Build a microcontroller-based functional tester

Save money by embedding test capabilities into fixtures, enclosures, or larger systems.

Overton Claborne, Overton Instruments

A typical PC-based test system may include analog and digital I/O cards and one or more communications buses that let you control external instruments. Budget restrictions, however, may force you to design your own functional tester based on a low-cost microcontroller. If you go the microcontroller route, you can apply such a system in engineering evaluation, production test, or quality assurance for testing components, semiconductors, PCB's, hybrid modules, cable assembles, and other devices. You can integrate the tester into a custom instrument enclosure, a mechanical test fixture, or larger ATE system.

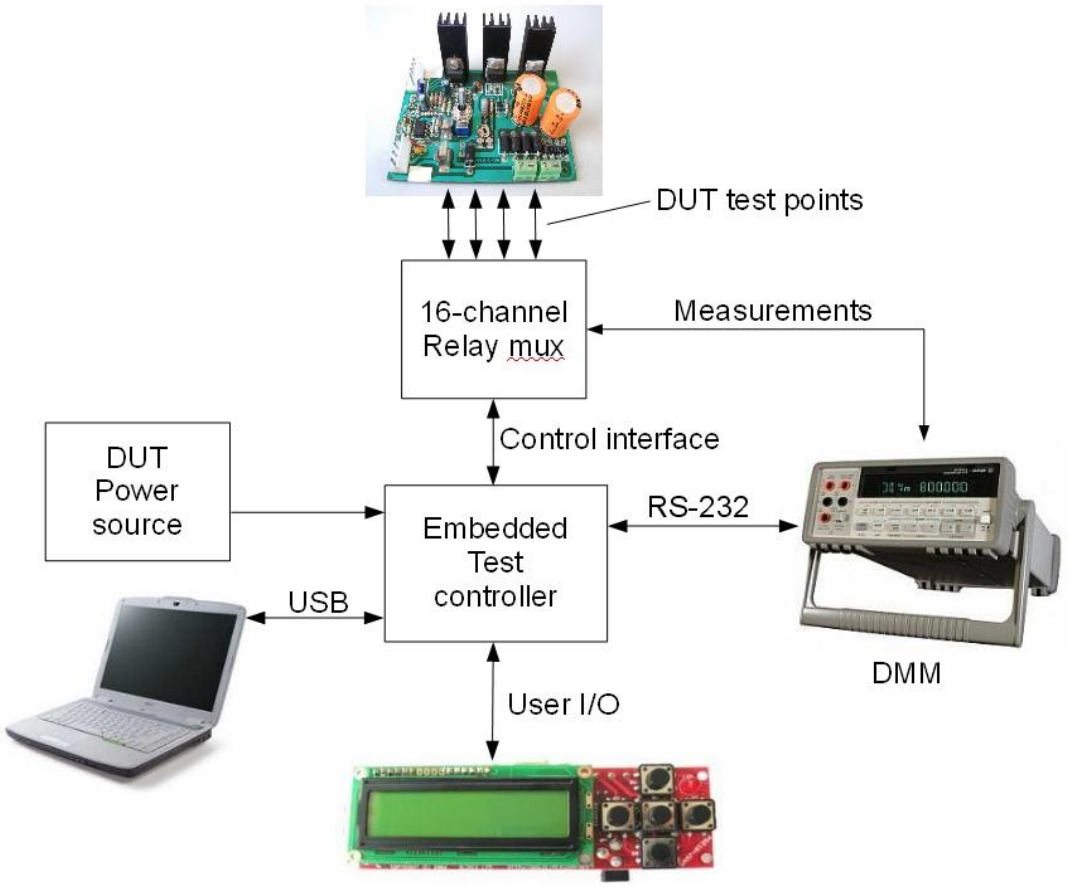

Figure 1. A test controller communicates with User I/O, the DUT, a DMM, and a PC (for program development).

