

EMBEDDED TEST SOLUTIONS

SF-MATE

8-CH, SHORT CIRCUIT
Measurement Module



USER'S MANUAL



Overton Instruments, Inc
5431 Auburn Blvd. #196
Sacramento, CA 95841
www.microATE.net

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.

WARRANTY 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

1.0 INTRODUCTION	4
1.1 Overview	4
1.2 Highlights	4
1.3 Solutions	5
1.4 Specifications	6
2.0 DESCRIPTION	7
2.1 Overview	7
2.2 Relay Scanner	7
2.3 Short-Finder	7
2.4 Board Layout	8
3.0 CONNECTIONS	9
3.1 J1 & J2, Relay Scanner	9
3.2 J3, External Power	9
3.3 J4, External Source	9
3.4 J5, Controller Interface	10
3.5 J6, Signal Consolidated	11
4.0 OPERATION	12
4.1 Embedded Control	12
4.1.1 Embedded Configuration	13
4.1.2 Embedded Programming	14
4.1.3 Embedded Program Example	15
4.2 PC Control	16
4.2.1 PC Programming	17
4.2.1.1 HyperTerminal	17
4.2.1.2 Virtual Instrument Panel	18
4.2.1.3 PC Programming Example	19
APPENDIX A. SERIAL COMMAND SET	21
APPENDIX B. SCHEMATIC	22
APPENDIX C. MECHANICAL DIMENSIONS	23

1. Introduction

1.1 Overview

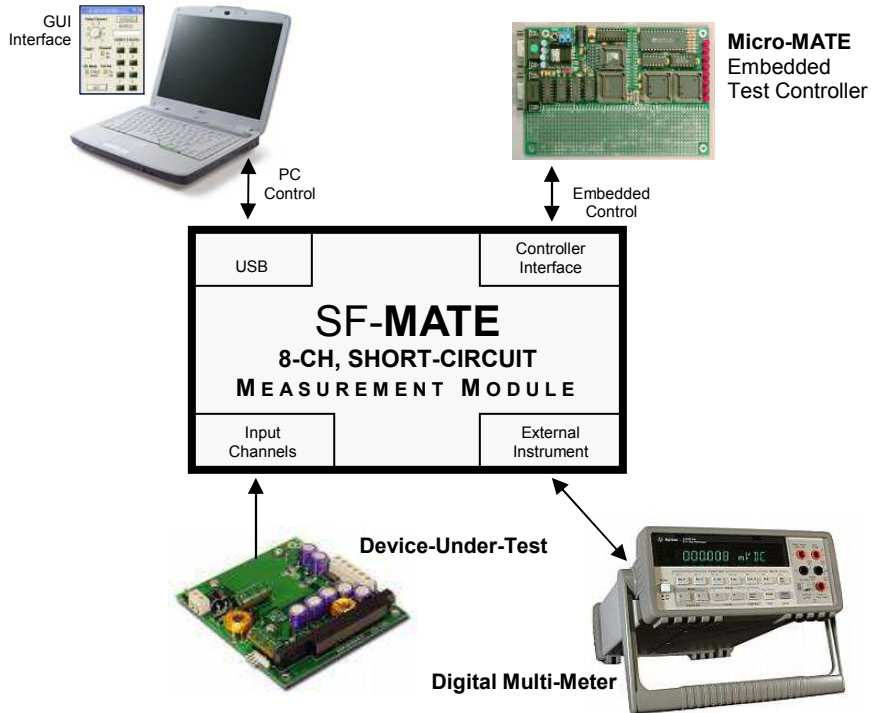
The SF-MATE (*or Short-Finder*), is a unique test instrument that adds ICT capability to Functional Test equipment. Rather than spending thousands of dollars to test all nodes on a PCB, the SF-MATE limits the number of checks to those defined as “critical” test points. For example, during a typical assembly process, a PCB can receive inadvertent “shorts” in the power section. By verifying certain test points are “short free” (prior to applying power to the PCB), the SF-MATE can prevent damage to the DUT, adjoining test equipment and possible injury to the test Operator.

The SF-MATE has 8 input channels that are connected to a special Ohm meter circuit. After a channel is selected, a constant current is supplied to the device-under-test and a voltage is measured that is proportional to the resistance. The Ohm meter limits the current source to 1mA, and the open-circuit voltage is just 200mV (which is less than the nominal turn-on voltage for most PN junctions). When the input exceeds a certain level, the SF-MATE outputs a digital bit that indicates a short .

1.2 Highlights

BENEFITS	APPLICATIONS	FEATURES
<ul style="list-style-type: none"> • A flexible, low-cost alternative to traditional ICT test equipment • Functions both as a Short-Finder and Voltage Scanner • Can be used in fully automated test equipment • Great for embedded solutions - place inside mechanical test fixtures, instrument boxes or rack-mount enclosures 	<ul style="list-style-type: none"> • Functional Test solutions • Automated Test Systems • QA/QC Quality Control • OEM Test Instruments 	<ul style="list-style-type: none"> • Verify “key” test points in <10msec • 8 DPDT relays isolate measurement • LED’s indicate on all active relay channels • USB interface or embedded control • Low cost • Compact size

