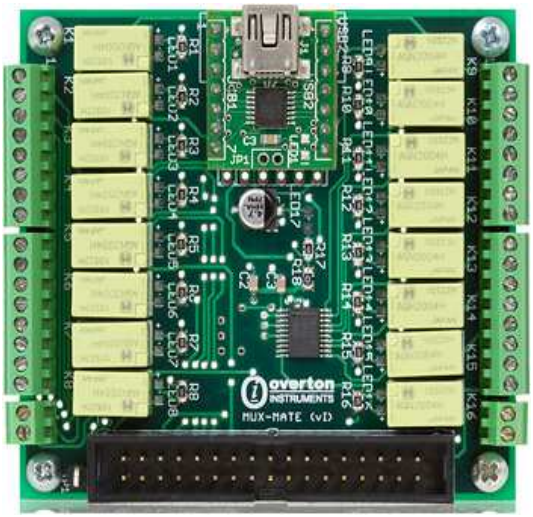
MUX-MATE(vI)

16-CH Signal Multiplexer Module



USER'S MANUAL

DESIGNED FOR TEST
PART OF MG-PRODUCTS

MG-Products B.V.
Rijkevoortsedijk 27A
5447 BD, Rijkevoort (NL)
Phone: +31 (0)485 382 133
www: www.designedfortest.com
E-mail: Info@designedfortest.com

NOTICE

The information contained in this document is subject to change without notice. To the extent allowed by local law, Overton Instruments (OI), shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of OI.

WARNING

The instrument you have purchased and are about to use may NOT be an ISOLATED product. This means that it may be susceptible to common mode voltages that could cause damage to the instrument. SUCH DAMAGE IS NOT COVERED BY THE PRODUCT'S WARRANTY. Please read the following carefully before deploying the product. Contact OI for all questions.

WARRENTY

OI warrants that this instrument will be free from defects in materials and workmanship under normal use and service for a period of 90 days from the date of shipment. OI obligations under this warranty shall not arise until the defective material is shipped freight prepaid to OI. The only responsibility of OI under this warranty is to repair or replace, at it's discretion and on a free of charge basis, the defective material. This warranty does not extend to products that have been repaired or altered by persons other than OI employees, or products that have been subjected to misuse, neglect, improper installation, or accident. OVERTON INSTRUMENTS SHALL HAVE NO LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES OF ANY KIND ARISING OUT OF THE SALE, INSTALLATION, OR USE OF ITS PRODUCTS.

SERVICE POLICY

1. All products returned to OI for service, regardless of warranty status, must be on a freight-prepaid basis.
2. Until otherwise noted, OI will repair or replace any defective product within 10 days of its receipt.
3. For in-warranty repairs, OI will return repaired items to buyer freight prepaid. Out of warranty repairs will be returned with freight prepaid and added to the service invoice.

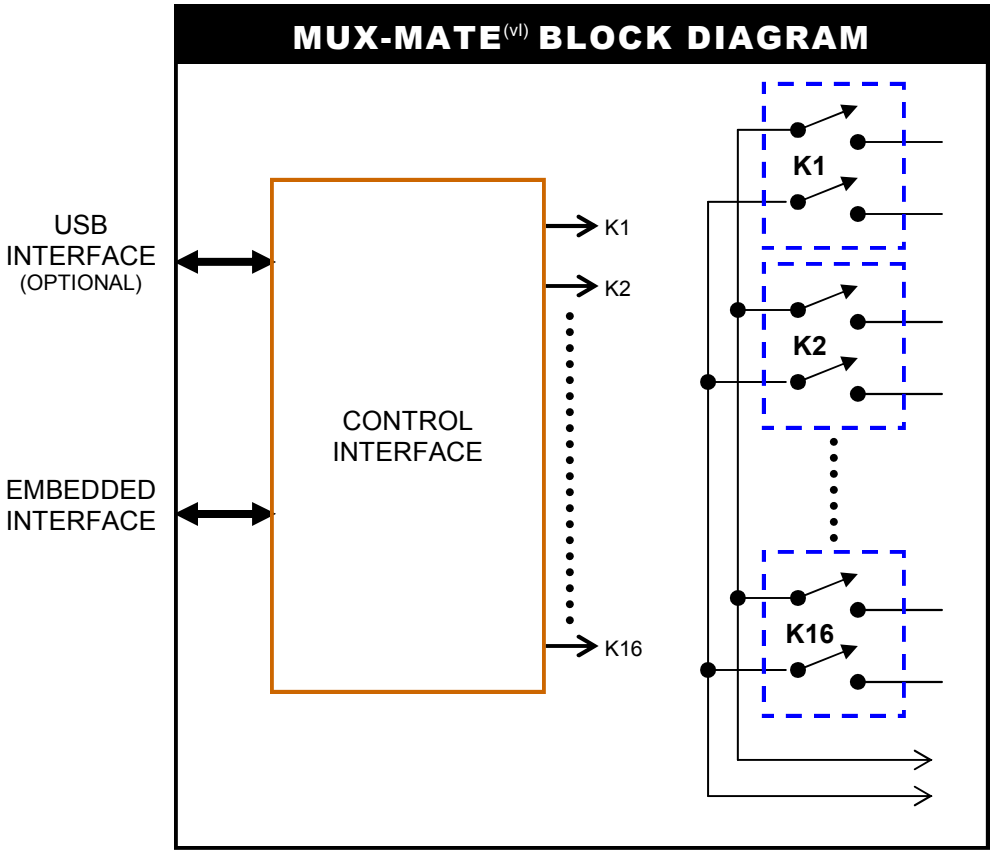
Table Of Contents

1. Introduction

1.1 Overview

The MUX-MATE(vI) is a 16-channel multiplexer that is used to route signals from the DUT (device-under-test), to external measurement equipment (i.e., oscilloscopes, volt meters and frequency counters). Designed for embedded applications, the MUX-MATE(vI) can be installed directly into custom Instrument Enclosures, Mechanical Test Fixtures, or support larger ATE "rack-n-stack" systems. Many test solutions can be quickly built by connecting the MUX-MATE(vI) to a PC laptop or desktop, and then running our GUI software. No external power source is required, since power is supplied through the USB interface. Easy access to the hardware is made available through a convenient collection of screw terminal connectors.

There are two options for controlling the MUX-MATE(vI) (with a Host PC or embedded microcontroller). In the case of the PC, the MUX-MATE(vI) is connected by a USB interface and responds to a simple set of ASCII commands. Programming is easy using Visual BASIC, C/C++, LabView, LabWindows or any language that allows access to through a serial port. For embedded control, the MUX-MATE(vI) uses an OI hardware interface to connect to most popular microcontrollers (i.e., ARM, PIC, AVR or STAMP).



