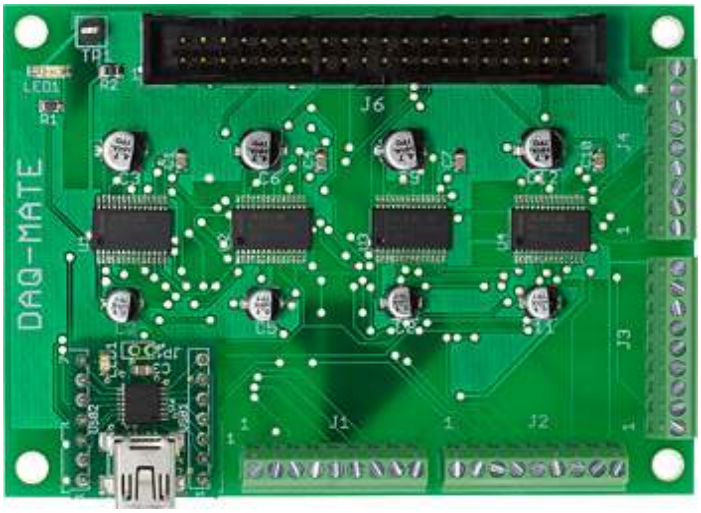
# DAQ-MATE

## 32-CH DATA ACQUISITION MODULE




USB UNIVERSAL SERIAL BUS

# USER'S MANAUAL

overton INSTRUMENTS

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 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.

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. Unless 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 5
1.3 Specifications 6

2.0 I/O DESCRIPTION 7
2.1 Hardware Details 7
2.2 Board Layout 8
2.3 Connections 9

3.0 OPERATION 11
3.1 Embedded Control 11
3.1.1 Embedded Configuration 12
3.1.2 Embedded Programming 13
3.1.3 Embedded Program Example 14
3.2 PC Control 15
3.2.1 PC Programming 16
3.2.1.1 HyperTerminal 16
3.2.1.2 Virtual Instrument Panel 17
3.2.1.3 PC Programming Example 18

