

Evaluation of Hardware Test Methods for VLSI Systems

By

Jens Eriksson

LITH-ISY-EX-ET--05/0315--SE

2005-06-07

Evaluation of Hardware Test Methods for VLSI Systems

Master Thesis

Division of Electronics Systems
Department of Electrical Engineering
Linköping Institute of Technology
Linköping University, Sweden


By
Jens Eriksson

LiTH-ISY- EX-ET--05/0315--SE

Supervisor:
Thomas Johansson

Examiner:
Kent Palmkvist

2005-06-07

 LINKÖPINGS UNIVERSITET	Avdelning, Institution Division, Department Institutionen för systemteknik 581 83 LINKÖPING	Datum Date 2005-06-07
--	---	------------------------------------

Språk Language Svenska/Swedish X Engelska/English	Rapporttyp Report category Licentiatavhandling X Examensarbete C-uppsats D-uppsats Övrig rapport _____	ISBN ISRN LITH-ISY-EX-ET--05/0315--SE Serietitel och serienummer ISSN Title of series, numbering _____
URL för elektronisk version http://www.ep.liu.se/exjobb/isy/2005/315/		

Titel Title Författare Jens Eriksson Author	Evaluation of Hardware Test Methods for VLSI Systems
--	--

Sammanfattning Abstract <p>The increasing complexity and decreasing technology feature sizes of electronic designs has caused the challenge of testing to grow over the last decades. The purpose of this thesis was to evaluate different hardware test methods/approaches based on their applicability in a complex SoC design. Among the aspects that were investigated are test implementation effort, test efficiency and the performance penalties implicated by the test.</p> <p>This report starts out by presenting a general introduction to the basics of hardware testing. It then moves on to review available standards and methodologies. In the end one of the more interesting methods is investigated through a case study. The method that was chosen for the case study has been implemented on a DSP, and is rather new and not as prolific as many of the standards discussed in the report. This type of method appears to show promising results when compared to more traditional ones.</p>

Nyckelord Keyword Hardware testing, Testability, JTAG, CTAG.

1. INTRODUCTION	1
1.1 BACKGROUND	1
1.2 HARDWARE TESTING	1
1.3 RECENT PUBLICATIONS AND LITERATURE	2
1.4 MOTIVATION	2
1.5 CHAPTER DESCRIPTION	2
2. LITERATURE REVIEW	3
2.1 TECHNICAL TERMINOLOGY	3
2.2 BASICS OF HARDWARE TESTING	3
2.2.1 <i>Test generation</i>	4
2.2.1.1 Manual test generation.....	5
2.2.1.2 Algorithmic test generation	5
2.2.1.3 Simulation aided test generation.....	5
2.2.1.4 Random test generation	5
2.2.2 <i>Test application</i>	5
2.2.3 <i>Response evaluation</i>	6
2.3 FAULT LISTS AND FAULT MODELS	7
2.4 TESTBENCHES	8
2.5 DESIGN FOR TESTABILITY (DFT).....	9
2.6 SCAN-PATH	9
2.7 BOUNDARY SCAN	12
2.7.1 <i>IEEE 1149.1 JTAG</i>	13
2.7.1.1 Test Access Port (TAP).....	13
2.7.1.2 TAP controller	15
2.7.1.3 Boundary scan cells.....	16
2.7.1.4 Testing.....	17
2.8 AUTOMATIC TEST EQUIPMENT (ATE).....	18
2.9 AUTOMATIC TEST PATTERN GENERATION (ATPG).....	19
2.9.1 <i>Self Test methodology</i>	19
2.9.2 <i>Test pattern generation</i>	20
2.9.3 <i>Compaction and response evaluation</i>	20
2.9.4 <i>ATPG algorithms</i>	21
2.10 BUILT-IN SELF-TEST (BIST)	24
2.11 SOFTWARE BASED SELF-TESTING	28
2.11.1 <i>A software based self test methodology</i>	29
2.12 TESTING OF EMBEDDED CORES	31
2.12.1 <i>IEEE P1500 CTAG</i>	34
3. SIMPLE CASE STUDY	37
3.1 AN INSTRUCTION-LEVEL BASED PROCESSOR TESTING APPROACH	37
4. DISCUSSION	43
4.1 IEEE STANDARDS AND DFT TECHNIQUES	43
4.2 SELF-TESTING METHODOLOGIES.....	45
5. CONCLUSIONS	47
BIBLIOGRAPHY	49

1. Introduction

This introduction is meant to provide a description of the thesis, why it was chosen and what results were expected.

First the underlying situation that raised the questions in need of answering and the purpose of this thesis will be presented.

Then the reader will be briefly introduced to the current situation of hardware testing.

Finally the motivation and the expected results of this thesis will be accounted for.

1.1 Background

Electronic systems (ES), a division at the University of Linköping, is running a project with the purpose of creating an instruction comparable version of the Motorola digital signal processor DSP 56002. This project will within foreseeable time be approaching the phase when it is time to implement the processor in hardware. At, or rather before that point, it is going to be necessary to choose a method for testing it in hardware.

The purpose of this work is to examine and evaluate different generic test-methods (standards), based on their applicability in this particular project. An additional objective is to give an account for how to design for testing and what to think about during the design process.

