D4110
User's Manual

Systems Engineering Associates, Inc.
14989 West 69th Avenue
Arvada, Colorado 80007 U.S.A.
Telephone: (303) 421-0484
Fax: (303) 421-8108

07/2001

D4110
User's Manual

Copyright © 1993 Systems Engineering Associates, Inc.

All Rights Reserved!

CONTENTS

1. General Description 1

1.1 Programming 1
1.2 Program Execution Times 1
1.3 Alphanumeric Display 1
1.4 Keypad 1
1.5 Digital Inputs 2
1.6 Interrupt Inputs 2
1.7 Digital Outputs 2
1.8 Interface Ports 3
1.9 Real Time Clock 3
1.10 Diagnostics/Fault Detection 3
1.11 LED Status Indications 4

2. General Description 5

3. System Configuration 7

3.1 Target Board 7
3.2 Network Baud Rate 7
3.3 Input0 Interrupt Enable 7
3.4 Input1 interrupt Enable 8
3.5 Fixed Scan Time Mode 8
3.6 Timed Interrupt 9

4. Variable Types/Memory Map 11

4.1 Variables 11
4.1.1 Flags (F) 11
4.1.2 Bytes (B) 11
4.1.3 Words (W) 12
4.1.4 Port-Pins (P) 12
4.1.5 Inputs (X) 13
4.1.6 Outputs (Y) 14
4.1.7 Constants 15

4.2 Data Memory Map 15
4.2.1 Volatile Data Memory 16
4.2.2 Non-Volatile (battery-backed) Data Memory 17

4.3 I/O Image Addressing 17
4.4 Special Function Variables 18
4.4.1 F104: User Port RS-$22 Mode Select 18
4.4.2 F105: Serial Network Port Select 18
4.4.3 B62 – B63: Timed Interrupt Immediate Input Variables 18

4.5 System Function Buffers 19

CONTENTS

5. Programming Reference ____ 21

5.1 Instruction Set ____ 21
5.1.1 Ladder ____ 21
5.1.2 High-Level ('C') ____ 22
5.1.3 Assembly ____ 22

5.2 System Functions ____ 23
5.2.1 System Function Types ____ 23
5.2.2 sfunc02: Current Time/Date Read ____ 24
5.2.3 sfunc03: Watchdog Timer Reset ____ 25
5.2.4 sfunc04: ASCII String Load Command ____ 26
5.2.5 sfunc07: General External Address Read ____ 27
5.2.6 sfunc08: General External Address Write ____ 28
5.2.7 sfunc09: System Fault Routine ____ 28
5.2.8 sfunc10: USER PORT Receive ____ 29
5.2.9 sfunc11: USER PORT Transmit ____ 29
5.2.10 sfunc13: Serial Network Communications ____ 30
5.2.11 sfunc18: Display Write (update) ____ 31

6. Using System Functions ____ 33

6.1 Writing (updating) the Display ____ 33
6.1.1 Writing Data to the Display (sfunc18) ____ 33
6.1.2 Display Control Codes ____ 35
6.1.3 Valid Display Characters ____ 36

6.2 Keypad Interface ____ 37
6.3 Serial Network Communications (sfunc13) ____ 38
6.3.1 Communicating on the Network (sfunc13) ____ 38

6.4 User Port Communications ____ 42
6.4.1 Receiving through the User Port (sfunc10) ____ 42
6.4.2 Transmitting through the User Port (sfunc11) ____ 43

6.5 Real Time Clock ____ 44
6.5.1 Setting the Time and Date ____ 45
6.5.2 Reading the Time and Date (sfunc02) ____ 45

7. Fault Detection ____ 47

7.1 Fault Routine Execution ____ 47
7.2 Viewing Fault Codes with SYSdev ____ 47
7.3 Fault Codes ____ 49
7.3.1 Watchdog Timer Timeout (40H and 41H) ____ 49
7.3.2 IBM PC to M4000 Communications Failure (42H) 50
7.3.3 Invalid Program Faults (5cH and 5dH) ____ 51
7.3.4 User Program sfunc09 System Fault Call (45H) 51
7.3.5 Internal D4110 Faults (43H, 44H, 52H, 59H – 5bH) ____ 51

7.4 Serial Network Communication Errors ____ 52
7.4.1 Serial Network Comm Error Codes ____ 52
7.4.2 No Response from Slave (04H and 05H) ____ 53
7.4.3 Serial Network Integrity Error (03H, 06H – 0eH, 10H) ____ 53
7.4.4 Address Outside Range (0fH) ____ 53

8. Hardware Confidence Test ____ 55

8.1 Tests Performed ____ 55
8.2 Performing the Hardware Confidence Test ____ 56
8.2.1 Equipment Required ____ 56
8.2.2 Executing the Test ____ 56

8.3 Interactive Interface ____ 57

9. Installation ____ 59

9.1 Mounting the D4110 ____ 59
9.2 Wiring Input Power ____ 59
9.3 Wiring 10-30VDC Digital Inputs ____ 60
9.4 Wiring Interrupt Inputs ____ 61
9.5 Wiring 10-30VDC Digital Outputs ____ 62
9.6 Wiring the Fault Interlock ____ 63
9.7 Serial Network Installation ____ 64
9.7.1 Wiring the Serial Network ____ 64
9.7.2 Setting the Network Addresses ____ 66

9.8 Power-up Sequence of D4110 modules ____ 66

CONTENTS

APPENDICES

Programming Example ____ Appendix A
RS-232 Pinouts/Cables ____ Appendix B
Field Wiring Connector Pinouts ____ Appendix C

LIST OF FIGURES

6.1 Keypad Key Locations ____ 37

9.1 Typical M4000 Input Wiring ____ 60
9.2 Typical Interrupt Input Wiring ____ 61
9.3 Typical Output Wiring ____ 62
9.4 Typical Fault Interlock Wiring ____ 63
9.5 Typical Network Wiring ____ 65
9.6 Alternative Serial Connector Wiring ____ 65
9.7 Recommended Panel Cut-out ____ 67

SECTION 1
GENERAL DESCRIPTION

The D4110 is a high performance programmable logic controller module which incorporates a 2-line by 40 character display and a 3 X 8 keypad. In addition, the D4110 incorporates a built- in processor, user program (24K bytes) and data memory (2K bytes), 16ea. 10-30VDC digital inputs, 16ea. 10-30VDC digital outputs, RS-232 programming port, RS-232/RS-422 USER port, two serial network interface ports, and a real time clock.

1.1 PROGRAMMING

Programming of the D4110 module is implemented using SYSdev, an IBM PC or compatible software package which allows the user to create, document, and compile the user application program as well as directly interface to the D4110 for program download and on-line monitoring. The program is developed off-line, compiled, and then downloaded to the module. SYSdev allows the D4110 to be programmed in a combination of languages: Ladder, High-level (subset of C) and Assembly (MCS-51).

1.2 PROGRAM EXECUTION TIMES

Typical program scan times are on the order of 0.6 milliseconds per K of user program with scan times as low as 80 microseconds for short programs. Two additional 10-30VDC interrupt inputs allow throughputs even less than 80 microseconds.

1.3 ALPHANUMERIC DISPLAY

A 2-line by 40 character alphanumeric display is built directly into the D4110. Complete control of the display is provided through commands accessed through the user program. Commands such as: “position cursor”, “advance cursor forward one space”, “backspace cursor one space”, “clear display”, “enter characters into display”, “line feed”, “scroll display” as well as an ASCII string conversion system function allow easy and complete control of the display directly in the D4110 user’s program. Vacuum fluorescent display technology at 0.2" character height provides both high visibility and high character density.

1.4 KEYPAD

The keypad is a 3 row by 8 column sealed keypad with interchangeable legends for easy user customization. Key depressed decode is performed automatically by the D4110 with the key number depressed mapped directly to an input byte (X20).

1.5 DIGITAL INPUTS

The 16 digital inputs are 10-30VDC sourcing (true high) which are used to interface to the application inputs such as proximity sensors, push-buttons, etc. The input is "on" ("1") when the input voltage exceeds 10VDC and is "off" ("0") when the input voltage is below 5VDC. Individual LED status indication is provided for each input. All inputs are optically isolated and provided with an input filter delay (nominally 1.0 milliseconds).

1.6 INTERRUPT INPUTS

The D4110 module contains two interrupt inputs which allow hardware interrupts to be implemented in the user's program. The inputs are 12-30VDC differential inputs which can be enabled as interrupts or disabled and used as standard inputs. When enabled as interrupts, an "off" to "on" transition of the enabled input, activates an interrupt call to a user programmed file (ufunc00 for input0 and ufunc01 for input1). This suspends the main program file until the interrupt file execution is completed, at which time program execution resumes at the place in the main file where the interrupt occurs. This mechanism allows ultra fast throughputs to be implemented if required.

1.7 DIGITAL OUTPUTS

The 16 digital outputs are 10-30VDC sourcing (true high) which are used to interface to the application outputs such as solenoids, lamps, PLC inputs, etc. Each output is rated at 1 amp DC (continuous) with an in-rush (pulsed) current drive capability of 5 amps for 100msec. The sum of the current within an 8 output group must not, however, exceed 6 amps. All outputs are optically isolated and contain a transient suppression circuit to protect the output when driving inductive loads. The outputs do not contain output fusing, therefore, external fusing should be provided.

1.8 INTERFACE PORTS

The D4110 module contains four interface ports: the PROGramming PORT, the USER PORT, and two Serial Network Comm ports.

PROG PORT: The PROG port is an RS-232 port dedicated for on-line monitoring and program download when the D4110 is connected to an IBM PC or compatible running SYSdev.

USER PORT: The USER PORT is available as a general RS-232/RS-422 port for use as defined by the user. Under software control of the user application program, communications to any other RS-232/RS-422 based device can be established. Typical applications include communications to a host computer for data acquisition, etc.

SERIAL NETWORK PORTS: The serial network ports conform to the S3000-N1 network. This network is a high speed (up to 344KBPS), twisted pair, serial network configured in a master/slave topology. Up to 32 D4110/M4000 modules and/or S3000 processors (nodes) can be connected on one network. Communications between the nodes on the network is controlled via commands (sfunc13) in the user application program resident in the node acting as the master.

1.9 REAL TIME CLOCK

The real time clock provides the current time and date. The time is provided in a 24 hour format in the form: hours, minutes, and seconds. The date is provided in the form: month, day of month, and year. The real time clock is accurate to within 1 minute per month even in the absence of power to the D4110.

1.10 DIAGNOSTICS/FAULT DETECTION

The D4110 contains comprehensive fault detection routines which verify the proper operation of the module at all times. Each detected fault has a corresponding fault code which can be viewed using SYSdev, providing a description of the fault and recommended corrective action. The D4110 contains a fault interlock (24VDC, 500mAMP, sinking) output which can be interlocked to the control system for system shut down or annunciation when a fault is detected. In addition to the fault code detection, a hardware confidence test is resident in the module to provide a complete test of the module hardware. This test is initiated through SYSdev and can be used to verify the D4110 for proper operation.

1.11 LED STATUS INDICATIONS

The following four status LEDs are located on the back of the D4110: RUN, COMM1, COMM2, and FAULT. The definitions of these LEDs are as follows:

RUN: "On" steady when the D4110 is running a valid user's application program. "Off" when an internal fault is detected or when a valid user's program has not been loaded. The RUN led is flashed during program download and also when the hardware confidence test is executed.

COMM1: This LED is flashed every time an access to the serial network port #1 is made by any board or module on the network. If the LED is on solid, continuous communications is occurring on the network. If the LED is "off", no communications is occurring. This is not a fault LED, but simply an indication of activity on the serial network attached to port #1.

COMM2: This LED is flashed every time an access to the serial network port #2 is made by any board or module on the network. If the LED is on solid, continuous communications is occurring on the network. If the LED is "off", no communications is occurring. This is not a fault LED, but simply an indication of activity on the serial network attached to port #2.

FAULT: "On" when an internally detected fault has occurred in the D4110. See section 7 for more details on the fault routines and error codes.

SECTION 2
PROGRAM STRUCTURE

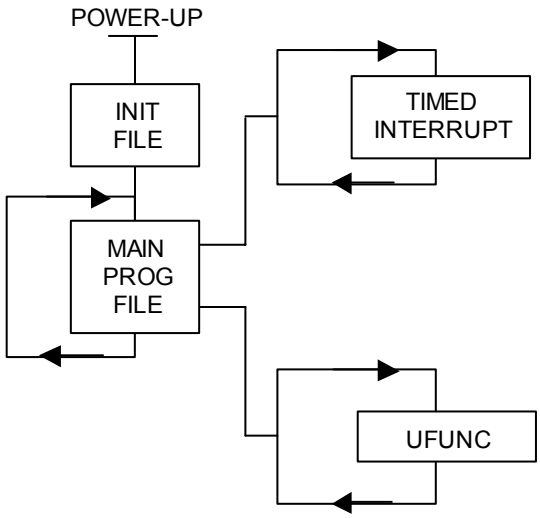
The SYSdev programming language is a combination of Ladder, High-level (subset of C) and Assembly (MCS-51). All the files shown in the following are programmed in the same language format. Each file can be written in any combination of the language types. The typical D4110 user program consists of the following files:

1) Initialization file (optional): executed once at power up.

2) Main Program file (required): scanned continuously.

3) Timed Interrupt file (optional): executed once every 0.5, 1.0, or 10.0 milliseconds as set by the user.

4) User Function files (optional): up to 100 user defined subroutines which can be called from any of the above files.



5) Input Interrupts (optional): the two input interrupts can be enabled or disabled. Input0 interrupt calls ufunc00 when activated ("off" to "on" transition of input0) while input1 interrupt calls ufunc01.

   Note: ufunc00 must be created by the user if the input0 interrupt is enabled and ufunc01 if the input1 interrupt is enabled.

Each file is executed sequentially from beginning to end. The main program file is executed (scanned) continuously unless interrupted by the timed interrupt or an input interrupt is activated. When this occurs, main program execution is suspended while the interrupt file is executed. At the completion of the interrupt, program execution resumes at the point in the main program where the interrupt occurred.

Each file is implemented as a series of consecutive blocks. Each block is defined as one of the three programming languages: Ladder, High-level or Assembly. Blocks of the different languages can be intermixed as necessary within the file.

All D4110 I/O is updated (inputs read, outputs written) at the beginning of each main program scan. These updates are stored in the 'X' and 'Y' I/O image bytes of the module (see section 4.1).

SECTION 2
PROGRAM STRUCTURE

When the timed interrupt is enabled, the 'X' input variables are updated at the beginning of the main program as normal, however, the 'Y' output variables are updated at the beginning of the timed interrupt execution instead of the beginning of the main scan. In addition to these I/O updates, the inputs are read at the beginning of the timed interrupt and stored at special function variables B62-B63 (see Section 4.4.2). This in effect constitutes an immediate I/O for the timed interrupt.

Note: 'Y' output variables cannot be used as coils in the main program if the timed interrupt is enabled. Any outputs that are to be activated by the main program file must be passed to the timed interrupt file as a flag ('F' variable) and then mapped to the 'Y' output in the timed interrupt.

See the SYSdev Programming Manual for more details on the typical program structure.

SECTION 3
SYSTEM CONFIGURATION

The system configuration defines the D4110 module configuration that the program will run in. This includes defining the serial network baud rate, enabling or disabling the input0 and input1 interrupts, and enabling or disabling the fixed scan mode or timed interrupt. These parameters are all set through SYSdev when the program is developed. See the SYSdev Programming Manual for more details.

3.1 TARGET BOARD

This is used to select the module that the program will be loaded into. For the D4110, this parameter is always set to D4110. Selecting a specific module, enables the complier to generate the appropriate I/O reads and writes corresponding to the available I/O of the module.

3.2 NETWORK BAUD RATE

Three serial network baud rates are available: 344KBPS (bits per second), 229KBPS, or 106KBPS.

Note: All the modules connected on the network must be set to the same baud rate, otherwise a communications error will occur.

For the most part, the baud rate is set as a function of the total network distance. The longer the total network , the slower the baud rate. As a general rule the baud rate can be set as follows: 344KBPS for network distance of 1000 feet or less; 229KBPS for 2000 feet or less; and 106KBPS for 4000 feet or less. The two serial ports on the D4110 cannot be set to different baud rates. Both ports will run at the rate selected in this parameter.

3.3 INPUT0 INTERRUPT ENABLE

If the Input0 interrupt is to be used, it must be enabled in the system configuration. The input0 interrupt calls ufunc00 when activated, thus the user must create ufunc00. The ufunc00 file is created and executed just like any other user function file with the exception that it is called when the input0 interrupt input makes an “off” to “on” transition, instead of being called from the main user program. If the input0 interrupt is disabled, interrupt input0 can be used as a standard input by reference P32 (see section 4.1.4).

3.4 INPUT1 INTERRUPT ENABLE

If the Input1 interrupt is to be used, it must be enabled in the system configuration. The input1 interrupt calls ufunc01 when activated, thus the user must create ufunc01. The ufunc01 file is created and executed just like any other user function file with the exception that it is called when the input1 interrupt input makes an "off" to "on" transition, instead of being called from the main user program. If the input1 interrupt is disabled, interrupt input1 can be used as a standard input by reference P33 (see section 4.1.4).

3.5 FIXED SCAN TIME MODE

When enabled, the fixed scan time mode allows the user to set the main program scan to a fixed time, either 0.5 milliseconds, 1.0 milliseconds, or 10.0 milliseconds. This allows the main program scan to be used as a high speed time base for either fixed rate sampling or high speed timer time bases (when "scan" time base timers are used).

Note: The actual main program execution time must be less than the selected fixed time, otherwise, the scan time will equal the actual scan time rather than the fixed scan time. If the fixed scan time mode is disabled, the scan time will be a function of the length of the user program and vary as a function of the true/false state of the logic.

The fixed scan mode is enabled by selecting 'Y' from the "Enable Fixed Scan or Timed Interrupt?" prompt, then selecting "0 = Fixed Main Scan" from the next prompt.

Note: Both the fixed scan mode and timed interrupt cannot be enabled at the same time.

3.6 TIMED INTERRUPT

If the timed interrupt file is to be used, it must be enabled in system configuration. The timed interrupt interval must also be selected as 0.5, 1.0, or 10.0 milliseconds. The timed interrupt file will be called at these intervals, thus the user must create the timed interrupt file. The timed interrupt file is created and executed just as any other file with the exception that it is executed at the specified interval (by interrupting the main program). In addition all 'Y' outputs are updated at the beginning of the timed interrupt as well as the inputs being read and stored at special function variables B62 - B63 (these are used as the immediate inputs for the timed interrupt).

