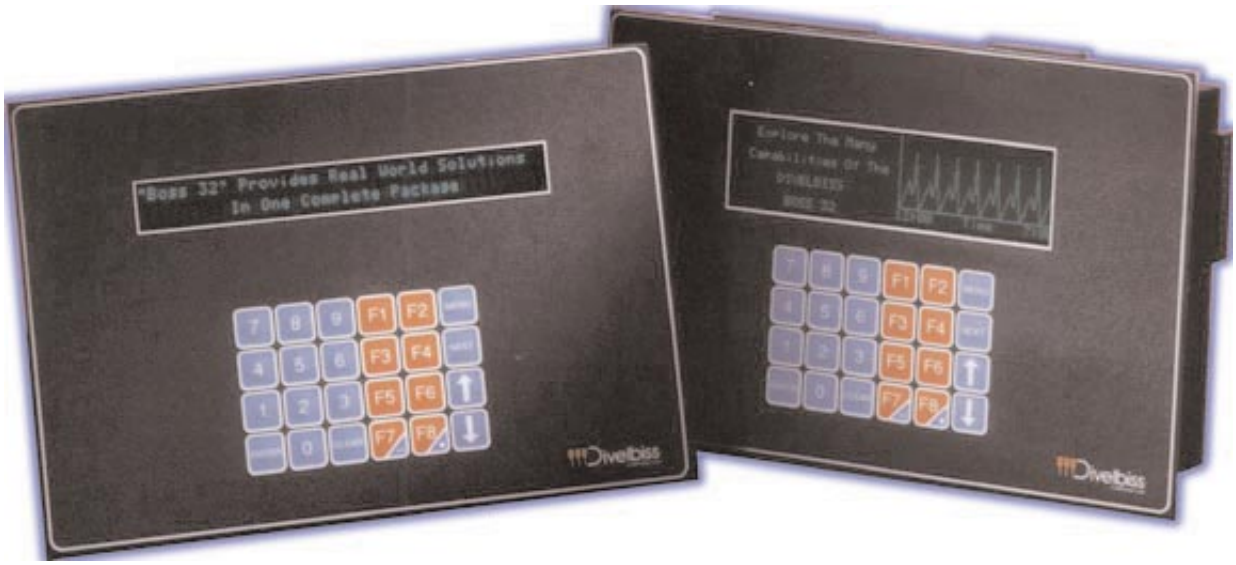
BOSS 32 Hardware Manual



Release Version 1.00

Divelbiss Corporation
9778 Mt. Gilead Rd.
Fredericktown, OH 43019-9533

800-245-2327
www.divelbiss.com

BOSS32 Hardware Manual
Table of Contents

Chapter One
Introduction

The Divelbiss Boss32 is a unique programmable control system that integrates many control functions into one easily used package. It starts as a compact, highly integrated industrial computer system that is programmed using an extended, compiled multitasking BASIC. The Boss32 is fully programmable to suit the user's requirements. It supports onboard control hardware such as a High Speed Counter, a Real Time Clock, Analog Inputs, Analog Outputs, Touch Memory, Four serial ports, nonvolatile memory, and FLASH EPROM storage of the user's programs. Three expansion ports are available for adding optional modules, including high speed counters, analog inputs, analog outputs, stepper motor control, etc.

Several Boss32's can be linked together with a network to provide control over a larger area or to return information to a central point. This network can then be linked to another computer system, which can provide long term data storage and display, supervisory control of the entire network, and perform Statistical Process Control.

WARNING!

The Boss32 Controller, as with other solid state controllers, must not be used alone in applications which would be hazardous to personnel in the event of failure of this device. Precautions must be taken by the user to provide mechanical and/or electrical safeguards external to this device. This device is NOT APPROVED for domestic or human medical use.

NOTICE: The contents and specifications contained in this manual are subject to change without notice.

PLEASE NOTE:
As used throughout this manual, Modbus is a registered trademark of Modicon, Inc.

Description of Features

A Boss32 system consists of the Boss32, options added onboard the Boss32 itself, option modules plugged into the Boss32, and I/O expander modules connected to the Boss32. By choosing the options carefully, an inexpensive system can be built with only the hardware necessary to complete the task.

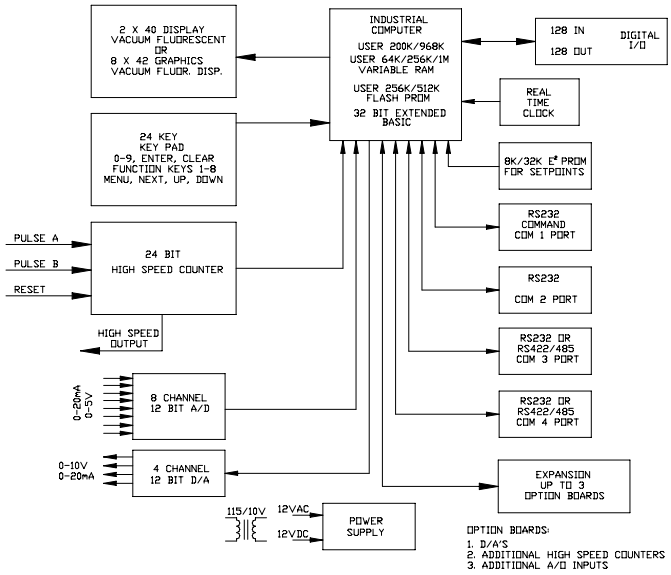
The minimal configuration of the Boss32 contains the microprocessor circuitry with the Bear BASIC compiler in PROM, 256K of program RAM, 256K FLASH EPROM, four serial ports, a real time clock, and the HDIO (High Density I/O) interface. The program RAM is used to hold the user's program source code while it is being written and compiled code to run. All constant data and variable is stored in battery backed up RAM separate from the program RAM. The serial port is used to interface with a terminal or personal computer to allow entry and editing of programs. The onboard FLASH EPROM programmer allows the user's program to be saved and loaded, either as source code or as executable object code.

The Boss32 may be purchased with a variety of options installed, such as a high speed counter, analog inputs, analog outputs, display & keypad, and more memory.

The high speed counter is a 24 bit up/down counter with built-in comparitors, providing a high speed output when a specified count value is reached. This counter may be programmed to support position control, rate metering, tachometer batch control, etc. It can support multiple setpoints under software control.

The Real Time Clock (RTC) maintains the current time and returns it as year, month, day, hour, minute, second, day of week, and day of month. It has battery back-up, and will keep the correct time even when the Boss32 is not powered.

The serial ports are configured as follows: Port 1 is RS-232; Port 2 supports RS-232. Ports 3 & 4 support RS232, RS422 and RS485. RS-232 is used to communicate over short distances. RS-422 is used over longer distances and in electrically noisy environments. RS-485 is used when several units need to communicate over two wires, as in a network. The four onboard serial ports support 2400, 4800, 9600, 19200, 38400 and 57600 baud.



Boss32 Block Diagram

How to Use this Manual

In this manual, the following conventions are used to distinguish elements of text:

BOLD	Denotes hardware labeling, commands, and literal portions of syntax that must appear exactly as shown.
italic	Used for variables and placeholders that represent the type of text to be entered by the user.
EXAMPLE	Used for example programs, sample command lines, and text displayed by the 32-bit controller. Note: For example programs to display properly, the terminal must be set to the correct emulation. Program modifications may be required if you are not using the correct terminal emulation.
SMALL CAPS	Used to show key sequences, such as CTRL-C, where the user holds down the <Ctrl> key and presses the <C> key at the same time.
[ ]	Brackets are used to indicate optional elements of a command, such as **LOAD** [*program_num*] where *program_num* is optional.

In addition, the following symbols appear periodocally appear in the left margin to call the readers attention to specific details in the text:

Indicates that the text contains information to which the reader should pay particularly close attention.

Warns the reader of a potential danger or hazard that is associated with certain actions.

Appears when the text contains a tip that is especially helpful in the installation or operation of the BOSS32 or its components.

This manual is for the Boss32 hardware only. For programming commands, pleaserefer to the appropriate Programming Manuals.

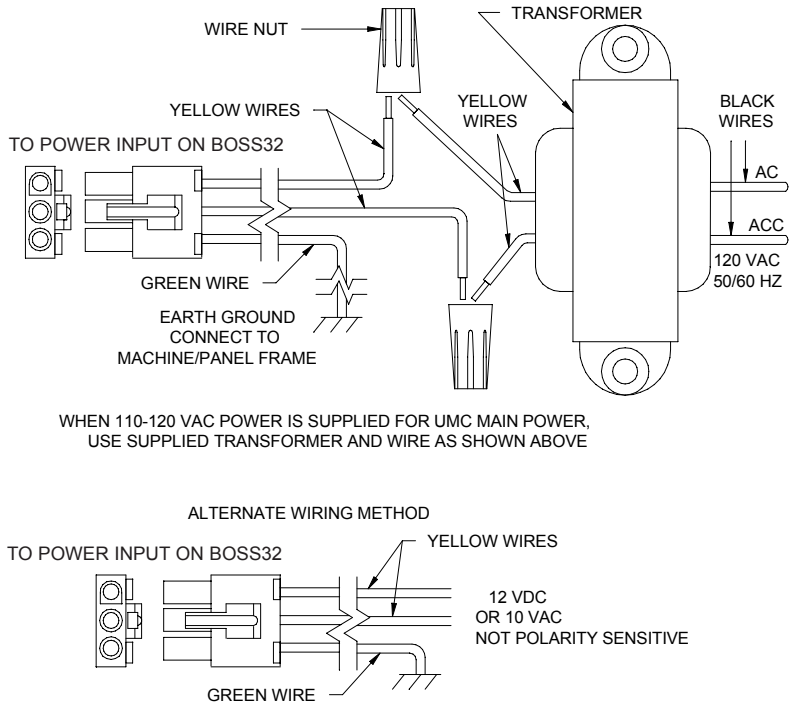
Chapter Two
Getting Started

The Boss32 is a relatively simple computer system to get up and running; normally, even an inexperienced user can have the unit operating a short while after unpacking it. This chapter describes how to set up the Boss32 and use its basic features. The chapter includes sample programs that may be entered to learn about the general functions of the system; these programs may then be used as a starting point from which to develop applications.

Powering Up the Boss32

The Boss32 requires a 10 volt AC or 12 volt DC power source in order to operate. A transformer is available from Divelbiss that allows the unit to be powered from the 120 volt AC power line. The unit has onboard rectification and regulation to generate the voltages used inside the system. 10-12 volts is fed into the system using the 3 pin **POWER INPUT** connector on the front of the unit. It is extremely important to provide an earth ground for the unit, both as a safety precaution and to minimize electrical noise related problems; earth ground is the third prong on a standard 3 prong electrical wall socket.

WARNING: DO NOT USE AUTO TRANSFORMER!



Boss32 Power Supply Wiring Schematic

Connecting to a Terminal or Personal Computer

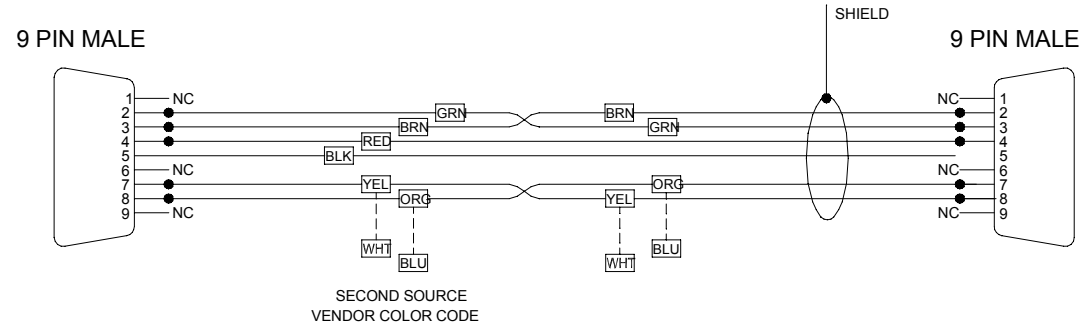
In order to program the Boss32, it must be attached to a terminal or a personal computer that is running a serial communications program. This manual assumes that a personal computer is being used; the operation of the unit is the same in either case. A cable connects the Boss32 to the personal computer; Divelbiss can supply a cable that will allow the unit to be used with an IBM PC or compatible, or a null modem cable can be used. The user can easily manufacture a cable to match other systems. Note that pins 7 and 8 (the RTS and CTS lines) must be connected to valid signals from the terminal/computer, or they must be connected to each other.

Once the cable is installed between the Boss32 COM1 connector and the personal computer, apply power to the personal computer and enter the communications program. The serial communications parameters must be set to 9600 baud, 8 data bits, no parity, 1 stop bit, and XON/XOFF flow control enabled; these values are required in order to match the

default values used by the Boss32. Make sure that the communications program is set to use the serial port that is connected to the Boss32. Apply power to the Boss32; it should respond with a sign-on message followed by the BASIC compiler prompt:

BOSS32 BASIC Compiler Version X.XX
>

Each time that the ENTER key is pressed, the prompt character will be printed on the next line down. The communications program should be set to emulate an ADM-3A or ADM-5 terminal type, in order for the cursor positioning commands on the Boss32 to work correctly. If no prompt appears, holding the ENTER key on the Boss32 panel while powering up will display the current COM 1 settings. Holding the CLEAR key on the Boss32 panel while powering up will cause the Boss32 to boot up at 9600 baud (this one time), but not run any already loaded compiled programs.



Serial Port COM1 Cable Wiring

Programming Languages

Boss32 Basic

The Boss32 comes standard with BOSS32 BASIC programming. This is a proprietary basic programming language utilizing many of the commands and functions from other standard basic languages and is similar to a “Q-BASIC” type programming language. This programming language is active at power up and can be accessed at any time with a terminal connected to COM1. BOSS32 Basic is a compiled and multitasking programming language.

“C” Programming

The BOSS32 has the capability of having programs written in “C”. An off board compiler is required to program “C” functions with the BOSS32. Contact the factory for more information.

Writing a Program in Bear BASIC

The best way to become familiar with the Boss32 is by writing some simple programs; the following sections will use several programs to demonstrate different features. We will start with a very simple program. The first program that is always run on a new computer system looks like this:

100 PRINT “HELLO, WORLD!”

Before entering this program, type **NEW** followed by the ENTER key; this will clear out any existing program that might be in the Boss32 memory. Although the examples in this manual primarily use upper case characters, the Boss32 is not case-sensitive, so lower case and upper case may be used interchangeably. Also, from here on it is assumed that the user presses ENTER at the end of each line.

Enter the one line of the program and then enter **RUN**. The Boss32 will print the message COMPILED followed by HELLO, WORLD!. It will then return with the prompt, awaiting further commands. The screen should look like this:

```
>   NEW
>   100 PRINT “HELLO, WORLD!”
>    RUN
COMPILED
HELLO, WORLD!
>
```

If an error message is printed instead of COMPILED, after typing **RUN**, then line 100 was probably entered incorrectly; try entering it again, being careful to type it exactly as shown. Remember to put the double quote marks around the HELLO, WORLD! message.

To modify a line of BASIC, just enter in a new line using the same line number; the new line will overwrite the existing line. To remove a line of BASIC, enter the line number with nothing after it; the line will be removed from the BASIC program. To look at your BASIC program, enter **LIST**; the program will be displayed on the screen.

Bear BASIC Command Line Operation
Bear BASIC has three different modes of operation: command line entry, compilation of the BASIC program, and execution of the BASIC program. The programmer interacts with Bear BASIC while in the command line mode; this is where programs are entered, edited, loaded, and saved. The user types a command after the compiler prompt (the ‘>’ character), then Bear BASIC performs the command and returns with the compiler prompt again. The BACKSPACE key may be used to back up while entering a command.

When a line is entered while in the command line mode, it is first examined to see if it is a valid direct command; if it is, then the command is performed. If it isn’t a direct command, then it is handled as a line of BASIC source code; if it isn’t a line of BASIC source code, then an error is displayed. If it is valid BASIC source code, then it is entered into the current program.

Using the Command Line Editor
If it is necessary to change a line of the current BASIC program, the line can be retyped; the new line will replace the existing line. This is fine for short lines, but for longer lines this can cause a lot of extra typing. In order to minimize the effort required to modify a line of the program, Bear BASIC includes a line editor as a direct command. The EDIT command allows a single line of BASIC source code to be modified without retyping the entire line.

For EDIT to work properly, the line must be less than 80 characters long, and it must fit on a single line of the terminal. For the cursor positioning to work correctly in EDIT, the console

terminal type must be set to the same terminal settings as the SETOPTION TERMINAL setting. The Boss32 supports ADM-5, VT52, VT100, VT220 and ANSI terminals.

To use EDIT, type **EDIT** followed by the line number of the BASIC line to be modified; for example, EDIT 120. The existing line will be displayed with the cursor at the first character in the line number. The cursor may be moved left and right with the following key combinations:

CTRL-S move the cursor left one character (The key board left arrow key may be used if the terminal program used supports its use.)
CTRL-D move the cursor right one character (The key board right arrow key may be used if the terminal program used supports its use.)
CTRL-A move the cursor to the beginning of the line (The key board HOME key may be used if the terminal program used supports its use.)
CTRL-F move the cursor to the end of the line (The key board END key may be used if the terminal program used supports its use.)

The editor can be toggled between insert and overwrite mode. In overwrite mode, any character typed will replace the existing character at the current cursor position. In insert mode, any character typed will be added at the current cursor position, causing the rest of the line to be moved right one character position. When the editor is invoked, it starts out in overwrite mode.

CTRL-V toggles between overwrite and insert mode

To delete characters from the line, either BACKSPACE or CTRL-G can be used. The BACKSPACE key deletes the character to the left of the current cursor position, causing the rest of the line to be moved left one character position. CTRL-G, on the other hand, deletes the character at the current cursor position, causing the rest of the line to the right to be moved left one character position.

CTRL-G deletes characters from line

When the desired changes have been made to the line, press ENTER to add the modified line back into the program; the modified line will replace the existing line. Pressing ESCAPE or any control key not listed above will abort the changes, leaving the existing line unchanged. After leaving the editor, the compiler prompt will be displayed again.

Loading and Saving Programs With a Personal Computer

If the Boss32 is being used with a personal computer as the console terminal, then it may be desirable to transfer the current BASIC program to and from the computer's disk. This allows the program to be edited on the personal computer, and to be printed out. The exact method of transferring files depends upon the communications program that is being used on the computer; it may be necessary to refer to the manual for that program to ensure proper file transfer.

