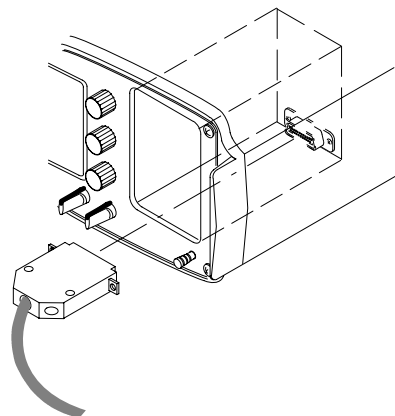
This chapter describes hardware and software protocol information necessary for parallel communication between the 150XB/C and host computer without using an SP232 module. All commands and instructions remain the same as when using an SP232 module.

Note: In order to use the 150XB/C in parallel communication mode, you must have a parallel interface cable that attaches between the 150XB/C Option Port connector and the host computer.

This cable is not furnished. You have to supply your own. Please refer to the next page for a list of pinouts necessary to manufacture the interface cable.

Also, the host computer must have a parallel I/O card (Anasco 41-404 for example) installed before parallel communication is possible.

Setup - Parallel Interface Cable Setup



1. With the 150XB/C turned off, plug the parallel interface cable into the connector at the back of the Option Port. There is no module installed in the Option Port.

2. Connect the interface cable to the host computer.
3. Power-up the 150XB/C. Power-up the host computer. Use the host computer as the controlling device. Protocol, commands, instructions, etc. are the same as described for SP232 serial communication mode except where noted in this chapter.

Parallel Interface Cable Pinouts

These pinouts are used for manufacturing a parallel interface cable that connects between the 150XB/C Option Port connector and a host computer that contains a parallel I/O card.

150XB/C		COMPUTER	
Signal	Pin	Pin	Signal
data0	2	37	data0
data1	1	36	data1
data2	25	35	data2
data3	24	34	data3
data4	23	33	data4
data5	22	32	data5
data6	21	31	data6
data7	20	30	data7
addr0	19	nc	
addr1	18	nc	
addr2	17	nc	
addr3	16	nc	
/RD	4	23	PC6 or /ACKA
/WR	3	25	PC4 or /STBA
/EN	5	nc	
/IA	6	nc	

Signal	Pin	Pin	Signal
/IR	7	29	PC0
+5 volts	9	nc	
+5 return	8	21	common
+16 volts	12 & 13	nc	
+16 return	10 & 11	nc	
Analog return	15	nc	
/TRIG	14	nc	

Hardware Protocol

The available hardware signals are:

Input/Output (bi-directional):		Connector Pin #
D0	least significant data bit	3
D1	data bit	1
D2	data bit	24
D3	data bit	22
D4	data bit	20
D5	data bit	18
D6	data bit	16
D7	most significant data bit	14

(Data lines are driven by a 74HC245 and are pulled down by 10K at the Option Port interface)

Output only (generated by 150XB/C):

A0	least significant address bit	12
A1	address bit	10
A2	address bit	8

A3	address bit	6
/RD	active low read signal	7
/WR	active low write signal	5
/CS	active low chip signal	9
(Above output lines are driven by a 74HC244 at the Option Port)		
/IA	acknowledge for /IR signal (below)	11

Input only (generated by Option Port device):

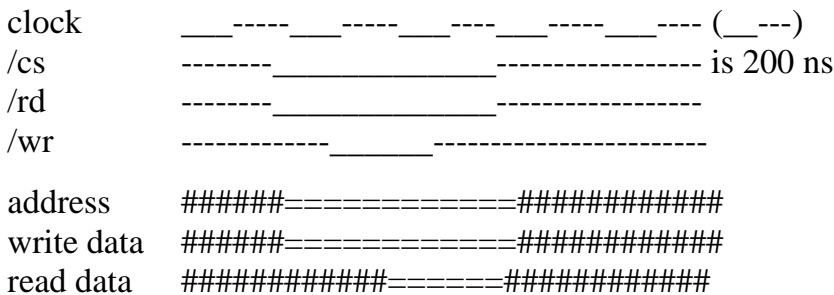
/IR	active low service request line	13
(/IR line is pulled up by a 10K to +5 volts)		

Switch Power Supplies:

+16 switched +16 volts supply	23 & 25
ret16 + 16 volt return lines	19 & 21
+5 switched +5 volts supply (100 ma)	17
ret5 + 5 volt return line	15

Tektronix Proprietary DO NOT USE 2, 4, 26

The general timing relationships of the control signals is diagrammed with reference to the 5 MHz instrument clock below (although the clock is not ported).



Where ___ is a low logic level and --- is a high logic level.
Where ### means don't care and === means either high or low but steady state.

