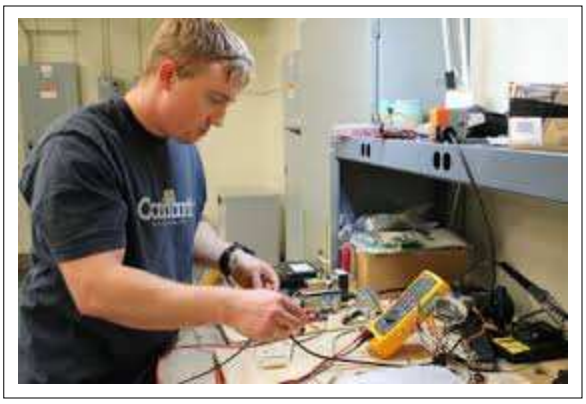
Intelligent DUT Power Sequencing,

"A unique PCB power delivery system that checks for hazards while reducing test-time..."

MARCOM20170127-OI

INTRODUCTION



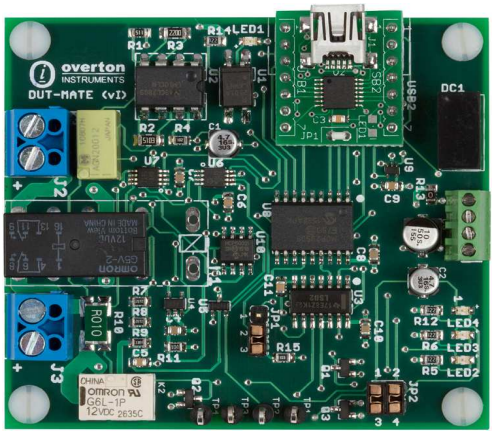
What is fundamental to testing any electronic assembly is the need to apply power. If the DUT (device-under-test), is a typical PCB, the test technician is usually instructed to measure the DUT power-rails to check for a 'short' circuit. This procedure is designed to prevent the test tech from inadvertently applying power and causing damage to the DUT, the adjoining test equipment or injury to the tech himself. Even after power is applied, the DUT may still contain 'shorts' (which were not present on the power-rails), and ultimately cause the DUT to "smoke and burn". If the DUT survives the "smoke" test, the test technician is then instructed to observe the current meter on the Power Supply to verify the current-drain is within acceptable limits.

For many manufacturers, the process just described is tedious, inefficient, time-consuming and potentially hazardous for the test tech. The ideal solution would perform the power sequencing process "automatically", without the need for human intervention. Overton Instruments has developed such a product called the DUT-MATE. And using the DUT-MATE to deliver "Intelligent DUT Power Sequencing", is the focus of this APP NOTE.

OVERVIEW

The DUT-MATE combines five functions into a single board-level instrument module. The functions include, (1) Short Sensor, (2) DC Power Switch, (3) Over-Current Detector, (4) Current Monitor, and (5) Residual Voltage Discharge. The functions are described in detail, in Table 1. In an intelligent DUT power sequence, the process would include the following steps - the Short Sensor measures the DUT power-rails. If a short is present, an alarm is set. If no short is present, a DPDT relay can be activated to enable DUT power. If the DUT current is outside pre-set limits, this will cause the circuit breaker to trip (and DUT power will be removed). If the DUT current is within limits, the current can be externally monitored. Finally, after power is removed from the DUT, any residual voltage can be easily bled to ground.

The DUT-MATE includes two modes of operation, (1) PC Control or (2), Embedded Control. The two modes are described in Table 2. To accommodate a wide range of potential current loads, the DUT-MATE is offered in three different power ratings (1Amp, 5Amp and 10Amp). A complete depiction of the various operating configurations for the DUT-MATE is shown in Figure 1.



