I$_{DDQ}$ Testing and Standby Current Testing with Series 2600 System SourceMeter® Instruments

Introduction

Manufacturers of CMOS integrated circuits and battery-powered electronic products need to measure the quiescent (standby) power supply current to verify quality during production testing. The process of measuring the leakage currents of CMOS integrated circuits or finished products that contain CMOS ICs is known as I$_{DDQ}$ testing. This test requires measuring the current of the V$_{DD}$ power supply while the IC is in the quiescent state. It is done to check for shorted gate oxide and other IC defects that may cause a failure over time. Similarly, the power supply current of battery-powered products that contain bipolar transistors or other ICs can be measured while these ICs are in a quiescent mode. These types of products include portable battery-powered consumer electronics, such as cellular phones, pagers, and notebook computers, as well as implantable medical devices, such as pacemakers and defibrillators. The goal of these tests is to ensure the products satisfy the consumer demand for longer operating periods from a given level of battery charge as well as operational quality. Testing must be performed as quickly as possible to ensure acceptable throughput, but also thoroughly to ensure product quality.

When choosing a measurement instrument to perform these tests, two of the most important considerations are speed and accuracy. However, when measuring small currents, sometimes it's necessary to make a trade-off between speed and accuracy, so custom hardware is often designed to perform these tests. Custom hardware may require lengthy design time and isn't always easy to maintain, whereas commercially available test systems are typically easy to use, readily available, and economical in terms of rack space.

Another Keithley application note (#804) describes how to perform I$_{DDQ}$ testing and quiescent-current measurements using Series 2400 SourceMeter instruments. This application note describes how to perform such tests using Keithley's new Series 2600 System SourceMeter instruments. This next-generation SourceMeter family includes the single-channel Model 2601 and dual-channel Model 2602. With a built-in Test Script Processor (TSP™) and a new inter-unit communication interface (TSP-Link™), Series 2600 instruments offer even more power and flexibility than their predecessors. Keithley can provide an example test script that can perform approximately 2500 I$_{DDQ}$ measurements per second. An electronic copy of this script, which works with both Series 2600 models, is available for download from Keithley's web site, www.keithley.com. A test script to test the current drain of a cellular phone is also available via Keithley's web site.

I$_{DDQ}$ Testing of CMOS ICS

Test Description

This test involves measuring the current draw of the V$_{DD}$ power supply of a CMOS IC when the inputs are at V$_{DD}$ or V$_{SS}$ and the outputs are not connected. *Figure 1* is a diagram of a test setup for a single CMOS inverter. In this example, the Model 2601/2602 is used to source the supply voltage (V$_{DD}$) and to measure the resulting quiescent current.
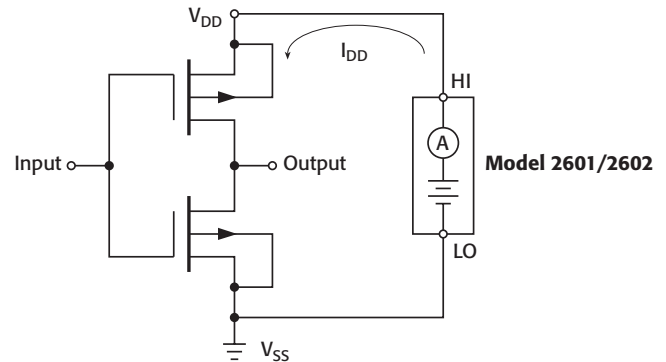
Figure 1. Measuring the quiescent current of a single CMOS inverter

While this example shows an IC with a single gate, many ICs have thousands of gates. Usually, a predetermined series of test vectors (i.e., a pattern of logical ones and zeros applied to the inputs) is used to reduce the number of quiescent current measurements that must be made to ensure that all the gates are toggled or the desired IC logic states are tested.