The diagrams show that data written to the Option Port by the instrument will be valid throughout the /CS period and that data to be read by the instrument from the Option Port must be valid for the last 200 nanoseconds of the /CS period.

Also implicit in the hardware configuration is that the Option Port device is a passive piece of the instruments memory map with a single active low service request line (/IR). This means that the Option Port device is a "slave" to the instrument processor which can present requests to the instrument but not assert any control over how and when those requests are actually fulfilled.

The correct way for the Option Port device to state its presence or request service is through the /IR signal line. The +5 volt supply to the Option Port will be turned on when /IR is recognized by the instrument. The instrument will then wait about 2 milliseconds for any device hardware using the +5 or +16 volt supplies to stabilize before attempting to read the device identification byte. Once the Option Port device has completed its power on sequence, it should latch the /IR line low until it receives an /IA signal (/IA is not latched in the instrument).

Master/Slave Handshake

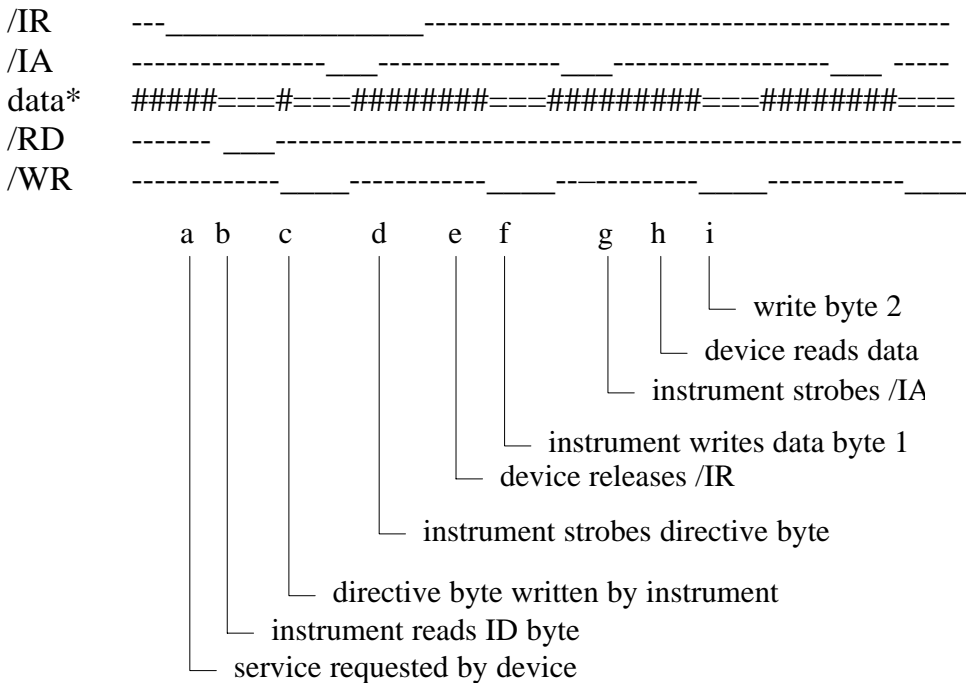
When the instrument sees the /IR signal low, it will read an identification byte from address 0 and then write a directive byte to address 0 of the Option Port device. The identification numbers are arbitrary and assigned by Tektronix. The directive byte indicates what action the device must perform. The possible actions are:

accept new frame	byte = 7
send new frame	byte = 6
reset	byte = 2
All other directive values are reserved.	

Once the directive byte has been written to the Option Port, the Option Port device has 10 milliseconds to respond to the directive given by the instrument.

If the directive is to accept a frame of information the device must release the /IR line and prepare to accept successive bytes on successive /IA strobes. Pulling the /IR line low during a frame transfer to the device from the instrument will cause an error and abort the transmission. The instrument will write an accept last frame directive at the next service opportunity.