Note: The actual timed interrupt execution time must be less than the selected timed interrupt time, otherwise, a main program scan watchdog time out will occur.

The timed interrupt is enabled by selecting 'Y' from the "Enable Fixed Scan or Timed Interrupt?" prompt then selecting "1 = TIMED INTRPT" from the following prompt.

Note: Both the fixed scan mode and timed interrupt cannot be enabled at the same time.

SECTION 3
SYSTEM CONFIGURATION

(This Page Intentionally Left Blank)

4.1 VARIABLES

Three classes of variables are used in the D4110. They are: bits, bytes, and words. Bits are a single bit in width and can have a value of 0 or 1. Bytes are 8 bits in width and can have a value between 0 and 255 decimal or 0 and ffH hex. Words are 16 bits in width and can have a value of 0 to 65535 decimal or 0 to ffffH hex. All numbers (values in variables and constants) are unsigned integer values. No signed or floating point numbers are supported. Numbers can be represented as decimal or hex (suffix 'H' following number).

Six different variable types are available in the D4110: flags (F), bytes (B), words (W), port-pins (P), inputs (X), and outputs (Y).

4.1.1 Flags (F):

Flags are single bit variables which are generally used as internal coils or flags in the user program. Flags can have a value of "0" or "1". The D4110 module contains 104 flags.

The format of the flag variable is:

Fzzz where: zzz is a three digit flag address (000 to 111).

Note: The leading 'F' must be a capital letter and that the flag address must be three digits (include leading zeros as necessary).

Examples: F000, F012, F103, etc.

4.1.2 Bytes (B):

Byte variables are 8 bit variables used as general purpose variables in the user program. Byte variables can have a value between 0 and 255 decimal or 0 and ffH hex. Byte variables are used as arithmetic variables in the High-level language, timer/counter presets and accumulators as well as shift register bytes in the ladder language. The D4110 module contains 200 'B' variables.

The format of the byte variable is:

Bzzz where: zzz is the three digit byte address (032 thru 231).

Note: The leading 'B' must be a capital letter.

Examples: B032, B150, B201, etc.

Individual bits within the byte can also be referenced by simply appending a '.' followed by the bit number (0-7) to the byte address. The form of this is:

Bzzz.y where: zzz is the byte address and y is the bit (0-7).

This allows any bit in the entire data memory to be referenced just as a flag is referenced. These "byte.bit" variables can be used in ladder blocks as contact and coil variables as well as in the High-level blocks. Execution times for instructions that use bits within a byte are longer than execution times for instructions using flags. Keep this in mind when using "byte.bit" references.

Examples: B080.0, B100.7, B072.4, etc.

4.1.3 Words (W):

Word variables are 16 bit variables used as general purpose variables in the user program. Words can have a value between 0 and 65535 decimal or 0 and ffffH hex. Word variables are used as arithmetic variables in the High-level language. The D4110 module contains 100 'W' variables.

The format of the word variable is:

Wzzz where: zzz is the three digit word address (032 thru 230).

Note: The leading 'W' must be a capital letter. Also, word addresses are always an even number (divisible by 2).

Examples: W034, W100, W076, etc.

4.1.4 Port-Pins (P):

Port-pins are single bit variables that map directly to specific hardware functions on the M4000 modules. These can be input or output hardware functions as defined by the specific port pin (see the following).

The format for port pins is:

Paa where: aa is the two digit port pin (10-17 or 30-37).

Note: The 'P' must be a capital letter and that the port pin address must be two digits.

The following port pins on the D4110 modules are mapped to the respective hardware functions:

P32: interrupt input0

The state of interrupt input0 is mapped to this port pin. If interrupt input0 is not enabled as an interrupt, it can be used as a standard (non-interrupt) input.

Note: The state of interrupt input0 is true low logic, thus when the input is "on", P32 will be a "0". When input0 is "off", P32 will be a "1".

P33: interrupt input1

Just as with interrupt input0, interrupt input1 is mapped to port pin P33. Input1 functions identically to input0.

4.1.5 Inputs (X):

Input variables are bytes that contain the data read from the D4110 inputs during the main program I/O update. One 'X' byte is allocated for each input byte, thus the D4110 has two 'X' bytes allocated for it, one byte for inputs 00 thru 07, and one for inputs 10 thru 07. The input bytes reside in the I/O image table of data memory and can only be accessed using the 'X' variable designation.

The format for the input byte is:

Xaab where: aa is the two digit I/O address (00-02) and b is the byte at the slot (0 or 1).

Note: The 'X' must be a capital letter. Also, 'X' variables can only be referenced for inputs that are actually available in the module. Any reference to input variables that do not correspond to existing inputs will result in a compiler error.

As with byte variables, individual bits within the 'X' variable can be referenced. These bits correspond to the respective I/O point of the input byte. The form of this is:

Xaab.c where: aa is the I/O address, b is the byte at the slot and c is the bit or input point.

Examples: X010, X000, X020.5, X000.7, etc.

4.1.6 Outputs (Y):

Output variables are bytes which contain the data that is written to D4110 outputs at the beginning of the main program I/O update. One 'Y' variable is allocated for each output byte, thus the D4110 module has two 'Y' variables allocated for it, one byte for for outputs 00 thru 07, and one byte for outputs 10 thru 17.

The format for the 'Y' variable is:

Yaab where: aa is the two digit I/O address (00-02) and b is the byte at the slot (0 or 1).

Note: The 'Y' must be a capital letter. Also, 'Y' variables can only be referenced for outputs that are actually available in the module. Any reference to output variables that do not correspond to existing outputs will result in a compiler error.

As with byte variables, individual bits within the 'Y' variable can be referenced. These bits correspond to the respective I/O point on the output board. The form of this is:

Yaab.c where: aa is the I/O address, b is the byte at the slot and c is the bit or output point.

Examples: Y021, Y000, Y001.5, Y021.7, etc.

4.1.7 Constants:

Constants are used as fixed numbers in High-level arithmetic and conditional statements as well as for presets in timer/counters in ladder blocks.

In High-level blocks, constants can be represented in decimal or hex. If the number is decimal, the constant is simply entered as the number to be referenced. No prefix or suffix is specified. If the number is hex, the suffix 'H' is added immediately following the hex number. Examples of both are:

25 (decimal)
25657 (decimal)
aeH (hex)
f000H (hex)

The hex letters (a,b,c,d,e,f) are case sensitive and must be typed as lower case letters. The hex suffix is also case sensitive and must be typed as a capital letter (H).

All constants are unsigned integers. When the variable class is byte, the range of values is 0 to 255 decimal or 0 to ffH hex. If the variable class is word, the range of values is 0 to 65535 decimal or 0 to ffffH hex.

In ladder blocks, the only constants allowed are in timer/counter presets. In this case, they are specified in decimal and preceded with the prefix '#'.

4.2 DATA MEMORY MAP

The D4110 module contains two distinct data memory spaces: 200 bytes of volatile (non-battery backed) data memory and 2K bytes of non-volatile (battery backed) data memory. The flag (F), byte (B) and word (W) variables, as described previously, are located in the 200 bytes of volatile data memory. The 2K bytes of non-volatile data memory can only be accessed using sfunc07 and sfunc08 (see Sections 5.2).

SECTION 4
VARIABLE TYPES/MEMORY MAP

4.2.1 VOLATILE DATA MEMORY

The memory map for the D4110 volatile data memory is shown below:

Address	Valid Variable References		
0032	F000-F007	B032	W032
0033	F008-F015	B033	——
0034	F016-F023	B034	W034
0035	F024-F031	B035	——
thru	thru	thru	thru
0043	F088-F095	B043	——
0044	F096-F103	B044	RESERVED
0045	RESERVED	RESERVED	RESERVED
0046	RESERVED	RESERVED	RESERVED
thru	thru	thru	thru
0062	RESERVED	RESERVED	RESERVED
0063	RESERVED	RESERVED	RESERVED
0064	————	B064	W064
0065	————	B065	——
0066	————	B066	W066
thru	thru	thru	thru
0230	————	B230	W230
0231	————	B231	——

These memory locations (B032 thru B231) are not battery backed and will not retain data at power down. At power-up or reset, these addresses are cleared.

Note: Flags F000 thru F103 are mapped into bytes B032 thru B044. Bytes B032 thru B230 are also mapped into W032 thru W230. These addresses can be referenced as any or all three of these variable types.

The flags are mapped into the bytes as shown as follows:

F000 = B032.0
F001 = B032.1
F002 = B032.2
F003 = B032.3
F004 = B032.4
F005 = B032.5
F006 = B032.6
F007 = B032.7
F008 = B033.0
F009 = B033.1
etc.

The bytes are mapped into the words with the even byte address as the low byte (lower 256 significance) of the respective word and the odd byte address as the upper byte (upper 256 significance) of the word as shown:

B032 = W032 (low byte)
B033 = W032 (high byte)

4.2.2 NON-VOLATILE (BATTERY-BACKED) DATA MEMORY

The memory map for the non-volatile (battery-backed) data memory is shown below.

Note: These memory locations are not referenced as user variables (F,B, and W), but instead are accessed using sfunc07 and sfunc08.

Address	Valid Variable References		
1900H	———	———	———
1901H	———	———	———
thru	thru	thru	thru
1feeH	———	———	———
1fefH	———	———	———

These variables are battery-backed and will retain data when powered down. This memory space provides a non-volatile data space for user variables such as timer/counter presets, etc. This memory space is not cleared at power-up.

4.3 I/O IMAGE ADDRESSING

The I/O of the D4110 module is mapped to the following I/O image bytes:

I/O Image	I/O Function	
Y000	I/O-0 outputs	0.0 – 0.7
Y001	I/O-0 outputs	1.0 – 1.7
X010	I/O-1 inputs	0.0 – 0.7
X011	I/O-1 inputs	1.0 – 1.7
X020	key number depressed on keypad	

SECTION 4
VARIABLE TYPES/MEMORY MAP

4.4 SPECIAL FUNCTION VARIABLES

The following variables are used as special function variables. These variables should not be used as general purpose variables within the user program, but only for the purposes described below:

4.4.1 F104: USER PORT RS-422 MODE SELECT

F104 is used to set the USER PORT in either RS-232 mode or RS-422 mode. When F104 is set to a "0", the USER PORT is in RS-232 mode and the RS-232 pins of the USER PORT connector are active. When F104 is set to a "1", the USER PORT is in RS-422 mode and the RS-422 pins of the USER PORT connector are then active. This flag is set or cleared based on the type of device the D4110 USER PORT will be connected to. See section 6.4 for more details.

4.4.2 F105: SERIAL NETWORK PORT SELECT

F105 is used to select which serial port (either #1 or #2) will be used when an sfunc13 network communications system function is called. When F105 is set to a "0", port #1 is selected. When F105 is set to a "1", port #2 is selected. F105 is set or cleared just prior to actually calling the sfunc13 for that particular port. See section 6.3 for more details.

4.4.3 B62 - B63: TIMED INTERRUPT IMMEDIATE INPUT VARIABLES

When the timed interrupt is enabled, B62 and B63 are used as the input image bytes of the I/O inputs. At the beginning of the timed interrupt, the corresponding inputs are read and the data from these inputs is stored at these variables in the same fashion that the ‘X’ variables are updated at the beginning of the main scan. Thus, bytes B62 and B63 should be used as the input image bytes inside of the timed interrupt file instead of the ‘X’ variables.

Note: The ‘X’ variables are still updated at the beginning of the main scan even when the timed interrupt is enabled.

The I/O of the D4110 module is mapped to the B62 - B63 variables when the timed interrupt is enabled as follows:

Input Image	Input Function	
B62	I/O-1 inputs	0.0 – 0.7
B63	I/O-1 inputs	1.0 – 1.7

4.5 SYSTEM FUNCTION BUFFERS

The following locations are the addresses of the corresponding system function buffers. These are external addresses that can be loaded or read directly using sfunc07 and sfunc08. See the corresponding description of each system function in section 6 for details on using the sfunc buffers directly:

System Function	Buffer Address (external)
sfunc 11	b202H to b2fcH
sfunc 13 (port #1)	b30aH to b3faH
sfunc 13 (port #2)	7e0aH to 7efaH
sfunc 18	b402H to b4fcH

SECTION 4
VARIABLE TYPES/MEMORY MAP

(This Page Intentionally Left Blank)

SECTION 5
PROGRAMMING REFERENCE

The following sections provide an overview of the SYSdev instruction set and the system functions available in the D4110 module. See the SYSdev Programming Manual for more details on the SYSdev programming language and the operation of the SYSdev software package.

5.1 INSTRUCTION SET

5.1.1 LADDER

The ladder language is generally used to implement the boolean logic of the user program. Networks of virtually any form (including nested branches) can be implemented. Ladder blocks are implemented as a 7 row X 9 column matrix. The following ladder instructions are available:

1) Contacts
   - Normally open
   - Normally closed
2) Coils
   - Standard
   - Latch
   - Unlatch
   - Inverted
3) Timers
   - 0.01 second time base
   - 0.10 second time base
   - 1.00 second time base
4) Counters
5) Shift Registers

Valid variables for contacts and coils are flags (F) or bits out of bytes (B).

Valid variables for timer/counter presets and accumulators are bytes (B). The maximum preset is 255.

Valid variables for shift registers are also bytes (B). The number of shifts per variable is 7.

5.1.2 HIGH-LEVEL ('C')

The High-level language is a subset of the 'C' programming language. High-level is used for all arithmetic, comparisons, conditional program execution, program looping, calling user functions (subroutines) and calling system functions. High-level blocks are implemented as a 57 row X 80 column text array.

The High-level language incorporates the following:

1) Operators:

+:	add	++:	increment
-:	subtract	—:	decrement
*:	multiply	==:	equate
/:	divide	>:	greater than
%:	remainder	>=:	greater than or equal
<<:	left shift	<:	less than
>>:	right shift	<=:	less than or equal
&:	bitwise AND	!=:	not equal
|:	bitwise OR	~:	complement
^:	bitwise EX-OR	*:	indirection (unary)
&&:	logical AND	&:	address operator
||:	logical OR	=:	equal (assignment)

2) Statements:

- program statements (equations)
- conditional program execution ("if else-if else")
- program looping ("for", "while", and "do while" loops)
- unconditional program jumping ("goto")
- user function calls ("ufuncXX" subroutines)
- system function calls ("sfuncXX" I/O operations)

5.1.3 ASSEMBLY

The Assembly language conforms to the Intel MCS-51 instruction set. The assembler syntax conforms to the UNIX system V assembler syntax.

5.2 SYSTEM FUNCTIONS

System functions provide the user with a means to perform extended functions such as communication on the serial network, etc. A summary of the system functions available in the D4110 module is as follows:

sfunc02: Current Time/Date Read
sfunc03: Watchdog Timer Reset
sfunc04: ASCII String Load
sfunc07: General External Address Read
sfunc08: General External Address Write
sfunc09: System Fault Routine
sfunc10: User Port Receive
sfunc11: User Port Transmit
sfunc13: Serial Network Communications
sfunc18: Display Write (Update)

System functions are entered in high-level blocks as text. Each system function has a parameter list associated with the system function call which defines such things as the address to read/write to, the number of bytes to send/receive, etc. In addition, some system functions return with an error code or function status which can be used to determine if the system function was successful, busy, etc.

5.2.1 SYSTEM FUNCTION TYPES

Two types of system functions exist: **suspended** and **simultaneous**.

Suspended system functions actually suspend program execution while they are executed. Thus, they are performed just as any other type of instruction, in order of sequence in which they occur.

Simultaneous system functions are executed simultaneously to program execution. By their nature, simultaneous system functions may take multiple main program scans to execute. These are basically “background” tasks which are executed while the user application program is executing, with insignificant impact on the user program scan time.

The simultaneous system function returns with one of four types of return values when called: Not Busy, Busy, Done or an error code representing a fault in the execution of the function. When the function is first executed, a return value of “Busy” is returned. This indicates the function is executing and is no longer available for use until it has been completed. Subsequent calls to the same system function will result in a “Busy” return value until the function has completed. At that time, a call to the system function will result in either a “Done” return value or an error code value representing a failure of the function to execute. The system function is now available to execute again. See the individual system function formats following for more details on the return values and error codes pertinent to each system function.

5.2.2 sfunc02: current time/date read

System function 02 is used to read the current time and date from the real time clock embedded in the D4110. When executed, sfunc02 reads the real time clock and stores the time and date in six consecutive bytes in variable memory as follows: hours (1-24), minutes, seconds, month, day, year (0-99).

Note: A 24-hour time format is used by the real time clock, thus for 9:00 am, hours = 9, for 5:00 pm, hours = 17. Also, the "year" byte will contain only the last two digits of the current year.

General form: sfunc02(dest);

Parameters: dest = First address of the six consecutive bytes where the current time/date will be stored.

The time and date is stored in the six consecutive bytes as follows:

byte #1: hours (1-24)
byte #2: minutes (0-59)
byte #3: seconds (0-59)
byte #4: month (1-12)
byte #5: day of month (1-31)
byte #6: year (0-99)

Variables types: 'B' or indirect 'B'

Return Value: none

Type: suspended

Valid File: Initialization, Main Program, Timed Interrupt, and user functions

Example: sfunc02(B100);

If the above example was called at 10:23:17 pm on November 12, 1992, the following bytes will be loaded with the corresponding values:

B100= 22 (hours = 10 pm)
B101= 23 (minutes)
B102= 17 (seconds)
B103= 11 (month of year)
B104= 12 (day of month)
B105= 92 (year)

5.2.3 sfunc03: watchdog timer reset

System function 03 resets the main program watchdog timer when called. The watchdog timer normally times out if the main program san time is longer than 100msec. This function can be used to extend this time by 100msec every time sfunc03 is called. This is desirable, for instance, if a long, intentional program loop ("for" loop, "while" loop, etc.) is executed which would exceed the normal 100msec scan time.

General form: sfunc03();

Parameters: none

Return Value: none

Type: suspended

Valid Files: Initialization, Main Program, Timed Interrupt, and user functions

5.2.4 sfunc04:ASCII string load command

System function 04 is used to convert the characters in an ASCII string to their equivalent ASCII codes and store these codes in consecutive byte addresses in variable memory (Bxxx variables) or external non-volatile memory (addresses 1900H-1fefH). System function 04 is typically used in conjunction with the display sfunc18 write (update) system function to write ASCII strings to the display.

