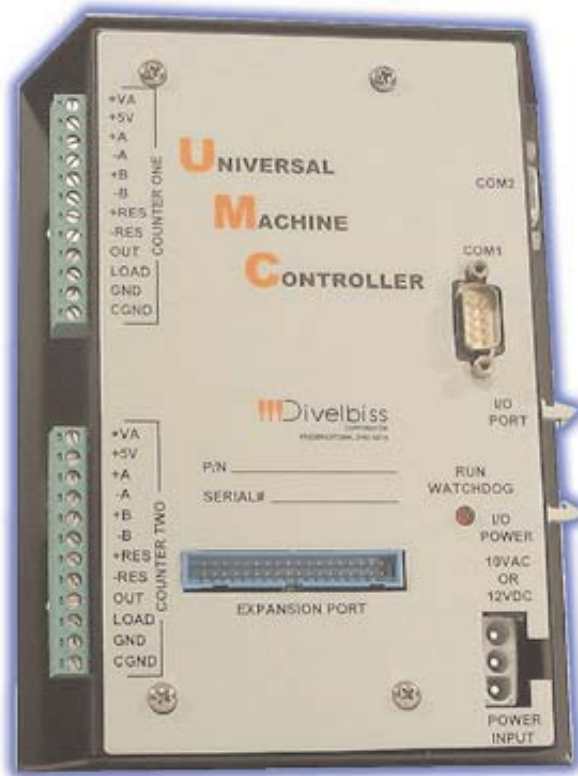




Universal Machine Controller Hardware Manual



Release Version 1.03

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

800-245-2327
www.divelbiss.com

UMC Hardware Manual

Table of Contents

Chapter One

Introduction	Page 4
Description of Features	Page 5
How to Use This Manual	Page 6

Chapter Two

Getting Started.	Page 7
Powering Up the UMC	Page 8
Connecting to a Terminal Or PC	Pages 8-9
Programming Languages	Page 9
Writing a Program in Bear Basic	Pages 9-11
Loading & Saving Programs with a PC.	Page 12
Load & Saving Programs with FLASH EPROM	Pages 12-13
Automatically Executing a Program on Power-up.	Page 13
Programming Statements	Pages 13-14
System and Direct Commands	Pages 15-16
Operators	Page 16
Functions	Pages 16-17
Accessing Help	Page 17

Chapter Three

System Information	Page 18
Determining Part Numbers.	Page 19
Using "STAT" for System Information	Pages 19-20
Dimensions and Mounting	Page 21
CPU.	Page 21
RAM	Page 21
Program Storage	Page 21
Watchdog LED	Pages 21-22
Real Time Clock.	Page 22

Chapter Four

UMC Power	Page 23
Input Power	Page 24
On-Board Power Supplies	Page 24

Chapter Five

Working With Serial Ports.	Page 25
COM1	Page 26
COM2	Pages 26-27

Chapter Six

Dual High Speed Counters.	Page 28
Counter Specifications	Page 29
Counter Input Channel Modes	Pages 29-30
High Speed Output.	Page 32
Connecting to an Encoder	Page 32
Counter Examples.	Pages 32-33

Chapter Seven

Network Connectivity	Page 34
ASCII.	Page 35
Modbus	Page 35
Network Registers	Page 35

Chapter Eight

Expansion Port.	Pages 36
Description.	Page 37
Module Installation	Page 37
High Speed Counter Module	Pages 38-40
12 Bit Analog to Digital Converter Module	Pages 40-43
10 Bit Digital to Analog Converter Module	Pages 44-47
Stepping Motor Interface Module	Pages 47-53

Chapter Nine

Real Time I/O.	Page 54
I/O Interface	Pages 55-56
Connecting I/O.	Page 56

Addressing I/O Cards Page 57

Chapter Ten

Upgrading Your UMC Page 58

Installation of COM2. Page 59

Replacing the Battery Pages 59-60

Firmware Upgrades Page 60

Disassembling the UMC. Page 61

Appendix A

Error Messages. Pages 62-65

Appendix B

ASCII Character Table Page 66

Index. Pages 67-68

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.
As used throughout this manual, ISaGRAF is a register trademark of CJ International.

Chapter One

Introduction

The Divelbiss UMC 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 UMC is fully programmable to suit the user's requirements. It supports onboard control hardware such as a high speed counter, a Real Time Clock, two serial ports, nonvolatile memory, and FLASH EPROM storage of the user's programs. One expansion port is available for adding optional modules, including high speed counters, analog inputs, analog outputs, stepper motor control, etc.

Several UMC'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 Universal Machine 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.

Description of Features

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

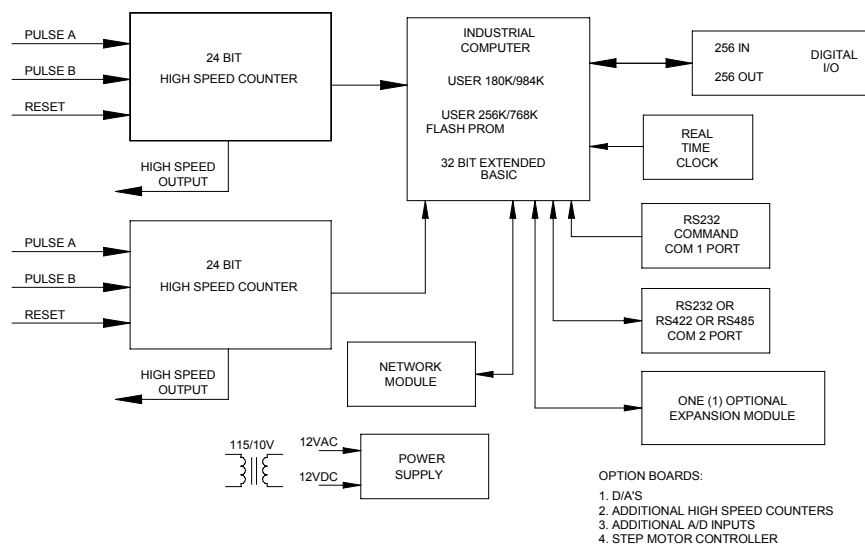
The minimal configuration of the UMC contains the microprocessor circuitry with the Bear BASIC compiler in PROM, 180K of program RAM, 256K FLASH EPROM, one serial port, a real time clock, and the HDIO (High Density I/O) interface. The program RAM is used to hold the user's program source code while it is being written, compiled code to run, and constant data and variable data while the user's program is executing. 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 UMC may be purchased with a variety of options installed, such as a high speed counter and more memory.

The dual high speed counters are 24 bit up/down counters with built-in comparators, providing a high speed output when a specified count value is reached. These counters 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 UMC is not powered.

The serial ports are configured as follows: Port 1 is RS-232; Port 2 supports RS-232, RS-422, and RS-485 serial port modules. 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 two onboard serial ports support 300, 600, 1200, 2400, 4800, 9600, 19200, 38400 and 57600 baud.



UMC 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.
<i>italic</i>	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 [<i>program_num</i>] where <i>program_num</i> is optional.

In addition, the following symbols periodically 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 UMC or its components.