Figure 1 shows how you might configure a microcontroller-based tester, which you can build for less than $500 plus the cost of any external instruments, to test a PCB. The system

includes the microcontroller board, a relay multiplexer (mux) board, a user I/O board, and an external DMM. You can write test software using high-level languages so there's no need to learn assembly code. **Figure 2** highlights the controller board, which uses a $24 Teensy++ 2.0 development module (www.pjrc.com/teensy). The Teensy module provides a highly integrated Atmel processor that includes a wide array of digital and analog resources. The module's 46-pin DIP package lets you easily integrate it into a custom test system. In effect, your tester's controller becomes the carrier for the Teensy module.

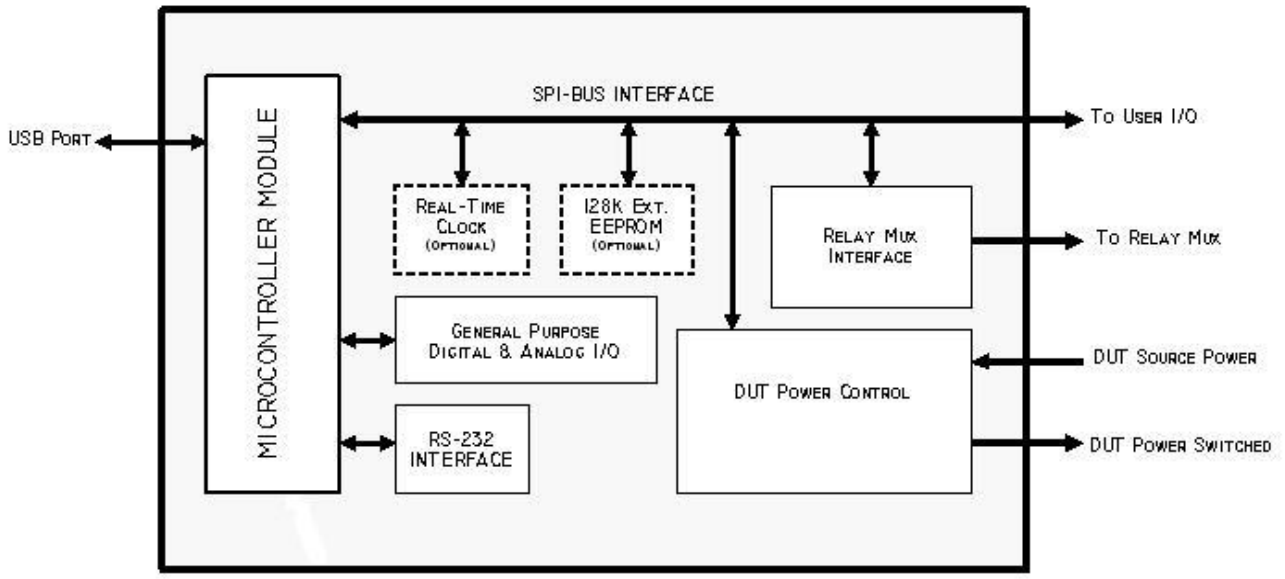

Figure 2. An embedded controller consists of a microcontroller and serial buses for user I/O and for communication with external test equipment.

DUT power

Test systems often need to supply power to the DUT (device under test). You can power your DUT through a DPDT (double pole, double throw) relay. You often need to measure the DUT's power consumption. The microcontroller's 10-bit ADC can measure DUT's current consumption and source voltage and calculate power. The ADC measures current through a high-side shunt-measurement circuit that produces an output voltage proportional to current. You can also measure the DUT source voltage with a voltage-divider network. A 1.25 V precision voltage reference sets the ADC's voltage range.

The test system needs the mux to connect test points to an external test instrument. A 16-channel relay mux with DPST relays is enough for many test applications. Relays rated at 30 VDC@1 A/125 VAC@3 A will accommodate most low-power measurements. The microcontroller drives the mux through a 10-pin interface, which can carry SPI-bus control

lines, chip-select logic, power, and ground.