To transfer a program from the personal computer to the Boss32, enter **NEW** to clear the current program out of memory, then enter **DOWNLOAD**. Execute the command in the com-

munications program to send a text (or ASCII) file, and type in the appropriate file name. The file will be transferred to the Boss32. The file should not be echoed to the screen; if it is being echoed, then it may indicate that a CTRL-Z is stored in the file. If any syntax errors are encountered while sending the file, they will be displayed on the screen. After the file has been transferred, the Boss32 will probably respond with the compiler prompt; if it doesn't, then press CTRL-Z, which turns off the DOWNLOAD mode. If two or more lines were entered using the same line number, then the message: ***Warning: duplicate line numbers detected will be printed***; this probably indicates an error in the BASIC source code.

To transfer a program from the Boss32 to the personal computer, type **LIST** but do not press ENTER. Execute the command in the communications program to receive a text (or ASCII) file, and type in an appropriate file name. After the program is ready to receive the file, press ENTER to make the Boss32 display the program; the program will capture it and store it in the selected file on the personal computer. After the entire BASIC program has been displayed, execute the command in the communications program to finish the reception of the file.

Loading and Saving Programs With FLASH EPROM

The Boss32 includes an onboard FLASH EPROM programmer that is used to save the user's programs. When a FLASH EPROM is erased, the entire contents of the FLASH EPROM are lost; there is no way to only erase part of the FLASH EPROM. The same file can be stored multiple times, allowing different versions to be saved while developing a program.

The programmer supports different sizes of FLASH EPROM. The DIR command will display the contents of the EPROM and the amount of unused space remaining. Two types of files can be stored on the EPROM: source code and compiled code. The source code is the human readable BASIC program; this should be stored so that it isn't lost. Compiled code is the executable code generated by the Bear BASIC compiler; this is stored on an EPROM so that it can be automatically executed on power up, or so that it can be CHAINed to from another program. Each type of file is numbered sequentially on the EPROM, starting at 1.

After a BASIC program (the source code) has been typed in, it should be saved before compiling it and attempting to execute it. This is so that the program won't be lost if the Boss32 crashes when the program is run. When entering a long program, it is also advisable to save the program periodically so that the entire program isn't lost in the event of a power failure. To save the source code to EPROM, type **SAVE** *progname*, where *progname* is the name of the program. The program name will be stored as the file name on the FLASH EPROM; if *progname* isn't entered, then it will default to all spaces.

After a program has been successfully compiled (i.e. no syntax errors were encountered by the compiler), then the compiled code can be saved to the EPROM, by typing **SAVE CODE** *progname*, where *progname* is the name of the program. The program name will be stored as the file name on the Flash EPROM; if *progname* isn't entered, then it will default to all spaces.

Automatically Executing a Program on Power Up

When the user's program is operational and the Boss32 is to be installed, it will probably be necessary for the Boss32 to automatically execute the program when it is turned on. If the SETOPTION RUN command is set to BASIC (run mode), then the Boss32 will load the last compiled code file from the Flash EPROM and execute it. Since the Boss32 executes the last compiled code file from the Flash EPROM, the latest revision of a program will always be executed. If the SETOPTION RUN command is set to OFF (program mode), then the Boss32 will respond with the compiler prompt when it is turned on.

Programming Statements

‘	Comment text
!	Comment text that is not kept in source listing
?	Identical to PRINT
CALL	Call an assembly language subroutine
CANCEL	Halt the rescheduling of a task
CHAIN	Load and run another program from EPROM
CLS	Erase the current FILE display device
CNTRMODE	Set operating mode for counter
DAC	Sets D/A output level
DATA	Store data for READ to access
DEBUG	Halt compile process to display variables
DEF	Mark beginning of a user defined function
DOUT	Control digital output channel
EEPOKE	Writes to a location in the EEPROM
EXIT	Abort execution of current task
EXMOD	Used to communicate with intelligent expander modules
FILE	Set current file I/O device and serial port settings
FINPUT	Formatted INPUT from current FILE
FNEND	Mark end of a user defined function
FOR	Beginning of a program loop; see NEXT
FPRINT	Formatted PRINT to current FILE
FUZZY	Fuzzy logic control
GETDATE	Get current date from real time clock
GETIME	Get current time from real time clock
GOSUB	Call a subroutine
GOTO	Jump to a line number
IF expr THEN...[:ELSE:...]	Conditional execution of rest of line
INPUT	Wait for user input from current FILE
INPUT$	Allow commas in user input
INTEGER	Declare signed integer variables
INTERRUPT	Attach a task to a hardware interrupt source
INTOFF	Disable all interrupt processing
INTON	Enable all interrupt processing
LINE	Draws a Line or Box on the Boss32 Graphics Display
LOCATE	Set cursor position for current FILE device
LOCK	User defined task resource locking
NETWORK 0	Initialize Boss32 network handler
NETWORK 3	Set the value of a network register
NETWORK 4	Read the value of a network register
NETWORK 10	Initializes network handler for Modbus
NETWORK 11	Used to access other ,Modbus units

NETWORK 12	Used to set status of device on Modbus network
NETSERVER	Run as a network server
NEXT	End of a program loop; see FOR
ON ERROR	GOTO linenum if an error occurs
ON expr, GOSUB	Call subroutine based on value of expr
ON expr, GOTO	Jump to linenum based on value of expr
POKE	Store byte at addr
POLY	Draws a Poly object on the Boss32 Graphics Display
PRINT	Send output to current FILE device
PRIORITY	Set priority of current task
PSET	Sets a Pixel to On/Off on the Boss32 Graphics Display
RANDOMIZE	Re-seed the random number generator
RDCNTR	Read counter value into varname
READ	Read values from DATA statements
REAL	Declare floating-point variables
REM	Comment text
RESTORE	Reset READ pointer to first DATA statement
RETURN	Continue at line following GOSUB
RUN	Begin task execution; set resched. interval
SERIALDIR	Sets the RTS and DTR level for the serial ports
SETDATE	Set current date of real time clock
SETIME	Set current time of real time clock
STOP	Halt execution of program or a task
STRING	Declare text string variables
TASK	Mark the beginning of a task area
TIMER	Controls the 221 timers
TMADDR	Sets the ID Number for Touch Memory
TMREAD	Reads Data from a Touch Memory Device
TMSEARCH	Reads the ID's for all connected Touch Memory Devices
TMWRITE	Writes Data to a Touch Memory Device
TRACEON	Enable line number trace on the FILE set by SETOPTION TRACE
TRACEOFF	Disable line number trace on the FILE set by SETOPTION TRACE
WAIT	Suspend task execution for X tics
WPOKE	Store integer value at a memory address
WRCNTR	Write to counter register(s)
ZPRINT	Prints a zoomed character on the Boss32 Graphics Display

System and Direct Commands

BYE	Reset/Reboot the Boss32
C	Abbreviation for COMPILE
CLEARFLASH	Eases the FLASH EPROM where the user programs are stored.
CLEARMEMORY	Clears and fills all RAM locations with zeros.
CLR	Erase the console display
COMPILE	Compile the BASIC program
DELETE [CODE]	Delete a file from the FLASH EPROM
DIR	Display a listing of files on the EPROM
DIRDEL	Deletes a program from the flash directory

DOWNLOAD	Disable echo while loading a program
E	Abbreviation for EDIT
EDIT	Enter the line editor to alter linenum
END	Reactivates echoing to the terminal
ERROR	Enable error checking in compiled BASIC
G	Abbreviation for GO
GO	Start execution of the compiled program
HELP	Display help text
L	Abbreviation for LIST
LIST	List all or part of the program
LFDELAY	Delay at the end of each line displayed
LOAD [CODE]	Load source or code from the EPROM
MEMDUMP	Do a memory dump starting at addr
NEW	Clear out the current BASIC program
NOERR	Disable error checking in compiled BASIC
NOLINE	Put in no line numbering mode
R	Abbreviation for RUN
RUN	Compile and execute the current BASIC program
SAVE [CODE]	Save source or compiled code to the EPROM
SETOPTION DAC	Sets the initial power up value for the DAC output
SETOPTION LFDELAY	Sets the initial power-up line feed delay
SETOPTION NOLINE	Sets the initial power-up defaule for noline number mode
SETOPTION TERMINAL	Sets the initial power up terminal emulation for COM 1
SETOPTION RESET	Sets the flag to reset all outputs at program termination
SETOPTION RUN	Sets the initial power-up flag to run or edit
SETOPTION SRL	Sets the initial power-up baud rate for COM 1
SETOPTION TRACE	Sets the file # to output trace line numbers on
SETOPTION TIC	Sets the context switcher multiplier (1,2,3 or 4 x 2.5 msec)
SETOPTION VFD	Sets the Intensity of the VFD Display on Power up
STAT	Display memory usage, hardware avaliable, and compiler version
SYSADC	Displays the reading of an ADC channel
SYSDAC	Displays the reading of a DAC channel
SYSDATE	Displays the date set in the real time clock
SYSDIN	Read digital input channel; 0 or 1
SYSDOUT	Control digital output channel
SYSSETDATE	Sets the date in the real time clock
SYSSETTIME	Sets the time in the real time clock
SYSTIME	Displays the time set in the real time clock
S19DOWNLOAD	Enables the Boss32 to recieve an off board compiled program in the S19 file format
S19GO	Runs a downloaded S19 program
S19UPLOAD	Enables the Boss32 to upload a compiled basic program in S19 format

Operators

*	Multiply
/	Divide
+	Add
-	Subtract
AND	Logical AND

OR		Logical OR
>=		Greater than or equal to
<=		Less than or equal to
=		Equality
<>		Inequality
>		Greater than
<		Less than
=		Assignment operator
Operator precedence:	highest	* /
		+ -
		> < <> >= <= unary-
	lowest	AND OR

Functions

ABS	Calculates the absolute value of an expression
ACOS	Arccosine of an expression
ADC	A/D value for chan
ADR	Address of specified variable
ASC	ASCII equivalent of first character in a string expression
ASIN	Arcsine of an expression
ATAN	Arctangent of expression
AUTOCNTR	Read the count from the auto reader
BAND	Bitwise logical AND of expressions
BOR	Bitwise logical OR of expressions
BXOR	Bitwise logical XOR of expressions
CHR$	One character equivalent of its ASCII expression
CONCAT$	Appends one string after another
COS	Cosine of expression
CVI	Converts binary string to integer
CVS	Converts binary string to real
DIN	Read digital input channel
EEPEEK	Reads the contents of a location in the EEPROM
ERR	Last error number generated
EXP	Exponential function e**expr
GET	Waits for one byte from current FILE
HEXVAL	Converts a Hex String to a Number
KEY	Reads one byte from FILE, doesn't wait
LEN	Returns length of a string expression
LOG	Natural logarithm (base e) of expr
LOG10	Common logarithm (base 10) of expr
MID$	Substring of string expression
MKI$	Converts an integer to a binary string
MKS$	Converts a real to a binary string
MODFUNCTION	Returns last Modbus function preformed
MODREGISTER	Returns the first register number of the last Modbus funtion performed
NETMSG	Sends and receives messages over the network
PEEK	Read a byte from an address
PID	Utilized to perform calculations required for PID closed-loop control

PIDSETUP	Used to setup user-defined PID loop
RND	Generate a pseudo-random number
SERIALSTAT	Reads the level of CTS, DSR, DCD, RI on the serial ports
SIN	Sine of an expression
SQR	Square root of an expression
STR$	Converts a number to a string
TAN	Tangent of an expression
VAL	converts a string to a number
WPEEK	Read an integer value from an address

Accessing Help

The Boss32 has a wealth of on line help available for programming statements, direct and sytem commands, operators and functions. The help screens gives arguments and descriptions with each statement or command. To access on line help, type **HELP** and return at the command prompt.

BOSS32 Compiler Version X.XX
>HELP

or

>HELP [command]

Chapter Three
System Information

The Boss32 is a verstatile multi-function controller for any control package. The Boss32 with its system and add on devices can tackle the toughest control needs. This chapter describes the Boss32 system from part numbering through additional Boss32 options and specifications.

Determing Part Numbers

The Boss32 is configurable with multiple options. To determine which Boss32 model you have, see the part number description chart below.

Typical Part No. **BOSS32B** **- GS -** **L -** **0**

Case & Display Style

BP Subplate Mount Unit
GS 8x42 Graphics (64 x256) Vacuum Fluorescent Display with 24 Mechanical Keys
VS 2x40 Vacuum Fluorescent Display with 24 Mechanical Keys

System Peripherals

G 4 Serial Ports, Real Time Clock, 200K Program RAM, 256K Flash Program Storage, 64K Battery Backed RAM, 8K EEPROM Setpoint Storage and I/O Interface.

H 4 Serial Ports, Real Time Clock, 200K Program RAM, 256K Flash Program Storage, 64K Battery Backed RAM, 8K EEPROM Setpoint Storage, I/O Interface and High Speed Counter.

I 4 Serial Ports, Real Time Clock, 200K Program RAM, 256K Flash Program Storage, 64K Battery Backed RAM, 8K EEPROM Setpoint Storage, I/O Interface, 8 Analog Inputs rated 0-5VDC and 4 Analog Outputs rated 0-10VDC.

J 4 Serial Ports, Real Time Clock, 200K Program RAM, 256K Flash Program Storage, 64K Battery Backed RAM, 8K EEPROM Setpoint Storage, I/O Interface and 8 Analog Inputs rated 0-5VDC

K 4 Serial Ports, Real Time Clock, 200K Program RAM, 256K Flash Program Storage, 64K Battery Backed RAM, 8K EEPROM Setpoint Storage, I/O Interface and 4 Analog Outputs rated 0-10VDC.

L 4 Serial Ports, Real Time Clock, 200K Program RAM, 256K Flash Program Storage, 64K Battery Backed RAM, 8K EEPROM Setpoint Storage, I/O Interface, High Speed Counter, 8 Analog Inputs rated 0-5VDC and 4 Analog Outputs rated 0-10VDC.

M 4 Serial Ports, Real Time Clock, 200K Program RAM, 256K Flash Program Storage, 64K Battery Backed RAM, 8K EEPROM Setpoint Storage, I/O Interface, 8 Analog Inputs rated 0-20mADC and 4 Analog Outputs rated 0-20mADC.

N 4 Serial Ports, Real Time Clock, 200K Program RAM, 256K Flash Program Storage, 64K Battery Backed RAM, 8K EEPROM Setpoint Storage, I/O Interface and 4 Analog Outputs rated 0-20mADC.

O 4 Serial Ports, Real Time Clock, 200K Program RAM, 256K Flash Program Storage, 64K Battery Backed RAM, 8K EEPROM Setpoint Storage, I/O Interface and 8 Analog Inputs rated 0-20mADC.

P 4 Serial Ports, Real Time Clock, 200K Program RAM, 256K Flash Program Storage, 64K Battery Backed RAM, 8K EEPROM Setpoint Storage, I/O Interface, High Speed Counter, 8 Analog Inputs rated 0-20mADC and 4 Analog Outputs rated 0-20mADC.

Options

0 No Options
1 256K Batter Backed RAM
2 1 Meg Batter Backed RAM
3 1 Meg Program Memory
4 512 K Flash
5 32K EEPROM

Using “STAT” for System Information

The Boss32 current hardware and software status can be accessed by typing **STAT** at the command prompt followed by a return. The Boss32 will respond with the following screen:

```
> STAT

   Code Start:  XXXXXXXX          Code End:  XXXXXXXX
Variable Start:  XXXXXXXX       Variable End:  XXXXXXXX

    XXXK Variable Bytes Used          XXXk Variable Bytes Free
    XXXK Temporary & Constant Bytes Used
    XXXK Source Bytes Used            XXXK Code Bytes Free
    XXXK Runtime Bytes Used
    XXXK FLASH EPROM Bytes Used       XXXK FLASH EPROM Bytes Free

                    press the space bar to continue
```

By pressing the space bar, the second status screen will be shown.

```
Serial baud rate set to:                        9600
No Line Numbering is                            OFF
Delay Carriage Return Line Feed is              OFF
Terminal Emulation is                           ADM5
RUN on power up is                              OFF
VFD LEVEL is set to                             1
RESET outputs on program termination is         OFF
TRACEON file number is set to                   0
                                                A/D                PRESENT
                                                D/A                PRESENT
                                                COUNTER            PRESENT
                                                REAL TIME CLOCK    PRESENT
                                                DISPLAY            8x42 Graphics VFD
                                                EEPROM             8K

EXPANSION MODULES:          J3 -                                   OPEN
                            J4 -                                   OPEN
                            J5 -                                   OPEN

BOSS32 BASIC Compiler Version X.XX
```

Status Memory Descriptions

Code Start: XXXXXXXX	Starting memory location of the program.
Code End: XXXXXXXX	Ending memory location of the program
Variable Start: XXXXXXXX	Starting memory location of the program variables.
Variable End: XXXXXXXX	Ending memory location of the program variables.
XXXK Variable Bytes Used	Number of bytes used for program variables
XXXk Variable Bytes Free	Number of bytes available for program variables
XXXK Temporary & Constant Bytes Used	

XXXK Source Bytes Used	Number of bytes used by program source code.
XXXK Code Bytes Free	Number of bytes available for use.
XXXK Runtime Bytes Used	Number of Runtime bytes used by program
XXXK FLASH EPROM Bytes	Number of bytes used on FLASH EPROM Used
XXXK FLASH EPROM Bytes Free	Number of bytes available on FLASH EPROM

Status Option Descriptions

Serial baud rate set to:	Current baud rate.
No Line Numbering is	ON, OFF: No Line Numbering mode
Delay Carriage Return Line Feed is	ON, OFF: Delay Carriage Return Line Feed for slowing transmission to serial ports.
Terminal Emulation is	Current Terminal Emulation, ADM 5, VT100, etc.
RUN on power up is	BASIC, OFF: Run program on power up
VFD LEVEL is set to	1-4: Power Level Display is set to.
RESET outputs on program Termination is	ON, OFF: Turn off all outputs when program stops executing.
TRACEON file number is	set to Default file for TRACEON printing. 0, 5, 6
A/D	PRESENT, ABSENT: On board Analog Inputs installed.
D/A	PRESENT, ABSENT: On board Analog Outputs installed.
Counter	PRESENT, ABSENT: On board counters installed.
Real Time Clock	PRESENT, ABSENT: Real Time Clock installed.
DISPLAY	8x42 Graphics VFD, 2x40 VFD: Type of installed display.
EEPROM	PRESENT, ABSENT: EEPROM Size
EXPANSION MODULES: J3 -J5	Type of Expansion module installed. No module returns OPEN

Dimensions and Mounting

The Boss32 is can be orded either subplate mount (without display) or panel mount (with display). Both are encased by rugged painted steel enclosure. The Boss32 size is approximately (model dependent) 12” X 8” X 2.75”.