Elements that need to be considered are extra area overhaul, extra pin count, required implementation effort and of course efficiency.

1.2 Hardware testing

Testing is nowadays seen as a major bottleneck in product development (PD). The complexity of large designs is increasing substantially, at the same time the technology feature sizes are decreasing. This evolution has caused the challenge of testing electronic systems to grow rapidly over the last decade. A circuit however needs to be tested to guarantee that it conforms to the specifications.

Testing of hardware is performed for two purposes:

1. To ensure the functionality.
2. To ensure the absence of process faults.

The first issue is primarily addressed through software simulation throughout the various iterations of the design.

The second issue is most widely performed on large scale production testing.

1.3 Recent publications and literature

The ever increasing complexity of hardware design demands the methods for testing to adhere and adapt to the challenges that arise. The development in the test sector advances rapidly, solutions to fit new problems contribute to the constant evolution of old, as well as the birth of new standards. As a result, much of the literature in this field becomes obsolete after a mere couple of years.

1.4 Motivation

There is a virtual plethora of methods and ideas on how to best approach testing. The intended result of this thesis is to evaluate the most widely used methods based on their applicability in a DSP core. Testing is nowadays a major bottleneck in design, and finding the appropriate test approach is anything but obvious. The purpose of this thesis is not to single out and advocate any single test methodology. Instead it will attempt to evaluate and compare different methods in the different steps of the testing process; test generation, test application, result analysis, based on the required test effort, test implementation tradeoffs and test efficiency.

1.5 Chapter description

The majority of this report is constituted by chapter 2, in which a look at current literature is presented. Common standards and testing methodologies, as well as the basics of hardware testing are reviewed. In chapter 3 one methodology that seemed particularly interesting is investigated more thoroughly through a case study.

Chapter 4 discusses the findings in previous chapters based on their applicability in a complex design. Finally conclusions and some thoughts on the future of hardware testing are accounted for in chapter 5.

2. Literature review

In order to make any decision as to which solution is best fit to fulfil the given requirements it was necessary to establish a foundation on which to base a discussion. Hence available methodologies were investigated through conducting a literature review.

2.1 Technical terminology

Throughout the report many technical terms occur, for a quick reference please refer to the glossary (appendix A).

Testability measures:

When discussing testability measures it is necessary to distinguish between *controllability* and *observability*. Controllability is the cost for placing an internal node at a given value. Observability is the cost of propagating the value of the node to an output. The cost for both increase with node depth, complexity and structure. A combinatorial circuit is less costly to control and observe than a sequential one.

Controllability can be measured for a given node as the number of clock cycles it takes to propagate the value from the primary inputs to the node. It can also be measured (more commonly) as the percentage of nodes that are actually controllable at all using a given test approach. Analogously the observability is the number of clock cycles required to propagate the value from the node to an output, or the percentage of nodes that can be observed.

The *fault coverage* is the percentage of detectable faults that are detected by the test, and thus determines how effective the test is.

$$\text{Fault coverage} = \frac{\text{Number of detected faults}}{\text{Total number of faults in the DUT.}} [1]$$

2.2 Basics of hardware testing

Testing is not design verification. The purpose of testing is to verify that the manufactured IC is free of errors.

All testing methods can be divided into two broad categories: *concurrent* testing and *explicit* testing. Concurrent testing means that normal application input patterns are used for test diagnosis, hence testing and normal computation occur concurrently. Explicit testing means applying test inputs especially crafted for the sole purpose of testing, and normal operation is on a halt. This report focuses mainly on the explicit testing category. The reasons for this are listed below.

Concurrent testing requires coding of the internal data with different types of error detecting codes. Special circuit augmentations, or checkers, are needed to monitor these data and signal fault detection. Such augmentations are infeasible in any real circuit if general fault detection is the target (could be used to signal specific faults). It also implicates an increased pin count (a minimum of two extra pins). Lastly, a concurrent testing approach is much more complicated than an explicit one. [2]

Figure 1 outlines the various possible testing techniques. Most of these will be discussed later on.