1.3 Solutions



1.4 Specifications

Relays (K1-K9)	
Relays	9
Relay type	DPDT (Form C)
Coil Voltage	+5Vdc
Nom. Switching Capacity	0.3 A, 125 V AC (Resistive Load) 1 A, 30 VDC (resistive load)
Max. Switching Voltage	110 VDC, 125 VAC
Max. Switching Current	1A
Contact resistance	100mΩ max
Relay lifetime	100,000,000 operations
Actuation time	4ms max operate or release
Short Detector Circuit	
Source voltage	200mV
Max Source Current	1mA
Continuity Threshold	4 ohms
Short Flag	A high, TTL level
General	
Power supply	+5VDC ±10%, 500mA min.
Operating temperature	0 to +70°C
Operating humidity	5% to 95% non-condensing
Dimensions	2.0" x 4.0"
Weight	

2. Description

2.1 Overview

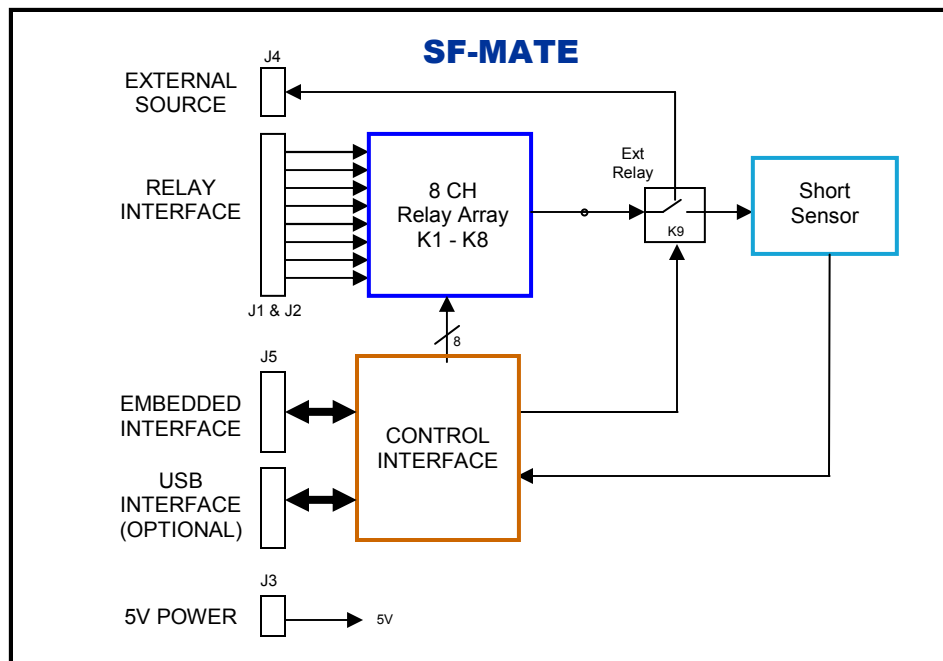
The SF-MATE performs two separate functions, (1) a Relay Scanner and (2) Short-Finder. A simplified block diagram is shown below to highlight key circuits.

2.2 Relay Scanner

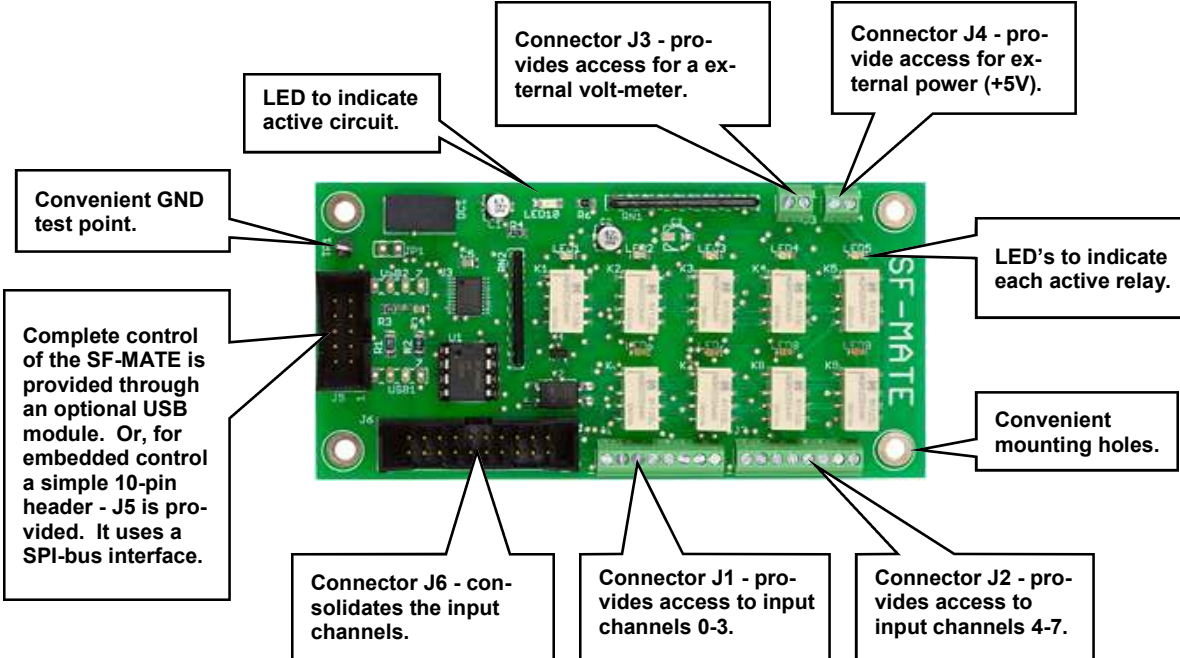
Eight DPDT Form C relays (K1 - K8) are bussed together on the “normally-open” side and are connected to the Short Sensor circuit via K9. The relays (K1- K9) are all general purpose (+5V coil voltage), with a nominal switch rating of 30Vdc @ 1A (125Vac @ 0.3A). Cycle time for a single relay channel is 8msec (cycle time combines both set and release time). When relay K9 is active, the Relay Scanner can be used to route signals to external test equipment.