1.2 Highlights

BENEFITS	APPLICATIONS	FEATURES
• Easy to use, simple to control and cost effective • Supports a wide-array of test & measurement applications • Great for embedded solutions - place inside mechanical test fixtures, instrument boxes or rack-mount enclosures	• Burn-In • Engineering • Depot Repair • Production Test • QA/QC Quality Control • OEM Test Instruments	• 16 X 2 X 1 Mux (16 relays, 2-poles and 1 common output) • LED indicate each active relay • Rapid switching, < 4msec activation time • USB interface or embedded control • Convenient screw terminals and pin headers for user connections • Low Cost • Compact size, a 2.50 x 2.75" PCB, with four #4 mounting holes in each corner (spacers and hardware included)

1.3 Specifications

Relays	
Channels	16
Type	DPDT (2 form C)
Max DC Volts	125Vac, 110Vdc
Max Current	1A
Max Load	0.50A/125Vac, 1A/30Vdc
Min Load	10uA, 10mVdc
Max Switching	62.50VA, 33W
Contact Resistance	100mΩ
Relay life	100M operations
Set / Release Time	Max 4msec (approx 2msec)
Input Control	
Embedded	Oi-Bus interface
USB Interface	Optional USB module
General	
Power Supply	+5VDC±10%@100mA
Operating Temp	0-50ºC
Dimensions	2.50" x 2.75"

2. I/O Description

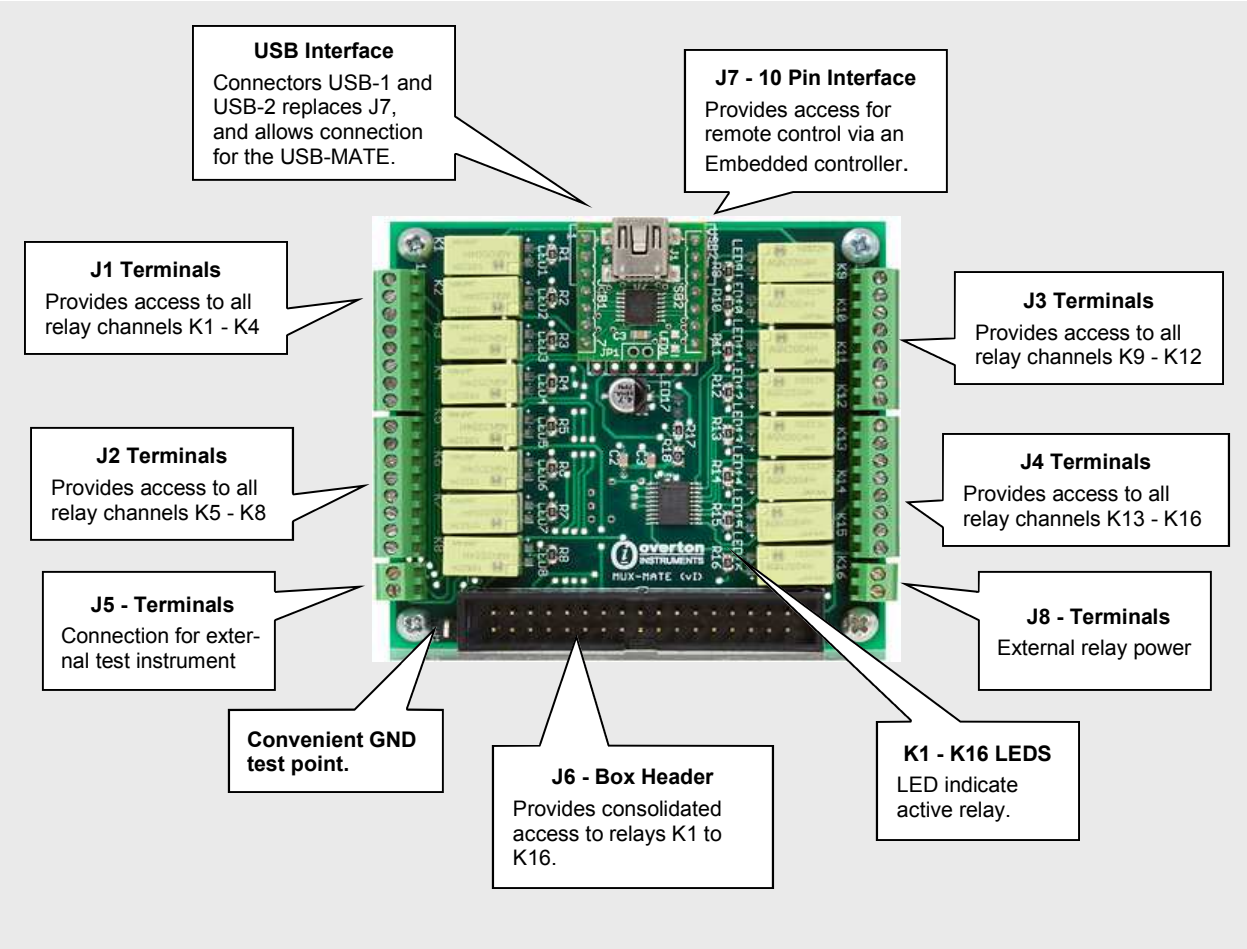
2.1 Hardware Details

Access to MUX-MATE(v1) hardware is made possible through a convenient set of screw terminal connections (J1 - J5), and J6 (which consolidates all signals into a single 34-pin header).

The external connection is made on the 'common' side of the relays.. On the normally-open side, the relays are all bussed together and connected to J5. Activating anyone of the relays will connect its inputs to J5.

External control of the MUX-MATE(v1) can be provided by a embedded controller (such as the Micro-MATE), or with a PC. Embedded control is supported by J7 (Oi-BUS interface), which is a 10-pin header that includes a 3-wire SPI-bus, chip select logic, power and ground. In PC applications, connector J7 is replaced with the USB--MATE. The USB-MATE contains a USB connector (for the PC), and a dual set of 7-pin headers that mount to the MUX-MATE(v1). The USB-MATE is designed to interpret a set of ASCII commands sent from the PC, and then perform various MUX-MATE(v1) functions. For more information, the MUX-MATE(v1) command set is provided in Appendix A. To support embedded applications, a complete driver for the MUX-MATE(v1) is provided in TES-MATE (or Test Executive Suite).

2.2 Board Layout



2.3 Connections

J1, K1 - K4		
Relay	**Pin**	**Name**
K1	J1-1	CH1-HI
K1	J1-2	CH1-LO
K2	J1-3	CH2-HI
K2	J1-4	CH2-LO
K3	J1-5	CH3-HI
K3	J1-6	CH3-LO
K4	J1-7	CH4-HI
K4	J1-8	CH4-LO

J2, K5 - K8		
Relay	**Pin**	**Name**
K5	J2-1	CH5-HI
K5	J2-2	CH5-LO
K6	J2-3	CH6-HI
K6	J2-4	CH6-LO
K7	J2-5	CH7-HI
K7	J2-6	CH7-LO
K8	J2-7	CH8-HI
K8	J2-8	CH8-LO

J3, K9 - K12		
Relay	**Pin**	**Name**
K9	J3-1	CH9-HI
K9	J3-2	CH9-LO
K10	J3-3	CH10-HI
K10	J3-4	CH10-LO
K11	J3-5	CH11-HI
K11	J3-6	CH11-LO
K12	J3-7	CH12-HI
K12	J3-8	CH12-LO

J4, K13 - K16		
Relay	**Pin**	**Name**
K13	J4-1	CH13-HI
K13	J4-2	CH13-LO
K14	J4-3	CH14-HI
K14	J4-4	CH14-LO
K15	J4-5	CH15-HI
K15	J4-6	CH15-LO
K16	J4-7	CH16-HI
K16	J4-8	CH16-LO

J7		
Pin	**Name**	**Description**
1	+5V	External Relay Pwr
2	GND	Ground

J5		
Pin	**Name**	**Description**
1	Mux I/O	Mux I/O Channel
2	GND	Ground

2.3 Connections, cont

J7			
Pin	**Name**	**Dir.**	**Description**
1	VCC		A regulated +5Vdc input . Current should be limited to roughly 100mA.
2	SCLK	I	Part of a 3-wire SPI-Bus, SCLK synchronizes the serial data transfer.
3	SET\	I	An TTL active-low "input' signal that causes relays K1-K16 to close.
4	DIN	I	Part of a 3-wire SPI-Bus, DIN is serial command and control data.
5			
6			
7			
8	DEV_CS\	I	An TTL active-low "input' signal that provides a chip-select for the relay driver circuit..
9	DGND		Digital Ground
10	RESET\	I	An TTL active-low "input' signal that causes relays K1-K16 to open.