Accept Frame Sequence



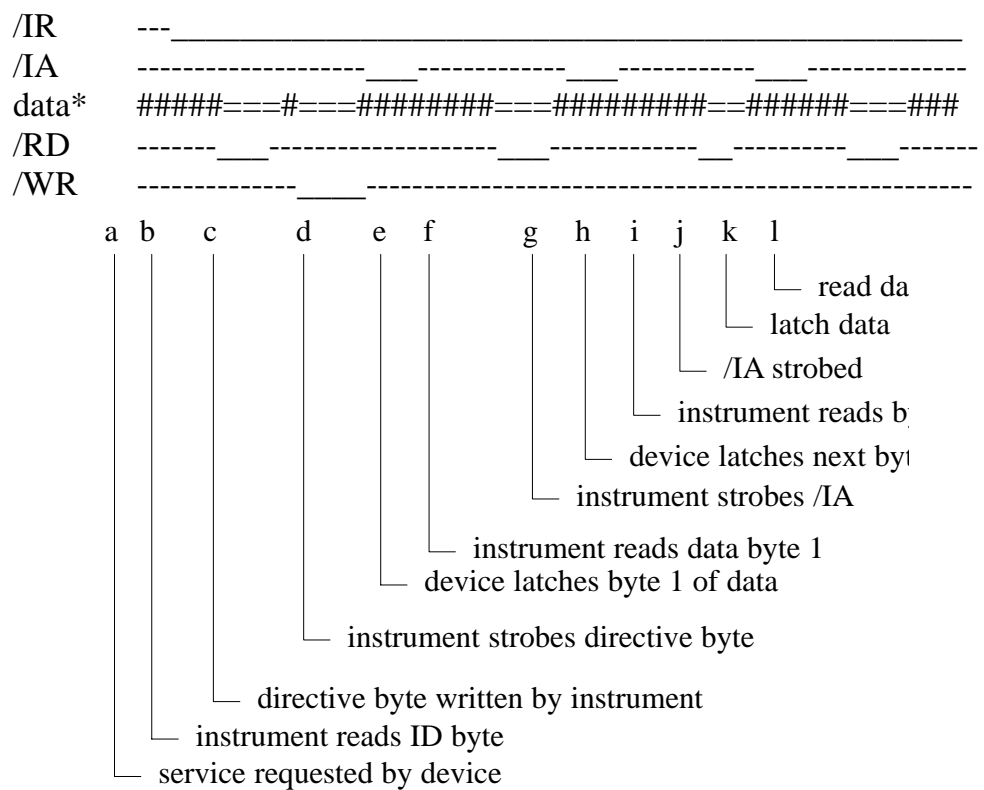
* data at the Option Port interface

Estimated Timing Intervals

a-b	indefinite
b-c	1 < x < 50 microseconds
c-d	1 < x < 50 microseconds
d-e	x < 10 milliseconds
e-f	1 < x < 500 microseconds
f-g	1 < x < 50 microseconds
g-h	1 < x < 50 microseconds
h-i	100 < x < 250 microseconds
f-i	100 < x < 300 microseconds (typically 200)

If the directive is to send a frame of information, the device must hold the /IR line low and place the first byte of the frame in its output buffer before the 10 millisecond time limit. After the instrument reads each byte, it will strobe /IA to indicate to the device that it is time to place the next byte of data into its output buffer. The device should release /IR upon receiving the /IA strobe after the last byte in the frame.

Send Frame Sequence



* data at the Option Port interface

Estimated Timing Intervals

- a-b indefinite
- b-c $1 < x < 50$ microseconds
- c-d $1 < x < 50$ microseconds
- d-e $x < 10$ milliseconds
- e-f $10 < x < 11$ milliseconds
- f-g $1 < x < 50$ microseconds
- g-h $1 < x < 100$ microseconds
- h-i $150 < x < 200$ microsecond

i-j	$1 < x < 50$ microseconds
j-k	$1 < x < 100$ microseconds
j-l	$150 < x < 200$ microseconds
i-l	$150 < x < 200$ microseconds
h-k	$150 < x < 200$ microseconds

If the directive is to wait until signaled, the device should release the /IR line at once and not assert it again until it sees another/IA strobe. When the strobe is seen, the device may request attention by asserting the /IR signal again.

If the directive is to reset, the device should release the /IR line at once and not assert it again until it has cleared all errors, partial frames and at least 20 milliseconds has passed for the instrument to also get into a reset state.

Software Protocol

The software protocol is based on a master/slave relationship that must exist between the instrument and the Option Port device. Since the actual data acquisition routines in the instrument cannot be interrupted, the instrument will provide service to the Option Port on a polled basis. The polling frequency will be determined by how much acquisition and display time is taken by the commands accepted for execution through the Option Port in most circumstances. There are a number of levels of remote operation of the instrument possible through the Option Port.

The first level is simply monitoring the instrument control settings and requesting data from time to time. In this mode the instrument operates normally (always acquiring) except for the small pieces of time required to respond to queries from the Option Port device. This mode is useful for manually setting the acquisition parameters and capturing the data for archive or processing by the Option Port device. This mode has the poorest Option Port response characteristic because most of the time is spent acquiring and displaying data in normal operation.

The second level of operation is taking complete remote control of the instrument. In this mode the instrument front panel is locked out (the front panel controls may still be monitored). The instrument

display may be turned off for faster Option Port response and nearly all of the instrument's CPU time is spent accepting and executing commands through the Option Port. At this level the instrument will acquire data only when commanded to through the Option Port.