General form: sfunc04(dest,"string");

Parameters: dest = The address where the first ASCII character of the string will be stored. The remaining ASCII characters will be stored in consecutive byte addresses following the first byte address. Variable types: 'B' or constant 1900-ffffH.

string = The string is from one to 60 printable characters. These characters will be converted to their equivalent ASCII codes and stored in consecutive byte addresses starting at the dest byte address.

Note: The string must be enclosed with double quotes as shown (these double quotes are not stored as part of the string, but are simply used as delimiters for the string). Any printable character can be incorporated in the string with the exception of the double quote " or back slash \. If these two characters are to be incorporated in the string, they must be preceded with the back slash (i.e. \" will incorporate the " only and \\ will incorporate just one \).

Return Value: none

Type: suspended

Valid Files: Initialization, Main Program, Timed Interrupt and user functions

Examples 1) sfunc04 (B100, "example #1");

The above example will load the following byte addresses with the corresponding ASCII codes (numbers):

B100 = 101	(101 = ASCII code for 'e')
B101 = 120	(120 = ASCII code for 'x')
B102 = 97	(97 = ASCII code for 'a')
B103 = 109	(109 = ASCII code for 'm')
B104 = 112	(112 = ASCII code for 'p')
B105 = 108	(108 = ASCII code for 'l')
B106 = 101	(101 = ASCII code for 'e')
B107 = 32	(32 = ASCII code for space)
B108 = 35	(35 = ASCII code for '#')
B109 = 49	(49 = ASCII code for '1')

2) sfunc04(B150,":");

   The above example will load B150 with 58 which is the ASCII code for ':'.

3) sfunc04(1a00H,"MOTOR\"on\"");

   The above example incorporates double quotes in the string and uses the back slash to designate that these double quotes are part of the string and not the string delimiters. The characters are stored in non-volatile memory as follows:

```
1a00H = 77      (77 = ASCII code for 'M')
1a01H = 79      (79 = ASCII code for 'O')
1a02H = 84      (84 = ASCII code for 'T')
1a03H = 79      (79 = ASCII code for 'O')
1a04H = 82      (82 = ASCII code for 'R')
1a05H = 32      (32 = ASCII code for space)
1a06H = 34      (34 = ASCII code for ")
1a07H = 111     (111 = ASCII code for 'o')
1a08H = 110     (110 = ASCII code for 'n')
1a09H = 34      (34 = ASCII code for ")
```

5.2.5 sfunc07: general external address read

System function 07 is used to read the battery-backed data memory which is not referenced as ‘B’ or ‘W’ variables. These are memory locations 1900H thru 1fefH. This system function reads one byte from the address specified.

General form: sfunc07(ext address,dest);

Parameters: ext address = The 16 bit external RAM address (1900H thru 1fefH) to be read. Variable types: ‘W’ or constant (1900H thru ffefH).

dest = The variable where the value read will be stored. Variable types: ‘B’ or indirect ‘B’.

Return value: sfunc07 returns with the value read from the external address.

Type: suspended

Valid files: Initialization, Main Program, Timed Interrupt, and User functions

Example: sfunc07(1900H,B100);

The above reads the non-volatile data byte address 1900H and stores the value read in B100.

5.2.6 sfunc08: general external address write

System function 08 is used to write data to the battery-backed data memory which is not referenced as 'B' or 'W' variables. These are memory locations 1900H thru 1fefH. This system function writes one byte to the address specified.

General form: sfunc08(ext address,srce);

Parameters: ext address = The 16 bit external RAM address (1900H thru 1fefH) to be written to. Valid variables: 'W' or constant (1900H thru ffefH).

srce = The variable where the value that will be written is stored. Variable types: 'B'.

Return value: sfunc08 returns with the value written to the external address.

Type: suspended

Valid files: Initialization, Main Program, Timed Interrupt, and User functions

Example: sfunc08(W100,B105);

With W100 = 1905H, the above writes the data in B105 to non-volatile data byte address 1905H.

5.2.7 sfunc09: system fault routine

System function 09 provides a means for the fault routine to be called in response to a software detected fault from the user application program. The fault routine is executed as described in section 7.1. The fault code will be set to 45H: sfunc09 generated fault.

Note: This function should only be called when a complete system shutdown is desired due to the fact that program execution will cease.

General form: sfunc09();

Parameters: none

Return value: none

Type: non-returning

Valid files: Initialization, Main Program, and User functions

5.2.8 sfunc10: USER PORT receive

System function 10 receives a consecutive number of bytes from the USER PORT. See Section 6.4.1 for a detailed description of the use of sfunc10.

General form: sfunc10(#max,dest);

Parameters:
- #max = This defines the size of the "dest" receive buffer. In essence, this is the maximum number of bytes which can be received between sfunc10 calls. Variable types: constant (1-250), 'B' or indirect 'B'.
- dest = The address of the first byte of the sfunc10 receive buffer. The receive buffer is where the bytes received from the USER PORT will be stored. Variable types: 'B' or indirect 'B'.

Return Values: The return value of sfunc10 is the number of bytes which have been received from the USER PORT and stored in the "dest" receive buffer. Unlike sfunc10 in other S300 boards and M4000 modules, the return value is not BUSY, DONE, or an error code. Once sfunc10 is called, the USER port indefinitely waits for data to be sent to it.

Type: simultaneous

Valid Files: Initialization, Main Program, and User functions

5.2.9sfunc11: USER PORT transmit

System function 11 transmits a consecutive number of bytes out the USER PORT. See Section 6.4.2 for a detailed description of the use of sfunc11.

General form: sfunc11(#sent,srce);

Parameters:
- #sent = The number of bytes to transmit out the USER PORT. Variable types: constant (1-250), 'B' or indirect 'B'.
- srce = The address where the first byte transmitted is stored. A consecutive number of bytes (= #sent) is transmitted out the USER PORT starting with this address. Variable types: 'B' or 'indirect 'B'. In addition, data may be transmitted directly from the internal sfunc11 buffer (external addresses b202H-b2fcH) by indirectly setting "srce" equal to 0 (see section 6.4.2).

Return Values:
- 0 = NOT BUSY/READY
- 1 = BUSY
- 2 = DONE (transmit successful)

Type: simultaneous

Valid Files: Initialization, Main Program, and User functions

5.2.10 sfunc13: serial network communications

System function 13 is used to communicate to other S3012s, S3014s, M4000 modules, or other D4110 nodes on the serial communication network. See section 6.3 for details on the use of sfunc13 and a description of the serial network.

General form: sfunc13(slave,#sent,s_srce,s_dest,#rcve,r_srce,r_dest);

Parameters:
- slave = Address of node to communicate with. This is the network address of the slave, each slave has a unique address. Variable type: constant (1-32), 'B' or indirect 'B'.
- #sent = Number of words to send to slave. Variable types: constant (0-120), 'B' or indirect 'B'.
- s_srce = Address of send stack in master which will be sent to slave. A consecutive number of words (= #sent) will be sent to the slave starting at this address. Variable type: 'W' or indirect 'W'. In addition, data may be transmitted directly from the internal sfunc13 buffer (external addresses b30aH-b3faH for port #1 and 7e0aH-7efaH for port #2) by indirectly setting "s_srce" equal to 0 (see section 6.3.1).
- s_dest = Starting address of stack in slave where words sent from master will be stored. Variable type: 'W' or indirect 'W'.
- #rcve = Number of words received from slave. Variable type: constant (0-120), 'B' or indirect 'B'.
- r_srce = Starting address of stack in slave where words will be sent from slave to master. Variable type: 'W' or indirect 'W'.
- r_dest = Starting address in master where words sent from slave will be stored. Variable type: 'W' or indirect 'W'. In addition, data may be read directly from the internal sfunc13 buffer (external addresses b30aH-b3faH for port #1 and 7e0aH-7efaH for port #2) by indirectly setting "r_dest" equal to 0 (see section 6.3.1).

Return values:
- 0 = NOT BUSY/READY
- 1 = BUSY
- 2 = DONE (comm with slave successful)
- 3-10H = ERROR CODE (see section 7.4.1 for serial network communication error code descriptions).

Type: simultaneous

Valid files: Initialization, Main Program, and User functions

5.2.11 sfunc18: display write (update)

System function 18 writes a consecutive number of bytes to the display. See section 6.1.1 for a detailed description of the use of sfunc18.

General form: sfunc18(#sent, srce);

Parameters: #sent = The number of bytes to write to the display. Variable types: constant (1-250), 'B' or indirect 'B'.

srce = The address where the first byte written is stored. A consecutive number of bytes (= #sent) is written to the display starting with this address. Variable types: 'B' or indirect'B'. In addition, data may be written directly from the internal sfunc18 buffer (external addresses b402H-b4fcH) by indirectly setting "srce" equal to 0 (see section 6.1.1).

Return values: 0 = NOT BUSY/READY
1 = BUSY
2 = DONE (display updated)

Type: simultaneous

Valid Files: Initialization, Main Program, and User functions

SECTION 5
PROGRAMMING REFERENCE

(This Page Intentionally Left Blank)

6.1 WRITING (UPDATING) THE DISPLAY

The display of the D4110 contains a separate slave processor which actually controls and updates the display. The processor of the D4110 transmits data and control codes to the display processor via sfunc18. This system function is similar to sfunc11 in that the number of bytes to be transmitted to the display and the starting address of these bytes are specified in the system function. Also like sfunc11, sfunc18 is a simultaneous system function such that once initiated, a return value of BUSY (1) or DONE (2) is returned based on whether the display update is complete or still in progress. New data cannot be sent to the display until the last update is complete (return = DONE).

The data sent to the display is a combination of ASCII characters which are to be displayed on the display and control codes which actually control the display (such as “clear” display, position cursor, etc.). Section 6.1.2 contains a list of the display control codes and sections 6.1.3 contains a table of the valid ASCII characters which can be displayed on the display. Both the ASCII data and the control codes are sent to the display via sfunc18. The data and control codes can be intermixed in one sfunc18 call as necessary.

6.1.1 WRITING DATA TO THE DISPLAY (sfunc18)

Using sfunc18, from 1 to 250 consecutive bytes can be written to the display in one command. System function 18 is a simultaneous function such that once it is initiated, program execution continues without waiting for the sfunc to complete. Subsequent calls of sfunc18 result in a return value of “BUSY” until the sfunc completes (return = “DONE”). Since sfunc18 is a simultaneous function, the impact on the user application program scan time is negligible when an sfunc18 is executed.

The general form of sfunc18 is: sfunc18(#send,srce); where “#send” is the number of bytes to write and “srce” is the starting address of the stack of bytes that will be written to the display. When sfunc18 is initiated, the data in “srce” is loaded into the sfunc18 buffer (addresses b402H thru b4fcH) are then written to the display from the sfunc18 buffer.

In cases where a small number of bytes are to be written, “srce” is generally a stack of “Bxxx” variables in data memory. However, when a large number of bytes are to be transmitted, the user can load the sfunc18 buffer directly and write from this buffer without using up “Bxxx” variables in data memory. This is done by loading the sfunc18 buffer (b402H - b4fcH) using the sfunc08 external write system function and then calling sfunc18 with “srce” equal to 0 (this is done by using an indirect ‘B’ variable loaded with an address of “0”). See example #2 below.

Prior to executing the sfunc18 call, the "srce" buffer is formatted, by the user program, with the appropriate control codes and ASCII character codes. The ASCII character codes can be generated using the sfunc04 ASCII string load command. Both examples #1 and #2 below give examples of formatting the "srce" buffer.

Examples:

1)
```
if (F2 == 1)                    /* update display? */
   {
   B100 = 1bH;                  /* position cursor control code */
   B101 = 5bH;                  /* line 2 - location 27 */
   B102 = 12H;                  /* data entry mode */
   B103 = 30H;                  /* write character "0" */
   B104 = 32H:                  /* write character "2" */
   if (sfunc18(5,B100) == 2)    /* update display */
      F2 = 0;
   }
```

execution:

The above code writes the characters "02" starting at location 27 of line 2 in the display.

Note: F2 would be set to "1" somewhere else in the user program to perform the display update. F2 stays at "1" until the display update is complete. The user program could then monitor F2 to determine when the display is available (F2 = 0) or BUSY (F2 = 1).

2)
```
sfunc08(b402H,15H);         /* clear display/cursor home */
sfunc08(b403H,0eH);         /* cursor invisible */
sfunc08(b404H,12H);         /* data entry mode */
sfunc04(b405H,"message to be displayed in line 1");
sfunc04(b42dH,"message to be displayed in line 2");
sfunc08(b455H,0dH);
B88 = 0;
sfunc18(84,*B88);
```

execution:

The above writes the entire display (both lines) in one sfunc18 command. The display is cleared and the cursor positioned at home (upper left-most position) with control code 15H, then the cursor is made invisible, and then the data to be displayed is sent.

Note: The sfunc18 buffer was loaded directly with the control codes and data with the sfunc18 call specified as buffer direct by using "*B88" with B88 = 0. Also, the sfunc04's were used to convert the characters "message to be displayed"..etc. to the equivalent ACSII codes directly into the sfunc18 buffer. These sfunc04s should contain 40 characters each (using spaces to fill in unused display positions) if the entire display is to be updated in one sfunc18 call.

See the example program in appendix A for more examples of using sfunc18 to update the display.

6.1.2 DISPLAY CONTROL CODES

The following is a list of valid display control codes. Care should be taken not to send undefined control codes to the display as this may cause unpredictable display operation.

CODE	DESCRIPTION
08H	Back space cursor location one position
09H	Advance cursor location one position
0aH	Line Feed (vertical scroll from bottom line; cursor positions to the left-most grid)
0dH	Carriage Return (returns cursor to left-most character position of the same line; does not clear display)
0eH	Make cursor indicator invisible (the cursor location counter continues to function but there is no visible indicator of next location)
0fH	Make cursor indicator visible
11H	DATA ENTRY MODE - Normal data entry with automatic carriage return and line feed (data enters beginning at the home position)
12H	DATA ENTRY MODE - Overwrite of right-most character / automatic carriage return off
13H	DATA ENTRY MODE - Horizontal scroll mode (from right to left on bottom line only, after line has been filled) **Note:** For control codes 11H through 13H, ASCII characters sent following these control codes are displayed starting from the original cursor location with the cursor automatically advancing one position after each character.
14H	Display reset
15H	Display clear (returns cursor to upper left-most position of display)
16H	Cursor home (returns cursor to upper left-most position of display)
1bH	Move cursor to following position (two byte instruction to locate cursor. Second byte is the cursor position where the location in binary is: 0LXX XXXX - upper left most location is zero, L=0 for upper line, L=1 for lower line and the six least significant bits specify the cursor position on the specified line in binary)
1dH	Dim display (20%)
1eH	Bright display (50%)
1fH	Brightest display (100%)

See examples #1 and #2 in section 6.1.1 for examples of using the above control codes.

SECTION 6
USING SYSTEM FUNCTIONS

6.1.3 VALID DISPLAYED CHARACTERS

The following is a chart showing the valid character codes which can be displayed on the display. The ASCII character codes 32H thru 7fH can be generated using the sfunc04 ASCII string load command. Examples of this are in examples #1 and #2 of section 6.1.1.

DATA BITS				b7 b6 b5 b4	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 0 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 1 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
b3	b2	b1	b0	HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0			SP	0	@	P	`	p			Ä	Ó	ä	í	▯	
0	0	0	1	1			!	1	A	Q	a	q			À	Ò	à	ì	▯	
0	0	1	0	2			"	2	B	R	b	r			Á	Ä	á	à	▯	
0	0	1	1	3			#	3	C	S	c	s			Ç	Ñ	ç	¢	▯	
0	1	0	0	4			$	4	D	T	d	t			Å	Ë	ä	ë	▯	
0	1	0	1	5			%	5	E	U	e	u			£	Ü	æ	ü	▯	
0	1	1	0	6			&	6	F	V	f	v			É	Ú	é	í	▯	
0	1	1	1	7			'	7	G	W	g	w			È	Ù	è	ì	▯	
1	0	0	0	8			(	8	H	X	h	x			ê	ô	ê	î	▯	
1	0	0	1	9			)	9	I	Y	i	y			ï	°	ï	ø	▯	
1	0	1	0	A			*	:	J	Z	j	z			ƒ	±	í	ø	SP	
1	0	1	1	B			+	;	K	[	k	{			¡	÷	¡	■	SP	
1	1	0	0	C			,	<	L	¥	l	£			¡	■	î	□	SP	
1	1	0	1	D			-	=	M	]	m	}			ö	⇨	ü	▮	SP	CDF
1	1	1	0	E			.	>	N	^	n	▫			Ñ	■	ñ	◇	SP	CDF
1	1	1	1	F			/	?	O	_	o	▒			Ø	ô	ø	▯	▮	CDF

6.2 KEYPAD INTERFACE

The keypad is a 3-row by 8-column sealed keypad. Key depressed decode is performed automatically during the I/O update at the beginning of the main program scan. The key number depressed is automatically updated in input byte X20. The corresponding number for each key is shown in figure 6.1. If no key is depressed, X20 is set to zero. If a key is depressed, X20 equals the key number as long as the key is depressed. In this way, the user program can monitor X20 for a depressed key, generating leading and trailing edge transitions of key depression with user logic if desired.