APPENDIX A. SERIAL COMMAND SET 19

APPENDIX B. SCHEMATIC 20

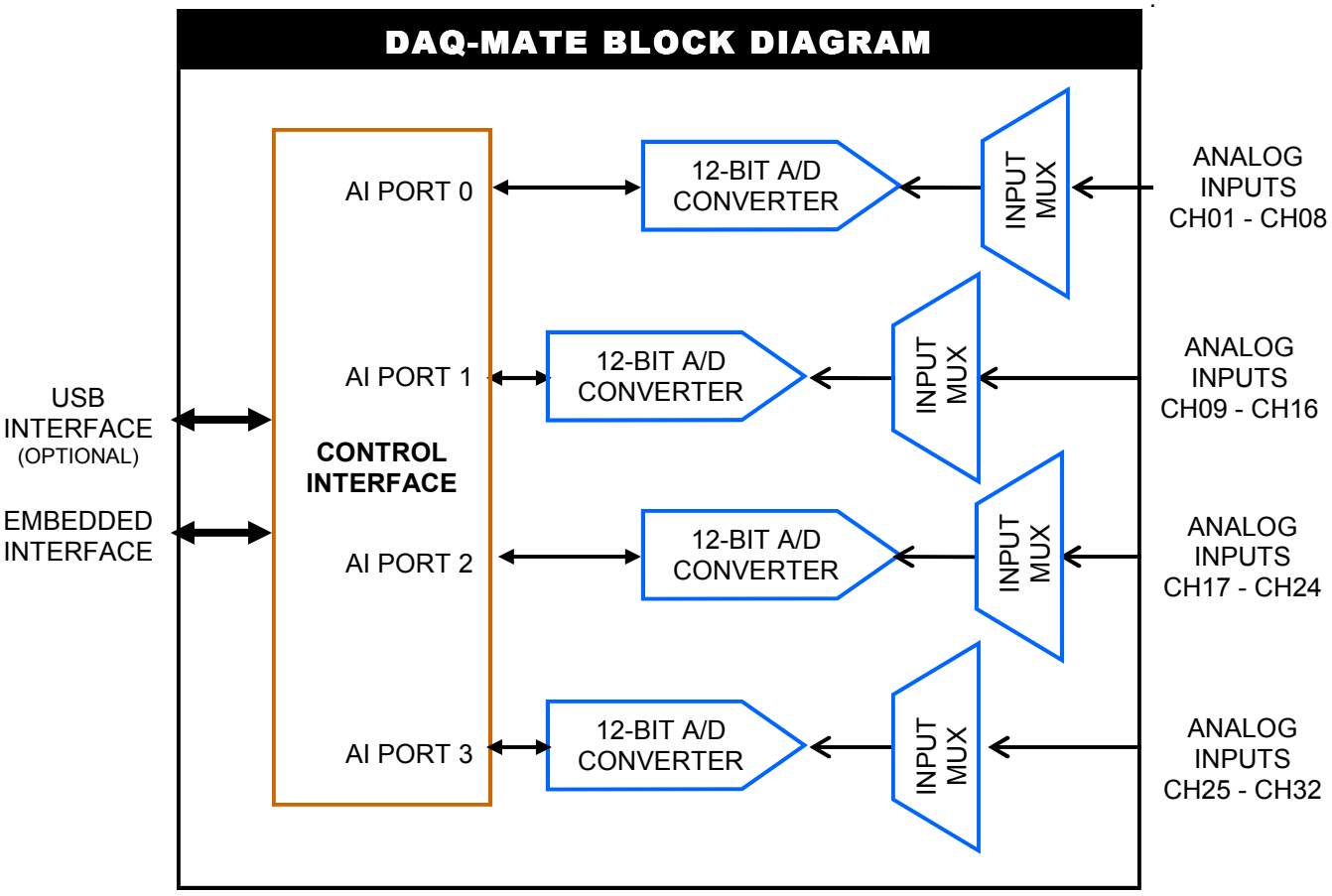
APPENDIX C. MECHANICAL DIMENSIONS 21

# 1. Introduction

## 1.1 Overview

The DAQ-MATE offers an impressive 32-channels of analog data acquisition, including 12-bit resolution (and a sample rate of 100ksps). In addition the channels can be independently programmed for either single-ended or differential mode, and operate in 4 different input ranges (0-5V, 0-10V, ±5 and ±10V).

The DAQ-MATE is offered in 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<br>• Quickly measure a wide array of analog signals. Each analog channel can be independently programmed for 4 different range modes<br>• Great for embedded solutions - place inside mechanical test fixtures, instrument boxes or rack-mount enclosures | • Burn-In<br>• Engineering<br>• Depot Repair<br>• Production Test<br>• QA/QC Quality Control<br>• OEM Test Instruments | • 32 12-bit A/D channels<br>• 100Ksps sample rate<br>• Programmable Single/ Differential modes<br>• 4 Programmable Input Ranges (0-5V, 0-10V, ±5V and ±10V)<br>• USB interface or embedded control<br>• Low Cost<br>• 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 SE / 16 Differential |
| Input Ranges | 0-5V, 0-10V, ±5V, ±10V programmable |
| Resolution / SR | 12-bit / 100ksps |
| Nonlinearity | ±1LSB, no missing codes |
| **Input Control** | |
| Embedded | Oi-Bus interface |
| USB Interface | Optional USB module |
| **General** | |
| Power Supply | +5VDC±10%@30mA |
| Operating Temp | 0-50ºC |
| Dimensions | 3.0" x 3.5" |

# 2. I/O Description

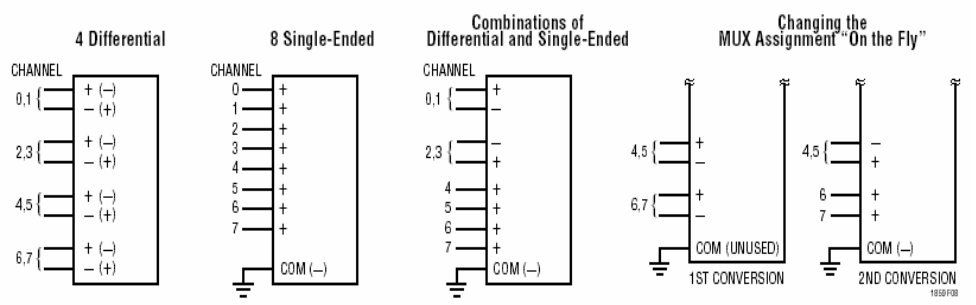
## 2.1 Hardware Details

Access to DAQ-MATE hardware is made possible through a convenient set of screw terminal connections (J1 - J4), and J6 (which consolidates all signals into a single 40-pin header).

