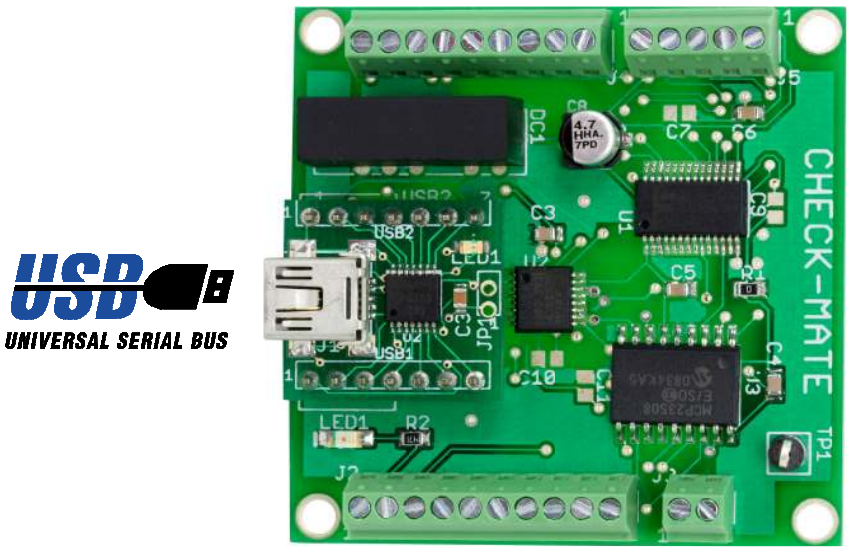
EMBEDDED TEST SOLUTIONS

# DIO-MATE

## 48-BIT DIGITAL I/O MODULE



## USER'S MANAUAL

overton INSTRUMENTS

Overton Instruments, Inc
5431 Auburn Blvd. #160
Sacramento, CA 95841
www.microATE.net

WARNING

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

WARRENTY

OI warrants that this instrument will be free from defects in materials and workmanship under normal use and service for a period of 90 days from the date of shipment. OI obligations under this warranty shall not arise until the defective material is shipped freight prepaid to OI. The only responsibility of OI under this warranty is to repair or replace, at it's discretion and on a free of charge basis, the defective material. This warranty does not extend to products that have been repaired or altered by persons other than OI employees, or products that have been subjected to misuse, neglect, improper installation, or accident. OVERTON INSTRUMENTS SHALL HAVE NO LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL 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 5 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.

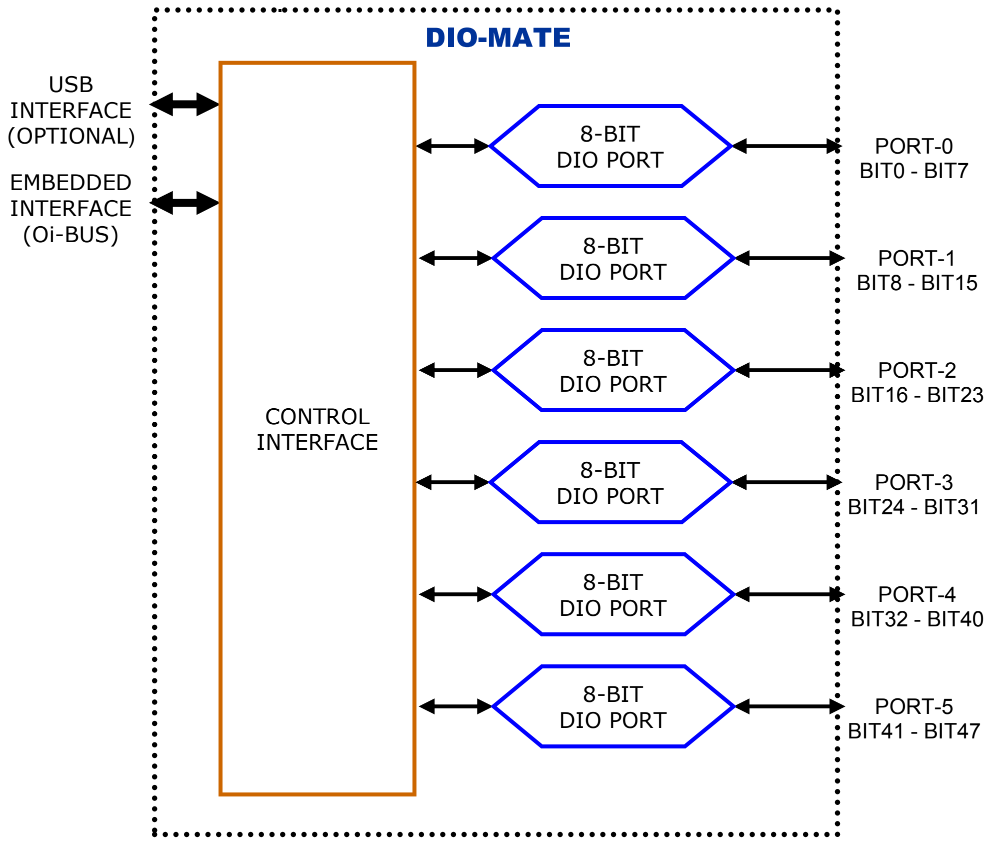
# Table Of Contents

# 1. Introduction

## 1.1 Overview

The DIO-MATE is a basic 48-bit digital I/O module, with many advance features. Test solutions can be quickly built by connecting the DIO-MATE to a PC laptop or desktop. No external power source is required, since power is supplied through the USB interface. Easy access to the hardware is made available through a convenient collection of screw terminal connectors.

Each of the 48 bits can be independently programmed for input or output. There is a polarity function, where input bits can be tagged for inversion when read. Also, a mechanism is provided to allow interrupts to occur if the input changes. In addition, the input bits can be configured with pull-up resistors.



## 1.2 Highlights

| BENEFITS | APPLICATIONS | FEATURES |
|---|---|---|
| • Versatile & Flexible<br>• Easy to program<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>• Automated Test Systems<br>• QA/QC Quality Control<br>• OEM Test Instruments | • 48 digital I/O lines<br>• Programmable port direction<br>• Programmable interrupts<br>• Programmable bit inversion<br>• Programmable pull-ups<br>• USB interface or embedded control<br>• Compact size<br>• Low cost |

## 1.3 Specifications

| **Digital I/O** | |
|---|---|
| Number I/O lines | 48 |
| Logic Levels | TTL compatible |
| Max output current | ±25mA |
| **Input Control** | |
| Embedded | SPI-bus & control logic |
| USB Interface | Optional USB module |
| **General** | |
| Power Supply | +5VDC±10%@30mA |
| Operating Temp | 0-50ºC |
| Dimensions | 3.0” x 4.0” |

# 2. Description

## 2.1 Hardware Details

Access to DIO-MATE hardware is made possible through a convenient set of screw terminal connections (J2 - J5), and J6 (which consolidates all signals into a single 40-pin header).