CPU

The Boss32 processor is a Zilog Z380. The Zilog Z380 is a full 32 bit processor and is running at 14.7456 Mhz.

RAM

The Boss32 comes standard with 256K of program RAM and can be ordered with up to 1Meg of RAM. The memory holds the users program source code and the compiled code to run. The constant and variable data while the user’s program is executing is stored in a separate 64K non-volatile memory section that can be expanded up to 1 Meg. The memory

contents are held by a internal lithium battery which should provide years of problem free operation.

Program Storage

The Boss32 stores programs on FLASH EPROM. The FLASH EPROM gives the programmer the abiltiy to store both source and compiled code. Multiple programs can be stored, deleted and accessed as needed. The FLASH EPROM is accessed using DIR, LOAD, LOADCODE, SAVE, SAVECODE, and CHAIN.

The Boss32 comes standard with 256K FLASH EPROM space. The Boss32 can be ordered with up to 1Meg FLASH EPROM.

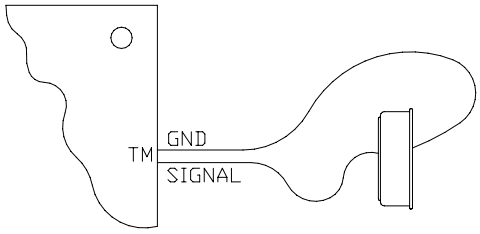
Real Time Clock

The Real Time Clock maintains the current time and returns it as year, month, day, hour, minute, second, day of week, and day of month. It has battery back-up, and will keep the correct time even when the Boss32 is not powered. The Boss32 Real Time Clock is Year 2000 compliant.

The real time clock allows the Bear BASIC program to determine the time and date at any point. The clock is accessed using GETDATE, SETDATE, GETIME, SETIME, SYSSETTIME, SYSSETDATE, SYSTIME, and SYSDATE.

Touch Memory

The Boss32 comes with a Touch Memory Port. Compatable with the Dallas Touch Memory devices (Time in a Can), external memory may be used as identification, storage for setpoints or many other items.



Touch Memory Connections

Chapter Four
BOSS32 Power

The Boss32 boasts a versatile power supply system. The Boss32 can be powered with 10VAC, 12VDC, or 120VAC(via tranformer). I/O and additonal devices may be connected without adding external power supplies. The Boss32 provides a 12VDC unregulated and 5VDC regulated output supply for use with I/O or additonal devices.

Input Power

120VAC may be used in conjuction with a transformer. Tranformer should be rated 2A with 120VAC input, 10VAC output. See Appendix D for Current Requirements.

Divelbiss Corporation recommends fusing the input power with a “fast” type fuse. Filtering of the incoming power is also recommended, using a line filter; Divelbiss Part No. ICM-LF-01 or equivalent.

On-Board Power Supplies

The Boss32 provides on board power supplies for driving off unit devices. The Boss32 power may be used to power sensors, prox switches, encoders and the Divelbiss standard I/O cards. The +VA output power is a non-regulated supply @ approximately 12VDC; while the 5VDC is a regulated supply. Each I/O card attatched to the Boss32 will use power from these supplies.

Chapter Five
Working With Serial Ports

Interfacing into other devices is a requirement in todays control applications. The Boss32 has the ability to connect to other "smart" devices via serial ports. The Boss32 is available with 4 serial ports and supports baud rates from 2400 baud to 57.6K baud. The Boss32 supports RS232, RS422 or RS485. The Boss32 supports ASCII transfer and can be a modbus master or slave.

Many devices interface with other equipment using serial data transfer. The Boss32 provides Four serial ports. All ports support asynchronous serial transfer at baud rates between 300 and 57600 baud. To set the operating mode for any COM port, the FILE statement is used. See the **FILE** and **SERIALDIR** statements for information.

COM1

COM1 supports RS-232 levels. While at the command line prompt, it is used to attach the console terminal, which is the main programmer's interface to the Boss32. On power up, it is set to 9600 baud, no parity, 8 data bits, and 1 stop bit. At runtime, **COM1** is accessed as FILE 0; it may be used as a general purpose serial port. The **COM1** connector is a 9 pin male D connector.

Pin	ID	Description
1	DCD	Data Carrier Detect
2	RX	Receive data
3	TX	Transmit data
4	DTR	Data Terminal Ready
5	GND	Signal ground
6	DSR	Data Set Ready
7	RTS	Request To Send (Output)
8	CTS	Clear To Send (Input)
9	--	Optional +VA Pin

Divelbiss can supply the following cables to connect the Boss32 COM1 port to a personal computer:

Part No.	Description
ICM-CA-28	9 pin female D to male 25 pin D, 6 ft long
ICM-CA-29	9 pin female D to female 25 pin D, 6 ft long
ICM-CA-34	9 pin female D to female 9 pin D, null modem

COM2

COM2 supports RS-232 and is identical to COM1. It is unused while at the command line prompt. At runtime, it may be accessed as FILE 1 as a general purpose serial port. On power up, it is set to 9600 baud, no parity, 8 data bits, and 1 stop bit. The COM2 pinout is identical to COM1.

COM3

COM3 supports RS-232, RS422 and RS485. It is unused while at the command line prompt. At runtime, it may be accessed as FILE 4 as a general purpose serial port. On power up, it is set to 9600 baud, no parity, 8 data bits, and 1 stop bit. The operating mode is selected by software.

RS-422	Pin	ID	Description
	1	TX-	Transmit data (-)
	2	—-	No connect
	3	—-	No connect

	4	RX-	Receive data (-)
	5	GND	Signal ground
	6	RX+	Receive data (+)
	7	—-	Jumper to 8
	8	—-	Jumper to 7
	9	TX+	Transmit data (+)

RS-232	**Pin**	**ID**	**Description**
	1	—-	No connect
	2	RX	Receive data
	3	TX	Transmit data
	4	—-	No connect
	5	GND	Signal ground
	6	—-	No connect
	7	—-	No connect
	8	—-	No connect
	9	—-	No connect

RS-485	**Pin**	**ID**	**Description**
	1	TX-	Data (-)
	2	—-	No connect
	3	—-	No connect
	4	—-	No connect
	5	GND	Signal ground
	6	—-	No connect
	7	—-	Jumper to 8
	8	—-	Jumper to 7
	9	TX+	Data (+)

COM4

COM4 supports RS-232, RS422 and RS485. COM4 is identical to COM3. It is unused while at the command line prompt. At runtime, it may be accessed as FILE 5 as a general purpose serial port. On power up, it is set to 9600 baud, no parity, 8 data bits, and 1 stop bit. The COM4 pinout is identical to COM3.

Note: Using CTRL-C or Chain Command resets com ports to default settings.

Using the Serial Ports

For applications which require a more complex user interface, a remote display, or remote data entry, a terminal may be attatched to any serial port (COM1 - COM4). If multiple data entry stations are required, terminals could even be installed on more than one serial port. All of the file I/O statements and functions work with the serial ports: COM1 is FILE 0, COM2 is FILE 1, COM3 is FILE 4 and COM4 is FILE 5. When a program starts executing, the default output device is FILE 0; each time that a task is RUN, that task's default device is FILE 0. Any of the file I/O statements and functions may be used to access the serial ports: CLS, FPRINT, LOCATE, PRINT, FINPUT, GET, INPUT, INPUT$ AND KEY.

It is important to remember that the characters are displayed relatively slowly when using a terminal: 1 millisecond per character at 9600 baud. This means that a long PRINT statement may take 80 msec (or more) to complete. Device resource locking is performed on each I/O file individually. In a multitasking program, other tasks will continue to run while the PRINT statement is taking place, but no other task will be able to PRINT to the same port until the first PRINT finishes. Unfortunately, another task could change the cursor position of the terminal between PRINT statements. An example will demonstrate this problem and simple solution.

```
100     INTEGER J: J=0
110     CLS
120     RUN 1,39: WAIT 30: RUN 2,60
130     J=J+1: WAIT 9: GOTO 130
200     TASK 1
220     LOCATE 10,10
230     FPRINT “S16U5Z”, “Task 1 value is “, J
250     EXIT
300     TASK 2
320     LOCATE 15,40
330     FPRINT “S19U5Z”, “Value in task 2 is “m, J
350     EXIT
```

The desired result of the program is to display the count value (J) at two places on the screen using two tasks. When the program is run, however; it will periodically display one of the strings in the wrong location. This happens whenever the context switcher (see multitasking in the 32 Bit Software Manual) causes task 2 to execute immediately after task 1 has executed the LOCATE statement. Task 2 will perform another LOCATE, then print its message at the correct location. When Task 1 runs again, it will print its message at the location following the message printed by task 2. This same situation will occur in the opposite order, causing task 2’s message to be printed at the location following that of task 1’s. The solution to this is to use a variable as a flag to prohibit another task from moving the cursor while a task is accessing the termina (or use the LOCK statement). The variable PF has been added to the example (see line 100); it will be nonzero when FILE 0 is in use. Lines 210, 215, 240, 310, 340 have been added to the previous example. When a task is ready to output to the terminal, it increments PF; interrupts must be disabled while updating PF (with INTOFFand INTON), to ensure that two tasks don’t conflict. If PF is nonzero, then the other task is already using the terminal, so the task just waits and tries again later. When PF is zero, it indicates that the other task is finished, so this task continues. See the 32 Bit Software manual for details on multitasking and the LOCK statement.

```
100     INTEGER J, PF: J=0:PF=-1
110     CLS
120     RUN 1,39: WAIT 30: RUN 2,60
130     J=J+1: WAIT 9: GOTO 130
200     TASK 1
210     INTOFF: PF=PF+1:INTON
215     IF PF THEN WAIT 1:GOTO 215
220     LOCATE 10,10
230     FPRINT “S16U5Z”, “Task 1 value is “, J
240     INTOFF: PF=PF-1:INTON
250     EXIT
```

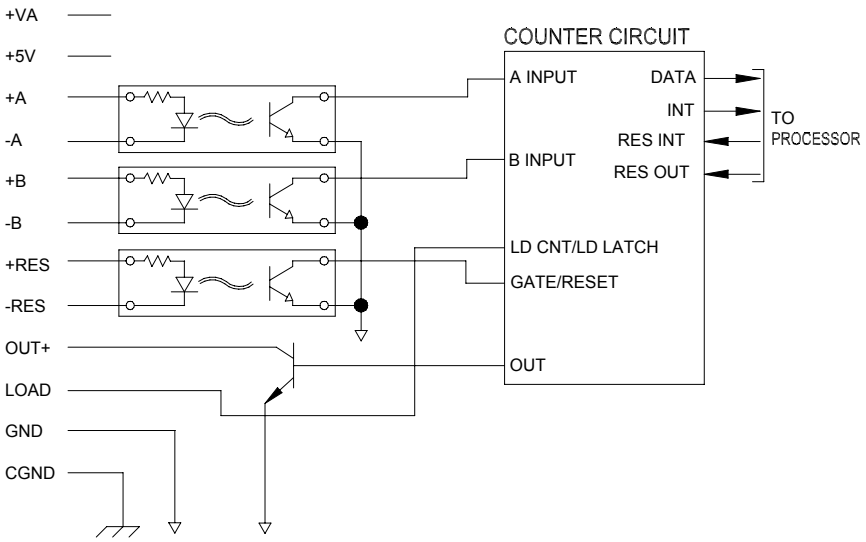
```
300     TASK 2
310     INTOFF: PF=PF+1
315     IF PF THEN WAIT 1: GOTO 315
320     LOCATE 15,40
330     FPRINT “S19U5Z”, “Value in task 2 is “m, J
340     INTOFF: PF=PF-1: INTON
350     EXIT
```

Chapter Six
High Speed Counter

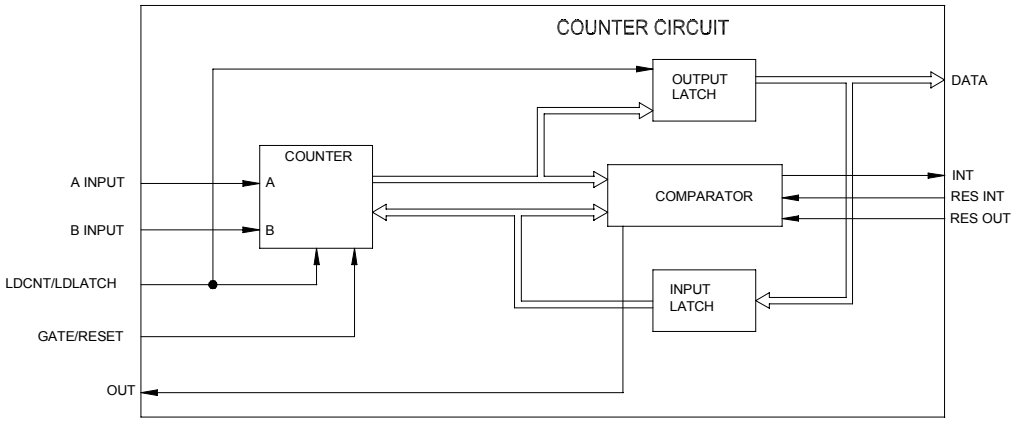
Counting is a very common operation in real time control systems. For low speed signals (less than 10 pulses/second), this can be handled easily in software. At higher rates, however, hardware counters are required. The Boss32 high speed counter circuit is very flexible, providing several operating modes. It is a 24 bit binary up/down counter, capable of greater than 75 kHz count rates (18 kHz in quadrature modes), with a high speed output. It supports quadrature mode, which allows it to operate with biphase shaft encoders.

Counter Specifications

In order to understand the operation of the counter, it is helpful to look at the hardware logic.



High Speed Counter logic



Internal counter logic

Counter Input Channel Modes

1. The **A** and **B** inputs are used in quadrature X1 mode, for use with biphase encoders.The count value changes once for each biphase cycle.

2. The **A** and **B** inputs are used in quadrature X4 mode, for use with biphase encoders. The count value changes with each input transition; X4 mode counts four times faster than X1 mode, given the same input signal.

3. A pulse on input **A** causes the count to increase, while a pulse on **B** causes the count to decrease.

4. The **B** input sets the count direction: high for increment, low for decrement. A falling edge on the **A** input causes the counter to count in the selected direction.

5. X1 quadrature mode. RES input will enable/disbale counter.

6. X4 quadrature mode. RES input will enable/disable counter.

7. A pulse on input **A** causes the count to increase, while a pulse on **B** causes the count to decrease. RES input will enable/disable counter.

8. Counter input **B** sets direction of counting (increase or decrease), and a pulse on input **A** causes the counter to count by 1. RES input will enable/disable counter.

If the counter is to be used in unidirectional mode, it should be put into mode 4. The count signal would be connected to the **A** input, and the **B** input would control the direction.

The **RES** input can be used to reset the counter. **RES** is active high. A pulse on **RES** will cause the counter hold its current value (disregarding **A** and **B**) or reset to 0, depending upon how the RDCNTR statement is used in the program.

A pulse on **LOAD** will cause the counter value to be preset from the input latch or written to the output latch, depending upon how the RDCNTR statement is used in the program. Both **RES** and **LOAD** are separate inputs allowing full access to the counter control signals. In this configuration, a single input signal can be wired to latch the current counter value into the output latch, and then reset the counter value to 0.

When the counter value matches the value stored in the input latch, the comparitor will assert the high speed output and also send an interrupt signal to the processor. These signals stay active until the processor resets each of them with the appropriate control line (RES_OUT and RES_INT). RDCNTR can reset the high speed output. The output circuit is open-collector.

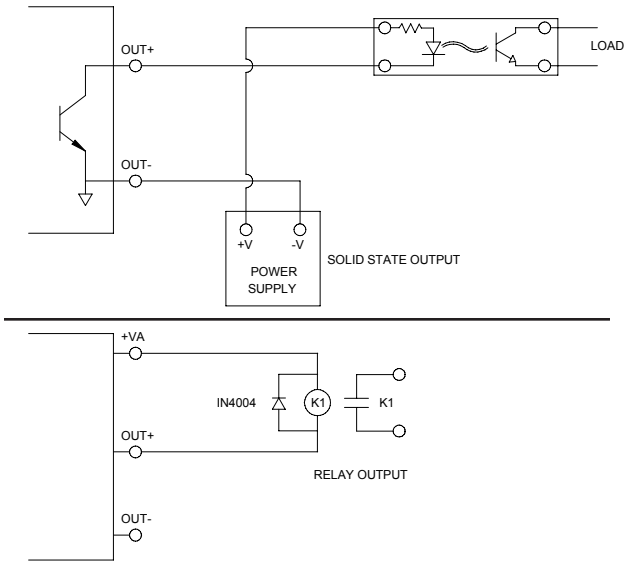
The counter inputs **A**, **B**, and **RES** are designed to be used with differential sensor outputs. Contact Divelbiss for encoder recommendations.

NOTE: The maximum voltage to be applied to any counter input is 5VDC.

High Speed Output

The High Speed Output is an open collector output available from each counter. This output can trigger another device and is not dependant upon the processor for control. The High Speed Output is driven and controlled by the on board counter. The maximum current ofr the High Speed Output is 100mA DC.

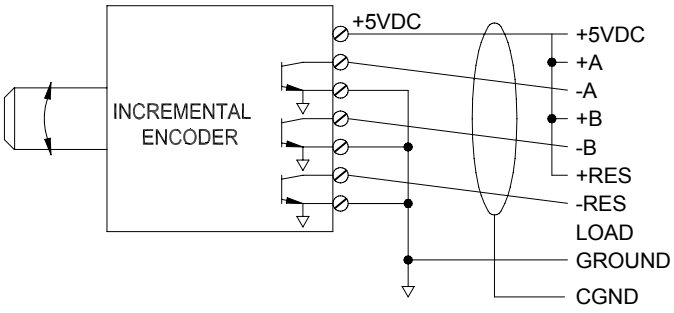
High Speed Output to optically isolated device.



High Speed Output controlling a dry contact.

Connecting to an Encoder

For best noise immunity, shielded cable should be used between the counter inputs and the device being monitored; this is especially important for long cable runs. To minimize any potential crosstalk problems, it is recommended that individually paired and shielded cables be used for each I/O device, especially at high counting speeds. Use the shield terminal provided or connect the shield to earth ground by other means. Do not connect the cable shield at both ends of the cable, as this can have an adverse effect. Always use separate returns (minus terminal) for each signal pair, as this lessens the chance of ground loops occurring by making all common terminations at the on-board counter.