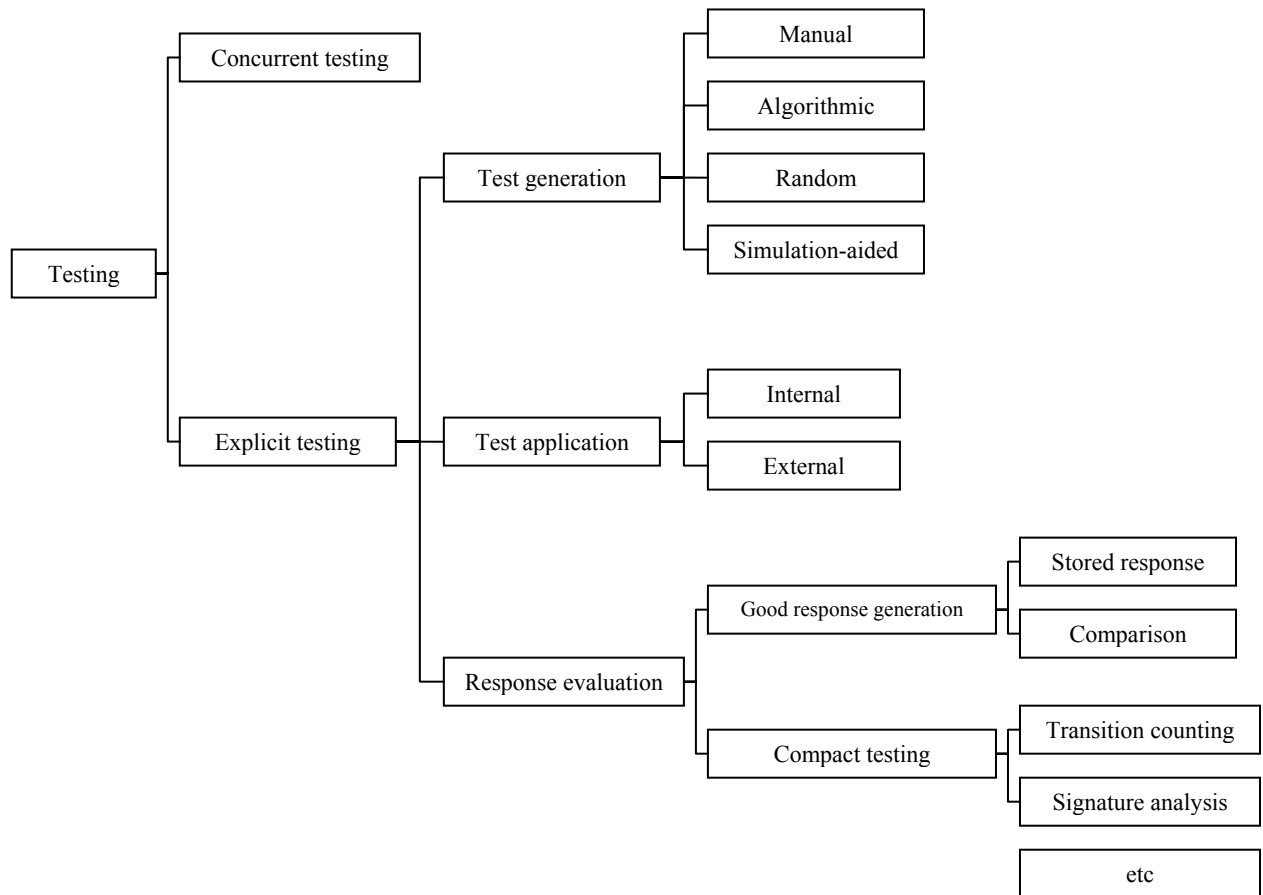


Figure 1 - Testing techniques

Explicit testing

As mentioned earlier, in explicit testing the testing is separated from normal operation. An explicit testing approach involves three steps:

- Test generation
- Test application
- Response evaluation.

2.2.1 Test generation

There are four different main approaches for test pattern generation: manual test generation, algorithmic test generation, simulation-aided test generation and random test generation. The efficiency of a test set depends on its length, extra hardware and data storage requirements and the fault coverage it achieves.

2.2.1.1 Manual test generation

Manual test generation requires extensive knowledge of the circuit under test. In manual testing the device under test (DUT) is carefully analyzed for the purpose of determining the patterns that will excite the faults. This tends to grant test patterns of optimal length and a high fault coverage. However, the manual pattern generation demands a great deal of effort and require a designer with special skills and knowledge of the DUT. Even so, a methodology for processor testing that utilizes manual generation of (deterministic) test patterns will be reviewed in 2.11.

2.2.1.2 Algorithmic test generation

Another (more common) approach is to devise one or more algorithms that generate the test patterns automatically. These algorithms can be modified in order to improve the fault coverage for the current DUT. Algorithmic test generation is a less tedious and less demanding task than manual test generation.

These algorithmic techniques can be based on gate-level descriptions, functional descriptions or register transfer level descriptions of the DUT. A few examples are discussed in 2.9.

2.2.1.3 Simulation aided test generation

This type of testing is used for fault simulation. That is, to interpret the behavior of the circuit under the presence of fault.

A circuit description, along with a list of faults to be simulated, is fed to a simulator. The simulator creates a fault simulator for the DUT that can simulate the circuit under both normal and faulty conditions. During simulation the response from the normal circuit is compared to that of various faulty circuits, under different input patterns. When a difference in the responses occurs, that pattern has detected a certain fault and that fault is deleted from the fault list. This process continues until there are no more input patterns (or no more faults). These types of tests are normally used to determine fault coverage.

2.2.1.4 Random test generation

This method uses a random pattern generator. The patterns are used as inputs both to a DUT and a known fault-free unit, and then the output results are simply compared. The fault-free response is usually acquired via simulation. Such random tests will clearly not compare to exhaustive or deterministic ones in terms of fault coverage. Another issue with random testing is the applicability on sequential circuits. In the case of sequential circuits the random patterns has to be supplemented in order to comply with the DUTs sequential behavior.

2.2.2 Test application

Test application can be either external or internal. External test application means that the test patterns are generated off-chip, and are loaded to the DUT through e.g. a tester memory. Internal test application means that the test patterns are either stored on-chip in a ROM, or they are generated by test objects that have been embedded into the design.

2.2.3 Response evaluation

The purpose of response evaluation is to detect an incorrect output response. This is achieved by comparing the actual output response to the expected, or good, response. This good response can either be derived manually, through simulation or from a golden model. These methods go by the collective term *good response generation*. They all suffer from the necessity to store large data volumes in internal memories, or the requirement of a golden model. This renders a large amount of data to analyze.