A constant voltage is applied to the V$_{DD}$ pin(s) of the IC throughout the test to keep the IC in a functional mode. A good CMOS component draws high current from the V$_{DD}$ supply only when switching; in the quiescent state, the current draw is fairly low. Depending on the type of defect involved, the I$_{DDQ}$ of a bad IC will be much higher. To make the measurements, a test vector is applied to the inputs of the IC, and after a specified settling time, the resulting current is measured. After a measurement is taken, it is compared to a preset threshold to determine if the part passes or fails. This threshold is often set at the microamp or nanoamp level and is usually determined by a statistical analysis of I$_{DDQ}$ for several good ICs. As devices continue to become more and more complex, I$_{DDQ}$ testing can't always be performed using a simple threshold test. In some cases, it may be necessary to perform a statistical analysis of the I$_{DDQ}$ data of the device-under-test (DUT) to determine pass/fail status reliably. Series 2600 SourceMeter instruments are well suited for both of these test scenarios.

Test System Configuration

Figure 2 is a Series 2600-based system for testing the I_{DDQ} for CMOS ICs.

As *Figure 2* shows, the HI and LO terminals of the 260X are connected to the V_{DD} and V_{SS} terminals of the CMOS IC. The 260X supplies a constant DC voltage to the IC throughout the test. The inputs of the IC are connected to a "digital test system" that ensures all the gates are toggled or the desired logic states are achieved. It is assumed that this test system also handles the mechanical positioning and probing of the DUT, and the disposition of good and parts.

The 260X can be controlled like a typical programmable instrument by sending it discrete commands via the IEEE-488 (GPIB) or RS-232 bus. Both communication interfaces are standard on the 260X. However, for maximum throughput, a complete test script can be downloaded to the instrument's Test Script Processor, which can then perform the entire test virtually independent of the host PC (system controller). When the 260X is connected to the host controller via GPIB, it can actually control another instrument via its RS-232 port. Thus, the 260X could send ASCII command strings to the digital control system and receive data from it if appropriate.

To enhance speed further, external hardware triggers are used to synchronize the I_{DDQ} measurements with the application of the test vectors. The 260X is equipped with 14 digital input/output lines, which can be used for digital control (pass/fail status in this example) or as input or output trigger lines. The digital test system triggers the 260X when a vector has been sent to the CMOS IC. After the I_{DDQ} value is evaluated, the 260X returns a trigger to the digital test system, which generates another test vector. This process is repeated until all test vectors are generated or the IC fails the test. When the test is finished, the 260X writes a predetermined bit pattern to its digital I/O (DIO) port to indicate the pass/fail status of the part to the digital test system.

In cases where the pass/fail status of an I_{DDQ} test is determined simply by
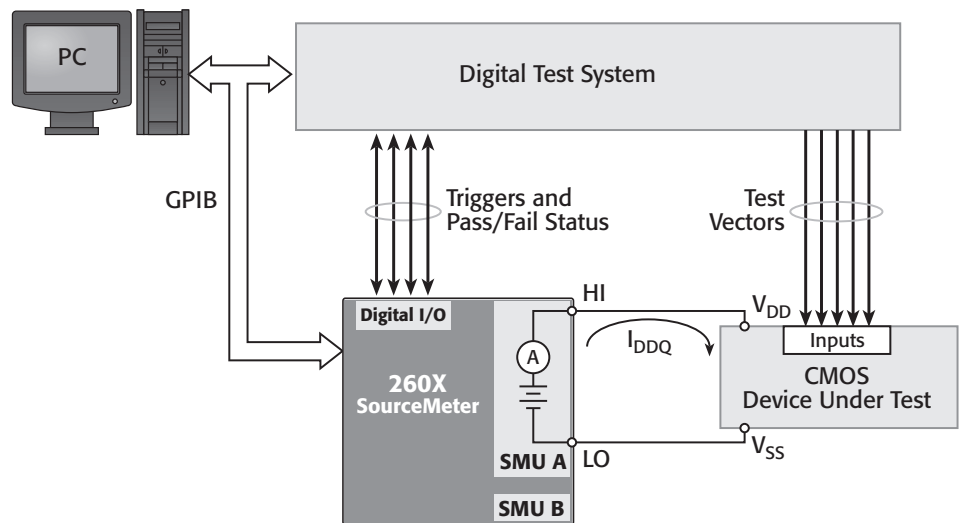