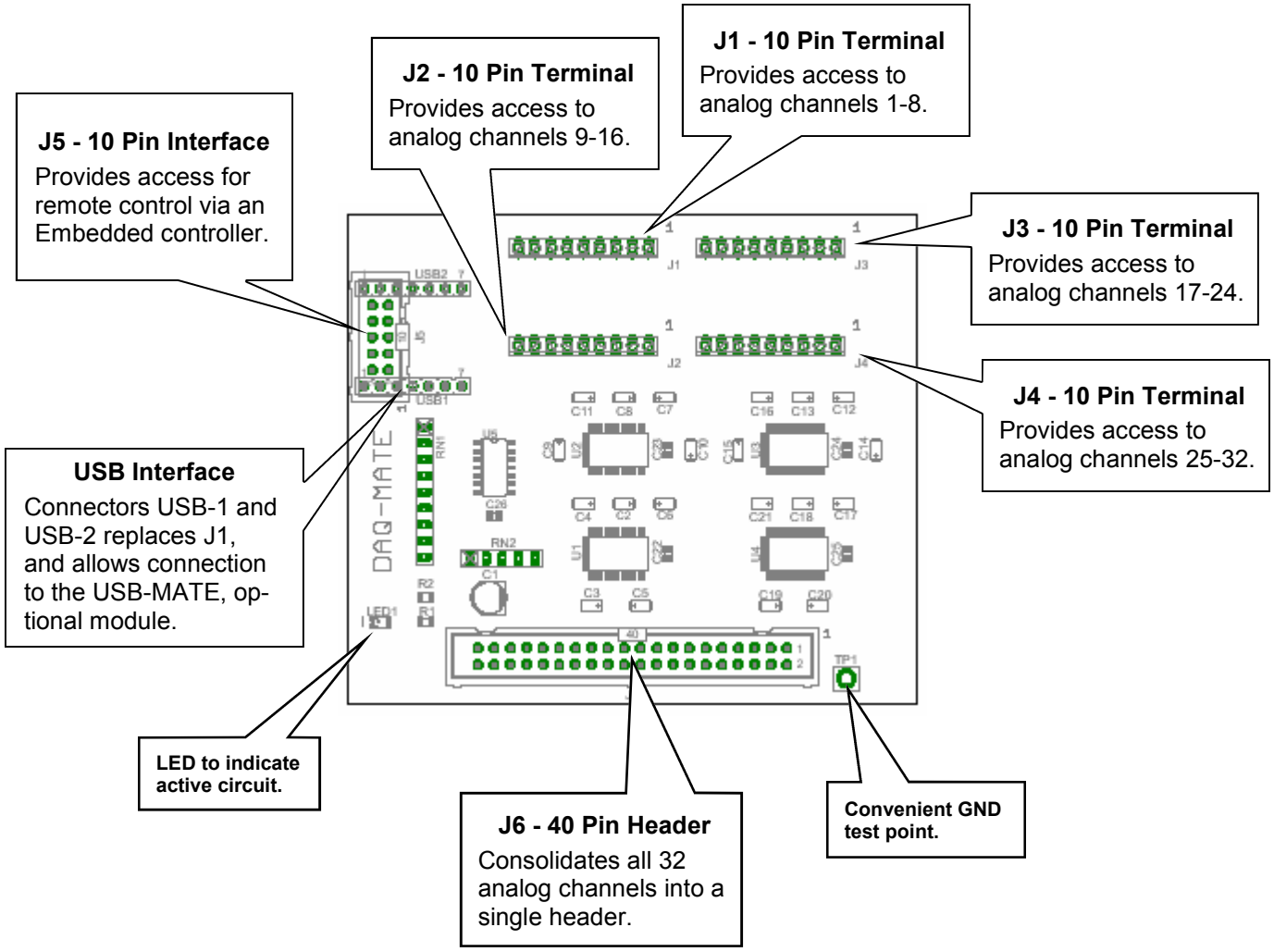
The analog inputs (or channels) can be programmed for any combination of single-ended or differential operation. The diagram below shows examples of various configurations. You will also note the polarity of connections related to differential operation can be transposed as well. Each channel can be programmed for anyone of 4 different range modes (i.e., 0-5V, ±5, 0-10V and ±10V). Keep in mind, the circuit provides ±25V protection on each channel.

External control of the DAQ-MATE can be provided by a embedded controller (such as the Micro-MATE), or with a PC. Embedded control is supported by J1 (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 J5 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 DAQ-MATE. The USB-MATE is designed to interpret a set of ASCII commands sent from the PC, and then perform various DAQ-MATE functions. For more information for the DAQ-MATE command set, go to Appendix A. To support embedded applications, a complete driver for the DAQ-MATE is provided in TES-MATE (or Test Executive Suite).

After power is applied to the DAQ-MATE, the analog inputs are configured for single-ended (0-5V range).



## 2.2 Board Layout



## 2.3 Connections

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

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

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

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

## 2.2 Connections cont.

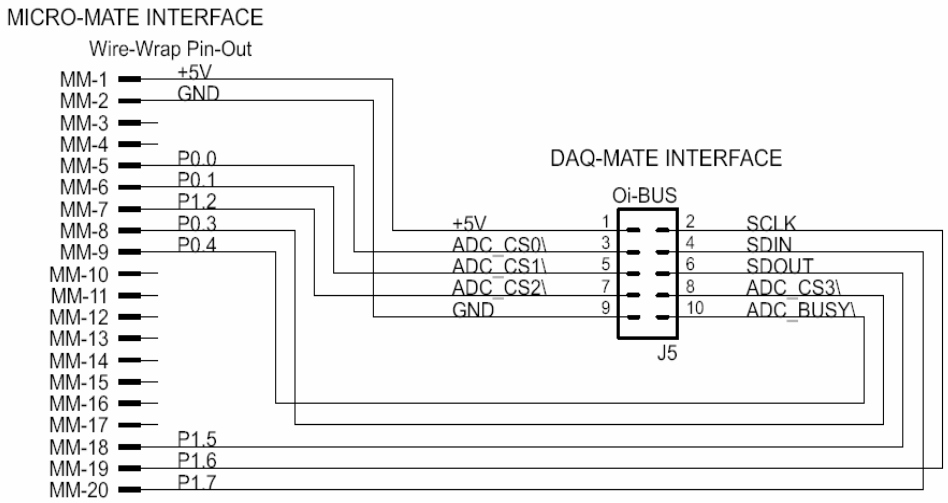
| J5 | | | |
|---|---|---|---|
| **Pin** | **Name** | **Dir.** | **Description** |
| 1 | VCC | | A regulated +5Vdc output for external use. Current limited to roughly 100mA. |
| 2 | SCLK | I | Part of a 3-wire SPI-Bus, SCLK synchronizes the serial data transfer for the DIN and DOUT signals. |
| 3 | ADC_CS0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