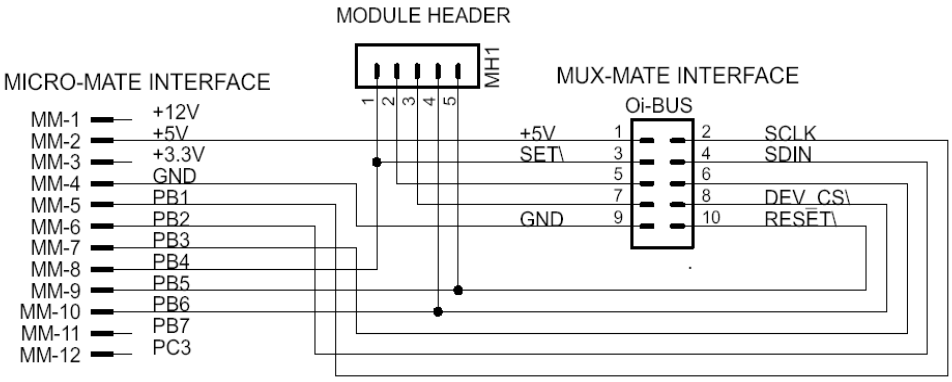
J6			
Pin	**Name**	**Pin**	**Name**
1	CH1-HI	26	CH13-HI
2	CH1-LO	27	CH13-LO
3	CH2-HI	28	CH14-HI
4	CH2-LO	29	CH14-LO
5	CH3-HI	30	CH15-HI
6	CH3-LO	31	CH15-LO
7	CH4-HI	32	CH16-HI
8	CH4-LO	33	CH16-LO
9	CH5-HI	34	CH17-HI
10	CH5-LO	35	CH17-LO
11	CH6-HI	36	CH18-HI
12	CH6-LO	37	CH18-LO
13	CH7-HI	38	CH19-HI
14	CH7-LO	39	CH19-LO
15	CH8-HI	40	CH20-HI
16	CH8-LO	41	CH20-LO
17	CH9-HI	42	CH21-HI
18	CH9-LO	43	CH21-LO
19	CH10-HI	44	CH22-HI
20	CH10-LO	45	CH22-LO
21	CH11-HI	46	CH23-HI
22	CH11-LO	47	CH23-LO
23	CH12-HI	48	CH24-HI
24	CH12-LO	49	CH24-LO
25	CH0-HI	50	CH0-LO

3. Operation

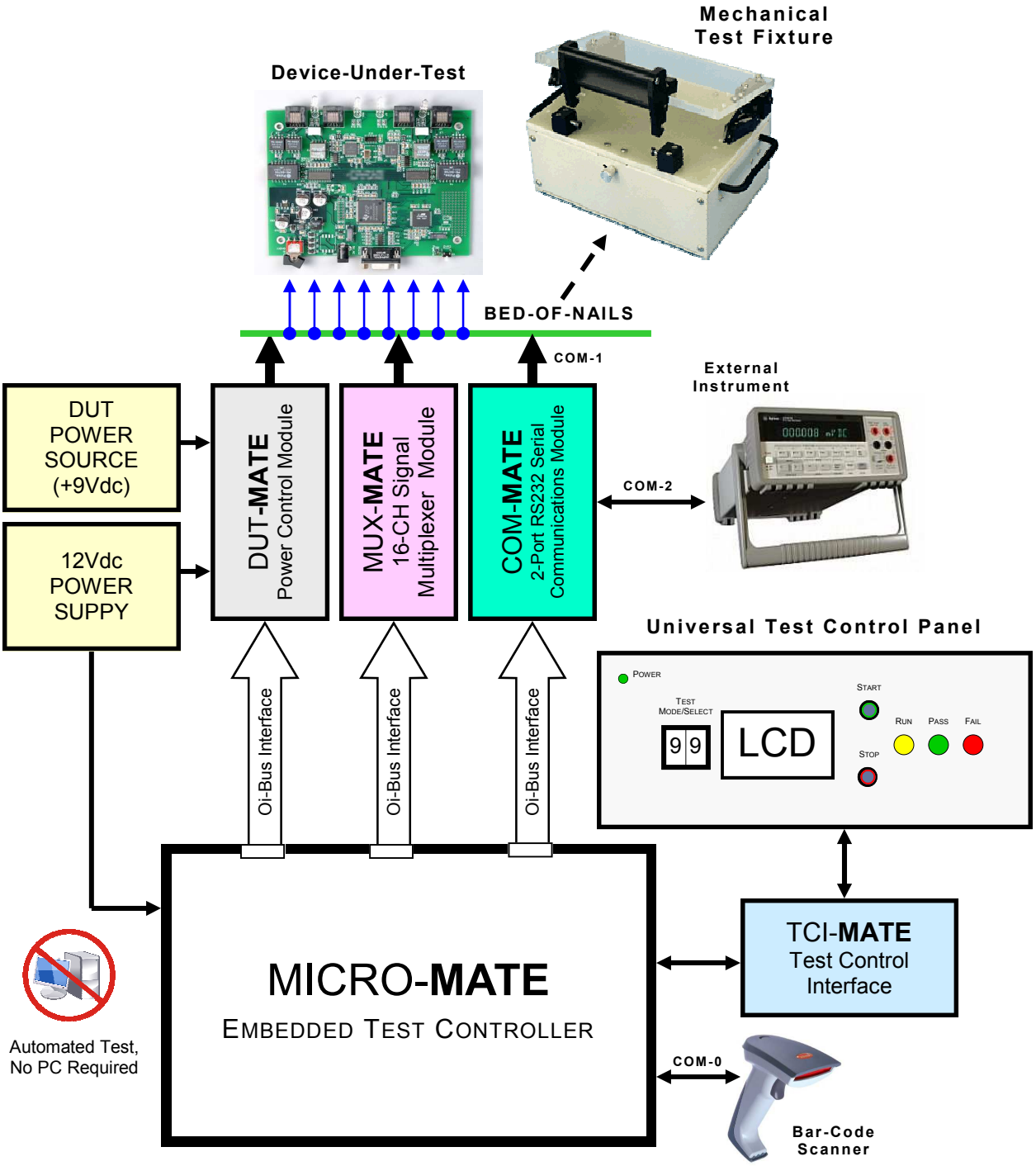
3.1 Embedded Control

In section 3.1.1 (on the next page), the MUX-MATE(v1) is shown integrated with other ETS Series components that collectively form a complete Embedded Test Solution. The diagram shows the MUX-MATE(v1) being driven by the Micro-MATE™. The Micro-MATE™ is a low-cost "Embedded Test Controller", which stores a special program that is designed to exercise the device-under-test and generate Go/No-Go test results. The Micro-MATE™ also provides a sizable breadboard area to support the development of custom circuits. Adjacent to the breadboard area is a series of wire-wrap pins (called the MM Interface), that comprises a goodly amount of general purpose Digital I/O. The schematic below shows the wire-wrap connections which create the interface between the Micro-MATE™ and the MUX-MATE(v1) (the Oi-BUS, 10-pin header connector).

Actually the MUX-MATE(v1) can be easily driven by most microcontrollers (including an ARM, AVR, PIC or even a STAMP). When developing a custom interface for the MUX-MATE(v1), it is recommended the designer start-by reviewing the interface requirements as outlined in the J6 Table (which is provided in the I/O Connections section). The next step is to review the MUX-MATE(v1) schematic, which is provided in Appendix B. What could be the most challenging aspect of the design effort is controlling the SPI-bus device. The MUX-MATE(v1) uses a relay driver chip from Maxim (part number MAX4820). Details for specific performance and SPI-bus operation can be found in the device data sheet. Go to the manufacturers website to download said documents.



