

# BenchSystem™ Test Controller, Btc1

Tools for Scientists and Engineers

## Features

- Direct programmable control of laboratory test equipment with GPIB and SERIAL control ports.
- Local and remote control of analog and digital circuits via the LIB expansion I/O modules.
- Console port for program development, data logging, and operator interface.
- Interprets standard ASCII text instructions.
- Embedded and Stand-alone operation.
- 32 bit timer with 1millisecond resolution.
- Small footprint on the bench, about 5 x 6 inches.
- Very low cost, single-board design.

## Overview

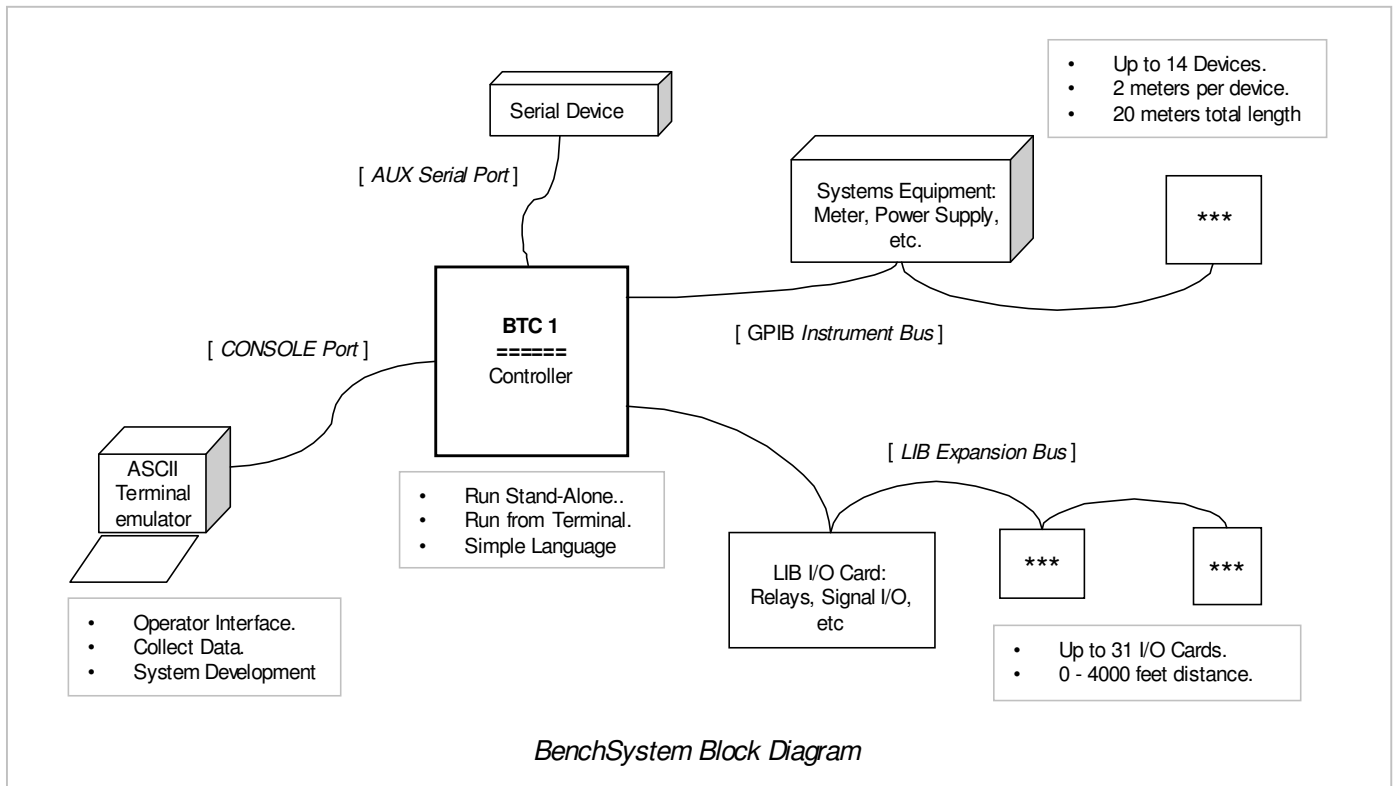
The BenchSystem Test Controller BTC1 allows a scientist or engineer, with minimal programming experience, to build small test and measurement control systems. The direct control of test equipment and a simple programming paradigm make it a natural tool for component qualification, incoming inspection, circuit development, etc.

