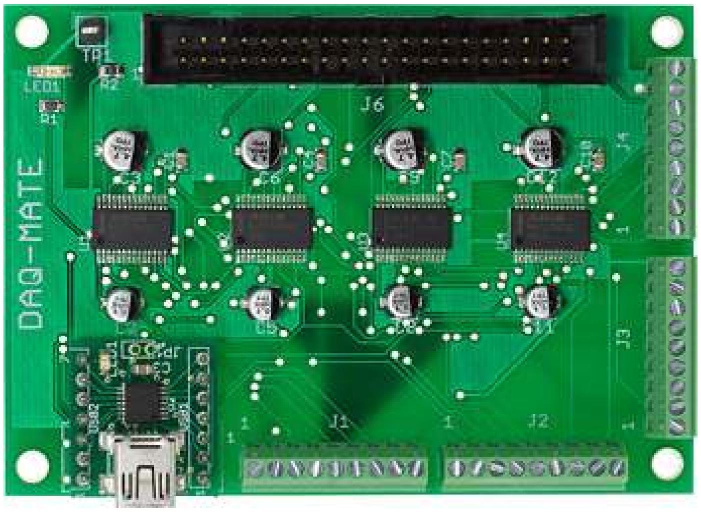
EMBEDDED TEST SOLUTIONS

DAQ-MATE

32-CH DATA AQUISITION MODULE



USER'S MANAUAL

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

NOTICE

WARNING

The instrument you have purchased and are about to use may be NOT 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 DEMAGES 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. 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

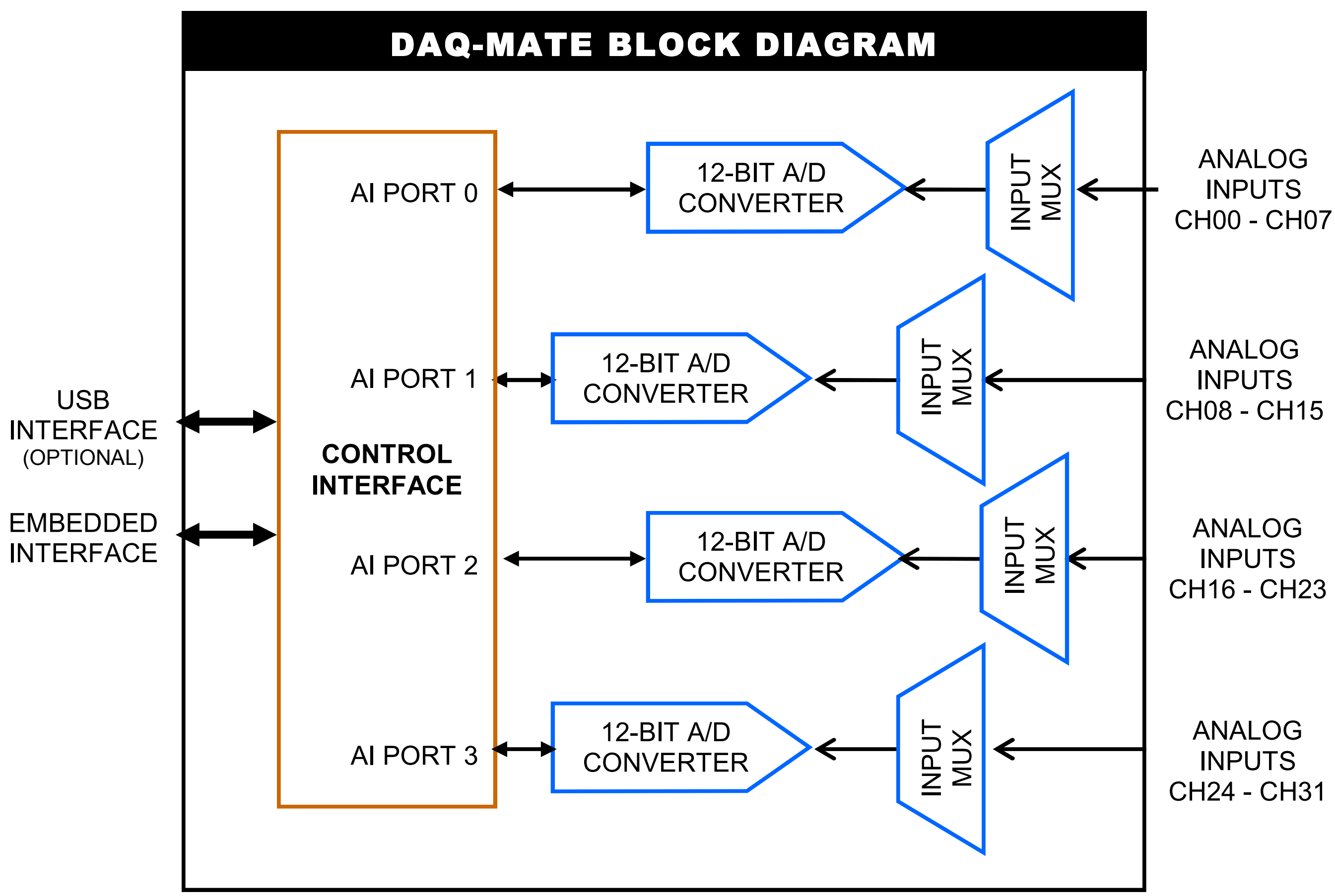
1. Introduction

1.1 Overview

The DAQ-MATE offers an impressive 32-channels of single-ended analog data acquisition, including 12-bit resolution (and a sample rate of 110KHz). In addition each channel can be independently programmed for 4 different input ranges.

The DAQ-MATE is made available is two versions, a standard model or with a USB option. The standard model is designed for embedded applications and provides a simple SPI-bus interface for control by a external microcontroller. With the USB option, many test solutions can be quickly built by connecting the DAQ-MATE 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. Any either case, easy access to the hardware is made available through a convenient collection of screw terminal connectors.