The software protocol is based on the assumption that all devices that connect to the instrument's Option Port contain a microprocessor capable of parsing a serial byte stream into status, command and data according to the following description.

All data sent through the Option Port is composed into logical groups of bytes called frames. A frame may be from two to several thousand bytes long depending on the frame type and amount of data to be conveyed. The frame length will always be known to the sender before the frame is sent. The majority of commands, queries and responses have fixed length frames. Those frames that may contain varying amounts of data will include a frame length field and be terminated by a CRC field to verify the integrity of the data sent.

Basic Frame Description

Byte 1 Frame type in high nibble:

- 1 = command (values 0 & F are illegal)
- 2 = query (values 6-E are reserved)
- 3 = response
- 4 = status
- 5 = test

(low nibble ignored)

Byte 2 Message byte containing specific command, query, status code, test code.

FIO Demonstration Program

The FIO demonstration program is an interactive way to test commands when in parallel communication mode. The program is useful for testing individual commands and for learning how data frames used in the protocol are constructed.

When you invoke the FIO program, the first prompt asks if you want to edit a command, request service from the 150XB/C or quit the FIO program.

Before requesting service, enter the codes for a command using the Edit function. Example:

- Run the FIO program and type 'e' to edit a new command.

In this example you are asked for the first 10 data points from the current 150XB/C waveform in 8-bit format for each data point. Enter all numbers in decimal.

- Enter '130' for the opcode (waveform query).
- Enter 'f' for fixed-length command.
- Enter '10' for response length.
- Enter '3' for command length.
- Enter '0' for the first command argument data byte.
- Enter '1' for the second command argument data byte.
- Enter '10' for the third command argument data byte.

You are now returned to the Main Menu. Send the just-entered commands to the 150XB/C:

- Enter 'r' to request 150XB/C service.
- Enter 'q' to query because you are asking the 150XB/C for data.

The FIO program will send frametype, opcode and data. Or print a message stating that a reset directive was received from the 150XB/C. *The 150XB/C always sends a reset directive the first time it responds to a service request after power-up.*

If the 150XB/C responded with a rest directive, request service again and the frame information should be sent.

The 150XB/C has now accepted request for data. To request the data:

- Enter 'r' to request service.
- Enter 'r' for a response from the 150XB/C.

The FIO program will show you frametype accepted and the 10 bytes of data sent in response to your query.

Commands and Remote Operation

When sending a command to the 150XB/C, use the 'c' option in the Request Service menu and you will not have to request service a second time.

Once you start sending commands to change 150XB/C settings, you will automatically put the 150XB/C into remote operating mode. This means that you must request a new data sweep to see the effects of your changes.

Sample Program

The following demonstration program, written in C, Pascal and Basic, provides a pattern for developing software for 150XB/C parallel protocol.

It may also help programmers to learn parts of parallel protocol that are common to serial protocol (i.e., everything except the routines that put a byte and read a byte are common to both protocols).

The parallel FIO program allows you to interact with the 150XB/C through Option Port protocol.

Its main loop is a menu-driven switch (or case) statement that allows you to either request service from the 150XB/C, define parameters for a command to send to the 150XB/C or quit the program. The typical sequence is: define a command, request service to send the command, then request service again to receive the result of the command.

The request-service routine first asks for either a command, query, response, status or test frame-type.

Next, the routine writes its ID number to the 8255 interface chip (which latches it for the 150XB/C to read later). It also asserts an active low service-request line to the 150XB/C.

The 150XB/C acknowledges the request by reading the ID number and writing a directive byte back to the computer. The program then branches to the correct routine to handle that directive.

The reset directive indicates that the 150XB/C has reset its internal variables (such as at power-on). This means that pending requests for

data are lost and the computer should reset its state to match the 150XB/C.

If the 150XB/C is holding the result of a prior request, it will send an accept-frame directive. Otherwise it will send a send-frame directive which allows the computer to send the user-defined command frame.

The accept-frame and send-frame routines follow the protocol descriptions exactly and should be easy to trace.

