ABSTRACT

Emulation based ATE is an appropriate technique to use when testing complex / microprocessor based boards containing one or more microprocessors, microprocessor-emulation-based ATE is an appropriate technique to be considered for testing. By using microprocessor emulation, the unit under test can be stimulated and controlled from its microprocessor buses. This technique often can complement other test strategies, providing test capability during the board’s complete life cycle from design through manufacture, system integration, and field test and repair. There are a number of testability guidelines that can reduce program development time and test fixture complexity. These techniques are often simple to design in and require very little board real estate. ROM-based emulation techniques are described in detail, where reset and boot ROM memory space are the primary tester interfaces. Cases are presented from actual board designs and several testing challenges are discussed, including: multiprocessors, dual ported devices, redundant processors, surface mounted processors, ASIC’s which contain processors, and boards which must be coated.

INTRODUCTION

New testability techniques such as Level Sensitive Scan and testability buses have assisted in making today’s complex VLSI-based boards more testable. Many of these boards contain one or more microprocessors, where microprocessor, or ROM emulation-based ATE may complement the other strategies providing test capability during the board’s complete life cycle from design through manufacturer, system integration, and field test and repair. While implementing test strategies on many different types of boards, we have learned that while nearly all boards could be tested using emulation-based systems, the work-around required on some existing designs could have been avoided with minor changes in board layout or circuit topology. Reviewing the test strategy is useful before discussing circuit details.

Most boards which contain microprocessors are designed with internal bus structures which lend themselves to inside-out testing, since most of the peripheral chips or functions are easily controlled from the microprocessor side. Functional partitioning of the board may be readily accomplished if the operating firmware of the board can be disabled, since the normal operating
environment usually enables many of the functions simultaneously. Closed-loop at-speed testing of functional partitions can often be extremely difficult to implement from the outside connectors only. An integrated approach of controlling the UUT’s functions from both sides of the functional block, (the microprocessor bus interface and the external connector interface) will enable most cost/effective test results. Many designers and test engineers provide for diagnostics which can accomplish this same test capability. However, it requires the internal processor kernel and usually some peripheral interface capability, to be fault-free. Emulation-based ATE provides a constant and reliable source of control of the inside part of the closed-loop approach. Driver/sensor capability provides control of the outside part of the closed-loop test. Both are shown in Figure 1.

Techniques for implementing Microprocessor or ROM emulation-based ATE vary from vendor to vendor, but the process involved can usually be included in one of three types. Microprocessor emulation describes the process where a pod, usually containing the processor and some control circuitry, replaces the board’s microprocessor. Microprocessor bus emulation describes the process where the board’s microprocessor is disabled and the microprocessor bus activity is simulated with an emulation pod attached to the bus. ROM-based microprocessor emulation describes the processor where the emulator controls the board’s microprocessor through the boot ROM socket. From the user’s point of view, all these techniques accomplish the same task, although there are many interface differences. We will discuss design techniques appropriate for ROM-based microprocessor emulation, although some of the issues discussed may be applicable to the other emulation implementations.

An additional word about the terminology may be appropriate at this time. ROM emulation means the ability to emulate read-only memory, and usually means the ability to replace the contents of the memory space at will from some control device. This technique allows one to replace and test new firmware without having to reprogram new ROMs. It has been used in R&D for years. It has been limited in that it has provided only one-way communication from the emulator to the processor. ROM-based microprocessor emulation means the ability to control the microprocessor through boot ROM emulation using the Navatek CPU Commander(tm) technique of establishing a bi-directional controlling communication link between the boot ROM emulator and the board’s microprocessor. No assumptions about the board design are made and the interface may be, as is normal, a read-only interface. This technique allows the microprocessor to be embedded in an ASIC or soldered to the UUT and only requires that an external boot ROM capability exists. From the user’s point of view, it appears that it is the microprocessor, since the controlling link software is transparent to the user.
GENERAL REQUIREMENTS

The technical interface requirements for ROM-based microprocessor emulation ATE include access to the microprocessor’s boot memory space and access to the microprocessor’s reset environment. Minimum requirements for the boot memory interface are access to the lines for the normal boot environment. These include data lines, address lines, and appropriate control signals (the pod requires normal ROM signals, i.e., output enable and chip select). Emulator generated reset must force the microprocessor to vector to the boot memory space. The minimum interface requirement does not include any working RAM on the unit under test, and the CPU Commander(tm) monitor program requires only 8K Byte of the boot memory space, although all the ROM address lines will be monitored by the system.