.



1.2 Highlights

BENEFITS	APPLICATIONS	FEATURES
• A flexible, low-cost alternative to expensive PC-based DAQ cards • Quickly measure a wide array of analog signals. Each analog channel can be independently programmed for 4 different range modes • 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	• 32-Analog Input Channels (SE), 110Khz sample rate • 12-bit Resolution • Unipolar and Bipolar modes (0-5Vdc, 0-10Vdc, ±5Vdc & ±10Vdc) • USB or embedded control interface • Low Cost • Compact size, a 2.5" x 3.5" PCB, with four #4 mounting holes in each corner (spacers and hardware included)

1.3 Specifications

Analog Inputs	
Number of inputs	32-CH, 12-bit, single-ended
Input Ranges	0-5V, 0-10V, ±5V, ±10V programmable
Max Sample Rate	110KHz
Nonlinearity	±1LSB, no missing codes
Input Control	
Embedded	SPI-bus & control logic
USB Interface	Optional USB module
General	
Power Supply	+5VDC±10%@30mA
Operating Temp	0-50ºC
Dimensions	2.5" x 3.5"

2. Description

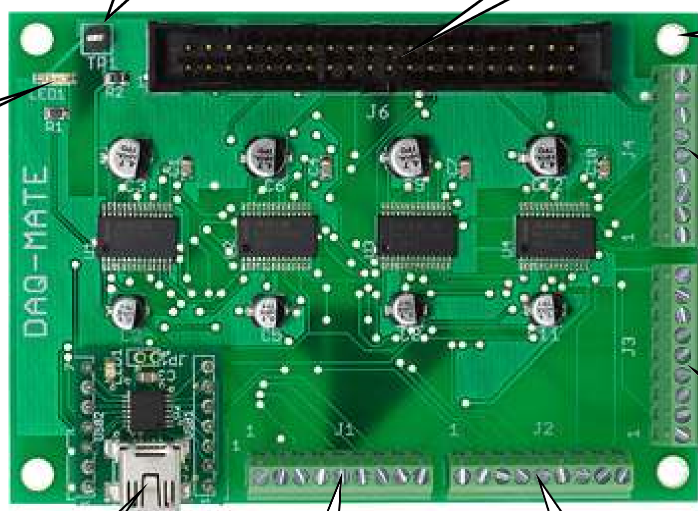
2.1 Board Layout

J6 - 40 Pin Header
Consolidates all 32 analog channels into a single header.

Convenient GND test point.

Convenient mounting holes.

LED to indicate active circuit.



J4 - 10 Pin Terminal
Provides access to analog channels 25-32.

J3 - 10 Pin Terminal
Provides access to analog channels 17-24.

USB Interface
Connectors USB-1 and USB-2 replaces J1, and allows connection to the USB-MATE.

J1 - 10 Pin Terminal
Provides access to analog channels 1-8.

J2 - 10 Pin Terminal
Provides access to analog channels 9-16.

J5 - 10 Pin Interface
Provides access for remote control via an Embedded controller.

2.2 Connections

J1			
Pin	**Name**	**Dir.**	**Description**
1	Port0-0	→	Input CH 1
2	Port0-1	→	Input CH 2
3	Port0-2	→	Input CH 3
4	Port0-3	→	Input CH 4
5	Port0-4	→	Input CH 5
6	Port0-5	→	Input CH 6
7	Port0-6	→	Input CH 7
8	Port0-7	→	Input CH 8
9	AGND	→	Analog Ground

J2			
Pin	**Name**	**Dir.**	**Description**
1	Port1-0	→	Input CH 9
2	Port1-1	→	Input CH 10
3	Port1-2	→	Input CH 11
4	Port1-3	→	Input CH 12
5	Port1-4	→	Input CH 13
6	Port1-5	→	Input CH 14
7	Port1-6	→	Input CH 15
8	Port1-7	→	Input CH 16
9	AGND	→	Analog Ground

J3			
Pin	**Name**	**Dir.**	**Description**
1	Port2-0	→	Input CH 17
2	Port2-1	→	Input CH 18
3	Port2-2	→	Input CH 19
4	Port2-3	→	Input CH 20
5	Port2-4	→	Input CH 21
6	Port2-5	→	Input CH 22
7	Port2-6	→	Input CH 23
8	Port2-7	→	Input CH 24
9	AGND	→	Analog Ground

J4			
Pin	**Name**	**Dir.**	**Description**
1	Port3-0	→	Input CH 25
2	Port3-1	→	Input CH 26
3	Port3-2	→	Input CH 27
4	Port3-3	→	Input CH 28
5	Port3-4	→	Input CH 29
6	Port3-5	→	Input CH 30
7	Port3-6	→	Input CH 31
8	Port3-7	→	Input CH 32
9	AGND	→	Analog Ground