2.3 Short-Finder

When the Short Sensor measures an impedance of ~4 ohms it produces a logic “high” output. A single measurement can take place in 2msec, or “scan and measure” all 8 channels in 80msec (includes relay settling time).



2.4 Board Layout



3. Connections

3.1 J1 & J2, Relay Scanner

Access to the Relay Scanner is made possible through connector J1 & J2. J1 & J2 contains two 8 screw terminal connections (pin assignments are presented in the table to the right). Each relay has a corresponding LED, which should turn-on when a relay is active.

J1 & J2		
Pin	Name	Relay
J1-1	CH0-HI	K1
J1-2	CH0-LO	K1
J1-3	CH1-HI	K2
J1-4	CH1-LO	K2
J1-5	CH2-HI	K3
J1-6	CH2-LO	K3
J1-7	CH3-HI	K4
J1-8	CH3-LO	K4
J2-1	CH4-HI	K5
J2-2	CH4-LO	K5
J2-3	CH5-HI	K6
J2-4	CH5-LO	K6
J2-5	CH6-HI	K7
J2-6	CH6-LO	K7
J2-7	CH7-HI	K8
J2-8	CH7-LO	K8

3.2 J3, External Power

J3 provides a set of screw terminals that allows connection to the relay array which accepts power from a fairly well-regulated +5Vdc power source (minimum 500mA). Connect the plus-lead to J3-1, and the negative (or ground) lead to J3-2. When power is applied LED-11 should turn-on.

J3			
Pin	Name	Dir.	Description
1	+5Vdc	→	A regulated +5Vdc input .
2	GND	→	Ground

3.3 J4, External Source

J4 provides a set of screw terminals that allows connection to external test equipment or to connect multiple SF-MATE modules together. Connect the plus-lead to J4-1, and the negative-lead to J4-2. When relay K9 is active, the output of the Relay Scanner is switched to J4.

J4			
Pin	Name	Dir.	Description
1	Ext. Source (+)	→	Ext Instrument (+)
2	Ext. Source (-)	→	Ext Instrument (-)

3.4 J5, Controller Interface

Control of the SF-MATE is made possible through connector J5 (a standard 10-pin dual row header). A description for the various pins are provided in the table on the right. All signals conform to TTL digital logic levels. For more information regarding specific requirements for interfacing to the SF-MATE, please visit the "Operating" section.

J5			
Pin	Name	Dir.	Description
1	VCC	I	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 for the DIN and DOUT signals.
3	RESET\	I	An TTL active-low "input" signal that causes relays K1-K8 to open.
4	DIN	I	Part of a 3-wire SPI-Bus, DIN is serial command and control data for the, ADC, DAC and DIO circuits.
5	EXT_SOURCE\	I	An TTL active-low "input" signal that enables K9 - External Relay.
6			
7	SHORT	O	An TTL active-high "output" signal that indicates a 'short-circuit' condition is present.
8	SF_CS\	I	An TTL active-low "input" signal that provides a chip-select for the DIO.
9	DGND	I	Digital Ground
10	SET\	I	An TTL active-low "input" signal that causes relays K1-K8 to close.

3.5 J6, Signal Consolidated

Access to the Relay Scanner is made possible through connector J1 & J2. J1 & J2 contains two 8 screw terminal connections (pin assignments are presented in the table to the right). Each relay has a corresponding LED, which should turn-on when a relay is active

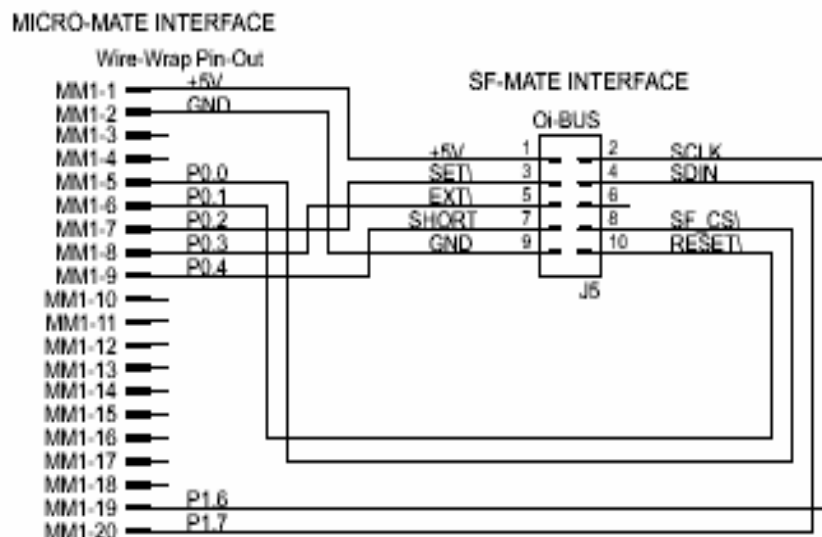
J6		
Pin	Name	Relay
1	CH0-HI	K1
2	CH0-LO	K1
3	CH1-HI	K2
4	CH1-LO	K2
5	CH2-HI	K3
6	CH2-LO	K3
7	CH3-HI	K4
8	CH3-LO	K4
9	CH4-HI	K5
10	CH4-LO	K5
11	CH5-HI	K6
12	CH5-LO	K6
13	CH6-HI	K7
14	CH6-LO	K7
15	CH7-HI	K8
16	CH7-LO	K8
17	EXT-HI	K9
18	EXT-LO	K9
19	+5Vdc	
20	GND	