Figure 2. I_{DDQ} test system configuration

comparing the source current to a threshold level, there are at least two ways the 260X can perform such a measurement and inspection. If the actual value of I_{DDQ} is required, then the 260X can measure the current and compare the values to the threshold. If the current exceeds the threshold level, the test fails; otherwise, it passes. The 260X can return any or all of the measured values and pass/fail status to the host PC as required. If the actual value of I_{DDQ} is not required, then the 260X can be configured as a digital comparator for higher test throughput. Set the current compliance limit of the 260X to the threshold value. Apply the test vector and determine the compliance state of the 260X. If the current draw tries to exceed the limit, the 260X it will go "in compliance" and clamp the current at the limit. When this condition occurs, the I_{DDQ} test fails. If the current does not exceed the threshold, the unit won't go into compliance and the test passes. A measurement isn't required to determine the compliance state of the instrument, so this latter method is generally faster than the former.

As previously mentioned, I_{DDQ} testing of complex devices can't always be performed using a simple threshold test. In such cases, it may be necessary to perform a statistical analysis of the I_{DDQ} data for the DUT to determine pass/fail status reliably. The host PC could do this after retrieving all of the test data from

the 260X. However, data transfer is a relatively slow process that can significantly impact test throughput. This can be a high price to pay if there is no requirement to archive the readings. The test script language used to program the Test Script Processor includes a math library and other capabilities that make it possible to perform extensive analyses within the instrument, thereby eliminating the need to transfer all of the data. This is further facilitated by the deep memory of the 260X. Each SMU channel has two non-volatile buffers, which can each hold up to 100,000 readings. Volatile memory is also available for even more data storage.

Remote Operation and Using the Test Script Processor

Before getting into an actual test example, let's discuss the new instrument feature that will be used for the test application. Series 2600 System SourceMeter instruments have a powerful embedded computer or Test Script Processor, which offers capabilities never seen before in rack-and-stack instruments. A complete test program (script) can be downloaded to the TSP. As with other common programming languages, a well-designed script creates reusable functions or subroutines, which can be called by a test executive or other functions. It's possible to pass parameters to these functions. In the I_{DDQ} test example presented here, a single function is created to perform the

I_{DDQ} test, inspect the data, and return the results. This function also handles all of the trigger synchronization and other digital I/O between the digital test system and the 260X. This function can be called by the system controller PC or from another test function that resides in the TSP. The script, which creates the function, must be downloaded to the 260X using either GPIB or RS-232. When the script is first downloaded, it is stored in volatile memory. However, it can be saved to nonvolatile memory if desired. The script must be run to create the test function. This function always resides in volatile memory, which means it must be recreated whenever power is cycled. The creation script can be explicitly run at any time or it can be set to run automatically at power-up. When the system controller PC calls the function, the 260X will execute the complete test sequence for the specified number of parts without any further intervention by the system controller, thereby saving communications time and increasing system throughput.

A script can be created using any text editor. However, Keithley provides a free application called Test Script Builder, which can be used to create, debug, and organize scripts. Test Script Builder can download scripts to volatile instrument memory and save them in nonvolatile instrument memory. It can also run the scripts. Scripts can also be loaded and run using applications created in other languages, such as Visual Basic®, Visual C/C++®, or LabVIEW®. Once the script is in memory, it can even be run from the front panel.

All measurements, calculations and inspections can be performed by the 260X, so it's unnecessary to send data to the host computer (system controller) for processing. However, it's possible to do so if desired for recordkeeping or other purposes. As seen in the example script, "print" statements are used to send data back to the host computer. The data listed in the print statement is placed in the instrument's output queue for retrieval by the host.

Application Example 1: I_{DDQ} test

In this example, 2000 test vectors will be sent to a CMOS IC that must be powered with a constant 2V. A threshold level will be used for this example and a quiescent current of $1\mu A$ or less will be considered an acceptable current. The actual measurements need not be retained, so only the compliance state of the 260X will be inspected. To ensure acceptable production throughput, measurements must be made as quickly as possible, preferably in one second or less. The measurement instrument must send a pass or fail indication to the digital test system.