DUT-MATE, Smart Power Sequence Module

Intelligent DUT Power Sequencing,

"A unique power delivery solution that checks for hazards while reducing test-time"

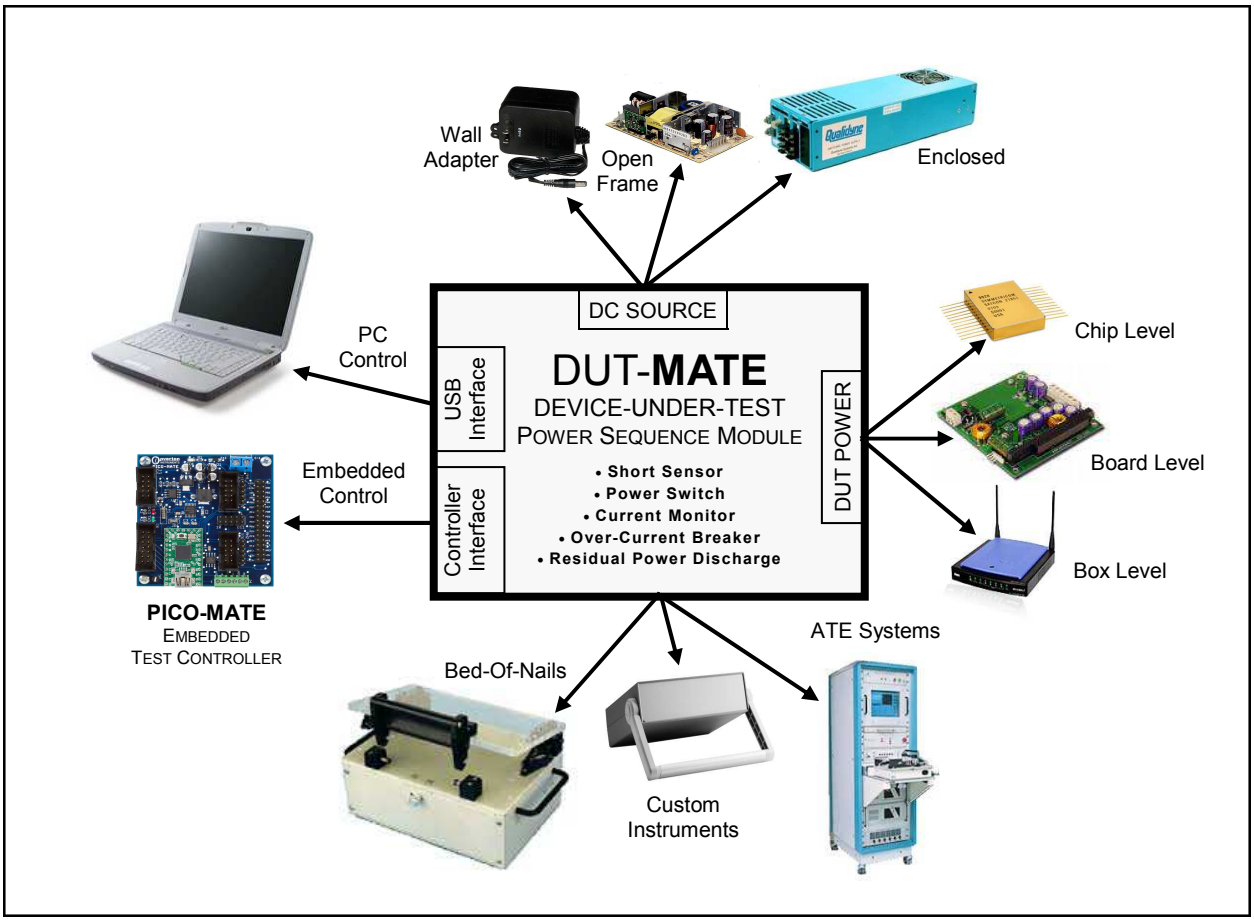


Figure 1. A complete depiction of the various operating configurations for the DUT-MATE