4. Operation

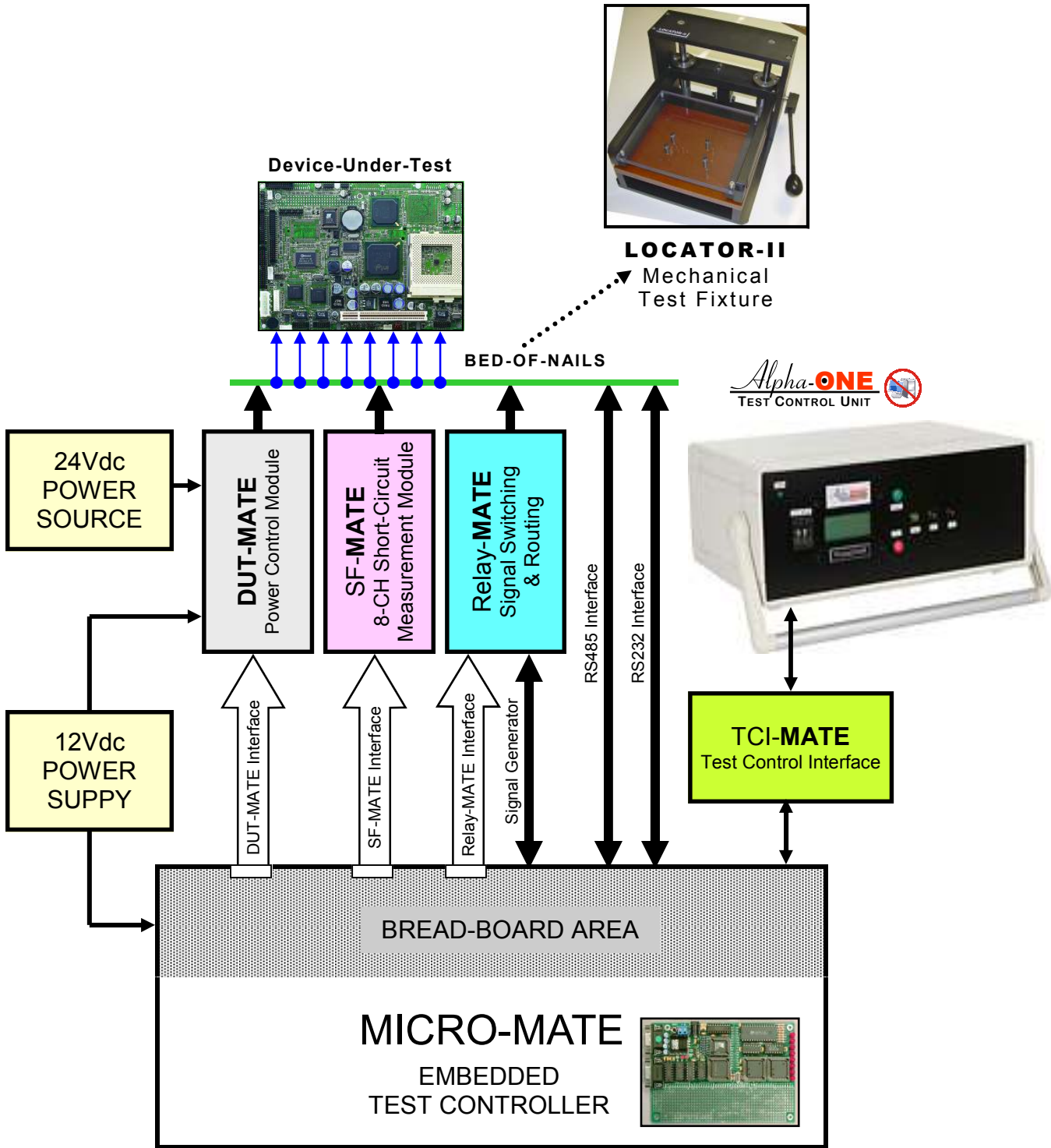
4.1 Embedded Control

In section 3.1.1 (on the next page), the SF-MATE is shown integrated with other ETS Series components that collectively form a complete Embedded Test Solution. The diagram shows the SF-MATE 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 that comprise 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 SF-MATE (J5, 10-pin header connector).

Actually the SF-MATE can be easily driven by most microcontrollers (including an ARM, AVR, PIC or even a STAMP). When developing a custom interface for the SF-MATE, it is recommended the designer start-by reviewing the interface requirements as outlined in the J5 Table (which is provided in the Connections section). The next step is to review the SF-MATE 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 SF-MATE 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 web-site to download said documents.