You'll likely need a test instrument such as a DMM. The system in Figure 1 uses an Agilent Technologies 34401A, but you can use any DMM with an RS-232 port. Even a handheld DMM with a serial port might work for you. The relay mux can connect your DUT test points to the DMM's input jacks. After setting the mux channel, the controller can configure the instrument using ASCII commands such as 'CONF:VOLT:DC' and make a measurement with the 'READ?' command. Furthermore, the microcontroller can trigger the external instrument through hardware or software.

A test system's user I/O lets a user control a test or get test status. **Figure 3** shows an example of a control panel for a basic operator interface. At the very least, the tester must indicate a pass or fail test result. A simple indicator may consist of a green LED for pass and a red LED for fail. You'll also need to initiate or abort the test sequence. In those cases, two pushbutton switches will do the job. You should consider adding a third LED that illuminates while a test runs.

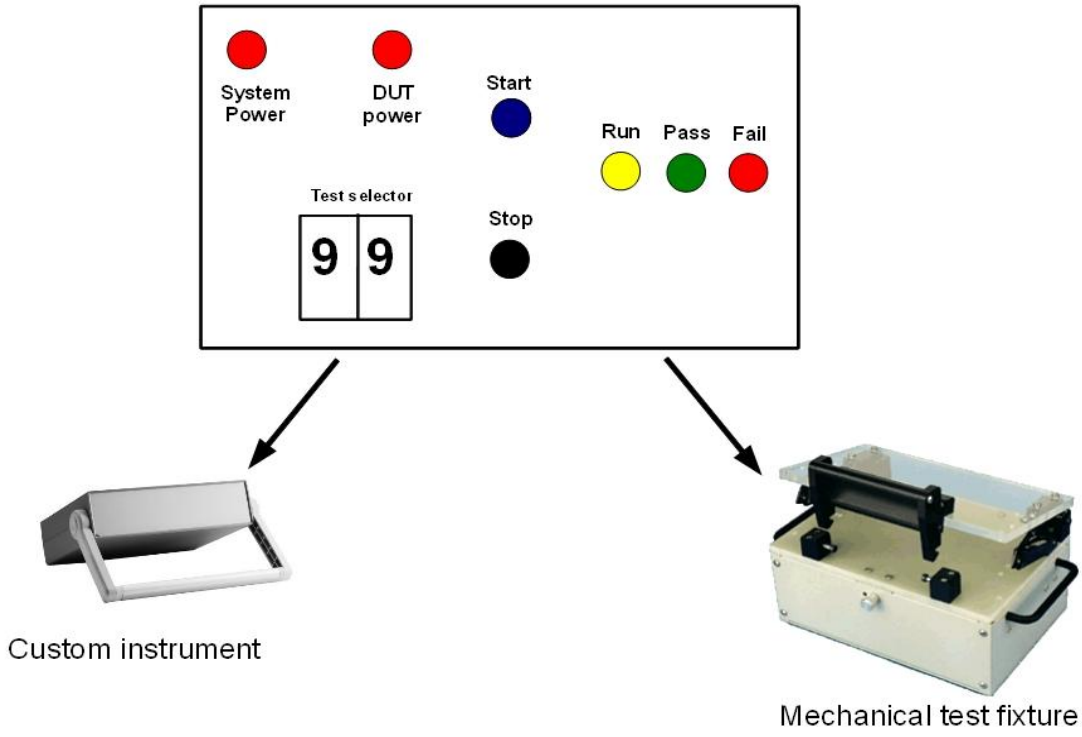

Figure 3. A user I/O panel lets operators run automated tests.

You can add an LCD display module if the user needs more information such as test status or error messages. A 2-line-by-16-character unit works well. The microconstroller can communicate to the LCD display over its SPI bus. Furthermore, you may need to vary the test sequence slightly to accommodate different configurations for a family of DUTs. You can use dual BCD thumbwheel switch to select a test sequence. Finally, the panel should inform the user if DUT power is active. I recommend a bicolor LED to indicate green (for DUT power-ON) and red (for DUT power-FAIL).

