A Behavioral Test Strategy For Board Level Systems

by

Qaisar Hameed

Thesis submitted to the Faculty of
Virginia Polytechnic Institute and State University
In partial fulfillment of the requirements of the degree of

Master of Science
in
Electrical Engineering

F. Gail Gray, Chairman
James R. Armstrong
Dong S. Ha

March 03, 1999
Blacksburg, Virginia

Keywords: Testing, Start Small, Legacy Systems, VHDL, Assembly, Smart Model, March X

A Behavioral Test Strategy For Board level Systems

Qaisar Hameed

Dr. F. Gail Gray, Chairman

Bradley Department of Electrical and Computer Engineering

(Abstract)

A digital board typically contains a heterogeneous mixture of chips: microprocessors, memory, control and I/O logic. Different testing techniques are needed for each of these components. To test the whole board, these techniques must be integrated into an overall testing strategy for the board. In this thesis, we have applied a behavioral testing scheme to test the board. Each component chip is tested by observing the behavior of the system in response to the test code, i.e. the component under test is not isolated from the rest of the circuit during test. This obviates the need for the extra hardware used for isolating the chips that is required for structural testing. But this is done at the cost of reduced fault location, although fault detection is still adequate. We have applied the *start small approach* to behavioral testing. We start by testing a small core of functions. Then, only those functions already tested are used to test the remaining behavior. The grand goal is testing the whole board. This is divided into goals for testing each of the individual chips, which is further subdivided into sub-goals for each of the sub-functions of the board or sub-goals for testing for the most common faults in a component. Each component is tested one by one. Once a component passes, it is put in a passed items set and then can be used in testing the remaining components. Using the start small approach helps isolate the faults to the chip level and thus results in better fault location than the simple behavioral testing scheme in which there is no concept of passed items set and its usage. As an example, this testing approach is applied to a microcontroller based temperature sensor board. This code is run on the VHDL model of the system, and then also on the actual system. For modeling the system in VHDL, Synopsys Smart model library components are used. Faults are injected in the system and then the performance of the strategy is evaluated. This strategy is found to be very effective in

detecting internal faults of the chip and locating the faults to the chip level. The interconnection faults are difficult to locate although they are detected in most of the cases. Different scenarios for incorporating this scheme in legacy systems are also discussed.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. F. Gail Gray for giving me the opportunity to work in the area of digital systems testing. He gave valuable help and support throughout the project. I would also like to thank Dr. Armstrong for his suggestions about the board level testing strategies and the improvements he recommended for the report. Special thanks are due to Dr. Frank of Research Triangle Institute, NC who gave useful ideas and tips for the project.

I would like to thank Dr. Ha for serving on my committee. And finally, I thank all my friends and colleagues who made my stay at Virginia Tech a memorable one.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

A *test* is an experiment done either to confirm or deny a hypothesis or to differentiate between two hypotheses. A device is tested to check whether it is *faulty* or working correctly within specified limits. A fault occurs when there is an *error* in a device and this error propagates to the output so that the input-output response is different from the expected response. An error is incorrect value of a signal. Testing a device means checking it whether it is operating as expected or not. In any type of testing, we need to know two things. First, which input stimuli should be applied to the device? We should apply only those inputs that *activate* the fault i.e. the inputs which result in different outputs from the faulty and fault free device. The second thing we should know is the response of an ideal device so that we can compare it with the response obtained from the device under test.

In the digital electronics realm, the input stimuli are called test patterns or test vectors. The test vectors and the resulting responses are patterns of 0's and 1's. In the analog world, the input stimulus and the output responses can be complex combinations of frequency, voltage, resistance and other parameters.[TurinoJ90]

At the register transfer level (RTL), functional testing can be used to test the chips. Functional testing treats a chip as a black box and tests it from the input-output relationships. The input-output relationships for commercial ICs can be easily obtained from their data sheets. In case of

ASICs, they must be provided by the manufacturer along with the chip. Functional testing can take many forms: through an Automatic Test Equipment (ATE), self-test or emulation testing. When employing an external ATE, the test program is written in the tester programming language. On the other hand, if self-test or emulation testing schemes are used, the language used is that of the system itself.