Solution: A TSP script, which creates the function "IddqTest(smu, ndevices, nvectors)," was developed. Portions of this script are listed below. A complete copy of the script can be downloaded from Keithley's web site, www.keithley.com. The script can be viewed, edited, loaded, and executed using Test Script Builder. An actual "digital test system" wasn't available, so the speed performance of the function was evaluated by a wrap around test using the trigger lines that would normally synchronize the I_{DDQ} measurement with the application of the test vector. The input trigger line of the 260X was connected to its output trigger line. Under this condition, the 260X was able to execute the I_{DDQ} test at a rate of approximately 2500 vectors per second.

The function definition is listed below. The actual script includes many more comments in addition to those shown here. Comments are identified by a double dash (--). Several local variables are declared within the function. All variables are global unless they are explicitly declared as local. The function performs some initial instrument setup before it executes the actual test. This setup includes setting the 2V source level and $1\mu A$ compliance limit, selecting the voltage sense mode, and configuring the trigger lines. The example uses local (two-wire) voltage sensing. It is straightforward to change it to remote (four-wire) sensing.

```
function IddqTest(smu, ndevices, nvectors)
    -- Pass parameters:
    --  smu is the SMU to use for the test (A or B)
    --  ndevices is the number of ICs to test
    --  nvectors is the number of vectors used for Iddq test sequence


    -- Default to smua if no smu is specified.
    if smu == nil then smu = smua end

    -- ***** Declare and initialize temporary variables *****

    -- Variables to hold boolean status of "wait for triggers"
    local l_sot_received
    local l_trig_received

    -- Abort test flag (boolean)
    local l_abort_test

    -- Variables to hold timing information
    local l_start_time, l_stop_time, l_elapsed_time

    -- Counter variables
    local l_i, l_nvectors_remaining

    -- Table used to simulate bins of a component handler
    local l_bins = {0,0} -- Initially set all bins to zero

    -- *************** Perform initial setup of the 260X ***************

    smu.reset()   -- Reset SMU to default settings
    smu.source.func = smu.OUTPUT_DCVOLTS   -- Source DCV
    smu.source.rangev = 2          -- Will automatically select 6V range
    smu.source.levelv = 2          -- Source 2V
    smu.source.limiti = 1E-6       -- Set current compliance to 1uA
    smu.sense = smu.SENSE_LOCAL    -- Use smu.SENSE_REMOTE for 4-W sensing
```

```
-- Configure Digital I/O Port

digio.writeprotect = 0          -- Unprotect all bits
digio.writeport(30)             -- Set bits/lines 2, 3, 4 and 5 high
digio.writeprotect = 30         -- Write protect trigger lines 2, 3, 4 and 5

-- Configure trigger line 2 (input SOT)
digio.trigger[2].mode = digio.TRIG_FALLING      -- Detect falling edge
digio.trigger[2].clear()                        -- Clear "latched" triggers

-- Configure trigger line 3 (output EOT)
digio.trigger[3].mode = digio.TRIG_FALLING      -- Output TTL-low pulse
digio.trigger[3].pulsewidth = 10E-6             -- Guaranteed minimum pulse

-- Configure trigger line 4 (input TRIG when test vector is set)
