

## LESSONS FROM THE TRENCHES

George Martin

### Manufacturing Testing

When producing a large quantity of products, you can easily justify investing in the testing equipment that is needed. But, what if you're only producing a small quantity? Is the cost really justified? This month, George finds himself in such a situation and explores the option of writing his own test software.



I'm trying to concentrate on writing this article in the wake of the attacks on

New York and Washington. I wanted to say something about the events but can't seem to find the words. Hopefully by the time you read this some sort of normalcy will be restored.

If you're doing a design with one of the current crop of embedded microprocessors, you are likely to be using a background debug module (BDM) pod for development. The newer processors are so compact and complicated that they no longer work with the classic emulators. The old-school emulators used hand-picked (the fastest) versions of the processors with special bonding options. This permitted the various busses to be monitored and, when data patterns were found, break points could be enforced. Needless to say, these emulators were and are quite expensive. On the other hand, the BDM units work with logic built into each micro. This logic can perform the break point and single step functions that

are needed to have a useful emulator.

All of this is good news. It means that instead of \$30,000 for an emulator, you only need to spend perhaps \$500 to \$5000. The supporting software also has advanced, so you can single step in your high-level language and view all of your data, including local and global variables. You can even peer into all of your data structures. This is a great advancement.

One of the features that was lost by going to the BDM pods was the ability to map memory in either the debug hardware or the user hardware. I have no idea how you would perform this mapping using a BDM pod. This means you need working hardware in order to get the debugger working. A catch-22 perhaps? Well, it's not so bad because all you need to get the micro running is power and clock. And, many of these micros have built-in RAM, so some code can be run as the new hardware is brought up and running. If you're building large quantities of boards, a manufacturing-defects test fixture and test set is easily justified. However, if you build in lots of 25 or 100 then justifying that investment would be extremely difficult. This is exactly the situation I find myself in this month.

#### THE PROBLEM

I've got a design using one of the Motorola Coldfire micros being built in small quantities. I've got four to six units that will not pass the production procedures for loading code and operating in a system.

There are several methods that can be used to get these units up and running. First and foremost is a visual inspection (i.e., looking for wrong parts, parts oriented incorrectly, and of course, shorts and opens on the printed circuit board). The visual inspection could be done by an assembly-level person using the proper documentation. If one of these problems

is found, there may be the same problem on several other units. Or, perhaps there's an area on the board prone to develop solder shorts.

The next method is comparing a working board to a non-working board, which could be performed by a technician using test equipment. This would work well if the micro came up and ran but an isolated feature did not function properly. Perhaps one of the serial ports wouldn't receive data.

The final method is to sit a hardware design engineer down with the BDM pod and a set of schematics looking for problems. If the design is new, perhaps there are some marginal timing or signal requirements that are not completely understood. This is an expensive proposition and, if there's no design defect, probably one that you can't justify. Add up the numbers and you'll see how much it really might cost.

This is the problem I find myself with. This product line will never see high quantities and build lots will be around 10 to 30 units. Let's explore the option of writing test software to diagnose the defective units.

## A POSSIBLE SOLUTION

The design uses a Motorola Coldfire 5602e. So I hit the Motorola site ([http://www.motorola.com/webapp/sps/library/pr od\\_lib.jsp](http://www.motorola.com/webapp/sps/library/pr od_lib.jsp)) and selected the specific device. There I found the following.

"The MCF5206 integrated micro-processor combines a ColdFire processor core with several peripheral functions such as a DRAM controller, timers, parallel and serial interfaces, and system integration. Designed for embedded control applications, the ColdFire core delivers enhanced performance while maintaining low system costs. To speed program execution, the on-chip instruction cache and SRAM provide single-cycle access to critical code and data. The MCF5206 processor greatly reduces the time required for system design and implementation by packaging common system functions on-chip and providing glueless interfaces to 8-, 16-, and 32-bit DRAM, SRAM, ROM and I/O devices."

You can also take a look at the

Motorola Product Brief for this processor in PDF format (download PDF).