1.1.1 Behavioral Vs. Structural Testing

To test a digital board, two different approaches can be taken. In the structural testing approach, the board is divided into component chips. Each chip is then tested in the circuit one by one. The chip under test is isolated from the rest of the system by special circuitry. This is called the guarding technique. The interconnections between the chips can also be tested if the test program has knowledge of the net list for the board. A typical example of this technique is the bed-of-nails testing approach. This requires a fixture whose pins make contact with the test points on the board. The input stimuli is applied and the response of the board is taken by the host test program through the bed-of-nails fixture. Fault resolution is excellent in this method, but it has a few drawbacks. This is an off-line method since the board under test is not powered up during testing. Also, the board faults resulting from improper timing relationships between different components cannot be detected.

In behavioral testing, all the individual chips are tested by checking the response of the *system* rather than the chip alone. Access to internal test points is not needed. Interconnections are not explicitly tested, but an interconnection fault manifests itself as failure of some chip during testing, so it is indirectly detected. No external fixture or guarding circuitry is required in this type of testing. Also, the programming language is that of the system itself. But, fault resolution is less than that for structural testing, because the components are not isolated during testing.

1.2 Survey of existing board level testing techniques

There are several testing techniques that are commonly used for board level testing. Each of them has advantages and disadvantages. The most effective test strategy is to combine the strengths of the individual techniques.

1.2.1 Functional testing

The basic idea of functional testing [BatesonJ85] is to verify the input-output relationship for the system. That is, we apply a stimulus at the input to the board and verify that the response at the output matches the expected response. The test is normally applied using Automatic Test Equipment (ATE). The functional test detects faults caused by engineering errors, manufacturing defects and faulty components. However, test generation to achieve acceptable fault coverage can be difficult and expensive due to lack of adequate controllability and observability. Fault location is difficult and sometimes impossible without observing internal data lines.

1.2.2 In-Circuit testing

The in-circuit testing approach [BatesonJ85] involves measuring the performance of individual components by electrically isolating them from the rest of the circuitry. A “Bed-of-nails” fixture is used to gain access to necessary points on the board. There can be four levels of test hierarchy in an in-circuit tester: shorts, passive components, IC orientation and digital logic test. Passive components are tested on an unpowered PCB, then power is applied and active components are tested.

The in-circuit tester detects faults in individual components. Some of the in-circuit testers also have the continuity testing feature so that PCB tracks can be tested. Although controllability and observability are almost unlimited in this method, it is difficult to apply to certain types of boards, such as boards that use surface mount technology (SMT), or boards that have a very high density of chips and interconnections. Also, while applying test stimuli to the input of a chip, the output of other chips can also be driven by the same signals and this may damage those chips.

This is called Back-driving and is another serious problem when using in-circuit testing. The contacts can also be unreliable due to contact pin capacitances at high frequencies.

1.2.3 Emulation techniques

Emulation techniques involve replacing one or more complex chips on the board with a test program chip or an external emulator that can be controlled by a tester. There are three major techniques: microprocessor emulation, memory emulation and bus cycle emulation.

In microprocessor emulation, a test pod is attached to the microprocessor leads or plugs into an empty microprocessor socket. This test pod is controlled by a tester and is capable of performing the same set of operations as the original microprocessor. The advantage of this method is that the programming language and the development tools are those of the microprocessor itself, and native code test programs are excellent at generating activity on the microprocessor bus. The main disadvantage of this method are lack of diagnostic resolution. There are no automated tools to locate the faults. This technique is limited to testing devices directly connected to the microprocessor bus, since it has little ability to drive and test signal states at random points in the board. Another disadvantage is that the signals are coming from the tester, not the actual microprocessor, so that the timing of the signals may not exactly match that of the microprocessor. Also, a different pod is required for each different microprocessor. In addition, microprocessor emulation does not test the microprocessor chip itself, since the chip has been removed.

In memory emulation [FiliS85], the tester substitutes its own memory for the board's ROM memory. The board's own microprocessor then executes the test program located in tester's memory. In addition to all the advantages listed for microprocessor emulation, memory emulation testers can test the microprocessor chip. Since the microprocessor is driving the bus, the test is done at full processor speed and the signal timings will be exactly correct. However, timing margins are not testable. Memory emulation has most of the disadvantages listed for microprocessor emulation, particularly the lack of diagnostic resolution.