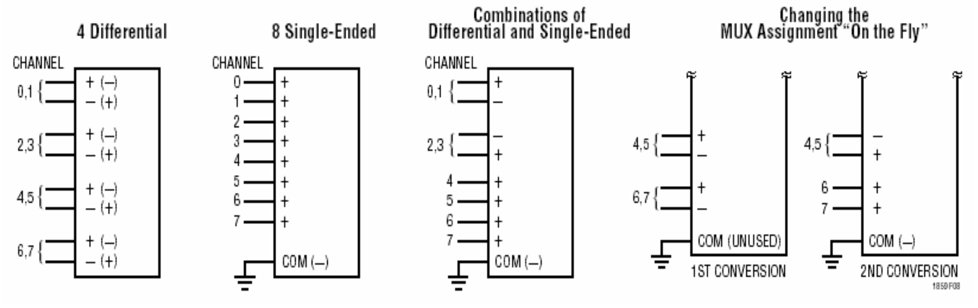
The analog inputs (or channels) can be programmed for any combination of single-ended or differential operation. The diagram below shows examples of various configurations. You will also note the polarity of connections related to differential operation can be transposed as well. Each channel can be 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.

The single analog output channel can be programmed for either unipolar (0-10V), or bipolar (±10V) operation.

The digital circuit includes 8 independent I/O bits. Each bit can be programmed for either input or output. While in the input mode, a bit can be programmed to provide a weak pull-up (~10K). Each bit provides a TTL logic level and can source/sink 25mA.

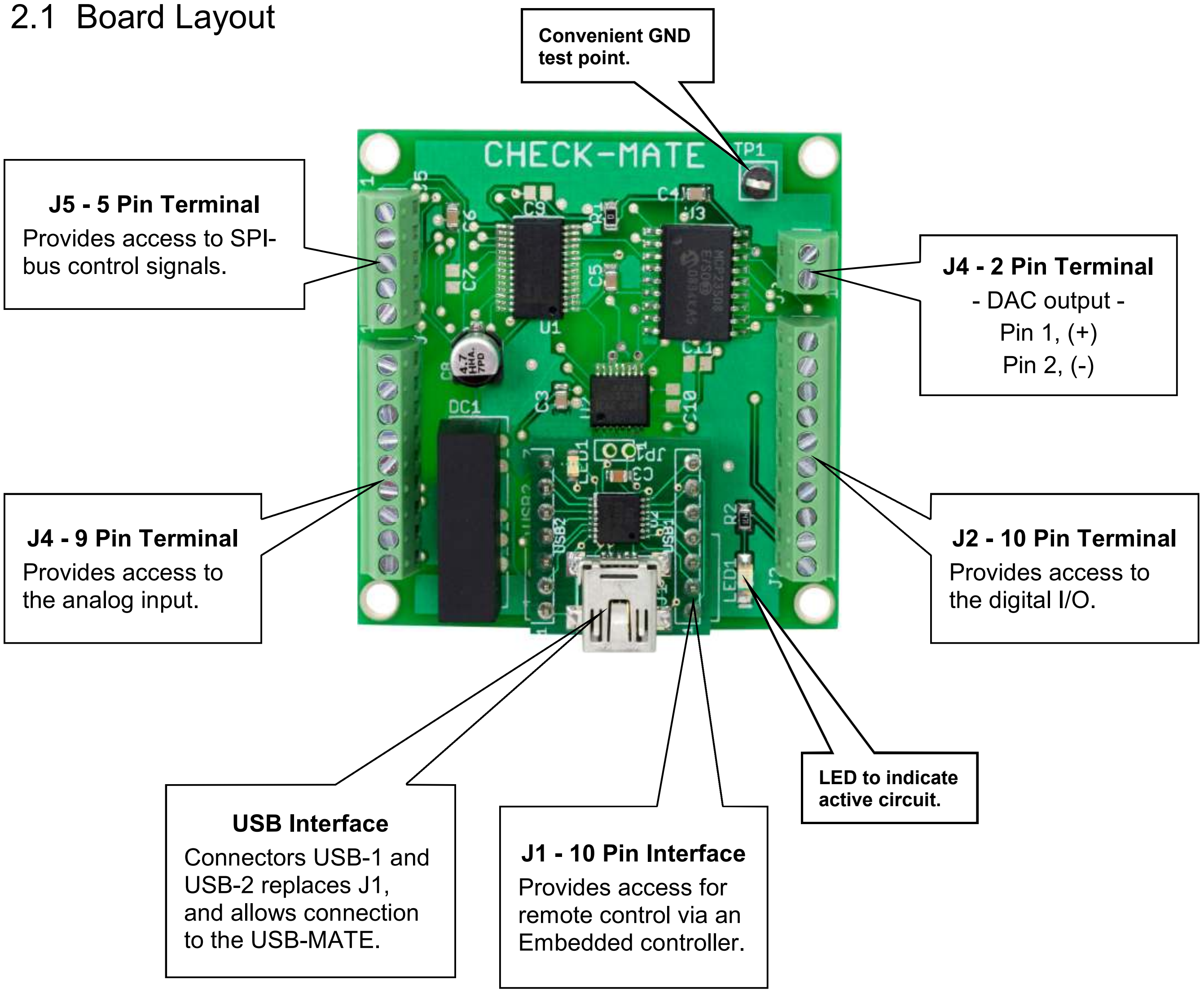
External control of the Check-MATE can be provided by a embedded controller (such as the Micro-MATE), or with a PC. Embedded control is supported by J1 (Oi-BUS interface), which is a 10-pin header that includes a 3-wire SPI-bus, chip select logic, power and ground. In PC applications, connector J1 is 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 Check-MATE. The USB-MATE is designed to interpret a set of ASCII commands sent from the PC, and then perform various Check-MATE functions. For more information for the Check-MATE command set, go to Appendix A. To support embedded applications, a complete driver for the Check-MATE is provided in TES-MATE (or Test Executive Suite).

After power is applied to the Check-MATE, the analog inputs are configured for single-ended (0-5V range), the analog output is set to zero (range is 0-10V), and the digital I/O circuit is cleared (all bits inputs).