3.1.1 Embedded Configuration



3.1.2 Embedded Programming

To build-on the PCB board test example (shown in section 3.1.1), we have constructed a demo program using BASIC. The demo program (which is outlined in section 3.2.3), illustrates the ease of controlling the MUX-MATE(v1) via the Micro-MATE™, Embedded Test Controller.

The program starts by initialing the Micro-MATE™ for proper operation. You will note that the program provides excellent bit-manipulation capabilities, as evident by the use of the ALIAS statement. The Micro-MATE™ (PB1 & PB2 port bits) are assigned unique label names (i.e., SCLK, DOUT), which are used to support various MUX-MATE(v1) functions. In the "Main" program section, the Micro-MATE™ receives "high level" serial commands from a host PC, parses them and then executes accordingly. When (for example), the "MX_SR051" command is entered, the "Mx_rly_sel(A_ch, A_num)" subroutine is called. This causes the MUX-MATE(v1) to activate relay #5.

Independent of the microcontroller hardware or programming language you choose, the program sequence described above will likely resemble the way you implement your MUX-MATE(v1) application. For this reason, we suggest that you go to our website and download the "MUX-MATE(v1).zip" file. In the Documents folder will contain more extensive examples of routines to control the MUX-MATE (v1) .

3.1.3 Embedded Program Example

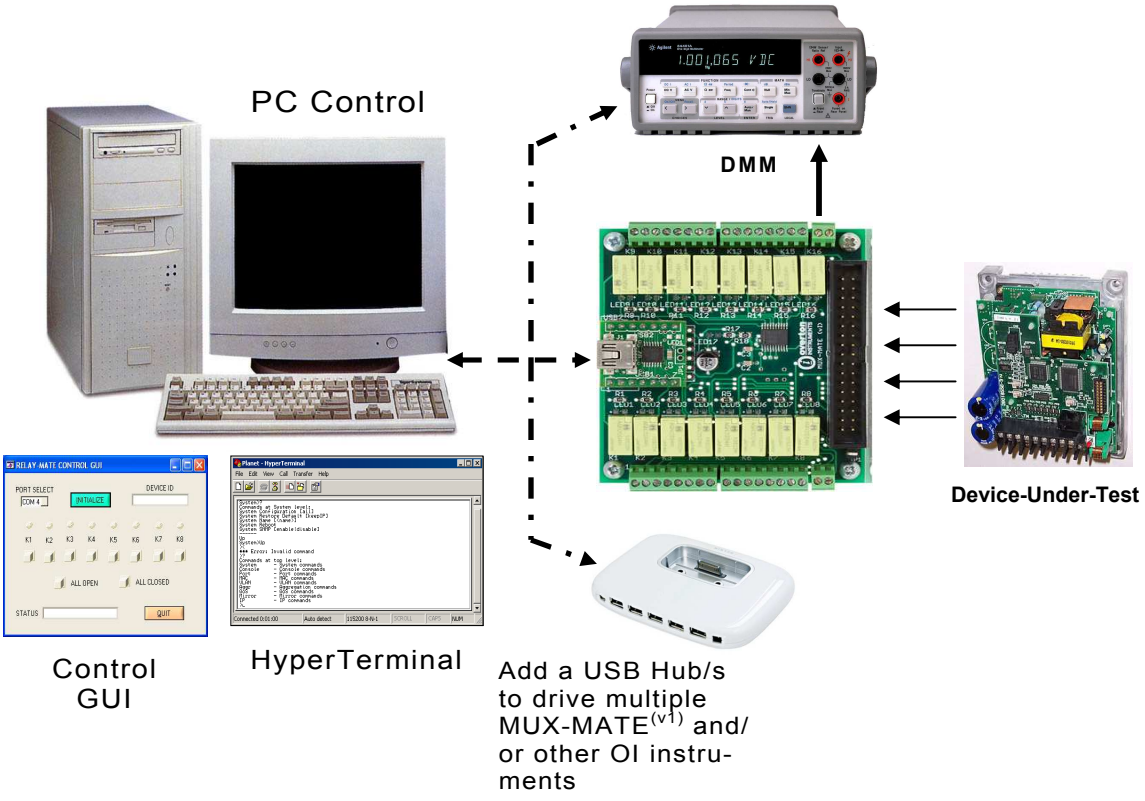
```
' Program: MUX-MATE(v1) Demo
'
---[ Initialization ]----------------------------------------------------------
'
$large
$romstart = &H0000
$default Xram

Dim Mx_bit As Bit
Dim A_num, A_byte, A_cnt As Byte
Dim Mx_byte, Mx_cnt, Mx_settle, Mx_status, Mx_num as Byte
Dim S As String * 10, A_resp AS String * 10, A_str AS String * 10
Dim Mx_str As String * 1, Sf_str AS String * 10
Dim A_word as Word
Dim A_val as Single
Dim True As Const 1
Dim False As Const 0

Sclk Alias PB1                      ' SPI-bus serial clock
Dout Alias PB2                      ' SPI-bus serial data output
Mx_cs Alias PB4                     ' Relay driver chip select
Mx_rst Alias PB5                    ' Reset relay driver chip
Mx_set Alias PB6                    ' Set relay driver chip

Declare Sub Print_ic                ' print invalid command
Declare Sub Print_orr               ' print out-of-range
Declare Sub Print_ur                ' print under range
Declare Sub Print_ok                ' print command is OK
Declare Sub Mx_rly_sel(Mx_num As Byte, Mx_bit as Bit) ' select specific relay

---[ Main ]---------------------------------------------------------
' In the Main the Operator or Host, is prompted to enter a command.  The com-
' mand is parsed and then executed if valid.

Set Sclk, Dout, Mx_cs, Mx_rst, Mx_set        ' Set to logic '1'
Mx_settle = 10                               ' 10msec relay settle
Do
  Input "-> " , S
  S = Ucase(s)
  A_resp = Left(s , 3)
  If A_resp = "MX_" Then
    A_resp = Mid(s , 4 , 2)
    Select Case A_resp

    Case "SR":                      ' select relay 1 to 8

      A_char = Mid(s , 6 , 1)
      If A_char = "?" Then
        A_str = "00000000"
        A_char = "1"
        For A_cnt = 1 To 8
            If Mx_status.A_cnt = Then Mid(A_str, A_cnt, 1) = A_char
        Next A_cnt
        Print "<"; A_str; ">"       ' print relay status
      Else
        Err_trap = False
        A_char = Mid(s, 6, 1)
        If A_char = "0" Then Err_trap = True
        A_char = Mid(s, 7, 1)
        A_ch = Val(A_char)
        If A_ch > 8 Then Err_trap = True
        A_char = Mid(s, 8, 1)
        A_num = Val(A_char)
        If A_num > 1 Then Err_trap = True
        If A_char <> "0" And A_char <> "1" Then Err_trap = True
        If Err_trap = False Then
          Call Mx_rly_sel(A_ch, A_num)   ' turn relay on/off
          Call Print_ok
        Else
          Call Print_oor            ' print out-of-range
        End If
      End If

    Case Else
      Call Print_ic                 ' invalid command
    End Select
  Else
    Call Print_ic                   ' invalid command
  End If
Loop
End

'---[ Sub-Routines]-------------------------------------------------------------
'
Sub Print_ic                        ' print invalid command
  Print "><"
End Sub

Sub Print_oor                       ' print out-of-range
  Print ">>"
End Sub

Sub Print_ur                        ' print under range
  Print "<<"
End Sub

Sub Print_ok                        ' print command is OK
  Print "<>"
End Sub

    ' Select specific relay
Sub Mx_rly_sel(Mx_num As Byte , Mx_bit As Bit)
  Mx_num = Mx_num - 1
  Mx_status.Mx_num = Mx_bit
  Reset Mx_cs
  For Mx_cnt = 7 Downto 0
    Dout = Mx_status.Mx_cnt         ' serial data out
    Set Sclk
    Reset Sclk
  Next Mx_cnt
  Set Mx_cs
  Waitms Mx_settle                  ' settling time
End Sub
```

3.2 PC Control

For those more comfortable building traditional PC-based "Automated Test Equipment" (ATE), the MUX-MATE(v1) offers many features that are well suited for that environment as well.