Another response evaluation technique, known as *compact testing*, eliminates these disadvantages. The idea is to compress the entire output data stream R , into a compact signature $f(R)$ that is compared to a compressed good response. Figure 2 outlines the different response evaluation approaches. A more in depth look at compact testing is given in 2.8-2.9.

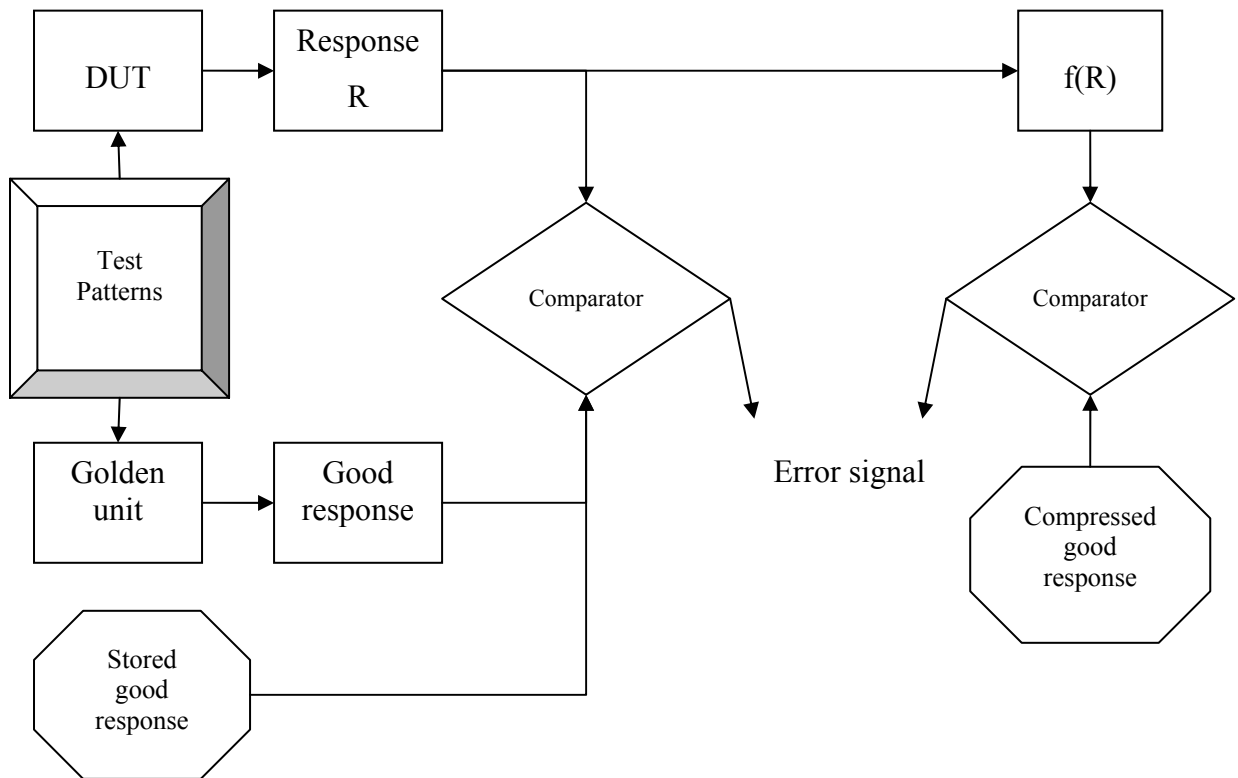


Figure 2 – Response evaluation

The simplest way of attaining the mentioned signature is through *transition counting*. In transition counting every logical transition on the output(s) increments a counter. So if the response is an n -bit data-stream, then the counter(s) need to be m bits, where $m = \lceil \log_2 n \rceil$. The positive sides of transition counting are that the memory volume required is very small, the circuitry is simple and it can operate at high speed. The downside is the amount of faults that might pass undetected. The order in which patterns are applied greatly affects the fault coverage. Multiple outputs will complicate the generation of good test patterns since an input stream that works well for one output might not work well for another.

2. Literature review

A more powerful method, based on LFSRs or MISRs (see 2.8 for LFSR and MISR), is called *signature analysis*. After all test patterns have been applied to the DUT the content of the LFSR or MISR is the signature. The extra circuitry required to implement signature analysis is very small (a small shift register being fed back through some simple gates per output). The probability of missing a fault is also very small. The signature can be seen as a unique fingerprint for the current test set. Let's say that a 16-bit register is used to compress an n-bit output-stream. An LFSR can be shown to equally distribute the 2^n data stream combinations over the 2^{16} signatures [2]. The probability of missing a faulty response, or aliasing, for a large output stream where $n \gg 16$ is:

$$\frac{2^{n-16} - 1}{2^n - 1} \cong 2^{-16}.$$

Most circuits have multiple outputs. This means that if a single feedback shift register is used for signature generation, each output line has to be fed to the register via a multiplexer. Alternatively (and more commonly), a multiple input register (MISR) could be used. An MISR is simply an LFSR with inputs at several stages.