This manual is for the Universal Machine Controller hardware only. For programming languages, please refer to the appropriate Programming Manuals.

Chapter Two

Getting Started

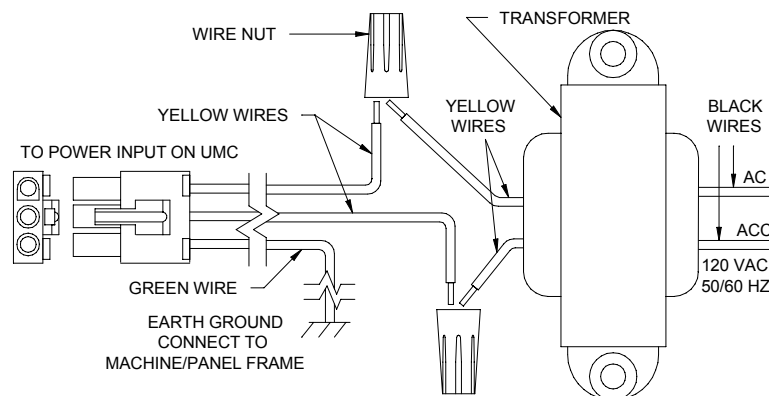
The UMC is a relatively simple computer system to get up and running; normally, even an inexperienced user can have the unit operating a short while after unpacking it. This chapter describes how to set up the UMC and use its basic features. The chapter includes sample programs that may be entered to 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 UMC

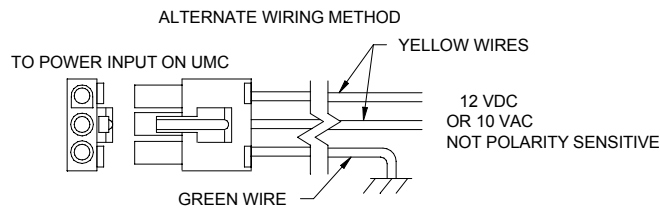
The UMC requires a 10 volt AC or 12 volt DC power source in order to operate. A transformer is available from Divelbiss that allows the unit to be powered from the 120 volt AC power line. The unit has onboard rectification and regulation to generate the voltages used inside the system. 10-12 volts is 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!



WHEN 110-120 VAC POWER IS SUPPLIED FOR UMC MAIN POWER, USE SUPPLIED TRANSFORMER AND WIRE AS SHOWN ABOVE



UMC Power Supply Wiring Schematic

Connecting to a Terminal or Personal Computer

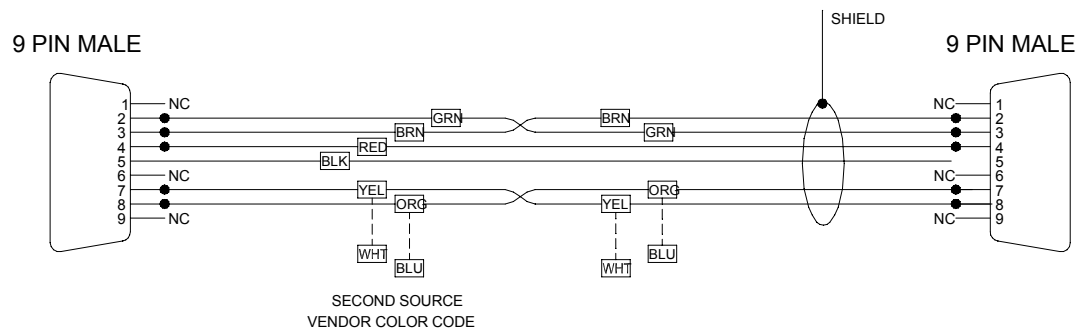
In order to program the UMC, it must be attached to a terminal or a personal computer that is running a serial communications program. This manual assumes that a personal computer is being used; the operation of the unit is the same in either case. A cable connects the UMC to the personal computer; Divelbiss 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 UMC 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 UMC. Make sure that the communications program is set to use the serial port that is connected to the UMC. Apply power to the UMC; it should respond with a sign-on message followed by the BASIC compiler prompt:

```
UMC 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 UMC to work correctly.



Serial Port COM1 Cable Wiring

Programming Languages

UMC Basic

The UMC comes standard with UMC 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. UMC Basic is a compiled and multitasking programming language.

“C” Programming

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

Writing a Program in Bear BASIC

The best way to become familiar with the UMC 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 UMC memory. Although the examples in this manual

primarily use upper case characters, the UMC 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 UMC 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 UMC 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 UMC 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 UMC, enter **NEW** to clear the current program out of memory, then enter **DOWNLOAD**. Execute the command in the communications program to send a text (or ASCII) file, and type in the appropriate file name. The file will be transferred to the UMC. 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 UMC 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 UMC 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 UMC 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 UMC 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 UMC 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 UMC is to be installed, it will probably be necessary for the UMC to automatically execute the program when it is turned on. If the SETOPTION RUN command is set to ON (run mode), then the UMC will load the last compiled code file from the Flash EPROM and execute it. Since the UMC 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 UMC 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
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 numeric user input from current FILE
INPUT\$	Allow characters 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
LOCATE	Set cursor position for current FILE device
LOCK	User defined task resource locking
NETWORK 0	Initialize UMC 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
PRINT	Send output to current FILE device
PRIORITY	Set priority of current task
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
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)

System and Direct Commands

BYE	Reset/Reboot the UMC
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 linenum	Abbreviation for EDIT
EDIT linenum	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 default 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 VARSIZE	Sets the memory size for variables
STAT	Display memory usage, hardware available, 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
SYSIN	Read digital input channel; 0 or 1
SYSOUT	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 UMC to receive an off board compiled program in the S19 file format
S19GO	Runs a downloaded S19 program
S19UPLOAD	Enables the UMC 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
ERR	Last error number generated
EXP	Exponential function e**expr
GET	Waits for one byte from current FILE
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 performed
MODREGISTER	Returns the first register number of the last Modbus function 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 UMC 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.

```
UMC Compiler Version X.XX  
>HELP
```

or

```
>HELP [command]
```

Chapter Three

System Information

The UMC is a versatile CPU for any control package. The UMC with its system and add on devices can tackle the toughest control needs. This chapter describes the UMC system from part numbering through additional UMC options and specifications.

Determining Part Numbers

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

		Typical Part No.	UMCB-	1	1	2
Base System	UMCB:	UMC w/Fuzzy Logic, Bear Direct and Modules				
Memory	0:	256K RAM/256K Flash Program Storage				
	1:	1MB RAM/640K Flash Program Storage				
Dual Counters	0:	No				
	1:	Yes				
COM2 Serial Port	0:	None				
	1:	RS232				
	2:	RS422				
	3:	RS485				

Using “STAT” for System Information

The UMC current hardware and software status can be accessed by typing **STAT** at the command prompt followed by a return. The UMC 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
RESET outputs on program termination is OFF
TRACEON file number is set to    0
                                   COUNTER          PRESENT
                                   REAL TIME CLOCK  PRESENT
EXPANSION MODULES:              J3 -          OPEN
                                   UCS -           OPEN
    
```