Biphase incremental encoder wiring example

Counter Examples

The simplest example of using the counter just reads the current count value and displays it to the terminal via COM1. To try this example, wire a pushbutton between **+5V** and **+A**, then connect the **-A** input to **GND**. Wire a pushbutton between **+5V** and **+B**, the connect **-B** input to **GND**. Each time button A is pushed, the counter will increment one or more times; the button will probably bounce when pressed, causing up to 20 pulses.

```
100  INTEGER COUNT
110  CNTRMODE 1,3                ' A counts up, B counts down
120  WRCNTR 1,0,10               ' Set counter value to 10
140  RDCNTR 1,0,COUNT            ' Read the current value
150  FPRINT "U6Z",COUNT          ' Now display it
160  GOTO 140
```

The next example shows how to handle counter interrupts and use the high speed output. This example sets up task 1 as the interrupt handler for counter 1. In lines 120 and 130 it initializes counter 1, sets its mode to A-count, B-direction and writes a 0 to the counter. In lines 140 and 150 it sets the counter reload variable to 10 and writes this reload value into the counter's compare register. Line 160 uses INTERRUPT to set task 1 as the interrupt handler for the counter. Lines 210 to 240 form the program's main loop, which just displays the latest counter value (COUNT) from the last interrupt; it also increments and displays J, just to cause some action on the display. Lines 300 to 350 form task 1, the interrupt handler; it reads the current counter value and updates the reload value. Lines 400 to 440 form task 2, which turns off the high speed output about ½ second after it is turned on.

In task 1, the first thing that it does is read the current counter value. At low pulse rates, this will be the same as RELOAD, since it is reading the same value that caused the interrupt. As the pulse rate increases, however, the counter will increment before the task 1 gets to read the counter. At a very high pulse rate, a problem will occur when the counter has already gone beyond the new RELOAD value before RELOAD is written to the counter (ie. COUNT=783 and RELOAD=780); this effectively stops the interrupt, since the counter will need to wrap completely around before the interrupt will occur again.

```
100  ' Program to demonstrate the  high speed counter interrupt
110  INTEGER J, RELOAD, COUNT, T2TEMP
120  CNTRMODE 1,4                ' A count, B direction
130  WRCNTR 1,0,0                ' Set count to 0
140  RELOAD=10: COUNT=0
150  WRCNTR 1,1,RELOAD           ' Set counter interrupt value
160  INTERRUPT 1,1,1             ' Set counter interrupt to task 1
170  J=0
200  ' Main program loop.
210  LOCATE 1,1
220  FPRINT "U5X5U5Z", J, COUNT
230  J=J+1                       ' Increment junk variable
240  GOTO 210
300  ' Counter interrupt handler task.
310  TASK 1
320  RDCNTR 1,0,COUNT            ' Get new count
330  RELOAD=RELOAD+10            ' Set up new reload
```

```
340  WRCNTR 1,1,RELOAD              ‘ Set counter interrupt value
350  RUN 2: EXIT                    ‘ Set up task 2 to turn output off.
400  ‘ Task to turn off high speed output
410  TASK 2
420  WAIT 50                        ‘ Wait ½ second, then turn
430  RDCNTR 1,3,T2TEMP              ‘ the output off
440  CANCEL 2: EXIT                 ‘ Don’t reschedule this task.
```

Chapter Seven
Built-in Displays

The Boss32 provides a powerful and easy to use operator interface and display. From printing simple text to drawing graphic representations, the Boss 32 provides the answer. The Boss32 comes standard in three display configurations.

1. No Display - Subplate Mount
2. 2x40 VFD - 2x40 Vacuum Fluorescent Display, Panel Mounted, NEMA 4X
3. 8x42 VFD Graphics - 8x42 Vacuum Fluorescnet Display, capable of Graphics & text, Panel Mounted, NEMA 4X.

Displaying Text with the Built-In Display

The display is accessed as FILE 6. Any of the file output statements can be used: CLS, FPRINT, LOCATE and PRINT. Each task maintains its own cursor position, so multiple tasks can write to the display concurrently without corrupting each others information. The following example demonstrates the use of the display. It starts by scrolling a message onto the top line of the display a counter value and the time of day on the second line.

```
100     INTEGER J,H,M,S
110     STRING A$(50)
200     A$="Divelbiss Boss 32 Controller     "
210     FILE 6                                  ! Set task 0 to onboard display
220     CLS                                     ! Clear the display
290     'Scroll a message onto the display
300     FOR J=1 TO LEN(A$)
310     LOCATE 1,41 - J                         ! Position the Cursor
320     PRINT MID$(A$,1,J)
330     WAIT 10
340     NEXT J
350     LOCATE 2,1: PRINT "Count:";
360     LOCATE 2,20: PRINT "Time:";
370     RUN 1,100                               ! Start Task 1, reschedule 1/second
380     J=0
390     LOCATE 2,8                              ! Position the Cursor
400     FPRINT "U5Z",J                          ! Display the Current Count Value
410     WAIT 15: J=J + 1: GOTO 390              ! Update Count & Loop Forever
500     TASK 1
510     FILE 6                                  ! Set task 1 to onboard display
520     GETIME H,M,S                            ! Get time from Real Time Clock
530     LOCATE 2,26                             ! Position the Cursor
540     PRINT H;":";                            ! Print H followed by a colon
550     if M < 10 THEN PRINT "0";               ! Print leading zero if necessary
560     PRINT M;":";                            ! Print minute followed by colon
570     IF S < 10 THEN PRINT "0";               ! Print leading zero if necessary
580     PRINT S;" ";                            ! Print Second
590     EXIT                                    ! Finished, Leave task for now
```

The FILE 6 statements are necessary to cause the output to go to the onboard display; BEAR BASIC will default to FILE 0, the COM1 Serial Port. Each task must have its own FILE 6 statement, because each task maintains its own current file number.

When writing to the display in a multitasking program, it is important to avoid corrupting a portion of the display that another task may be using. In fact, it is often easier to handle all display operations in one task, just to simplify the program debugging.

The display is a relatively slow device, by computer standards. When a PRINT or FPRINT statement is executed, it is very likely that other tasks will get to execute before the statement returns.

Displaying Graphics with the Built-In Graphics Display

One of the special features of the Boss32 is the optional Graphics Vacuum Fluorescent Display (GVFD). Displaying Text on this display is exactly the same as the 2x40 VF Display with the exception, there are more lines and columns. Displaying Graphics, is different in that the graphics statements are only used on a Boss32 with a GVFD installed. This means that no matter what the FILE number is set to, the graphics statements will always display to the GVFD.

Any of the graphics output statements may be used (LINE, PSET and POLY) to display graphics on the GVFD. The FILE 6,"[*n"*] statement allows the mode of the graphics to be set on the GVFD. There are three different modes of operation for the intermingling of text and graphics on the GVFD: "O" = OR, 1 = XOR, 2 = AND. The default mode is OR. This means that the text and graphics are binary ORed pixel by pixel on the display. With the other two modes, the programmer can make the display do inverted text or display partial text.

Chapter Eight
Built-in Keypad

Boss32s with the Built - In Display also include a keypad for user interface. This keypad is programmable for use with multiple functions: numeric input, button pressed and others.

Reading the Built-In Keypad

The built-in keypad is composed of 24 keys (4 rows by 6 columns): **0-9**, **ENTER**, **CLEAR**, **MENU**, **NEXT**, ↑, ↓, and **F1-F8**. It is accessed using FILE 6. Any of the file input statements may be used to read the keypad: FINPUT, INPUT, INPUT$, GET, KEY, DIN(256). The figure below shows the values that are return for each key that is pressed. The keypad is scanned continuously in the background; if a key press is detected, that key is inserted into an 8 key buffer. Each input operation returns the next character from this buffer.

The keypad should not be read from more than one task at the same time. Some of the keypresses would go to one task while some would go to the other task, causing erroneous inputs. Multiple tasks can read the keypad, but not concurrently.

KEYPAD OVERLAY	INTEGER GET/KEY	INPUT STRING VARIABLE	INPUT NUMERIC VARIABLE
0	48	"0"	0
1	49	"1"	1
2	50	"2"	2
3	51	"3"	3
4	52	"4"	4
5	53	"5"	5
6	54	"6"	6
7	55	"7"	7
8	56	"8"	8
9	57	"9"	9
ENTER	13	CR/LF	CR/LF
CLEAR	8	BACKSPACE	BACKSPACE
F1	65	"A"	
F2	66	"B"	
F3	67	"C"	
F4	68	"D"	
F5	69	"E"	
F6	70	"F"	
F7	45	MINUS (-) SIGN	MINUS (-) SIGN
F8	46	DECIMAL (.) POINT	DECIMAL (.) POINT
MENU	71	"G"	
NEXT	72	"H"	
↑	73	"I"	
↓	74	"J"	

CR/LF Indcates a Carriage Return & Line Feed Pair

Boss32 Keypad Return Values for Statements

The value returned by the keypad is dependent upon the operation being performed. The KEY and GET functions return an integer that corresponds to the key that was pressed. The INPUT and INPUT$ statements, when used with a string variable, return a text string; when used with a numeric variable, they return the number that the user typed in. The following example shows the values that are returned as keys are pressed.

```
100     INTEGER J,K
110     STRING A$(40)
120     FILE  6
130     'Start by displaying GET values until ENTER is pressed
140     K=GET
150     LOCATE 1,1
160     PRIINT K;" ";CHR$(K)
170     WAIT 100
180     IF K <> 13 THEN GOTO 140
190     'Now display INPUT with an integer variable
200     CLS: PRINT "Enter a number > ";
210     INPUT J
220     CLS: PRINT "The number was ";J
230     'Now display INPUT with a string variable
240     CLS:PRINT "Enter anything > ";
250     INPUT A$
260     CLS: PRINT A$
270     'Now show how FINPUT works
280     PRINT "Formatted input > ";
290     FINPUT "I4", J
300     CLS: PRINT " The number was ";J
```

Line 120 sets the current file to FILE 6, the onboard display and keypad. Lines 140 through 180 use the GET function to read the keypad; both the integer value and the string equivalent value are displayed. Because of the WAIT 100 in line 170, it is possible to type faster than keypresses are being read; the first eight keys are buffered, so none should be lost. When the ENTER key is pressed, the program continues to line 200, where it uses the INPUT to get a numeric value from the user. Lines 230 through 260 demonstrate the INPUT statement with a string variable. Lines 270 through 300 use FINPUT to get a 4 character integer number.

Refer to the 32Bit Software Manual for more details on programming.

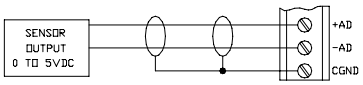
Chapter Nine
Analog I/O

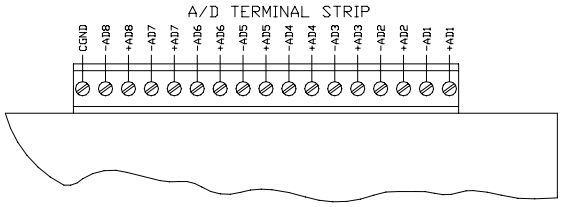
The Boss32 onboard optional circuitry includes up to Eight Analog Input channel; 0-5VDC or 0-20mADC and up to Four Analog Output Channels; 0-10VDC or 0-20mADC. The analog inputs and outputs designed to give 12 bits of accuracy (or 0.05%). Additonal analog inputs and outputs may added by connecting Expansion modules to the Boss32. See Chapter Eleven for details.

Analog Inputs (Analog - Digital Converter)

The Boss32 (optionally ordered) comes with Eight differential analog input channels. Each channel is 12 bit in either 0-20mA or 0-5VDC (Factory configured). The analog inputs are read using the ADC function. The ADC function scales the analog input and returns an numerical (integer) value between 0 (at 0VDC, 0mA) and 32767 (at 5VDC, 20mA). The analog input conversion time is approximately 50 microseconds using the ADC function. The analog inputs are accurate to approximately 0.05%.

The analog input voltage (current) is limited by diodes in the A/D Converter. If an input goes above 5VDC (20mA), the analog input will not be guaranteed to read correctly. Therefore, the system should be designed so the input signal does not go above 5VDC (20mA). All signal cables should be shielded running to the sensor; she shield should be attatched to chassis ground (CGND) on the A/D terminal strip) at the Boss32 end ONLY. See figure below.





The following example shows the use of the ADC function. It simply displays the voltage being read by the first eight analog input channels. In line 140, it reads the A/D value and converts it from A/D units (0-32767) to voltage (0.0 to 5.0). For 0-20mA models, replace line 140 with the following: 140 ADVAL=ADC(CHAN)/32767.0 * 20.0

```
100     'Display the first 8 A/D Channels
110     INTEGER CHAN
120     REAL ADVAL
130     FOR NUM = 1 to 8
140     ADVAL = ADC(CHAN)/32767.0 * 5.0
150     PRINT "Channel "; CHAN; " = "; ADVAL
160     NEXT CHAN
```

It is often necessary to convert an analog input reading into the appropriate engineering units, either for display purposes or to make the program easier to understand. In the previous example, the number was displayed as voltage (or current). In general, the following formula performs the necessary scaling:

$$\frac{Reading}{32767} * (Maxval-Minval) + Minval$$

where *Reading* is the analog input reading, *Maxval* is the maximum value that the sensor can measure, and *Minval* is the minimum value that the sensor can measure. For example, if a temperature sensor returns 0VDC at -50 degrees C and 5VDC at 200 degrees C, then this formula becomes:

$$\frac{Reading}{32767} * (200-(-50)) + (-50)$$

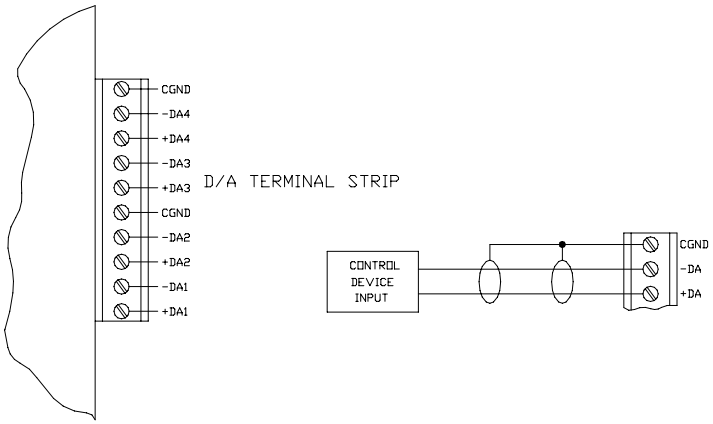
This formula may be reduced when possible as long as the proper mathematics are applied to the new formula.

The following points must be observed to get the greatest accuracy from the Boss32 analog inputs:

- The Boss32 chassis ground (ground lead on the power input connector) must be attatched to earh ground.
- The unused analog inputs on the A/D connector should be shorted (ie: ADx + tied to ADx -).
- Shielded cable should be used to attatch to the sensor. The shield should be attatched to the CGND at the Boss32 A/D connector. The other end of the shield should not be tied to ground, as this could cause a ground loop. Depending upon the construction and mounting of the sensor, the shield may or may not attatched to the sensor chasis. If the sensor shield (which is probably its housing or chasis) is isolated from ground, then it is ok to attatch the shield to the sensor. If the sensor is attatched to ground, then do not attatch the shield, as this could cause a ground loop.

Analog Outputs (Digital - Analog Converter)

The Boss32 also optionally provides onboard, four channels of analog outputs that are 12 bit in accuracy and support either 0-10VDC or 0-20mA. Each analog output channel is written to (controlled by) the DAC statement, which accepts integer values between 0 (0VDC, 0mA) and 32767 (10VDC, 20mA). The figure below shows the analog output wiring.



Boss32 Analog Output Wiring

The Boss32 Analog Outputs provide low current only. They are not capable of driving larger loads such as relays, etc.

Chapter Ten
Network Connectivity

One of the forthcoming requirements in todays control world is the ability to network control systems together. The Boss32 provides network connectivity with Modbus and pure ASCII Communications.

ASCII

The Boss32 can host ASCII serial communication up to 57.6K baud. By utilizing commands, ASCII data can be formatted, sent and received via the serial port(s).

Modbus

The Boss32 supports the Modbus protocol for serial data communication. The Boss32 can operate as slave or Master on a modbus network. The following is a list of modbus commands used by the Boss32:

Modfunction	Returns the last function of the Master device
Modregister	Returns the first register of the last Modbus function performed
Network10	Initializes the network handler for Modbus
Network11	Access other devices on the Modbus Network
Network12	Sets the status of current device
Network3	Writes data to local network register
Network4	Reads data from the local network register

Network Registers

The Boss32 supports the use of Modbus. The Boss32 uses the modbus register assignments to support Master and Slave units on a network. Using the statements above provide all the functionality the user needs to network two or more devices to the Boss32.

For addtional information, consult the proper programming Manual or contact Divelbiss Corporation.

Chapter Eleven
Expansion Ports

The Boss32 has expansion capability for additional control signals not supported on-board. The Expansion Port gives the Boss32 access to Analog to Digital converters, Digital to Analog Converters, High Speed Counters and Stepper Motor Control.

Description

The Boss32 will accept one expansion module mounted into each of its three Expansion Ports. Any of the existing modules can be installed into the Expansion Ports. The expansion modules are powered by the Boss32, and do not require an external power supply.

The Boss32 determines the channel numbers used for expansion modules based on the hardware that it finds. Channels are assigned starting with channel 1; channels are assigned in this order: onboard hardware, then the Expansion Port.

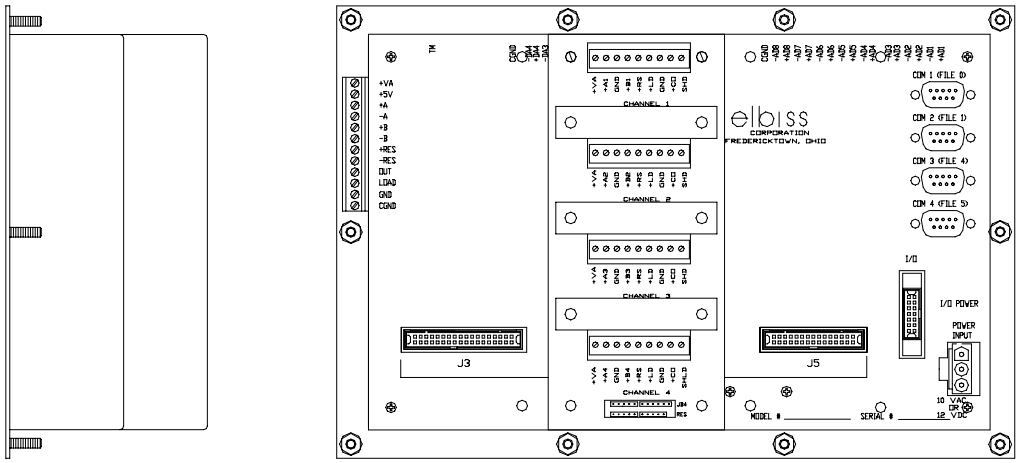
Module Installation

WARNING: Do not install or remove any expansion module with power applied to the Boss32, as damage to the module and/or Boss32 could result.