2.2 Connections cont.

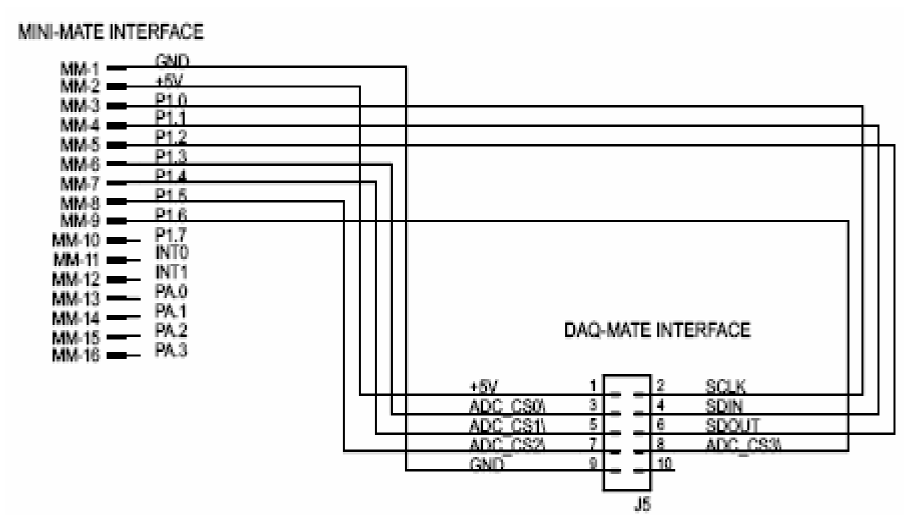
J5			
Pin	**Name**	**Dir.**	**Description**
1	VCC	→	A regulated +5Vdc output for external use. Current limited to roughly 100mA.
2	SCLK	→	Part of a 3-wire SPI-Bus, SCLK synchronizes the serial data transfer for the DIN and DOUT signals.
3	ADC_CS0	→	A TTL active-low "input' signal that provides a chip-select for the ADC, Port 0.
4	DIN	→	Part of a 3-wire SPI-Bus, DIN is serial command and control data for the, ADC ports.
5	ADC_CS1	→	A TTL active-low "input' signal that provides a chip-select for the ADC, Port 1.
6	DOUT	←	Part of a 3-wire SPI-Bus, DOUT is serial output data from the ADC and DIO circuits.
7	ADC_CS2	→	A TTL active-low "input' signal that provides a chip-select for the ADC, Port 2.
8			
9	DGND	→	Digital Ground
10	ADC_CS3	→	A TTL active-low "input' signal that provides a chip-select for the ADC, Port 3.

3. Operation

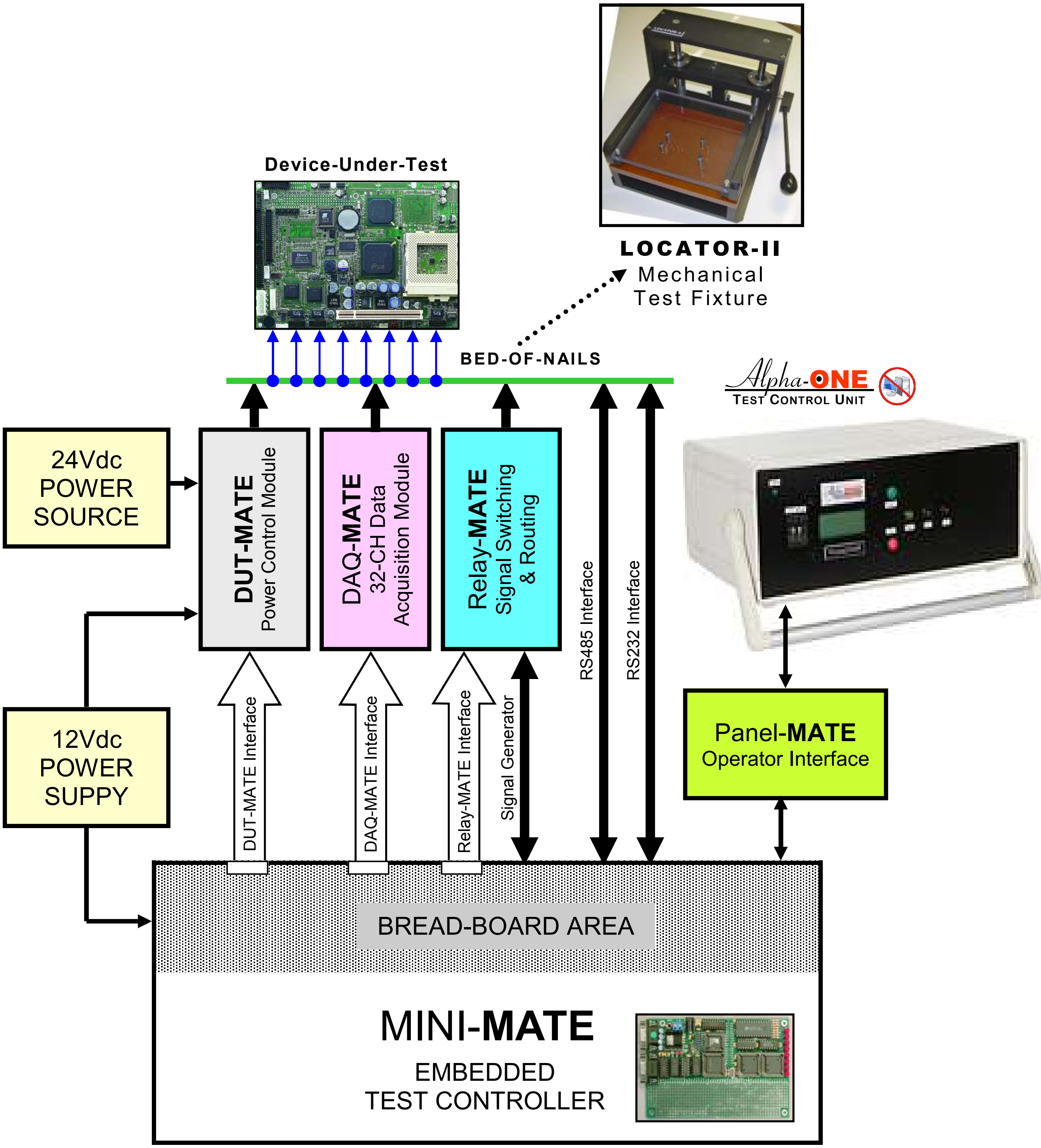
3.1 Embedded Control

In section 3.1.1 (on the next page), the DAQ-MATE is shown integrated with other ETS Series components that collectively form a complete Embedded Test Solution. The diagram shows the DAQ-MATE being driven by the Mini-MATE. The Mini-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 Mini-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 Mini-MATE and the DAQ-MATE (J1, 10-pin header connector).