Bus cycle emulation [BroyS84] involves the application of bus signals from the tester. It is applicable to all boards whether or not a microprocessor is present. If there is a microprocessor on the board being tested, it is enticed into giving up control of the bus so that the tester can control the bus instead. The most straightforward way to do this is to apply a “bus request” to the microprocessor and wait for it to acknowledge that it has tri-stated its bus interface. This is an ideal technique for off-line testing of boards that do not contain microprocessors but are connected to microprocessor buses. The classic example is that of add-on cards for a PC expansion board. Also, since the emulator may vary the timing of the signals, timing margins can be verified. Disadvantages include: (1) the microprocessor cannot be directly tested since it is in a wait state, (2) microprocessor wait states and clocking need to be controlled, (3) programming the tester to emulate correct timing for bus cycles can be difficult and is processor specific.

1.2.4 Boundary Scan method

To overcome some of the problems of testing complex boards, IEEE standard 1149.1 specifies a boundary scan architecture that greatly simplifies testing [IEEEs90]. This architecture provides a single serial scan path through the input/output pins of individual chips on a board. The scan path is created by connecting the normal inputs/outputs of the internal chip logic to the input/output pins of the chip through boundary scan cells [LalaP97]. The scan cells are used for various test functions. Many algorithms have been proposed to fully test system interconnects. Most of these can be easily adapted to use the boundary scan architecture. The main drawback of this method is that most existing microprocessor and support chips do not support boundary scan architecture.

1.3 Goal based Test Strategy

Lin has proposed a behavioral test strategy for DSP systems modeled in VHDL [LinM98]. The idea is to divide a complex main, or grand goal into less complex sub-goals. This division continues in a hierarchical manner until very basic primitive goals are obtained. By realizing the primitive goals, the top level grand goal is achieved. The primitive goals are simple enough so that test strategies can easily be specified to achieve them.

Formally, a goal tree [LinM98] represents a test plan. The development of a test plan is a top-down partitioning process. It starts with the desired test goal and breaks that goal into simpler subgoals. The goal tree consists of a node set and edge set. The node set is partitioned into goal nodes G, test group nodes TG and operator nodes O. The directed edge set indicates relationships between the nodes. The goal nodes G represent the test goals of a test plan. The root of a goal tree is a goal node which corresponds to the grand goal of a test plan. The test group nodes TG represent the actual tests that are applied to the system to achieve the primitive goals and hence are the leaves of a goal tree. An operator node O can be either a logical AND or an OR operator. It decomposes a goal into subgoals, and helps a primitive goal define two or more test groups in and/or fashion. All children of an AND operator must be satisfied to fulfill the objective of its parent node. On the other hand, fullfilment of any child of an OR node suffices.

1.4 Proposed Technique

In this thesis, we apply this goal tree approach combined with the behavioral test strategy as discussed in 1.1.1, to develop tests for a microprocessor based board. We will first develop a general testing strategy that can be applied to any microprocessor based board. Then, we will apply this strategy to a temperature monitoring system.

The grand goal is to verify board operation at startup using a self-test procedure. By self-test, we mean that the system tests itself without the need for any external hardware. The programming language is also that of the system itself and the test code resides in the same memory as the system program. When the system powers up, the self-test code is executed before the application program starts. Successful completion of the self-test is followed by the application program execution. But if there is a fault in any part of the system, an error message is displayed and the system is halted.

The grand goal is divided into sub-goals that test individual components. The sub-goals for testing components are satisfied successively in order to confirm the grand goal. In general, a

digital board consists of a microprocessor, ROM, RAM, decoder, latches, I/O interface circuitry and some display mechanism. We divide the grand test goal G into sub-goals for testing the display (G11), RAM (G12), ROM (G13) and microprocessor (G14) as shown in Figure 1.1. These sub-goals are then further divided into primitive goals for detecting specific faults in each component. For example, the RAM testing sub-goal is further divided into sub-goals that test the RAM for stuck-at faults (TG121), transition faults (TG122), coupling faults (TG123), and address decoder faults (TG124). Test algorithms are written for each of these leaf level test groups. The ROM test goal (G13) is satisfied using a checksum approach (TG131). The microprocessor test goal (G14) is subdivided into primitive goals for testing each instruction (TG141 to TG14n) and register (TG14(n+1) to TG14(n+m)). Satisfying the sub-goals G11, G12, G13 and G14 satisfies the grand goal G. The order of testing is important, as will be discussed later. The interconnection circuitry as well as support chips are tested indirectly through this test.