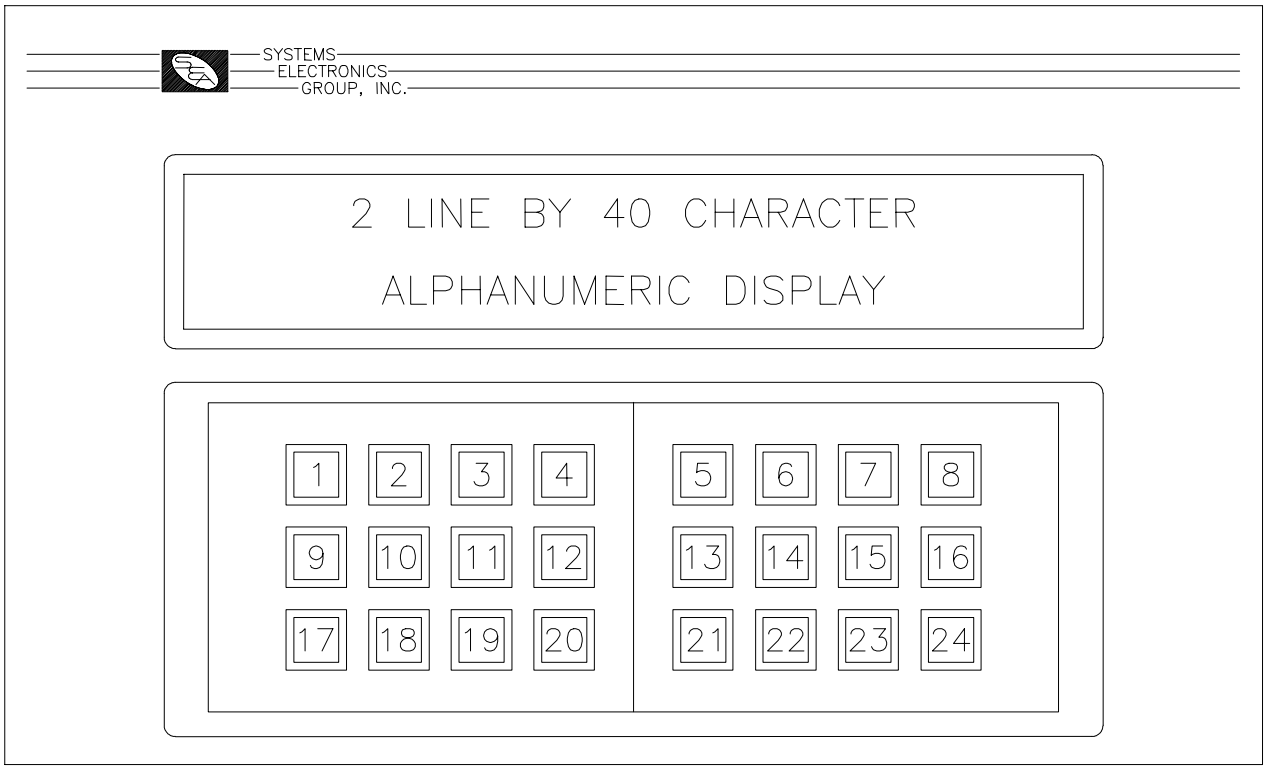


Figure 6.1 – Keypad Key Locations

Refer to the example program in appendix A for examples of decoding the key number depressed.

6.3 SERIAL NETWORK COMMUNICATIONS (sfunc13)

The serial network provides a means for multiple S3012s, S3014s, M4000 modules or D4110 modules (hereafter referred to as nodes) to communicate with each other. The network operates in a master/slave topology. One module acts as the master node and controls all communications on the network. The remaining nodes act as slaves and simply respond to communications requests from the master. The master can send up to 120 consecutive words and receive up to 120 consecutive words from a slave in one command. If data is to be sent from one slave to another slave, it must be done through the master (i.e. the master reads the data from the first slave and then sends it to the second slave).

Up to 32 S3012s, S3014s, M4000 modules, D4110 modules or other S3000 network compatible boards can be installed on one network. These 32 nodes consist of the one master and up to 31 slaves. Each node on the network is assigned a unique network address. This number is a number between 1 and 32. The network address is used to specify which slave the master is communicating to. The network address is set in the D4110 module from the SYSdev Target board Interface menu and is downloaded directly to the module from the IBM PC or compatible running SYSdev. See section 9.3.2.

6.3.1 COMMUNICATING ON THE NETWORK (sfunc13)

System function 13 is used to execute the communications command to the slave. The parameter list of sfunc13 contains:

1) Slave network address to communicate to.
2) Number of words to be sent to slave.
3) Starting address of stack, in master, of words which will be sent to slave (m_srce).
4) Starting address of stack, in slave, where the words are to be stored (s_dest).
5) Number of words to be received from slave.
6) Starting address of stack, in slave, where the words will be sent from (s_srce).
7) Starting address of stack, in master, where the words from the slave will be stored (m_dest).

See section 5.2.10 for a complete description of the above parameters, the general form of sfunc13, and the return values possible with sfunc13.

Note: sfunc13 is used only in the master, the slaves respond to network communications completely transparently. No commands are added to the slave programs in order to implement the serial network. Thus, only one program (the master's) in the entire network has any commands pertaining to network communications.

System function 13 is a simultaneous function such that once it is initiated, program execution continues without waiting for the sfunc to complete. Subsequent calls of sfunc13 result in a return value of "BUSY" until the sfunc completes (return = "DONE") or detects an error (return = "ERROR CODE"). See section 7.4.1 for a description of the serial network error codes. Since sfunc13 is a simultaneous function, the impact on the user application program scan time is negligible when executed. This is also true for the responding slave. Reception and transmission on the serial network occurs concurrently with program execution, no significant increase in the scan time of the slave occurs when a slave is communicated with.

The sequence of events in a serial network comm event are as follows:

1) Master node initiates comm event by executing an sfunc13. Program execution in the master proceeds concurrently with the transmission of the words to the slave.

2) The slave receives the words from the master concurrently with its program execution. Once all words are received from the master, the slave starts transmission of the words that are to be sent from the slave to the master. This also occurs concurrently with the slave program execution.

3) The master receives the words sent from the slave concurrently with its program execution. Once all the words from the slave have been received, the subsequent call to sfunc13 results in a return value of "DONE". Until this step, calls to sfunc13 would have resulted in a "BUSY" return value.

In the case of the D4110, which is equipped with two network serial ports, F105 is used to select which port the sfunc13 is to communicate through. When F105 is set to "0", port #1 is selected. When F105 is set to "1", port #2 is selected. Example #2 provides an example of alternating sfunc13 between the two ports.

In cases where a small number of words are to be transmitted and received, "m_srce" and "m_dest" are generally stacks of "Wxxx" variables in data memory. However, when a large number of words are to be transmitted, the user can load and read the sfunc13 buffer directly and transmit from this buffer without using up "Wxxx" variables in data memory. This is done by loading the sfunc13 buffer (b30aH - b3faH for port #1 and 7e0aH - 7efaH for port #2) using the sfunc08 external write system function and then calling sfunc13 with "m_srce" and "m_dest" equal to 0 (this is done by using an indirect 'W' variable loaded with an address of "0"). Once the sfunc13 is complete (return value = 2), the sfunc13 buffer (b30aH - b3faH for port #1 and 7e0aH - 7efaH for port #2) can be read directly using sfunc08. See example #1 below.

See section 9.3 for details on installing and wiring the network.

Examples:

1) Communicating from the master D4110 port #2 to a slave:

   Master D4110 main program:

```
F105 = 1;                    /* select port #2 */
W080 = 0;                    /* transmit/receive directly */
                             /* from sfunc13 buffer */
B070 = sfunc13(4,10,*W080,W100,5,W090,*W080);
```

execution:

The above command transmits 10 words directly from the sfunc13 buffer (7e0aH thru 7e1eH) in the master D4110 to the slave at network address 4, storing the data in W100 thru W118. The slave then transmits 5 words (W090 thru W098) to the master, leaving this data in the sfunc13 buffer (7e0aH - 7e14H).

Note: The 10 words transmitted to the slave would have to be loaded to the sfunc13 buffer using sfunc08 prior to initiating sfunc13 and that the 5 words received from the slave would have to be read from the sfunc13 buffer using sfunc07 once sfunc13 was complete. The transmission of the data was done concurrently with the program executions of both the master and the slave.

Note: The “F105 = 1;” statement selects serial port #2 and the use of “*W080” in “m_srce” and “m_dest” of the sfunc13 call with W080 = 0 specifies that the sfunc13 buffer is to be written and read directly.

The return value of the sfunc13 is stored in B070. Once the sfunc13 is initiated, the return value of the sfunc13 is “BUSY” (B070 = 1) until the transmission is complete. At that time, the return value is “DONE” (B070 = 2) or an error code (B070 = ERROR CODE) if an error occurred in transmission.

2) Communicating from D4110 port #1 and port #2 alternately:

Master D4110 main program:

```
if (F105 == 0)
  {
  /* network comm through port #1 */
  B070 = sfunc13(4,10,W080,W100,6,W060,W110);
  if (B070 >= 2)              /* sfunc13 DONE or error? */
    {
    if (B070 > 2)             /* error? */
      B071 = B070;            /* yes, save error code */
    F105 = 1;                 /* switch to port #2 */
    }
  }
else
  {
  /* network comm through port #2 */
  B070 = sfunc13(5,10,W150,W100,6,W060,W180);
  if (B070 >= 2)              /* sfunc13 DONE or error? */
    {
    if (B070 > 2)             /* error? */
      B072 = B070;            /* yes, save error code */
    F105 = 0;                 /* switch to port #1 */
    }
  }
```

execution:

The above code alternates communication through ports #1 and #2 (this assumes port #1 and port #2 are connected to two separate networks). When F105 is “0”, communications is performed to slave address 4 on network #1. Once the return value of this sfunc13 is 2 (DONE) or greater (error code), F105 is set to “1” and communications to slave address 5 on network #2 is performed. Likewise, when this sfunc13 return value is 2 (DONE) or greater (error code), F105 is set back to “0” and the comm on network #1 is performed. Communications between the two networks is toggled back and forth in this manner indefinitely.

SECTION 6
USING SYSTEM FUNCTIONS

6.4 USER PORT COMMUNICATIONS

The USER PORT is a general purpose RS-232/RS-422 port available for connection to any RS-232/RS-422 user devices. Communications through the USER PORT is achieved using sfunc10 (USER PORT read) and sfunc11 (USER PORT write). These sfuncs allow any ASCII codes from 0 to 255 to be read from or written to the port. The port is configured as follows: baud rate of 9600, 1 start bit, 8 data bits, 1 stop bit and no parity.

The USER PORT can be selected as either RS-232 or RS-422 (but not both). This is done by setting F105 in the user program to either "0" (RS-232 mode), or to "1" (RS-422 mode).

Note: Different pins on the USER PORT connector are used for the RS-232 lines and RS-422 lines (see Appendix B).

6.4.1 RECEIVING THROUGH THE USER PORT (sfunc10)

Note: sfunc10 functions differently on the D4110 than it does on other S3000 boards or M4000 modules. With other S3000/M4000 boards, sfunc10 must receive the exact number of bytes specified in the sfunc10 call from the user device within a time-out period. In the other S3000/M4000 boards, the return value of sfunc10 specifies whether the sfunc10 is still "BUSY" (has not received the number of bytes specified yet), is "DONE" (all bytes received), or timed-out (not all bytes were sent within the time-out period).

For the D4110, calling sfunc10 essentially enables the USER PORT to receive data from the user device indefinitely. The format of sfunc10 for the D4110 is sfunc10(#max,dest);. Using sfunc10, any number of bytes can be read from a user's ASCII device through the USER port. The return value of the sfunc10 in the D4110 is the number of bytes received since the last sfunc10 call. The bytes that were received are saved in the buffer specified by "dest" in the sfunc10 call.

Note: This buffer should be considered a temporary buffer such that if a string of bytes was expected to be received, the bytes would be "stripped" off this buffer as they were received and saved at some other address locations. The "#max" is the maximum number of bytes that can be saved in the temporary buffer before an overflow occurs. This should be set higher then the maximum number of bytes that can be received between sfunc10 calls.

The operation of sfunc10 in the D4110 allows for a much more flexible format than the previous use of sfunc10 in the other S3000/M4000 boards. With this format, any number of bytes can be sent from the user device at any time. No software handshaking is then required. Also, it is much easier to make the D4110 the slave to the user device than the previous format used.

See the program example in Appendix A for complete example of using sfunc10 to receive a data string from a user device acting as the master. See section 5.2.8 for the general form and parameter list of sfunc10.

Example

1) Receiving through the USER PORT:

```
B080 = sfunc10(10,B100);
```

execution:

The above enables the USER PORT to receive data from the user device through the USER PORT. The sfunc10 specifies that the temporary receive buffer is 10 bytes long located at B100 through B109. The return value in B080 is the number of bytes received since the last call to sfunc10 occurred. As an example, assume 3 bytes were received since the last call of sfunc10, these bytes would be loaded into B100, B101, and B102 as they were received. B080 would be set to 3 after the sfunc10 was executed. With B080 = 3, it is now necessary to read B100, B101, and B102 and store them in some other locations since this data will be lost the next time the sfunc10 is called.

6.4.2 TRANSMITTING THROUGH THE USER PORT (sfunc11)

Using sfunc11, from 1 to 250 consecutive bytes can be transmitted out the USER PORT in one command. System function 11 is a simultaneous function such that once it is initiated, program execution continues without waiting for the sfunc to complete. Subsequent calls of sfunc11 result in a return value of “BUSY” until the sfunc completes (return = “DONE”). Since sfunc11 is a simultaneous function, the impact on the user application program scan time is negligible when an sfunc11 is executed.

The general form of sfunc11 is: sfunc11(#send,srce); where “#send” is the number of bytes to transmit and “srce” is the starting address of the stack of bytes that will be transmitted. When sfunc11 is initiated, the data in “srce” is loaded into the sfunc11 buffer (addresses b202H thru b2fcH) and then transmitted out the port from the sfunc11 buffer.

In cases where a small number of bytes is to be transmitted, “srce” is generally a stack of “Bxxx” variables in data memory. However, when a large number of bytes is to be transmitted, the user can load the sfunc11 buffer directly and transmit from this buffer without using up “Bxxx” variables in data memory. This is done by loading the sfunc11 buffer (b202H - b2fcH) using the sfunc08 external write system function and then calling sfunc11 with “srce” equal to 0 (this is done by using an indirect ‘B’ variable loaded with an address of “0”). See example #2 below.

Examples

1) B080 = sfunc11(6,B120);

execution:

The above transmits the 6 bytes between B120 and B125 out the USER PORT. The return value of sfunc11 is stored in B080. When sfunc11 is first called, the return value will equal “BUSY” (B080 = 1). Subsequent calls of sfunc11 will result in a “BUSY” (B080 = 1) return value until all 6 bytes have been transmitted, at which time a return value of “DONE” (B080 = 2) is obtained.

2) B082 = 0;
B080 = sfunc11(100,*B82);

execution:

The above transmits 100 bytes directly from the sfunc11 buffer out the USER PORT. The “*B82” specifies that B82 contains the address of the start of the “srce” buffer which in this case is 0. This informs sfunc11 that the data to be transmitted is already loaded in the sfunc11 buffer (b202H-b2fcH). This data would have to be loaded using sfunc08 prior to calling the sfunc11. As with the previous example, when the sfunc11 is first initiated, the return value stored in B080 is “BUSY” (B080 = 1). Subsequent calls of sfunc11 result in a return value of “BUSY” until all 100 bytes have been transmitted at which time the return value is “DONE” (B080 = 2).

Note: Program execution is not suspended while sfunc11 is executing. Once initiated, program execution continues with subsequent calls of sfunc11 determining when all the bytes have actually been transmitted. The time it takes for sfunc11 to complete is a function of the number of bytes to be transmitted.

6.5 REAL TIME CLOCK

The D4110 contains a real time clock which provides the current time and date. The time is provided in a 24 hour format (military time) in the form: hours, minutes, and seconds. In the 24 hour format, AM hours are the same as a 12 hour format (for 1:00 AM, hours = 1), PM hours are equal to the 12 hour format PM hours plus 12 (thus for 5:00 pm, hours = 17). The date is provided in the form: month, day of month, and year. The time and date is set in the D4110 via the “Target Board interface” menu of SYSdev. The time and date can be read from the user’s application program running in the D4110 by using the sfunc02 system function.

6.5.1 SETTING THE TIME AND DATE

To set the time and date in the D4110, perform the following:

1) Connect an IBM PC or compatible running SYSdev from the COM port on the PC to the “PROG PORT” on the D4110 using an RS-232 interface cable (see appendix B).

2) Invoke SYSdev from the root directory of the drive SYSdev is loaded on by typing SYSdev<Enter> at the DOS prompt.

3) From the SYSdev shell, select the program that is loaded in the D4110 and press <Enter>.

4) From the Main Development Menu, select “6: Target Board Interface”.

5) From the Target Board Interface menu, select “8: Set time and date in target board”.

6) SYSdev will read the time and date in the D4110 and display these in the “Target Board time” and “Target Board date” fields of the time and date menu. SYSdev will then prompt you to change the time and date, answer “Y” if the time and date is to be changed, “N” if not.

7) If “Y” was answered to the “change time and date” prompt, SYSdev will then prompt for the new time. Enter the time in the form “hours:mins:secs” where hours is 1 to 24, mins is 0 to 59, and secs is 0 to 59 and then press <Enter>.

   Note: The time variables entered are not actually loaded into the D4110 until the <Enter> key is depressed.

8) SYSdev will now prompt for the new date. Enter the date in the form “month-day-year” where month is a number between 1 and 12, day is 1 to 31, and year is the last two digits of the year 0 to 99 and then press <Enter>.

9) SYSdev will then prompt to change the time and date again, answer “N” to exit back to the Target Board Interface menu.

6.5.2 READING THE TIME AND DATE (sfunc02)

System function 02 is used to read the current time and date in the user’s application program running in the D4110. When executed, sfunc02 reads the real time clock and stores the time and date in six consecutive bytes in variable memory as follows: hours, minutes, seconds, month, day of month, year. See section 5.2.2 for complete details on the use of sfunc02.

SECTION 6
USING SYSTEM FUNCTIONS

(This Page Intentionally Left Blank)

SECTION 7
FAULT DETECTION

The D4110 module contains comprehensive fault detection routines which verify the proper operation of the module at all times. If the module detects a fault condition, the "FLT" LED on the front of the module is illuminated and the fault routine is executed. The sources of these faults range from a hardware failure of the module to an error in the user's program (infinite loop, etc.).

7.1 FAULT ROUTINE EXECUTION

When a fault is detected, the following fault routine is executed:

1) User program execution is suspended.
2) If possible, all outputs in the system are disabled.
3) "FAULT" LED on the back of the module is illuminated.
4) "RUN" LED is extinguished.
5) Fault interlock is opened.
6) Fault code representing the detected fault is saved in internal memory of the module for viewing with SYSdev.

The first step in correcting a fault condition (FLT LED "on") in a D4110 module, is viewing the fault code saved inside the module with SYSdev.