The BenchSystem Operating language (BSOP) allows both mathematical and logical operations on integer, floating point, and text string data types. Simple port commands and string operations make it easy to control peripheral equipment or to format output data for a spreadsheet. Informal formatting rules and a modular programming model facilitate code organization and reliability.

Programs can be written with any text editor and then uploaded to the BTC1 using terminal emulation. Single stepping, break operations, and descriptive error messages aid in code debugging. Interactive Console commands make it easy to experiment with attached equipment. A program can be saved on-board to run automatically at power up.

The Local Interface Bus is a local/remote expansion bus for a family of I/O modules that provides electronic signal interfacing. Up to 31 modules are easy to attach to both the hardware and the program.

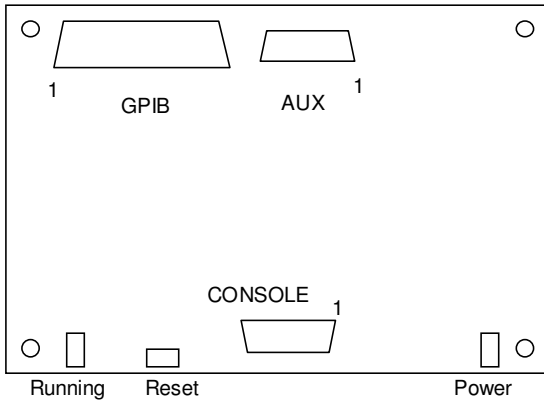
The BTC1 requires 12 volts at 250 mA. It is a board-level product and measures 4.9" x 6.2" x 0.8".



## Hardware Description

This section explains the functionality of the BTC data ports. *Console* commands and *Programming* commands for each port are shown. The programming examples also illustrate features of the language and programming techniques.

In the following definitions: **(in)** refers to a BTC input signal, **(out)** to a BTC output, and **(i/o)** can be either.



*Connector Layout and Pin One Locations*

(To mount, wire, and program a BenchSystem see the “Bsys Getting Started.pdf” file.)

### Buttons and Indicators

**Power Lamp** ON while 12Vdc power is applied at the LIB connector.

**Running Lamp** ON while the program is running. ON momentarily during the power-up memory test.

**Reset Button** Press to create a power-on reset condition on the board, without affecting the 12Vdc-system power.

### Power-on Reset

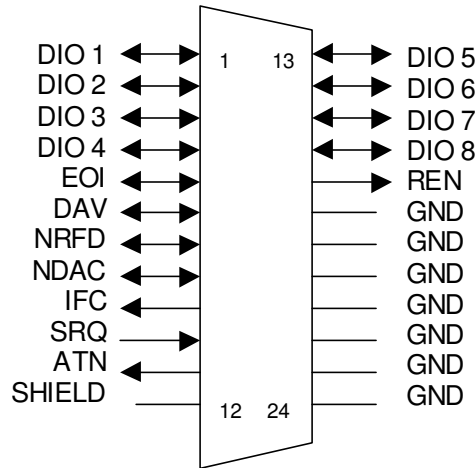
Applying 12Vdc power at the LIB connector or pressing the Reset button causes the following events:

- On-board microprocessor memory is tested
- Control ports are initialized.
- BSOP program memory is tested cleared.
- BSOP storage memory is checked for a program. If one exists, it is loaded and run after a short delay; otherwise the Console interface goes operational.