Actually the DAQ-MATE can be easily driven by most microcontrollers (including an ARM, AVR, PIC or even a STAMP). When developing an interface for the DAQ-MATE, it is recommended the designer start-by reviewing the interface requirements as outlined in the J1 Table (which is provided in the I/O Description section). The next step is to review the DAQ-MATE schematic, which is provided in Appendix A. What could be the most challenging aspect of the design effort is controlling the SPI-bus devices. The DAQ-MATE contains 4 SPI-bus devices which are exactly the same analog-to-converter chip. The ADC is a 12-bit 8-channel data acquisition IC from Maxim (part number MAX1270). Details for specific device performance and SPI-bus operation can be found in the 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 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 3.2.3), illustrates the ease of controlling the DAQ-MATE via the Mini-MATE microcontroller.

The program starts by initialing the Mini-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 Mini-MATE (P1 port bits) are assigned unique label names (i.e., SCLK, DOUT), which are used to support various DAQ-MATE functions. In the "Main" program section, the Mini-MATE receives "high level" serial commands from a host PC, parses them and then executes accordingly. When (for example), the "DQ_CS17" command is entered, the program selects analog channel number 17. And then when command "DQ_AR1" is entered, the program selects the analog channel range (which is ±5Vdc). Finally, when command "DQ_RA?" is entered, the program call's subroutine "Daq_rd_adc(chk_adc_val , Daq_ch , Daq_adc_range)". This causes the DAQ-MATE to take an analog measurement and return the results in a 4 character hexadecimal "ASCII" string.

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

3.1.3 Embedded Program Example

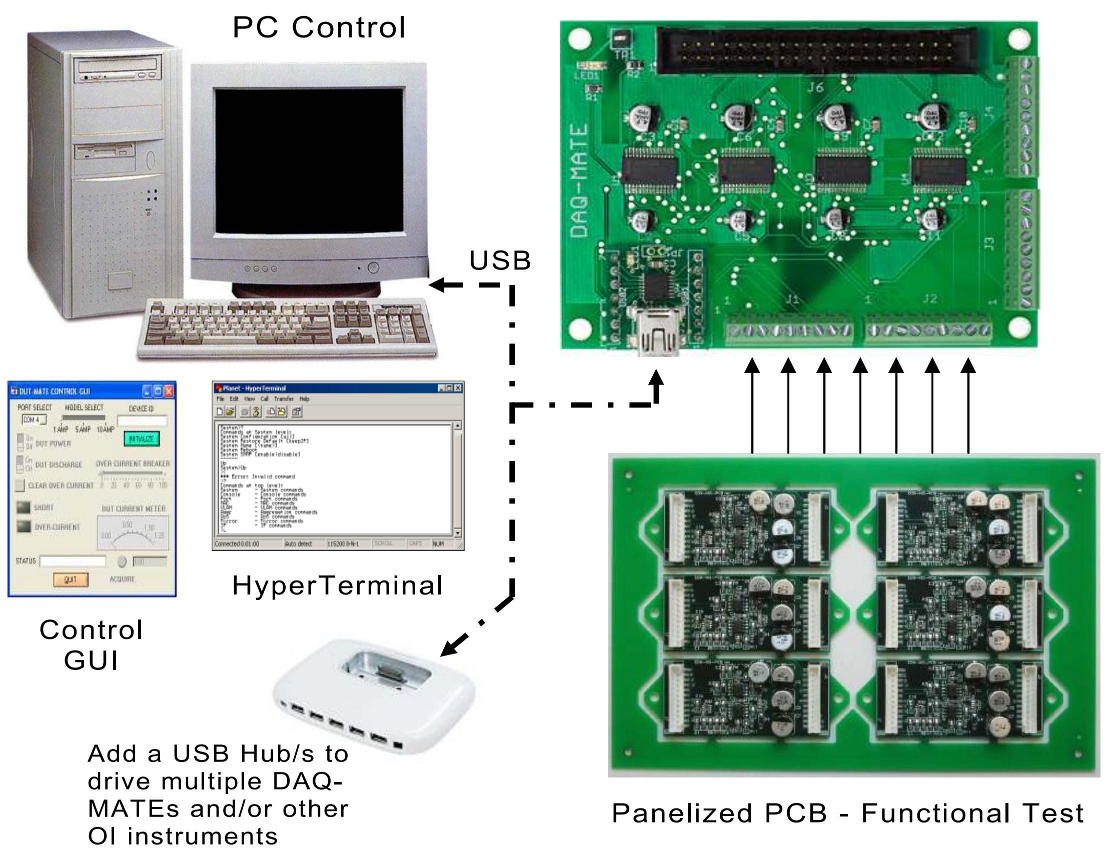
```
' Program: DAQ-MATE Demo
'
---[ Initialization ]----------------------------------------------------------
'
$large
$romstart = &H2000
$default Xram

Dim Daq_adc_word As Word
Dim Daq_adc_val As Single
Dim A_num, A_byte, A_cnt As Byte
Dim Daq_ch, Daq_adc_range, Daq_num, Daq_cnt, Daq_dev, Daq_cntl-byte 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 P1.0                     ' SPI-bus serial clock
Dout Alias P1.1                     ' SPI-bus serial data output
Din Alias P1.2                      ' SPI-bus serial data input
Adc0_cs Alias P1.3                  ' ADC port0 chip select
Adc1_cs Alias P1.4                  ' ADC port1 chip select
Adc2_cs Alias P1.5                  ' ADC port2 chip select
Adc2_cs Alias P1.6                  ' ADC port3 chip select

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 Daq_rd_adc(chk_adc_val As Single , Daq_ch As Byte , Daq_adc_range As
Byte)

---[ Main ]----------------------------------------------------------
' In the Main the Operator or Host, is prompted to enter a command.  The command is
parsed and then executed if valid.  Only three command examples are shown.

Set Sclk, Dout, Adc0_cs, Adc1_cs, Adc2_cs, Adc3_cs              ' Set to logic '1'

Do
  Input "Enter command " , S
  S = Ucase(s)
  A_resp = Left(s , 3)
  If A_resp = "CK_" Then
    A_resp = Mid(s , 4 , 2)
    Select Case A_resp

      Case "AR":               'Set ADC Range

        A_resp = Mid(s , 6 , 1)
        If A_resp = "?" Then
          If Daq_adc_range = Daq_adc_5v Then A_str = "0"
          If Daq_adc_range = Daq_adc_10v Then A_str = "1"
          If Daq_adc_range = Daq_adc_5v5v Then A_str = "2"
          If Daq_adc_range = Daq_adc_10v10v Then A_str = "3"
          Print "<" ; A_str ; ">"
          Print
        Else
          A_num = Val(a_resp)
          If A_num < 0 Or A_num > 3 Then
            Call Print_oor                        ' out-of-range
          Else
            If A_num = 0 Then Daq_adc_range = Daq_adc_5v
            If A_num = 1 Then Daq_adc_range = Daq_adc_10v
            If A_num = 2 Then Daq_adc_range = Daq_adc_5v5v
            If A_num = 3 Then Daq_adc_range = Daq_adc_10v10v
          End If
        End If

   Case "SC":               'Set ADC channel

        A_resp = Mid(s , 6 , 1)
        If A_resp = "?" Then
          A_str = Str(chk_ch)
          Print "<" ; A_str ; ">"
          Print
        Else
          A_num = Val(a_resp)
          If A_num < 0 Or A_num > 7 Then
            Call Print_oor                        ' out-of-range
          Else
            Daq_ch = A_num
          End If
        End If

   Case "RV":               ' read voltage

        A_resp = Mid(s , 6 , 1)
        If A_resp = "?" Then
          Call Daq_rd_adc(chk_adc_val , Daq_ch , Daq_adc_range)
          A_str = Str(chk_adc_val)
          Print "<" ; A_str ; ">"
          Print
        Else
          Call Print_ic          ' invalid command
        End If

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

'---[ Sub-Routines]----------------------------------------------------------
'
Sub Daq_rd_adc(chk_adc_val As Single , Daq_ch As Byte , Daq_adc_range As Byte)
   Daq_adc_val = &H0000
        ' Select range
    Daq_num_2 = Daq_ch
    If Daq_ch < 8 Then
      Daq_dev = 0
    Elseif Daq_ch => 8 And Daq_ch <= 15 Then
      Daq_num = Daq_ch - 8, Daq_dev = 1
    Elseif Daq_ch => 16 And Daq_ch <= 23 Then
      Daq_num = Daq_ch - 16, Daq_dev = 2
    Elseif Daq_ch => 24 And Daq_ch <= 31 Then
      Daq_num = Daq_ch - 24, Daq_dev = 3
    End If
        ' Select analog channel
    Daq_ch = Daq_ch_buf(daq_num)
    Daq_cntl_byte = Daq_range || Daq_ch
    Reset Sclk
    Delay
        ' take X measurements
   For Daq_cnt = 1 To Daq_m_cnts
    Daq_adc_word = &H0000, Daq_num = 7, Daq_num_2 = 11
         ' Select device
    If Daq_dev = 0 Then Reset Adc_cs0
    If Daq_dev = 1 Then Reset Adc_cs1
    If Daq_dev = 2 Then Reset Adc_cs2
    If Daq_dev = 3 Then Reset Adc_cs3
    For Daq_cnt_2 = 1 To 24
      If Daq_cnt_2 < 9 Then
         ' Send control byte
        Dout = Daq_cntl_byte.chk_num
        Set Sclk
        Reset Sclk
        Decr Daq_num
      Elseif Daq_cnt_2 > 12 Then
         ' Get ADC value
        Set Sclk
        Reset Sclk
        Daq_adc_word.chk_num_2 = Din
        Decr Daq_num_2
      Else
         ' dummy clocks
        Set Sclk
        Reset Sclk
      End If
    Next Daq_num
         ' disable devices
    Set Adc0_cs, Adc1_cs, Adc2_cs, Adc3_cs
         ' collect results
    Daq_adc_val = Daq_adc_val + Daq_adc_word
    Waitms 1
   Next Daq_cnt
         ' compute average
   Daq_adc_val = Daq_adc_val / Daq_m_cnts

End Sub
```