# 2. I/O Description

## 2.1 Board Layout


Convenient GND test point.
**J5 - 5 Pin Terminal** Provides access to SPI-bus control signals.
**J4 - 2 Pin Terminal** - DAC output - Pin 1, (+) Pin 2, (-)
**J4 - 9 Pin Terminal** Provides access to the analog input.
**J2 - 10 Pin Terminal** Provides access to the digital I/O.
LED to indicate active circuit.
**USB Interface** Connectors USB-1 and USB-2 replaces J1, and allows connection to the USB-MATE.
**J1 - 10 Pin Interface** Provides access for remote control via an Embedded controller.
CHECK-MATE

## 2.2 Connections

| J1, Port0 | | | |
|---|---|---|---|
| **Pin** | **Name** | **Dir.** | **Description** |
| 1 | VCC | ← | +5V Power |
| 2 | DIO-0 | ←→ | Bit 0 |
| 3 | DIO-1 | ←→ | Bit 1 |
| 4 | DIO-2 | ←→ | Bit 2 |
| 5 | DIO-3 | ←→ | Bit 3 |
| 6 | DIO-4 | ←→ | Bit 4 |
| 7 | DIO-5 | ←→ | Bit 5 |
| 8 | DIO-6 | ←→ | Bit 6 |
| 9 | DIO-7 | ←→ | Bit 7 |
| 10 | DGND | ← | Digital Ground |

| J2, Port1 | | | |
|---|---|---|---|
| **Pin** | **Name** | **Dir.** | **Description** |
| 1 | VCC | ← | +5V Power |
| 2 | DIO-0 | ←→ | Bit 0 |
| 3 | DIO-1 | ←→ | Bit 1 |
| 4 | DIO-2 | ←→ | Bit 2 |
| 5 | DIO-3 | ←→ | Bit 3 |
| 6 | DIO-4 | ←→ | Bit 4 |
| 7 | DIO-5 | ←→ | Bit 5 |
| 8 | DIO-6 | ←→ | Bit 6 |
| 9 | DIO-7 | ←→ | Bit 7 |
| 10 | DGND | ← | Digital Ground |

| J3, Port2 | | | |
|---|---|---|---|
| **Pin** | **Name** | **Dir.** | **Description** |
| 1 | VCC | ← | +5V Power |
| 2 | DIO-0 | ←→ | Bit 0 |
| 3 | DIO-1 | ←→ | Bit 1 |
| 4 | DIO-2 | ←→ | Bit 2 |
| 5 | DIO-3 | ←→ | Bit 3 |
| 6 | DIO-4 | ←→ | Bit 4 |
| 7 | DIO-5 | ←→ | Bit 5 |
| 8 | DIO-6 | ←→ | Bit 6 |
| 9 | DIO-7 | ←→ | Bit 7 |
| 10 | DGND | ← | Digital Ground |

| J4, Port3 | | | |
|---|---|---|---|
| **Pin** | **Name** | **Dir.** | **Description** |
| 1 | VCC | ← | +5V Power |
| 2 | DIO-0 | ←→ | Bit 0 |
| 3 | DIO-1 | ←→ | Bit 1 |
| 4 | DIO-2 | ←→ | Bit 2 |
| 5 | DIO-3 | ←→ | Bit 3 |
| 6 | DIO-4 | ←→ | Bit 4 |
| 7 | DIO-5 | ←→ | Bit 5 |
| 8 | DIO-6 | ←→ | Bit 6 |
| 9 | DIO-7 | ←→ | Bit 7 |
| 10 | DGND | ← | Digital Ground |

## 2.2 Connections, cont.

**J5, Port4**

| Pin | Name | Dir. | Description |
|---|---|---|---|
| 1 | VCC | ← | +5V Power |
| 2 | DIO-0 | ←→ | Bit 0 |
| 3 | DIO-1 | ←→ | Bit 1 |
| 4 | DIO-2 | ←→ | Bit 2 |
| 5 | DIO-3 | ←→ | Bit 3 |
| 6 | DIO-4 | ←→ | Bit 4 |
| 7 | DIO-5 | ←→ | Bit 5 |
| 8 | DIO-6 | ←→ | Bit 6 |
| 9 | DIO-7 | ←→ | Bit 7 |
| 10 | DGND | ← | Digital Ground |

**J6, Port5**

| Pin | Name | Dir. | Description |
|---|---|---|---|
| 1 | VCC | ← | +5V Power |
| 2 | DIO-0 | ←→ | Bit 0 |
| 3 | DIO-1 | ←→ | Bit 1 |
| 4 | DIO-2 | ←→ | Bit 2 |
| 5 | DIO-3 | ←→ | Bit 3 |
| 6 | DIO-4 | ←→ | Bit 4 |
| 7 | DIO-5 | ←→ | Bit 5 |
| 8 | DIO-6 | ←→ | Bit 6 |
| 9 | DIO-7 | ←→ | Bit 7 |
| 10 | DGND | ← | Digital Ground |

**J7**

| Pin | Bit Name | Pin | Bit Name |
|---|---|---|---|
| 1 | Port0-0 | 2 | Port0-1 |
| 3 | Port0-2 | 4 | Port0-3 |
| 5 | Port0-4 | 6 | Port0-5 |
| 7 | Port0-6 | 8 | Port0-7 |
| 9 | Port1-0 | 10 | Port1-1 |
| 11 | Port1-2 | 12 | Port1-3 |
| 13 | Port1-4 | 14 | Port1-5 |
| 15 | Port1-6 | 16 | Port1-7 |
| 17 | Port2-0 | 18 | Port2-1 |
| 19 | Port2-2 | 20 | Port2-3 |
| 21 | Port2-4 | 22 | Port2-5 |
| 23 | Port2-6 | 24 | Port2-7 |
| 25 | Port3-0 | 26 | Port3-1 |
| 27 | Port3-2 | 28 | Port3-3 |
| 29 | Port3-4 | 30 | Port3-5 |
| 31 | Port3-6 | 32 | Port3-7 |
| 33 | Port4-0 | 34 | Port4-1 |
| 35 | Port4-2 | 36 | Port4-3 |
| 37 | Port4-4 | 38 | Port4-5 |
| 39 | Port4-6 | 40 | Port4-7 |
| 41 | Port5-0 | 42 | Port5-1 |
| 43 | Port5-2 | 44 | Port5-3 |
| 45 | Port5-4 | 46 | Port5-5 |
| 47 | Port5-6 | 48 | Port5-7 |
| 49 | DGND | 50 | +5V |