# 3. Operation

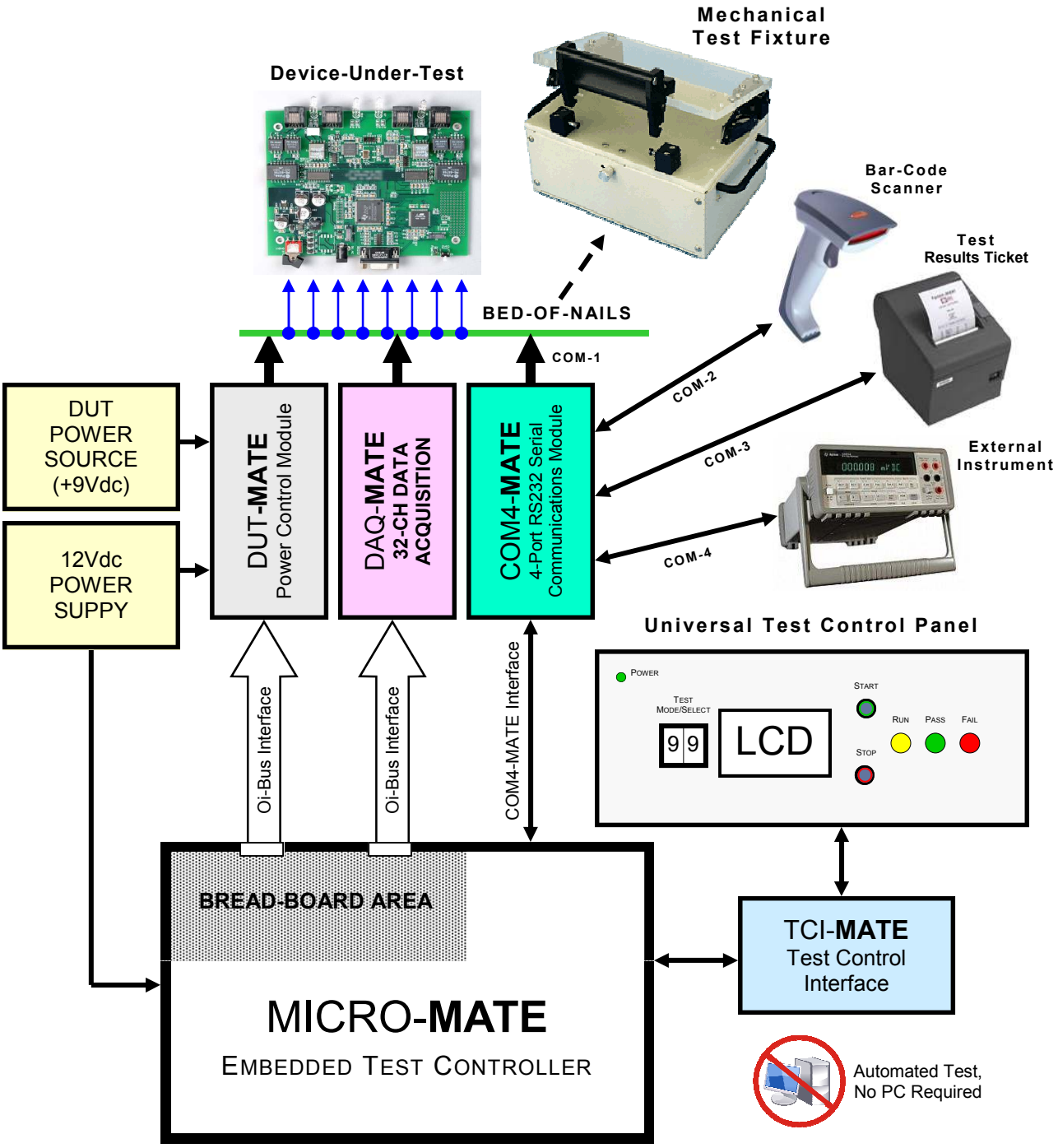
## 3.1 Embedded Control

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 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 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 Linear Technology (part number LTC1857). 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 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 (P1.7 & P1.6 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 Micro-MATE receives "high level" serial commands from a host PC, parses them and then executes accordingly. When (for example), the "DQ_RC17S01" command is entered, the program selects analog channel number 17 ('S' for single-ended, '0' for +/- polarity, and '1' for 5V range. After the command is deciphered, the program call's subroutine "Daq_rd_adc(chk_adc , Daq_ch , Daq_mode, Daq_pol, Daq_range)", which causes the DAQ-MATE to take an analog measurement and return the results in a 3 character hexadecimal "ASCII" string.