To illustrate the general test strategy, we applied it to a 68HC11 microcontroller based board that is used for temperature sensing. The system contains an external EEPROM, an external RAM, a programmable peripheral interface (PPI), latch, decoder, analog to digital converter, display driver, and a 68HC11 microcontroller, which also has internal RAM and ROM.

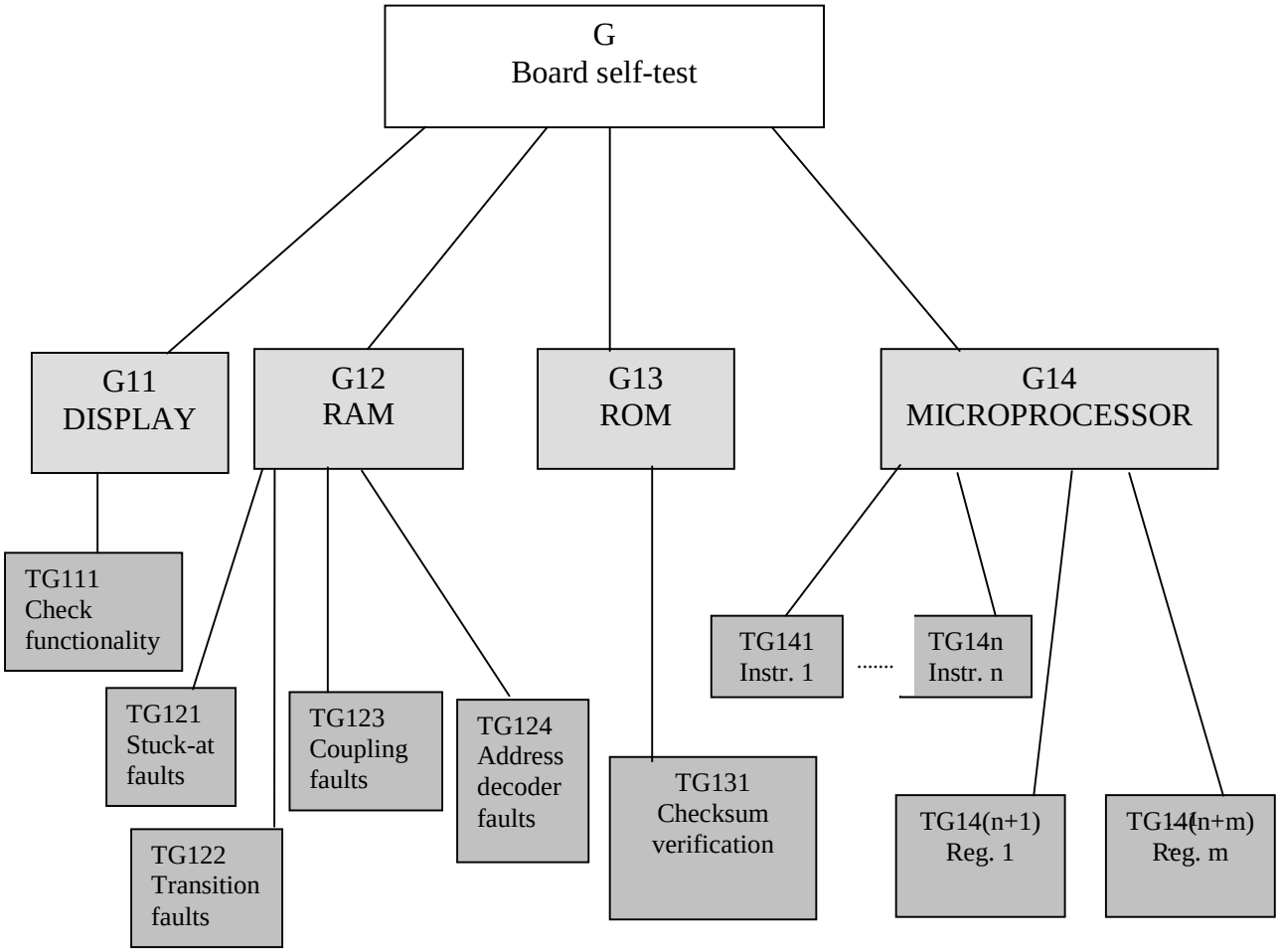

Figure 1.1 Goal tree for board testing strategy

1.5 Start Small Approach

A behavioral test approach, as it stands, is very weak in fault resolution. To improve the fault resolution, we propose a *start small approach* to behavioral testing. We start by verifying a small core of functional behavior using as few system components as possible. Each successive behavior and each additional component is then tested using only those functions and components that are previously verified. Then, we start testing the first component. If it passes, we put it in our passed items set. We can also put those items in the set that were used during the test of this component, because these components have to be working in order for the component under test to pass. However, a thorough test for a component is recommended before putting it in the passed items set. With the first component passed, we test the second component. If it passes, it is again put in the passed items set. But if it fails, we know that the failure can only be due to this component or some other component which is not in the passed items set at that point. So, our suspect set is reduced from all components to all components minus the components in the passed items set minus the components that were not used in testing this component. As the testing process advances, the passed items set continues to grow while the suspect set continues to decrease. As an example, suppose the total component set of the board is S, passed items set is P, suspect set is SS and set of components not used at a particular point is N. At start, we have:

$$S = \{A, B, C, D, E, F\}$$
$$P = \{\}$$

$$SS = \{A, B, C, D, E, F\}$$

$$N = \{A, B, C, D, E, F\}$$

Now, we test the component A first. Suppose, components D and F were also used in this test. Successful completion of this test means that the component A, as well as components D and F are working. So, the passed items set becomes

$$P = \{A, D, F\}$$

Next, we test component B. Suppose component C was also used in testing component B. Hence if the test fail, then the suspect set SS is:

$$SS = S - [P\mathbf{U}N] = \{A, B, C, D, E, F\} - [\{A, D, F\}\mathbf{U}\{A, D, E, F\}]$$

$$\Rightarrow SS = \{A, B, C, D, E, F\} - \{A, D, E, F\} = \{B, C\}$$