## 2.2 Connections, cont.

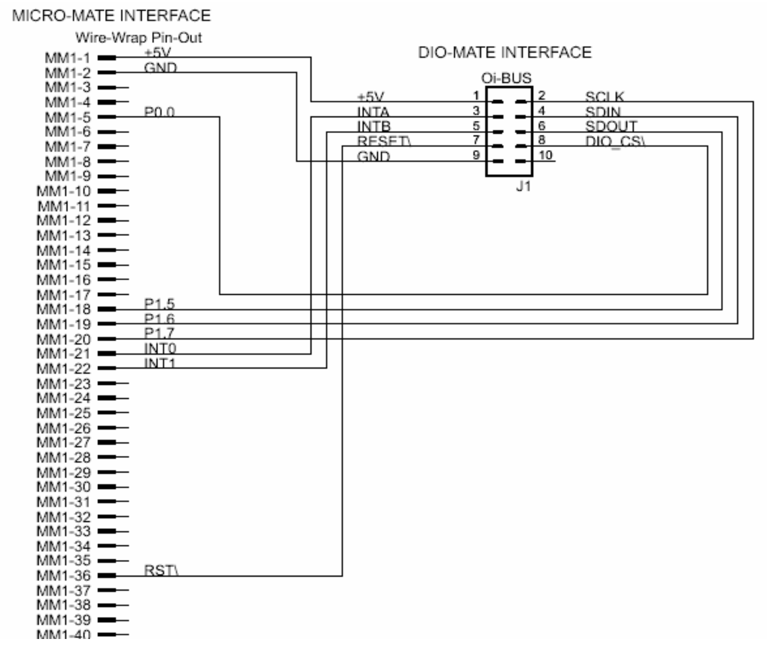
| J8 | | | |
|---|---|---|---|
| **Pin** | **Name** | **Dir.** | **Description** |
| 1 | VCC | → | A regulated +5Vdc input . Current should be limited to roughly 100mA. |
| 2 | SCLK | → | Part of a 3-wire SPI-Bus, SCLK synchronizes the serial data transfer for the DIN and DOUT signals. |
| 3 | INTA | ← | A TTL active-high signal that is a Port-A output interrupt . |
| 4 | DIN | → | Part of a 3-wire SPI-Bus, DIN is serial command and control data for the, ADC, DAC and DIO circuits. |
| 5 | INTB | ← | A TTL active-high signal that is a Port-B output interrupt . |
| 6 | DOUT | ← | Part of a 3-wire SPI-Bus, DIN is serial command and control data for the, ADC, DAC and DIO circuits. |
| 7 | RESET\ | → | A TTL active-low “input' signal that provides a master reset. |
| 8 | DIO_CS\ | → | A TTL active-low “input' signal that provides a chip-select for the DIO. |
| 9 | DGND | → | Digital Ground |
| 10 | - | - | - |