The Boss32 expansion modules are secured to the Boss32 with four 6-32 machine screws, each 1 5/8 inch long. The module is installed into the Expansion Port connectors (J3-J5).

The Expansion Port J3 on the Boss32 is intended for enhanced function modules. Because of this, it is not self-aligning with some modules and requires more care when installing a module. Carefully position the expansion module over the Expansion Port. J3. Press the module firmly into place. Install the long screws into the corners of the module. If these screws will not thread into the Boss32, then the module is mis-mated and must be removed and re-installed properly.

Side View **Back View**



Expansion Module Installation to a Boss32

High Speed Counter Module

The high speed counter module extends the Boss32 beyond the counter channel that can be installed onboard; by using three modules in addition to the on-board counter, the Boss32 can support up to 13 counter channels. The counter module is available in four models:

Divelbiss Part Number	Description
EX-MOD-CTR24-01	1 channel counter module
EX-MOD-CTR24-02	2 channel counter module
EX-MOD-CTR24-03	3 channel counter module
EX-MOD-CTR24-04	4 channel counter module

Each channel is independent of the others, and each can accept a signal up to 100kHz without loss of counts on any channel; each channel is configured independently of the others. Each channel is similar to the onboard counter on the Boss32, with **A**, **B**, **RESET**, and **LOAD** inputs and a high speed output. The **A**, **B**, and **RESET** inputs have individual low pass filters which the user can configure for 20Hz, 5kHz, or 100kHz. The **RESET** and **LOAD** inputs can be set individually by the user to accept active low or active high signals. The module is housed in an aluminum enclosure, with removable panels to allow user access to the configuration jumper blocks; it bolts to the Boss32 with the mounting screws that are provided with the module.

Wiring Recommendations

For best noise immunity, shielded cable should be used between the counter inputs and the device being monitored; this is especially important for long cable runs. To minimize any potential crosstalk problems between channels, it is recommended that individually paired and shielded cables be used for each channel, especially at high counting speeds. Use the shield terminal provided or connect the shield to earth ground by other means. ***Do not connect the cable shield at both ends of the cable, as this can have an adverse effect.*** Always use separate returns (minus terminal) for each channel, as this lessens the chance of ground loops occurring by making all common terminations at the counter module.

Counter Module Terminal Block Description

+VA Unregulated +10 to +15VDC. This can be used for excitation of external transducers. This is supplied to the terminal block of all channels.

+AX Counter input A for this channel.

GND Input/Output return. This is electrically connected to the Boss Bear digital common. It is not internally connected to earth ground, and should not be connected externally to earth ground.

+BX Counter input B for this channel.

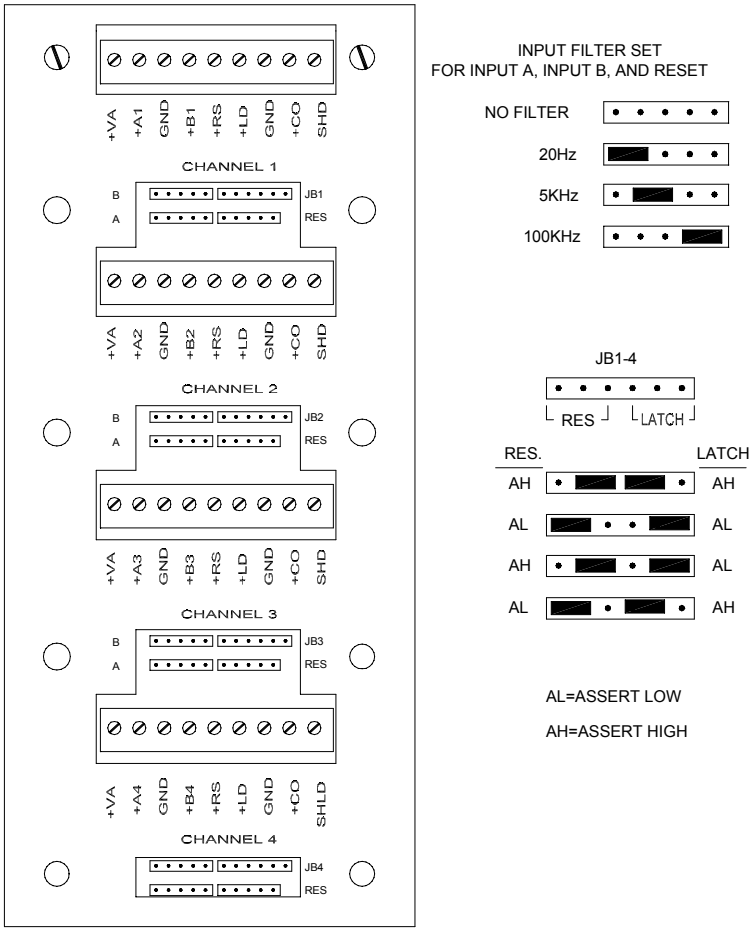
+RX Counter reset input for this channel.

+LX Counter load input for this channel.

GND Input/Output return. This is electrically connected to the Boss32 digital common. It is not internally connected to earth ground, and should not be connected externally to earth ground.

+CX Counter high speed output for this channel.

SHD Cable shield termination. This is used for the drain wire termination of cables coming to this counter channel. This is electrically connected to the Boss32 earth ground system, which includes all of the metal on the Boss32 enclosure. This depends upon the Boss32 power connector being wired correctly to earth ground.



Counter Module Jumper Configurations

Specifications

Channels:	1, 2, 3, or 4, depending upon part number
Resolution:	24 bits
Inputs:	Each channel has 4 inputs: **A**, **B**, **RESET**, and **LOAD**. All inputs are self-sourcing and can be driven with open collector (open drain) drivers. The open circuit potential is 10-15VDC.
Input current:	Logic low: 3.2mA maximum Logic high: 1µA at 15VDC maximum
Input low level:	1VDC maximum

Input high level:	3.5VDC minimum
Max. high level input (sourcing):	+12VDC
Input filtering:	Individually user selectable for inputs **A**, **B**, and **RESET** on each channel: 20Hz (±20%), 5kHz (±20%), 100kHz (±20%).
High Speed Output drive:	Open drain (open collector). Maximum sink current: 100 mA continuous.
Available power:	**+VA** is available for each channel: 10 to 15VDC, unregulated. Maximum collective load current: 80mA.
Wiring termination:	Detachable terminal blocks on 5mm centers. Maximum wire size: #18AWG.

12 Bit Analog to Digital Converter Module

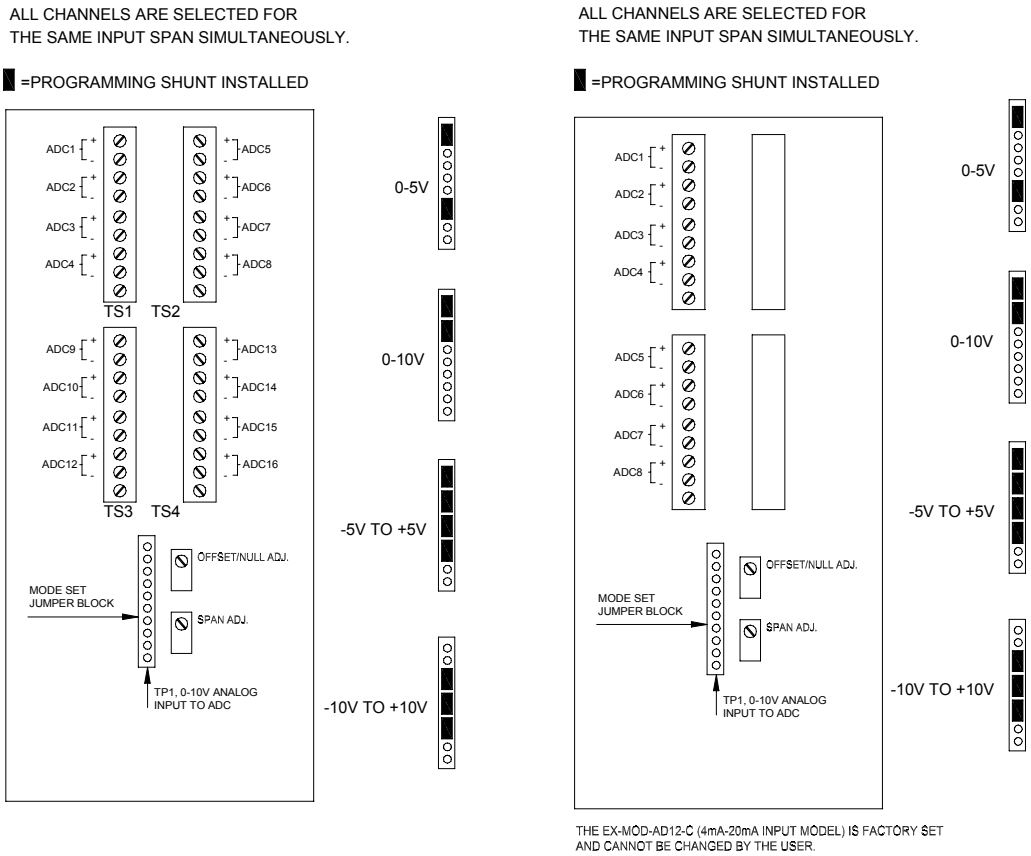
The 12 bit A/D converter module provides enhanced analog measurement capabilities to the Boss32. It can be configured with 16 single-ended channels, 8 differential channels, or 8 current loop (4-20mA) channels. The input span can be set to 0 to +5V, 0 to +10V, -5 to +5, -10 to +10V, or 4 to 20mA; because the channels are multiplexed, all channels are set to the same mode. The 4 to 20mA mode is factory configured and cannot be changed by the user. The A/D module is accessed using the ADC() function of Bear BASIC. The A/D module is available in three models:

Divelbiss Part Number	Description
EX-MOD-AD12-S	16 single-ended inputs, field selectable as 0 to +5V, 0 to +10V, -5 to +5, or -10 to +10V.
EX-MOD-AD12-D	8 differential inputs, field selectable as 0 to +5V, 0 to +10V, -5 to +5, or -10 to +10V.
EX-MOD-AD12-C	8 current loop (4 to 20mA) inputs. This features a “true” current loop, in that it is not referenced to any common. This analog input mode provides the best immunity to noise and cable loss.

Wiring Recommendations

For best noise immunity, shielded cable should be used between the analog inputs and the device being monitored; this is especially important for long cable runs. Except in cases where the analog signals are very dynamic, all channels can be sheathed in the same cable, with one common overall shield. Use the shield terminal provided or connect the shield to earth ground by other means. ***Do not connect the cable shield at both ends of the cable, as this can have an adverse effect.*** Always use separate returns (minus terminal) for each channel, as this lessens the chance of ground loops occurring by making all common terminations at theA/D module; of course, separate returns are required when using the differential and 4-20mA modules.

12 bit A/D Module Jumper Configuration



EX-MOD-AD12-S ***EX-MOD-AD12-D & EX-MOD-AD12-C***

12 bit A/D Module Calibration

The EX-MOD-AD12-S and EX-MOD-AD12-D modules are shipped from the factory set and calibrated for the -5 to +5V single-ended input range. If an input span other than this is selected, then the module must be calibrated. The EX-MOD-AD12-C module is, of course, shipped set and calibrated for 4 to 20mA. Any of the modules can be calibrated at any time, if the user feels that it is necessary.

Before beginning the calibration, allow at least a five minute warm-up period with the power applied to the Boss32 and modules. This allows the OP-AMP buffers to reach operating temperature and stabilize. A very clean and stable DC power source is required for accurate calibration; a laboratory power supply or mercury battery are suitable. A multimeter that is accurate to 0.5mV (with a 10VDC input) is also required.

The following Boss32 BASIC program is used to display the A/D readings during calibration; it should be entered and running:

```
100     'Program to calibrate 12 bit A/D expander module
        integer mode, chanl
        real anlgrd, zero, span
        integer j,ch
        string a$
        data "0 to 5V", "0 to 10V", " 5 to 5V", " 10 to 10V", "4 to 20mA"
        data 0.0, 5.0, 0.0, 10.0,  5.0, 10.0,  10.0, 20.0, 4.0, 16.0

300     'Get operating mode (0 5, 0 10, etc.) from the user
        erase
        print " 1:0 to 5V    2:0 to 10V    3: 5 to 5V"
        print "4: 10 to 10V  5:4 to 20mA     Press 1 5";
320     ch = get
        if ch < $31 or ch > $35 then 320
        mode = ch - $30

        'Get channel number to calibrate
        erase
        print "Enter channel number to calibrate>";
        finput "u2",chanl

        'Display range that was chosen (ie. " 5 to 5V")
        erase
        restore
        for j = 1 to 5
        read a$
        if j = mode then print a$;
        next j
        locate 1,15
        print "Channel ";chanl;
        locate 2,10
        print "Press a key to exit";

        'Get the zero and span values for the chosen mode
        for j = 1 to mode
        read zero,span
        next j
        'Main loop to read the A/D and display the value.  Exit this loop
        ' when a key is pressed.
400     anlgrd = adc(chanl)/32767.0 * span + zero
        locate 1,26
        fprint "f3.3z",anlgrd
        if mode < 5 then print "V";
        if mode = 5 then print "mA";
        wait 20                     'Delay to allow user to read number
        if key=0 then 400
        goto 300
```

Calibrating the unipolar input modes (0 to 5V, 0 to 10V):

With the BASIC A/D calibration program running, short the plus and minus inputs of channel 1 with a jumper wire. Adjust the NULL adjustment until 0.000V is displayed by the program. Remove the shorting jumper and apply ½ of the input span voltage level to the channel 1 input terminals, observing polarity; this is 2.500V for the 0-5V mode, 5.000V for the 0-10V mode. Adjust the span pot until the displayed value matches the input level. The calibration can be verified by applying 5.000V or 10.000V (depending on the mode selected) to the channel 1 input terminals; the displayed value should match the input voltage level.

Calibrating the bipolar input modes (-5 to +5V, -10 to +10V):

With the BASIC A/D calibration program running, short the plus and minus inputs of channel 1 with a jumper wire. Adjust the NULL adjustment until 0.000V is displayed by the program. Remove the shorting jumper and apply +5.000V or +10.000V (depending upon the mode selected) to the channel 1 input terminals, observing polarity. Adjust the span pot until the displayed value matches the input level. The calibration can be verified by applying -5.000V or -10.000V (depending on the mode selected) to the channel 1 input terminals; the displayed value should match the input voltage level.

Calibrating the current loop (4 to 20mA) input mode:

With the BASIC A/D calibration program running, apply 4.00mA to channel 1 and adjust the null pot until 4.00 is displayed. Apply 20.00mA to channel 1 and adjust the span pot until 20.00 is displayed.

Specifications

Channels:	8 or 16, depending upon part number
Resolution:	12 bits
Input level:	For models EX-MOD-AD12-S and EX-MOD-AD12-D, user selectable: 0 to +5V, 0 to +10V, -5 to +5, -10 to +10V. For model EX-MOD-AD12-C, factory set: 4 to 20mA.
Input linearity error:	±1 least significant bit.
Input noise rejection:	±1 least significant bit.

Input impedance:

Model	Impedance	Channel tracking
EX-MOD-AD12-S	2.3M min.	N/A
EX-MOD-AD12-D	3.3M min.	N/A
EX-MOD-AD12-C	250W, ±2%	±0.2%,0-60 C

Maximum input voltage:	For models EX-MOD-AD12-S and EX-MOD-AD12-D, the maximum safe input voltage on any channel is 24VDC.
Input termination:	Detachable terminal blocks on 5mm centers. Maximum wire size: #18AWG. Two terminals are provided for each channel, with a shield terminal for each group of four channels.
Conversion time:	600µsec, using the ADC() function of Boss32 BASIC.

10 Bit Digital to Analog Converter Module

The 10 bit D/A converter module provides analog output capabilities to the Boss32. Using this module, up to 16 analog outputs can be added to a Boss32. Each channel can be individually set to 0 to +5V, 0 to +10V, -5 to +5, -10 to +10V, or 4 to 20mA. The D/A module is available in four models:

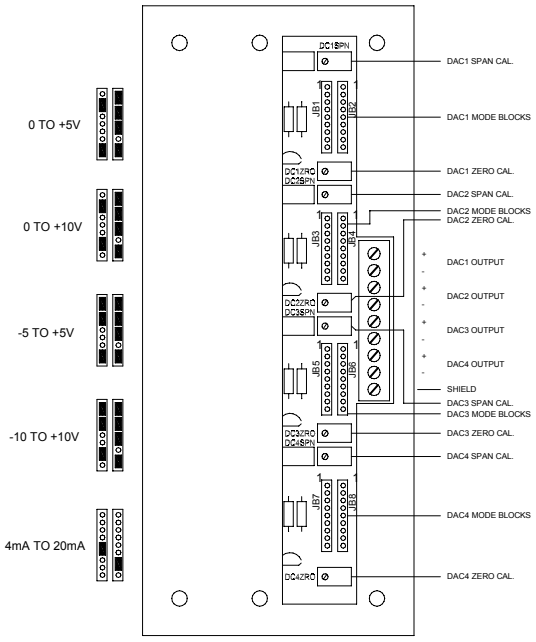
Divelbiss Part Number	Description
EX-MOD-DA10-01	1 channel DAC module
EX-MOD-DA10-02	2 channel DAC module
EX-MOD-DA10-03	3 channel DAC module
EX-MOD-DA10-04	4 channel DAC module

In the 4 to 20mA mode, this unit implements a “true” current loop; it is not referenced to any common. This is the best D/A drive to use for optimum immunity to noise and cable loss. The output current is regulated and can maintain the chosen analog output over a wide range of load impedance.

Wiring Recommendations

For best noise immunity, shielded cable should be used between the analog outputs and the device being controlled; this is especially important for long cable runs. Except in cases where the analog signals are very dynamic, all channels can be sheathed in the same cable, with one common overall shield. Use the shield terminal provided or connect the shield to earth ground by other means. ***Do not connect the cable shield at both ends of the cable, as this can have an adverse effect.*** Always use separate returns (minus terminal) for each channel, as this lessens the chance of ground loops occurring by making all common terminations at the D/A module. #22AWG or larger wire can be used for runs of less than 15 feet; for longer runs #18AWG is recommended for minimum signal loss due to wire resistance. For very long cable runs, the 4-20mA mode is recommended, since it compensates for cable loss (within design limitations).