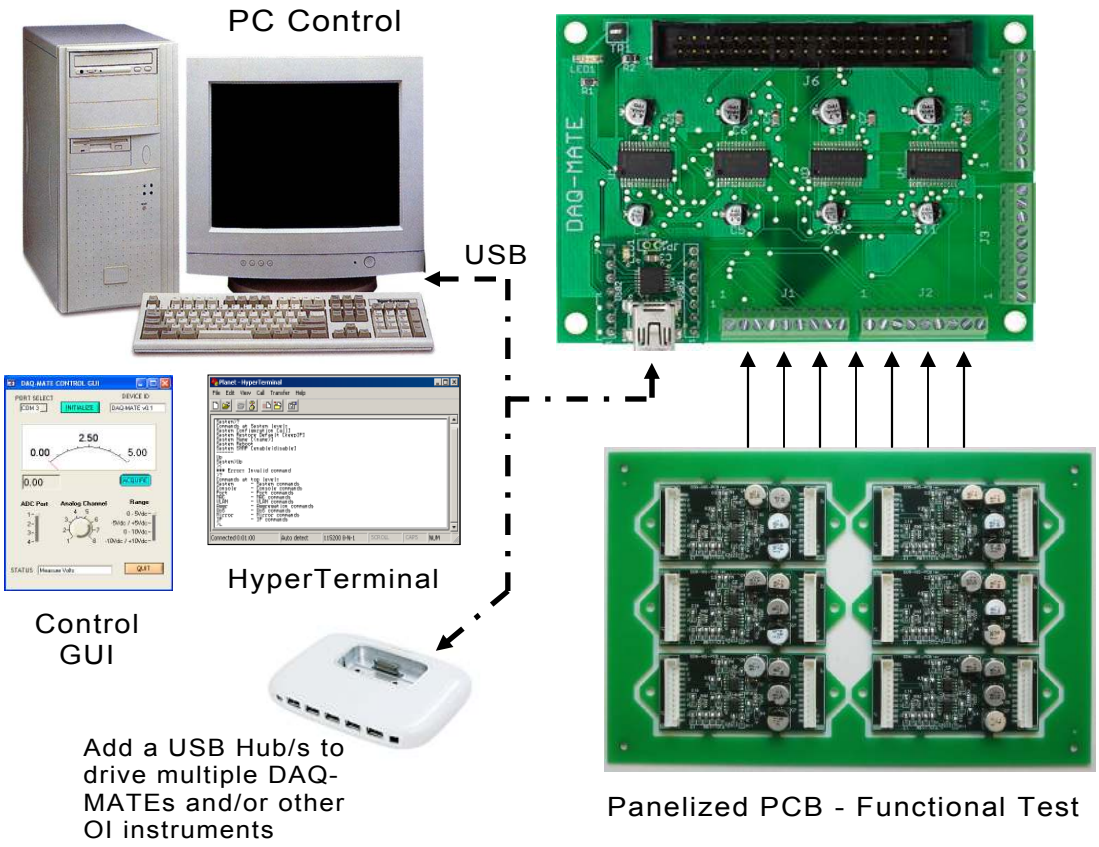
Independent of the microcontroller hardware or programming language you choose, the program sequence described above will likely resemble the way you implement your DAQ-MATE application. For this reason, we suggest that you go to our website and download the "DAQ-MATE.zip" file. In the Documents folder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

## 3.2 PC Control

For those 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 hardware 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.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 DAQ-MATE is used to perform a quick "Acceptance" test by collecting analog measurements from a full panel of PCBs.