### GPIB Control Port

The General Purpose Interface Bus (GPIB), per the standard IEEE-488-1978, is a byte-wide asynchronous interface to instrumentation and computer peripherals. The BTC1 GPIB controller does not support SRQ, Parallel Poll, or the ability to Pass Control.

You can connect up to 14 devices to the port, setting the device addresses to unique values from 1 to 30. Interconnect lengths must not exceed 2 meters between devices, or 20 meters total. Configure the cabling in linear or *daisy chain* fashion, not as a star or a loop.



GPIB CONTROLLER CONNECTOR

### Console example

Console commands are interactive and work only when typed directly at the console terminal.

Type and Enter:	<b>putgpib 16,read?</b>	to tell a Digital Volt meter to take a reading.
Type and Enter:	<b>getgpib 16</b>	to read the Volt meter and print that text to the console.

### BSOP Program example

Program commands work only from within a program file that has been “uploaded” to the controller and “run”.

```

Code Sample
/ Run GPIB communications. Illustrates automatic type conversions as the Vmeter returns a text representation of a number.
/ Will wait for device when reading from it.

alias    Vmeter gpib 16;           / create a name for this equipment at address 16 on the GPIB port

float    x ;                       / create some storage variables to illustrate automatic type conversion
int      y;
string   s;

start:   clrgpib ;                 / initialize devices on gpib bus.
         x = gpib 16 "read? ";     / send command and then read from device. Store reading as a floating point value.
         y = gpib 16 "";           / read without sending, and store the reading as an integer value.
         gpib 16 "read? ";        / send command but don't read from device.
         s = gpib 16 "read? ";    / send command and then read from device. Store reading as a text string.

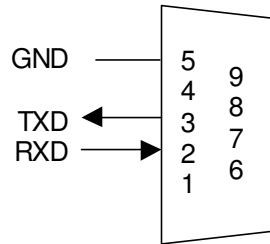
         s = Vmeter "read? ";     / Same as above but easier to follow.
```

## AUX Serial Control Port

The Auxiliary serial port conforms to the RS-232 standard for signal levels. The bit rate and data formats are programmable. The port is full duplex, configured as DTE, and does not support hardware or software handshaking.

Line length is generally limited to 50 feet. Protocol adapters are available to extend the range by changing to a different signal scheme, such as RS-422 for up to 4000 feet or modems which have unlimited range when using established phone lines.

**Note:** Some AUX port hardware is shared with the LIB port, so all data exchanges must be completed before the other port is used. Using any of the port commands automatically connects the required hardware, while maintaining the respective bit rates and data formats.



AUX CONTROL PORT CONNECTOR

## Console example

Console commands are interactive and work only when typed directly at the console terminal.

Type and Enter	<b>putcomm read?</b>	to tell a Digital Volt meter to take a reading.
Type and Enter	<b>getcomm</b>	to read the receive buffer and print that text to the console.

## BSOP Program example

Program commands work only from within a program file that has been “uploaded” to the controller and “run”.

```

                                Code Sample
/ Run AUX communications. Illustrates automatic type conversions as the Vmeter returns the text representation of a number.
/ Will wait when reading an empty receive buffer.

alias    Vmeter comm;           / create a name for this equipment on the AUX comm port

float    x ;                    / create some storage variables to illustrate automatic type conversion
int      y;
string   s;

start:   defcomm 19200 '7E1\r'; / define baud rate, character formatting and termination character.

         x = comm 'read? ';      / send command and then read the rx buffer. Store reading as a floating point value.
         y = comm "";            / read without sending, and store the reading as an integer value.
         comm 'read? ';         / send command without reading the receive (rx) buffer.
         s = comm 'read? ';     / send command and then read the rx buffer. Store reading as a text string.

         s = Vmeter 'read? ';   / Same as above but easier to follow.

         s = commchr ;          / read the next character from the receive buffer. Wait until there is one if it's empty.
                                / Allows program action to be taken on any character, not just the terminator.
    
```

### CONSOLE Command Port

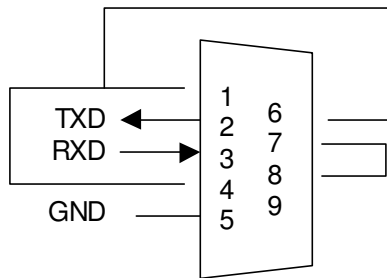
The Console port conforms to the RS-232 standard for signal levels. The same discussion of distance and alternate protocols, as described in the above AUX port section, applies to the Console port.