GENERAL MODEL

A general purpose CPU board usually consists of a single CPU chip, boot ROM, RAM peripheral devices (UART, PIO, PIC, DMAC, etc.), and maybe an expansion bus. The ROM is normally socketed, since firmware is not all that firm. Reset is generally implemented through power-up circuitry with a push button circuit for manual use. Given this environment, the general technique for connecting the ROM-based emulation system to the board, in low volume applications, is as follows: Replace the boot ROM with a pod (or pods, if it’s more than an eight bit boot bus). Place the ROM in the pod ZIP pocket. Attach the reset stimulus line to the appropriate circuit. Apply power to the board. Issue the reset pulse (See Figure 2). The processor vectors to the boot ROM space and the emulator establishes communication with the processor. If this process succeeds, the kernel, which consists of the processor, ROM socket, and appropriate decoding and control circuitry, is sufficiently active to continue testing. If this process fails, fault isolation techniques in the kernel can be implemented at this time. Since the first functional partition of the board failed its functional test, the user can troubleshoot the kernel by using the reset stimulus as a way to activate the CPU. Probing techniques can be used to isolate the fault to the failing node. For high volume applications, all of the above connections would typically be made with a bed-of-nails fixture.

TYPICAL VARIATIONS FROM THE GENERAL MODEL

Now let us consider some typical variations from this scenario. For various reasons, reset circuitry can be implemented in many ways. As shown in Figure 1a, an RC network is often included in the reset environment to provide pulse stretching, de-bounce, or power-up reset. The emulator reset pulse width must be long enough to discharge the capacitor sufficiently to force a reset. Also, the capacitor charge time constant cannot exceed the delay time specified before the emulator times out and assumes a failure. This situation can usually be resolved in existing designs by relocating the
Design for Testability

stimulus to another node in the reset circuitry, reducing the time constant, or adjusting the emulator timeout. However, if the design were implemented as in Figure 1b, then the tester can assert reset with a simple pulse on the tester input to the gate and save time during the boot sequence. That can payoff during dead kernel troubleshooting, when looping on reset will provide a repeatable stimuli.

Another variation is a watchdog timer mechanism. Often this is implemented in hardware, such that the processor is held in reset if the timer times out due to lack of attention from the processor. Considering the watchdog as just another functional block to be tested, we want to be able to disable it, either in hardware with a jumper as shown in Figure 2, or in software.

Some designs use hardware memory swapping and overlay techniques. If these are implemented in certain ways, the test function may be hard to implement. ROM-based interrupt vector space reduces flexibility for testing interrupt devices. There might be a bank switch which moves or replaces the boot ROM space. In addition, we might want to be able to disable cache memory. The design solution to these situations is to ensure some flexibility, perhaps through jumpers, or a test port in the hardware to minimize interfacing problems. Other designs require soldered ROM. Several solutions are appropriate, depending on the board functionality or the flexibility of the design. One technique is to gate one of the ROM control signals (output enable or chip select). Gating is better than overdriving because unlimited overdriving will cause damage. This may allow clipping over the soldered-in ROM with the pod to pick up the address, data and other lines as necessary, and the control signal can be jumpered to the pod (see Figure 3).
A better solution may be to provide an extra parallel socket for the pod and a test jumper that disables the normal boot ROM. An even better solution may be to provide a test connector that accomplishes the same thing, i.e., provide the normal ROM signals plus allow the disabling of the normal ROM. Our recommended solution is one which will solve this, as well as several other problems. We recommend that the ROM enable (either output enable or chip select) circuit include an address decoding mechanism, such as a PALS or GAL, which has an extra input to indicate the presence of the tester. When the tester is attached to the test socket (or to the board in whatever manner is appropriate), the decoding is such that the tester occupies the normal boot space and the normal boot ROM is relocated to some out-of-the-way address space. This allows access to the contents of the normal boot ROM at all times, which should contain address-independent subroutines whenever possible (BIST). This recommended solution, also incorporating a reset connection, is shown in Figure 4.

As mentioned above, the existence of functioning RAM can be used to speed certain tests and may be necessary for certain tests (specifically those which require a stack). If the board has no RAM, a scratch pad RAM may be added to the test adapter, if one is being used. If possible, at least 8K of RAM should be designed into the board for testing purposes, if none exists for other reasons.

ROMless boards, i.e., expansion bus boards, accelerators, or boards which plug into High Address Bus motherboards and are booted from downloaded RAM, present their own set of requirements. Again, a test environment allowing an external boot ROM space to be implemented is the solution. It often can be as simple as an extra input on the address decoding circuit, since, in these environments, the address and data buses are usually available on the edge connector. It should be realized that if a test adapter is being implemented from the address bus, it may be necessary to provide a mechanism to indicate the normal access time to the emulation pod. Also, if a test connector interface, or test adapter to a bus is being implemented, and the processor has dynamically allocated bus width capability, provide for reducing the boot ROM bus width to the smallest width possible to reduce interconnect hardware requirements. An example of such circuit with no boot ROM is shown in Figure 5.
The circuit shown assumes that the boot ROM address must be forced into RAM prior to reset by a driver/sensor system emulating a motherboard or master controller board. If this function, i.e., the first eight bytes of code fetched after reset, is provided by the hardware, then the driver/sensor requirement for emulation is not necessary. However, it may still be desirable for other test purposes. The important feature of Figure 5 is that the test adapter provides an input to the memory decode circuitry, which then provides the appropriate ROM enable signal, and changes the timing to assure valid data from the ROM pod. If such circuitry does not exist on the board, similar circuitry must be designed into the test adapter.