UMC BASIC Compiler Version X.XX

Status Memory Descriptions

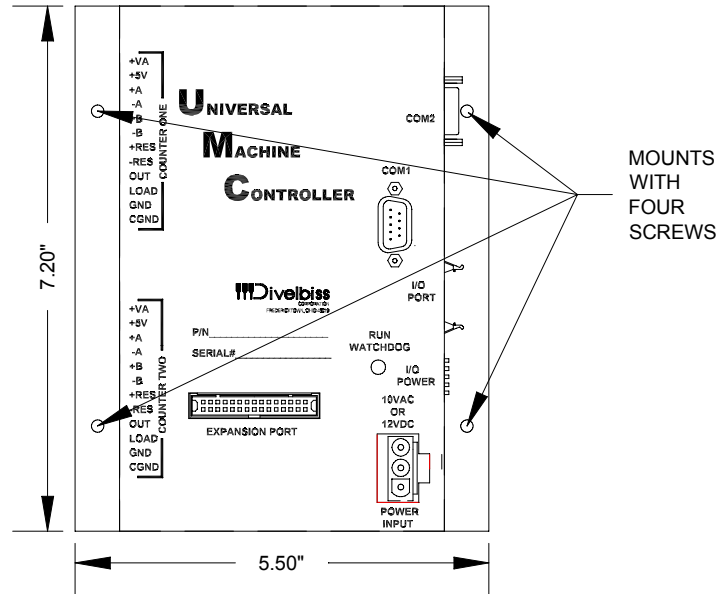
Code Start: XXXXXXXX	Starting memory location of the program.
Code End: XXXXXXXX	Ending memory location of the program
Variable Start: XXXXXXXX	Starting memory location of the program variables.
Variable End: XXXXXXXX	Ending memory location of the program variables.
XXXX Variable Bytes Used	Number of bytes used for program variables
XXXX Variable Bytes Free	Number of bytes available for program variables
XXXX Temporary & Constant Bytes Used	
XXXX Source Bytes Used	Number of bytes used by program source code.
XXXX Code Bytes Free	Number of bytes available for use.
XXXX Runtime Bytes Used	Number of Runtime bytes used by program
XXXX FLASH EPROM Bytes	Number of bytes used on FLASH EPROM Used
XXXX 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	ON, OFF: Run program on power up
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
Counter	PRESENT, ABSENT: On board counters installed.
Real Time Clock	PRESENT, ABSENT: Real Time Clock installed.
EXPANSION MODULES: J3 -	Type of Expansion module installed. No module returns OPEN
UCS -	Type of UCS Expansion module installed. No module returns OPEN

Dimensions and Mounting

The UMC is encased by rugged painted steel subplate mount case. The UMC size is 7.2" X 5.5" X 2". The UMC mounts easily onto any subplate with 4 screws.



CPU

The UMC processor is a Zilog Z382. The Zilog Z382 is a full 32 bit processor and is running at 14.7456 Mhz.

RAM

The UMC comes standard with 180K of non-volatile program RAM and can be ordered with up to 948K of non-volatile program RAM. The Non-volatile memory holds the users program source code, the compiled code to run, the constant and variable data while the user's program is executing. The memory contents are held by a internal lithium battery which should provide many years of problem free operation.

Program Storage

The UMC stores programs on FLASH EPROM. The FLASH EPROM gives the programmer the ability 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, LOAD-CODE, SAVE, SAVECODE, and CHAIN.

The UMC comes standard with 256K FLASH EPROM space. The UMC can be ordered with up to 768K FLASH EPROM.

Watchdog LED

The Watchdog LED is an optical device to view the status of the UMC. The Watchdog will blink approximately once per second if the UMC has booted properly (the UMC is waiting at



a command prompt). If the UMC is executing a program, the Watchdog LED will blink approximately 4 times per second. Note: If the watchdog LED does not blink at all, this may signify the UMC is not operating correctly.

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 UMC is not powered. The UMC 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, GETTIME, and SETIME.

Chapter Four

UMC Power

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

Input Power

Specifications:

Input Power	Min (I)	Typical (I)	Maximum (I)
10VAC or 12VDC	442mA AC	445mA AC	2 A AC

120VAC may be used in conjunction with a transformer. Transformer should be rated 2A with 120VAC input, 10VAC output.

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 UMC provides on board power supplies for driving off unit devices. The UMC 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 attached to the UMC will use power from these supplies.

Typical UMC Power Consumption:

Power Supply	Power Consumption
INPUT POWER	445 mA AC
12VDC (+VA)	0
5VDC	165 mA DC

Output Power Specifications

Power Pin	Min	Maximum
+VA	0	800 mA DC
5VDC	0	800 mA DC

Chapter Five

Working With Serial Ports

Interfacing into other devices is a requirement in todays control applications. The Universal Machine Controller has the ability to connect to other "smart" devices via serial ports. The UMC is available with up to to 2 serial ports and supports baud rates from 300 baud to 57.6K baud. The UMC supports RS232, RS422 or RS485. The UMC supports ASCII transfer and can be a modbus master or slave.

Many devices interface with other equipment using serial data transfer. The UMC provides up to two serial ports. All ports support asynchronous serial transfer at baud rates between 300 and 57600 baud.

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 UMC. 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.

<u>Pin</u>	<u>ID</u>	<u>Description</u>
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	RI	Ring Indicator

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

<u>Part No.</u>	<u>Description</u>
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 any one of RS-232, RS-422, and RS-485 serial port modules. 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.

<u>RS-422</u>	<u>Pin</u>	<u>ID</u>	<u>Description</u>
	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	<u>Pin</u>	<u>ID</u>	<u>Description</u>
	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	<u>Pin</u>	<u>ID</u>	<u>Description</u>
	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 (+)

To set the operating mode for **COM2**, the **FILE** statement is used. See the **FILE** and **SERIALDIR** statements for information.

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

For RS422 or RS485, module pins 7 and 8 must be shorted together.