4.1.1 Embedded Configuration



4.1.2 Embedded Programming

To build-on the PCB board test example (shown in section 4.1.1), we have constructed a demo program using BASCOM. BASCOM is a BASIC language compiler that includes a powerful Windows IDE (Integrated Development Environment), and a full suite of “QuickBASIC” like commands and statements. The demo program (which is outlined in section 4.1.3), illustrates the ease of controlling the SF-MATE via the Micro-MATE microcontroller.

The program starts by initialing the Micro-MATE for proper operation. You will note that the BASCOM software provides excellent bit-manipulation capabilities, as evident by the use of the ALIAS statement. The Micro-MATE (P1 port bits) are assigned unique label names (i.e., SCLK, DOUT), which are used to support various SF-MATE 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 “SF_SS” command is entered, the “Sf_short_scan” and “Sf_get_short(sf_str)” subroutines are called. This causes the SF-MATE to scan all relay channels for shorts, the program then converts the measurement to an ASCII byte and the results are returned. Next, the “SF_SE?” command is entered, the program then returns the current status of the External relay (active or not active, represented by logic “1” or “0”). “

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

4.1.3 Embedded Program Example

```

' Program: SF-MATE Demo
'
---[ Initialization ]-----
$large
$romstart = &H2000
$default Xram

Dim Sf_bit As Bit
Dim A_num, A_byte, A_cnt As Byte
Dim Sf_byte, Sf_cnt, Sf_settle, Sf_status, Sf_num as Byte
Dim S As String * 10, A_resp As String * 10, A_str As String * 10
Dim Sf_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 P1.6           ' SPI-bus serial clock
Dout Alias P1.7          ' SPI-bus serial data output
Din Alias P1.5           ' SPI-bus serial data input
Sf_cs Alias P0.0         ' Relay driver chip select
Sf_rst Alias P0.1        ' Reset relay driver chip
Sf_set Alias P0.2        ' Set relay driver chip
Sf_ext Alias P0.3        ' External relay control
Sf_short Alias P0.4      ' Short condition

Declare Sub Print_ic      ' print invalid command
Declare Sub Print_oor     ' print out-of-range
Declare Sub Print_ur      ' print under range
Declare Sub Print_ok      ' print command is OK
Declare Sub Sf_short_scan ' check for shorts on all channels
Declare Sub Sf_get_shrot(sf_str As String) ' get ascii short byte
Declare Sub Sf_rly_sel(sf_num As Byte, Sf_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. Only two command examples are
' shown.

Set Sclk, Dout, Sf_cs, Sf_rst, Sf_set, Sf_ext ' Set to logic '1'
Do
  Input "Enter command ", S
  S = Ucase(s)
  A_resp = Left(s, 3)
  If A_resp = "SF_" Then
    A_resp = Mid(s, 4, 2)
    Select Case A_resp
      Case "SS":           ' scan relays & chk for shorts

        A_char = Mid(s, 6, 1)
        If A_char = "?" Then
          Call Sf_short_scan
          Call Sf_get_shrot(sf_str)
          Print "<"; Sf_str; ">"
        Else
          Call Print_ic
        End If
      Case "SE":           ' set/get ext relay

        A_char = Mid(s, 6, 1)
        If A_char = "?" Then
          If Sf_ext = 1 Then A_char = "1"
          If Sf_ext = 0 Then A_char = "0"
          Print "<"; A_char; ">"
        Else
          If A_char <> "0" And A_char <> "1" Then Call Print_oor
          If A_char = "0" Then Reset Sf_ext
          If A_char = "1" Then Set Sf_ext
          Call Print_ok
        End If
    End Select
  End If

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

'---[ 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

' Scan 8 relay channels and check short condition
Sub Sf_short_scan
  For Sf_cnt = 0 To 7
    Call Sf_rly_sel(sf_cnt, 1)
    Sf_num = 7 - Sf_cnt
    Sf_byte.sf_num = Sf_short
  Next Sf_cnt
  Call Sf_rly_clr
End Sub

' Select specific relay
Sub Sf_rly_sel(sf_num As Byte, Sf_bit As Bit)
  Sf_status = 0
  Sf_status.sf_num = Sf_bit
  Sf_status = Not Sf_status
  Reset Sf_cs
  For Sf_cnt = 7 Downto 0
    Dout = Sf_status.sf_cnt ' serial data out
    Set Sclk
    Delay
    Reset Sclk
    Delay
  Next Sf_cnt
  Set Sf_cs
  Waitms Sf_settle         ' settling time
  Sf_status = Not Sf_status
End Sub

' Identify shorts & convert to ascii
Sub Sf_get_shrot(sf_str As String)
  Sf_str = "00000000"
  For Sf_cnt = 7 Downto 0
    Sf_bit = Sf_byte.sf_cnt
    If Sf_bit = 0 Then Sf_char = "0"
    If Sf_bit = 1 Then Sf_char = "1"
    Sf_num = 8 - Sf_cnt
    Mid(sf_str, Sf_num, 1) = Sf_char
  Next Sf_cnt
End Sub

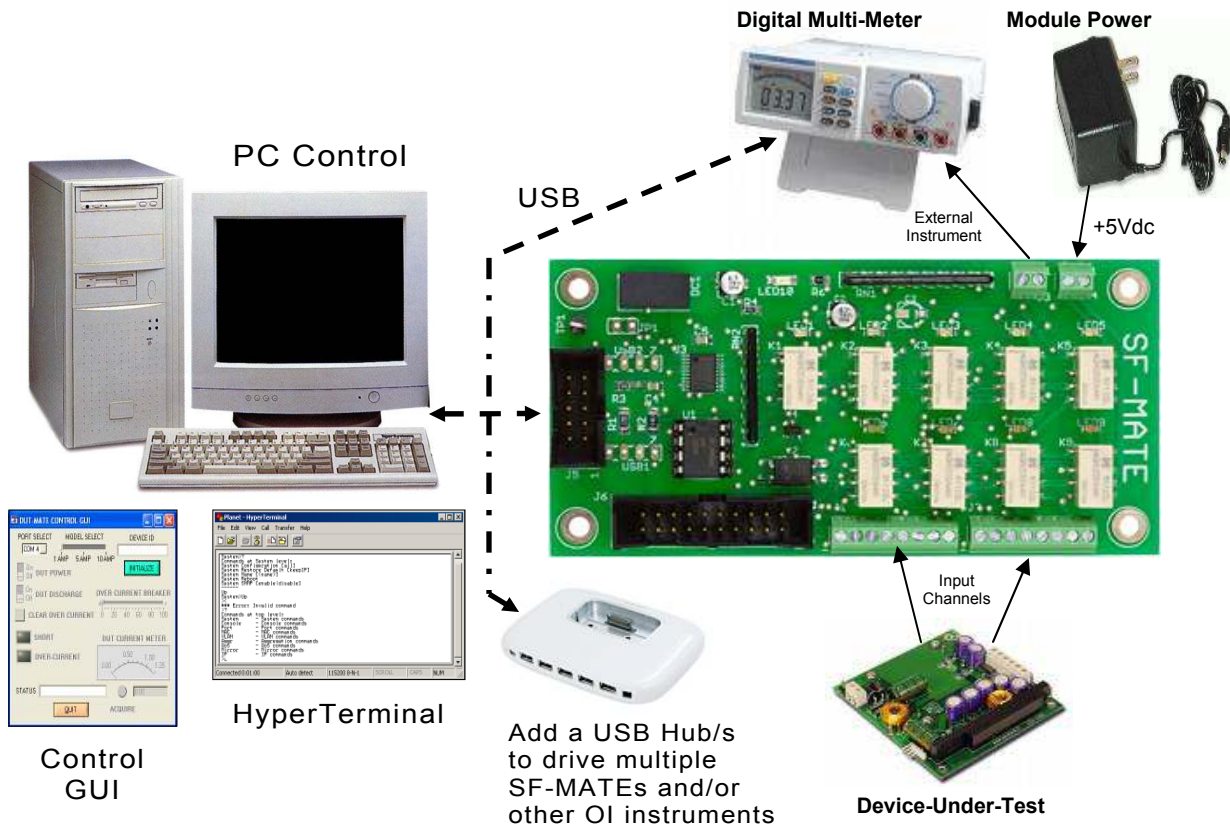
```