10 bit D/A Module Jumper Configuration



10 bit D/A Module Jumper Configurations

10 bit D/A Module Calibration

All channels are factory set and calibrated for the -10 to +10V mode. If an output mode other than -10 to +10V is selected, the calibration will be necessary. The operating mode should be set before connecting the module to any machine or interface device, in order to avoid any confusion and possible damage. Often, it is not necessary to calibrate the module to an absolute standard, because the channels will need to be adjusted in conjunction with other equipment (for example, adjusting the zero potentiometer to cause a motor to stop all motion).

Before beginning the calibration, allow at least a five minute warm-up period with the power applied to the Boss32 and modules. This allows the OP-AMP buffers to reach operating temperature and stabilize. A multimeter that is accurate to 0.005V (with a 10V input) is required.

The following Boss32 BASIC program is used to set the analog outputs during calibration; it should be entered and running:

```
100     'Program to calibrate 10 bit D/A expander module
        integer mode, chanl
        real anlgset, zero, span
        integer j,ch
        string a$
        data "0 to 5V", "0 to 10V", " 5 to 5V", " 10 to 10V", "4 to 20mA"
        data 0.0, 5.0, 0.0, 10.0,  5.0, 10.0,  10.0, 20.0, 4.0, 16.0

300     'Get operating mode (0 5, 0 10, etc.) from the user
        erase
        print " 1:0 to 5V    2:0 to 10V    3: 5 to 5V"
        print "4: 10 to 10V  5:4 to 20mA     Press 1 5";
320     ch = get
        if ch < $31 or ch > $35 then 320
        mode = ch - $30

        'Get channel number to calibrate
        erase
        print "Enter channel number to calibrate>";
        finput "u2",chanl

        'Display range that was chosen (ie. " 5 to 5V")
        erase
        restore
        for j = 1 to 5
        read a$
        if j = mode then print a$;
        next j
        locate 1,15
        print "Channel ";chanl;
        'Get the zero and span values for the chosen mode
        for j = 1 to mode
        read zero,span
```

```
        next j
        'Main loop to get the desired output level from the user and set the D/A to
        'this value.
400     locate 2,1
        print "Enter output level >                ";
        locate 2,29
        if mode < 5 then print "V";
        if mode = 5 then print "mA";
        locate 2,22
        finput "f3.3",anlgset
        if anlgset < zero or anlgset > (zero+span) then 400
        dac chanl, (anlgset - zero) / span * 1023
        locate 2,1
        print "F1 next level    Any other key to exit";
        ch = get
        if ch = $41 then 400
        goto 300
```

Calibrating the unipolar input modes (0 to 5V, 0 to 10V):

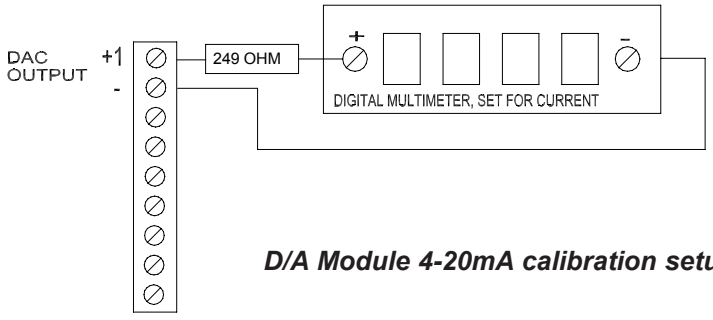
With the BASIC D/A calibration program running, select the desired channel to calibrate, and specify 0 as the DAC output value. Adjust the proper zero potentiometer for 0.00V on the output terminals of the channel being calibrated. Next, specify 2.5 or 5.0 (depending upon the range) as the DAC output value. Adjust the proper span potentiometer for the mid-point of the voltage range selected (2.499V for the 0-5V range, or 4.99V for the 0-10V range). The calibration can be verified by specifying 5.0 or 10.0 as the DAC output value; the output should measure 4.99 to 5.00V, or 9.99 to 10.00V, depending upon the mode.

Calibrating the bipolar input modes (-5 to +5V, -10 to +10V):

With the BASIC D/A calibration program running, select the desired channel to calibrate, and specify 0 as the DAC output value. Adjust the proper zero potentiometer for 0.00V on the output terminals of the channel being calibrated. Next, specify 5.0 or 10.0 as the DAC output value. Adjust the proper span potentiometer for the maximum value of the voltage range selected (4.99V for the ±5V range, or 9.99V for the ±10V range). The calibration can be verified by specifying -5.0 or -10.0 as the DAC output value; the output should measure -4.99 to -5.00V, or -9.99 to -10.00V, depending upon the mode.

Calibrating the current loop (4 to 20mA) input mode:

Use a 250 (5% or better) load across the channel being calibrated With the BASIC D/A calibration program running, select the desired channel to calibrate, and specify 4.0 as the DAC output value. Adjust the proper zero potentiometer for 4.00mA through the test load. Next, specify 20.0 as the DAC output value. Adjust the proper span potentiometer for a 20.0mA reading. The calibration can be verified by specifying 12.0 as the DAC output value; the mul-



D/A Module 4-20mA calibration setup

timeter should read 12.0mA.

Specifications

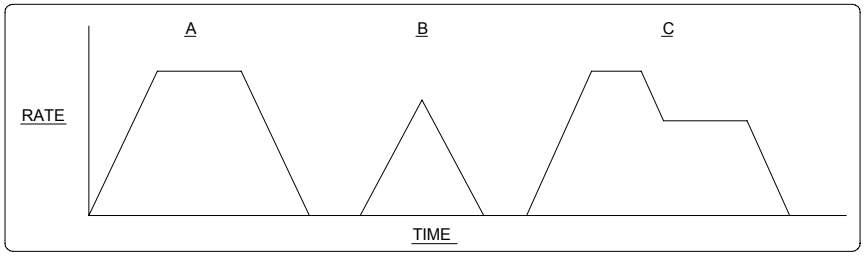
Channels:	1, 2, 3, or 4, depending upon part number
Resolution:	10 bits
Output level:	Each channel is user selectable: 0 to +5V, 0 to +10V, -5 to +5, -10 to +10V, or 4 to 20mA.
Output null/offset adjustments:	±250mV, or 0-6mA in current (4 to 20mA) mode.
Output linearity error:	±0.05% FSR maximum.
Output noise:	Less than 4mV peak to peak.
Output drive current:	For voltage output modes: 10mA minimum at 10V.
Compliance voltage:	For current (4 to 20mA) mode: 9VDC minimum.
Load tolerance:	For current (4 to 20mA) mode: 250 , ±20%.
Output termination:	Detachable terminal blocks on 5mm centers. Maximum wire size: #18AWG. Two terminals are provided for each channel, with a single shield terminal.

Stepping Motor Interface Module

- Controls up to three stepping motors
- Contains a dedicated microprocessor, allowing stepping motors to be controlled without burdening the Boss32C processor
- Supports step rates from 20 to 10000 pulses/sec
- Provides linear acceleration and deceleration ramps
- Motors can run independently, or one channel can output a pulse rate that is proportional to another channel's pulse rate
- Optically isolated, open collector step and direction outputs are provided for each channel; each channel supports a separate power source, from 5 to 24VDC
- Supports separate acceleration and deceleration rates for each motor
- An interrupt can be sent to the Boss32 at the completion of a movement
- Plugs into any of the Boss32 expansion ports
- Bear BASIC statement EXMOD is used to communicate with the module

The Divelbiss Stepping Motor Interface is an expansion module for use with the Boss32; the module contains its own microprocessor and I/O circuitry. It controls up to three stepping motors simultaneously, providing separate step and direction lines to each motor translator. The module handles all of the processing required to generate the movement profile specified by the Boss32 program; a profile consists of a linear acceleration, running at the specified pulse rate, and linear deceleration to a stop. The drawing on the next page illustrates some possible movement profiles. Curve A shows the normal profile, in which the motor ramp is up to its running rate, then ramps down to a stop. Note that the deceleration and acceleration slopes are different. Curve B shows a profile in which the specified step distance is too short to allow the motor to reach its running speed; it simply ramps up and ramps down to a stop. Curve C shows a profile in which the ramps up to its running rate, and then the Boss32 sends a new (slower) running rate; the motor ramps down to this new rate, runs for the specified number of pulses, and then ramps down to a stop.

The module is housed in an aluminum enclosure; it bolts to the Boss32 with the mounting screws that are provided with the module.



Sample Motor Profile Curve

Terminal Block Description

S1 Step output for channel 1. This is an opto-isolated, open collector output that is referenced to terminal G1.

D1 Direction output for channel 1. This is an opto-isolated, open collector output that is referenced to terminal G1.

P1 Power source for channel 1 opto isolator circuit. This is a 5 VDC to 24 VDC source that comes from a separate external power supply.

G1 Ground return for channel 1. This is the common lead for the power source attached to P1.

S2 Step output for channel 2. This is an opto-isolated, open collector output that is referenced to terminal G1.

D2 Direction output for channel 2. This is an opto-isolated, open collector output that is referenced to terminal G1.

P2 Power source for channel 2 opto isolator circuit. This is a 5 VDC to 24 VDC source that comes from a separate external power supply.

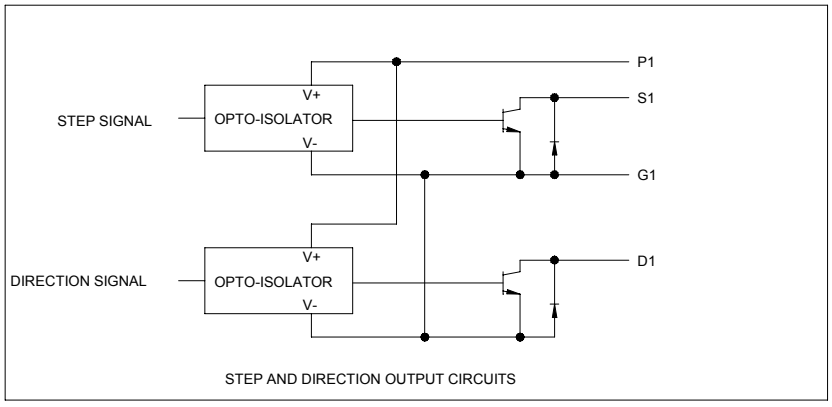
G2 Ground return for channel 2. This is the common lead for the power source attached to P1.

S3 Step output for channel 3. This is an opto-isolated, open collector output that is referenced to terminal G1.

D3 Direction output for channel 3. This is an opto-isolated, open collector output that is referenced to terminal G1.

P3 Power source for channel 3 opto isolator circuit. This is a 5 VDC to 24 VDC source that comes from a separate external power supply.

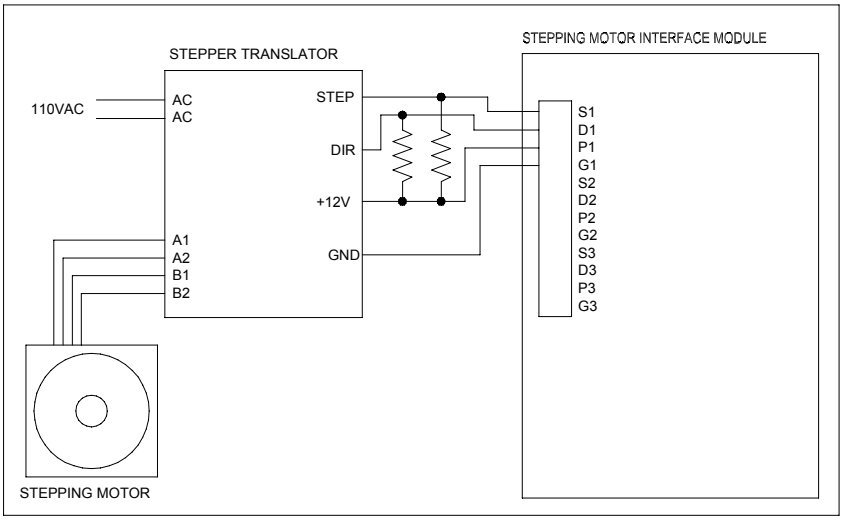
G3 Ground return for channel 3. This is the common lead for the power source attached to P1.



Output Circuit

Wiring

For best noise immunity, shielded cable should be used between the Stepper Motor Module and the stepper driver. Use the shield terminal provided or connect the shield to earth ground by other means. Do not connect the cable shield at both ends of the cable, as this can have an adverse effect. The following wiring diagram shows how the module is connected to a stepper motor translator.



Module Wiring

Specifications

- Controls up to three stepping motors
- Contains a dedicated microprocessor, allowing stepping motors to be controlled without burdening the Boss32 processor
- Supports step rates from 20 to 10000 pulses/sec
- Provides linear acceleration and deceleration ramps
- Motors can run independently, or one channel can output a pulse rate that is proportional to another channel's pulse rate
- Optically isolated, open collector step and direction outputs are provided for each channel
- Supports separate acceleration and deceleration rates for each motor
- An interrupt can be sent to the Boss32 at the completion of a movement

Software Interface

The BASIC program uses the EXMOD statement to interface with the stepper module; the module supports five commands: IDENTIFY, READ_STATUS, SET_MODE, SET_ACCEL, and STEP.

The IDENTIFY command (code $80) returns the module identification string, which can be used to verify that the correct module is installed. It will return 15 bytes: "103StepMod3 ". The first three bytes are the version number (in this example, version 1.03).

```
        Input string:
                command         1 byte          $80
        Output string:
                ID string       15 bytes        "103StepMod3    "
100  INTEGER PORT
```

```
110  STRING EX$
300  PORT=3                                  ‘ Module in Expansion Port
310  EX$=CHR$($80)                           ‘ IDENTIFY command
320  EXMOD PORT,EX$,15                       ‘ Read 15 bytes from port 4 into EX$
330  PRINT EX$                               ‘ Display “103StepMod3    “
```

The READ_STATUS command (code $84) returns the current status for the specified channel. It returns the current mode, step count, and step rate for the channel.

Input string:

command	1 byte	$84
channel	1 byte	1, 2, or 3

Output string:

mode	1 byte	
step count	4 bytes	number of pulses left in movement
step rate	2 bytes	current pulse rate (pulses per second)

```
100  INTEGER PORT,CHNL
110  STRING EX$(127)
300  PORT=3: CHNL=1                          ‘ Module in Expansion Port, channel 1
310  EX$=CONCAT$(CHR$($84),CHR$(CHNL))‘ Command and channel
320  EXMOD PORT,EX$,7                        ‘ Read back 7 bytes
330  FPRINT “h2”,ASC(EX$)                    ‘ Display mode in hexadecimal
340  PRINT CVI(MID$(EX$,2,2))                ‘ Display low word of step count
350  PRINT CVI(MID$(EX$,6,2))                ‘ Display current pulse rate
```

The SET_MODE command (code $85) sets the operating modes for the specified channel.

Input string:

command	1 byte	$85
channel	1 byte	1, 2, or 3
mode	1 byte	d0=1 to enable Boss Bear interrupt for this channel d1=1 to link this channel to channel 1 d2=direction when link mode is enabled
link ratio	2 bytes	0.04 to 255.96 (in 8:8 format)

Output string:

no return string

```
100  INTEGER PORT, CHNL
110  INTEGER MODE, LRATIO
120  STRING EX$
300  PORT=3: CHNL=2                          ‘ Module in Expansion Port, channel 2
310  MODE=2                                  ‘ Enable link mode with direction=0
320  LRATIO=.25*$100                         ‘ Make channel 2 run at 1/4 channel 1 rate
330  EX$=CONCAT$(CHR$($85),CHR$(CHNL))‘ Command and channel
340  EX$=CONCAT$(EX$,CHR$(MODE))             ‘ Operating mode
350  EX$=CONCAT$(EX$,MKI$(LRATIO))           ‘ Link ratio
360  EXMOD PORT,EX$
```

The SET_ACCEL command (code $83) sets the acceleration and deceleration rates for the specified channel. These rates will be used for all succeeding STEP commands. The

acceleration rate should not be changed while a channel is accelerating, because the new rate takes effect immediately. The deceleration rate can be changed at any time, because the rate that is in effect when the STEP command is issued will be used, even if the deceleration rate is changed before the deceleration actually occurs.

Input string:

command	1 byte	$83
channel	1 byte	1, 2, or 3
accel value	2 bytes	10 to 10000 pulses/second/second
decel value	2 bytes	10 to 10000 pulses/second/second

Output string:

no return string

```
100  INTEGER PORT, CHNL
110  INTEGER ACRATE, DCRATE
120  STRING EX$
300  PORT=3: CHNL=1                          ‘ Module in Expansion Port, channel 1
310  ACRATE=2000: DCRATE=1500                ‘ Set acceleration/deceleration rates
320  EX$=CONCAT$(CHR$($83),CHR$(CHNL))‘ Command and channel
330  EX$=CONCAT$(EX$,MKI$(ACRATE))           ‘ Acceleration rate
340  EX$=CONCAT$(EX$,MKI$(DCRATE))           ‘ Deceleration rate
350  EXMOD PORT,EX$
```

The STEP command (code $82) causes a step motor movement to occur. The channel, pulse count, pulse rate, and direction are specified in the command. The module will output the specified pulse count, accelerating and decelerating as necessary using the current acceleration and deceleration rates. If the channel is currently in the middle of a previous movement command, then it will smoothly accelerate or decelerate to the new rate and use the new pulse count; note, however, that it can’t change directions smoothly while moving (it instantly goes from one direction to the other).

If the pulse count is 0, then the module will output pulses in continuous mode: it will accelerate or decelerate to the specified rate, then output pulses until another STEP command is issued. If the pulse count is -1 (ie. all four bytes are $FF) then the pulse rate is changed without modifying the current pulse count.