Signature analysis, like transition counting, requires only a small volume of data to be stored and very simple hardware augmentations. This, the high fault coverage and the fact that fault coverage is not depending on pattern application order makes signature analysis the most attractive response evaluation technique (in the case where testing implicates application of large data streams e.g. automatic test pattern generation).

2.3 Fault lists and fault models

A fault list is the list of all faults possible with the fault model in use. A fault model can take on a number of forms, where the most common is the stuck at fault model. This model looks at faults where a line or a node is stuck at logical one or zero. Other common models look at bridging between pins or lines, opens, shorts and delay faults (signal propagation slower than it should be). Since the stuck at fault model requires a gate-level description, and is really not feasible for complex designs, it will not be discussed in greater detail in this thesis. Structural testing techniques, which uses single stuck at fault models, are based on gate-level descriptions. Gate-level descriptions are often not available and even if they are, such techniques are not able to model actual failures in VLSI designs. This thesis will focus on functional descriptions or netlist descriptions of the circuit and behavioral level fault models that can represent complex failures in VLSI designs.

In functional testing the DUT is tested to ensure that its behavior conforms to the specification. It takes place after the design specification is met during simulation and analysis. The functional test philosophy is to subdivide the DUT into several functional modules and verify their behavior individually. These modules can be seen as black boxes, whose contents are unknown but with a well-known behavior. Functional testing is more complex than structural because the circuit's sequential behavior has to be considered and it also needs to be initialized.

In order to illustrate a more advanced fault model (that could actually be used for a VLSI circuit) let's look at a model that was developed for testing of microprocessors, based on their functional description. The model is the one used by the Thatte-Abraham technique, which is really the mother of all functional testing techniques.

2. Literature review

Thatte-Abraham describes the faulty behavior of the DUT without detailed knowledge of the actual implementation of it. It was developed to target test generation for microprocessors based on their functional descriptions. This means that it is based on the processors instruction set and on what functions these instructions perform. The model categorizes different faults based on the different types of functions: data transfer function, data storage function, instruction decode function and control function, register decode function and data manipulation function. A fault model is derived for each of these types of faults. In order to keep test generation as simple as possible, the tests are generated allowing only one function to take place at a time.

The various faults for the different categories are listed below.

- The fault model for the register decoding function looks at faults meaning an incorrect register is accessed.
- The fault model for the instruction decoding and control function allows three types of faults: $F(I_j/I_k)$, $F(I_j/I_j+I_k)$ and $F(I_j/\theta)$. Where $F(I_j/I_k)$ means that instead of the correct instruction I_j , an incorrect instruction I_k is executed. $F(I_j/I_j+I_k)$ means that the correct instruction is executed, but an additional incorrect instruction is also executed. Lastly, $F(I_j/\theta)$ means that no instruction is executed at all.
- The fault model for the data storage looks at faults where any storage cell is stuck at one or zero.
- The fault model for the data transfer function looks at two types of faults: Faults meaning that a line in the instruction execution path is stuck at one or zero, and faults meaning that two lines are coupled, meaning their logical values cannot differ.
- Since the functional units of the circuit vary in many ways, no specific fault model exists for the data manipulation function.

Thatte-Abraham utilizes a data flow graph model in which nodes represent registers or active units, while links represent data transfer paths. It incorporates a number of algorithms, one for each class of faults.

The main philosophy of Thatte-Abraham is in many ways typical for any functional testing technique. The subdividing into functional modules, the segregation into different types of functions and the data flow graph are ubiquitous in all functional approaches.

2.4 Test benches

A test bench can be used to validate the functionality of a design. Once the test bench has been designed, the same test vectors can be used to validate all iterations of the design. Thus one test bench can be used for testing a model described by designer code, an extracted netlist and a software generated post layout model, since the I/O signals are the same. This is possible even though the post layout model bares little resemblance to the source code (with timing and such).

The event of creating a test bench can vary in approach. There are three basic approaches.

Tabular approach

Tabular approach means that the test bench is self-contained. The test vectors are embedded within the test bench. The test environment can be seen as a socket in which the device under test is placed. This approach is not well suited for testing that requires large data exchange.

File I/O approach

A more commonly used way of designing test benches is the File I/O approach. This is more suited for tests that require a large amount of test vectors. The vectors are listed in a file and are read into the test bench via a computer. This offers flexibility as the same environment can run multiple test cases by alternating test files. Procedures to read/write lines from/to a file as well as reading/writing vectors are defined by the IEEE 1076 standard.

The DUT is instantiated and a process reads and applies the test vectors and controls the clocking. The output vectors are stored in an output file, thus presynthesis results can be compared to post layout model results by reviewing these files.

Procedural approach

The procedural approach implements an algorithm that computes the expected outputs and compares them to the actual outputs. This becomes very tedious for complex operations. [3]

2.5 Design For Testability (DFT)

The decreased testability, due to decreased accessibility to internal nets as design complexity grows, has spawned various design techniques that increase testability. Among those are scan path and boundary scan design. Another one is the Built In Self Test (BIST) method.

Design for testability (DFT) is a general term for any feature incorporated into the design for the purpose of making circuit validation easier. The purpose of any DFT technique is to increase controllability and observability, facilitate the test generation and fault coverage estimation process and to decrease requirements on test data volume and test time.