digio.trigger[4].mode = digio.TRIG_FALLING      -- Detect falling edge
digio.trigger[4].clear()                        -- Clear "latched" triggers
-- Configure trigger line 5 (output TRIG after Iddq measure is complete)
digio.trigger[5].mode = digio.TRIG_FALLING      -- Output TTL-low pulse
digio.trigger[5].pulsewidth = 10E-6             -- Guaranteed minimum pulse

-- Clear the error queue
errorqueue.clear()

-- ************************ RUN TEST ************************

-- Display some status info on front panel
display.clear()
display.setcursor (1,1)
display.settext(" Test In Progress")
display.setcursor (2,1)
display.settext(" Testing "..tostring(ndevices).." Parts")
timer.reset()
l_start_time = timer.measure.t()

for l_i = 1, ndevices do

    -- Wait for SOT indicating part is ready to test; timeout after 10ms
    l_sot_received = digio.trigger[2].wait(0.01)

    -- Turn ON SMU output; stays on until test is completed
    smu.source.output = smu.OUTPUT_ON

    -- Initialize abort flag and vector counter
    l_abort_test = false
    l_nvectors_remaining = nvectors

    digio.trigger[4].clear()   -- Clear any "latched" triggers

    -- Repeat test until no more test vectors or test fails
    while (l_nvectors_remaining >0) and not(l_abort_test) do

        -- Wait for trigger from digital test system; timeout after 10ms
        l_trig_received = digio.trigger[4].wait(10E-3)

        -- delay(0.0005)       Insert delay here if settling time is required

        -- Check compliance state; returns boolean true or false
        l_incompliance = smu.source.compliance

        -- If source is in compliance, then fail part and abort test
        if l_incompliance then

            digio.writeport(32) --Write FAIL pattern to DIO
            l_bins[2] = l_bins[2]+1      -- "Bin" the part
            l_abort_test = true -- Set abort test flag TRUE to exit loop

        else    -- Otherwise part is still good so continue test

            digio.trigger[5].assert()    -- Output trig to "dig test system"
            l_nvectors_remaining = l_nvectors_remaining - 1      -- New count
        end --if
```

```lua
   end --while
   -- If did not abort test, then part is GOOD
   if not(l_abort_test) then

      digio.writeport(64)              -- Write PASS pattern to DIO
      l_bins[1] = l_bins[1]+1          -- "Bin" the part

   end --if

   --Turn OFF SMU output
   smu.source.output = smu.OUTPUT_OFF

   -- Output EOT trigger
   digio.trigger[3].assert()

   -- Clear binning code (set all unprotected bits to zero)
   delay(0.0001)                -- Delay in seconds before clearing binning code
   digio.writeport(0)

   end --for

   l_stop_time = timer.measure.t()
   l_elapsed_time = l_stop_time - l_start_time

   -- Display throughput rate and final binning results on 260X front panel
   display.clear()
   display.setcursor (1,1)
   display.settext("Parts per sec = "..tostring(ndevices / l_elapsed_time))
   display.setcursor(2,1)
   display.settext("Bin Count: Good= "..l_bins[1].." Bad= "..l_bins[2])

   -- Write speed & binning results to output queue for retrieval by host PC
   print("Elapsed time = "..l_elapsed_time.." sec")
   print("Parts per sec = "..tostring(ndevices / l_elapsed_time))
   print("Bin Count: Good Parts: "..l_bins[1].." Bad Parts: "..l_bins[2])

end --function IddqTest
```