Input string:

command	1 byte	$82
channel	1 byte	1, 2, or 3
pulse count	4 bytes	0: output pulses in continuous mode 1 to 2147483647: output this many pulses -1: change rate without changing pulse count
pulse rate	2 bytes	10 to 10000 pulses/second
direction	1 byte	0 or 1

Output string:

no return string

```
100  INTEGER PORT, CHNL
110  INTEGER NSLO, NSHI                      ‘ Number of steps (low word and high word)
120  INTEGER RATE, DIR                       ‘ Pulse rate and direction
130  STRING EX$
```

```
300  PORT=3: CHNL=1
310  NSHI=0: NSLO=400                          ‘ Step 400 pulses
320  RATE=500: DIR=1                           ‘ 500 pulses/second
330  EX$=CONCAT$(CHR$($83),CHR$(CHNL))‘ Command and channel
340  EX$=CONCAT$(EX$,MKI$(NSLO))               ‘  Low word of number of steps
350  EX$=CONCAT$(EX$,MKI$(NSHI))               ‘ High word of number of steps
360  EX$=CONCAT$(EX$,MKI$(RATE))               ‘ Pulse rate
370  EX$=CONCAT$(EX$,CHR$(DIR))                ‘ Direction
380  EXMOD PORT,EX$
```

Example Program

```
    !*****************************************************************
    !
    ! Program to test stepper module.
    !
    !*****************************************************************

    integer port, chan, mode
    integer acrate, dcrate
    integer numstep, nshi, rate, dir
    integer mdset, lratio
    string stm$(40)

    integer chan1
    integer mode1, count, raterd
    string stm1$(40)

    port = 4
    run 1,10

500  chan = 1: acrate = 500: dcrate = 500: gosub 1100
      chan = 2: acrate = 500: dcrate = 500: gosub 1100
      mdset = 2: lratio = $0080: gosub 1300

510  chan=1: nshi = 0: numstep = 4000: rate = 1000: dir = 0: gosub 1200
      wait 22
600  if mode then 600
      goto 510
      stop

    !*****************************************************************
    !
    ! Subroutine to set acceleration and deceleration rates.
    !
    !*****************************************************************
1100 stm$=concat$(chr$($83),chr$(chan))
    stm$=concat$(stm$,mki$(acrate))
    stm$=concat$(stm$,mki$(dcrate))
```

```
    exmod port,stm$
    return

    !*********************************************************************
    !
    ! Subroutine to step the motor.
    !
    !*********************************************************************
1200 stm$=concat$(chr$($82),chr$(chan))
    stm$=concat$(stm$,mki$(numstep))
    stm$=concat$(stm$,mki$(nshi))
    stm$=concat$(stm$,mki$(rate))
    stm$=concat$(stm$,chr$(dir))
    exmod port,stm$
    return

    !*********************************************************************
    !
    ! Subroutine to set the mode.
    !
    !*********************************************************************
1300 stm$=concat$(chr$($85),chr$(chan))
    stm$=concat$(stm$,chr$(mdset))
    stm$=concat$(stm$,mki$(lratio))
    exmod port,stm$
    return
2000 task 1
    chan1=1
    gosub 2100
    mode = mode1
    print "       ";
    chan1=2
    gosub 2100
    print
    exit

    !*********************************************************************
    !
    ! Subroutine to read stepper status and print it on the current file.
    !
    !*********************************************************************
2100 stm1$=concat$(chr$($84),chr$(chan1))
    exmod port,stm1$,7
    mode1 = asc(stm1$)
    count = cvi(mid$(stm1$,2,2))
    raterd = cvi(mid$(stm1$,6,2))

    fprint "h2x2z",mode1                    ' mode
    fprint "i5x2z",cvi(mid$(stm1$,4,2))     ' step count high word
    fprint "i5x2z",count                    ' step count low word
    fprint "i5z",raterd                     ' current rate
    return
```

Chapter Twelve
Real Time I/O

The Boss32 I/O Bus allows up to 128 inputs and 128 outputs, using the standard Divelbiss High Density Bear Bones Expanders and Presto Panel; I/O boards of different types can be intermixed in any combination. The DIN function is used to read the status of inputs, and the DOUT statement is used to control outputs. The tables in this chapter show how the expander I/O address is related to the Boss32 I/O number.

I/O Interface

When programming with I/O boards, remember to take the response time of the board into account. Inputs may have debounce circuitry that delays the response to a signal change by 10 to 20 msec. Outputs may take 20 msec to change state; this will limit the pulse rate that can be generated.

DIN/DOUT Addr	Hex	Dec.	DIN/DOUT Addr	Hex	Dec.	DIN/DOUT Addr	Hex	Dec.	DIN/DOUT Addr	Hex	Dec.
0/0	$00	0	3/0	$30	48	6/0	$60	96	9/0	$90	144
0/1	$01	1	3/1	$31	49	6/1	$61	97	9/1	$91	145
0/2	$02	2	3/2	$32	50	6/2	$62	98	9/2	$92	146
0/3	$03	3	3/3	$33	51	6/3	$63	99	9/3	$93	147
0/4	$04	4	3/4	$34	52	6/4	$64	100	9/4	$94	148
0/5	$05	5	3/5	$35	53	6/5	$65	101	9/5	$95	149
0/6	$06	6	3/6	$36	54	6/6	$66	102	9/6	$96	150
0/7	$07	7	3/7	$37	55	6/7	$67	103	9/7	$97	151
0/8	$08	8	3/8	$38	56	6/8	$68	104	9/8	$98	152
0/9	$09	9	3/9	$39	57	6/9	$69	105	9/9	$99	153
0/10	$0A	10	3/10	$3A	58	6/10	$6A	106	9/10	$9A	154
0/11	$0B	11	3/11	$3B	59	6/11	$6B	107	9/11	$9B	155
0/12	$0C	12	3/12	$3C	60	6/12	$6C	108	9/12	$9C	156
0/13	$0D	13	3/13	$3D	61	6/13	$6D	109	9/13	$9D	157
0/14	$0E	14	3/14	$3E	62	6/14	$6E	110	9/14	$9E	158
0/15	$0F	15	3/15	$3F	63	6/15	$6F	111	9/15	$9F	159
1/0	$10	16	4/0	$40	64	7/0	$70	112	10/0	$A0	160
1/1	$11	17	4/1	$41	65	7/1	$71	113	10/1	$A1	161
1/2	$12	18	4/2	$42	66	7/2	$72	114	10/2	$A2	162
1/3	$13	19	4/3	$43	67	7/3	$73	115	10/3	$A3	163
1/4	$14	20	4/4	$44	68	7/4	$74	116	10/4	$A4	164
1/5	$15	21	4/5	$45	69	7/5	$75	117	10/5	$A5	165
1/6	$16	22	4/6	$46	70	7/6	$76	118	10/6	$A6	166
1/7	$17	23	4/7	$47	71	7/7	$77	119	10/7	$A7	167
1/8	$18	24	4/8	$48	72	7/8	$78	120	10/8	$A8	168
1/9	$19	25	4/9	$49	73	7/9	$79	121	10/9	$A9	169
1/10	$1A	26	4/10	$4A	74	7/10	$7A	122	10/10	$AA	170
1/11	$1B	27	4/11	$4B	75	7/11	$7B	123	10/11	$AB	171
1/12	$1C	28	4/12	$4C	76	7/12	$7C	124	10/12	$AC	172
1/13	$1D	29	4/13	$4D	77	7/13	$7D	125	10/13	$AD	173
1/14	$1E	30	4/14	$4E	78	7/14	$7E	126	10/14	$AE	174
1/15	$1F	31	4/15	$4F	79	7/15	$7F	127	10/15	$AF	175
2/0	$20	32	5/0	$50	80	8/0	$80	128	11/0	$B0	176
2/1	$21	33	5/1	$51	81	8/1	$81	129	11/1	$B1	177
2/2	$22	34	5/2	$52	82	8/2	$82	130	11/2	$B2	178
2/3	$23	35	5/3	$53	83	8/3	$83	131	11/3	$B3	179
2/4	$24	36	5/4	$54	84	8/4	$84	132	11/4	$B4	180
2/5	$25	37	5/5	$55	85	8/5	$85	133	11/5	$B5	181
2/6	$26	38	5/6	$56	86	8/6	$86	134	11/6	$B6	182
2/7	$27	39	5/7	$57	87	8/7	$87	135	11/7	$B7	183
2/8	$28	40	5/8	$58	88	8/8	$88	136	11/8	$B8	184
2/9	$29	41	5/9	$59	89	8/9	$89	137	11/9	$B9	185
2/10	$A	42	5/10	$A	90	8/10	$A	138	11/10	$BA	186
2/11	$B	43	5/11	$B	91	8/11	$B	139	11/11	$BB	187
2/12	$C	44	5/12	$C	92	8/12	$C	140	11/12	$BC	188
2/13	$D	45	5/13	$D	93	8/13	$D	141	11/13	$BD	189
2/14	$E	46	5/14	$E	94	8/14	$E	142	11/14	$BE	190
2/15	$F	47	5/15	$F	95	8/15	$F	143	11/15	$BF	191

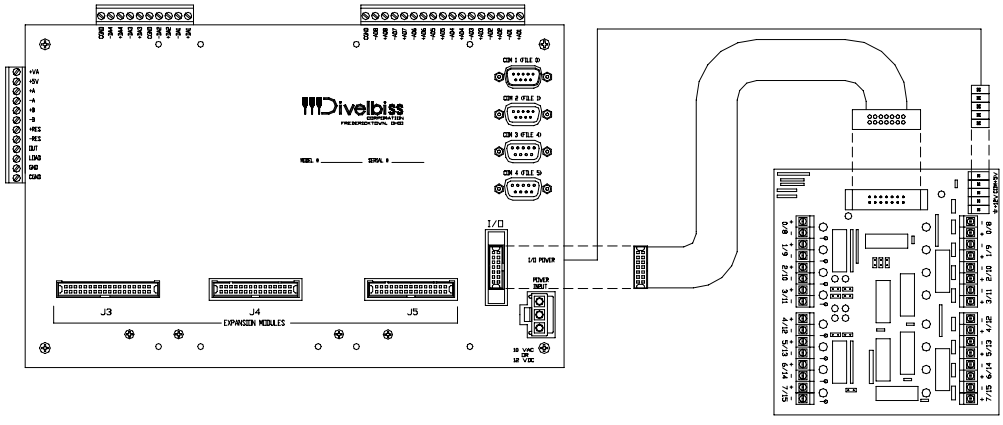
Addr	Hex	Dec.	Addr	Hex	Dec.	Addr	Hex	Dec.	Addr	Hex	Dec.
12/0	$C0	192	13/0	$D0	208	14/0	$E0	224	15/0	$F0	240
12/1	$C1	193	13/1	$D1	209	14/1	$E1	225	15/1	$F1	241
12/2	$C2	194	13/2	$D2	210	14/2	$E2	226	15/2	$F2	242
12/3	$C3	195	13/3	$D3	211	14/3	$E3	227	15/3	$F3	243
12/4	$C4	196	13/4	$D4	212	14/4	$E4	228	15/4	$F4	244
12/5	$C5	197	13/5	$D5	213	14/5	$E5	229	15/5	$F5	245
12/6	$C6	198	13/6	$D6	214	14/6	$E6	230	15/6	$F6	246
12/7	$C7	199	13/7	$D7	215	14/7	$E7	231	15/7	$F7	247
12/8	$C8	200	13/8	$D8	216	14/8	$E8	232	15/8	$F8	248
12/9	$C9	201	13/9	$D9	217	14/9	$E9	233	15/9	$F9	249
12/10	$CA	202	13/10	$DA	218	14/10	$EA	234	15/10	$FA	250
12/11	$CB	203	13/11	$DB	219	14/11	$EB	235	15/11	$FB	251
12/12	$CC	204	13/12	$DC	220	14/12	$EC	236	15/12	$FC	252
12/13	$CD	205	13/13	$DD	221	14/13	$ED	237	15/13	$FD	253
12/14	$CE	206	13/14	$DE	222	14/14	$EE	238	15/14	$FE	254
12/15	$CF	207	13/15	$DF	223	14/15	$EF	239	15/15	$FF	255

The Addr column shows the Expander I/O address; the Hex column shows the Boss32 I/O number in hexadecimal form, while the Dec. column shows it in decimal form.

Relationship Between I/O Board Address and Boss32 I/O Number

Connecting I/O

The Boss32 communicates to I/O cards via a ribbon cable connected to the I/O port connector on the Boss32. The I/O cards receive their DC power for operation from a power cable connected to the Boss32. Connecting I/O is as easy as connecting these two cables. The maximum recommended length for these cables is 12 feet. Using cables greater than 12 feet may result in a degredation of signals and produce random I/O occurances from the introduction of noise onto the real time bus. For longer runs, the Divelbiss Real Time Bus Driver is recommended.



I/O Expander Connection Example

Addressing I/O Cards

Each I/O card for addressing purposes must be assigned a Page and a specific I/O point. 16 Inputs and 16 Outputs equals one page. I/O cards can have 16 I/O or just 8 I/O. When an I/O card is only 8 I/O, then it is considered half of the page. The card can be configured as either the upper half (U) or lower half (L) of the I/O page via a jumper on the card. If the lower half is selected, then I/O 0-7 will be used; if the uppper half is selected, then I/O 8-15 will be used.

The bus addressing system is based on the binary system. For each jumper removed from the board represents a “bit”. Reading them all together as a binary number is the first step to determing the page. If you convert the binary number to an integer, that is the Page that the I/O board is addressed to.

NOTE: The jumper installed will give a 0 for the bit it is installed on.

For more information on the addressing of I/O, refer to each individual I/O Card Data Sheet.

Card Page Address	Card Paging Shunts (Address Selector) 8 4 2 1	U/L Selector Lower / Upper	DIN/ DOUT 8 I/O Cards	DIN/ DOUT 16 I/O Cards	Card Page Address	Card Paging Shunts (Address Selector) 8 4 2 1	U/L Selector Lower / Upper	DIN/ DOUT 8 I/O Cards	DIN/ DOUT 16 I/O Cards
0	■ ■ ■ ■	U○ L● / U● L○	0-7 8-15	0-15	8	○ ■ ■ ■	U○ L● / U● L○	128-135 136-143	128-143
1	■ ■ ■ ○	U○ L● / U● L○	16-23 24-31	16-31	9	○ ■ ■ ○	U○ L● / U● L○	144-151 152-159	144-159
2	■ ■ ○ ■	U○ L● / U● L○	32-39 40-47	32-47	10	○ ■ ○ ■	U○ L● / U● L○	160-167 168-175	160-175
3	■ ■ ○ ○	U○ L● / U● L○	48-55 56-63	48-63	11	○ ■ ○ ○	U○ L● / U● L○	176-183 184-191	176-191
4	■ ○ ■ ■	U○ L● / U● L○	64-71 72-79	64-79	12	○ ○ ■ ■	U○ L● / U● L○	192-199 200-207	192-207
5	■ ○ ■ ○	U○ L● / U● L○	80-87 88-95	80-95	13	○ ○ ■ ○	U○ L● / U● L○	208-215 216-223	208-223
6	■ ○ ○ ■	U○ L● / U● L○	96-103 104-111	96-111	14	○ ○ ○ ■	U○ L● / U● L○	224-231 232-239	224-239
7	■ ○ ○ ○	U○ L● / U● L○	112-119 120-127	112-127	15	○ ○ ○ ○	U○ L● / U● L○	240-247 248-255	240-255

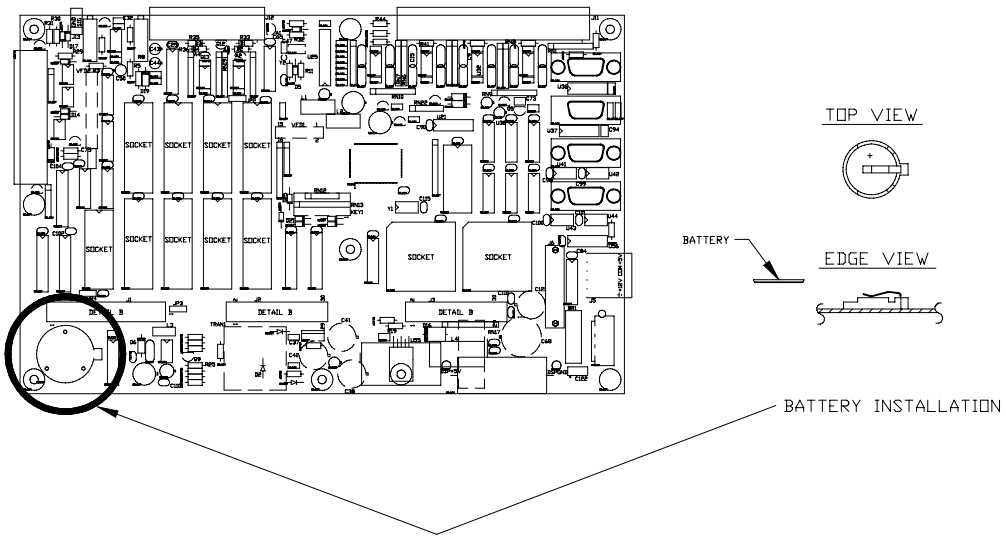
Card Paging Shunt Configurations

Chapter Thirteen
Upgrading Your BOSS32

Replacing the Battery

The Boss32 non-volatile memory relies on a 3V lithium battery for its integrity. The lithium battery should provide years of problem free operation. In the event that the battery needs replaced, complete the steps outlined below. Consult factory for replacement batteries.

1. Dissasemble Boss32. See Disassembling the Boss32.
2. Paying attention to polarity, carefully slide the coin battery from its socket labled BAT1.
3. Carefully slide the new battery into the socket paying attention to the proper polarity.
4. Reassemble the Boss32.



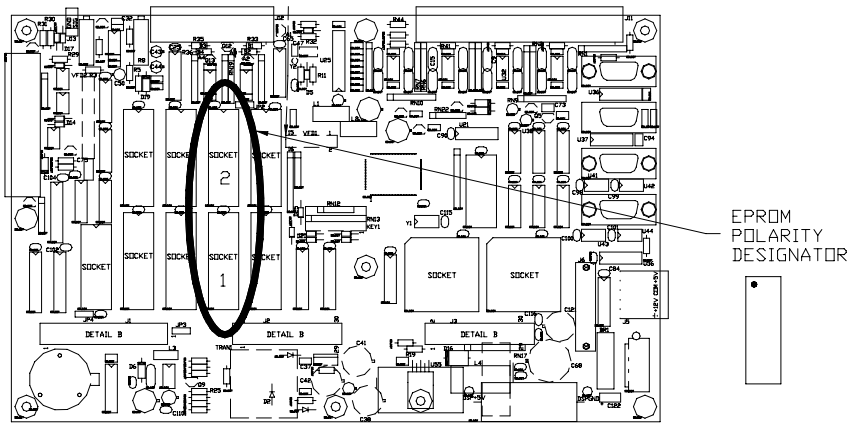
Replacing the Battery Diagram

Firmware Upgrades

To upgrade the firmware of your Boss32, complete the steps outlined below. Consult factory for availablity of firmware upgrades and details of added features.

1. Disassemble Boss32. See Disassembling the Boss32.
2. Remove the installed firmware from UXX and UXX. Use caution not todamage the EEPROM sockets or the Boss32 board.
3. Install the firmware upgrade EPROM 1 into UXX and EPROM 2 into UXX as shown, pay ing close attention to polarity.
4. Connect a Terminal to COM 1 of the Boss32. Power up the Boss32. The terminal should read the new installed version.
5. Clear the FLASH using the CLEARFLASH command. See the 32Bit Software Manual.