3.2 PC Control

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

Controlling the DAQ-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 DAQ-MATE side, the PIC controller processes the serial commands and then drives the DAQ-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.DAQ-MATE.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 of a PC-driven configuration. As shown, the DAQ-MATE is used to perform a quick "Acceptance" test by collecting analog measurements from a full panel of PCBs.

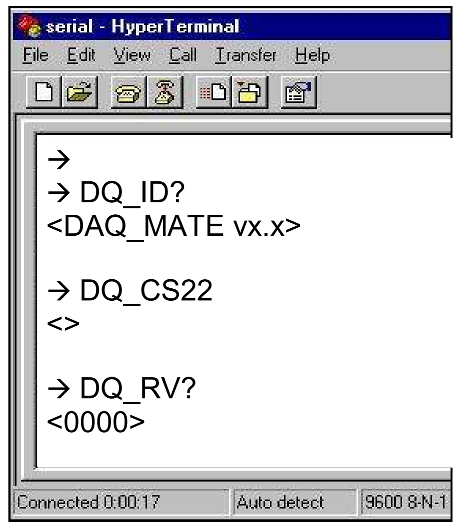


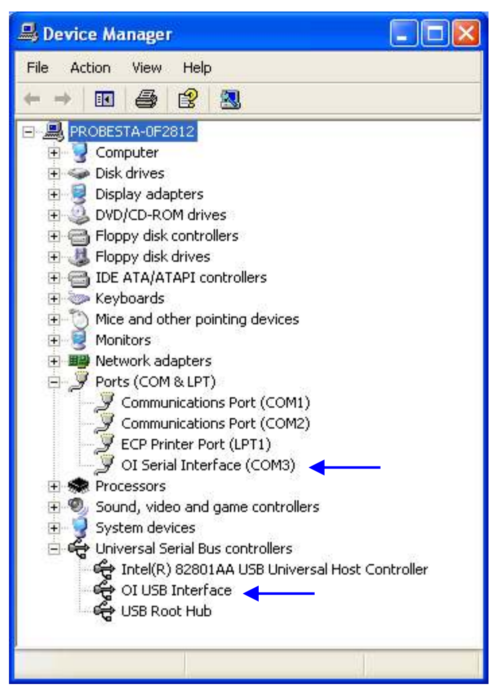
3.2.1 PC Programming

The starting point for developing code to control the DAQ-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 DAQ-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).