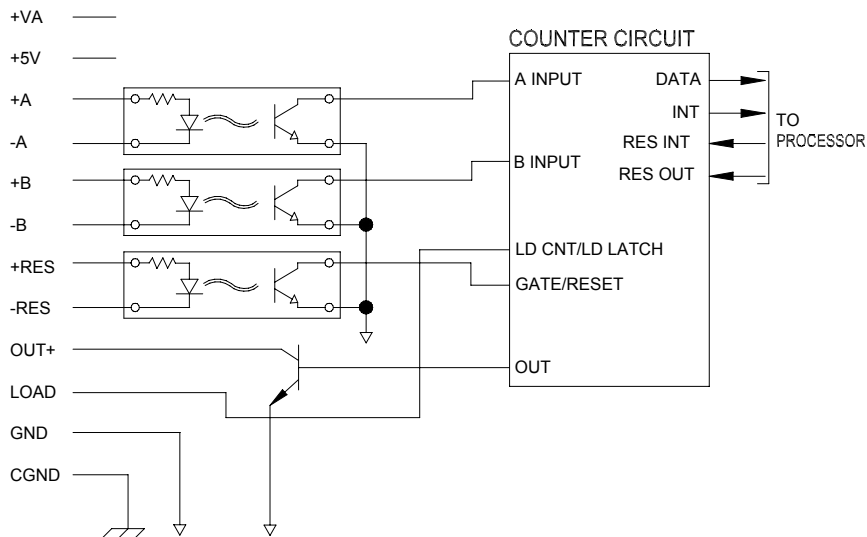
Chapter Six

Dual High Speed Counter

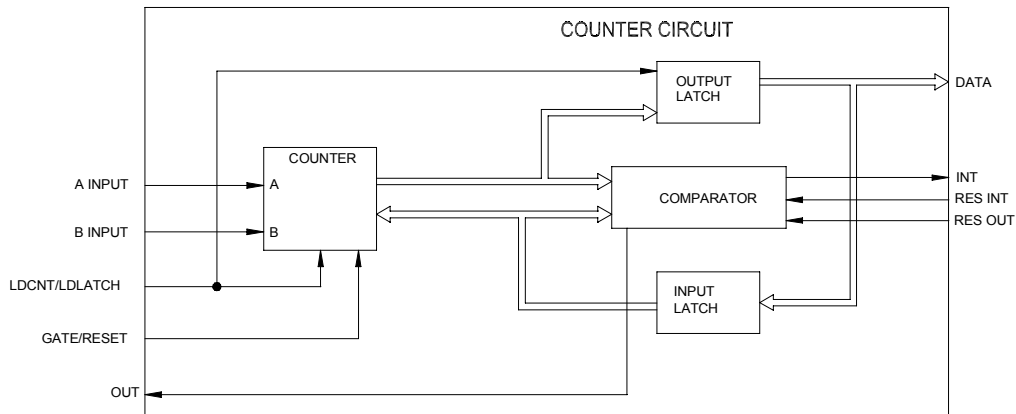
Counting is a very common operation in real time control systems. For low speed signals (less than 10 pulses/second), this can be handled easily in software. At higher rates, however, hardware counters are required. The UMC dual high speed counter circuit is very flexible, providing several operating modes. They are 24 bit binary up/down counters, capable of greater than 75 kHz count rates (18 kHz in quadrature modes), with a high speed output. They support 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 biphas encoders. The count value changes once for each biphas cycle.
2. The **A** and **B** inputs are used in quadrature X4 mode, for use with biphas encoders. The count value changes with each input transition; X4 mode counts four times faster than X1 mode, given the same input signal.

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/disable counter.
6. X4 quadrature mode. 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 comparator 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.

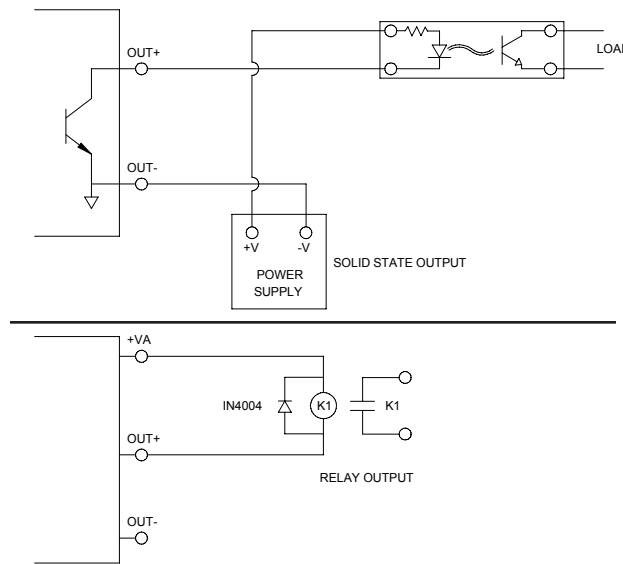


NOTE: The maximum voltage to be applied to any counter input is 5VDC.

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 of the High Speed Output is 100mA DC.

High Speed Output to optically isolated device.

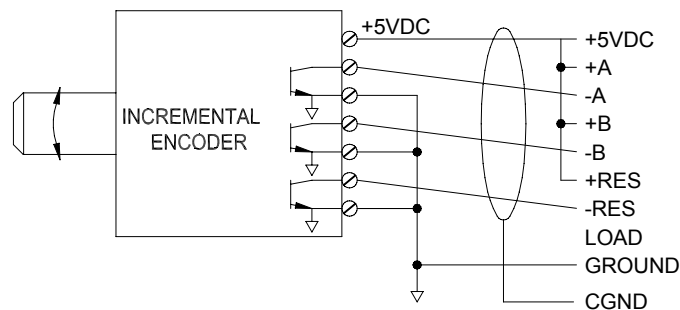


High Speed Output controlling a dry contact.



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

Network Connectivity

One of the forthcoming requirements in today's control world is the ability to network control systems together. The Universal Machine Controller provides network connectivity with Profibus, DeviceNet, Modbus and Divelbiss' own Bear Direct Network.

ASCII

The UMC can host ASCII serial communication up to 57.6K baud. The serial port is a standard RS232 protocol. By utilizing commands, ASCII data can be formatted, sent and received via the serial port.

Modbus

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

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 UMC supports the network using a set of network registers for Modbus. These registers can be accessed by both the BASIC program and the network driver software, independently of each other. In Bear BASIC, the registers are written to with the NETWORK 3 statement; they are read with the NETWORK 4 statement. There are three types of network registers: integer, real, and string. A network register is NOT automatically transferred across the network when the BASIC program writes to that register; the register must be explicitly requested by the “master” on the network. The network registers can be thought of as an intermediate storage area between BASIC and the actual network handler; they relieve the programmer from the having to worry about handling network messages at the moment that they come in.

The network register method is well suited to situations where one unit needs to monitor the current state of another unit. For example, suppose that a PC is used to display the current production totals for the machines on the network. Each UMC will update its network registers as items are produced; the PC will read this data every few seconds and display it. Notice that there is no need in this example for the UMC to send a specific piece of data (ie. production data for a particular part) to the PC. This is more difficult using network registers, because a register must be used as a flag to indicate that this specific data is available. The example below demonstrates this operation.

The NETWORK 3 and NETWORK 4 statements write into and read out of network registers, respectively. This happens independently of network accesses to the network registers, so that the BASIC program could be writing data into a network register while the network master is reading data out of a network register. Each individual register access is protected, however, so that a register won't be read halfway through a write cycle; in other words, once data is being written into a register, that register won't be read until the data is entirely written.

Chapter Eight

Expansion Port

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

Description

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

The UMC 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.