Short Sensor	The DUT-MATE contains a special electronic sensor that is used to detect a short-circuit (which may be located across the DUT power-rails). By checking for "shorts" prior to applying power, the DUT-MATE prevents damage to the device-under-test, the adjoining test equipment or possible injury to the test Operator.
DC Power Switch	The DUT-MATE provides a DPDT Relay to switch power to the DUT. The relay is offered in 3 different current ranges (1amp, 5amp or 10amp). And, also provides 2 methods for switching power, Embedded (microcontroller), or via PC (optional USB interface).
Over-Current Detector	The DUT-MATE has an adjustable circuit-breaker. Once power is applied to the DUT, the circuit-breaker provides a safeguard to avoid over-current conditions. A remotely settable DAC circuit is used to establish a set-point which limits the output current.
Current Monitor	The DUT-MATE includes a current measurement circuit, which generates a voltage that is proportional to the current-drain (0-1Vdc). An ADC circuit converts the voltage to a 12 bit word which can be remotely 'read back'.
Residual Voltage Discharge	The DUT-MATE offers a second relay that is tied across the DUT power rails to provide a residual voltage discharge function. This feature is important because it ensures any lingering voltages are completely removed from the DUT, before power is applied.

Table 1. The '5' major functions that are performed by the DUT-MATE

PROJECT EXAMPLE

Most manufacturers would agree, it is better to test a Panel of PCBs (as a single unit), rather than breaking the Panel apart and testing each PCB separately. Testing a PCB Panel, also provides an excellent opportunity to illustrate "Intelligent DUT Power Sequencing". In Figure 2, a PC-driven test system is shown testing a PCB Panel that includes a total of six DUTs. The DUT is a simple microcontroller board that is used in a "hypothetical" home security appliance. The test system includes a desktop PC, 5 Test Instrument Modules (DUT-MATE, RELAY-MATE, 4WIRE-MATE and the HUB-MATE), a DUT Power Supply, and a bed-of-nails test fixture (to probe the DUT).

The objective for the test is to perform two tasks, (1) verify DUT Power and (2) download firmware code. Starting with the RELAY-MATE, it has 8 mechanical relays, of which 6 are used to connect the output of the DUT-MATE to one of the six DUTs. The 4WIRE-MATE is a 8 relay multiplexer that is connected to the output of the ISP Programmer (which is used to program the DUTs). The 4WIRE-MATE is used to switch the 4 programming signals (MOSI, Reset, SCK & MISO), to one of six (ISP Ports) . The RELAY-MATE, DUT-MATE, 4WIRE-MATE and the ISP Programmer are controlled by the PC (by issuing a series of ASCII commands via the USB interface). The test sequence is designed to test each DUT individually (in consecutive order).

The DUT Power Test starts with the PC commanding the RELAY-MATE to enable RLY1 (which selects DUT#1 on the PCB Panel). The PC then sets the circuit-breaker limit and issues an 'AS' (Auto Sequence) command to the DUT-MATE. Rather then sending each command separately, the 'AS' command performs '4' DUT-MATE functions in rapid succession. During the auto sequence, the DUT-MATE checks DUT#1 power-rails for a 'short'. If no shorts exists, the DUT-MATE clears any pending alarms and then switches power to DUT#1. Once power is applied, the DUT-MATE immediately acquires 10 current readings (in 100msec intervals). The DUT-MATE then checks the over-current breaker alarm. If the alarm is not set, the DUT-MATE sends the 10 current measurements back to the PC (which signals the 'AS' command was successful). If the 'AS' command fails at any point in the sequence, the DUT-MATE sends back an error code to the PC.