C Version of Program

```
#define MAXBYTES 550
#define PORT 0x300
#define DEVICE 0x81

#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <conio.h>

/* a few global variables */
/* output - is data from computer to 150XB/C */
/* input - is data from 150XB/C to the computer */

int in[MAXBYTES]; /* input data array */
int out[MAXBYTES]; /* output data array */
int type; /* type of service requested */
int directive; /* frame directive */
int command; /* frame op code or command byte */
int ftype; /* frame type byte */
int leno; /* output frame length category */
int leni; /* input frame length category */
int crc; /* crc for variable length frames */
int data; /* general purpose data byte variable */
int item; /* user menu choice */
int ch; /* general purpose character */
int olen; /* output frame length */
int ilen; /* input frame length */
int index; /* loop index */

putbyte(byte)
int byte;
{
```



```
    outp(PORT, byte);
    byte = inp(PORT + 2);
    while((byte & 128) == 0) byte = inp(PORT + 2);
}

getbyte()
{
    int byte;

    byte = inp(PORT);
    byte = inp(PORT + 2);
    while((byte & 32) == 0) byte = inp(PORT + 2);
    byte = inp(PORT);
    return(byte);
}

resetit()
{
    outp(PORT+3, 1);
    system("cls");
    printf("\n\nReceived Reset Directive!!!\n");

    printf("\nPress space bar to return to Main Menu ");
    getch();
}

menuchoice()
{
    system("cls");
    printf("\nProgram to test 150XB Option Port Protocols\n");
    printf("\n\nS. Simpson   4 July 1987\n\n");
    printf("\nR - Request Service from 150XB");
    printf("\nE - Edit Commands.");
    printf("\nQ - Quit\n");
    printf("\nEnter Selection letter ");
    scanf("%1s", &ch);
    return(toupper(ch));
}

editcommand()
{
    system("cls");
    printf("\nCommand Definition Section\n");
```

```
printf("\nEnter Op Code? ");
scanf("%3d", &command);
printf("\nEnter F for fixed or V for variable length command? ");
scanf("%1s", &lenu);
lenu = toupper(lenu);
printf("\nEnter Response length? ");
scanf("%3d", &ilen);
printf("\nEnter Command frame length? ");
scanf("%3d", &olen);
crc = 0;
for( index = 1; index <= olen; index++)
{
    printf("%s %d %s", "\nEnter data byte #", index, " = ");
    scanf("%3d", &out[index]);
    crc = 2 * crc;
    crc = (crc + (crc / 256) + out[index]) & 255;
}
if (lenu == 'V') printf("\n%s %d\n", "Variable Length CRC is:",
    crc);

printf("\nPress space bar to return to Main Menu ");
getch();
}

acceptframe()
{
    outp(PORT+3, 1);
    ftype = getbyte();
    switch(ftype)
    {
        case 0x10 :
        case 0x20 :
        case 0x30 :
        {
            command = getbyte();
            leni = (command & 128) ? 'V' : 'F';
            break;
        }
        case 0x40 :
        {
            command = getbyte();
            ilen = 0;
        }
    }
}
```

```

        leni = 'F';
        break;
    }
    case 0x50 :
    {
        command = getbyte();
        leni = 'V';
        leno = 'V';
    break;
    }

    default:
    {
        command = getbyte();
        leni = (command & 128) ? 'V' : 'F';
        break;
    }
}
if(leni == 'V')
{
    ilen = getbyte();
    data = getbyte();
    ilen = (256 * (data & 0x7F) + ilen);
    if (ftype == 0x50) olen = leni;
}
crc = 0;
for(index = 1; index <= ilen; index++)
{
    data = getbyte();
    in[index] = data;
    if (ftype == 0x50) out[index] = data;
    crc = 2 * crc;
    crc = (crc + (crc / 256) + data) & 255;
}
if(leni == 'V')
{
    data = getbyte();
    if(data != crc) printf("%s %d\n", "CRC should have been ",
    crc);
}

```

```
printf("\n %s %d\n", "frame type = ", ftype);
printf(" %s %d\n", "command = ", command);

for(index = 1; index <= ilen; index++)
    printf(" %d %s %d\n", index, " data = ", in[index]);

if (ftype == 0x50) printf("\nTest Frame!\n");

printf("\nPress space bar to return to Main Menu ");
getch();
}
sendframe()
{
    putbyte(ftype);
    putbyte(command);
    if(leno == 'V')
    {
        crc = 0; /* zero out crc accumulator */
        data = olen & 255;
        putbyte(data);
        data = olen / 256;
        putbyte(data);
    }
    for(index = 1; index <= olen; index++)
    {
        data = out[index];
        putbyte(data);
        crc = 2 * crc;
        crc = (crc + (crc / 256) + data) & 255;
    }
    if(leno == 'V')
    {
        putbyte(crc);
    }
    outp(PORT+3, 1);

    printf("\n %s %d\n", "frame type = ", ftype);
    printf(" %s %d\n", "command = ", command);

    for(index = 1; index <= olen; index++)
        printf(" %d %s %d\n", index, " data = ", out[index]);

    if (ftype == 0x50) printf("\nTest Frame!\n");
```

```
printf("\nPress space bar to return to Main Menu ");
getch();
}

requestservice()
{
system("cls");
printf("Enter C-ommand, Q-uery, R-esponse, S-tatus or
T-est? ");
scanf("%1s", &type);
type = toupper(type);
switch(type)
{
case 'C': ftype = 0x10; break; /* command */
case 'Q': ftype = 0x20; break; /* query */
case 'R': ftype = 0x30; break; /* response */
case 'S': ftype = 0x40; break; /* Status */
case 'T': ftype = 0x50; break; /* Test */
default:
{
ftype = 0x10;
printf("\nDefaulting to command\n");
break;
}
}
printf("\nRequest Service\n");
outp(PORT, DEVICE);
outp(PORT+3, 0);
putbyte(DEVICE);
directive = getbyte();
switch(directive)
{
case 7:
case 5: acceptframe(); break;
case 6:
case 4: sendframe(); break;
case 2: resetit(); break;

default:
{
printf(" %s %d", "Illegal Directive!!!", directive);
printf("\nPress space bar to return to Main Menu ");
}
```

```
    getch();
    break;
}
}
outp(PORT+3, 1); /* put IR line back to unasserted - high */
}

main(argc, argv, envp)
int argc;
char *argv[];
char *envp[];

{
    outp(PORT+3, 0xc0); /* set 8255 to mode 2          */
    outp(PORT+3, 1);   /* set IR high - unasserted    */

    while((item = menuchoice()) != 'Q')
    switch(item)
    {
        case 'R': requestservice(); break;
        case 'E': editcommand(); break;
        case 'Q': break;

        default: printf("\nIllegal Command!"); break;
    }
    system("cls");
    outp(PORT+3, 1); /* set IR high - unasserted again */
}
}
```