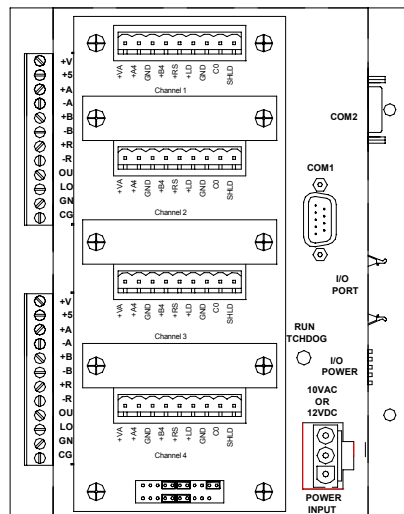
Module Installation



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

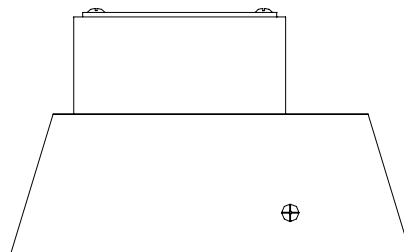
The UMC expansion modules are secured to the UMC with four 6-32 machine screws, each 1 5/8 inch long. The module is installed into the Expansion Port connector.

The Expansion Port on the UMC 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. Press the module firmly into place. Install the long screws into the corners of the module. If these screws will not thread into the UMC, then the module is mis-mated and must be removed and re-installed properly.



Top View

Universal Machine Controller
with Expansion Module
installed



Side View

High Speed Counter Module

The high speed counter module extends the UMC beyond the dual counter channels that can be installed onboard; by using one module in addition to the dual on-board counters, the UMC can support up to 6 counter channels. The counter module is available in four models:

<u>Divelbiss Part Number</u>	<u>Description</u>
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 UMC, 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 UMC 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 UMC digital common. It is not internally connected to earth ground, and should not be connected externally to earth ground.

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 ($\pm 20\%$), 5kHz ($\pm 20\%$), 100kHz ($\pm 20\%$).
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 UMC. 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:

<u>Divelbiss Part Number</u>	<u>Description</u>
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

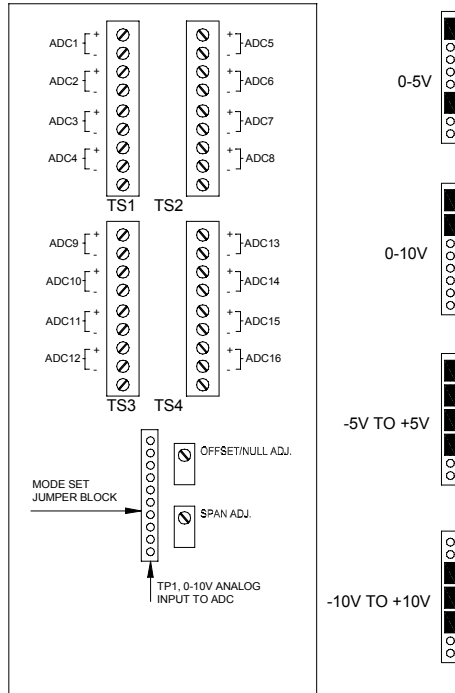
For best noise immunity, shielded cable should be used between the analog inputs and the device being monitored; this is especially important for long cable runs. Except in cases where the analog signals are very dynamic, all channels can be sheathed in the same cable, with one common overall shield. 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 the A/D module; of course, separate returns are required when using the differential and 4-20mA modules.



12 bit A/D Module Jumper Configuration

ALL CHANNELS ARE SELECTED FOR THE SAME INPUT SPAN SIMULTANEOUSLY.

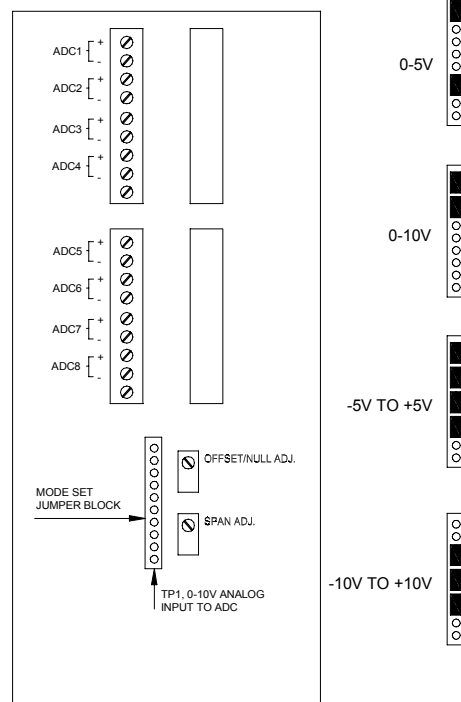
■ =PROGRAMMING SHUNT INSTALLED



EX-MOD-AD12-S

ALL CHANNELS ARE SELECTED FOR THE SAME INPUT SPAN SIMULTANEOUSLY.

■ =PROGRAMMING SHUNT INSTALLED



THE EX-MOD-AD12-C (4mA-20mA INPUT MODEL) IS FACTORY SET AND CANNOT BE CHANGED BY THE USER.

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 UMC 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 UMC 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, chan1
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",chan1

'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 ";chan1;
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(chan1)/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 $\frac{1}{2}$ 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
	EX-MOD-AD12-S	2.3M min.
	EX-MOD-AD12-D	3.3M min.
	EX-MOD-AD12-C	250W, $\pm 2\%$
		Channel tracking
		N/A
		N/A
		$\pm 0.2\%$, 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:	600 μ sec, using the ADC() function of UMC BASIC.	

10 Bit Digital to Analog Converter Module

The 10 bit D/A converter module provides analog output capabilities to the UMC. Using this module, up to 4 analog outputs can be added to a UMC. 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:

<u>Divelbiss Part Number</u>	<u>Description</u>
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

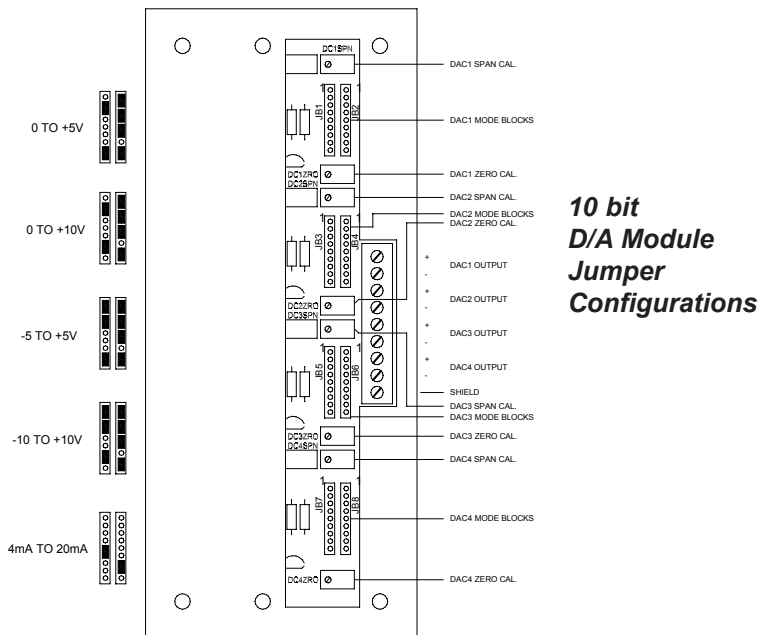
In the 4 to 20mA mode, this unit implements a “true” current loop; it is not referenced to any common. This is the best D/A drive to use for optimum immunity to noise and cable loss. The output current is regulated and can maintain the chosen analog output over a wide range of load impedance.