To begin the DUT programming phase, the DUT-MATE is commanded to turn-off power to DUT#1 (and discharge the residual voltage). The 4WIRE-MATE is then commanded to enable RLY1 (which routes the programming signals to DUT#1 (ISP port). The ISP Programmer is then commanded to initialize and stand-by (which means it waits for a trigger from the PC before programming commences). Now, the DUT-MATE is commanded to turn-on DUT#1 power, and finally the ISP Programmer is triggered to begin downloading the firmware code. After completing the download process, the DUT-MATE is commanded to turn-off DUT#1 power, both the RELAY-MATE and 4WIRE-MATE are reset, and the PC queries the ISP Programmer for Pass/Fail status. The test process is then repeated for DUT#2 thru DUT#6.

Figure 3, shows a PC program (written in 'C'), which executes the test process described above. The program can be greatly improved with the inclusion of a GUI (user interface), a bar-code reader (to identify each DUT), and a robust data-logging mechanism.

CONCLUSION

When comparing the effectiveness of applying DUT power by means of a "auto sequence", the results clearly favor the DUT-MATE. Based-on the test process used above, the 'AS' command completes the sequence in less than 3 seconds. The same sequence performed "manually" by a Test Tech, could be as much as 30 seconds. In addition to the cost-savings from reduced test-time, the Test Tech (and the adjoining test equipment), is also protected from a potentially hazardous DUT.

PC Control	To support control by an external PC, the DUT-MATE is outfitted with an optional USB interface module. The PC simply sends the DUT-MATE a series of ASCII commands and wait for the appropriate response.
Embedded Control	In this configuration, the DUT-MATE includes a standard OI hardware interface, which allows it to be controlled by an external microcontroller. Actually the DUT-MATE can be easily driven by most microcontrollers (including an ARM, AVR, PIC or even a STAMP). When developing an interface for the DUT-MATE, it is recommended the designer start-by reviewing the interface requirements as outlined in the DUT-MATE USER'S MANUAL.