7.2 VIEWING FAULT CODES WITH SYSDEV

When a fault occurs, an IBM PC or compatible, running SYSdev, can be connected to the PROG port of the module to view the fault codes. To view the fault codes, perform the following:

1) Connect IBM PC "COM1" port to D4110 "PROG" port using the appropriate cable (see appendix B).

2) Initiate SYSdev from the DOS prompt and select the user program currently loaded in the module.

3) From the main menu, select "Target Board Interface".

4) From the Target Board Interface menu, select "Target Board Fault Codes/Status".

SECTION 7
FAULT DETECTION

The SYSdev fault display reads the fault codes from the module and displays the following:

Target Board Internal Fault Code
1) Curr Flt:
2) Last Flt:
3) Co-cpu slot:
4) Corrective action:

Communications Network Error Codes
5) Current comm error:
6) Last comm error:

Curr Flt: This is the D4110 fault code corresponding to the current detected fault along with a short description of the fault. This fault code is cleared at power-up or optionally by the user after it is displayed in the SYSdev fault display.

Last Flt: This is the last D4110 fault code detected, shown just as the Curr Flt is shown. Unlike the Curr Flt, this fault code is not cleared at power-up. This field retains the last detected fault even when power to the module is cycled. This fault code can only be cleared after it is displayed in the SYSdev fault display.

Co-cpu slot: not used by the D4110 module.

Corrective action: This field contains a short description of the action which can be taken to correct the particular fault that was detected.

Current comm error: This field displays the current serial network comm error along with a short description describing the error. This field is cleared as soon as the current comm error clears.

Last comm error: This field displays the last error displayed in the Current comm error field. Unlike the Current comm error, this field retains the error code even after the error condition clears. This provides a history of the last comm error to occur.

The user has the option of clearing the fault codes when exiting the SYSdev fault display.

7.3 FAULT CODES

The following is a list of the fault codes and descriptions, as displayed in the SYSdev fault display, detected by the module:

Code	Description
00H	No internal fault has occurred
40H	Watchdog timer timeout
41H	Secondary Watchdog timer timeout
42H	Cannot communicate with target board
43H	RAM battery low - program corrupted
44H	Program memory checksum error
45H	User program system fault sfunc09 call
52H	Slave processor did not acknowledge master
59H	Program execution out of bounds
5AH	Address out of program memory range
5BH	Invalid interrupt
5CH	Program invalid - execution suspended
5DH	Program dump timeout - program not sent

7.3.1 WATCHDOG TIMER TIMEOUT (40H and 41H)

The watchdog timeout fault occurs when the main program scan time exceeds 100 milliseconds. The cause of this fault ranges from an error in the user program (unintentional infinite loop entered in the user program, unintentional indirect access to program memory) to a hardware failure of the M4000 module.

Troubleshooting:

1) Check the program for any unintentional infinite loops. These are loops where the exit condition of the loop can never be satisfied. This can occur in “for”, “while” and “do-while” loops. Also check for any “goto” jumps that cause the program to jump to a previous location in the program with no condition to stop executing the jump.

2) Check for any loop instructions that may take longer than 100 milliseconds to execute (a large number of iterations through the loop).

3) When the 40H fault code is displayed in the SYSdev fault display, a field is displayed that reads (PC=xxxxH). The “xxxx” is a four digit hex number which equals the address (program counter) that the program was at when the watchdog timed out. If the program was in an infinite loop, this would give an indication of where the loop was. To see which block this address is in, add an assembly block at the end of the program with just the one word “test” typed into it and then compile the program. The program will compile with no errors, but will assemble with one error (no hex file created). The compiler will create a file named “assem.lst” which is the assembly list file complete with program addresses. This file can be viewed with any text editor or with the MS-DOS “type” command. The numbers in the furthest left column are the program addresses. Locate the address in this file which was displayed in the (PC=xxxxH) field. The assembly instructions for each block are headed with the block number they are in. From this, it is possible to find what block the program was at when the timeout occurred. Remove the assembly block created above to re-compile the program without error.

4) If the problem persists, try another D4100 module to verify if a hardware problem exists.

7.3.2 IBM PC TO M4000 COMMUNICATIONS FAILURE (42H)

If an attempt to read the fault codes from the D4110 module results in an error code of “42H: Cannot communicate with target board”, the PC cannot communicate with the module. This is not an internal D4110 fault, but instead a fault detected by SYSdev. The cause of this fault ranges from catastrophic failure of the module to a misconnection of the PC to the module.

Troubleshooting:

1) Verify that +5VDC power is applied to the module.

2) Verify that the RS-232 cable is connected to “COM1” on the PC and “PROG” port on the module.

3) Verify that the RS-232 cable connecting the PC to the module is wired correctly. See appendix B for the pin out of the cable.

4) If the above verifies, replace the D4110 module and try again. If the problem still persists, verify the “COM1” port for proper operation (see manual from PC manufacture).

7.3.3 INVALID PROGRAM FAULTS (5CH and 5DH)

The “Program invalid” (5CH) fault occurs when the module does not contain a valid user program. This typically occurs when a new module is installed which has never had a user program downloaded to it or after the hardware confidence test is performed, which erases the program memory. The “Program dump timeout” (5DH) fault occurs when program download to the D4110 module is interrupted while program download is in progress.

Troubleshooting:

1) Dump the user program to the D4110 module. These faults will clear once the module is loaded with a valid user program.

2) If re-loading the module with the user program does not clear the fault, replace the D4110 module and try again.

7.3.4 USER PROGRAM sfunc09 SYSTEM FAULT CALL (45H)

This fault code is set when the user program performs an sfunc09(); system function fault call. See the user program for the purpose of the system fault call. See section 5.2.7 for details on sfunc09.

7.3.5 INTERNAL D4110 FAULTS (43H,44H,52H,59H-5BH)

The remainder of the fault codes detected by the D4110 module represent an internal failure of the module. These can range from the RAM battery low to invalid interrupt requests.

Troubleshooting:

1) Perform the hardware confidence test on the D4110 module. It may be desirable to remove the suspect module from the system and to install another module to get the application being controlled back up and running. See section 8 for details on the test.

2) Based on the results of this test, return the module for repair, or re-install the module in system.

SECTION 7
FAULT DETECTION

7.4 SERIAL NETWORK COMMUNICATION ERRORS

Unlike the system faults, the serial network communication errors do not cause the D4110 module to shutdown, but instead are simply logged into the Current and Last comm error registers, with user program execution continuing. The Current comm error represents an error that is present at the time the fault codes are viewed, while the Last comm error represents the last comm error detected. The comm error codes are viewed from the SYSdev fault display, see section 7.2 for more details.

The error codes saved in the Current and Last comm error registers are the same error codes returned from the sfunc13 call. The return values from the sfunc13 calls should be saved in separate 'B' variables such that when a comm error occurs, the slave that it occurred with can be determined.

7.4.1 SERIAL NETWORK COMM ERROR CODES

The following is a list of the detected serial network communication errors:

Code	Description
00H	No network comm error
03H	More than one bus master detected (port #1)
04H	sfunc13 xmitt timeout - no response (port #1)
05H	sfunc13 receive timeout - no response (port #1)
06H	Invalid command received from master (port #1)
07H	Receive overflow (port #1)
08H	Receive collision detected (port #1)
09H	Receive alignment error (bad frame) (port #1)
0AH	Receive CRC error (port #1)
0BH	Unknown (undefined) error (port #1)
0CH	Transmit no acknowledge (port #1)
0DH	Transmit underrun error (port #1)
0EH	Transmit collision detected (port #1)
0FH	Address error (outside data memory) (port #1)
10H	Unexpected slave responding (port #1)
23H	More than one bus master detected (port #2)
24H	sfunc13 xmitt timeout - no response (port #2)
25H	sfunc13 receive timeout - no response (port #2)
26H	Invalid command received from master (port #2)
27H	Receive overflow (port #2)
28H	Receive collision detected (port #2)
29H	Receive alignment error (bad frame) (port #2)
2AH	Receive CRC error (port #2)
2BH	Unkonwn (undefined) error (port #2)
2CH	Transmit no acknowledge (port #2)
2DH	Transmit underrun error (port #2)
2EH	Transmit collision detected (port #2)

2FH Address error (outside data memory) (port #2)
30H Unexpected slave responding (port #2)

7.4.2 NO RESPONSE FROM SLAVE (04H and 05H)

The no response errors occur when the master executes an sfunc13 addressed to a particular slave, but receives no response from that slave. For every execution of sfunc13, the slave will always respond to the request, even if no data is to be sent from the slave to the master. This verifies that the slave did, in fact, receive the data sent to it.

Troubleshooting:

1) Verify that the network continuity is good between the master and the slave. This can be done by observing the “COMM” LEDs on the network interface boards. Every time sfunc13 is executed, the “COMM” LEDs will flash (or be on solid for continuous communications).

2) Verify that the master and all slaves on the network are set to the correct network address they have been assigned. For each node on the network, the address must be a number between 1 and 32 and must be unique. See section 9.7.2.

3) If the problem persists, replace the slave D4110 module where the problem is occurring. Next replace the D4100 master module.

7.4.3 SERIAL NETWORK INTEGRITY ERROR (03H, 06H-0EH, 10H)

The serial network integrity errors occur when corruption of the transmitted frame is detected. The sources of these errors range from multiple masters attempting communications on the network to excessive induced EMI on the network.

Troubleshooting:

1) Verify that only one master is communicating on the network. The master is defined as the node which is executing the sfunc13 system functions. If two nodes are executing sfunc13s simultaneously, a network collision will occur with the corresponding corruption of data.

2) Verify that the network wiring is isolated from other high voltage wiring which could induce EMI into the network. The network should be routed in a conduit separate from other wiring.

3) Replace the slave D4110 module with which the error occurred. If the problem persists, replace the D4110 module at the master node.

7.4.4 ADDRESS OUTSIDE RANGE (0FH)

This error occurs when an attempt to write to memory outside the data memory range occurs in either the master or slave. Verify the corresponding sfunc13 call specifies the proper data range.

(This Page Intentionally Left Blank)

SECTION 8
HARDWARE CONFIDENCE TEST

The hardware confidence test allows the internal D4110 module hardware to be verified for proper operation. The test is resident in all modules and is initiated through SYSdev. The hardware confidence test is the same test used at the factory to initially test the production D4110 modules.

The test is provided to the user to verify whether or not the module hardware is functional; not as a tool to repair the modules. If a fault is detected, the module should be returned to the factory for repair. Any attempt to repair a D4110 module will void the warranty.

8.1 TESTS PERFORMED

The following is a list of the tests performed by the hardware confidence test:

1) Microcontroller RAM test
2) Internal Fault detection test
3) RAM memory test
4) Serial network interface test
5) RS-232 PROG PORT test

Tests 1, 3, 4 and 5 are not optional and are always performed. Test 2 is normally enabled, but can be disabled if desired.

Note: If test 2 is to be performed, the FLT interlock output must be wired to the '-' terminal of both the interrupt input0 and input1 inputs. The '+' terminals of both interrupt input0 and input1 must be wired to the '+' terminal of the power input (+24VDC). Test 2 uses these two inputs to verify the FLT interlock output. Failure to connect these inputs as described will result in a fault detected when test 2 is performed. Test 6 is optional and may be disabled if desired. All tests are automatic and require no interaction once the test is initiated.

Each test performs a complete check of the respective hardware area of the module. If a fault is detected, the test is stopped and a test fault code is displayed to indicate the nature of the hardware failure.

Note: The actual input and output points hardware is not checked with these tests. This can be done using the on-line monitoring mode of SYSdev to view the states of the inputs and set the states of the outputs.

SECTION 8
HARDWARE CONFIDENCE TEST

8.2 PERFORMING THE HARDWARE CONFIDENCE TEST

WARNING: The hardware confidence test should not be performed in a D4110 module installed in a user's control system. Unpredictable output states may result while the test is being performed.

8.2.1 EQUIPMENT REQUIRED

In order to perform the hardware confidence test, the following is required:

1) IBM PC or compatible with SYSdev installed.
2) RS-232 interface cable to connect "COM1" on the PC to "PROG" port on the D4110 module.
3) +5VDC and +24VDC power supplies to power the module.
4) D4110 module to be tested.

8.2.2 EXECUTING THE TEST

To execute the test, perform the following steps:

1) Power up the D4110 module to be tested.
2) Power up PC and enter SYSdev. Enter any user program name to proceed to the SYSdev Main Development Menu.
3) Connect Interface cable to "COM1" on PC and "PROG" port on module.
4) Select "Target Board Interface" from the Main Development Menu then select "Target Board Hardware Confidence Test" from the Target Board Interface menu.
5) Select "D4110 Confidence test" from the confidence test menu. A prompt will be displayed verifying to proceed with the test.

   Note: Proceeding with the test will clear the program and data memory in the module. The user application program will have to be re-downloaded to the module once the test is complete. Press "ESC" to abort the test, any other key to proceed.
6) Select "Perform Test" from the Test Functions Menu to start the test. Once the test is initiated, all tests enabled will be executed repeatedly, starting with test1 thru the last enabled test, until any key is depressed.

If no faults are detected, the tests will continue to execute repeatedly, displaying "test passed" messages after the successful completion of each test. If a fault does occur, the test will stop and display the following:

Fault Code = XX	(test fault code and description)
Address of fault:	(memory address or I/O address where fault occurred)
Actual data at fault:	(data actually obtained at address of fault)
Expected data at fault:	(data that should have been obtained at address of fault)
Diagnostics test number:	(for factory use only)

Once a fault occurs, exit back to the Main Test Menu and re-initiate the test to reset the fault code.

Once testing is complete, exit back to the Main Development Menu. The user application program will now have to be re-downloaded to the D4110 module.

8.3 INTERACTIVE INTERFACE

The interactive interface menu contains selections to read the fault code (same as displayed when a fault is detected), perform diagnostics routines (for use by the factory only) and to read and write, via the RS-232 ports, to any address in the module. In general, all these selections are for factory use and are of little significance to the user.

SECTION 8
HARDWARE CONFIDENCE TEST

(This Page Intentionally Left Blank)

SECTION 9
INSTALLATION

The following sections provide information on mounting and wiring the D4110 module as well as a description of the power-up sequence.

Note: All wiring is implemented with removable field wiring connectors. The connectors are removed by gently pulling the connectors from the socket. Install the connectors by firmly seating the connector to the socket, observing the proper polarity of the connector. Refer to appendix C for the pin-outs of the various connectors on the D4110 module.

9.1 MOUNTING THE D4110

The D4110 module was designed to mount from the front in the door of the user's panel. Use the recommended cut-out in figure 9.7. Remove the field wiring connectors (INPUTS, OUTPUTS, and IN0/IN1) from the module and slide the D4110 into the cut-out from the front. Attach the D4110 to the door with the supplied hardware. If the P4110-XXX power pack is to be used with the D4110, it is mounted on the right side (from the back) of the D4110 on the two right mounting studs.

9.2 WIRING INPUT POWER

The D4110 module is powered with +5VDC, +-5% power. The P4110-XXX power pack provides this +5VDC power. To connect the P4110-XXX power pack to the D4110, mate the female power pack cable to the male +5VDC power input on the D4110 (the black lead of the P4110-XXX cable is +5VDC, the white lead is COM). If a P4110-XXX power pack is not used, +5VDC power can be wired directly into the D4110 using a 09-50-3031 Molex connector housing to mate with the male connector of the D4110. Be sure to observe the proper polarity to the +5VDC power input.

9.3 WIRING 10-30VDC DIGITAL INPUTS

The digital inputs are 10-30VDC sourcing (true high) inputs which are used to interface to sourcing application inputs such as proximity sensors, push-buttons, etc. The inputs are internally mapped to terminals on the input connected numbered with the corresponding input number. All inputs are commoned to the 'C' (common or return) terminal of the connector.

Note: The inputs are optically isolated, thus the common of the input voltage does not have to be commoned with the +24VDC power used to power the module. Figure 9.1 shows typical input wiring.

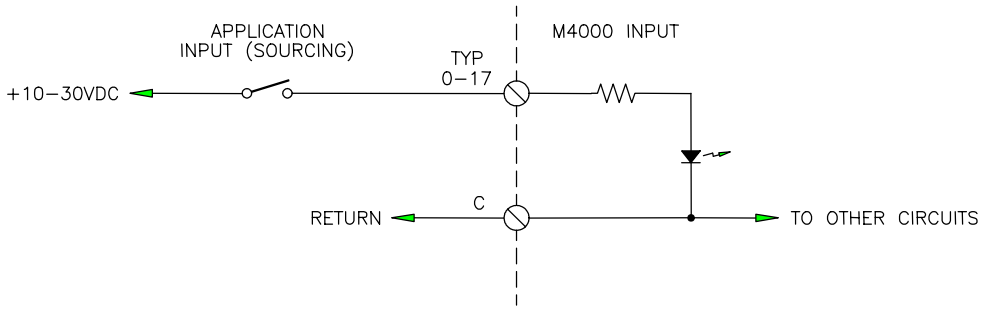


Figure 9.1 – Typical M4000 Input Wiring

9.4 WIRING INTERRUPT INPUTS

Interrupt input0 and input1 are 12-30VDC differential inputs which can be wired as sourcing (true high), sinking (true low), or as true differential inputs (driven by a differential output). Each input is provided with a '+' and '-' terminal. Figure 9.2 shows wiring examples of all three types of input configurations.