## 3.1.3 Embedded Program Example

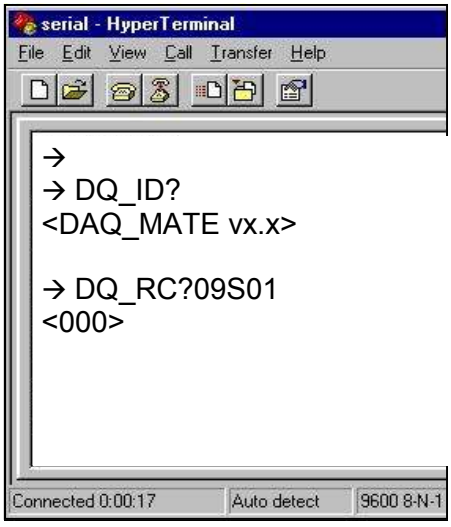
```
' Program: DAQ-MATE Demo
'
---[ Initialization ]----------------------------------------------------
'
$large
$romstart = &H2000
$default Xram

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 P1.7                    ' SPI-bus serial clock
Dout Alias P1.6                    ' SPI-bus serial data output
Din Alias P1.5                     ' SPI-bus serial data input
Daq _adc_0cs Alias P0.0            ' ADC port0 chip select, active low
Daq _adc_1cs Alias P0.1            ' ADC port1 chip select, active low
Daq _adc_2cs Alias P0.2            ' ADC port2 chip select, active low
Daq _adc_3cs Alias P0.3            ' ADC port3 chip select, active low
Daq_busy Alias P0.4                ' ADC busy, active low input

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(daq_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 two command examples are shown.
' Set to logic '1'
Set Sclk, Dout, Daq _adc_0cs, Daq _adc_1cs, Daq _adc_2cs, Daq _adc_3cs Do
  Print
  Err_trap = False
  Input "-> " , S Noecho
  S = Ucase(s)
  A_num = Len(s)
  If A_num > 0 Then
    A_resp = Left(s , 3)
    If A_resp = "DQ_" Then
      A_resp = Mid(s , 4 , 2)
      Select Case A_resp
        Case "CC":                 'Configure ADC channel
          A_resp = Mid(s , 6 , 1)
          If A_resp = "?" Then
            Print "<" ; Daq_conf_code ; ">"
          Else
            A_resp = Mid(s , 6 , 2)
            A_ch = Val(a_resp)
            If A_ch > 32 Then Err_trap = True
            A_char = Mid(s , 8 , 1)
            If A_char <> "D" And A_char <> "S" Then
              Err_trap = True
            Else
              If A_char = "D" Then Daq_mux_mode = 0
              If A_char = "S" Then Daq_mux_mode = 1
            End If
            A_char = Mid(s , 9 , 1)
            If A_char <> "0" And A_char <> "1" Then
              Err_trap = True
            Else
              If A_char = "0" Then Daq_mux_pol = 0
              If A_char = "1" Then Daq_mux_pol = 1
            End If
            If A_ch > 4 And A_ch < 9 Then
              If Daq_mux_mode = 0 Then Err_trap = True
            Elseif A_ch > 12 And A_ch < 17 Then
              If Daq_mux_mode = 0 Then Err_trap = True
            Elseif A_ch > 20 And A_ch < 25 Then
              If Daq_mux_mode = 0 Then Err_trap = True
            Elseif A_ch > 28 And A_ch < 33 Then
              If Daq_mux_mode = 0 Then Err_trap = True
            End If
            A_char = Mid(s , 10 , 1)
            A_num = Val(a_char)
            If A_num < 1 Or A_num > 4 Then Err_trap = True
            If Err_trap = False Then
              If A_num = 1 Then Daq_range = Daq_adc_5v
              If A_num = 2 Then Daq_range = Daq_adc_5v5v
              If A_num = 3 Then Daq_range = Daq_adc_10v
              If A_num = 4 Then Daq_range = Daq_adc_10v10v
              If Daq_mux_mode = 1 Then
                Daq_ch = Daq_ch_buf(a_ch)      ' set single-ended ch
              Else
                Daq_ch = Daq_ch_buf_d(a_ch)    ' set differential-ended ch
                Daq_ch.6 = Daq_mux_pol         ' set polarity (+/-)
              End If
              Daq_conf_code = Mid(s , 6 , 5)   ' set configuration code
              Call Print_ok
            Else
              Call Print_oor
            End If
          End If
        Case "RV":                 ' read voltage
          A_resp = Mid(s , 6 , 1)
          If A_resp = "?" Then
            Call Daq_rd_adc(daq_word , Daq_ch , Daq_range)
            If Daq_word > 4095 Then
              Call Print_oor
            Else
              A_str = Str(daq_word)
              Print "<" ; A_str ; ">"
            End If
          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
  End If
Loop
End

'---[ Sub-Routines]-------------------------------------------------------
'
    Daq_val = &H0000
    Daq_num_2 = Daq_ch             ' Select analog channel
    If Daq_ch < 9 Then
      Daq_dev = 0
    Elseif Daq_ch => 9 And Daq_ch <= 16 Then
      Daq_num = Daq_ch - 8
      Daq_dev = 1
    Elseif Daq_ch => 17 And Daq_ch <= 24 Then
      Daq_num = Daq_ch - 16
      Daq_dev = 2
    Elseif Daq_ch => 25 And Daq_ch <= 32 Then
      Daq_num = Daq_ch - 24
      Daq_dev = 3
    End If
    Daq_ch = Daq_ch_buf(daq_num)
    Daq_cntl_byte = Daq_range Or Daq_ch
    While Daq_busy = 0                          ' check busy flag
    Wend
    Reset Sclk
    Delay
    For Daq_cnt = 1 To Daq_m_cnts               ' take X measurements
      Daq_word = 0
      If Daq_dev = 0 Then Reset Daq_adc_cs0     ' assert low using alias port pin
      If Daq_dev = 1 Then Reset Daq_adc_cs1
      If Daq_dev = 2 Then Reset Daq_adc_cs2
      If Daq_dev = 3 Then Reset Daq_adc_cs3
      For Daq_cnt = 15 Downto 0
        If Daq_cnt >= 8 Then                    ' transceive serial data
          Daq_num = Daq_cnt - 8
          Dout = Daq_cntl_byte.daq_num
        End If
        Set Sclk
        Delay
        Daq_word = Din
        Delay
        Reset Sclk
      Next Daq_cnt
      Set Daq_adc_cs0                           ' assert high using alias port pin
      Set Daq_adc_cs1
      Set Daq_adc_cs2
      Set Daq_adc_cs3
    Next Daq_cnt
    Daq_val = Daq_val / Daq_m_cnts              ' compute average
    Daq_ch = Daq_num_2
End Sub
```