Controlling the MUX-MATE(v1) from a PC, requires that it be equipped with an optional USB-MATE module. The USB-MATE module contains a USB bridge-chip and a PIC microcontroller. On the PC side, the USB bridge-chip receives a special set of serial commands. On the MUX-MATE(v1) side, the PIC controller processes the serial commands and then drives the MUX-MATE(v1) accordingly. In order to be recognized by the PC, the USB-MATE module requires a set of Windows' drivers be installed. To do so, go to "www.MUX-MATE(v1).com", click "Download", select the "OI VCP Interface" file and follow the prompts. The letters VCP stands for "Virtual COM Port", and is a method by-which the USB interface can appear to the PC as a standard serial COM port. With the drivers installed and the USB-MATE connected to the PC, go to the Device Manager (click on Ports) and verify "OI Serial Interface (COM#)" is included.

The diagram below provides a basic illustration for a PC-driven configuration. As shown, the MUX-MATE(v1) can be used to route various signals from the DUT (device-under-test), to an external instrument.

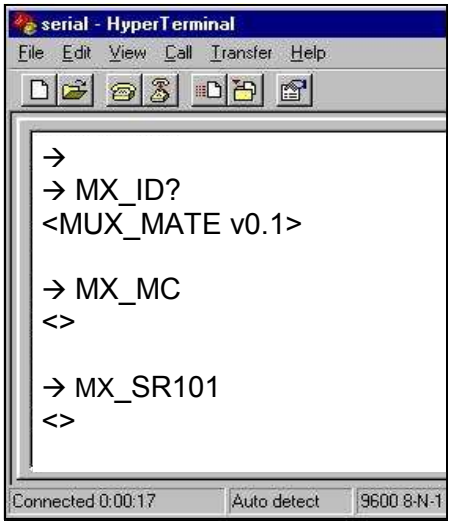


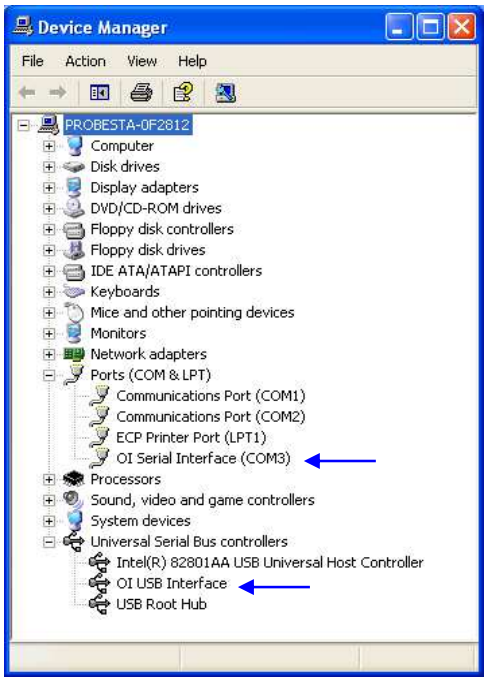
3.3.2 PC Programming

The starting point for developing code to control the MUX-MATE(v1), begins with acquainting yourself with its Serial Command Set. The serial commands are a set (or group) of ASCII characters that originate from the PC and are designed to instruct the MUX-MATE(v1) to perform specific functions. The complete serial command set is detailed in Appendix B. There are two ways to exercise the serial commands, (1) using HyperTerminal or (2), run our Virtual Instrument Panel software (GUI Control).

3.2.1.1 HyperTerminal

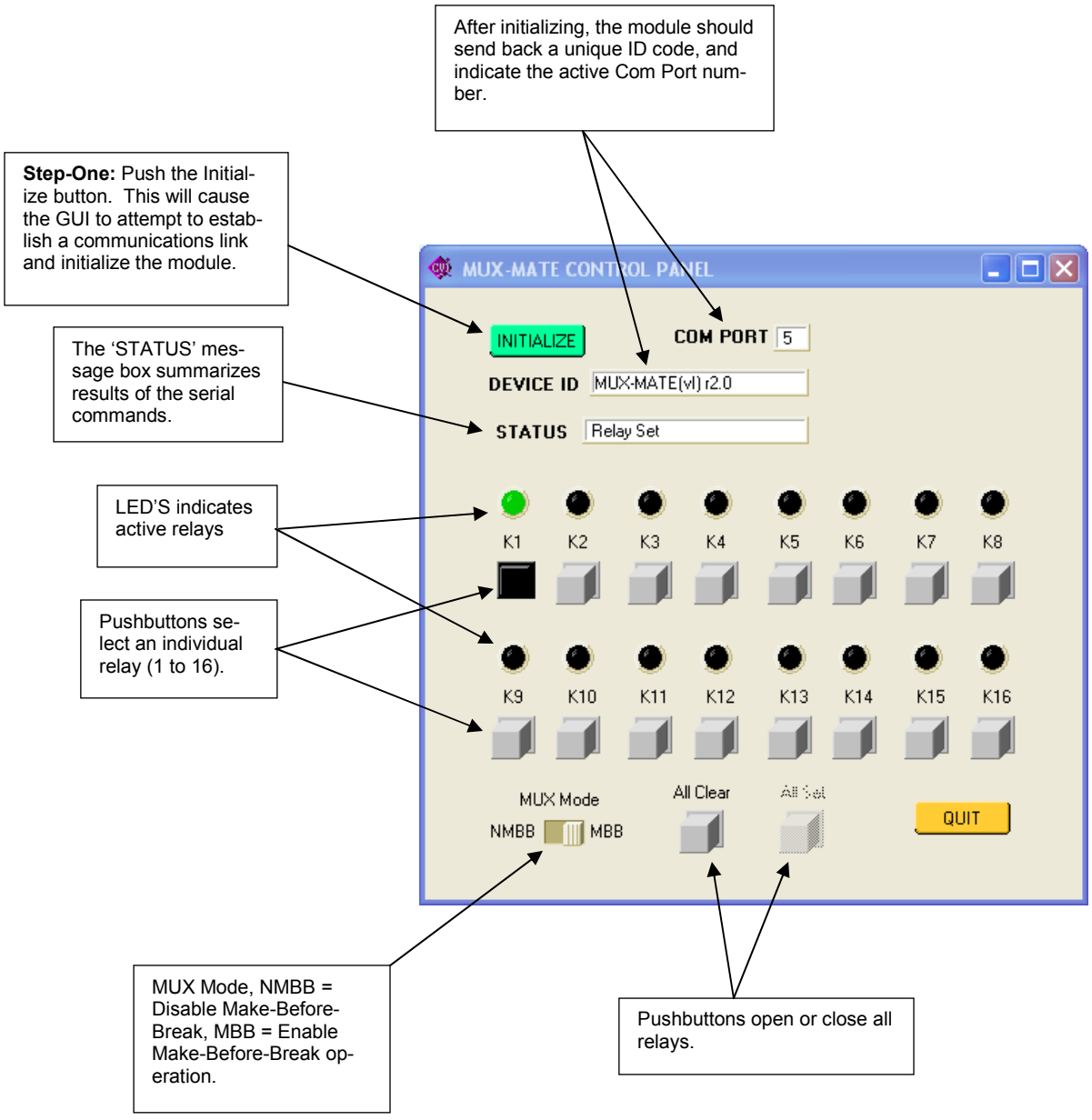
HyperTerminal is a serial communications program that comes with the Windows OS and is located in the Accessories folder. Use the USB cable to connect the PC to the MUX-MATE(v1). Run HyperTerminal and configure the settings for 19200 bps, 8 data bits, no parity, 1 stop bit and no flow control. Select the COM port based on the available COM port as indicated in the Device Manager (example shown below). Press the 'Enter' key and the '→' prompt should appear on the screen (as demonstrated in the example on the right). Refer to the table in Appendix A, to begin to experiment with the serial commands.