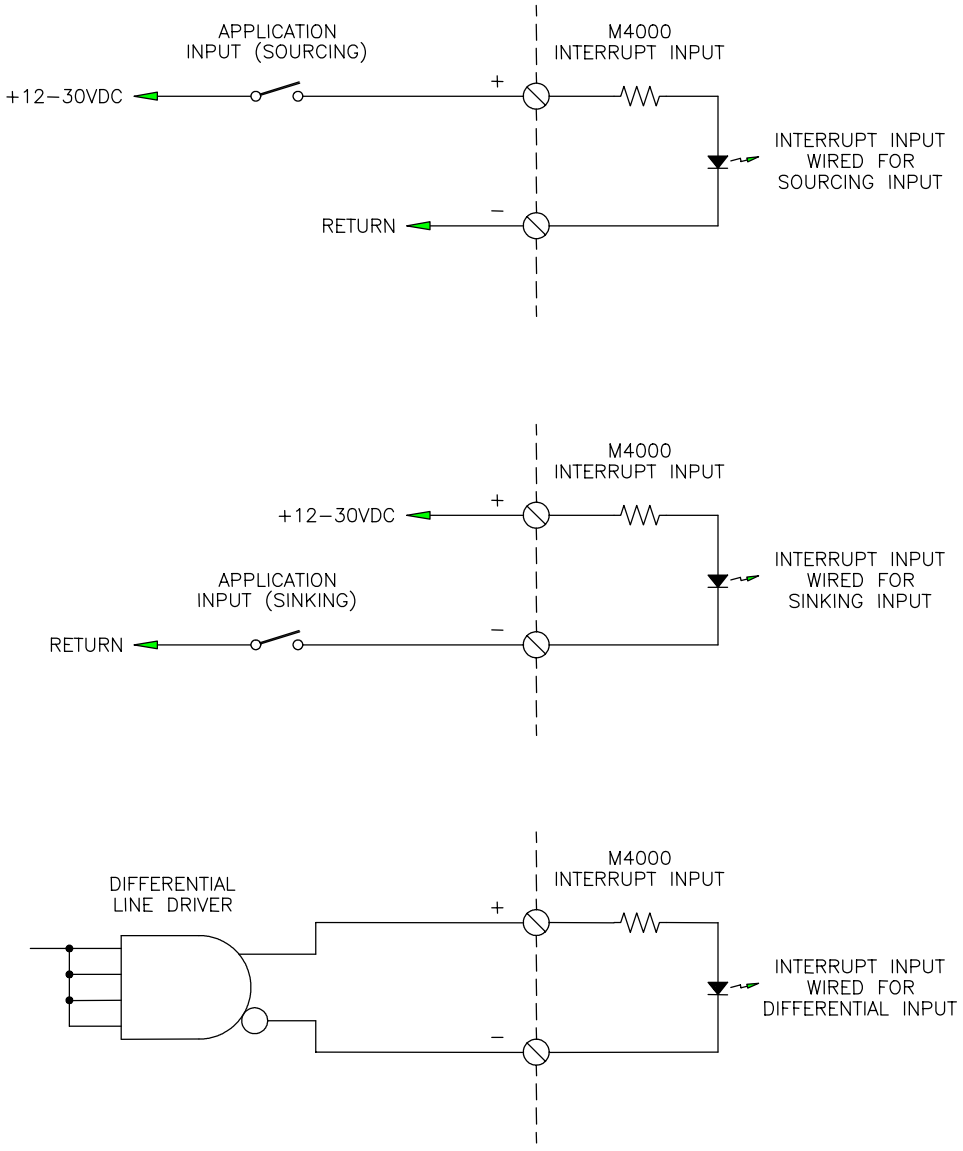


Figure 9.2 – Typical Interrupt Input Wiring

9.5 WIRING 10-30VDC DIGITAL OUTPUTS

The digital outputs are 10-30VDC sourcing (true high) which are used to interface to the application outputs such as solenoids, lamps, PLC inputs, etc. Each output is rated at 1 amp DC (continuous) with an inrush (pulsed) current drive capability of 5 amps for 100msec. The outputs do not contain output fusing or short circuit protection, therefore external fusing should be provided. Power for the digital outputs is wired to the '+' (power) and 'C' (common or return) terminals on the output connector. Be sure to observe the proper polarity of the '+' power and 'C' wiring, otherwise damage to the module may occur.

Note: The outputs are optically isolated, thus the 10-30VDC power applied to the output connector does not have to be commoned with the +24VDC power used to power the module. Figure 9.3 shows an example of the typical output wiring.

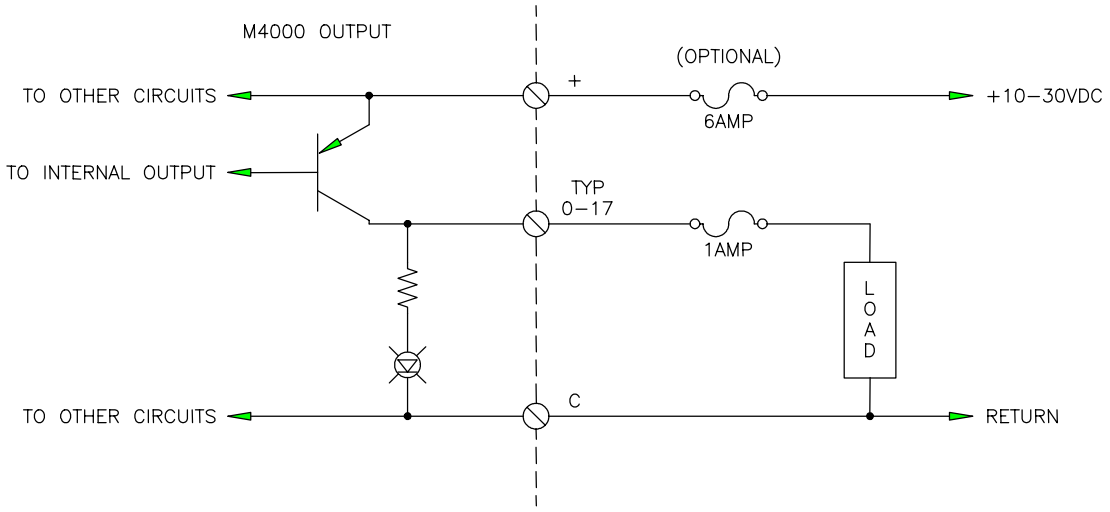


Figure 9.3 – Typical Output Wiring

9.6 WIRING THE FAULT INTERLOCK

The fault interlock is a +24VDC sinking (true low) output which can be interfaced to an external relay or PLC input to indicate a fault condition with the D4110 module. The output is capable of sinking 500 milliamps. The fault output is "on" (true low - sinking current) when the module is executing the user program properly. If a fault condition is detected, the fault output is turned "off" (high). Figure 9.4 shows the fault output wired to a +24VDC relay. This relay could be interlocked with the digital outputs power to remove power from the outputs if the module was to fault out.

Note: +24VDC power (+24VDC at +V terminal, and COM at RET terminal) is required at the fault output connector in order for the fault output to function.

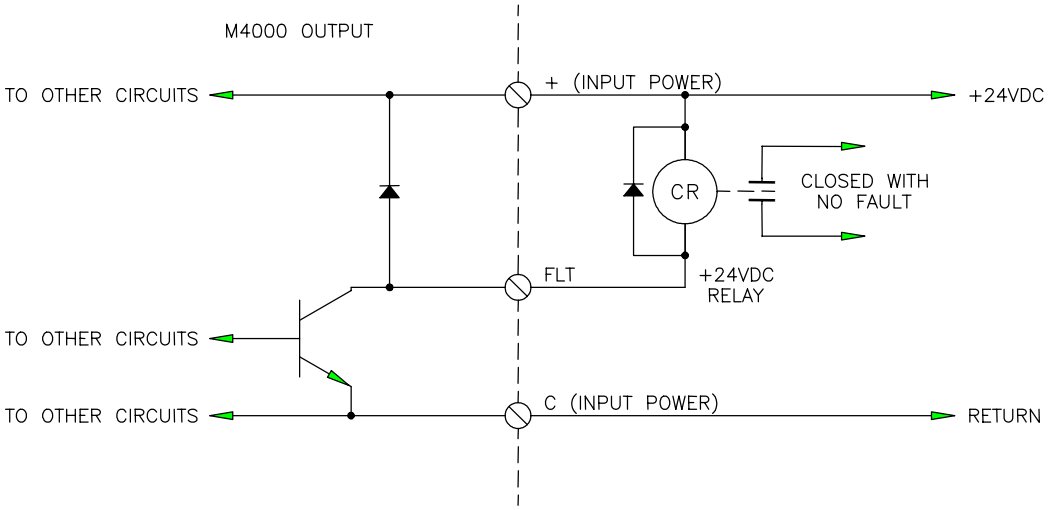


Figure 9.4 – Typical Fault Interlock Wiring

SECTION 9
INSTALLATION

9.7 SERIAL NETWORK INSTALLATION

The serial network installation consists of wiring the network and setting each module on the network with a unique network address. Up to 32 S3000/M4000/D4110 modules can be installed on one network.

9.7.1 WIRING THE SERIAL NETWORK

Refer to figure 9.5 for a typical schematic of the network and for the pin outs of the network interface connectors. When wiring the network, the following rules must be followed:

1) Wire the network using Belden #9182 single-shielded twisted pair cable or an equivalent data communications cable meeting the following spec:

Wire gauge:	22AWG
Nom. impedance:	150 ohms/ft.
Nom. attenuation at 1MHZ:	0.004 db/ft.
Twisted pair, single-shielded	

2) The total wire length of the network cannot exceed 1000 ft. if the network baud rate is set to 344KBPS, 2000 ft. for 229KBPS, and 4000 ft. for 106KBPS. See section 9.7.2 for details on setting the network baud rate.

3) The maximum number of nodes connected to one network is limited to 32 nodes.

4) The shield of the cable should be carried through the entire network, using the shield tie points on the interface connectors to achieve this. The shield tie-points on the connectors are not internally tied to anything, they are strictly tie points. One of these tie points should then be tied to earth ground.

5) The two extreme ends of the network should be terminated with 150 ohm resistors as shown in figure 9.5.

6) The network wiring should be isolated from other high voltage wiring by routing the network in a separate conduit dedicated to the network.

7) The network should be wired directly to the network comm port connectors. No intermediate terminations or splices should be used. The network should be wired in a direct connect topology as shown, not in multi-drop or cluster topologies.

   Note: The network comm interface connectors contain two sets of + and - terminals. The two sets of terminals are tied together internally on the module (+ to +, - to -) and are provided as tie points to ease wiring. Communications across the network will continue even if one of the nodes has failed provided all the connectors are installed in their respective module. However, if a connector is pulled from it's module, communications to the modules downstream will be lost (the internal tie point will be broken). If it is desired, this situation can be avoided by wiring the connector as shown in figure 9.6.

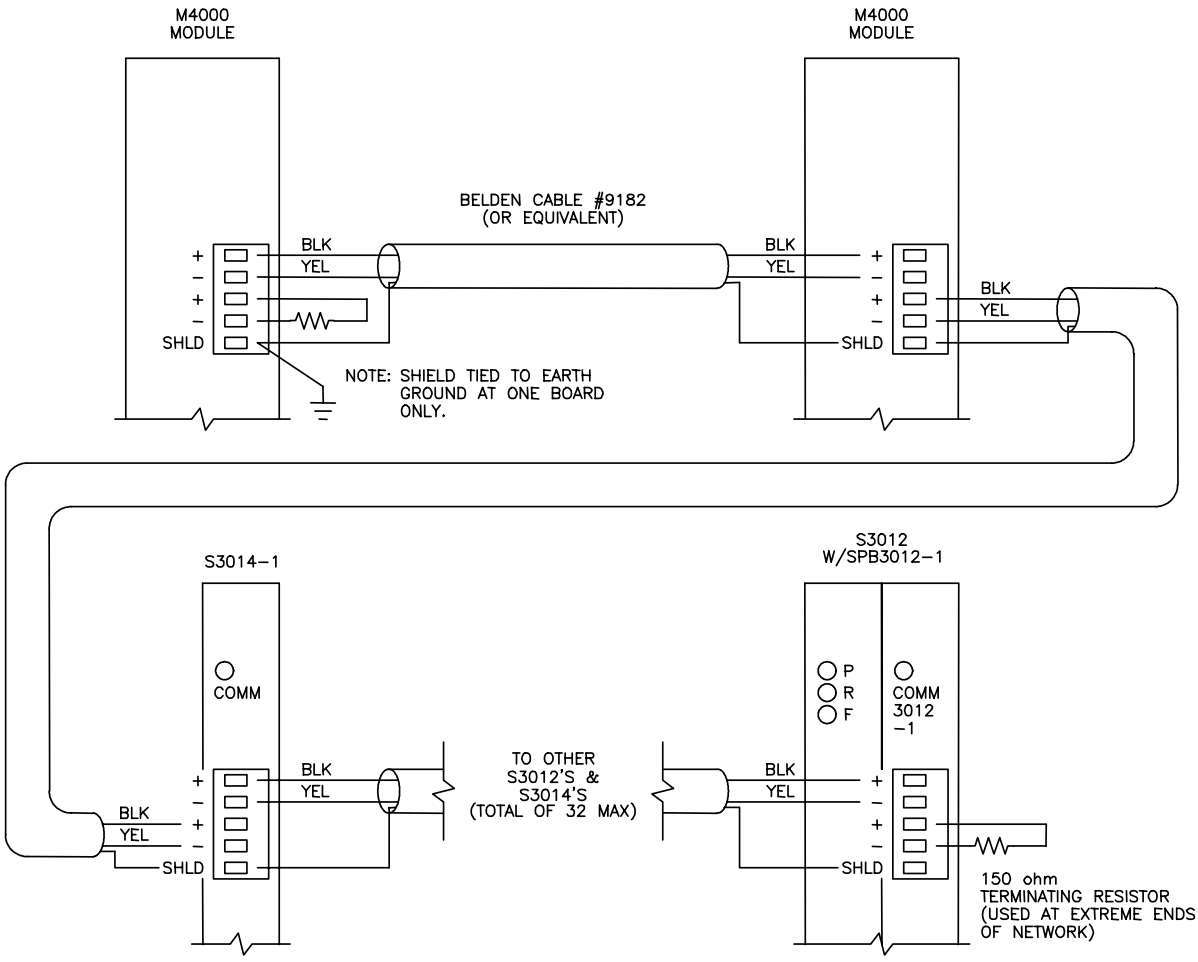


Figure 9.5 – Typical Network Wiring

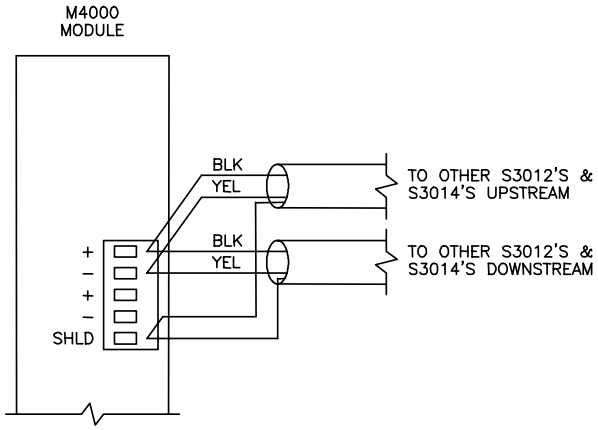


Figure 9.6 – Alternative Serial Connector Wiring

9.7.2 SETTING THE NETWORK ADDRESSES

Each S3000/M4000/D4110 module on the network must be set with a unique network address between 1 and 32. This is how the modules can distinguish one node from another. The D4110 has two serial network ports which requires that both ports be assigned network addresses. To set the network addresses for the D4110 module, perform the following:

1) Connect an IBM PC or compatible running SYSdev from “COM1” on the PC to “PROG” port on the D4110 using the RS-232 interface cable (see appendix B).

2) From the SYSdev Main Development Menu, select “Target Board Interface”.

3) From the Target board Interface Menu, select “Target board Network Address”.

4) SYSdev will read the current network addresses from the D4110 module and display them in the network display. If the network addresses are to be changed, follow the directions displayed and enter the new addresses.

The above steps must be done for all S3000/M4000/D4110 modules on the network. This is true when the network is first installed, and when a new module is added or replaced (that module must have the network address set it in).

9.8 POWER-UP SEQUENCE OF D4110 MODULES

Once all connectors are wired and re-installed in their respective sockets, apply +5VDC power to the module. The power-sequence occurs as follows:

1) At initial power-up, the module is reset for approximately half a second. During this time, the fault interlock will be “off”, and the “FLT” LED will all be “on”. The outputs of the D4110 will also be “off” during this reset.

2) Once the reset cycle is complete, the module will begin to execute the program previously loaded. The fault interlock will turn “on” (sink true low), the "RUN" LED will turn "on", and the “FLT” LED will extinguish. The outputs will then be activated in the states as controlled by the user program.

3) If a user program has not been loaded (new module or module which the confidence test has just been executed), the “FLT” LED will stay “on” and the “RUN” LED will extinguish. Download the user program and data files to the module. The “RUN” LED will flash while the user program is downloading. When the download is complete, the “FLT” LED will extinguish and the “RUN” LED will turn “on”. If the “FLT” LED turns “on” after the download is complete, read the fault code in the D4110 module (see section 7). See the SYSdev program manual for details on downloading programs to the D4110 module.