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 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 UMC 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 UMC 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):

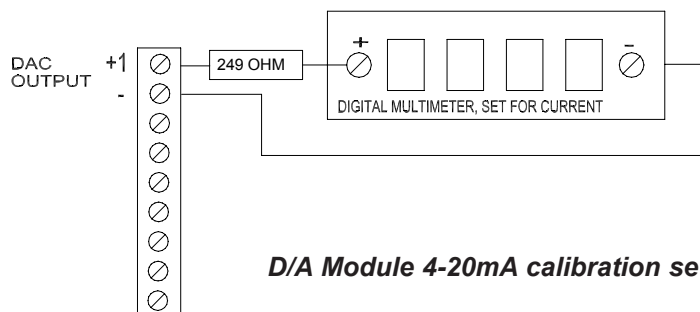
With the BASIC D/A calibration program running, select the desired channel to calibrate, and specify 0 as the DAC output value. Adjust the proper zero potentiometer for 0.00V on the output terminals of the channel being calibrated. Next, specify 2.5 or 5.0 (depending upon the range) as the DAC output 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 $\pm 5V$ range, or 9.99V for the $\pm 10V$ 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 multimeter should read 12.0mA.



Specifications

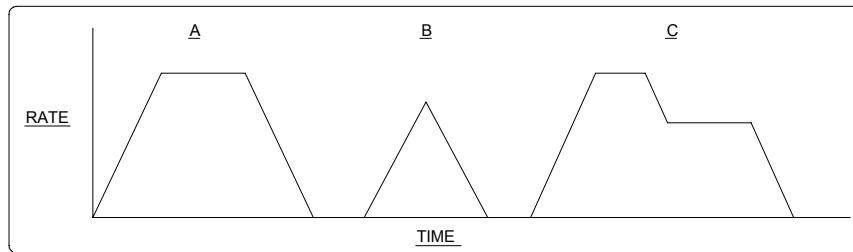
Channels:	1, 2, 3, or 4, depending upon part number
Resolution:	10 bits
Output level:	Each channel is user selectable: 0 to +5V, 0 to +10V, -5 to +5, -10 to +10V, or 4 to 20mA.
Output null/offset adjustments:	$\pm 250\text{mV}$, or 0-6mA in current (4 to 20mA) mode.
Output linearity error:	$\pm 0.05\%$ 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 , $\pm 20\%$.
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

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

The Dixelbiss Stepping Motor Interface is an expansion module for use with the UMC; 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 UMC 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 ramps 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 UMC 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.

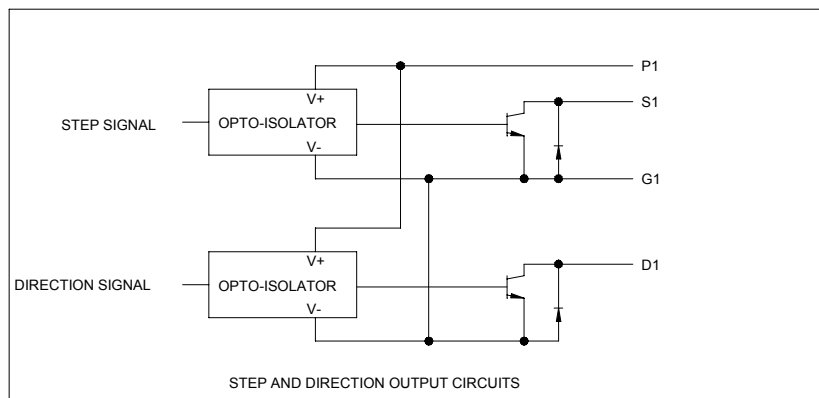
The module is housed in an aluminum enclosure; it bolts to the UMC with the mounting screws that are provided with the module.



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.

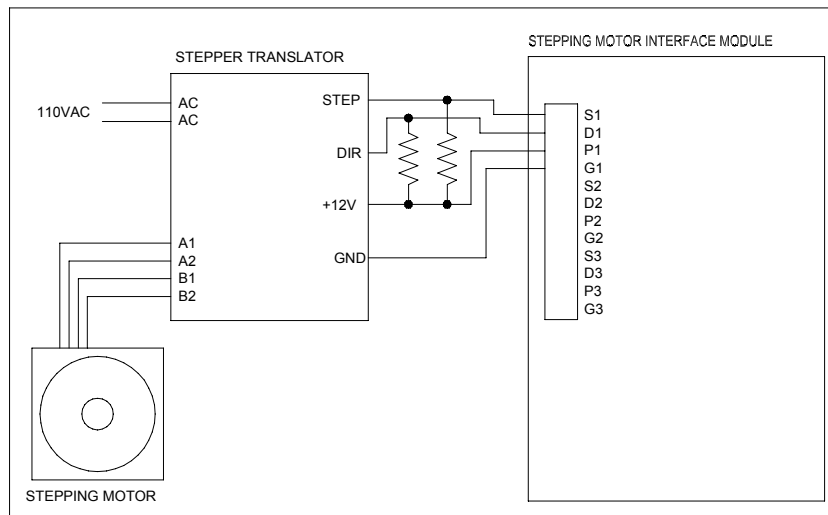


Output Circuit



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

- 1 Controls up to three stepping motors
- 1 Contains a dedicated microprocessor, allowing stepping motors to be controlled without burdening the UMC processor
- 1 Supports step rates from 20 to 10000 pulses/sec
- 1 Provides linear acceleration and deceleration ramps
- 1 Motors can run independently, or one channel can output a pulse rate that is proportional to another channel's pulse rate
- 1 Optically isolated, open collector step and direction outputs are provided for each channel
- 1 Supports separate acceleration and deceleration rates for each motor
- 1 An interrupt can be sent to the UMC 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
channel1 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
channel1 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
channel1 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
channel1 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.
!
!*****
2100 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.
!
!*****
2130 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.
!
!*****
2110 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 Nine

Real Time I/O

The UMC I/O Bus allows up to 256 inputs and 256 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 UMC 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			DIN/DOUT			DIN/DOUT			DIN/DOUT		
Addr	Hex	Dec.	Addr	Hex	Dec.	Addr	Hex	Dec.	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