Running the example I_{DDQ} test script using Test Script Builder or another application only creates the function; it doesn't perform any tests. Executing the I_{DDQ} test requires calling the IddqTest() function. For example, to test 100 devices using 2000 test vectors, the system controller must send the command "IddqTest(smua, 100, 2000)." In response to the function call, the 260X waits for a Start-of-Test (SOT) trigger from the external digital test system for each DUT. When the 260X gets the SOT, it turns on the SMU output and waits for a trigger from the digital test system. The digital test system applies a test vector to the inputs of the IC and then sends a trigger to the 260X. Upon receipt of the trigger, the 260X waits a predetermined settling time and then checks its compliance state. If it is NOT in compliance, that particular I_{DDQ} test passes and the 260X outputs a trigger to the digital test system and loops around and waits for the next vector to be applied. This process continues until all test vectors are completed or the instrument goes into compliance. If all test vectors are completed successfully, the 260X writes a decimal 64 to its DIO port to indicate a PASS to the digital test system. The table element "l_bins[1]" is incremented by one to simulate a part being binned. The 260X then outputs an end-of-test (EOT) trigger to the digital test system, indicating that the I_{DDQ} test sequence for the DUT has been completed.

If the 260X goes into compliance, the I_{DDQ} test fails. When an individual test fails, the 260X uses an "immediate" binning scheme, which means it immediately writes a FAIL bit pattern (decimal 32) to the digital test system, aborts the remainder of the test sequence and then outputs an EOT trigger to the digital test system. After the FAIL pattern is output, the table element "l_bins[2]" is incremented by one to simulate the binning process. The 260X clears the PASS/FAIL bit pattern a predetermined time interval after it issues the EOT trigger by writing a decimal 0 to its DIO port. If there are more DUTs to be tested, the 260X loops around and waits for the next SOT trigger. Once all tests are complete, the throughput rate and binning results are displayed on the 260X front panel; they are also printed to the output queue for retrieval by the system controller.

Standby Current Testing

Test Description

This test involves measuring the current of a battery-powered product while it is in a standby state. A typical test setup is shown in *Figure 3*.

In this example, the HI and LO terminals of the 260X are connected to the HI and LO battery terminals of the product to be tested. The 260X's voltage source simulates the internal battery of the product and its ammeter measures the current while the part is in a standby mode. This measurement is compared to a specified limit to determine if the product passes or fails. An
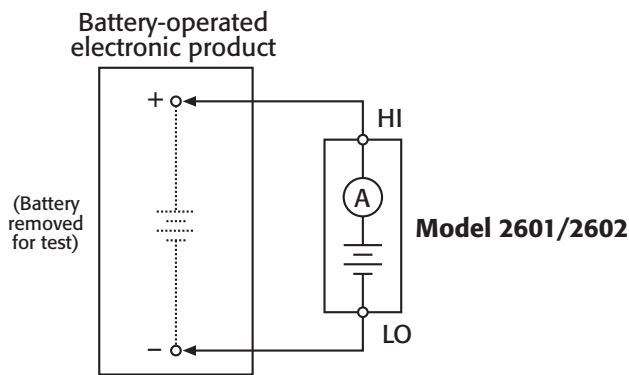
Figure 3. Testing the standby drain current of a battery-powered electronic product

acceptable current is often in the milliamp or microamp range, but can vary greatly depending on the application.

Test System Configuration

Figure 4 is a block diagram of a 260X-based standby current production test system. The battery-powered product is placed in a test fixture connected to the 260X. When triggered, the 260X outputs a voltage and measures the resulting current. As *Figure 4* illustrates, the Model 260X has both an IEEE-488 and RS-232 communication ports, as well as a digital I/O port. When the 260X is connected to the host PC via GPIB, it can communicate directly with the handler via RS-232. The digital I/O port can also send signals directly to and receive signals from the handler. The measured current of the DUT is compared to limit values that have been preset in the 260X. A TTL level signal generated from the 260X's digital I/O port indicates whether the device passed or failed the tests. Based on the received signal, the automated handler will route the device to the appropriate bin or other location.

Application Example 2: Testing the current draw of a cellular phone

In this application, the 260X takes the place of the cellular phone's rechargeable battery and measures the current while the phone is in the "talk," "standby," and "off" modes. A 4.5V test voltage is used for all three measurements. For this test, a cellular phone fails if the measured current exceeds the threshold levels for the given mode. Each threshold current has its corresponding pass and fail bit patterns:

Mode	Threshold Current	Pass Bit Pattern	Fail Bit Pattern
Talk	400mA	0001	0010
Standby	10mA	0001	0100
Off	$100\mu A$	0001	1000

Solution: A TSP script that creates functions to test the current draw of a cell phone was developed. The script is similar to one for the I_{DDQ} test, except the source current is actually measured and inspected instead of simply using the compliance as the threshold limit. A complete copy of the script can be downloaded from Keithley's web site, www.keithley.com. It can be viewed, edited, loaded and executed using Test Script Builder.