The BTC1 CONSOLE port is full duplex and is configured as DCE. It operates at 9600 Baud and 8N1 data format. Although it does not support handshaking, the handshake signals are wired together in case the console device must “see” the proper hardware levels.

Use the “console” and “conschr” BSOP commands to communicate with the console device from your program.

When connected to an ASCII terminal, or a PC running a terminal emulation program, the console device can be used for:

- Code development: edit the (BSOP) code file, upload it to the BTC1, and debug the program.
- System design and development via interactive controller port commands.
- An interactive operator interface to your BSOP program, i.e. display messages and receive keyboard inputs.
- The downloading of data, from the BSOP program, to a file or spreadsheet.



AUX CONTROL PORT CONNECTOR  
(showing handshake signal connections)

### BSOP Program example

Program commands work only from within a program file that has been “uploaded” to the controller and “run”.

#### Code Sample

```

/ Run an operator interface at the CONSOLE. When reading an operator response, 'console' will wait for the Enter key.
/ Illustrates a form of smart prompting.

int    ch, chan ;

start:  chan = 1;                               / initialize an automatic channel selection value
lp2:    prompt = "Calibrate CHAN 1-16: " @ chan ; / show the automatic channel choice, chan
        ch = console prompt ;                   / to Accept it press Enter, else type and Enter a value
        if( ch > 0 )    chan = ch ;              / if something was typed use it, otherwise use chan
        if( chan > 16 ) qt2;                     / quit when all channels have been used

        --- some code that uses the value of chan ---

        chan = chan + 1; goto lp2;               / auto increment the channel and re-prompt.
qt2:    exit;

        s = conschr ;                            / read the next character from the console receive buffer. Wait until there is one if it's empty.
                                                / Allows program action to be taken on any key, not just Enter.
    
```

## LIB Expansion Port

The Local Interface Bus LIB port connects a network of I/O modules that provides various functional and interface capabilities. The LIB modules conform to a “plug in” networking and programming scheme. From one simple program command ( **lib** ), message data is routed over the network and verified automatically.

The LIB network supports up to 31 modules, a network length of 0 to 4000 feet, and uses RS-485 signal specifications. It uses standard phone cables (4 or 6 wires) and plugs (4-6 or 6-6). The network carries data and 12Vdc power to the modules. Each module has two identical connectors to simplify the daisy-chained connection. Cable and plug assemblies or low-cost crimp tools are readily available.

Termination resistance (120 ohms from Comm A to Comm B) is necessary to prevent signal reflections from mismatched impedance. For network lengths 0 to 20 feet use one termination resistor, over 20 feet use two (one at each end of the network).

**Note:** Some AUX port hardware is shared with the LIB port, so all data exchanges must be completed before the other port is used. Using any of the port commands automatically connects the required hardware, while maintaining the respective bit rates and data formats.

## Console example

Console commands are interactive and work only when typed directly at the console terminal.

Type and Enter:	<b>putlib 1,E3S2</b>	to set relay #3 on a BRR1 module and check the status of relay #2
-----------------	----------------------	---

## BSOP Program example

Program commands work only from within a program file that has been “uploaded” to the controller and “run”.