# 3. Operation

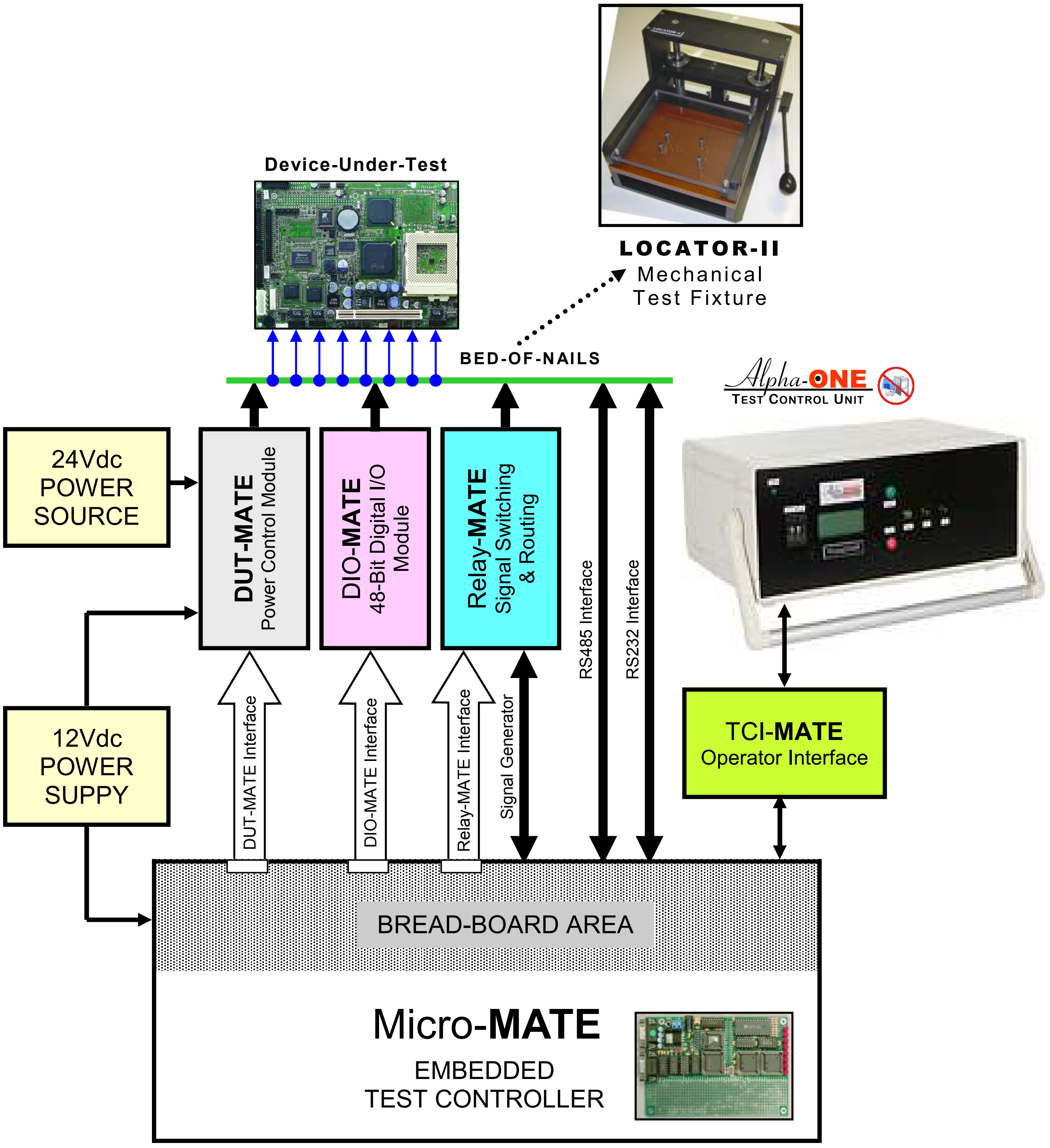
## 3.1 Embedded Control

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

Actually the DIO-MATE can be easily driven by most microcontrollers (including an ARM, AVR, PIC or even a STAMP). When developing an interface for the DIO-MATE, it is recommended the designer start-by reviewing the interface requirements as outlined in the J1 Table (which is provided in the Description section). The next step is to review the DIO-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 DIO-MATE contains 3 SPI-bus devices (which are identical DIO circuits). The DIO circuits are 16-bit devices from Micro-CHIP (part number MCP230S08). Details for specific device performance and SPI-bus operation can be found in their respective data sheets. 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 DIO-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 (P0 & P1 port bits) are assigned unique label names (i.e., SCLK, DOUT), which are used to support various DIO-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 "IO_PN0" command is entered, the program selects 8-bit port '0'. And then when command "IO_PB?" is entered, the program outputs the status for port '0'. Finally, when the command "CK_RA?" is entered, the program call's subroutine "Chk_rd_adc(chk_adc_val , Chk_ch , Chk_adc_range)". This causes the Check-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 DIO-MATE application. For this reason, we suggest that you go to our website and download the "DIO-MATE.zip" file. In the Documents folder will contain more extensive examples of routines to control the DIO-MATE.

## 3.1.3 Embedded Program Example

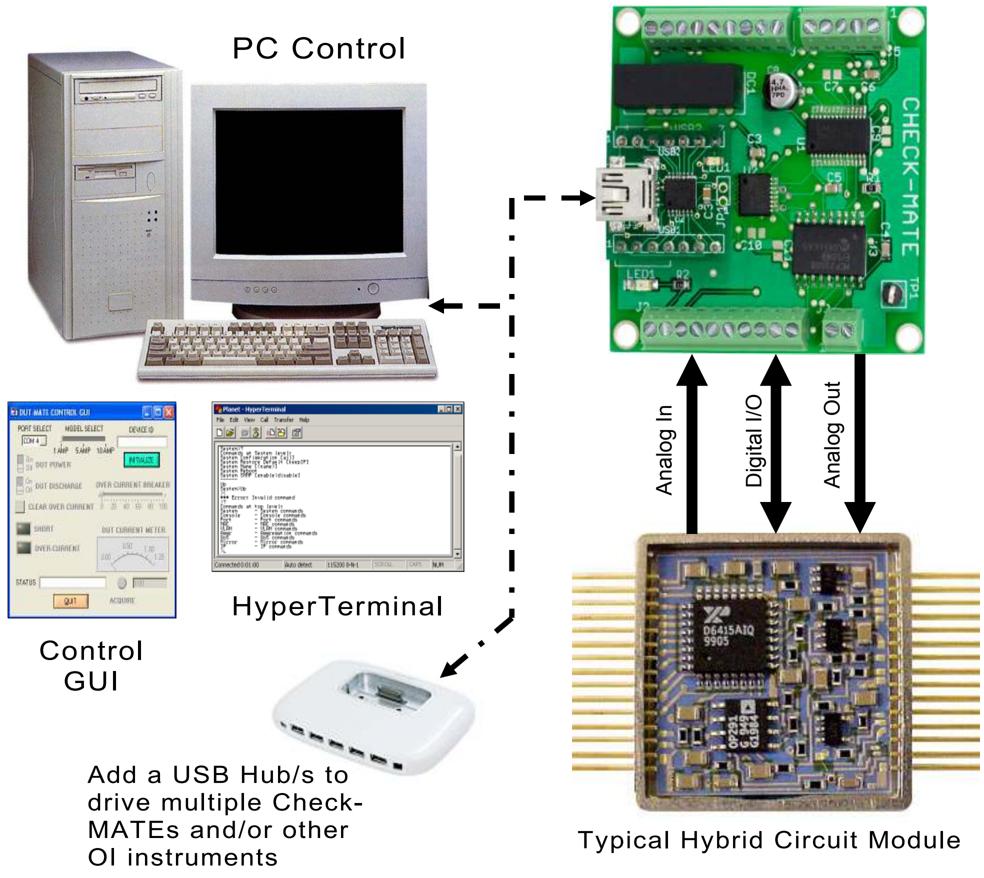
```
' Program: CHECK-MATE Demo
'
---[ Initialization ]----------------------------------------------------------
'
$large
$romstart = &H2000
$default Xram

Dim Chk_adc_word As Word
Dim Chk_adc_val As Single
Dim A_num, A_byte, A_cnt As Byte
Dim Chk_ch, Chk_adc_range, Chk_num, Chk_cnt, Chk_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
Adc_cs Alias P1.3                      ' ADC chip select
Dac_cs Alias P1.4                      ' DAC chip select
Dio_cs Alias P1.5                      ' DIO chip select
Dac_mode Alias P1.6                    ' DAC mode, (1) unipolar, (0) bipolar

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 Chk_rd_adc(chk_adc_val As Single , Chk_ch As Byte , Chk_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 Sclk, Dout, Adc_cs, Dac_cs, Dio_cs, Dac_mode   ' 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 Chk_adc_range = Chk_adc_5v Then A_str = "0"
          If Chk_adc_range = Chk_adc_10v Then A_str = "1"
          If Chk_adc_range = Chk_adc_5v5v Then A_str = "2"
          If Chk_adc_range = Chk_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 Chk_adc_range = Chk_adc_5v
            If A_num = 1 Then Chk_adc_range = Chk_adc_10v
            If A_num = 2 Then Chk_adc_range = Chk_adc_5v5v
            If A_num = 3 Then Chk_adc_range = Chk_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
            Chk_ch = A_num
          End If
        End If

      Case "RV":                 ' read voltage

        A_resp = Mid(s , 6 , 1)
        If A_resp = "?" Then
          Call Chk_rd_adc(chk_adc_val , Chk_ch , Chk_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 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

Sub Chk_rd_adc(chk_adc_val As Single , Chk_ch As Byte , Chk_adc_range As Byte)
   Chk_adc_val = &H0000
           ' Select range
   If Chk_adc_range = Chk_adc_5v Then Chk_cntl_byte = Chk_adc_5v
   If Chk_adc_range = Chk_adc_10v Then Chk_cntl_byte = Chk_adc_10v
   If Chk_adc_range = Chk_adc_5v5v Then Chk_cntl_byte = Chk_adc_5v5v
   If Chk_adc_range = Chk_adc_10v10v Then Chk_cntl_byte = Chk_adc_10v10v
           ' Select analog channel
   Chk_cntl_byte = Chk_cntl_byte || Chk_ch
   Reset Sclk
           ' take X measurements
   For Chk_cnt = 1 To Chk_m_cnts
    Chk_adc_word = &H0000
    Chk_num = 7
    Chk_num_2 = 11
             ' Select device
    Reset Adc_cs
    For Chk_cnt_2 = 1 To 24
      If Chk_cnt_2 < 9 Then
             ' Send control byte
        Dout = Chk_cntl_byte.chk_num
        Set Sclk
        Reset Sclk
        Decr Chk_num
      Elseif Chk_cnt_2 > 12 Then
             ' Get ADC value
        Set Sclk
        Reset Sclk
        Chk_adc_word.chk_num_2 = Din
        Decr Chk_num_2
      Else
             ' dummy clocks
        Set Sclk
        Reset Sclk
      End If
    Next Chk_num
             ' disable device
    Set Adc_cs
             ' collect results
    Chk_adc_val = Chk_adc_val + Chk_adc_word
    Waitms 1
   Next Chk_cnt
             ' compute average
   Chk_adc_val = Chk_adc_val / Chk_m_cnts

End Sub
```