Early DFT techniques introduced insertion of test points into the circuit in a “for the purpose” approach. Since then various more systematic techniques have surfaced. The following chapters (2.5-2.10) will focus on some DFT techniques.

2.6 Scan-path

Regardless of how the actual read of data is implemented some form of scan chain will be required in a DFT technique. The idea behind scan-path design is that all synchronous sequential circuits can be modelled according to figure 3(a). The circuit's sequential parts (storage elements) are connected into a scan chain and the circuit is modelled as combinatorial parts connected to this scan chain.

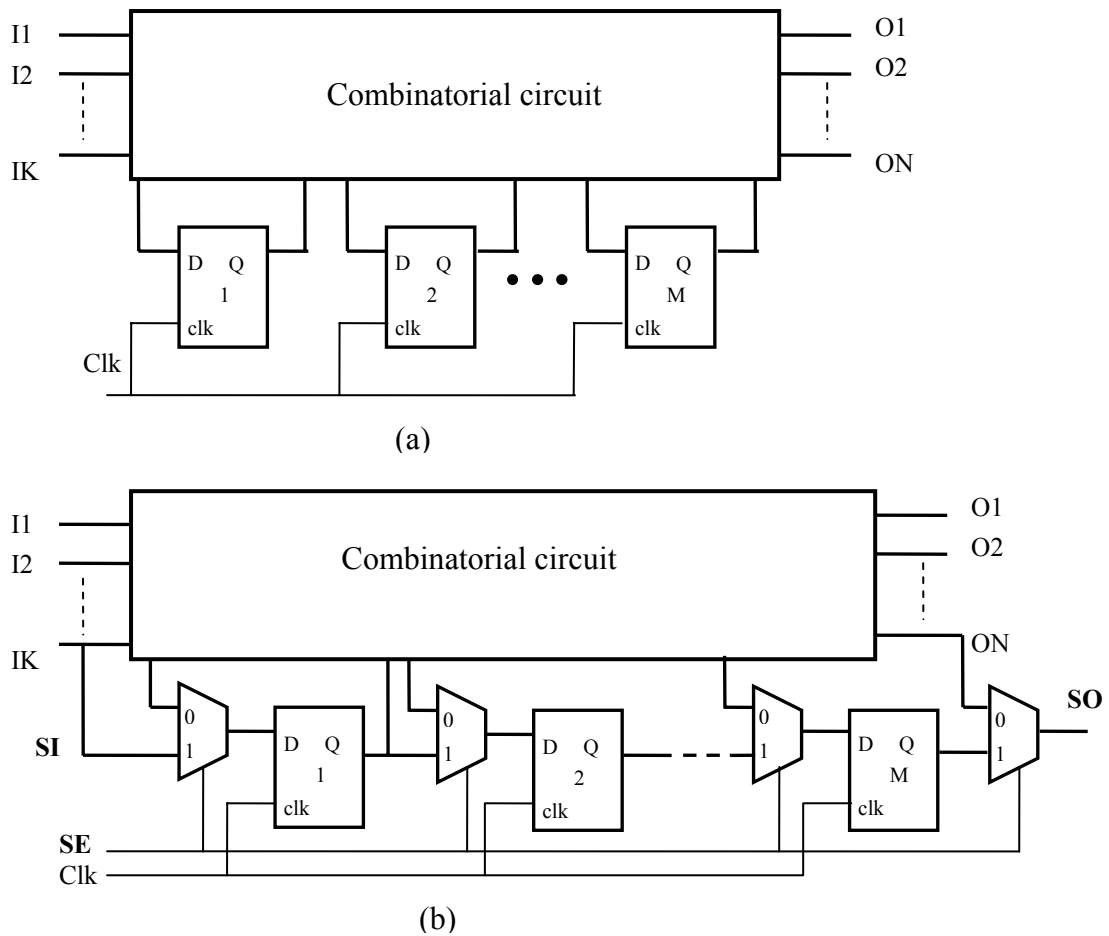


Figure 3 - Scan-Path design

In scan-path design every storage element (flip-flop or latch) is preceded by a mux as seen in figure 3(b). The select signal on all muxes, scan enable (SE), is common. For SE = 1 all flip-flops are connected as a shift register, which is shifted out serially through the scan-out (SO). Data is shifted in through scan-in (SI). SE = 0 puts the circuit in normal operation; the internal logic is not affected. This approach would implicate that scan-path requires three extra pins, SI, SO and SE. It is however possible to multiplex SI with an ordinary input and SO with an ordinary output.

The scan chain allows test input data to be inserted to points within the circuit (that are otherwise inaccessible), and to shift out inside test values that would otherwise have to be propagated to the primary outputs.

The test procedure looks as follows:

1. Set SE = 1 to put the circuit in scan mode.
2. Shift a test vector in.
3. Put the circuit in normal operation by setting SE = 0.
4. Apply the primary inputs required for the current test vector and observe the outputs.

2. Literature review

5. Capture the circuit's internal response to the test by clocking the storage elements once.
6. Set $SE = 1$ again and shift the response out through the scan chain by clocking the circuit M times (where M is the number of storage elements). While shifting out the response, shift in the next test vector.
7. Check the response.

There are alternatives to the architecture with a MUX preceding every flip-flop. It is for example possible to use two port flip-flops or double latches.