3.2.1.2 Virtual Instrument Panel

The Virtual Instrument Panel (or Control GUI), removes the hassle of "manually " typing ASCII commands and provides the User a more efficient method to interact and control the MUX-MATE(v1). Download the panel from our website at www.MUX-MATE(v1).com, click on downloads and select "MUX-MATE(v1)xxx.exe".



3.2.1.3 PC Programming Example

```
//  MUX-MATE(v1) programming example in 'C'
//
//  The following program provides a Go/No Go test sequence for testing
//  a printed circuit board (the DUT).  The test equipment includes a MUX-
//  MATE(v1), DUT-MATE(v1) and a 34401A DMM (equipped with a RS-232
//  remote interface).  The DUT accepts a +24Vdc input and (in-turn) gener-
//  ates +5Vdc (logic) and ±12Vdc (analog) voltages.
//
//  The DUT-MATE(v1) is used to switch power to the device-under-test.  After
//  power is applied, the MUX-MATE(v1) is used to route various signals to
//  the DMM for measurement.  After confirming the DUT input/output
//  voltages are within spec., the program also checks other 'key' test
//  points.
//

#define     MSWIN                   //  serial comm libraries from
#define     MSWINDLL                //  www.wcscnet.com

#include <comm.h>
#include <stdlib.h>
#include <stddio.h>

int stat, port=0, a_byte = 0, a_cnt = 0, int idx = 0;
int dut_ch = 0, dut_gain =0, gain_sel = 0;
int dio_bit[10] = 0;

long value = 0, limit = 0;

char dio_byte[10], dir_byte[10], results[64];
char send_data[64], read_data[64];

char clear_relays[] = "MX_CR"         // clear all channel relays
char select_relay[] = "MX_SR"         // select specific relay
char sf_master_clear[] = "MX_MC"      // master clear
char sf_get_device_id[] = "MX_ID?"    // get device ID

char auto_sequence[] = "DT_AS";       // auto DUT power sequence
char set_dut_power[] = "DT_DP";       // set dut power On/Off
char dt_get_device_id[] = "DT_ID?";   // get device ID
char set_breaker_limit[] = "DT_SO";   // set over current breaker limit
char dt_master_clear[] = "DT_MC";     // master clear

main()
{                           // initialize COMM ports
    mx_port = OpenComPort(1,256,64);        // Open COM 1, MUX-MATE
(v1)
    dt_port = OpenComPort(2,256,64);        // Open COM 2, DUT-MATE
    dmm_port = OpenComPort(3,256,64);   // Open COM 3, DMM 34401A

    SetPortCharacteristics(sf_port,BAUD19200,PAR_EVEN,
        LENGTH_8,STOPBIT_1,PROT_NONNON);
    CdrvSetTimerResolution(sf_port,1);          // 1 msec ticks
    SetTimeout(mx_port,2000);       // 2000 ticks = 2 sec time-out period
    FlushReceiveBuffer(sf_port);    // clear receiver buffer
    FlushTransmitBuffer(sf_port);   // clear transmit buffer

    SetPortCharacteristics(dt_port,BAUD19200,PAR_EVEN,
        LENGTH_8,STOPBIT_1,PROT_NONNON);
    CdrvSetTimerResolution(dt_port,1);          // 1 msec ticks
    SetTimeout(dt_port,2000);       // 2000 ticks = 2 sec time-out period
    FlushReceiveBuffer(dt_port);    // clear receiver buffer
    FlushTransmitBuffer(dt_port);   // clear transmit buffer

    SetPortCharacteristics(dmm_port,BAUD19200,PAR_EVEN,
        LENGTH_8,STOPBIT_1,PROT_NONNON);
    CdrvSetTimerResolution(dmm_port,1);     // 1 msec ticks
    SetTimeout(dmm_port,2000);  // 2000 ticks = 2 sec time-out period
    FlushReceiveBuffer(dmm_port);           // clear receiver buffer
    FlushTransmitBuffer(dmm_port);          // clear transmit buffer

for (a_cnt = 1; a_cnt <= 3; a_cnt++) {
    if (a_cnt == 1) || (a_cnt == 2) {
        if (a_cnt == 1) port = sf_port;         // MUX-MATE(v1) com port
        if (a_cnt == 2) port = dt_port;         // DUT-MATE com port

                        // Get device prompt
        sprintf (send_data, "%s\r", "");
        PutString(port,send_data);              // send CR
        if ((resp_len = GetString(port,sizeof(read_data),read_data)) == 0); {
            printf ("time-out error");
            exit();
        }
        if (strcmp("-> ", read_data)) {
            printf ("prompt error");
            exit();
        }               // Get device ID

        if (a_cnt == 1) sprintf (send_data, "%s\r", mx_get_device_id);
        if (a_cnt == 2) sprintf (send_data, "%s\r", df_get_device_id);
        PutString(port,send_data);
        if ((resp_len = GetString(port,sizeof(read_data),read_data)) == 0); {
            printf ("time-out error");
            exit();
        }
        if (a_cnt == 1) sprintf(a_str, %s, "<MUX-MATE(v1) v0.1>");
       if (a_cnt == 2) sprintf(a_str, %s, "<DUT-MATE01 v0.1>");
       if (strcmp(a_str, read_data)) {
           printf ("device ID error");
           exit();
        }               // Master Clear
        if (a_cnt == 1) sprintf (send_data, "%s\r", mx_master_clear);
        if (a_cnt == 2) sprintf (send_data, "%s\r", dt_master_clear);
        PutString(port,send_data);
    }
    else {              // Get 34401A ID
        sprintf (send_data, "%s\r", "*IDN?");
        PutString(dmm_port,send_data);
        if ((resp_len = GetString(dmm_port,sizeof(read_data),
            read_data)) == 0); {
            printf ("time-out error");
            exit();
        }
        sprintf(a_str, %s, "HEWLETT-PACKARD,34401A,0,11-5-2");
        if (strcmp(a_str, read_data)) {
            printf ("34401A ID error");
            exit();
        }
    }
}                       // Execute test sequence
test_fail = False;
for (a_cnt = 1; a_cnt <= 9; a_cnt++) {
    switch (a_cnt) {
        case 1:         // DUT input power Test
            sprintf (send_data, "%s%s\r", set_breaker_limit, "2048");
            PutString(dt_port,send_data);       // send DT_OS2048
            sprintf (send_data, "%s%s\r", auto_sequence, "011");
            PutString(dt_port,send_data);       // send DT_AS011
            GetString(dt_port,sizeof(read_data),read_data);
            sprintf(a_str, %s%s, "DUT Input Power Test failure - ", read_data);
            if (strcmp(">0<", read_data)==0) {
                printf (a_str, read_data);          // short detected
                test_fail = True;
                break;
            }
            else {      // over-current breaker
                if (strcmp(">1<", read_data)==0) {
                    printf (a_str, read_data);
                    test_fail = True;
                  break;
                }
            }
            break;
```