3.2.1.1 HyperTerminal

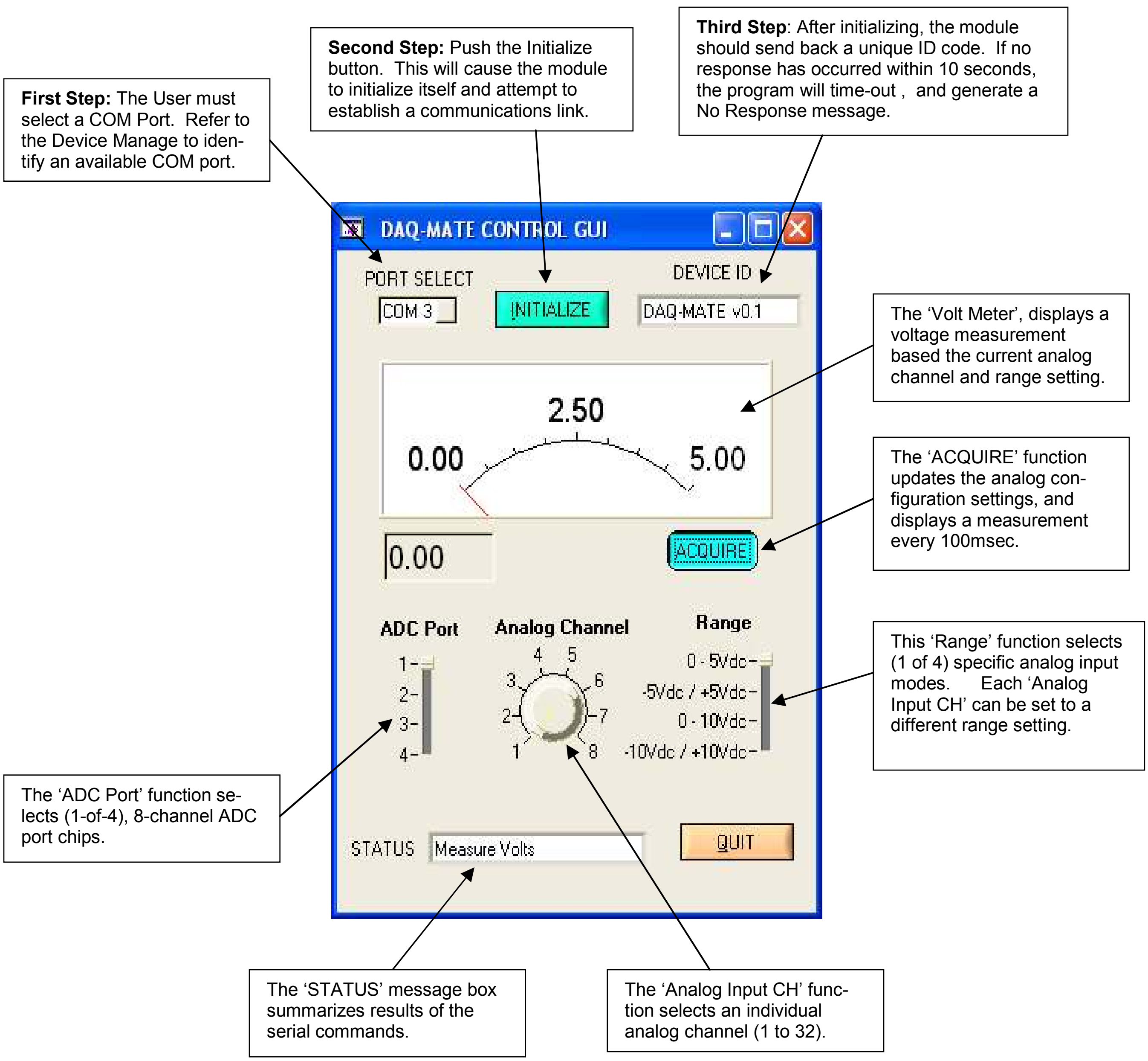
HyperTerminal is a serial communications program that comes with the Windows OS and is located in the Accessories folder. Use the USB cable to connect the PC to the DAQ-MATE. Run HyperTerminal and configure the settings for 19200 bps, 8 data bits, no parity, 1 stop bit and no flow control. Select the 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.





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 DAQ-MATE. Download the panel from our website at www.check-mate.com, click on downloads and select "DAQ-Matexxx.exe".