4.2 PC Control

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

Controlling the SF-MATE 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 SF-MATE side, the PIC controller processes the serial commands and then drives the SF-MATE 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.SF-MATE.info”, 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 of a PC-driven configuration. As shown, the SF-MATE relay channels are connected to the outputs of a multiple output Power Distribution PCB. Prior to applying power to the device-under-test, the SF-MATE is commanded to “scan for shorts. If no shorts are detected, then power can be safely applied to the DUT. After DUT power is ON, the SF-MATE can be commanded to enter external mode, which will the allow the voltage outputs from the DUT to be routed to an external instrument for measurement.

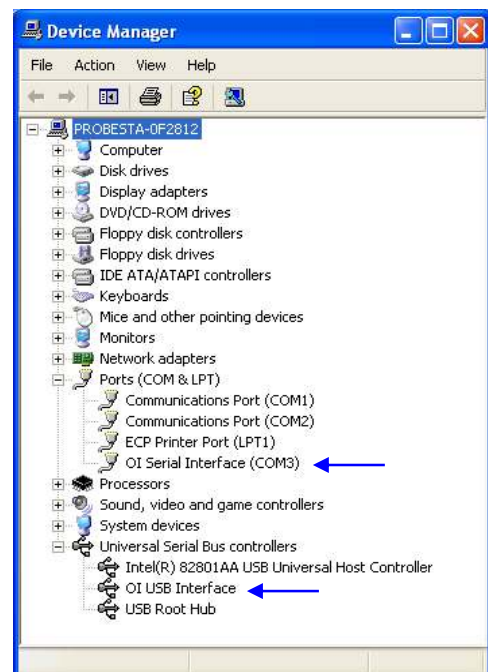
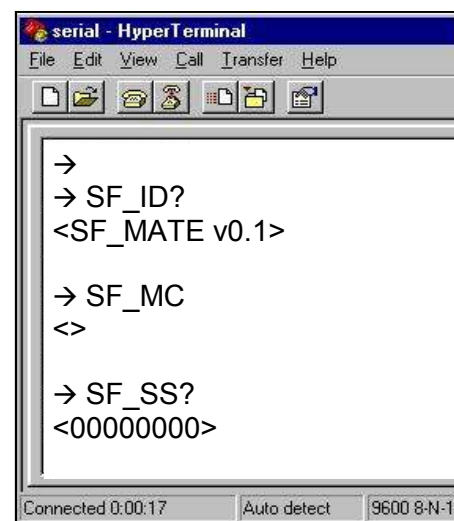


4.2.2 PC Programming

The starting point for developing code to control the SF-MATE, 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 SF-MATE 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).