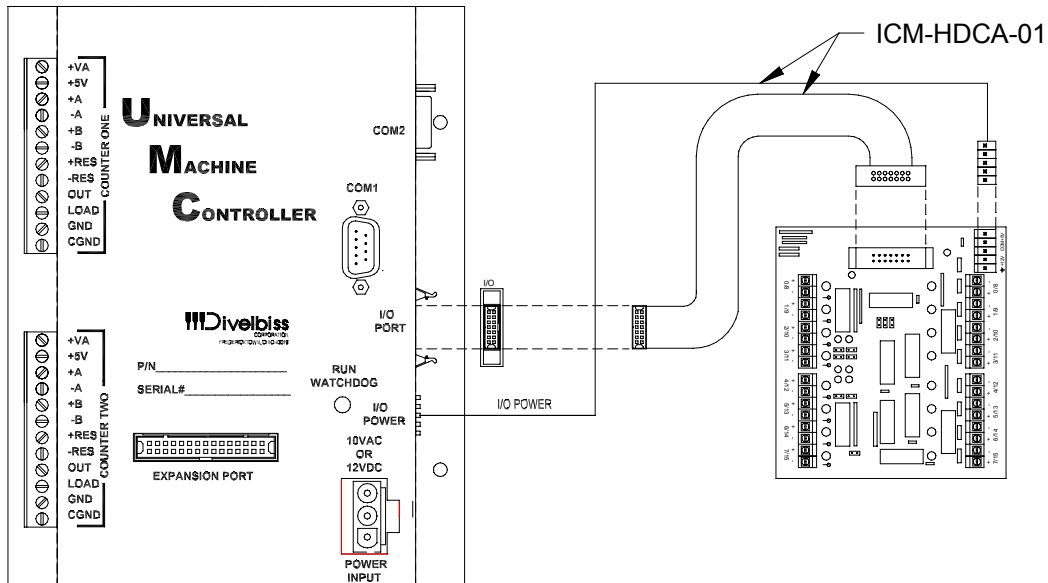
The Addr column shows the Expander I/O address; the Hex column shows the UMC I/O number in hexadecimal form, while the Dec. column shows it in decimal form.

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



Connecting I/O

The UMC communicates to I/O cards via a ribbon cable connected to the I/O port connector on the UMC. The I/O cards receive their DC power for operation from a power cable connected to the UMC. 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 degradation of signals and produce random I/O occurrences 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 upper 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 determining 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 U L Upper U L	DIN/ DOUT		Card Page Address	Card Paging Shunts (Address Selector) 8 4 2 1	U/L Selector Lower U L Upper U L	DIN/ DOUT	
			8 I/O Cards	16 I/O Cards				8 I/O Cards	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

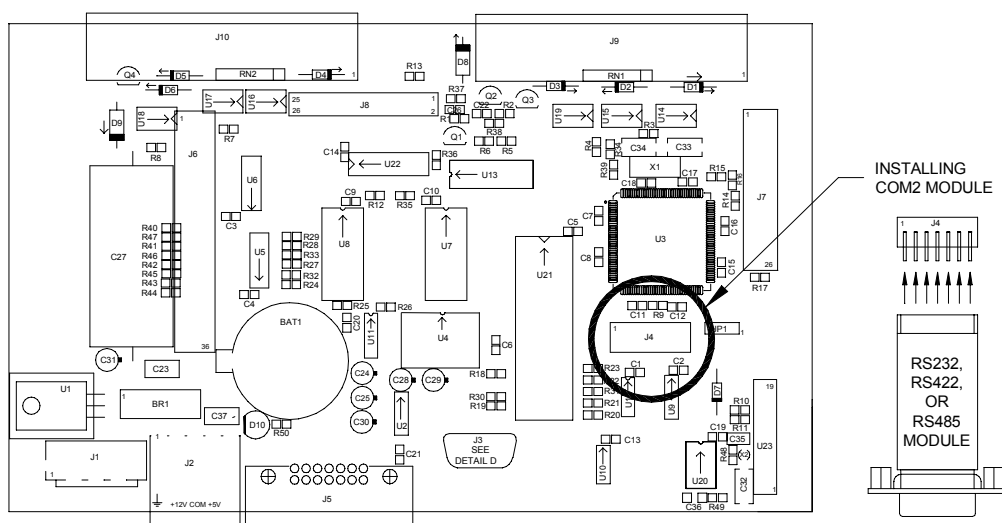
Chapter Ten

Upgrading Your UMC

Installation of COM2

The UMC has the capability of accepting serial port modules for RS232, RS422 or RS485. Any one of these modules may be installed as COM2. These modules support baud rates from 300 to 57.6K. To install a serial port module, complete the steps outlined below. Consult factory for module availability.

1. Disassemble UMC. See *Disassembling the UMC*, page 61.
2. Gently slide serial port module as shown onto J4. The serial port should be upside down and match the cutout in the case.
3. Reassemble the UMC. Some finesse may be required to install the case and seat it properly with the added comport.
4. Restart the UMC. The serial port should now be active on File 5.

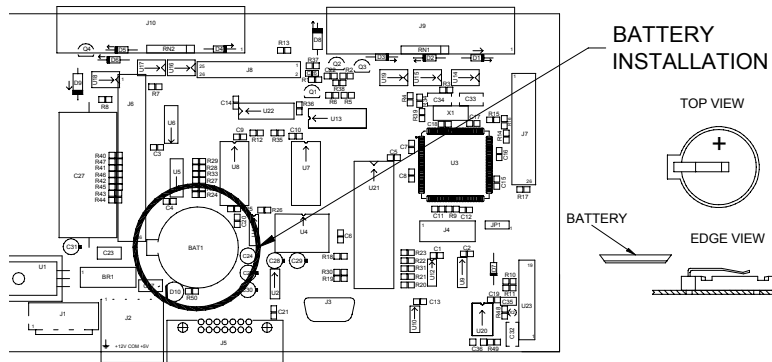


Installing COM2

Replacing the Battery

The UMC 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. Disassemble UMC. See *Disassembling the UMC*
2. Paying attention to polarity, carefully slide the coin battery from its socket labeled BAT1.
3. Carefully slide the new battery into the socket paying attention to the proper polarity.
4. Reassemble the UMC.



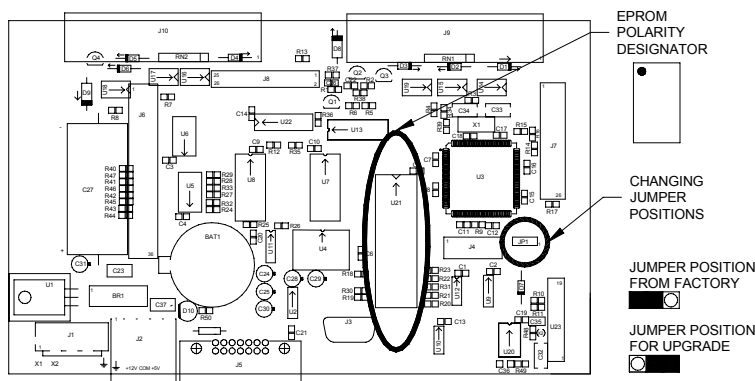
Battery Installation

Firmware Upgrades

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

1. Disassemble UMC. See Disassembling the UMC
2. Install the firmware upgrade EPROM into U21 as shown, paying close attention to polarity.
3. Connect a Terminal to COM 1 to view messages. See Connecting to a Terminal or PC.
4. Change jumper position of JP1 as shown.
5. Power up the UMC. The UMC should automatically download and install the new firmware.
6. Power down the UMC when prompted on the terminal. Remove the upgrade firmware EPROM and return JP1 to its original settings.

Note: When a firmware upgrade is done, the compiled code on FLASH must be downloaded, recompiled and saved for proper operation.