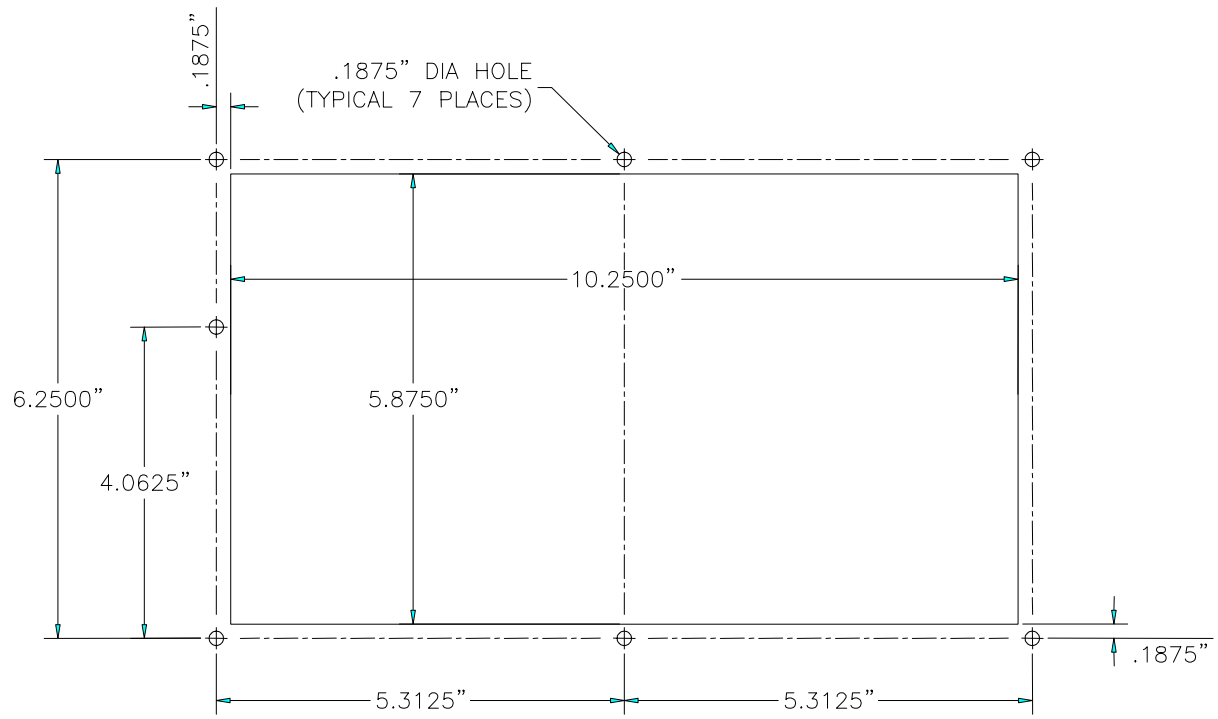


Figure 9.7 – Recommended Panel Cut-out

SECTION 9
INSTALLATION

(This Page Intentionally Left Blank)

APPENDIX A
PROGRAMMING EXAMPLE

D4110 Functional Test:
SYS51 System Configuration: D4110T.LCF

System Configuration

Target Board:	D4110 16-input/16-output Display Module
Network Baud Rate:	344KBPS
Input0 Interrupt Enable:	Yes
Input1 Interrupt Enable:	Yes
Timed Interrupt Enabled:	Yes
Timed Interrupt Time:	1.0msec

APPENDIX A
PROGRAMMING EXAMPLE

```
D4110T.LIN

************************************************************************
block:  1 - High-level

 0:B73 = f0H;        /* expected char = SOF */
 1:
 2:for (B77 = 255; B77 > 0; ++B77)     /* initiate delay */
 3:    sfunc03();
 4:
 5:/* display default test message */
 6:sfunc08(b402H,15H);     /* clear display/cursor home */
 7:sfunc08(b403H,0eH);     /* cursor invisible */
 8:sfunc08(b404H,12H);     /* data entry mode */
 9:sfunc08(b455H,0dH);     /* carriage return (end of message) */
10:sfunc04(b405H," TEST IN PROGRESS - NO FAULTS DETECTED  ");
11:sfunc04(b42dH,"         PRESS KEY NUMBER: 01           ");
12:B68 = 0;
13:B69 = 0;
14:
15:B88 = 0;
16:while (sfunc18(84,*B88) < 2)
17:    ;
18:
19:W140 = 1234H;     /* initialize serial port test registers */
20:W142 = 5678H;
21:W144 = abcdH;
22:
23:W158 = 4321H;
24:W160 = 8765H;
25:W162 = dcbaH;
26:
27:/* I/O test variables initialize */
28:W180 = &B100;     /* flt_ptr */
29:B200 = 0;         /* point number */
30:W184 = 1;         /* bit mask */
31:F51 = 1;          /* start with test #1 */
32:W186 = 1;         /* outputs image */
33:


F051  (test#1 )  test #1  fail    "off"
B068  (Des Key)  KeyTest Desired   Key
B069  (KeyPrev)  KeyTest Desired  Prev
B073  (expchar)          expectd  char
B077  (reg(i) )  general registr  (i)
B088  (sf18ptr)  sfunc18 address pointer
B100  (flt_i0 )  input0   fault   code
B200  (point# )   test    point   number
W140  (msrce1 )  sfunc13  xmit   buffer
W142  (msrce2 )  sfunc13  xmit   buffer
W144  (msrce3 )  sfunc13  xmit   buffer
W158  (ssrce1 )  sfunc13  slave   xmit
W160  (ssrce2 )  sfunc13  slave   xmit
W162  (ssrce3 )  sfunc13  slave   xmit
W180  (flt ptr)           fault  pointer
W184  (bitmask)  point    bit     mask
W186  (outimge)          outputs  image
```

APPENDIX A
PROGRAMMING EXAMPLE

APPENDIX A
PROGRAMMING EXAMPLE

```
D4110T.LMN

************************************************************************
block:  1 - High-level

 0:if (F2 == 1)                   /* update display? */
 1:   {
 2:   if (sfunc18(5,*B88) == 2)
 3:      F2 = 0;
 4:   }
 5:
 6:if (X20 != 0 && B66 == 0)      /* key depressed L.E.S.S.? */
 7:   F3 = 1;                     /* set "New Key" */
 8:B66 = X20;
 9:
10:if (B68 != B69 && F4 == 1)     /* new desired key? */
11:   {
12:   B67 = (B68/10) + 48;        /* convert to ascii */
13:   sfunc08(b405H,B67);
14:   B67 = (B68%10) + 48;        /* convert to ascii */
15:   sfunc08(b406H,B67);
16:   sfunc08(b402H,1bH);         /* position cursor */
17:   sfunc08(b403H,5bH);         /* line 2 - loc 27 */
18:   sfunc08(b404H,12H);         /* data entry mode */
19:   B88 = 0;
20:   F2 = 1;
21:   B69 = B68;
22:   }
23:
24:/* advance to next key */
25:if (B68 == X20 && F4 == 1)     /* key prompted for depressed? */
26:   {
27:   ++B68;                      /* advance to next key */
28:   if (B68 > 24)               /* all keys tested? */
29:      B68 = 1;                 /* start over */
30:   }
31:


F002  (updtdsp)          update  display
F003  (New Key)    New     Key   Pressed
F004  (KeyDone)    Key    Press   Done
B066  (old key)    old     key    number
B067  (ascikey)   ascii    key   number
B068  (Des Key)  KeyTest Desired   Key
B069  (KeyPrev)  KeyTest Desired  Prev
B088  (sf18ptr)  sfunc18 address pointer
X020  (  key  )  keypad   key    number
```

```
D4110T.LMN

*************************************************************************
block:  2 - Ladder

      New                                                                 Key
      Key                                                                Press
    Pressed      Timer                                                   Done
     F003     +---------+                                               F004
 0:+--]/[--+--|         |--+-------+-------+-------+-------+-------+--( )--
    New Key   | P:#00010|                                              KeyDone
              | TB:0.01 |
              | A:B065  |
              |(KeyDely)|
              |   Key   |
              |  Press  |
 1:           |  Delay  |
              +---------+

                                                                         New
                                                                         Key
                                                                       Pressed
                                                                        F003
 2:+-------+-------+-------+-------+-------+-------+-------+-------+--(U)--
                                                                       New Key
```

APPENDIX A
PROGRAMMING EXAMPLE

```
D4110T.LMN

*************************************************************************
block:  3 - High-level

The USER PORT is tested by an S3012 which is connected from the S3012
USER port to the D4110 USER PORT. The S3012 then acts as the master
and transmits a frame of known data to the D4110 which compares the
received from to what was expected and then transmits a known response
back the S3012 if the received frame was valid. If the S3012 does not
receive the response or the response is incorrect, the S3012 will fault
out indicating the USER PORT on the D4110 is not functioning properly.

 0:/* USER PORT test (sfunc10/sfunc11) */
 1:
 2:F104 = 0;                      /* RS-232 USER PORT mode */
 3:B70 = sfunc10(8,B90);          /* read user port receive buffer */
 4:if (B70 != 0 && B70 < 17)      /* any bytes received and no overflow? */
 5:   {
 6:   B71 = &B90;                 /* point to 1st addr of rcve buf */
 7:   while ( B70 > 0)            /* read all bytes received */
 8:      {
 9:      B72 = *B71;              /* read nth byte */
10:      ufunc10();               /* call receive_frame() */
11:      ++B71;
12:      --B70;
13:      }
14:   }
15:
16:if (F10 == 1)                  /* xmit response? */
17:   ufunc11();                 /* call transmit_frame() */
18:


F010  (xmitrsp)   xmit   respons in prog
F104  (RS-422 )  RS-422   PORT    mode
B070  (sf10rtn)          sfunc10 return
B071  (bufpntr)   data   buffer  pointer
B072  (rcvchar)          receivd  char
B090  (rcvebuf)   USER   receive buffer
THRU
B097  (-------)  ------- ------- -------
```

```
D4110T.LMN

************************************************************************
block:  4 - High-level

The serial ports are tested by connecting port #1 to port #2 and
communicating from port #1 (master) to port #2 (slave).

 0:/* serial ports test */
 1:
 2:F105 = 0;   /* select port 1 */
 3:B137 = sfunc13(2,3,W140,W152,3,W158,W146);
 4:if (B137 == 2)          /* test values xmitted */
 5:   {
 6:   if (W152 != 1234H)   /* test words received from master side */
 7:      B136 = 11H;       /* slave receive error */
 8:   else if (W154 != 5678H)
 9:      B136 = 11H;
10:   else if (W156 != abcdH)
11:      B136 = 11H;
12:   else if (W146 != 4321H)    /* test words received from slave side */
13:      B136 = 12H;       /* master receive error */
14:   else if (W148 != 8765H)
15:      B136 = 12H;
16:   else if (W150 != dcbaH)
17:      B136 = 12H;
18:   else
19:      ;
20:   if (B136 != 0)
21:      {
22:      B99 = 2;
23:      ufunc09();
24:      }
25:   }
26:else if (B137 >= 3)
27:   {
28:   B136 = B137;         /* save fault return value */
29:   B99 = 2;             /* serial port error code */
30:   ufunc09();           /* call fault routine */
31:   }
32:else
33:   ;
34:if (B136 > B139)
35:   {
36:   B139 = B136;
37:   }
38:
```

APPENDIX A
PROGRAMMING EXAMPLE

```
F105  (serport)  serial   port   select
B099  (fltcode)           fault   code
B136  (ser err)  serial   port    error
B137  (sf13ret)  sfunc13 return   value
B139  (-------)  ------- ------- -------
W140  (msrce1 )  sfunc13  xmit   buffer
W142  (msrce2 )  sfunc13  xmit   buffer
W144  (msrce3 )  sfunc13  xmit   buffer
W146  (mdest1 )  sfunc13  rcve   buffer
THRU
W150  (mdest3 )  sfunc13  rcve   buffer
W152  (sdest1 )  sfunc13  slave  receive
W154  (sdest2 )  sfunc13  slave  receive
W156  (sdest3 )  sfunc13  slave  receive
W158  (ssrce1 )  sfunc13  slave   xmit
```

```
D4110T.LMN

************************************************************************
block:  5 - High-level

 0:/* call fault routine if I/O test fault occurred */
 1:
 2:if (B99 >= 3 && B99 <= 8)
 3:   ufunc09();
 4:
 5:sfunc02(B210);          /* read time and date */
 6:


B099  (fltcode)           fault   code
B210  ((hours))           time   (hours)
B211  ( (min) )           time    (min)
B212  ( (sec) )           time    (sec)
B213  ( (mon) )           date    (mon)
B214  ( (day) )           date    (day)
B215  ((year) )           date   (year)
```

APPENDIX A
PROGRAMMING EXAMPLE

```
D4110T.LTD

*************************************************************************
block:  1 - Ladder

    test #1
     fail   test #1                                                 test #1
     "off"   iter        Timer                                      iter
     F051    F060     +---------+                                   F060
 0:+--] [--+--]/[--+--|         |--+-------+-------+-------+-------+--( )--
    test#1  test#1p   | P:#00002|                                   test#1p
                      | TB:0.01 |
                      | A:B170  |
                      |(t1 time)|
                      | test #1 |
                      |  iter   |
 1:                   |  time   |
                      +---------+

    test #2
     fail   test #2                                                 test #2
     "on"    iter        Timer                                      iter
     F052    F061     +---------+                                   F061
 2:+--] [--+--]/[--+--|         |--+-------+-------+-------+-------+--( )--
    test#2  test#2p   | P:#00002|                                   test#2p
                      | TB:0.01 |
                      | A:B171  |
                      |(t2 time)|
                      | test #2 |
                      |  iter   |
 3:                   |  time   |
                      +---------+

    test #3
    pattern test #3                                                 test #3
     test    iter        Timer                                      iter
     F053    F062     +---------+                                   F062
 4:+--] [--+--]/[--+--|         |--+-------+-------+-------+-------+--( )--
    test#3  test#3p   | P:#00002|                                   test#3p
                      | TB:0.01 |
                      | A:B172  |
                      |(t3 time)|
                      | test #3 |
                      |  iter   |
 5:                   |  time   |
                      +---------+
```

```
D4110T.LTD

************************************************************************
block:  2 - High-level

 0:B188 = B062;
 1:B189 = B063;
 2:if (B99 != 0)
 3:   goto io_fault;
 4:
 5:/* inputs "off" test */
 6:
 7:if (F60 == 1)
 8:   {
 9:   if (W188 & W184 == 0)      /* input fail "off" */
10:      {
11:      *B180 = 3;              /* fault = "input failed off" */
12:      B099 = 3;               /* I/O fault occurred */
13:      goto io_fault;
14:      }
15:
16:   /* next I/O point */
17:   W184 = W184 << 1;          /* bit mask */
18:   ++W180;                    /* fault array pointer */
19:   ++B200;                    /* point number */
20:   if (W180 == &B108)
21:      {
22:      W180 = &B110;
23:      B200 = 10;
24:      }
25:
26:   W186 = W184;         /* outputs = bit_mask */
27:   /* start next test if done */
28:   if (B200 == 18)
29:      {
30:      W180 = &B100;     /* flt_ptr = 1st point fault address */
31:      B200 = 0;         /* point number */
32:      F51 = 0;          /* test #1 done */
33:      F52 = 1;          /* start test #2 */
34:      W184 = 1;         /* bit mask = 1st point */
35:      W186 = 0;         /* clear output image */
36:      }
37:   }
38:
```

APPENDIX A
PROGRAMMING EXAMPLE

```
F051  (test#1 )  test #1  fail    "off"
F052  (test#2 )  test #2  fail    "on"
F060  (test#1p)          test #1  iter
B062  (iinput0)  intrpt  inputs  image
B063  (iinput1)  intrpt  inputs  image
B099  (fltcode)           fault   code
B100  (flt_i0 )  input0   fault   code
B108  (in0fcnt)   in0     fail    count
B110  (flt_i10)  input10  fault   code
B180  (flt ptr)           fault  pointer
B188  (input0 )          inputs  image
B189  (input1 )          inputs  image
B200  (point# )   test    point   number
W180  (flt ptr)           fault  pointer
W184  (bitmask)  point    bit     mask
W186  (outimge)          outputs  image
W188  (input0 )          inputs  image
```

```
D4110T.LTD

************************************************************************
block:  3 - High-level

 0:/* inputs fail "on" test */
 1:
 2:if (F61 == 1)
 3:   {
 4:   W186 = 0;            /* outputs all cleared */
 5:
 6:   if (W188 & W184 != 0)      /* input fail "off" */
 7:      {
 8:      *B180 = 4;              /* fault = "input failed on" */
 9:      B099 = 4;               /* I/O fault occurred */
10:      goto io_fault;
11:      }
12:
13:   /* next I/O point */
14:   W184 = W184 << 1;          /* bit mask */
15:   ++W180;                    /* fault array pointer */
16:   ++B200;                    /* point number */
17:   if (W180 == &B108)
18:      {
19:      W180 = &B110;
20:      B200 = 10;
21:      }
22:
23:   /* start next test if done */
24:   if (B200 == 18)
25:      {
26:      W180 = &B100;     /* flt_ptr = 1st point fault address */
27:      B200 = 0;         /* point number */
28:      F52 = 0;          /* test #2 done */
29:      F53 = 1;          /* start test #3 */
30:      W184 = 1;         /* bit mask = 1st point */
31:      W186 = 1;         /* output image for next test */
32:      }
33:   }
34:


F052  (test#2 )  test #2  fail    "on"
F053  (test#3 )  test #3 pattern  test
F061  (test#2p)          test #2  iter
B099  (fltcode)           fault   code
B100  (flt_i0 )  input0   fault   code
B108  (in0fcnt)   in0     fail    count
B110  (flt_i10)  input10  fault   code
B180  (flt ptr)           fault  pointer
B200  (point# )   test    point   number
W180  (flt ptr)           fault  pointer
W184  (bitmask)  point    bit     mask
W186  (outimge)          outputs  image
W188  (input0 )          inputs  image
```

APPENDIX A
PROGRAMMING EXAMPLE

```
D4110T.LTD

************************************************************************
block:  4 - High-level

 0:/* inputs pattern test */
 1:
 2:if (F62 == 1)
 3:   {
 4:   if (W188 != W184)          /* input fail "off" */
 5:      {
 6:      *B180 = 5;             /* fault = "input pattern fail" */
 7:      B099 = 5;              /* I/O fault occurred */
 8:      goto io_fault;
 9:      }
10:
11:   /* next I/O point */
12:   W184 = W184 << 1;          /* bit mask */
13:   ++W180;                    /* fault array pointer */
14:   ++B200;                    /* point number */
15:   if (W180 == &B108)
16:      {
17:      W180 = &B110;
18:      B200 = 10;
19:      }
20:
21:   /* start next test if done */
22:   if (B200 == 18)
23:      {
24:      W180 = &B100;     /* flt_ptr = 1st point fault address */
25:      B200 = 0;         /* point number */
26:      F53 = 0;          /* test #3 done */
27:      F54 = 1;          /* start test #4 */
28:      W184 = 1;         /* bit mask = 1st point */
29:      B179 = 0;         /* reset filter delay time */
30:      }
31:   W186 = W184;         /* outputs = bit_mask */
32:   }
33:


F053  (test#3 )  test #3 pattern  test
F054  (test#4 )  test #4 filter   delay
F062  (test#3p)          test #3  iter
B099  (fltcode)           fault   code
B100  (flt_i0 )  input0   fault   code
B108  (in0fcnt)   in0     fail    count
B110  (flt_i10)  input10  fault   code
B179  (filter )   input   filter  delay
B180  (flt ptr)           fault  pointer
B200  (point# )   test    point   number
W180  (flt ptr)           fault  pointer
W184  (bitmask)  point    bit     mask
W186  (outimge)          outputs  image
W188  (input0 )          inputs  image
```

```
D4110T.LTD

************************************************************************
block:  5 - High-level

 0:/* inputs filter delay test */
 1:
 2:if (F54 == 1)
 3:   {
 4:   ++B179;                    /* input filter delay time */
 5:
 6:   if (B179 >= 5)             /* input fail "off" */
 7:      {
 8:      *B180 = 6;              /* fault = "input delay too long" */
 9:      B099 = 6;               /* I/O fault occurred */
10:      goto io_fault;
11:      }
12:
13:   if (W188 == W184)
14:      {
15:      /* next I/O point */
16:      W184 = W184 << 1;          /* bit mask */
17:      ++W180;                    /* fault array pointer */
18:      ++B200;                    /* point number */
19:      if (W180 == &B108)
20:         {
21:         W180 = &B110;
22:         B200 = 10;
23:         }
24:
25:      /* start next test if done */
26:      if (B200 == 18)
27:         {
28:         W180 = &B100;     /* flt_ptr = 1st point fault address */
29:         B200 = 0;         /* point number */
30:         F54 = 0;          /* test #4 done */
31:         F55 = 1;          /* test complete */
32:         W184 = 1;         /* bit mask = 1st point */
33:         }
34:      W186 = W184;         /* outputs = bit_mask */
35:      B179 = 0;
36:      }
37:   }
38:


F054  (test#4 )  test #4 filter   delay
F055  (tstcomp)   tests   comp   (wait)
B099  (fltcode)           fault   code
B100  (flt_i0 )  input0   fault   code
B108  (in0fcnt)   in0     fail    count
B110  (flt_i10)  input10  fault   code
B179  (filter )   input   filter  delay
B180  (flt ptr)           fault  pointer
B200  (point# )   test    point   number
W180  (flt ptr)           fault  pointer
W184  (bitmask)  point    bit     mask
W186  (outimge)          outputs  image
```