Scan-path can be arranged in different architectures, one of these is scan-set. Scan-set means that an additional test clock and select signal is introduced. The storing units themselves are not serially connected. Instead an additional chain of flip-flops, controlled by a separate select signal and clock is introduced. This way online observation of the system is possible.

A full scan chain implementation requires about thirty percent of extra chip area [4]. An attempt to remedy this is known as Partial Scan. The idea is to only grant scan capabilities to some of the storage elements. While Partial Scan leads to faster circuit operation (smaller test sets and fewer test vectors due to the smaller scan chain, avoidance of critical paths) and reduced extra area, it demands a careful assessment of which storage elements to include. This is not a simple task.

Another scan design method is the Random Access Scan. This method requires additional circuit augmentations and additional pins. (Address decoder, Scan address pin). In return it allows the storage elements (latches) to be addressed individually, thus bypassing the need to sequentially shift test vectors through the entire scan chain. The approach is outlined in figure 4.

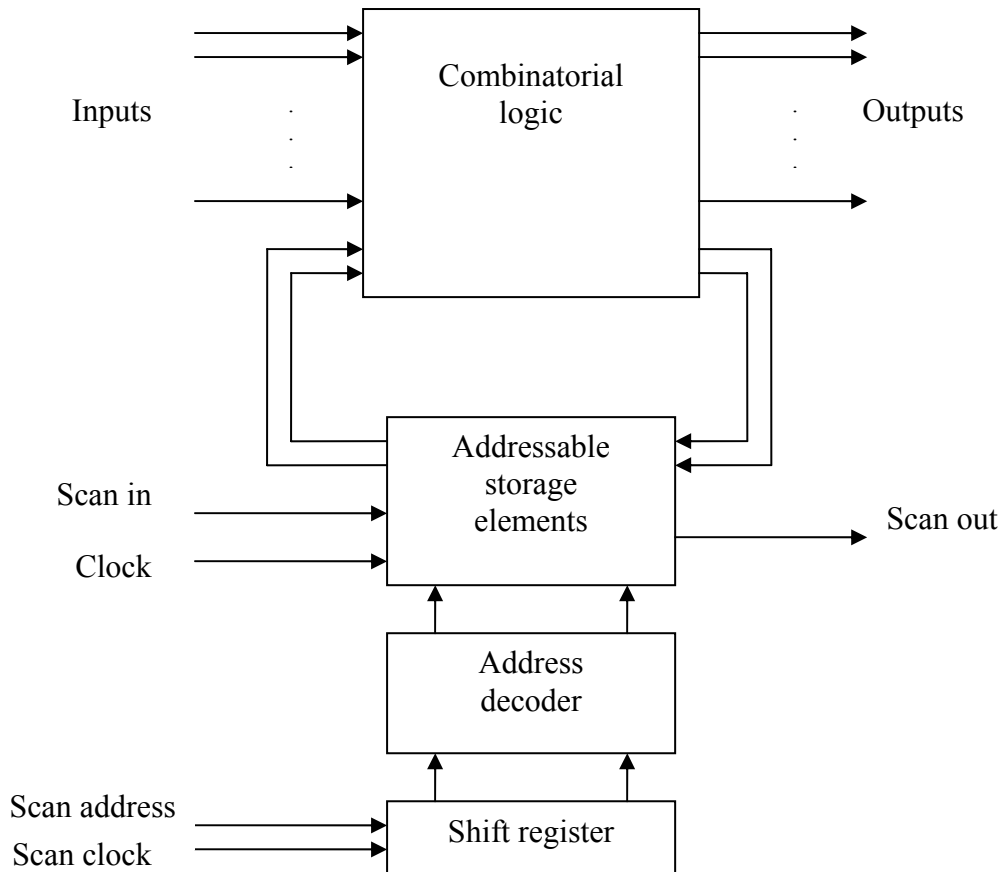


Figure 4 – Sequential circuit with random access scan.

The address decoder points to a latch that can subsequently be read or written to via the scan input/output.

In general for all scan-path approaches the time for testing can be shortened by introducing several parallel scan-chains. The cost for this of course is additional pins and extra area.

2.7 Boundary scan

The most common scan-path implementation is the boundary scan (BS) architecture. Due to the proliferation of the JTAG IEEE 1149.1 standard (see 2.7.1), it is often seen as synonymous with boundary scan. This is not the case. There are other boundary scan implementations. One of these is the CTAG P1500 (see 2.12.1), which is geared more towards testing of embedded cores and is currently under development.

Boundary Scan supports both structural and functional testing. If the purpose is to retest internal device logic as part of an assembly-level test, the test procedure is called Boundary Functional Test (BFT).

2. Literature review

The types of tests supported by Boundary Scan can be divided into three major types:

1. External: Faults related to interconnections between different modules within the DUT, such as shorts, opens, stuck-at and bridging faults are tested.
2. Internal: Test data may be inserted to internal modules via the boundary cells, built-in self-tests or any other supported instruction may be run.
3. Sample: An internal snapshot may be taken, so that internal values can be shifted out without disturbing normal operation. [1, 4, 5]

2.7.1 IEEE 1149.1 JTAG

The Joint Test Action Group (JTAG) developed a specification for boundary scan testing that was standardized as the IEEE Std. 1149.1 in 1990. This boundary scan test (BST) architecture offers the capability to efficiently test components where external test probes and “bed-of-nails” (actual probing) test fixtures are not feasible.