Pascal Version of Program

```
{ output - is data from computer to 150XB/C          }
{ input - is data from 150XB/C to the computer      }
    }

program fio;

const
    Maxbytes = 550;
    Padr = 768;
    DevID = 129;
```

```

COMMAND = 16; { command' }
QUERY = 32; { query' }
RESPONSE = 48; { response' }
STATUS = 64; { Status' }
TEST = 80; { Test' }

type
  darray = array [1..Maxbytes] of integer;

var
  indat:darray; { input data array' }
  outdat:darray; { output data array' }
  directive:integer; { frame directive' }
  opcode:integer; { frame op code or command byte' }
  ftype:integer; { frame type byte' }
  crc:integer; { crc for variable length frames' }
  data:integer; { general purpose data byte variable' }
  item:char; { user menu choice' }
  leno:char; { output frame length category' }
  leni:char; { input frame length category' }
  olen:integer; { output frame length' }
  ilen:integer; { input frame length' }

procedure putbyte(byte:integer);
begin
  port[Padr] := byte;
  repeat
    byte := port[Padr+2];
  until odd(byte div 128); { test for seventh bit set }
end; { putbyte }

function getbyte: integer;
var
  byte: integer;
begin
  byte := port[Padr];
  repeat
    byte := port[Padr+2];
  until odd(byte div 32); { test for third bit set }
  getbyte := port[Padr];
end; { getbyte }

```

```
procedure resetit;
var
  ch :char;
begin
  port[Padr+3] := 1;
  writeln;
  writeln(' Received Reset Directive!!! ');
  writeln(' press <Enter> for Main Menu... ');
  readln(ch);
end; { resetit }

function menuchoice: char;
var
  ch :char;
begin
  ClrScr;
  writeln;
  writeln('Program to test 150XB/C Option Port Protocols');
  writeln('S. Simpson   4 September 1987');
  writeln;
  writeln('R - Request Service from 150XB/C');
  writeln('E - Edit Commands. ');
  writeln('Q - Quit');
  writeln;
  write('Enter Selection letter? ');
  readln(ch);
  menuchoice := ch;
  ClrScr;
end; { menuchoice }

procedure editcommand;
var
  index:integer;
  ch:char;
begin
  writeln;
  writeln('Command Definition Section');
  writeln;
  write('Enter Op Code? ');
  readln(opcode);
  write('Enter F for fixed or V for variable length command? ');
  readln(leno);
```



```

write('Enter Response length? ');
readln(ilen);
write('Enter Command frame length? ');
readln(olen);
writeln;
crc := 0;
for index := 1 to olen do
begin
  write('Enter data byte #', index, ' = ');
  readln(outdat[index]);
  crc := 2 * crc;
  crc := (crc + (crc div 256) + outdat[index]) mod 256;
end;
if leno in ['v','V'] then writeln('Variable Length CRC is: ', crc);
writeln;
write('Press <Enter> to return to Main Menu ');
readln(ch);
end; { editcommand }

procedure acceptframe;
var
  index:integer;
  ch:char;
begin
  port[Padr+3] := 1;
  ftype := getbyte;

  { set up default conditions }

  opcode := getbyte;
  if odd(opcode div 128) then leni := 'V' else leni := 'F';

  { above are also conditions for ftypes of COMMAND, QUERY
  and RESPONSE }

  case ftype of
    STATUS :
      begin
        ilen := 0;
        leni := 'F';
      end;