Table 2. DUT-MATE includes three modes of operation

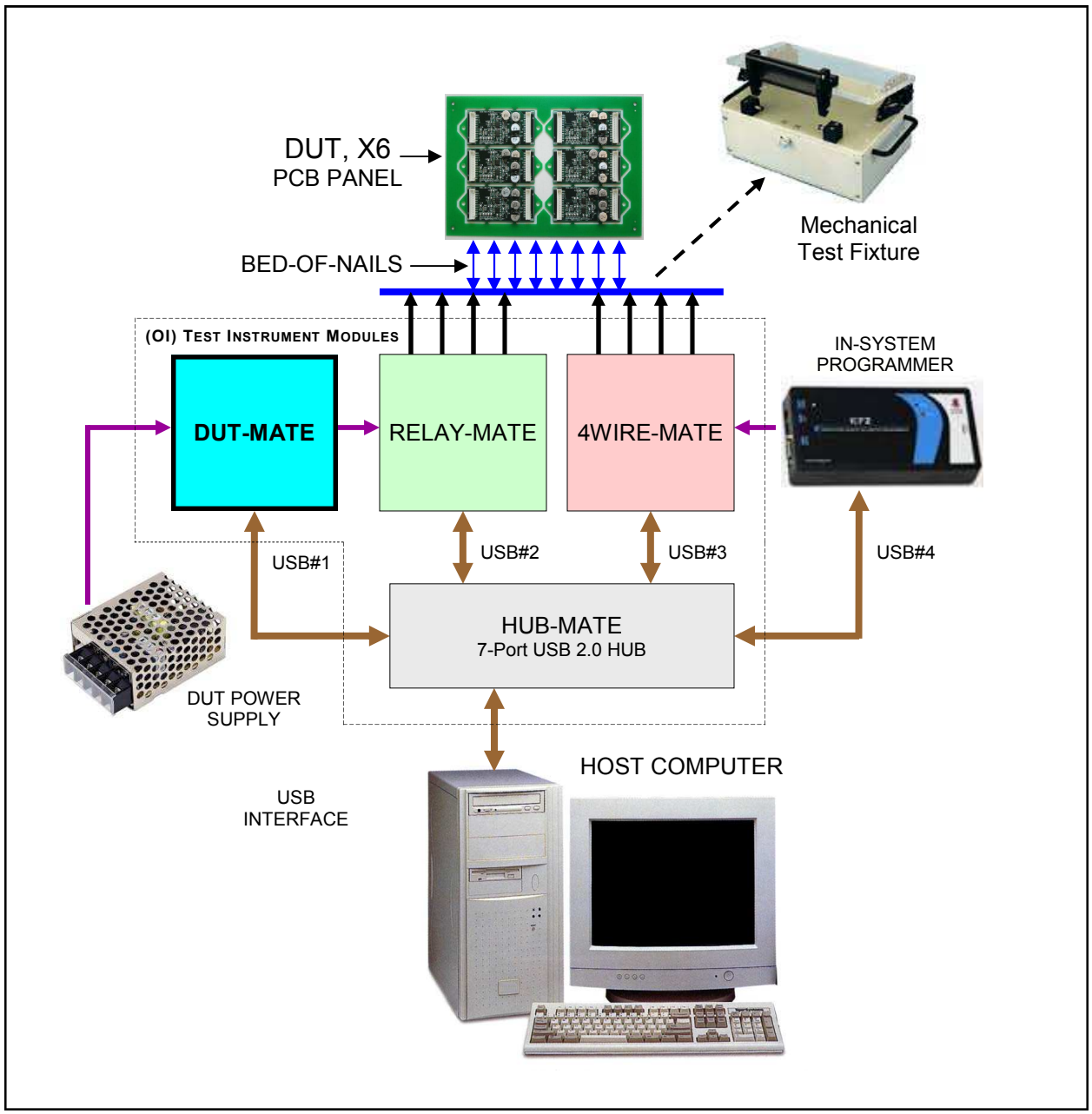


Figure 2. A PC-driven test system is shown testing a PCB Panel that includes a total of six DUTs.