4.2.1.1 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 SF-MATE. 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 B, to begin to experiment with the serial commands.



4.2.1.2 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 SF-MATE. Download the panel from our website at www.sf-mate.info, click on downloads and select “SF-Matexxx.exe”.

First Step: The User must select a COM Port. Refer to the Device Manage to identify an available COM port.

Second Step: Push the Initialize button. This will cause the module to initialize itself and attempt to establish a communications link.

Third Step: After initializing, the module should send back a unique ID code. If no response has occurred within 10 seconds, the program will time-out , and generate a No Response message.

The 'Relay Channel' function selects an individual relay channel (1 to 8).

The 'Ext Sw' switch enables the external relay.

The 'Channel' switch enables an individual channel relay.

The 'CH Mode' switch selects 'SCAN' or 'SINGLE'

The 'Trigger' function updates the switch settlings.

The 'STATUS' message box summarizes results of the serial commands.

The 'SHORT' STATUS' LED'S indicates an short-circuit condition.

The screenshot shows the SF-MATE CONTROL GUI with the following elements: 'PORT SELECT' dropdown set to 'COM 1', 'INITIALIZE' button, 'Relay Channel' rotary switch, 'Ext Sw' and 'Channel' toggle switches, 'CH Mode' toggle switch set to 'SINGLE', 'Trigger' checkbox, 'SHORT STATUS' 8 LED indicators, 'STATUS' text box showing 'DEVICE READY', and 'QUIT' button.

4.2.1.3 PC Programming Example

```

// SF-MATE 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 SF-
// MATE, DUT-MATE 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. Before DUT power is
// applied, the SF-MATE is used to verify no shorts exist on the power-rails.
// The DUT-MATE is used to switch power to the device-under-test. After
// power is applied, the SF-MATE 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.wcsnet.com

#include <comm.h>
#include <stdlib.h>
#include <stdio.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 scan_shorts[] = "SF_SS?" // scan all channels for shorts
char clear_relays[] = "SF_CR" // clear all channel relays
char select_relay[] = "SF_SR" // select specific relay
char set_ext_relay[] = "SF_SE" // set ext relay On/Off
char get_ext_relay[] = "SF_SE?" // get ext relay status
char sf_master_clear[] = "SF_MC" // master clear
char sf_get_device_id[] = "SF_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
    sf_port = OpenComPort(1,256,64); // Open COM 1, SF-MATE
    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(sf_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; // SF-MATE 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", sf_get_device_id);
            if (a_cnt == 2) sprintf (send_data, "%s\r", dt_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, "<SF-MATE 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", sf_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 <= 10; a_cnt++) {
        switch (a_cnt) {
            case 1: // Short-Circuit Test
                sprintf (send_data, "%s\r", scan_shorts);
                PutString(sf_port,send_data);
                GetString(sf_port,sizeof(read_data),read_data);
                if (strcmp("<00000001>", read_data)) {
                    printf ("Short-Circuit failure - %s", read_data);
                    test_fail = True;
                }
                break;
            case 2: // 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;
                }
        }
    }
}