### Code Sample

/ Set a pattern to a BRR relay card on the LIB. Illustrates device path naming and test naming in alias statements.

```
alias relays lib 1; / create a name for this equipment at address 1 on the LIB port
alias set_shunt relays 'D2;E3-5'; / create a name for a simple test configuration Use a subroutine if complex.
string s;

start:
lib 1 'D2;E3-5'; / Break #2, delay , Make #3-5. see the BRR1 data sheet
s = lib 1 'S2 '; / read the 'E' or 'D' status of relay #2

relays 'D2;E3-5'; / this works the same as the previous command but may be more clear
set_shunt ; / or better yet keep it all in the alias. If it appears elsewhere you need only to define it once.
```

## Time Clock

The time clock is a 32-bit counter with a 1-millisecond time base that returns a signed integer value. The count begins at 0, at power up, and continues to 2,147,483,647 then rolls over to -2,147,483,648 and counts up to -2, -1, and 0 again; taking about 50 days round trip. This form of output makes differential time measurements a simple subtraction, even through the roll over (given the 32 bit truncation).

## BSOP Program example

Program commands work only from within a program file that has been “uploaded” to the controller and “run”.

### Code Sample

/ Run a time of day clock on the console. Illustrates operational modules using subroutines.

```
alias  NL      console "\r\n"      ;           / doesn't clutter code this way
string Time, ts;                    / time: h,m,s

start:  setclock ;                   / get the set time

lp:     calc_time ; console Time ;     / get time of day and print it to the console.
        console " \r";                / space-over any remains of last print and stay on the same line.
        waitms 1000; goto lp;         / run forever, ctrl-c to bail out. (it's ok to put 2 statements on a line)

/***** TIME ROUTINES *****/
alias  C      " : " ;                / format separator for 'hr : min : sec'
string ts;                               / set time input: h,m
int    t, b;                             / tod in seconds, baseline set time

setclock
{
    ts = console "Enter current 24hr TIME (hr,min); "; NL;           / time of day
    b = clockms - (arg 1 ts * 3600 + (arg 2 ts * 60 )) * 1000 ;     / baseline for tod
}

calc_time
{
    t = (clockms - b) / 1000 % (3600 * 24);                          / 24hr time in secs
    Time = t/3600 @C @t/60%60 @C @t%60 ;
}

```

## Console Commands

This section gives the syntax and a description of each command recognized by the **console interface**. Commands are grouped by function. *These commands WILL NOT WORK within a Program.*

The terminal must be configured as: ASCII, 9600baud, 8 data bits, No Parity, 1 stop bit, no handshake, half-duplex (or echo). Attaching line feeds to sent and /or received strings is a matter of taste; line feeds when sending works well for most things.

Press **Enter** after typing a command. Use **Backspace** to correct any typos. If a command gets **stuck** waiting for a program or equipment response that is not coming, press **ctrl-c** to abort the command

### Interactive Control Port commands

See the hardware descriptions above for usage examples.

putgpib a,text	Send text to unit 'a'(1-30) on the GPIB port.
getgpib a	Read text from unit 'a'(1-30) on the GPIB port, and print it on the console screen.
putcomm text	Send text to equipment on the AUX COMM port.
getcomm	Read the receiver buffer from the AUX COMM port, and print it on the console screen.
putlib a,text	Send text to unit 'a'(1-31) on the LIB port, and print the response on the console screen.

### Program Development commands

load	Following this command, upload an BSOP program text file from the console to the BTC1.
list	List the program code lines to the console screen.
save	Save the BTC1 program code to memory. The program will run at power up or following a RESET.
erase	Erase the program code from memory so that no program runs from power up.
run	Run the BSOP program from the "start" label. Press <b>ctrl-c</b> to ABORT the program.
step	Single step the BSOP program code from the "start" label. Press 'space bar' to take the next step, ' v' to step over a subroutine, or 'r' to run (Esc to stop). Press <b>ctrl-c</b> to ABORT the program.