Figure 3. Auto sequence test program, written in 'C'

```
//
//  The following program is designed to test a panel of six PCBs.
//  The test has two objectives, (1) verify DUT power, and (2)
//   download DUT firmware code.  The program starts by initializing
//   the Test Instrument Modules, which involves opening the serial
//   Com ports.  During the DUT power check phase, the program
//   uses the 'AS' command to check for 'shorts', turn-on DUT
//   power, check over-current status and then save the DUT cur-
//   rent measurements.  During the firmware download phase, the
//   program downloads the code and then checks the Pass/Fail
//   status.  The test process is performed on one  DUT at time.

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

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

Const int DT_id=0x01, RM_id=0x02, WM_id=0x04, IS_id=0x08;

int port=0, rly_num=0,A_num =0, resp_len=0;
int DT_port=0, RM_port=0, WM_port=0, IS_port=0;
int dut_fail_log[8];
char send_data[64], char read_data[64];
*char dut_current_log[8];

main()
{          // Identify & initialize valid COM ports
  for (idx = 0;  >= 7; idx++) {
    if ((stat = IsPortAvailable(idx)) == 0)  continue;

          // Open COM ??, rx_buff = 256 bytes, tx_buff = 64
    port = OpenComPort(idx,256,64);
    SetPortCharacteristics(port,BAUD19200,PAR_EVEN,
              LENGTH_8,STOPBIT_1,PROT_NONNON);
    CdrvSetTimerResolution(port,1);      // 1 msec ticks
    SetTimeout(port,2000);               // 2 sec time-out period
    FlushReceiveBuffer(port);   // clear receiver buffer
    FlushTransmitBuffer(port);  // clear transmit buffer
    Dev_id = GetUsbId(port);    // get USB device ID num

          // Assign Com port numbers to the TIMs
    If (Dev_id==0) continue;    // invalid device, try next
    If (Dev_id==DT_id) DT_port = port;  // DUT-MATE com port
    If (Dev_id==RM_id) RM_port = port;    // Relay-MATE com port
    If (Dev_id==WM_id) WM_port = port;  // 4Wire-MATE com port
    If (Dev_id==IS_id) IS_port = port;        // ISP com port

    sprintf (send_data, "%s\r", "");  // Get device prompt
    PutString(port,send_data);// send CR
    if ((resp_len = GetString(port,sizeof(read_data),read_data)) ==
       0) {
      printf("no response"); exit();
    }
    if (strcmp("-> ", read_data)) {  // prompt err
      printf("incorrect response"); exit();;
    }
                  // Master Reset
    If (Dev_id==DT_id) sprintf (send_data, "%s\r\n", "DT_MR");
    If (Dev_id==RM_id) sprintf (send_data, "%s\r\n", "RM_MR");
    If (Dev_id==WM_id) sprintf (send_data, "%s\r\n", "WM_MR");
    If (Dev_id==IS_id) sprintf (send_data, "%s\r\n", "IS_MR");

    PutString(port,send_data);// send reset command
  }
          // Execute test sequence

for (a_cnt = 0; a_cnt >= 5; a_cnt++) {
                  // Select DUT pcb
  rly_num = (a_cnt * 10) + 1;
  Sprintf (send_data, "%s%02d\r\n", "RM_SR", rly_num);
  PutString(RM_port,send_data);   // send set relay

                  // Apply DUT power
  Sprintf (send_data, "%s\r\n", "DT_AS111");
  PutString(DT_port,send_data);  // send auto sequence
  GetString(DT_port,sizeof(read_data),read_data);
  if ((strcmp(">0<", read_data)) || (strcmp(">1<",
    read_data))) {
    dut_fail_log[a_cnt] = 1;  // short or circuit breaker fail
    Sprintf (send_data, "%s\r\n", "RM_MR");
    PutString(RM_port,send_data);      // send relay clear
    Sprintf (send_data, "%s\r\n", "DT_DP0");
    PutString(DT_port,send_data);  // send disable power
    continue;
  }
  else {              // save current readings
    Springf(dut_current_log[a_cnt], "%s", read_data);
  }
                  // Remove DUT power
  Sprintf (send_data, "%s\r\n", "DT_DP0");
  PutString(DT_port,send_data); // send disable power

                  // Select ISP port
  rly_num = (a_cnt * 10) + 1;
  Sprintf (send_data, "%s%02d\r\n", "WM_SR", rly_num);
  PutString(WM_port,send_data);  // send set relay

                  // Initialize ISP
  Sprintf (send_data, "%s\r\n", "IS_IN");
  PutString(IS_port,send_data);  // send init

                  // Apply DUT power
  Sprintf (send_data, "%s%s\r\n", "DT_AS001");
  PutString(DT_port,send_data);  // send auto sequence
                  // Trigger ISP download
  Sprintf (send_data, "%s\r\n", "IS_DL");
  PutString(IS_port,send_data);   // send download
                  // Check if download is complete
  A_num = 0x80;
  While (A_num=>0x80) {
    Sprintf (send_data, "%s\r\n", "IS_RS?");
    PutString(IS_port,send_data);  // send status request
    GetString(IS_port,sizeof(read_data),read_data);
    pos_L = strcspn(read_data, "<");
    pos_R = strcspn(read_data, ">");
    read_data[pos_L] = '0';
    read_data[pos_R] = '\0';
    A_num = atoi(read_data)
  }
                  // Check if download is successful
  If ((A_num && 0x40)==0))  dut_fail_log[a_cnt] = 1;
            // download failure

                  // Reset the TIMs
  Sprintf (send_data, "%s\r\n", "WM_MR");
  PutString(WM_port,send_data);  // send clear relay
  Sprintf (send_data, "%s\r\n", "RM_MR");
  PutString(RM_port,send_data);   // send clear relay
  Sprintf (send_data, "%s\r\n", "DT_MR");
  PutString(DT_port,send_data);   // send reset
}
```