## 3.2 PC Control

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

Controlling the Check-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 Check-MATE side, the PIC controller processes the serial commands and then drives the Check-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.Check-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 Check-MATE is used to stimulate a hybrid module in a test & measurement application. The hybrid module is a mix-signal device that requires Analog I/O, as well as Digital I/O to function properly.

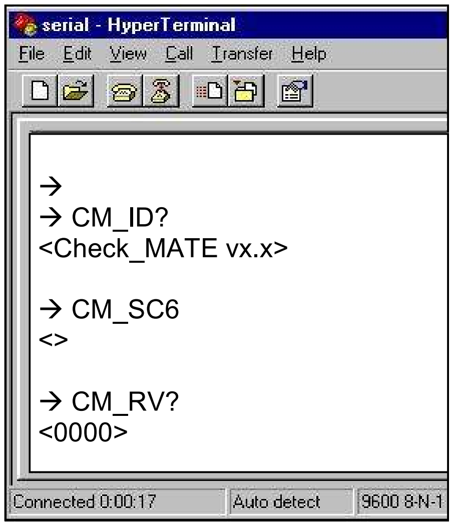

PC Control
Control GUI
HyperTerminal
Add a USB Hub/s to drive multiple Check-MATEs and/or other OI instruments
Analog In
Digital I/O
Analog Out
Typical Hybrid Circuit Module

## 3.2.1 PC Programming

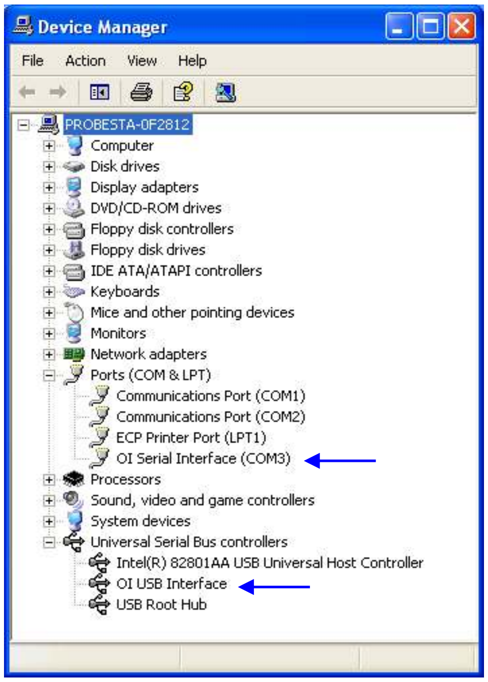
The starting point for developing code to control the Check-MATE, begins with acquainting yourself with its Serial Command Set. The serial commands are a sequence of ASCII characters that originate from the PC and are designed to instruct the Check-MATE to perform specific functions. The complete serial 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 (Control GUI).

### 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 Check-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.



```
→
→ CM_ID?
<Check_MATE vx.x>

→ CM_SC6
<>

→ CM_RV?
<0000>
```



### 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 inter-act and control the Check-MATE. Download the panel from our website at www.check-mate.com, click on downloads and select "Check-Matexxx.exe".

**First Step:** The User must select a COM Port. Refer to the Device Manage to iden-tify 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.

This 'Range' function selects (1 of 4) specific analog input modes. Each 'Analog Input CH' can be set to a different range setting.

The 'Volt Meter', displays a voltage measurement based the current analog channel and range setting.

The 'Analog Input CH' func-tion selects an individual analog channel (1 to 8).

The 'ACQUIRE' function updates the analog con-figuration settings, and displays a measurement every 100msec.

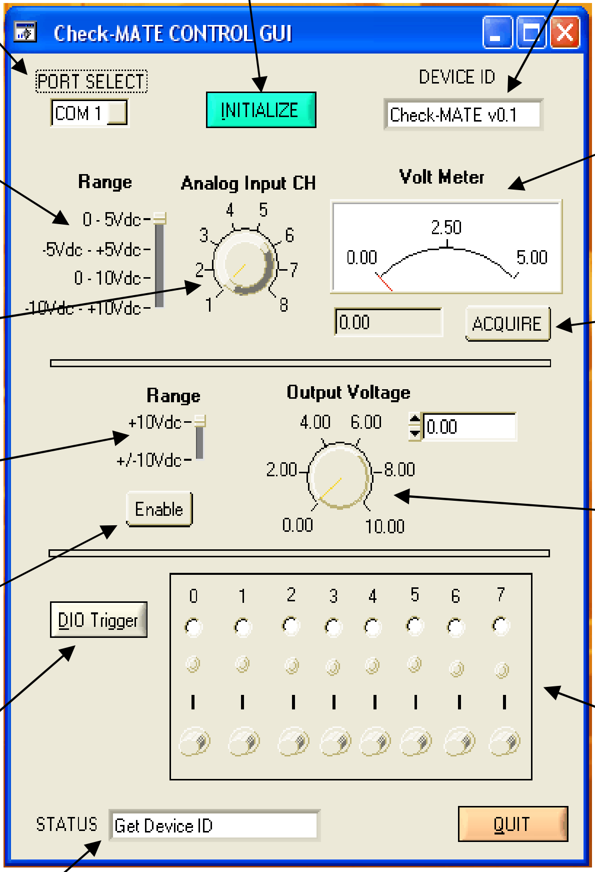
This 'Range' function selects either Unipolar or Bipolar operation.