3.2.1.3 PC Programming Example

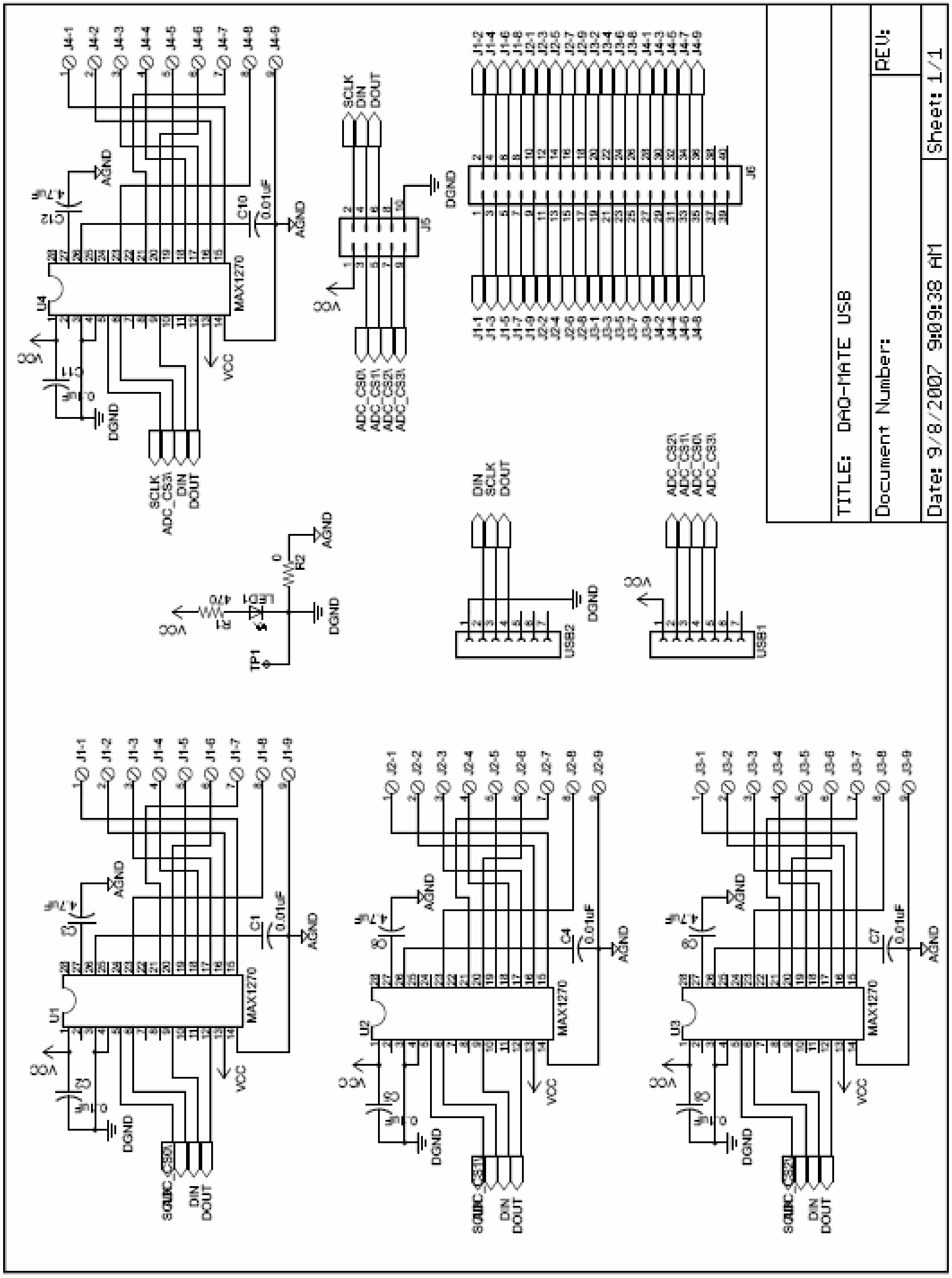
```
// DAQ-MATE programming example in 'C'
//
// The following program tests an array of 16 multi-color LED's. Each LED
// can produce Red, Green and Blue light. The objective of test is to
// measure each LED for color & intensity, and save the results to a file.
// Sixteen sensors are used to convert the LED light to an analog voltage.
// Each LED sensor provides two outputs (a 0-5Vdc for color and a 0-10Vdc
// for intensity). The outputs of the LED sensors are connected to the DAQ-
// MATE's 32 analog channels. The DIO-MATE (a 48-bit digital I/O module)
// is used to turn-On/Off the individual LEDs. The test sequence involves
// selecting an analog channel, turning-On the LED, measuring color &
// intensity, and then save the results and repeat the cycle.

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

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

int stat, port=0, dq_port=0, io_port-0, a_byte = 0, True;
int a_cnt = 0, a_ch = 0, idx = 0, first_shift, dio_bit[10] = 0, False;

char a_str[10], a_str_1[10], a_str_2[10];
char dio_byte[10], dir_byte[10];
char send_data[64], read_data[64];

char dq_get_id[] = "DQ_ID?";            // get module ID
char dq_master_clr[] = "DQ_MC";         // set master clear
char set_adc_ch[] = "DQ_SC";            // set ADC channel
char set_adc_range[] = "DQ_AR";         // set ADC range
char get_adc_volts[] = "DQ_RV?";        // read voltage

char set_dio_dir[] = "IO_PD";           // set DIO port direction
char set_dio_port[] = "IO_PN";          // set DIO port number
char set_dio_byte[] = "IO_PB";          // set DIO byte, write
char io_get_id[] = "IO_ID?";            // get module ID
char io_master_clr[] = "IO_MC";         // set master clear

main()
{
    dq_port=OpenComPort(1,256,64);   // Open COM 1 - DAQ-MATE
    io_port=OpenComPort(1,256,64);   // Open COM 2 - DIO-MATE
    for (a_cnt = 0; a_cnt <= 1; a_cnt++) {
        if (a_cnt == 0) port = dq_port;
        if (a_cnt == 0) port = dq_port;

        if ((stat = SetPortCharacteristics(port,BAUD19200,PAR_EVEN,
            LENGTH_8,STOPBIT_1,PROT_NONNON)) != RS232ERR_NONE) {
            printf("Error #%d setting characteristics\n",stat);
            exit();
        }
        CdrvSetTimerResolution(port,1);   // 1 msec ticks
        SetTimeout(port,2000);            // 2000 ticks = 2 sec time-out period
        FlushReceiveBuffer(port);         // clear receiver buffer
        FlushTransmitBuffer(port);        // clear transmit buffer

                                          // 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\n");
            exit();
         }
         if (strcmp("-> ", read_data)) {
            printf("Incorrect promt\n");
            exit();
         }                    // Master Clear
         If (a_cnt == 0) sprintf (send_data, "%s\r", dq_master_clr);
         If (a_cnt == 1) sprintf (send_data, "%s\r", io_master_clr);
         PutString(port,send_data);
    }
                        // Set DIO-MATE, port direction
    sprintf (a_str, "%s\r", "00000000");   // output direction
    for (a_cnt = 0; a_cnt <= 5; a_cnt++) {
        sprintf (send_data, "%s%d\r", set_dio_port, a_cnt);
        PutString(io_port,send_data);
        sprintf (send_data, "%s%s\r", set_dio_dir, a_str);
        PutString(io_port,send_data);
        sprintf (send_data, "%s%s\r", set_dio_byte, a_str);
        PutString(io_port,send_data);
    }
                        // Set DAQ-MATE, ADC range
    a_num = 0;
    for (a_cnt = 1; a_cnt <= 32; a_cnt++) {
        sprintf (send_data, "%s%02d\r", set_adc_ch, a_cnt);
        PutString(dq_port,send_data);
        if (a_num == 0) sprintf (a_str, "%s\r", "0");       // 0-5Vdc
        if (a_num == 1) sprintf (a_str, "%s\r", "2");       // 0-10Vdc
        sprintf (send_data, "%s%s\r", set_adc_range, a_str);
        PutString(dq_port,send_data);
        a_num = !a_num;
    }                   // Execute test sequence
    a_ch = 1;
    first_shift = False;
    FILE *fp;
    for (a_cnt = 1; a_cnt <= 16; a_cnt++) {
        idx = 0;
        do {   // Measure LED color & intensity and store to file.
            a_num = 0;
            Shift_dio();        // select LED
            sprintf (file_name, "%s","C:\\LEDTEST.TXT");
            fp=fopen(file_name, "a+");
            do {
                sprintf (send_data, "%s%02d\r", set_adc_ch, a_ch);
                PutString(dq_port,send_data);
                sprintf (send_data, "%s\r", get_adc_volts);
                PutString(dq_port,send_data);
                GetString(dq_port,sizeof(read_data),read_data);
                if (a_num == 0) sprintf(a_str_1, "%s", read_data); // get color
                if (a_num == 1) sprintf(a_str_2, "%s", read_data); // get intensity
                a_num++;
                a_ch++;
            } while (a_num < 2);
            sprintf(a_str, "LED# %d - %s%s\r\n", a_cnt, a_str_1, a_str_2);
            fprintf(fp, a_str);
            fclose(fp);
            idx++;
        } while (idx < 3);
    }
    printf ("Test Complete\r\n");
}
                // Shift a logic '1' through 48 bits
void Shift_dio(void) {

    int dio_bit_cnt, dio_bit_num, dio_idx, dio_port;
    char dio_bits[50];
    If (first_shift==False) {
        sprinf (dio_bits, "%s",
            "000000000000000000000000000000000000000000000001");
        first_shift = True;
    }
    else {
        for (dio_bit_cnt = 47; dio_bit_cnt >=1; dio_bit_cnt--) {
            if (dio_bits[dio_bit_cnt] == '1') {
                dio_bits[dio_bit_cnt] = '0';
                dio_bits[dio_bit_cnt-1] = '1';
                break;
            }
        }
    }
    dio_bit_num = 47;
    for (dio_port = 0; dio_port <= 5; dio_port++) {     // extract DIO byte
        for (dio_idx = 7; dio_idx >= 0; dio_idx--) {
            dio_byte[dio_idx] = dio_bits[dio_bit_num];
            dio_bit_num--;
        }
        dio_byte[8] = '\0';
        sprintf (send_data, "%s%d\r", set_dio_port, dio_port); // select DIO port
        PutString(io_port,send_data);
        sprintf (send_data, "%s%s\r", set_dio_byte, dio_byte); // write DIO byte
        PutString(io_port,send_data);
    }
}
```