Microcontrollers with internal boot ROM need special attention to their test strategy. Many such devices have a control line to force external boot. If this control line is brought to a test socket along with the boot address and data lines, no additional space needs to be utilized. Although some microcontrollers do not have this external boot capability (and they should be avoided given this test strategy), they are often used as intelligent or programmable peripherals and usually can be tested as such. Similar comments can be applied to digital signal processors (DSP) although the issue is often ROM access speed rather than internal boot ROM (the CPU Commander can support 55nsec or higher ROM access time). These devices will normally be used in a multiprocessor environment which is discussed below. Another generality, given this test strategy, is to allow easy access to fault tolerant RAM spaces from the CPU. These circuits are error detecting or error correcting in nature. It is desirable to force errors without hardware jumpers in order to test the error handling circuitry. It is also desirable to disable and to test the extra RAM independently.

**FEEDBACK LOOPS**

Without some method to interrupt feedback, feedback loops are extremely difficult to troubleshoot. This is especially true for testing methods which use back-tracing algorithms to isolate faults. Simple design rules can eliminate the problem entirely. The addition of one or two gates in the feedback loop is normally all that is required to disable the loop. For example, if an inverter is used within the loop, replace it with a two-input NAND gate that has the second input tied high with a pull-up resistor. For testing, the resistor input node is driven low by the tester, forcing the output of the gate high. This disables the normal signal from feeding through. The schematic below illustrates the technique.

![Figure 6](image-url)
OTHER VARIATIONS FROM THE GENERAL MODEL

Up to this point, we have only discussed typical variations to the general purpose model used as an example. What about the unusual cases? Let's start with multiprocessor designs. If the processors are independent, they normally require individual reset and boot environments, and the implementation of emulation control is normal for each processor. Simultaneous control of multiple processors is usually desirable to test certain functions. If any of the processors have any of the scenarios described above, the solution is the same as discussed above. Some simplification is desired in several cases, however. For instance, if one of the processors is a microcontroller with internal boot ROM and no external ROM capability, then reset for that processor should be software-controlled from the controlled processor. This allows one to force the microcontroller circuit into a known condition for fault isolation. This is especially true for designs which use slave processors, such as a DSP or a bit-slice, on the same board. If the slave processor RAM can be accessed by the controlled processor, then much of the circuitry can be tested independently of the slave processor. Then, with the software reset capability, the slave processor functions can be tested under the control of the controlled processor.

The multiprocessor design may use redundant processors, i.e., equivalent circuits running identical code and voting on the outcome with some type of compactor circuit between processors. Here, we want a mechanism for resetting all processors simultaneously, as well as a mechanism for resetting them individually. Even if the design uses a single boot ROM, each processor needs its own boot space for simultaneous control when the tester is connected. Also, we want the ability to switch from one mechanism to the other under tester control, so that each processor can be tested individually, as well as together, to test the error detecting circuitry. Similarly, for parallel processors which are equivalent circuits running different code, we want individual boot and reset environments available to the tester. Circuits which contain dual-ported devices can be easily tested if the processors on each side are simultaneously controlled by the tester. Similarly, if one side is processor-controlled and the other is available to an external bus environment, the external bus can be simulated by the driver/sensor system and integrated into the test of the processor side. These special circuits need to be exercised simultaneously in order to force contentions inside the control circuitry for test or design debug purposes.

Finally, some environments require the use of a test connector. For example, both surface-mounted designs and designs which will be coated for military environments lend themselves to the test connector concept. Using the techniques discussed above, a test connector — which provides for reset; relocation of normal boot ROM to an unused address space; access to address, data, and control lines to emulate the boot ROM space; and, access to internal jumper requirements — can provide a sufficient window into the board to allow inside-out functional
Innovation and Engineering

Design for Testability

testing. If the test connector access is designed carefully, it will usually provide for system level or field testing. The incremental cost of providing this connector is very low and the benefit can be significant over the product life cycle.

SUMMARY

ROM-emulation-based ATE can provide additional test capability for many normally difficult-to-test designs. However, significant savings can be made in test program and test fixture development if certain considerations are taken into account at the design stage. We have attempted to provide the reasons for these considerations and have described possible implementations for many of the environments we have seen. We believe that those design techniques should be used even if emulation-based ATE is not being implemented at this time in your application. Other departments of your company may implement this type of ATE to meet future needs. This is especially applicable to the field diagnostic and repair processes.

ADDITIONAL APPLICATION NOTES ARE AVAILABLE FROM GEOTEST