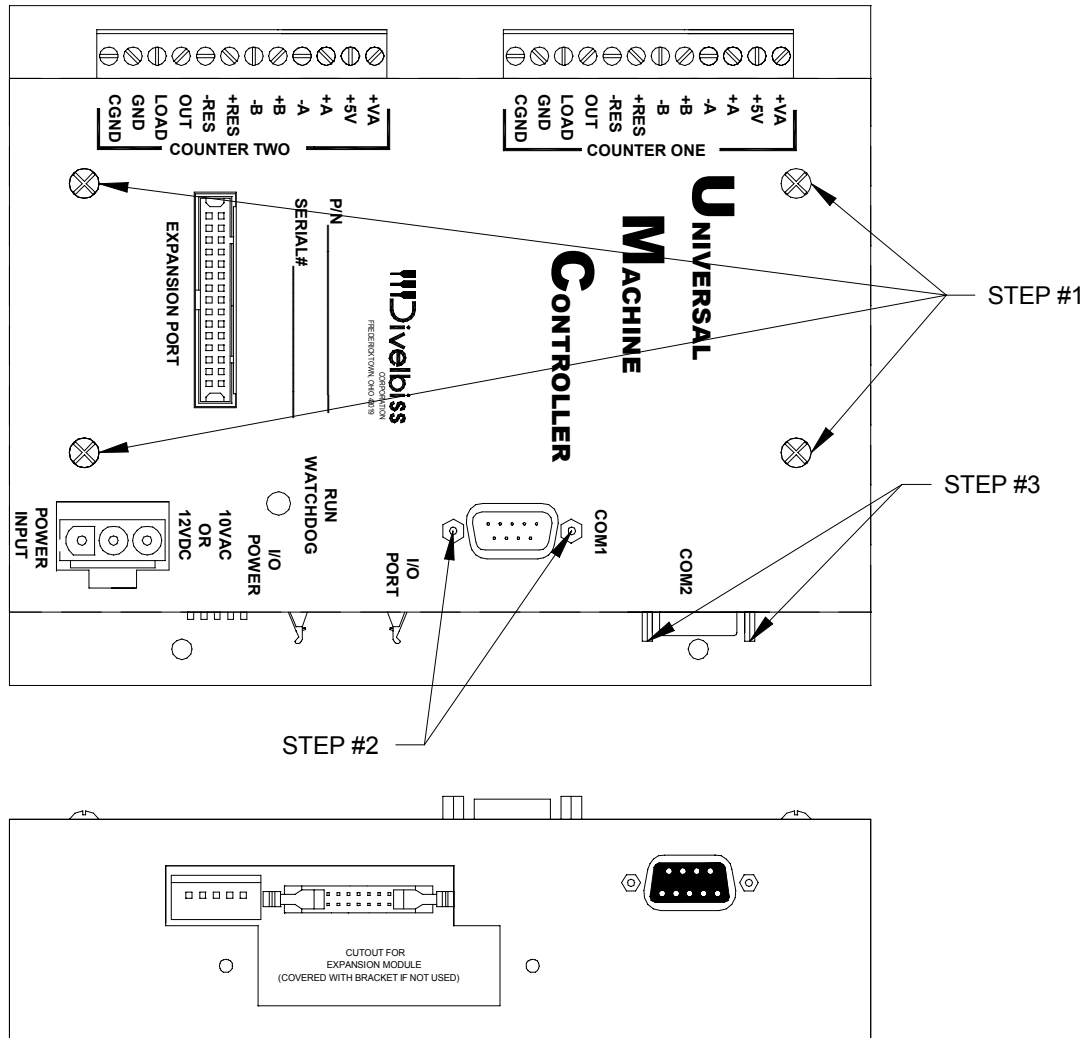
Upgrading Your Firmware

Disassembling the UMC

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



CAUTION: Damage could result if power is not removed.



Disassembly of the Universal Machine Controller

1. Using a #2 Phillips Screwdriver, remove the four screws on the top of the case.
2. Using a 3/16" nut driver, remove the two comport standoffs on COM1.
3. If installed, remove the two 3/16" comport standoffs on COM2.
4. Gently slide the case off. Some finesse may be required if COM2 is installed.

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

Appendix A

Error Messages

The following error messages may be displayed by UMC 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, UMC 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 32,767.

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 UMC 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 UMC 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 UMC 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	0	79	4F	O	111	6F	o
16	10	DLE	48	30	1	80	50	P	112	70	p
17	11	DC1	49	31	2	81	51	Q	113	71	q
18	12	DC2	50	32	3	82	52	R	114	72	r
19	13	DC3	51	33	4	83	53	S	115	73	s
20	14	DC4	52	34	5	84	54	T	116	74	t
21	15	NAK	53	35	6	85	55	U	117	75	u
22	16	SYN	54	36	7	86	56	V	118	76	v
23	17	ETB	55	37	8	87	57	W	119	77	w
24	18	CAN	56	38	9	88	58	X	120	78	x
25	19	EM	57	39	:	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”).

A

Accessing Help: 17
Addressing I/O Cards: 57
ASCII: 35
ASCIICharacter Table: 68
Automatically Executing a Program on Power Up: 12

C

Connecting I/O: 56
Connecting to a Terminal or Personal Computer: 8
Connecting to an Encoder: 31
Counter Examples: 32
Counter Input Channel Modes: 29
Counter Specifications: 29
CPU: 21

D

Description of Features: 5
Determining Part Numbers: 19
DeviceNet: 35
Dimensions and Mounting: 21
Disassembling the UMC: 61
Dual High Speed Counter: 28

E

Error Messages: 62
Expansion Port: 37
Description: 38

F

Firmware Upgrades: 60
Functions: 16

G

Getting Started: 7

H

High Speed Counter Module: 39
High Speed Output: 31
How to Use this Manual: 6

I

I/O Interface: 55
Input Power: 24
Installation of COM2: 59
Introduction: 4

L

Loading and Saving Programs With a Personal Computer: 11
Loading and Saving Programs With FLASH EPROM: 12

M

Modbus: 35

N

Network Connectivity: 34
Network Registers: 35

O

On-Board Power Supplies: 24
Operators: 15

P

Powering Up the UMC: 8
Profibus: 35
Program Storage: 21
Programming Languages: 9
"C" Programming: 9
UMC Basic: 9
Programming Statements: 13

R

RAM: 21
Real Time Clock: 22
Real Time I/O: 54
Replacing the Battery: 59

S

Stepping Motor Interface Module: 48
System and Direct Commands: 14
System Information: 18

T

Table of Contents: 1
10 Bit Digital to Analog Converter Module: 45
10 bit D/A Module Calibration: 46
10 bit D/A Module Jumper Configuration: 45
12 Bit Analog to Digital Converter Module: 41
12 bit A/D Module Calibration: 42
12 bit A/D Module Jumper Configuration: 42

U

UMC BASIC: 9

UMCPower: 23

Upgrading Your UMC: 58

disassemble the UMC: 61

firmware upgrades: 60

installed as COM2: 59

replacement batteries: 59

Using "STAT" for System Information: 19

W

Watchdog LED: 21

Working With Serial Ports: 25

COM1: 26

COM2: 26

RS-232: 26

RS-422: 26

RS-485: 27

Writing a Program in Bear BASIC: 9