You should use a separate PCB for your user I/O functions. It can attach directly to the LCD interface pins (or you can mount it to the front panel itself). The microcontroller can directly drive the LCD. Also, the UIO board has two 8-bit DIO port IC's, which the controller uses to gain access to the remaining components on the front panel--the start and stop switches, and pass/fail LED's). A buzzer provides audible feedback and it can announce alarms to the user.

Add features

The system I've described so far uses about 60% of the Teensy microcontroller module's resources. You can add features that attain more flexibility, control, and communications. For example:

- Incorporate an EEPROM with at least 128 kbytes of storage. With it, you can store test limits and test results and transfer them to a PC for analysis and archiving.
- Install a battery-backed real-time clock circuit, which lets you timestamp test results.
- Add an Ethernet module to quickly gain access to the network. You can then upload and download data and programs as required.
- If you a need to control external test instruments or you need a bar-code scanner or printer, then simply add a second RS-232 COM Port with a COM module.
- Upgrade the resolution and accuracy when measuring DUT current flow and source voltage by adding a dual channel 12-bit ADC circuit to the microcontroller.
- If your DUT has more than 16 test points, simply add more relays. With proper consideration for the PCB layout constraints, you can double the number of relays to 32 channels. You may have to use an additional relay driver circuit, though. If you need even more relays, then you either add more relay mux boards or use smaller relays on your board.

Software development

No automated tester is complete without software. Your system software should, at a minimum, include the means to manage the operator Interface, execute the test sequence, control the instrumentation, determine Pass/Fail and log the test results. **Figure 4** highlights the functional blocks for the test-system software. If you develop software for PC-based test equipment, you'll appreciate the similarities.

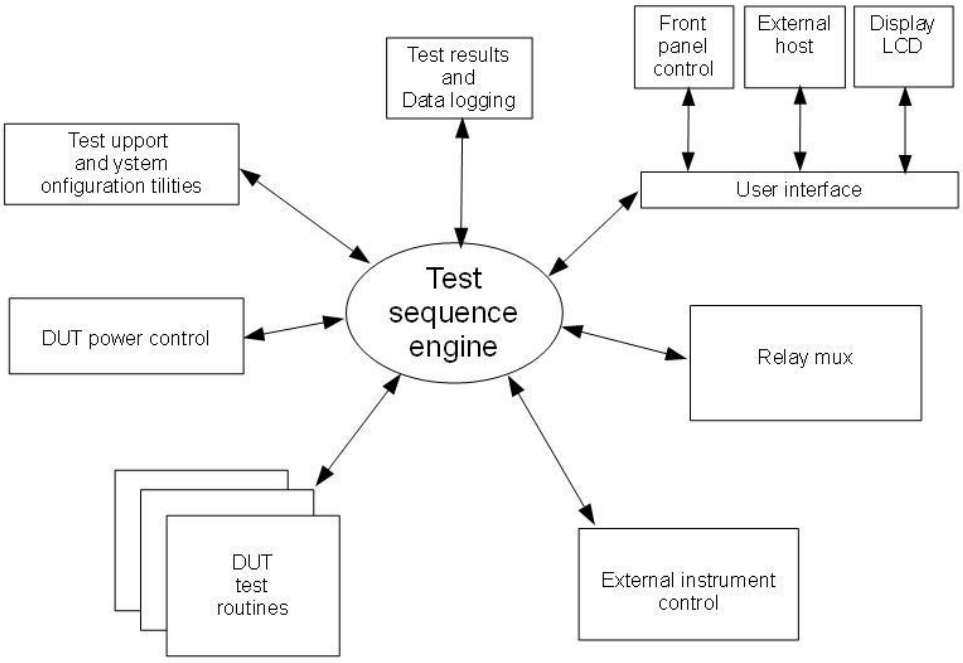

Figure 4. The test sequence engine calls routines that control the tester's subsystems.