APPENDIX A
PROGRAMMING EXAMPLE

```
W188  (input0 )          inputs  image
```

```
D4110T.LTD

************************************************************************
block:  6 - Ladder

     tests
     comp                                                          tests
    (wait)       Timer                                           comp
     F055     +---------+                                        F056
 0:+--] [--+--|         |--+-------+-------+-------+-------+-------+--( )--
    tstcomp   | P:#00050|                                       tstcomp
              | TB:0.01 |
              | A:B173  |
              |(tstcomp)|
              |  test   |
              |  comp   |
 1:           |  time   |
              +---------+
```

APPENDIX A
PROGRAMMING EXAMPLE

```
D4110T.LTD

************************************************************************
block:  7 - High-level

 0:if (F55 == 1)
 1:   {
 2:   W186 = ffffH;        /* all outputs "on" */
 3:   if (F56 == 1)        /* test complete display done? */
 4:      {
 5:      W180 = &B100;     /* reset test variables for first test */
 6:      B200 = 0;
 7:      W184 = 1;
 8:      F55 = 0;
 9:      F51 = 1;
10:      W186 = 1;
11:      ++B178;           /* number of tests performed */
12:      if (B202 < 3 || B202 > 4)
13:         {
14:         B99 = 7;       /* in0 failed */
15:         B108 = B202;
16:         }
17:      else if (B203 < 3 || B203 > 4)
18:         {
19:         B99 = 8;       /* in1 failed */
20:         B109 = B203;
21:         }
22:      else
23:         ;
24:      B202 = 0;
25:      B203 = 0;
26:      }
27:   }
28:
29:Y00 = B186;
30:Y01 = B187;
31:io_fault:
32:


F051  (test#1 )  test #1  fail    "off"
F055  (tstcomp)   tests   comp   (wait)
F056  (tstcomp)           tests   comp
B099  (fltcode)           fault   code
B100  (flt_i0 )  input0   fault   code
B108  (in0fcnt)   in0     fail    count
B109  (in1fcnt)   in1     fail    count
B178  (#tests )  number  of I/O  tests
B186  (outimge)          outputs  image
B187  (-------)  ------- ------- -------
B200  (point# )   test    point   number
B202  (in0 cnt)  input0  intrpt  count
B203  (in1 cnt)  input1  intrpt  count
W180  (flt ptr)           fault  pointer
W184  (bitmask)  point    bit     mask
W186  (outimge)          outputs  image
Y000  (-------)  ------- ------- -------
Y001  (-------)  ------- ------- -------
```

APPENDIX A
PROGRAMMING EXAMPLE

APPENDIX A
PROGRAMMING EXAMPLE

```
D4110T.L00

*************************************************************************
block:  1 - High-level

 0:++B202;
 1:


B202  (in0 cnt)  input0  intrpt  count
```

```
  D4110T.L01

  ************************************************************************
  block:  1 - High-level

   0:++B203;
   1:


B203  (in1 cnt)  input1  intrpt  count
```

APPENDIX A
PROGRAMMING EXAMPLE

```
D4110T.L09

************************************************************************
block:  1 - High-level

 0:/* fault routine - display fault and call system fault routine */
 1:
 2:if (B99 == 1)           /* USER PORT fault? */
 3:   {
 4:   sfunc04(b405H,"FAULT CODE: 1 - USER PORT xmit/receive  ");
 5:   sfunc04(b42dH,"error. USER PORT error code: 0          ");
 6:   B67 = B76 + 48;      /* embed error code */
 7:   sfunc08(b44bH,B67);
 8:   }
 9:else if (B99 == 2)      /* serial port error */
10:   {
11:   sfunc04(b405H,"FAULT CODE: 2 - serial port 1 or 2 comm ");
12:   sfunc04(b42dH,"error. sfunc13 error return value:   H  ");
13:   B67 = (B136>>4)+48;  /* embed sf13 return value */
14:   sfunc08(b450H,B67);
15:   B67 = (B136&0fH);
16:   if (B67 < 10)
17:      B67 = B67 + 48;
18:   else
19:      B67 = B67 + 55;
20:   sfunc08(b451H,B67);
21:   }
22:else if (B99 >= 3 && B99 <= 6)      /* I/O point fail */
23:   {
24:   if (B99 == 3)
25:      {sfunc04(b405H,"FAULT CODE: 3 - input/output fail \"off\" ");}
26:   else if (B99 == 4)
27:      {sfunc04(b405H,"FAULT CODE: 4 - input/output fail \"on\"  ");}
28:   else if (B99 == 5)
29:      {sfunc04(b405H,"FAULT CODE: 5 - input/outpt pattern fail");}
30:   else if (B99 == 6)
31:      {sfunc04(b405H,"FAULT CODE: 6 - input filter too long     ");}
32:   else
33:      ;
34:   sfunc04(b42dH,"   Last I/O point that failed:          ");
35:   B67 = (B200/10) + 48;
36:   sfunc08(b44cH,B67);
37:   B67 = (B200%10) + 48;
38:   sfunc08(b44dH,B67);
39:   }
40:else if (B99 == 7)
41:   {
42:   sfunc04(b405H,"FAULT CODE: 7 - interrupt input0        ");
43:   sfunc04(b42dH,"failed to change state.                 ");
44:   }
45:else if (B99 == 8)
46:   {
47:   sfunc04(b405H,"FAULT CODE: 8 - interrupt input1        ");
48:   sfunc04(b42dH,"failed to change state.                 ");
49:   }
50:else
```

- block continued on next page -

```
      - block originated on prev page -

  51:   {
  52:   sfunc04(b405H,"    Undefined FAULT detected.           ");
  53:   sfunc04(b42dH,"                                        ");
  54:   }
  55:


B067  (ascikey)   ascii    key   number
B076  (USERerr)   user    port    error
B099  (fltcode)           fault   code
B136  (ser err)  serial   port    error
B200  (point# )   test    point   number
```

APPENDIX A
PROGRAMMING EXAMPLE

```
D4110T.L09

************************************************************************
block:  2 - High-level

 0:sfunc08(b402H,15H);     /* (clear display/cursor home) */
 1:sfunc08(b403H,0eH);     /* cursor invisible */
 2:sfunc08(b404H,12H);     /* data entry mode */
 3:sfunc08(b455H,0dH);     /* carriage return (end of message) */
 4:B88 = 0;
 5:while (sfunc18(84,*B88) < 2)
 6:   ;
 7:
 8:sfunc09();
 9:


B088  (sf18ptr)  sfunc18 address pointer
```

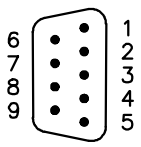
```
D4110T.L10

************************************************************************
block:  1 - High-level

 0:/* receive_frame() user function */
 1:
 2:if (B73 == f0H)          /* expected char == SOF? */
 3:   {
 4:   if (B72 == f0H)       /* received char == SOF? */
 5:      {
 6:      B73 = f8H;         /* expected char = DATA */
 7:      B74 = 8;           /* data byte length = 8 */
 8:      B75 = &B80;        /* point to 1st addr of receive data buffer */
 9:      return;
10:      }
11:   else
12:      {
13:      B76 = 1;           /* error = BAD_SOF */
14:      B79 = B72;         /* debug */
15:      B99 = 1;           /* USER PORT error code */
16:      ufunc09();         /* call fault routine */
17:      }
18:   }
19:if (B73 == f8H)          /* expected char = DATA? */
20:   {
21:   *B75 = B72;           /* save nth received byte in data buffer */
22:   ++B75;
23:   --B74;
24:   if (B74 == 0)         /* all data bytes received? */
25:      B73 = f4H;         /* expected char = EOF */
26:   return;
27:   }
28:if (B73 == f4H)          /* expected char == EOF? */
29:   {
30:   if (B72 == f4H)       /* received char == EOF? */
31:      F10 = 1;           /* xmit response */
32:   else
33:      {
34:      B76 = 2;           /* error = BAD_EOF */
35:      B99 = 1;
36:      ufunc09();         /* call fault routine */
37:      }
38:   B73 = f0H;            /* expected char = SOF */
39:   }
40:


F010  (xmitrsp)   xmit   respons in prog
B072  (rcvchar)          receivd  char
B073  (expchar)          expectd  char
B074  (#rcved )  num of   bytes  receivd
B075  (bufpntr)   data   buffer  pointer
B076  (USERerr)   user    port    error
B079  (rcvchar)   rcved   char   aterror
B080  (rcvbuf1)  receive  data   buffer
B099  (fltcode)           fault   code
```

APPENDIX A
PROGRAMMING EXAMPLE

```
D4110T.L11

********************************************************************************
block:  1 - High-level

 0:/* transmit_response() user function */
 1:
 2:if (F11 == 0)           /* receive buffer not checked? */
 3:   {
 4:   B75 = &B80;          /* check all received data bytes */
 5:   for (B77 = 0; B77 < 8; ++B77)
 6:       {
 7:       if (*B75 != (B77+10))
 8:          {
 9:          B76 = 3;      /* error = BAD_DATA */
10:          B99 = 1;      /* USER PORT error code */
11:          ufunc09();    /* call fault routine */
12:          }
13:       ++B75;
14:       }
15:   F11 = 1;             /* receive buffer checked */
16:
17:   W120 = b202H;         /* set-up xmit buffer */
18:   for (B77 = 0; B77 < 8; ++B77)
19:       {
20:       B75 = B77+30;
21:       sfunc08(W120,B75);
22:       ++W120;
23:       }
24:   }
25:
26:if (B76 == 0)           /* no error detected? */
27:   {
28:   B88 = 0;
29:   B78 = sfunc11(8,*B88);     /* xmit xmit_buf */
30:   if (B78 == 2)        /* xmit done? */
31:      {
32:      F10 = 0;          /* yes, reset xmit response */
33:      F11 = 0;          /* reset receive buffer checked */
34:      }
35:   }
36:


F010  (xmitrsp)   xmit   respons in prog
F011  (buf chk)  receive buffer  checked
B075  (bufpntr)   data   buffer  pointer
B076  (USERerr)   user    port    error
B077  (reg(i) )  general registr  (i)
B078  (sf11ret)          sfunc11 return
B080  (rcvbuf1)  receive  data   buffer
B088  (sf18ptr)  sfunc18 address pointer
B099  (fltcode)           fault   code
W120  (sf11ptr)  sfunc11 buffer  pointer
```

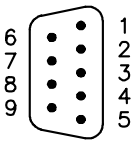
(This Page Intentionally Left Blank)

APPENDIX B
RS-232 PINOUTS/CABLES



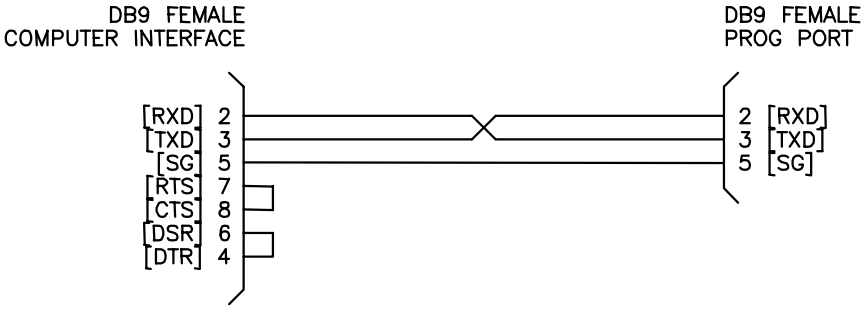
1: NOT USED
2: RXD (RECEIVE DATA – IN)
3: TXD (TRANSMIT DATA –OUT)
4: NOT USED
5: SG (SIGNAL GROUND)
6: NOT USED
7: NOT USED
8: NOT USED
9: NOT USED

PROG Port Pin Out

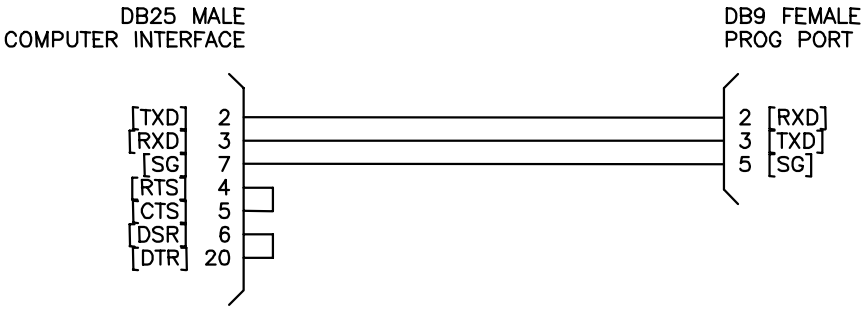


1: NOT USED
2: RS–232 RXD (RECEIVE DATA – IN)
3: RS–232 TXD (TRANSMIT DATA –OUT)
4: NOT USED
5: SG (SIGNAL GROUND)
6: RS–422 TXD+ (TRANSMIT DATA – OUT)
7: RS–422 TXD– (TRANSMIT DATA – OUT)
8: RS–422 RXD+ (RECEIVE DATA – IN)
9: RS–422 RXD– (RECEIVE DATA – IN)

USER Port Pin Out

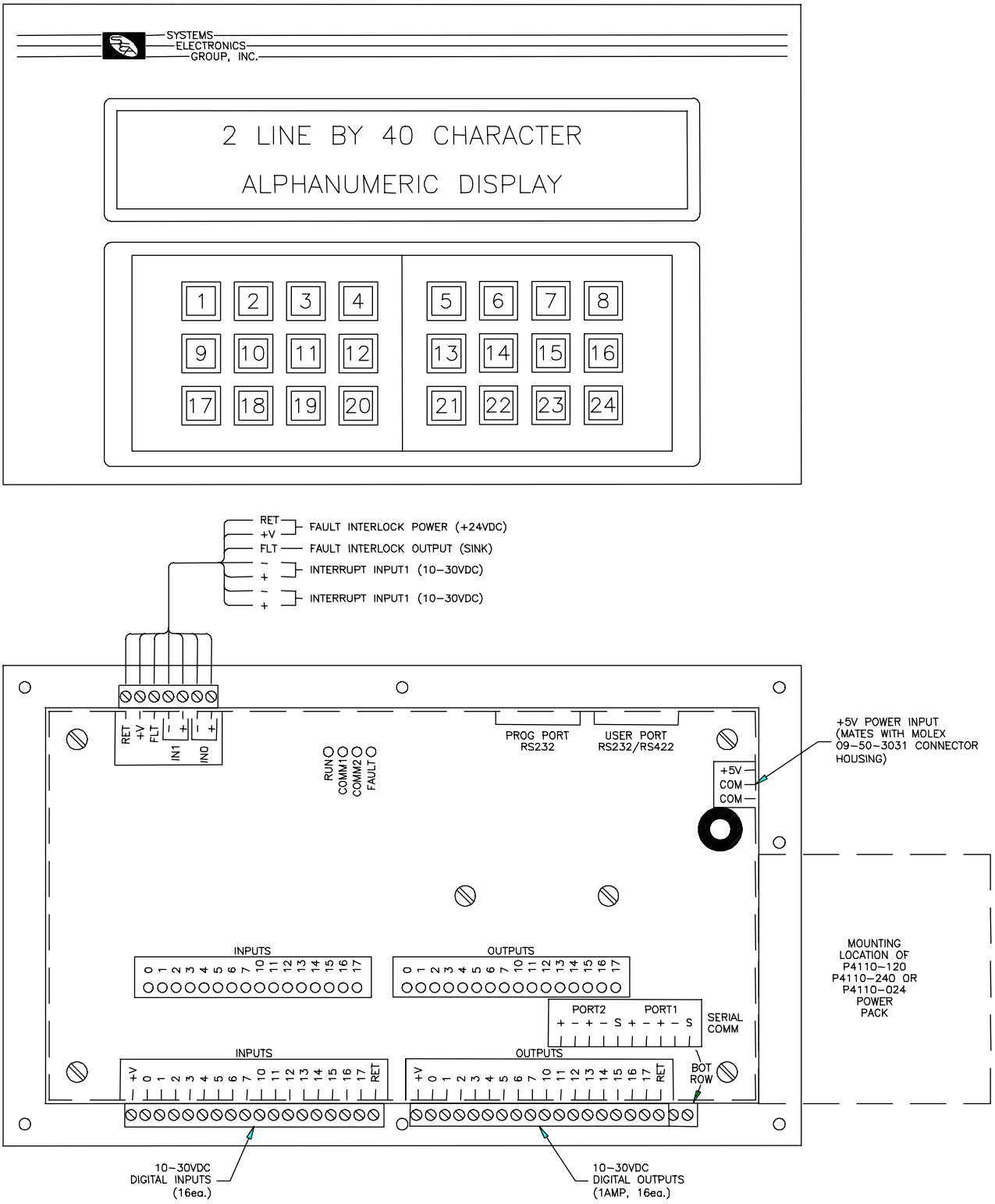


DB9 (com1) to PROG Port Cable



DB25 (com1) to PROG Port Cable

APPENDIX B
RS-232 PINOUTS/CABLES

(This Page Intentionally Left Blank)

APPENDIX C
FIELD WIRING CONNECTOR PINOUTS