```

```

TEST :
begin
  leni := 'V';
  leno := 'V';
end;

end; { case ftype of }

if leni in ['v','V'] then
begin
  ilen := getbyte;
  data := getbyte;
  ilen := (256 * (data mod 128) + ilen);
  if (ftype = TEST) then olen := ilen;
end;

crc := 0;

for index := 1 to ilen do
begin
  data := getbyte;
  indat[index] := data;
  if (ftype = TEST) then outdat[index] := data;
  crc := 2 * crc;
  crc := (crc + (crc div 256) + data) mod 256;
end;

if leni in ['V','v'] then
begin
  data := getbyte;
  if(data <> crc) then writeln('CRC should have been ', crc);
end;

writeln;
writeln('frame type = ', ftype);
writeln('command = ', opcode);
writeln;

for index := 1 to ilen do
  writeln(index, ' data = ', indat[index]);

if (ftype = TEST) then writeln('Test Frame!');
writeln;

```

```
write('Press <Enter> to return to Main Menu ');
readln(ch);
end; { acceptframe }

procedure sendframe;
var
  index:integer;
  ch:char;
begin
  putbyte(ftype);
  putbyte(opcode);
  if leno in ['V','v'] then
  begin
    crc := 0; { zero out crc accumulator }
    data := olen mod 256;
    putbyte(data);
    data := olen div 256;
    putbyte(data);
  end; { if leno in ['v','V'] }
  for index := 1 to olen do
  begin
    data := outdat[index];
    putbyte(data);
    crc := 2 * crc;
    crc := (crc + (crc div 256) + data) mod 256;
  end; { for index := 1 to olen }
  if leno in ['v','V'] then putbyte(crc);
  port[Padr+3] := 1;

  writeln;
  writeln('frame type = ', ftype);
  writeln('command = ', opcode);
  writeln;

  for index := 1 to olen do
    writeln(index,' data = ',outdat[index]);

  if (ftype = TEST) then writeln('Test Frame!');
  writeln;

  write('Press <Enter> to return to Main Menu ');
  readln(ch);
end; { sendframe }
```

```
procedure requestservice;
var
  ch:char;
begin
  writeln;
  write('Enter C-ommand, Q-uery, R-esponse, S-tatus or T-est? ');
  readln(ch);
  writeln;

  { set up default case }
  ftype := COMMAND; { command }

  case ch of
    'Q','q': ftype := QUERY; { query }
    'R','r': ftype := RESPONSE; { response }
    'S','s': ftype := STATUS; { Status }
    'T','t': ftype := TEST; { Test }
  end; { case ch of }

  writeln('Request Service');
  port[Padr] := DevID;
  port[Padr+3] := 0;
  putbyte(DevID);
  directive := getbyte;

  case directive of
    7, 5: acceptframe;
    6, 4: sendframe;
    2: resetit;
  end; { case directive of }

  port[Padr+3] := 1; { put IR line back to unasserted - high }
end; { requestservice }

begin { main }
  port[Padr+3] := 192; { set 8255 to mode 2 }
  port[Padr+3] := 1; { set IR high - unasserted }

  repeat
    item := menuchoice;
    case item of
      'R','r': requestservice;
      'E','e': editcommand;
    end; { case item of }
  repeat
```

```

until item in ['q','Q'];
port[Padr+3] := 1; { set IR high - unasserted again }
end.

```

Basic Version of Program

```

100 DEFINT B-Z
110 DIM V(550), P(550)
120 ' Main Menu
130 CLS
140 OUT &H303, &HC0 : OUT &H303, 1
150 PRINT "Program to test 150XB/C Option Port Protocols"
160 PRINT "S. Simpson 24 June 1987" : PRINT
170 PRINT "1) Request Service from 150XB/C"
180 PRINT "2) Edit Commands."
190 PRINT "3) quit" : PRINT
200 PRINT "Enter Selection number & press <ENTER>";:IN-
PUT A
210 IF A < 1 THEN BEEP : GOTO 200
220 IF A > 3 THEN BEEP : GOTO 200
230 ON A GOSUB 250, 430, 1190
240 GOTO 130

250 ' General Request Service Dialog
260 CLS
270 PRINT "Enter a 0 for command or 1 for query - ";:INPUT A
280 IF A = 0 THEN F = 16 ELSE F = 32
290 PRINT : PRINT "Request Service" : PRINT
300 OUT &H300, 129 : OUT &H303, 0
310 ' PRINT "ID written & IR asserted"
320 GOSUB 1080
330 ' PRINT "ID read by instrument"
340 GOSUB 1100
350 ' PRINT "directive byte read = ";HEX$(D)
360 IF (D=7) OR (D=5) THEN GOTO 570 ' accept frame
370 IF (D=6) OR (D=4) THEN GOTO 820 ' send frame
390 IF D = 2 THEN GOTO 1170 ' reset directive
400 BEEP : PRINT : PRINT "illegal directive!"
410 PRINT "Enter Directive to use "; : INPUT D : PRINT :
GOTO 350
420 RETURN ' should never get to here