I recommend that you build test programs using a “top-down” approach. The development process involves three stages: pre-test, test sequence, and post test (**Figure 5**). During pre-test, the system should perform all the necessary tasks needed to support the upcoming test sequence, which includes initialize the software and hardware resources, prompt the operator, and check the start button and other fixture-support functions. During the test, the system should follow a fixed test sequence, which should include:

- Display the relevant test info,
- Retrieve the test limits,
- Configure the test instruments,
- Acquire the measure,
- Determine pass/fail, and
- Log the test results

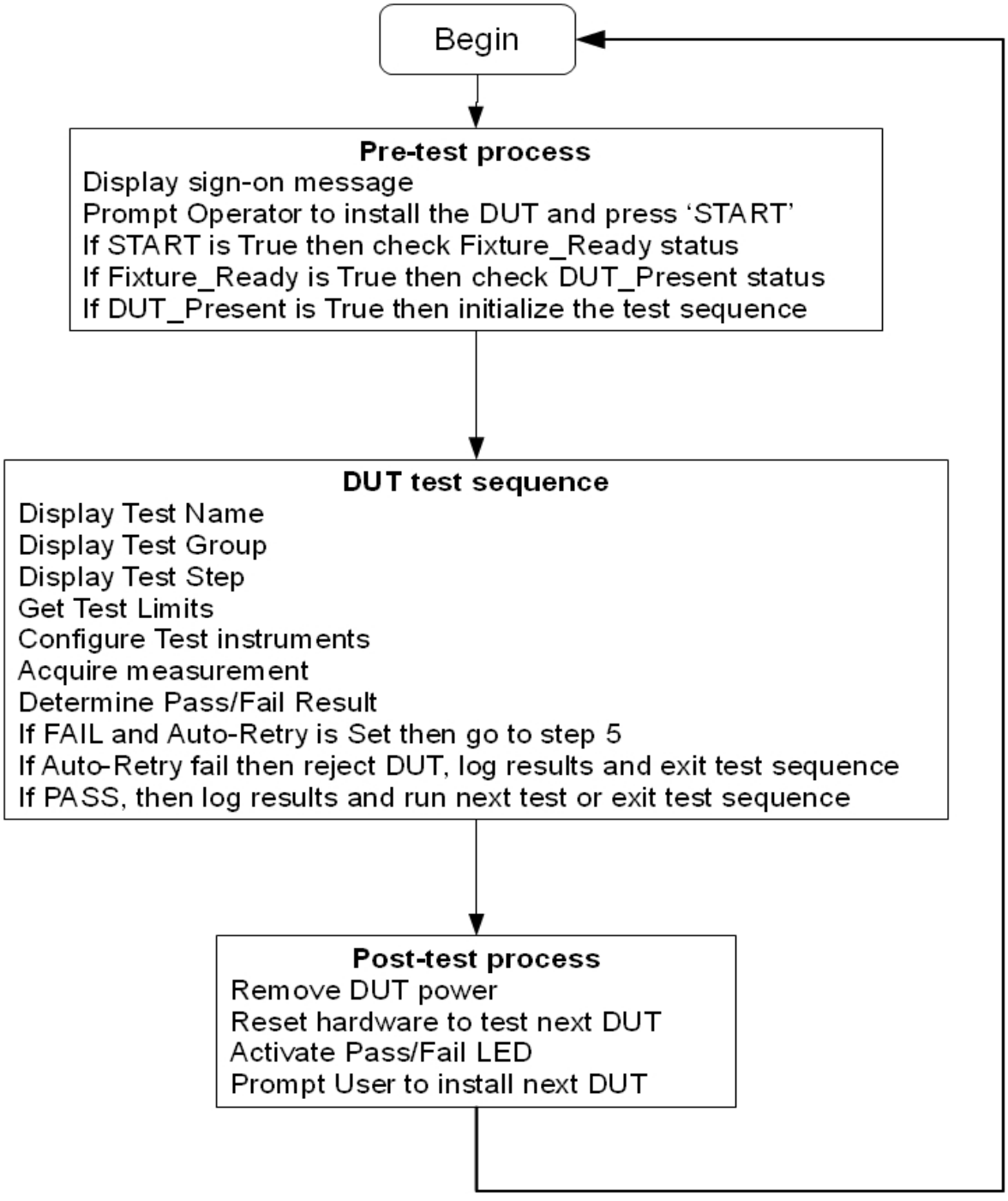

Figure 5. Test software consists of pretest processes, test sequences, and post-test processes.

Following the test, the system should prepare to repeat the test cycle, which includes resetting DUT power, indicating pass/fail status, and prompting the operator to install the next DUT.

Develop a test sequence

You don't need to write your test code in assembler. In fact, you can write code in BASIC using a compiler such as BASCOM-AVR from MCS Electronics. For about $130, you can get a compiler that lets you control all aspects of the system hardware including the Teensy controller. The compiler lets you create and edit programs on a PC. Once completed, the compiler creates object code from your source code and downloads it to the Teensy controller.

If you prefer to write your code in C or C++, then I recommend WinAVR, a suite of executable, open-source software-development tools for the Atmel AVR series of RISC microprocessors hosted on the Windows platform. WinAVR includes the open-source GNU GCC compiler (gcc.gnu.org) for C and C++.

When building a test software application, you will likely put most of your effort into the DUT test-sequence routines. The tester should verify that the DUT is free of any manufacturing related defects before it applies power to the DUT. For that you need some in-circuit tests. Define key test points such as checking for shorts on the power-rails, missing and wrong value components, or misconfigured circuits. You can use the DMM to run these tests, just plan your relay allocation so that you have enough available for these tests. Then, configure the DMM to measure resistance, capacitance, diodes, or continuity as needed.