Note: When a firmware upgrade is done, the compiled code on FLASH will not operate correctly. The source code must be downloaded, recompiled and saved for proper operation.

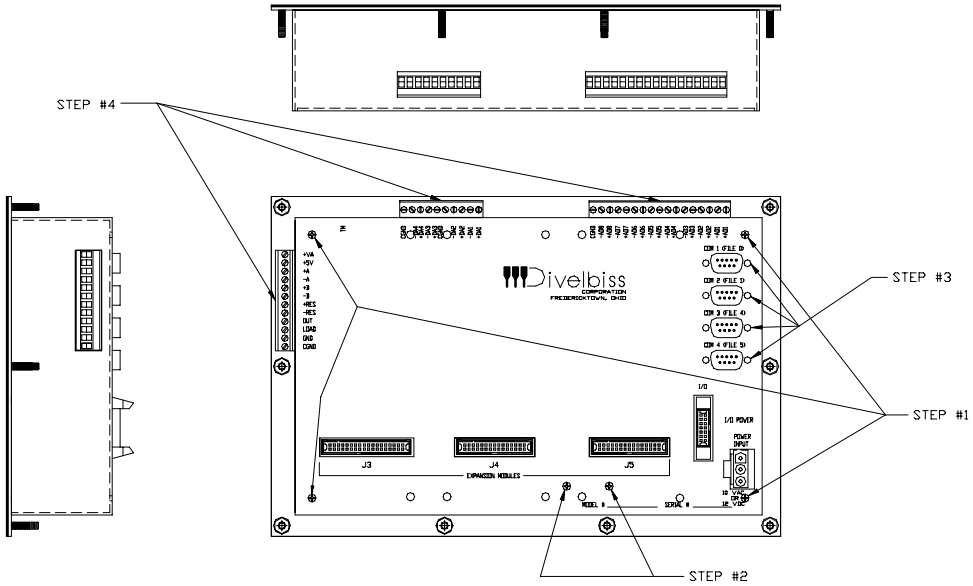


Upgrading the Boss32 Firmware

Disassembling the Boss32

The first step in any upgrade is to disassemble the Boss32 and gain access to the mother board. To disassemble the Boss32, complete the following steps: Remove all power and external connections.

CAUTION: Damage could result if power is not removed.



Disassembly of the Boss32 Controller

1. Using a #2 Phillips Screwdriver, remove the four screws on the top of the case.
2. Using a #2 Phillips Screwdriver, remove the two beveled heatsing mounting screws.
3. Using a 3/16” nut driver, remove the eight comport standoffs on COM1 - COM4.
4. Disconnect any/all terminal strips, connectors and cables.
5. Gently slide the case off. Some finesse is required to slide the case around the connec tions.

Note: Re-assembly of the Boss32 is done in the reverse order of the steps above.

Appendix A
Error Messages

The following error messages may be displayed by Boss32 BASIC on the console terminal. If the error is associated with a line number in the BASIC program, then a line number will be displayed before the error message. When an error occurs, Boss32 BASIC returns to the compiler prompt after displaying the error message.

`*** BAD INPUT. PLEASE RE-ENTER ***`
Displayed at runtime if the data entered in response to an INPUT statement doesn't match the arguments given. Just retype the input data properly. This error can be avoided by using the FINPUT statement.

`Data Statement Does Not Match Read`
Occurs when a READ or RESTORE is encountered, but no DATA statements have been found. All DATA statements must be before the first READ.

`Expression Error`
Occurs when a mathematical expression has been formatted incorrectly.

`Failure Programming EPROM or EEPROM`
Indicates that a byte didn't verify correctly after being programmed in the FLASH.

`File not Found`
FLASH file not found.

`File Verify Failure`
An error was detected after writing a file to the FLASH.

`Function Error`
Occurs when a program is compiled if the function had an argument in the wrong mode, or if the wrong number of arguments were given for the function. At runtime, a FUNCTION ERROR may appear if the argument to a function is out of range, for example, if the square root of a negative number is taken.

`Illegal Direct Command`
Occurs if an unknown direct command is entered.

`Illegal File Number`
On EPROM LOAD [*filenum*] command, this error occurs when the specified [*filenum*] is not present on the EPROM.

`Illegal Print/Input Format`
Is displayed at runtime if the format given in an FPRINT or FINPUT statement does not match the number of arguments in the FPRINT/FINPUT statement (if there are more things to be printed/input than there are formats), or if the format specified is not correct.

`Improper Data to INPUT Statement`
Occurs when data is being read from a file and the data doesn't match the arguments of the INPUT statement.

`Insufficient EPROM Space`
The user's EPROM is full.

`Line Number Does Not Exist`
Occurs if a line number is referenced in a GOTO, GOSUB, or IF statement and that line cannot be found.

`Line Number Error`
Occurs when an illegal line number is used. Line numbers must be integers from 1 to 65,534.

`Misuse of String Expression`
Occurs when a string expression is used in a place where a real or integer expression is needed, or if a real or integer expression is used in lieu of a string.

`No Compiled Code`
Occurs when the GO command is issued but there is no compiled code to execute. If any change is made to the program, it must be recompiled.

`Not Enough Memory for Network Register`
This occurs when there is not enough variable memory to allocate for the network register. See the SETOPTION VARSIZE direct command for more information.

`Not Enough Memory to Compile Program`
This can be issued under two circumstances. First, the compiled code and variable space may be too large; try shrinking arrays, compiling with NOERR, and using subroutines to eliminate duplicate code. Second, the source code may be too large, which doesn't leave enough room for the compiler's temporary storage; try removing comments to save space.

`Not Enough Variable Memory to Compile Program`
This occurs when the variable memory needed exceeds the allocated variable memory. See the SETOPTION VARSIZE direct command for more information.

`Overflow`
Indicates that a math overflow occurred, such as divide by 0, LOG of a negative number, etc.

`PROM is incompatible with this Compiler Version`
This occurs when a program on the user's EPROM is executed on power up or from a CHAIN statement. The compiled code on the user's FLASH was created with a previous version of the Boss32 BASIC software. Clear the FLASH, load the source code, recompile the program, and save it.

`Quote or Parenthesis Mismatch`
Occurs when a program is being entered and a line with an odd number of parenthesis or double quotes is found.

`RETURN Without GOSUB`
Occurs during program execution if a RETURN is encountered when no GOSUB is active. All GOSUBs must have a corresponding RETURN.

`Statement Formed Poorly`
Occurs when a statement is entered that does not conform to the syntax rules for Boss32 BASIC.

`Statement Ordering Error`
Occurs if an INTEGER, REAL, or STRING statement appears after executable statements in the program.

`String Length Exceeded`
Occurs when a string exceeds 127 characters or a string variable exceeds the maximum size assigned to it in the STRING statement.

`String Space Exceeded`
Occurs if one line of the program requires too many string temporaries to evaluate. Try splitting the line into several simpler ones.

`String Variable Error`
Displayed when a variable is used incorrectly, for example, if a string is used as a real or integer.

`Subscript out of Range`
Occurs if the subscript of a dimensioned variable exceeds the range assigned in an INTEGER or REAL statement.

`Task Error`
can be caused by any of the following:
receiving a hardware interrupt which was vectored to a non-existent task.
trying to RUN a non-existent task.
trying to RUN TASK 0.
trying to use more than 31 TASK statements.

`Task Mismatch`
Internal task error.

`Too many FOR statements`
More than 255 FOR statements found.

`Too Many Variables`
More than 128 variable declarations found.

`Undefined error`
Internal compiler error.

`Undefined Variable`
Occurs during compilation if a variable is encountered which was not defined in an INTEGER, REAL or STRING statement.

`Unmatched FOR..NEXT Pair`

Occurs during execution of a program if a NEXT is found without a FOR.

`Unrecognizable Statement`

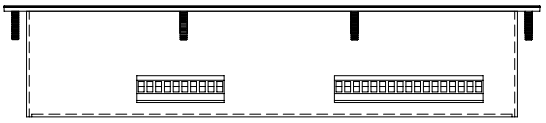
Occurs when a statement is entered that does not conform to the syntax rules for Boss32 BASIC.

Appendix B
ASCII Character Table

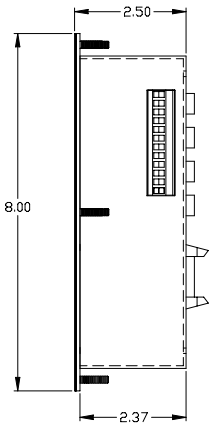
Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	NUL	32	20		64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	“	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	‘	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

ASCII, which is an acronym for American Standard Code for Information Interchange, is one of the main ways in which computers process text data. Each printable character is represented as a number between 32 and 126 ($20 and $7E). The numbers between 0 and 31 ($00 and $1F) form the “control codes”, such as carriage return (“CR”), backspace (“BS”), and horizontal tab (“HT”).

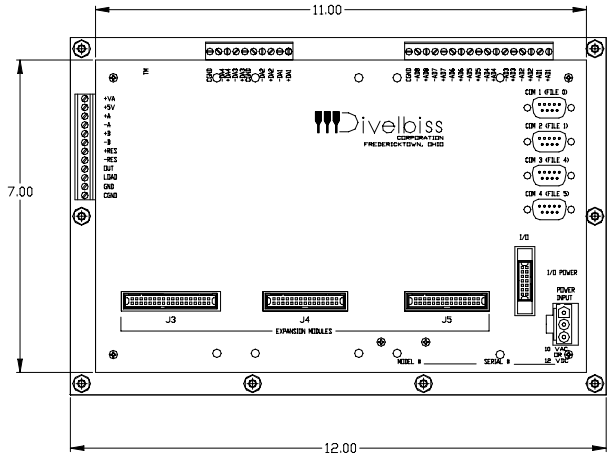
Appendix C
Dimensions



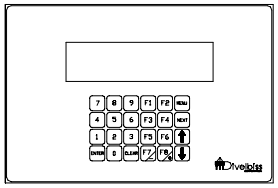
TOP VIEW
6"=1'-0"



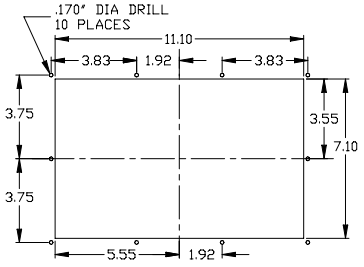
SIDE VIEW
6"=1'-0"



REAR VIEW
6"=1'-0"

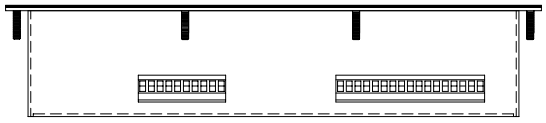


LARGE DISPLAY
FRONT VIEW
3"=1'-0"

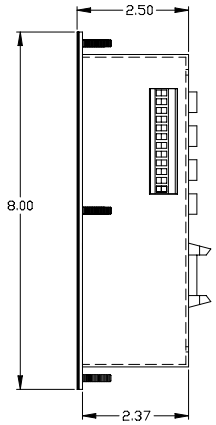


CUT OUT FOR MOUNTING
3"=1'-0"

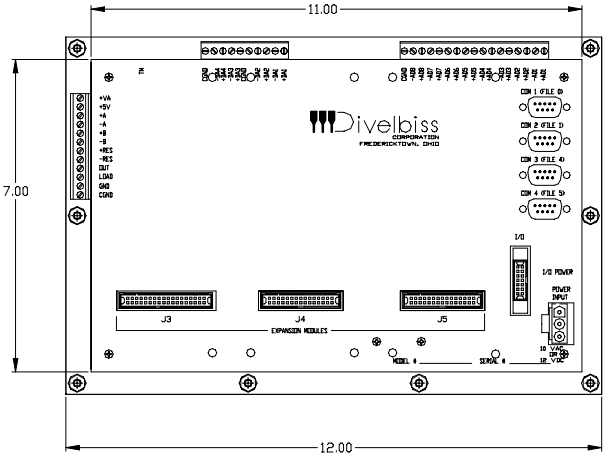
Boss32 with Graphics Display - Dimensions



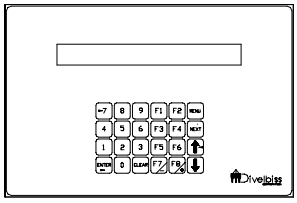
TOP VIEW
6'=1'-0"



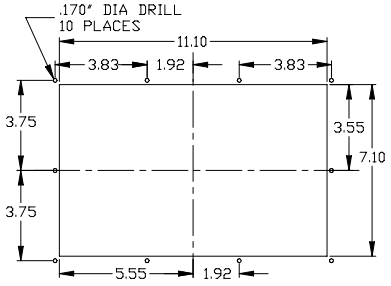
SIDE VIEW
6'=1'-0"



REAR VIEW
6'=1'-0"

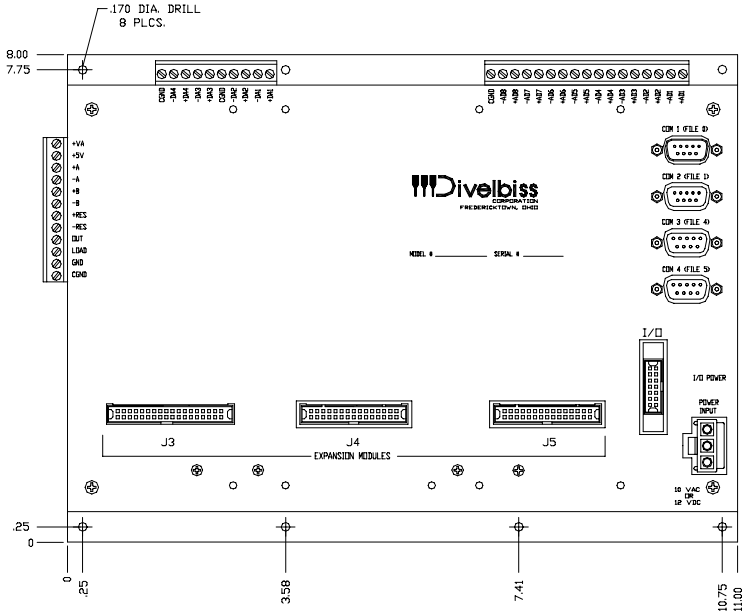


SMALL DISPLAY
FRONT VIEW
3'=1'-0"



CUT OUT FOR MOUNTING
3'=1'-0"

Boss32 with 2x40 VFD Display - Dimensions



Boss32 Subplate Mount - Dimensions

Appendix D
Specifications

Hardware

Processor:	Z380 @ 14.7456 MHz
System PROMs:	256K/512K
System RAM:	256K/1Meg.
System NVRAM:	Battery backed up: 64K/256K/1Meg.
System EEPROM:	nonvolatile storage: 8K/32K
User FLASH EPROM:	256K/512K/1Meg.
Serial Port 1:	RS232 levels 2400bps to 57.6Kbps 5,6,7,8 Data Bits 1 or 2 Stop Bits Non, Even, or Odd Parity 16 Byte FIFO XON/XOFF& RTS/CTS Handshaking
Serial Port 2:	RS232 levels 2400bps to 57.6Kbps 5,6,7,8 Data Bits 1 or 2 Stop Bits Non, Even, or Odd Parity 16 Byte FIFO XON/XOFF Handshaking RTS/CTS Handshaking
Serial Port 3:	RS232, RS422, or RS485 levels 2400bps to 57.6Kbps 5,6,7,8 Data Bits 1 or 2 Stop Bits Non, Even, or Odd Parity 16 Byte FIFO XON/XOFF RS485 supports network of up to 32 nodes without repeater Leakage +/- 1 μA Dynamic Impedance: 1.6M @ 1KHz, 1.6K @ 1000KHz
Serial Port 4:	RS232, RS422, or RS485 levels 2400bps to 57.6Kbps 5,6,7,8 Data Bits 1 or 2 Stop Bits Non, Even, or Odd Parity XON/XOFF Handshaking 16 Byte FIFO RS485 supports network of up to 32 nodes without repeater Leakage +/- 1 A Dynamic Impedance: 1.6M @ 1KHz, 1.6K @ 1000KHz
A/D:	8 Channels 12 bit Resolution 0-5VDC or 0-20mA Input

	Approx. 100 microseconds per BASIC conversion
D/A:	4 Channels 12 bit Resolution 0-10VDC or 0-20mA Output
Counter:	Single Channel 24 bit Binary up/down Counter Support for Quadrature X1 & X4 Mode Opto-isolated Input Circuitry High Speed Open Collector Direct Output Sensor Power Supply: 5VDC regulated or 12VDC @ 100mA unregulated
Display:	Vacuum Fluorescent: 2 Lines by 40 Columns or Vacuum Fluorescent Graphics: 8 Lines by 42 Columns / 256 by 64 pixels
Keyboard:	24 Keys Mechanical or Membrane
Discrete Input/Output:	Up to 128 Inputs and 128 Outputs using standard Divelbiss Bear Bones I/O panels, which are available in a variety of configurations
Real Time Clock:	Battery Backed up (Year, Month, Day, Hour, Minute, Second)
Expansion Connectors:	3 Expansion Bus Connectors allow optional modules.
Operating Temperature:	0-60 degrees C (2x40 Vacuum Fluorescent Display model) 0-50 degrees C (8x42 Graphics Display model)
Power Requirements:	Input Voltage: 115VAC, 10 VAC or 12VDC

	Max Primary	*Max Secondary*	*DC Only*
Model	I@120VAC	I@10VAC	I@12VDC
2x40 VFD	160mAAC	670mAAC	700mADC
8x42 GVFD	210mAAC	1.75AAC	1.85ADC
No Display	130mAAC	400mAAC	425mADC

Size:	Subplate Mount: 8” x 11” x 2” Panel Mount VFD: 8” x 12” x 2” Panel Mount GVFD: 8” x 12” x 2.5”

Bear BASIC Compiler

Language:	Compiled BASIC with enhancements
Memory Available:	200K - 968K for BASIC Sourc Code and for Compiled Code 64K - 1Meg. Variable Storage
Multitasking:	Supports up to 32 Tasks Preemptive context switcher; switches tasks from 100 to 400 times per second (software selectable)
Variables:	Up to 1024 Variables in a Program 7 Character Variable Names Supports INTEGER, REAL, and STRING types Supports one and two dimensional arrays
User Defined Functions:	Allows user to add extensions to BASIC Up to 64 User Defined Functions in program.

BOSS32 Hardware Manual Index