```

4.2.1.3 PC Programming Example cont.

```

case 3: // DUT Input Power Test - 24Vdc
    sprintf(send_data, "%s%s\r", set_ext_relay, "1");
    PutString(sf_port, send_data); // enable ext relay
    sprintf(send_data, "%s%s\r", select_relay, "11");
    PutString(sf_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 4: // DUT +5Vdc Logic Power Test
    sprintf(send_data, "%s%s\r", select_relay, "21");
    PutString(sf_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 5: // DUT +12Vdc Analog Power Test
    sprintf(send_data, "%s%s\r", select_relay, "31");
    PutString(sf_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 6: // DUT -12Vdc Analog Power Test
    sprintf(send_data, "%s%s\r", select_relay, "41");
    PutString(sf_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 7: // DUT 32Vac Power Test
    sprintf(send_data, "%s%s\r", select_relay, "51");
    PutString(sf_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 8: // DUT RTC Oscillator Test - 32.768 kHz
    sprintf(send_data, "%s%s\r", select_relay, "61");
    PutString(sf_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 9: // DUT Heater Element Test - 13.5 ohms
    sprintf(send_data, "%s%s\r", select_relay, "71");
    PutString(sf_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 10: // DUT 0-Ohm Jumper Test
    sprintf(send_data, "%s%s\r", select_relay, "81");
    PutString(sf_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_DPO
    exit();
}
else {
    sprintf(send_data, "%s\r", clear_relays);
    PutString(sf_port, send_data); // clear channel relays
}
}
printf("Test Passed\n");
}

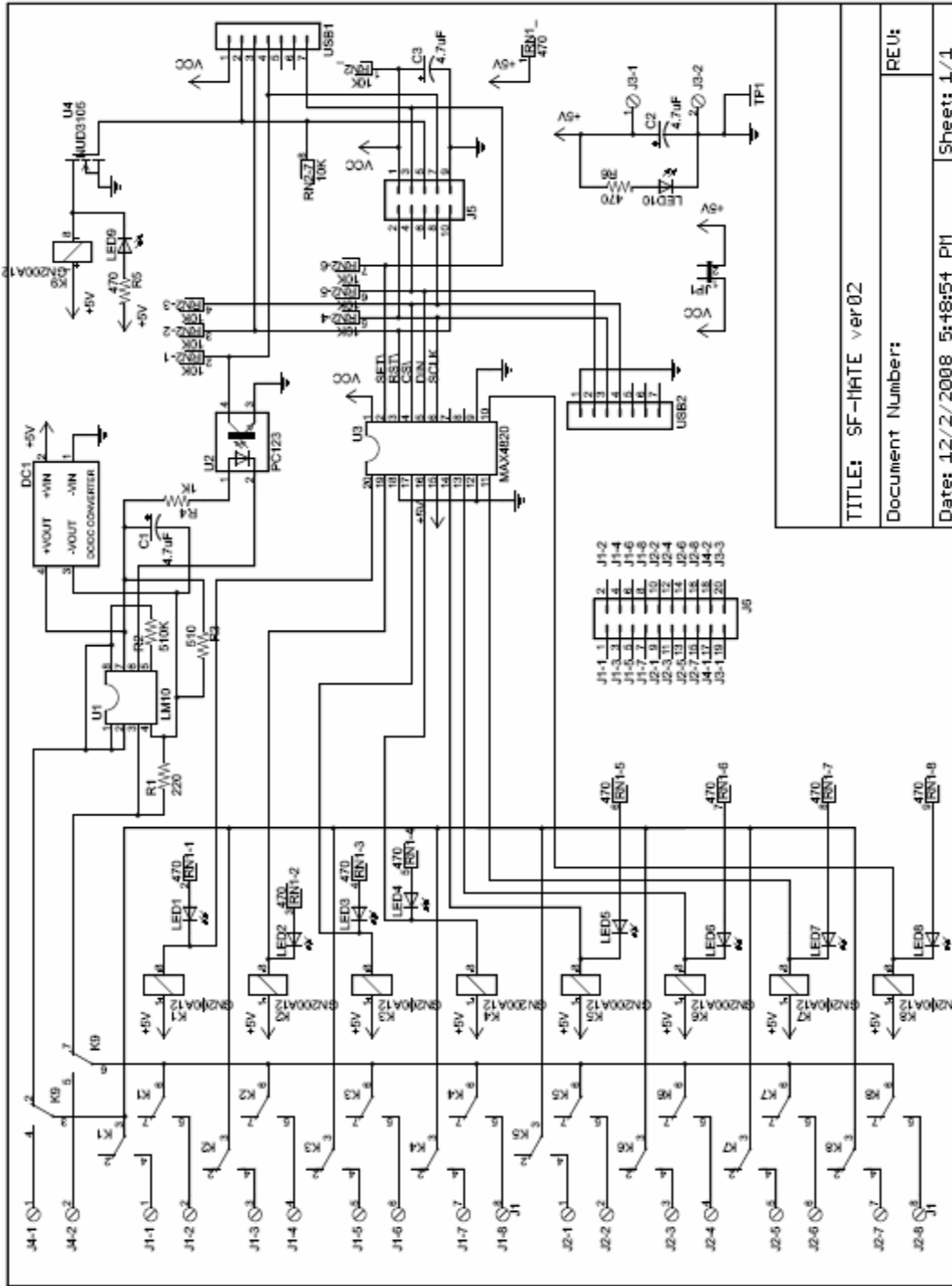
```

Appendix A. Serial Command Set

To facilitate remote control for the SF-MATE, 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 SF-MATE 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 'SF_', 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 SF-MATE will return either a '<>', or a bracketed result (i.e. '<010>'). If the SF-MATE receives a carriage-return or line-feed alone (without a command), then a '→' is returned (this response is a "prompt" to signal the SF-MATE is ready). If the SF-MATE 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 that prematurely times-out 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
SF_BRn	Set baud rate code	<n>	Select one 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).
SF_BR?	Get baud rate code	<n>	Get current baud rate code (-n- is the return code 0 to 3).
SF_CR	Clear channel relays	<>	All relays (excluding Ext), are cleared (non-energized).
SF_ID?	Get module ID	<SF-MATE vx.x>	Get module current identification and version number.
SF_MC	Maser Clear	<>	Reset & initialize the module
SF_SEn	Set external relay	<>	Activate or disable the External Source relay. The -n- represents logic state (1 or 0, On or Off).
SF_SE?	Get external relay	<n>	Get current status of the External Source relay. The -n- represents logic state (1 or 0, On or Off).
SF_SRnnn	Set channel relay	<>	First -nn- represents relay channel (00 to 07). Third -n- represents logic state (1 or 0).
SF_SR?	Get channel status	<bbbbbbbb>	The results are placed in 8 ASCII bytes (channel 0 is high-order-byte and channel 7 is low-order-byte). The -b- represents the channel status (1 or 0, On or Off).
SF_STnnn	Set relay settling time	<>	Set channel relay settling-time. The -nnn- represents a number between 001 to 255 (padded zero's are required). The timing is stated in milliseconds and the default setting is 10msec.
SF_ST?	Get relay settling time	<nnn>	Get current channel relay settling-time
SF_SS?	Get short status	<bbbbbbbb>	Sequentially scan all relay channels and check for shorts. The results are placed in 8 Ascii bytes (channel 1 is high-order-byte and channel 8 is low-order-byte). The -b- represents the short status (1 = Short, 0 = No Short).

Appendix B. Schematic



TITLE: SF-MATE ver02
Document Number:
Date: 12/2/2008 5:48:54 PM
Sheet: 1/1

Appendix C. Mechanical Dimensions