If the fault had occurred in some component that had been tested near the end, the suspect set would have been even smaller. So, it is clear that using the start small approach, fault resolution continues to improve as the test progresses.

1.6 Demonstration of the technique

We applied our technique to a 68HC11 microcontroller based board that is used for temperature sensing. The system contains an external EEPROM, a Programmable peripheral interface (PPI), latch, decoder, analog to digital converter (ADC), LCD display driver, and a 68HC11 microcontroller chip. In the first phase of our work, a VHDL model for the system was developed. The self-test was then run from this model. The steps taken to apply the test technique are:

1. Write self-test routines for different parts of the board
2. Add this self-test code to the main program
3. Load this new program in the EEPROM
4. Simulate the board using the VHDL simulator and confirm the test pass messages
5. Inject faults in the VHDL model of the board
6. Again simulate the board and check the test messages; no message also indicate test failure
7. Evaluate the performance of the technique

In the second phase of the work, the self-test code was run on an actual system. For that purpose, the code written in phase I was adapted to run on the actual system. The steps taken to apply the test were similar to those of phase I.

1.7 Thesis Organization

In this chapter, we have described various approaches that have been taken to test a digital board. We have also outlined the method that we are going to use for testing the board. Chapter 2 discusses the VHDL modeling and implementation of the temperature sensor system. In chapter 3, we describe the self-test of the system using our test code. In chapter 4, we describe the faults injected in the system and fault coverage obtained using our testing approach. Finally, in chapter 5, conclusions drawn from the project are mentioned. Various scenarios in which our technique is applicable is also described. We have two versions of our self-test code written in Motorola 68HC11 assembly language. One was written for the VHDL model and the other for the actual system. These two codes are different because (1) 68HC11A8 microcontroller was used in VHDL model whereas 68HC11E9 micrcontroller was used in hardware implementation. The two microcontrollers differ in the size of internal RAM, 256 bytes for the 68HC11A8 and 512 bytes for the 68HC11E9, (2) the LCD driver model and the physical LCD driver/display used are different in terms of control signals. In order to avoid making the thesis excessively lengthy, we have just given the code for the actual system. It can be found in appendix A.