```

```
430 ' Command Editing Routines
440 CLS : PRINT : PRINT "Command Definition Section" :
PRINT
450 PRINT "Enter Op Code "; : INPUT C
460 PRINT "Enter 0 for fixed or 1 for variable length command
";:INPUT Z
470 PRINT "Enter Response length "; : INPUT R
480 PRINT "Enter Command frame length "; : INPUT L : Q = 0
490 FOR I = 1 TO L
500 PRINT "Enter data byte #";I;" = "; : INPUT V(I)
510 Q = 2 * Q : Q = (Q - ((Q AND 256) = 256) + V(I)) AND
255
520 NEXT I
530 IF Z = 0 THEN GOTO 550 ' not variable length
540 PRINT : PRINT "Variable length command CRC is:
";HEX$(Q)
550 PRINT : PRINT "Press <ENTER> to return to Main Menu
"; : INPUT A$
560 RETURN

570 ' accept a frame directive
580 PRINT "Directive to Accept a Frame"
590 OUT &H303, 1
600 ' PRINT "IR line released (now high again)."
```

```
610 GOSUB 1100 : F = D
620 ' PRINT "Frame Type received = ";HEX$(D)
630 GOSUB 1100 : C = D
640 ' PRINT "Opcode received = ";HEX$(D)
650 IF (C AND 128) = 0 THEN GOTO 700 ' not variable length
660 GOSUB 1100 : R = D
670 ' PRINT "Low frame count received = ";HEX$(D)
680 GOSUB 1100 : R = (256 * (D AND &H7F) + R)
690 ' PRINT "High frame count received = ";HEX$(D)
700 Q = 0
710 FOR I = 1 TO R
720 GOSUB 1100 : P(I) = D
730 Q = 2 * Q : Q = (Q - ((Q AND 256) = 256) + D) AND
255
740 ' PRINT I;" Data received = ";HEX$(D)
750 NEXT I
760 IF (C AND 128) = 0 THEN GOTO 800 ' not variable length
770 GOSUB 1100 : Q1 = D
```

```
780 ' PRINT "CRC byte received = ";HEX$(D)
790 IF Q1 <> Q THEN BEEP : PRINT "CRC should have been
";HEX$(Q)
800 PRINT : PRINT "Press <ENTER> to return to Main Menu";
: INPUT A$
810 RETURN

820 ' send a frame directive
830 PRINT "Directive to Send a Frame"
840 D = F : GOSUB 1060
850 ' PRINT "Wrote frame type ";HEX$(D)
860 D = C : GOSUB 1060
870 ' PRINT "Wrote command ";HEX$(D)
880 IF Z = 0 THEN GOTO 940 ' not variable length command
890 Q = 0 ' zero out crc accumulator
900 D = L AND 255 : GOSUB 1060
910 ' PRINT "Wrote low frame length "; HEX$(D)
920 D = INT(L/256) : GOSUB 1060
930 ' PRINT "Wrote high frame length "; HEX$(D)
940 FOR I = 1 TO L
950 D = V(I) : GOSUB 1060
960 Q = 2 * Q : Q = (Q - ((Q AND 256) = 256) + D) AND
255
970 ' PRINT I;" Wrote data byte ";HEX$(D)
980 NEXT I
990 IF Z = 0 THEN GOTO 1020 ' not variable length command
1000 D = Q : GOSUB 1060
1010 ' PRINT "Wrote CRC byte ";HEX$(D)
1020 OUT &H303, 1
1030 ' PRINT "Put IR line back to high (unasserted)."
```

```
1040 PRINT: PRINT "Press <ENTER> to return to Main
Menu.";
1050 INPUT A$ : RETURN

1060 ' Wait for 150XB/C to accept a byte from us
1070 OUT &H300, D
1080 S = INP(&H302) : IF (S AND 128) = 0 THEN GOTO 1080
1090 RETURN

1100 ' wait for 150XB/C to write us a byte
1110 D = INP(&H300)
1120 S = INP(&H302) : IF (S AND 32) = 0 THEN GOTO 1120
```

```
1130 D = INP(&H300)
1140 RETURN

1170 ' reset routine
1180 RETURN

1190 ' Clean up & exit the program
1200 OUT &H303, 1 'put IR line back to high
1210 SYSTEM
1220 END
```