The 'Output Voltage' func-tion updates the analog configuration settings, and displays a measurement every 100msec.

The 'Enable' function updates the analog output settings.

The 'DIO Trigger' function updates the DIO configura-tion settings.

This function panel allows the User to control the DIO circuit. The top section provides a tool for setting the 'bit' direction. A blank-circle (indicates input), and a dot-circle (indicates out-put). The middle section includes a set of eight LED's (which indicate input status). The bottom section includes eight push-button switches (which allow the setting of output bits). When the switch is the out position (that represents a logic '0'). When the switch is in the in position (that represents a logic '1').



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

## 3.2.1.3 PC Programming Example

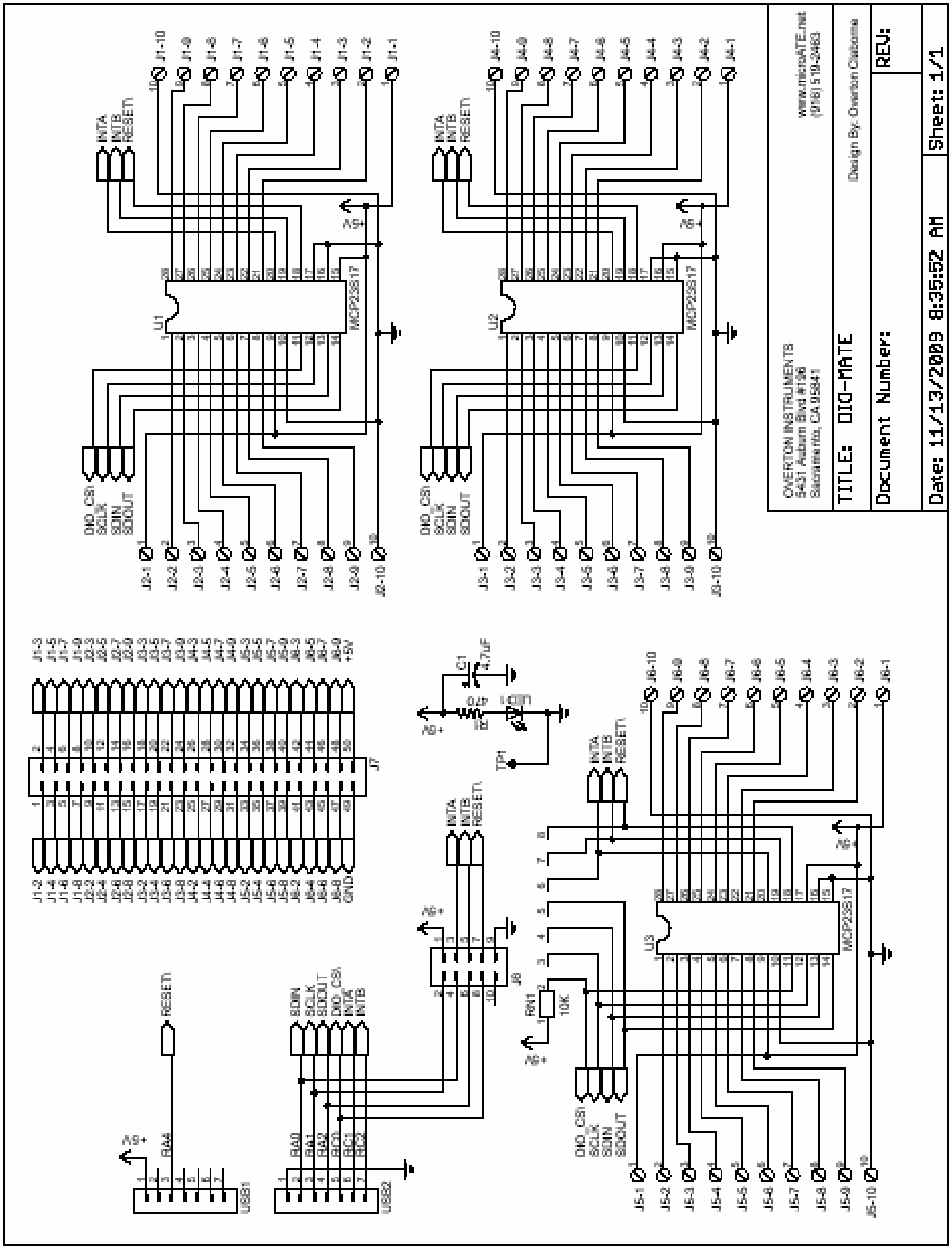
```
//  Check-MATE programming example in 'C'
//
//  The following program provides a Go/No Go test sequence for testing
//  a hypothetical electronic module.  The electronic module is a mix-
//  signal hybrid device that contains 8 programmable amplifiers.  The
//  electronic module is controlled by a Check-MATE via the DIO lines.  DIO
//  bits 0-3 (select one of 8 DUT amplifiers).  DIO bits 4 & 5 (selects the
//  gain range).  DIO bit 6 is active-low (provides a DUT chip-select).  DIO
//  bit 7 is active-high (which indicates the DUT is ready).  The outputs of
//  the DUT amplifiers are connected to the inputs of the Check-MATE ana-
//  log channels.  The objective for the program is to verify each of the 8
//  amplifiers will perform properly at each gain setting and over a varying
//  range of input voltage levels.  During the test sequence, the program
//  first selects both the DUT amplifier and the Check-MATE ADC chan-
//  nel.  Then the DUT gain is selected and the Check-MATE updates the
//  DUT by writing the control byte (which asserts the chip-select).  The
//  Check-MATE then reads the DIO-bit-7 to determine if the DUT is
//  ready.  Once the DUT is ready, the Check-MATE will stimulate the
//  DUT amplifier input by supplying a voltage from the DAC output.  To
//  verify the DUT amplifier, the program reads the Check-MATE analog
//  channel and determines the PASS/FAIL results.

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

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

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

long value = 0, limit = 0;

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

char set_adc_range[]  = "CK_AR";      // set ADC voltage range
char set_adc_ch[]       = "CK_SC";      // set ADC channel
char get_adc_volts[]   = "CK_RV?";     // read voltage
char set_dac_range[]   = "CK_DM";      // set DAC voltage range
char set_dac_out[]      = "CK_SA";      // set DAC output voltage
char set_dio_dir[]       = "CK_PD";      // set DIO port direction
char set_dio_pullup[]  = "CK_PU";      // set DIO port pull-up
char set_dio_port[]     = "CK_PB";      // set DIO port write
char get_dio_port[]     = "CK_PB?";     // get DIO port
char get_device_id[]   = "CK_ID?";     // get module ID
char master_clear[]    = "CK_MC";      // master clear

main()
{
    port=OpenComPort(1,256,64);  // Open COM 1, rx_buff = 256 bytes, tx_buff = 64

    if ((stat = SetPortCharacteristics(port,BAUD19200,PAR_EVEN,
                LENGTH_8,STOPBIT_1,PROT_NONNON)) != RS232ERR_NONE) {
         printf("Error #%d setting characteristics\n",stat);
        exit(1);
    }
    CdrvSetTimerResolution(port,1);        // 1 msec ticks
    SetTimeout(port,2000);                 // 2000 ticks = 2 sec time-out
    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(1);
    }
    if (strcmp("-> ", read_data)) {
        printf("Incorrect promt\n");
        exit(1);
    }
                // Master Clear

        sprintf (send_data, "%s\r", master_clear);
        PutString(port,send_data);          // send CK_MC

                // Set DIO direction & weak pull-up

    sprintf (send_data, "%s%s\r", set_dio_dir, "10000000");
    PutString(port,send_data);          // send CK_PD10000000
    sprintf (send_data, "%s%s\r", set_dio_pullup, "10000000");
    PutString(port,send_data);          // send CK_PU10000000

                // Execute test sequence

    for (dut_ch = 0; dut_ch >= 7; dut_ch++) {

                //  set check-mate ADC channel & range

        sprintf (send_data, "%s%d\r", set_adc_ch, dut_ch);
        PutString(port,send_data);                      // send CK_SC
        sprintf (send_data, "%s%d\r", set_adc_range, 1);
        PutString(port,send_data);        // send CK_AR - 0-10Vdc

                        // exercise DUT gain performance

       for (gain_sel = 0;  >= 3; gain_sel++) {
            if (gain_sel == 0) dut_gain = 4095;         // x1 range
            if (gain_sel == 1) dut_gain = 409;          // x10
            if (gain_sel == 2) dut_gain = 40;           // x100
            if (gain_sel == 3) dut_gain = 4;            // x1000

                        // build dio control byte

            a_byte = dut_ch + (gain_sel + 8)
            for ( idx = 0; idx <= 7; idx++ ) {
                dio_bit[idx] = a_byte % 2;
                a_byte = a_byte / 2;
                sprintf (dio_byte[idx], "%d", dio_bit[idx]);
            }

                        // Select DUT, gain & amp ch

            sprintf (send_data, "%s%s\r", set_dio_port, dio_byte);
            PutString(port,send_data);   // send CK_PBxxxxxxxx

            do {            // Get DIO input - check DUT ready

                sprintf (send_data, "%s\r", get_dio_port);
                PutString(port,send_data);  // send CK_PB?
                GetString(port,sizeof(read_data),read_data);

            } while (atoi (read_data[1])); // loop while msb = '0', DUT not ready

            do {            // Set check-mate DAC output

                sprintf (send_data, "%s%04d\r", set_dac_out, dut_gain);
                PutString(port,send_data);                  // send CK_SAnnnn

                        // Get check-mate ADC input

                sprintf (send_data, "%s\r", get_adc_ch);
                PutString(port,send_data);                  // send CK_SA?
                GetString(port,sizeof(read_data),read_data);
                for ( idx = 1; idx <= 4; idx++ ) {
                    results[idx] = read_data[idx];
                }
                        // determine pass/fail results

                Value = atoi(results);
                 if (gain_sel == 1) dut_gain = dut_gain * 10;
                 if (gain_sel == 2) dut_gain = dut_gain * 100;
                 if (gain_sel == 3) dut_gain = dut_gain * 1000;
                 limit = asb(value - dut_gain);
                 if (limit > (0.001 * 4096)) {
                     printf ("Test Failed - ADC Ch:", "%d", " Gain Range:",
                             "%d", " Gain Value", "%d", dut_ch, gain_sel, dut_gain);
                             exit(1);
                 {
                dut_gain--;

            } while (dut_gain != 0);

                        // De-select DUT

            sprintf (send_data, "%s%s\r", set_dio_port, "00000000");
            PutString(port,send_data);                  // send CK_PB00000000
        }
    }
    printf ("Test Passed");
}
```