The specific design I'm going to discuss has DRAM, flash memory and battery-backed-up SRAM (BBSRAM), and the memory devices. Peripherals range from simple (such as a latch to hold LED drive signals) to complex (an HDLC controller with DRAM memory attached to the back side of that controller). I just happened to have defective boards that have problems in each of these areas.

I wanted to load code that would execute test routines to exercise all the signals lines associated with each of these devices. Clearly, if all of the memory is operating, code to exercise the HDLC controller would be just a matter of design, code, and test. But, what happens when the DRAM is not functioning? How do you load any code?

The '5206 fetches vectors from address 0x0000 at power on reset. So, you need memory located at that address to load the code you need to execute. Also, that memory type needs to be RAM so that you can load code into that memory. In production, the flash memory is located at 0x000, and vectors and code are fetched from the flash memory.

The chip selects are controlled by configuration registers. Powerup defaults for these registers are detailed in the processor's users manual. The processor starts up after a power on reset, and the flash memory is connected to the boot device chip select pin. The powerup default settings are such that data can be fetched using an 8-bit wide data bus with maximum wait states.

Using the BDM pod, you can reset the processor and change the contents of the configuration registers such that the boot device is pointing to either BBSRAM or DRAM. If the BBSRAM or DRAM is working, you can load and execute code. If either the BBSRAM or DRAM is in trouble, you can use the other. If both are in trouble, you could program the flash memory and attempt to execute code from it on powerup. Unfortunately, that's not a workable solution in my

case because the flash memory devices are soldered to the board.

The final option would be to map any internal SRAM to 0x0000. The users manual for the MCF5206e section 5 states that you can do this, but you need to set the SRAM base address registers using supervisors instructions. It also says that the BDM device can perform these operations. I don't know if all BDM modules will support this operation, but perhaps this is a feature you should consider when deciding which BDM device is best to purchase. Spending an extra \$1000 for a great BDM, one that would permit using internal SRAM as boot device, would get you a gold star. If only management understood how great we are.

## TEST SETUP

So, what would various tests look like? I would first divide the tests into two parts. One would be a functional test and the other would be assistance for troubleshooting. The functional test would be just that—a test that guarantees 100% functionality of the device being tested. The other test would exercise the various signals and pins associated with the interface to the device. A technician could then attach test equipment and look for defective signals.

The first type of device is a memory device. The traditional device test that I execute is the following:

- Write all zeros, then read all zeros.
- Write all ones, then read all ones.
- Write a unique value into each location, then read that unique value.
- Fill the memory with a pattern of AAA. Then, for each location, read and check the AAAs, and shift the pattern right one time. Write the pattern and read the pattern. Keep repeating this shifting, writing, and reading till you have written and read zeros. Then move on to the next location looking for the original AAA pattern.

This is a brute force test. You're looking for data stuck at 0 and 1, all unique locations (no address lines shorted), and then for pattern sensi-

tivity. Believe it or not, you'll find a few devices that fail these tests. Not many, but one is enough.

This test was developed for 4- or 8-KB SRAM devices. But, what if the memory gets big? How long will these tests take to run? I contend that if it's a well-documented test procedure that a technician can run, time doesn't matter. If it's going to take too long, look at dedicated test fixtures instead of reducing these patterns.

What if I've got 8 MB (or more) of DRAM that I'm attempting to test? That type of testing could take a long time. I would still say, so what. Any lesser a test is just more of a guess as to whether or not you've got a good memory. Obviously some sort of progress output as the tests are running would be a great help. But, adding that output (via a serial port perhaps) will require a call to a subroutine. That calling routine needs a functioning stack (an area in memory to save the address of the calling routine and area for passing parameters). Keep this in mind when mapping out your various memory areas. You can't have valid stack operation in defective memory.