Before applying DUT power, engage the discharge circuit to bleed-off any lingering current or voltage. This assures the DUT will power-up in the proper state. After applying power, measure DUT current with the DMM and compare it to a preset limit. If the DUT draws too much current, remove power and alert the operator. If the DUT current draw is within limits, then use the DMM to measure important power points such as power-rails and voltage references.

Build your own

To develop your own microcontroller-based system, you'll need some hardware design and schematic capture experience. I used Eagle CAD software (www.cadsoftusa.com), to create the schematics for my controller board, relay mux, and user I/O board. Once you have hardware, you'll need to program it. If you're new to programming a microncontroller, then I suggest you start with the ATMega8 Educational Board ($35.90 from Futurlec). You can also find many books on the subject, but you might consider purchasing the *Atmel AVR Microcontroller Primer* ($31.58 from Amazon).

First, decide whether you will build the proposed ATS system described here, or if you need more circuits. Next, decide if you'll use the schematics I've provided or crete modified versions or custom circuits for the controller, relay mux and user I/O boards. You should be able to send the design files to any competent board house (with Eagle CAD experience) to fabricate your circuit boards.

Assembly and Test

After receiving the PCBs, you can order parts and assemble the boards. CAD software can create a complete net list and bill of materials. So can save time and effort by using a contract manufacturer to fabricate the PCBs and assemble the parts. Once you have assembled the PCBs, you should run tests, then apply power and check for proper operation before you connect it to a DUT to a load. Use components such as resistors to simulate the load that the DUT will place of your tester before you risk damaging the DUT.

Now that you have confirmed the boards work properly, you can decide when to use the tester system to in an application. In my experience, the general rule for deciding between building a traditional PC-based test system or a microcontroller-based system, it all comes down to cost, complexity and urgency. If you need a simple low-cost tester, then consider building your own. *T&MW*

Overton Claborne i*s president of Overton Instruments where he develops custom test equipment using embedded systems. E-mail: overton@microate.net.*