Appendix A. Serial Command Set

To facilitate remote control for the DAQ-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 DAQ-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 **'DQ_',** 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 DAQ-MATE will return either a **'<>'**, or a bracketed result (i.e. **'<2108>'**. If the DAQ-MATE receives a carriage-return or line-feed alone (without a command), then a **'→'** is returned (this response is a "prompt" to signal the DAQ-MATE is ready). If the DAQ-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
DQ_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).
DQ_BR?	Get baud rate code	<n>	Get current baud rate code (-n- is the return code 0 to 3).
DQ_ID?	Get module ID	<DAQ-MATE vx.x>	Get current identification and version number.
DQ_MR	Master Reset	<>	Reset & initialize the module
DQ_WC	Write configuration	<>	Store current instrument settings in EEPROM. Save settings related to the ADC, DAC and DIO hardware.
DQ_RC	Recall configuration	<>	Retrieve stored instrument settings
DQ_SCnn	Set ADC channel	<>	Select a ADC voltage channel. The -nn- represents a channel number from 01 to 32.
DQ_SC?	Get ADC channel	<n>	Get the current ADC voltage channel.
DQ_ARn	Set ADC range	<>	Set the ADC range code (-n- is 0 = 0-5Vdc, 1 = 0-10Vdc, 2 = ±5Vdc, and 3 = ±10Vdc).
DQ_AR?	Get ADC range	<n>	Get the current ADC range code.
DQ_RV?	Get voltage measurement	<nnnn>	Get a voltage measurement based on the current ADC channel and range selection. The measurement contains 4 ASCII bytes representing a 12-bit decimal value (0-4095).
DQ_CS?	Scan all ADC ch's	<ch1,ch2,...,ch32>	Measure and output 32 ADC channels. Each channel contains 4 ASCII bytes representing a 12-bit decimal value (0-4095). A comma ',' separates each channel

Appendix B. Schematic



Appendix C. Mechanical Dimensions