### Operational commands

status	<p>Read the Status of the BTC1 and print it to the console. An example status string is: 80,BTC1a,Unit_Id</p> <ul style="list-style-type: none"> <li>- 80 is the BTC1 Error byte in hexadecimal format b8-b1 where: <ul style="list-style-type: none"> <li>♣ b8 = Power up or RESET has occurred</li> <li>♣ b3 = The unit ID text has been lost due to an Eeprom fault.</li> <li>♣ b2 = LIB receiver error(s): parity, data overrun, framing. Check the LIB termination rules.</li> <li>♣ b1 = LIB receive buffer overrun.</li> </ul> </li> </ul> <p>For example, 06 hex indicates b3 and b2 are set. Reading the status clears the error byte to '00'.</p> <ul style="list-style-type: none"> <li>- BTC1a is the Unit Revision as set by the factory.</li> <li>- Unit_Id is the user-defined text string from the 'id' command.</li> </ul>
id text	Write ID identification text (0-15chars) to non-volatile memory, for display with the 'status' command.
help	List the commands and syntax on the console screen.

## BSOP Program Commands

This section describes the BSOP programming commands and procedures that are specific to the BTC1. See the BSOP language data sheet for complete language and programming information. *These commands WILL NOT WORK from the Console terminal.*

In the following definitions:

- **s1** arguments are string constants, string variables, or string results; 0 to 126 characters long.
- **n1** arguments are any constants, variables, or results, that represent a number, i.e. 10 or "12.3e-2".

where "results" is the return value of a command or of a parenthetical operation.

### Control Port commands

To read a device without sending it a command, send the null string ("") for s1. A device is "optionally read" only if the code statement is expecting a result; i.e. for assignment or as an argument of another command. See the BSOP datasheet.

<b>clrgpib</b>	Force IFC (interface clear), DCL (device clear), and initialize the GPIB port to be the System Controller.
<b>gpib n1 s1</b>	Send command s1 to the device at address n1 (1-30) on the GPIB port, and then optionally read the device.
<b>comm s1</b>	Send command s1 to the device on the AUX COMM port, and then optionally read the device.
<b>commchr</b>	Read one character from the device on the AUX COMM port.
<b>defcomm n1 s1</b>	Define the AUX COMM port baud rate n1 (e.g. 9600) and protocol s1 (e.g. '8N1\r'). <ul style="list-style-type: none"> <li>– The baud rate range is 150 to 500000.</li> <li>– The protocol is "data bits, parity, stop bits, receiver string terminator", with:           <ul style="list-style-type: none"> <li>– 5 to 8 data bits.</li> <li>– E, D, or N for Even, oDd, or No parity</li> <li>– 1 or 2 stop bits</li> <li>– Usually \r (carriage return) or \n (new line) are terminators, but any ASCII character is possible. See the Escape Character section in the BSOP data sheet for more information.</li> </ul> </li> <li>– The DEFAULT configuration is 9600 '8N1\r'.</li> <li>– <b>Note:</b> When defcomm executes, it switches the LIB/AUX hardware over to the AUX port.</li> </ul>
<b>lib n1 s1</b>	Send command s1 to the device at address n1 (1-31) on the LIB port, and then optionally read the device.
<b>console s1</b>	Send command s1 to the device on the CONSOLE port, and then optionally read the device.
<b>conschr</b>	Read one character from the device on the CONSOLE port.

**Note:** LIB port hardware is shared with the AUX port, so all data exchanges must be completed before the other port is used.

#### Code Sample

/ Show various uses of port commands. Illustrates send-only, and receive-only usages. Shows automatic type conversions and / and in-line programming efficiencies.

```
int    num_cyc ;
float  dmm;
string name, info;

start: num_cyc = console "Enter number of test cycles >"; / Prompt and wait for input. Interpret input as integer.
       name   = console "Enter name of test >";          / Prompt and wait for input. Interpret input as string.
       ...
       dmm    = gpib 7 "";                               / Read from instrument at address 7 without prompting. Interpret as float.

       info   = name @ dmm @ "Volts \r\n";              / build string from the test name and reading. Format units and end of line.
       console info ;                                  / send to operator or spreadsheet. Not waiting for input.

/--- advanced coding techniques ---
console (name @ dmm @ "Volts \r\n") ; / Or do the same thing in-line. Eliminates one line and one variable.
console (name @ gpib 7 "" @ "Volts \r\n") ; / Or put more in-line. Eliminates one more line and one more variable.
```

### Time Clock commands

The time clock is a 1-millisecond resolution timer that is useful for differential measurements.