3.2.1.3 PC Programming Example cont.

```
case 2:         // DUT Input Power Test - 24Vdc
    sprintf (send_data, "%s%s\r", select_relay,"011");
    PutString(mx_port,send_data);    // select relay channel 1

    sprintf (send_data, "%s\r", "MEAS:VOLT:DC:RANG:AUTO?");
    PutString(dmm_port,send_data);  // get reading
    GetString(dt_port,sizeof(read_data),read_data);
    low_limit = 23.0;
    high_limit = 25.0;
    value = atoi(read_data);
    if (value < low_limit) || (value > high_limit) {
        printf ("DUT Input Power Test failed - %d\n", value);
        test_fail = True;
    }
    Break;
case 3:         // DUT +5Vdc Logic Power Test
    sprintf (send_data, "%s%s\r", select_relay,"021");
    PutString(mx_port,send_data);    // select relay channel 2

    sprintf (send_data, "%s\r", "MEAS:VOLT:DC:RANG:AUTO?");
    PutString(dmm_port,send_data); // get reading
    GetString(dt_port,sizeof(read_data),read_data);
    low_limit = 4.75;
    high_limit = 5.25;
    value = atoi(read_data);
    if (value < low_limit) || (value > high_limit) {
        printf ("DUT +5Vdc Logic Power Test failed - %d\n", value);
        test_fail = True;
    }
    break;
case 4:         // DUT +12Vdc Analog Power Test
    sprintf (send_data, "%s%s\r", select_relay,"031");
    PutString(mx_port,send_data);     // select relay channel 3

    sprintf (send_data, "%s\r", "MEAS:VOLT:DC:RANG:AUTO?");
    PutString(dmm_port,send_data);  // get reading
    GetString(dt_port,sizeof(read_data),read_data);
    low_limit = 11.75;
    high_limit = 12.25;
    value = atoi(read_data);
    if (value < low_limit) || (value > high_limit) {
        printf ("DUT +12Vdc Analog Power Test failed - %d\n",
                value);
        test_fail = True;
    }
    break;
case 5:         // DUT -12Vdc Analog Power Test
    sprintf (send_data, "%s%s\r", select_relay,"041");
    PutString(mx_port,send_data);     // select relay channel 4

   sprintf (send_data, "%s\r", "MEAS:VOLT:DC:RANG:AUTO?");
   PutString(dmm_port,send_data);   // get reading
   GetString(dt_port,sizeof(read_data),read_data);
   low_limit = -12.25;
   high_limit = -11.75;
   value = atoi(read_data);
   if (value < low_limit) || (value > high_limit) {
       printf ("DUT -12Vdc Analog Power Test failed - %d\n", value);
       test_fail = True;
   }
   break;
        case 6:         // DUT 32Vac Power Test
            sprintf (send_data, "%s%s\r", select_relay,"051");
            PutString(mx_port,send_data);     // select relay channel 5
            sprintf (send_data, "%s\r", "MEAS:VOLT:AC:RANG:AUTO?");
            PutString(dmm_port,send_data);  // get reading
            GetString(dt_port,sizeof(read_data),read_data);
            low_limit = 22.0;
            high_limit = 44.0;
            value = atoi(read_data);
            if (value < low_limit) || (value > high_limit) {
                printf ("DUT 32Vac Power Test failed - %d\n", value);
                test_fail = True;
            }
           break;
        case 7:         // DUT RTC Oscillator Test - 32.768 kHz
            sprintf (send_data, "%s%s\r", select_relay,"061");
            PutString(mx_port,send_data);     // select relay channel 6
            sprintf (send_data, "%s\r", "MEAS:FREQ?");
            PutString(dmm_port,send_data);  // get reading
            GetString(dt_port,sizeof(read_data),read_data);
            low_limit = 32767;
            high_limit = 32769;
            value = atoi(read_data);
            if (value < low_limit) || (value > high_limit) {
                printf ("DUT RTC Oscillator Test failed - %d\n", value);
                test_fail = True;
            }
            break;
        case 8:         // DUT Heater Element Test - 13.5 ohms
            sprintf (send_data, "%s%s\r", select_relay,"071");
            PutString(mx_port,send_data);     // select relay channel 7
            sprintf (send_data, "%s\r", "MEAS:RES:RANG:AUTO?");
            PutString(dmm_port,send_data); // get reading
            GetString(dt_port,sizeof(read_data),read_data);
            low_limit = 12.5;
            high_limit = 14.5;
            value = atoi(read_data);
            if (value < low_limit) || (value > high_limit) {
                printf ("DUT Heater Element Test failed - %d\n", value);
                test_fail = True;
            }
            break;
        case 9:         // DUT 0-Ohm Jumper Test
            sprintf (send_data, "%s%s\r", select_relay,"081");
            PutString(mx_port,send_data);     // select relay channel 8
            sprintf (send_data, "%s\r", "MEAS:CONT?");
            PutString(dmm_port,send_data);  // get reading
            GetString(dt_port,sizeof(read_data),read_data);
            low_limit = 0.0;
            high_limit = 0.0;
            value = atoi(read_data);
            if (value < low_limit) || (value > high_limit) {
                printf ("DUT 0-Ohm Jumper Test failed - %d\n", value);
                test_fail = True;
            }
            break;
        default:
            break;
    }
    if test_fail = True {               // turn-OFF DUT power & exit
        sprintf (send_data, "%s%s\r", set_dut_power, "0");
        PutString(dt_port,send_data);           // send DT_DP0
        exit();
    }
    else {
        sprintf (send_data, "%s\r", clear_relays);
        PutString(mx_port,send_data);           // clear channel relays
    }
  }
  printf("Test Passed\n");
}
```