# Appendix A. Serial Command Set

To facilitate remote control for the DIO-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 DIO-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 **'IO_',** 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 DIO-MATE will return either a **'<>'**, or a bracketed result (i.e. **'<2108>'**. If the DIO-MATE receives a carriage-return or line-feed alone (without a command), then a **'→'** is returned (this response is a "prompt" to signal the DIO-MATE is ready). If the DIO-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 |
|---|---|---|---|
| **IO_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). |
| **IO_BR?** | Get baud rate code | <n> | Get current baud rate code (-n- is the return code 0 to 3). |
| **IO_ID?** | Get module ID | <CHECK-MATE vx.x> | Get current identification and version number. |
| **IO_MR** | Maser Reset | <> | Reset & initialize the module |
| **IO_WC** | Write configuration | <> | Store current instrument settings in EEPROM. Save settings related to the DIO hardware. |
| **IO_RC** | Recall configuration | <> | Retrieve stored instrument settings |
| **IO_PDbbbbbbbb** | Set DIO direction | <> | Set (or write) the DIO port direction. The direction byte is represented by eight ASCII bytes starting with the most-significant-bit (-b- left most) to the least-significant-bit (-b- right most). A logic '1' is input and '0' is output. |
| **IO_PD?** | Get DIO direction | <bbbbbbbb> | Get (or read) the current DIO port direction setting. |
| **IO_PUbbbbbbbb** | Set weak pull-ups | <> | Set (or write) pull-ups on the DIO port inputs. The pull-up byte is represented by eight ASCII bytes starting with the most-significant-bit (-b- left most) to the least-significant-bit (-b- right most). A logic '1' is active and '0' is not. |
| **IO_PU?** | Get weak pull-ups | <bbbbbbbb> | Get (or read) the current DIO port pull-up status. |
| **IO_PBbbbbbbbb** | Set DIO port | <> | Set (or write) the DIO port output bits. Depending on the condition of the direction byte, the output bits are represented by eight ASCII bytes starting with the most-significant-bit (-b- left most) to the least-significant-bit (-b- right most). The -b- bit is a logic '1' or '0'. |
| **IO_PB?** | Get DIO port | <bbbbbbbb> | Get (or read) the current DIO port status. |

# Appendix B. Schematic



# Appendix C. Mechanical Dimensions