Equipment List

The following equipment is required to assemble the test systems described in this application note and run the example scripts available from Keithley:

1. Keithley Model 2601 or 2602 SourceMeter instrument

2. "Digital test system" and/or component handler with test fixture

3. IEEE-488 (GPIB) Interface Card (KUSB-488, KPCI-488 or equivalent)

4. Keithley 7007 IEEE-488 interface cables

5. Custom DB-25 digital I/O handler interface cable to interface the instrument to the digital test system and/or handler

6. Test leads to connect the instrument to the test fixture

Alternative Solutions

The 260X ammeter can measure current from 3A full scale to 100nA full scale with 10ppm resolution on any range. If greater current resolution is desired, an instrument with a more sensitive ammeter must be used. Examples are the Model 6430 Sub-Femtoamp Remote SourceMeter instrument and the Model 236, 237 and 238 Source-Measure Units. However, due to the greater sensitivity of these instruments, the test will run more slowly than when the 260X is used. If currents greater than 3A must be measured, the Model 2440 5A SourceMeter instrument can be used.

Another alternative solution is to use a readback power supply for both sourcing voltage and measuring current. However, the readback resolution of most programmable sources is in the milliamp range, so they can't be used for low current tests. In addition, these supplies often have sense lines with relatively low input impedance and correspondingly high current (microamp range), which can affect measurement accuracy.
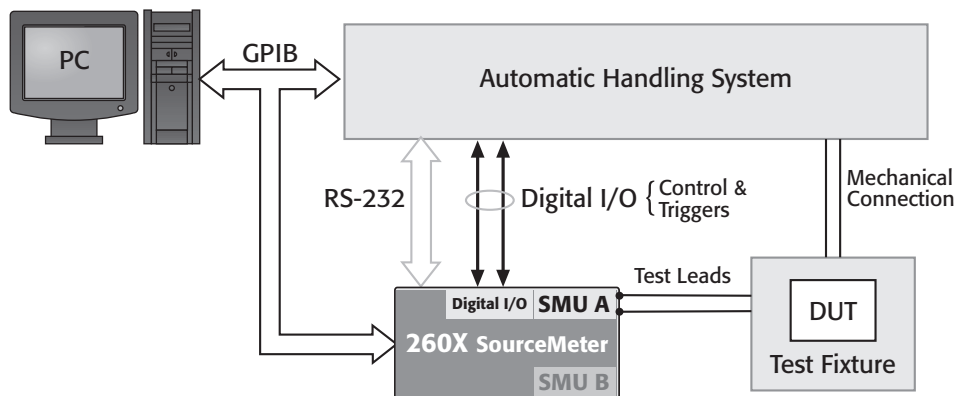

Figure 4. Block diagram of a 260X-based standby current production test system.

Test System Safety

Many electrical test systems or instruments are capable of measuring or sourcing hazardous voltage and power levels. It is also possible, under single fault conditions (e.g., a programming error or an instrument failure), to output hazardous levels even when the system indicates no hazard is present. These high voltage and power levels make it essential to protect operators from any of these hazards at all times. Protection methods include:

- Design test fixtures to prevent operator contact with any hazardous circuit.

- Make sure the device under test is fully enclosed to protect the operator from any flying debris. For example, capacitors and semiconductor devices can explode if too much voltage or power is applied.

- Double insulate all electrical connections that an operator could touch. Double insulation ensures the operator is still protected, even if one insulation layer fails.

- Use high reliability, fail-safe interlock switches to disconnect power sources when a test fixture cover is opened.

- Where possible, use automated handlers so operators do not require access to the inside of the test fixture or have a need to open guards.

- Provide proper training to all users of the system so they understand all potential hazards and know how to protect themselves from injury.

It is the responsibility of the test system designers, integrators, and installers to make sure operator and maintenance personnel protection is in place and effective.

For Further Reading

D. Leslie, "QTAG: The Evolution of a Standard Monitor: A Progress Report," *Evaluation Engineering*, pp 26-32, Oct. 1995.

M.J. Riezenman, "Technology 1996: Test & Measurement," *IEEE Spectrum*, pp. 65-69, Jan. 1996.

S.S. Sabade and D.M. Walker, "I_{DDX}-based Test Methods: A Survey," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 9, No. 2, pp 159-198, April 2003.

J.M. Soden and C.F. Hawkins, "I_{DDQ} Testing and Defect Classes — A Tutorial," in *Proc. of Custom Integrated Circuits Conf*, 1995, pp. 633-642.

KEITHLEY