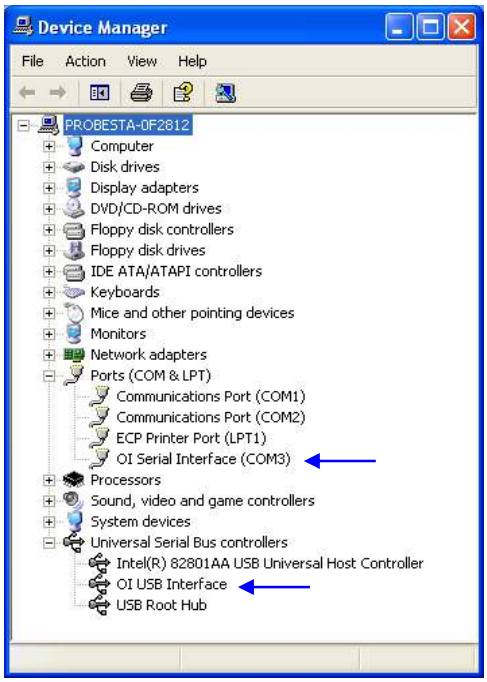
### 3.2.1 PC Programming

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

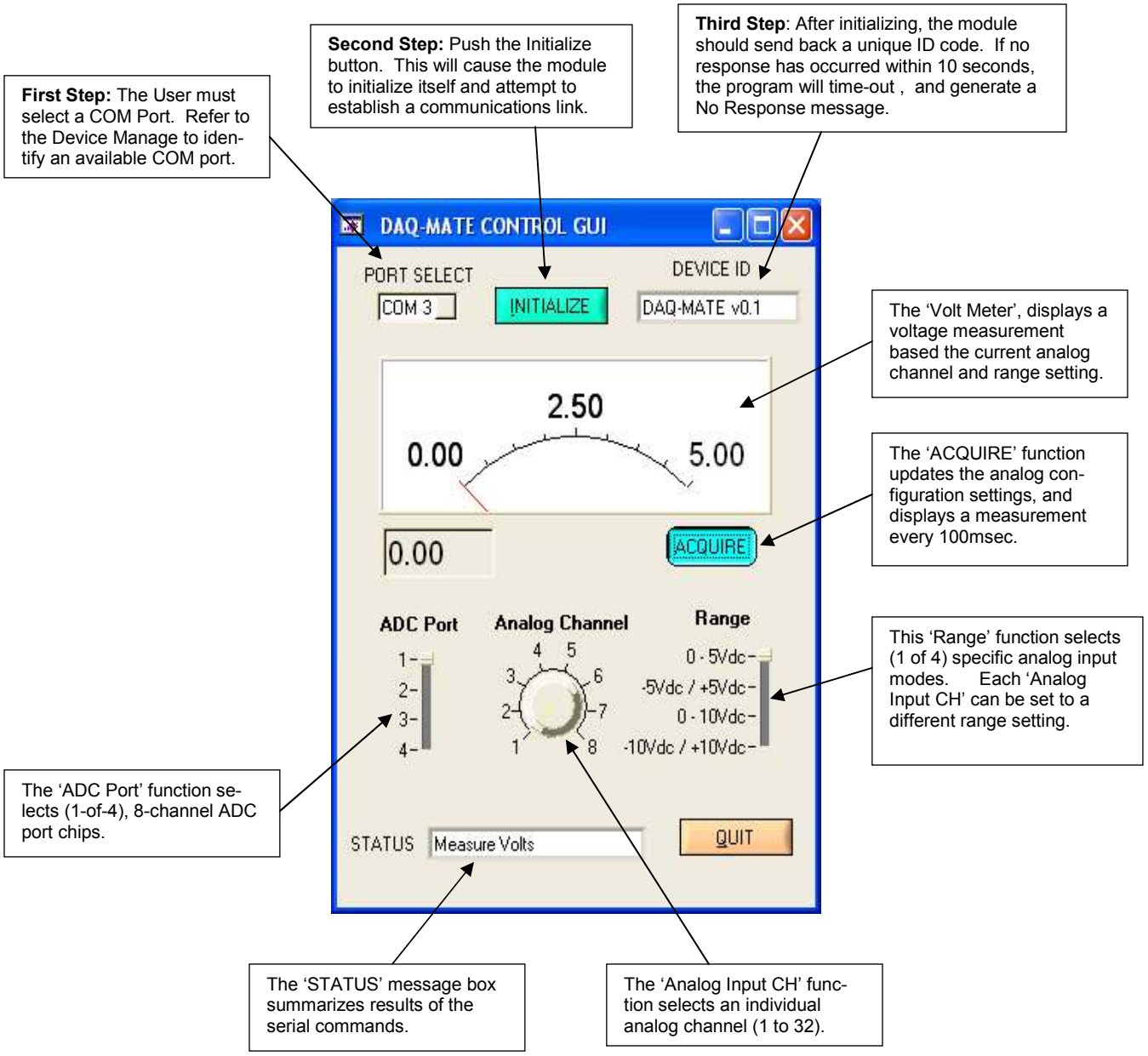
HyperTerminal is a serial communications program that comes with the Windows OS and is located in the Accessories folder. Use the USB cable to connect the PC to the DAQ-MATE. Run HyperTerminal and configure the settings for 19200 bps, 8 data bits, no parity, 1 stop bit and no flow control. Select the 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 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 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 <stddio.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_CC";                    // set ADC channel
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);
    }
                                // Configure 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);
    }
                                // 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 {
                if (a_num == 0) sprintf (a_str, "%s\r", "S01");// 0-5Vdc, single-ended
                if (a_num == 1) sprintf (a_str, "%s\r", "S03");// 0-10Vdc, single-ended
                sprintf (send_data, "%s%02d%s\r", set_adc_ch, a_cnt, a_str);
                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