I've seen procedures written up for testing DRAM in PCs. Perhaps some of these tests could be modified as your embedded DRAM gets larger. The exercise test for assisting a technician in troubleshooting would probably shift a single one through each of the data bits, writing then reading that data. Then, write and read all zeros and ones but this time shift a one bit through all of the address lines. This is straightforward for SRAM because all of the address and data lines are non-multiplexed. But, this probably isn't the best for DRAM.

## POP QUIZ

If you look closer at how the DRAM is connected to the micro, you'll see the control signals CAS[0,1,2,3], RAS[0,1], and DRAMW. All are active low, but that's not significant for this discussion. And of course, there are the address and data lines. On different micros you'll find different detail connections, but similar DRAM devices need the same sig-

nals presented to them so the end results should apply to all.

A normal read transfer goes something like this:

- Row addresses valid
- RAS goes low (active)
- Column addresses valid
- CAS goes low (active)
- Data is valid

And, a write transfer is much the same as the read transfer except DRAMW is active (low). Physical address lines from the inter core of the micro are mapped onto the DRAM devices. On the MCF5206e, Address [9...27] can be used to drive the DRAM devices. Using 4Mx8 and 1Mx8 devices, CPU A[10] is connected to DRAM A[0]. And so on, up to CPU A[21] connected to DRAM A[10]. A good test would be to present an address pattern that places all addresses low, then raises one address bit at a time. This would permit you to go in with a scope, trigger on the start of a DRAM cycle, and look at each address line for a valid signal. In my example, a walking 1 from CPU address A[0] to A[20] would do the trick. Each micro will have a different mapping but the technique is much the same.

Again, on the high-performance devices, the data bus can be wired in several ways to the DRAM. I would just cycle a 1 pattern on each of the data bus lines, but keep in mind your specific wiring.

DRAMs have burst transfers in different page modes. If you have a DRAM that passes simple tests but does not pass a burst-mode test...well, let's leave that for another day. I would suspect that's a power supply/decoupling problem.

One important point to remember is that these micros have instruction and data caching. Be sure that you have turned these features off! If they are enabled, you are only testing the cache and not the external devices.

## FINAL EXAM

Now let's look at testing a complex device attached to the micro. The specific device I need to test is an HDLC

controller with DRAM attached behind that device. Obviously it's a large complex device. But, broken down from a testing or troubleshooting point of view, it's made up data and control signals coming from the micro, data and control signals between the controller and the DRAM, and then the HDLC specific input and output signals.

These types of devices typically have registers to control their configuration and operation. The first type of test I would perform is to write all the registers and then read them back and check for correctness. Don't write and read each register because, if the device is defective, you'll likely be reading the values remaining on the data bus (the ones you just wrote). Look to see if you can use one of these registers as a register to write and read all data values. If the registers fail, it's time to pull out the scope and look for the problem. If they pass, you need more sophisticated tests. The tests become more device specific, but basically they would start with simple functions and grow into the more complicated operations.

Take for example, the DRAM connected to the back of the HDLC controller. First, you need to set up the registers in the HDLC device to control the DRAM. You can view the DRAM refresh cycles without any DRAM test software. Then you can write the same DRAM test software that was just covered. This time, access would be through the HDLC device.

So, consider developing your test suite early in your product design cycle. Ask yourself how you're going to test these units. Also, decide how you're going to troubleshoot the units that fail in production. When the time comes, it will be a lot easier to implement those plans. ☒

*George Martin began his career in the aerospace industry in 1969. After five years at a real job, he set out on his own and cofounded a design and manufacturing firm. Typical systems that George designs include servo-motion control, graphical input and output, data acquisition, and remote*

*control. George is a charter member of the Ciarcia Design Works Team and most recently, he's been working on the people-tracking system for Bill Gates' new house. You can reach him*

Circuit Cellar, the Magazine for Computer Applications.  
Reprinted by permission. For subscription information,  
call (860) 875-2199, [subscribe@circuitcellar.com](mailto:subscribe@circuitcellar.com) or  
[www.circuitcellar.com/subscribe.htm](http://www.circuitcellar.com/subscribe.htm).