- clockms**            Return the system clock time in milliseconds (- 2,147,483,648 to 2,147,483,647).
- waitms n1**        Delay the program n1 milliseconds. Valid delays are 0 to 65000 ms.

**Code Sample**

/ Show time commands. Illustrates measuring the time of an operation.

```

int      t, elapsed_time ;

start:   waitms 2000;           / wait here for 2 seconds.
        ...
        t = clockms ;         / Read clock time.
        ... run a test here ...
        elapsed_time = clockms - t; / record the time for the test to run
    
```

## Specifications

### Electrical

Parameter	Conditions	Min	Typical	Max	Units
+12VDC Input	0- 40 degC (1)	8.5	12	17.5	V
Input Current			220		mA

Notes: 1. Derate max from 40-70 degC: 17.5V – 140mV/degC

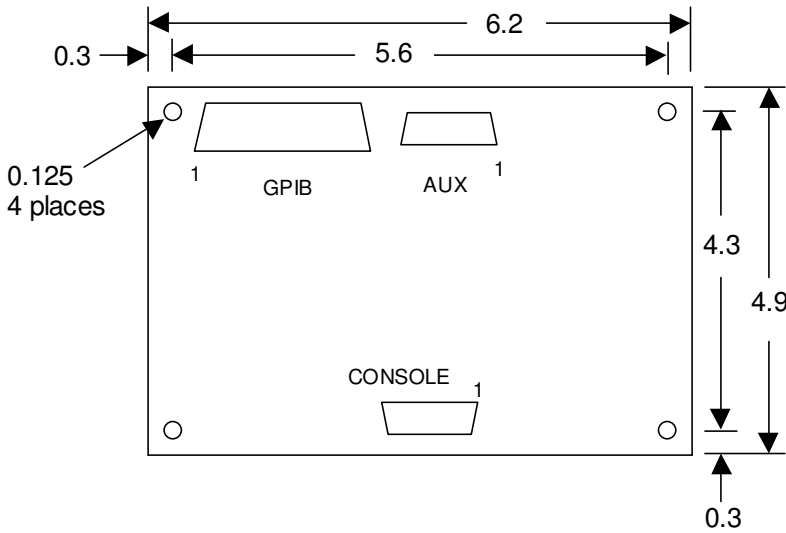
### Timing

Parameter	Min	Typical	Max	Units
Program measurements:				
increment, test and loop		4		mS
multiply, divide		3		
find, arg, copy		3		
comment, subroutine		1		
waitms		1 + time		
convert a float to a text string		5		
LIB character transfer rate	(calculated)	1mS + 260 uS / byte		
GPIB character transfer rate	(calculated)	1mS + 5 uS / byte		
CONSOLE character transfer rate	(calculated)	1mS + 1050 uS / byte		
Auto start delay after power up		1		Second

### Environmental

Operating Temperature: 0 to 70 degrees C.  
 Storage Temperature: -20 to 85 degrees C.^^

## Mechanical



## Revisions

Preliminary: 6/18/03.  
 Beta-1: 11/10/03  
 Beta2: 12/26/03

## Troubleshooting

Power led is not ON.

The 12Vdc is not present at the LIB connector, it is reversed, or the fuse on the card is blown. Measure 11Vdc on both sides of the fuse, F1. Verify that the LIB bus RJ11 plugs are wired one-to-one.

There is no communication at the console terminal and the status led remains ON.

1. There is a program running. Press **ctrl-c** at the console terminal to make it stop.
2. 'He's dead, Jim'. Contact the factory.

## Known Issues

## Board Revision History

7/27/03 Beta for board and spec sheet.

## Future Changes

ExacTest Corporation may make changes to or discontinue the availability of its products and literature at any time.