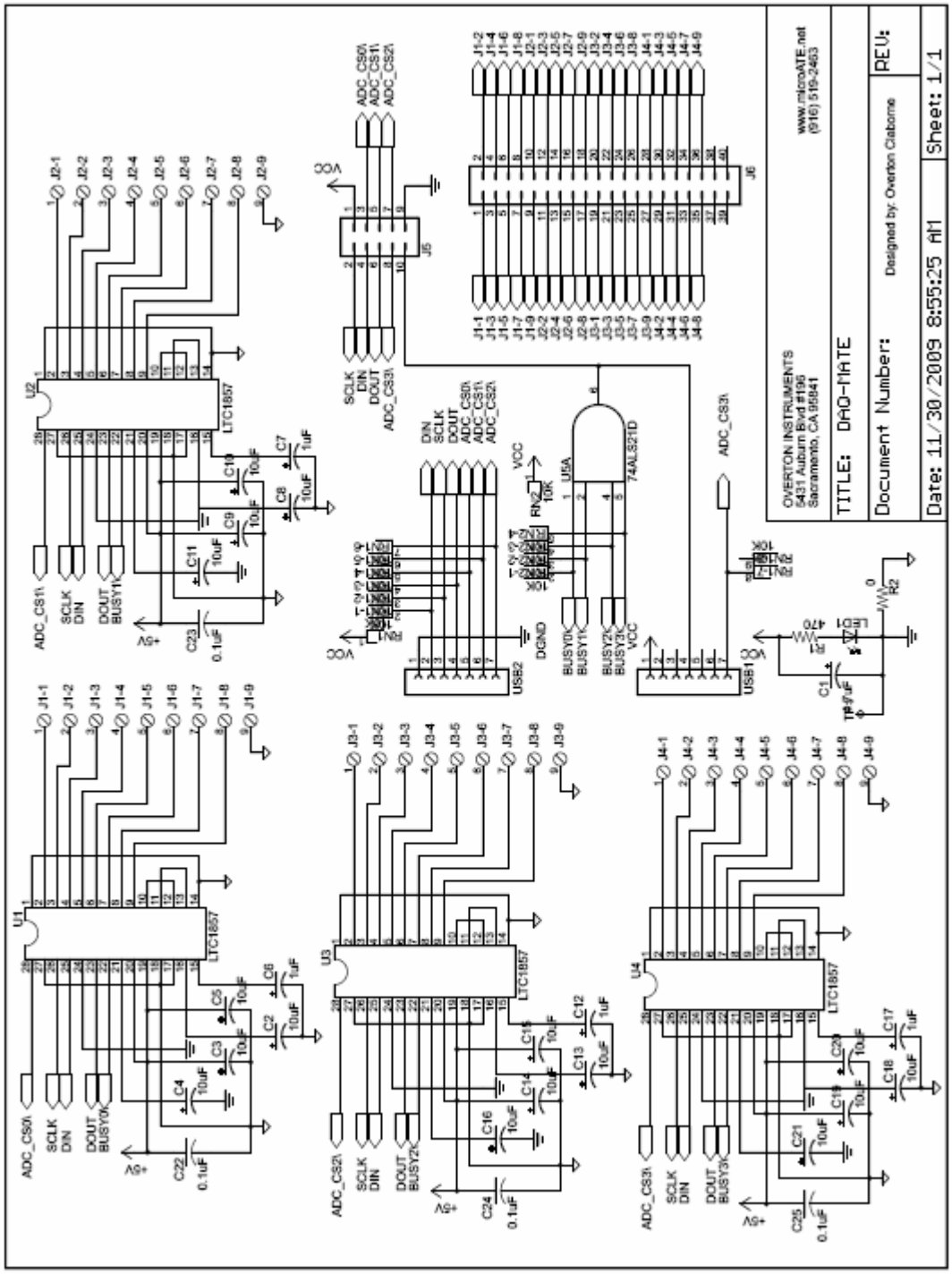
To facilitate remote control for the DAQ-MATE, a USB interface is required. When connected to a host PC, the USB connection appears as a "Virtual Com Port", which establishes a serial data communications link between the two. The default protocol is 19200 baud rate, no parity, 1 stop bit and no flow 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. **'<F74>'**. 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_SSccr** | Set single-ended configuration | <> | Set single-ended channel configuration.<br>cc = ADC channel number (01 to 32)<br>r = ADC range (1 = +5V, 2 = ±5V, 3 = 10V, 4 = ±10V)<br>If cc=00, then all channels are set to 'r' (same range) |
| **DQ_SDccpr** | Set differential configuration | <> | Set differential channel configuration.<br>cc = ADC channel number (01 to 16)<br>p = ADC polarity (0 = +, 1 = -)<br>r = ADC range (1 = +5V, 2 = ±5V, 3 = 10V, 4 = ±10V)<br>If cc=00, then all channels are set to 'p' and 'r' (same polarity and range) |

# Appendix A. Serial Command Set cont.

| Command | Function | Response | Description |
|---|---|---|---|
| **DQ_RV?ccmprf** | Configure channel and get voltage measurement | <n> | Configure and read a single ADC channel.<br>cc = ADC channel number (01 to 32 SE or 01 to 16 Diff)<br>m = ADC mode ("S" = Single-ended, "D" = Differential)<br>p = ADC polarity (0 = +/-, 1 = -/+)<br>r = ADC range (1 = +5V, 2 = ±5V, 3 = 10V, 4 = ±10V)<br>f = Data format ("D" = Decimal, "H" = Hexa-decimal<br>The voltage measurement contains a series of ASCII bytes representing a 12-bit value which is expressed in counts (0-4095 or 000-FFF). |
| **DQ_AS?nf** | Scan all channels and return voltage measurements | <CH1mpr=n, CH2mpr=n,.., CH32mpr=n> | Auto scan all ADC channels and return readings based-on presets from channel configuration commands 'CK_SS' and 'CK_SD'. The measured data is returned in one of two forms, Basic or Extended. In Extended each channel is identified (including the mode, polarity and range codes). The voltage measurements are a series of ASCII bytes representing a 12-bit value that is expressed in counts (0-4095 decimal or 000 to FFF hex). A comma is used to separate each channel reading. In Basic mode, the measured data is provided alone. When n=0 (Basic mode is active), and n=1 (Extended mode is active). When f="D" (decimal data), f="H" (hexadecimal data). |
| **DQ_MSnnn** | Set ADC measurement sample count | <> | Analog inputs can be averaged with a measurement sample count. The sample count value -nnn-, must be a 3 byte ASCII decimal from "000" to "255". |
| **DQ_MS?** | Get ADC measurement sample count | <n> | Get the current ADC sample count. |

# Appendix B. Schematic



# Appendix C. Mechanical Dimensions