Appendix A. Serial Command Set

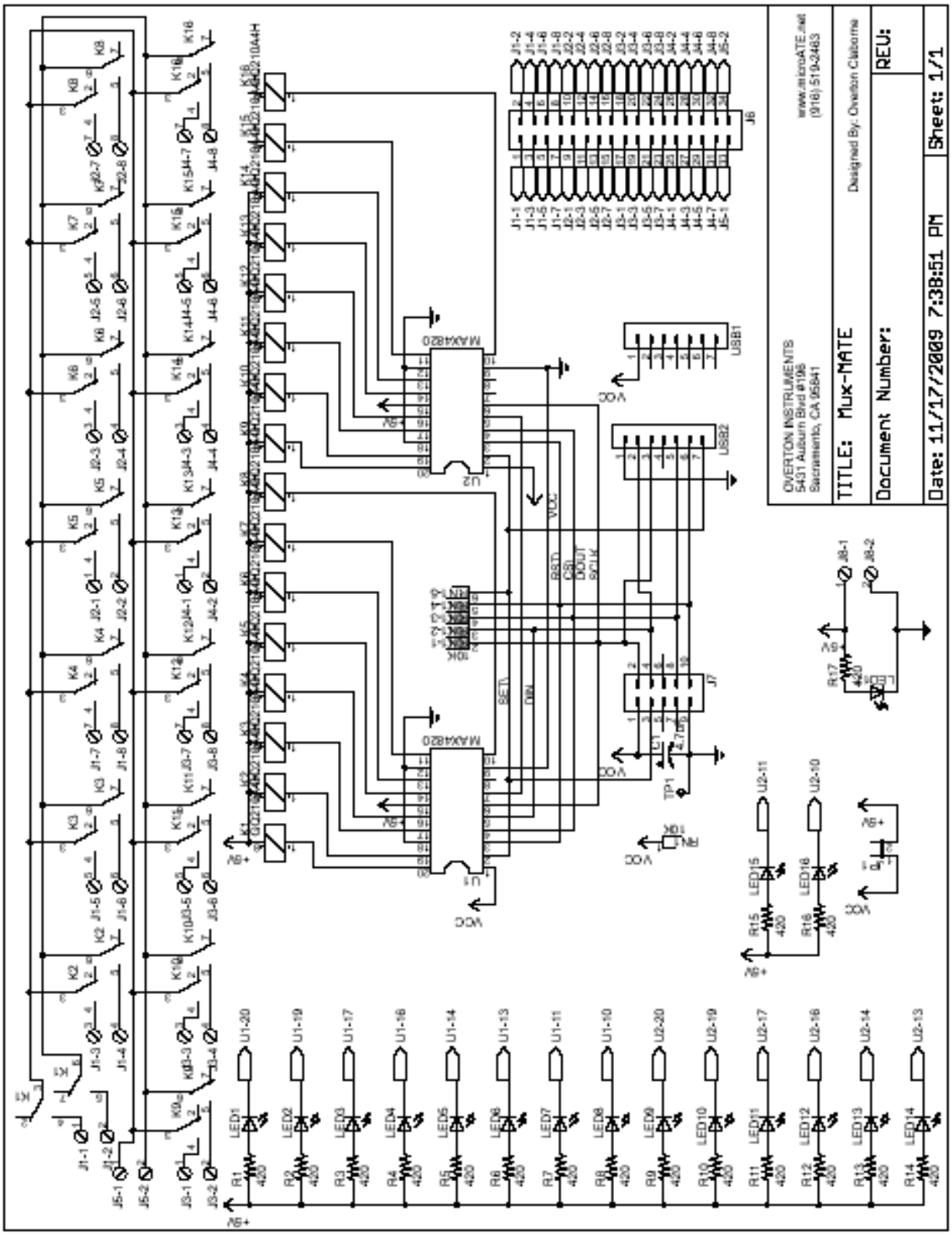
To facilitate remote control for the MUX-MATE(v1) , a USB interface is required. When connected to a host PC, the USB connection appears as a "Virtual Com Port", which establishes a serial data communications link between the two. The default protocol is 19200 baud rate, no parity, 1 stop bit and no flow control. The MUX-MATE(v1) will respond to a unique set of ASCII serial data commands (listed below). The first three bytes of the command string starts with the prefix **'MX_',** followed by a code that represents the actual command. All commands are upper case sensitive and are terminated with a carriage-return. If the command is valid, the MUX-MATE(v1) will return either a **'<>'**, or a bracketed result (i.e. **'<3>'**. If the MUX-MATE(v1) receives a carriage-return or line-feed alone (without a command), then a **'→'** is returned (this response is a "prompt" to signal the MUX-MATE(v1) is ready). If the MUX-MATE(v1) detects an incorrect command then one of three error symbols will be generated, (1) invalid command then a **'><'** is returned, (2) a command that is out-of-limits then a **'>>'** is returned, and (3) a command is incorrect for a given mode, then a **'<<'** is returned. In some cases the error symbol will include a bracketed result (i.e. '**>1<'**), which defines a specific error code.

Command	Function	Response	Description	Example
MX_BRn	Set baud rate code	<n>	Select 1 of 4 different baud rates by changing -n-code. 0 = 1200, 1 = 2400, 2 = 9600 & 3 = 19200. Baud will remain set. Default code is 3 (19200).	MX_BR2
MX_BR?	Get baud rate code	<n>	Get current baud rate code (-n- is the return code 0 to 3).	
MX_ID?	Get module ID	<MUX-MATE(v1) rx.x>	Get module current identification and version number.	
MX_MR	Maser Reset	<>	Reset & initialize the module	
MX_SRnnn	Select relay & turn On/ Off	<>	Select relays by changing the first two -nn- codes (01 to 08). The third –n– code sets the relay On/ Off (1 or 0).	MX_SR110
MX_RS?	Get relay status	<ssssss...s>	The results are placed in 16 ASCII bytes (relay 16 to1, left to right). The -s- represents the relay status (1 or 0, Close or Open).	<0000010000111101>
MX_MC	Clear relays	<>	Open all relays	
MX_MS	Set relays	<>	Close all relays	
MX_MM	Set Mux Mode	<n>	Set Mux Mode ('1' = Enable Break-Before-Make operation, '0' = Disable Break-Before-Make operation)	
MX_MM?	Get Mux Mode	<n>	Get Mux Mode Status ('1' = Break-Before-Make, '0' = Break-Before-Make)	

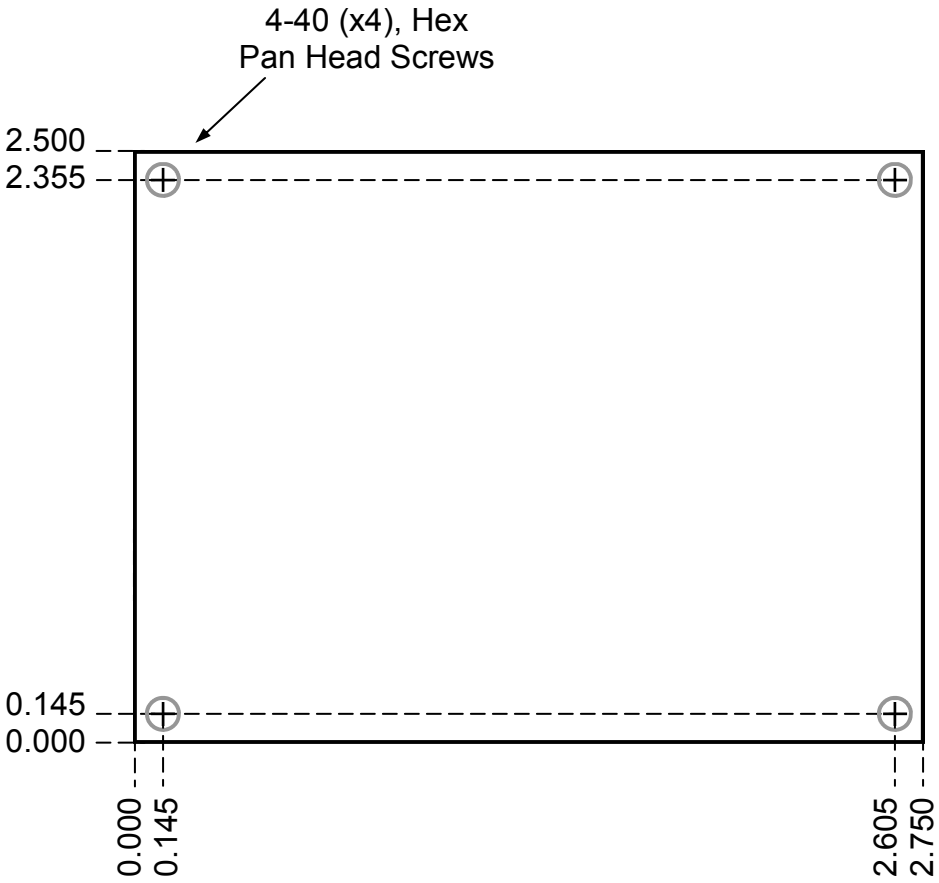
Appendix A. Serial Command Set cont.

Command	Function	Response	Description
MX_STnnn	Set settling time	<>	Set the relay settling time. Once a relay is closed, a variable time delay is provided to allow the relay to stop bouncing. The settling time value -nnn-, must be a 3 byte ASCII number from "001" to "255", which represents 1 to 255 msec. The default value is 10msec.
MX_ST?	Get settling time	<nnn>	Get the current relay settling time.

Appendix B. Schematic



Appendix C. Mechanical Dimensions



MG-Products B.V.
Rijkevoortsedijk 27A
5447 BD, Rijkevoort (NL)
Phone: +31 (0)485 382 133
www: www.designedfortest.com
E-mail: Info@designedfortest.com