This BST architecture can test pin connections without using physical test probes and capture functional data while a device is operating normally. Boundary scan cells in a device can force signals onto pins, or capture data from pin or core logic signals. Forced test data is serially shifted into the boundary scan cells. Captured data is serially shifted out and externally compared to expected results.

In the JTAG standard, each IC (or embedded core or module) that is a part of the scan chain is equipped with the following components:

- A test access port (TAP) with four or five pins
- A group of registers; an instruction register (IR), a boundary scan register (BSR) and data registers (DR).
- A TAP controller; a 16 state FSM

The IEEE Std. 1149.1 BST circuitry requires the following registers:

- The instruction register, which is used to determine the action to be performed and the data register to be accessed.
- The bypass register, which is a 1-bit-long data register used to provide a minimum-length serial path between TDI and TDO (see next section), used for bypassing parts redundant for the test.
- The boundary scan register, which is a shift register, composed of all the boundary scan cells of the device.

During normal operation the additional hardware is transparent.

2.7.1.1 Test Access Port (TAP)

The TAP has four mandatory and one optional pin.

- TDI – Test data in. Serial input for instructions as well as test and programming data. Data is shifted in on the rising edge of TCK.
- TDO – Test data out. Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of TCK. The pin is tri-stated if data is not being shifted out.

2. Literature review

- TMS – Test mode select input that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur at the rising edge of TCK. Therefore, TMS must be set up before the rising edge of TCK. TMS is evaluated on the rising edge of TCK.
- TCK – Test clock.
- TRST – Test reset (optional). Active-low input to asynchronously reset the boundary scan circuit.

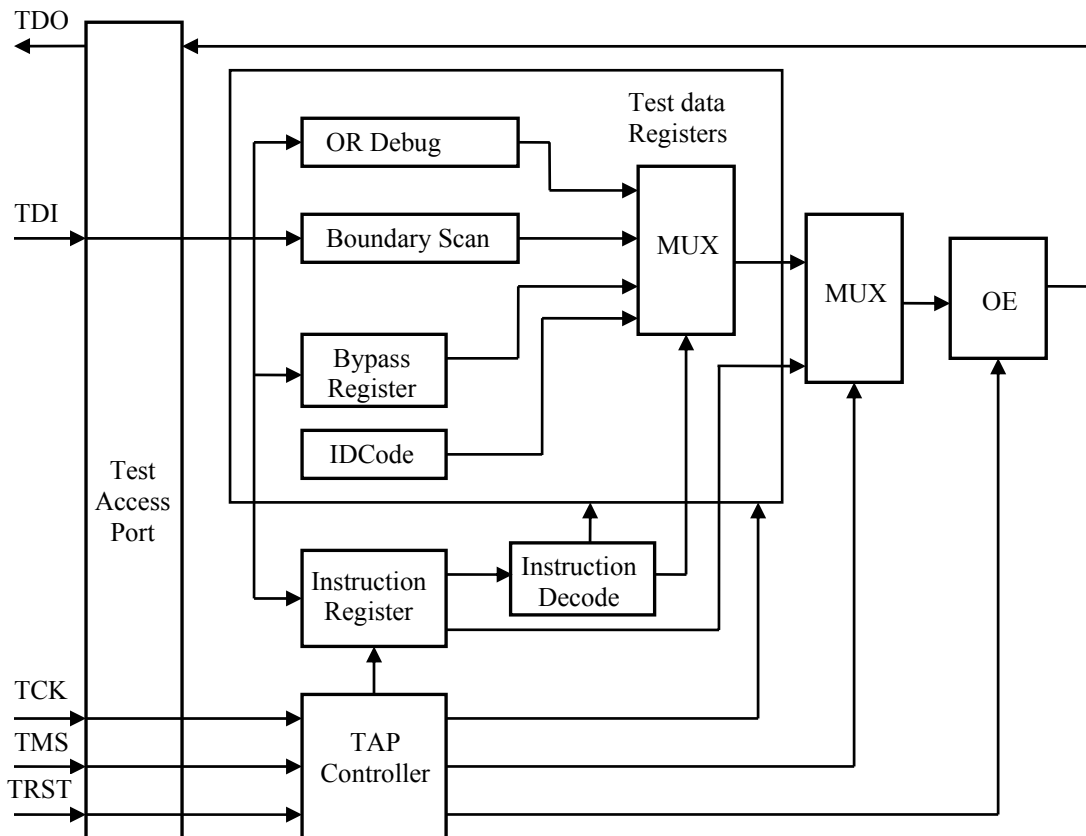


Figure 5: Test logic

Figure 5 shows the block diagram of the TAP test logic.

During test mode all input signals are scanned in through TDI and test data is shifted out via TDO. The information is shifted out through a register of flip-flops along the contour of the circuit. The TDI pin also provides data to the instruction register, which then generates control logic for the data registers. The boundary scan register is a large serial shift register that uses the TDI pin as an input and the TDO pin as an output. The boundary scan register can be used to test external pin connections or to capture internal data.

2.7.1.2 TAP controller

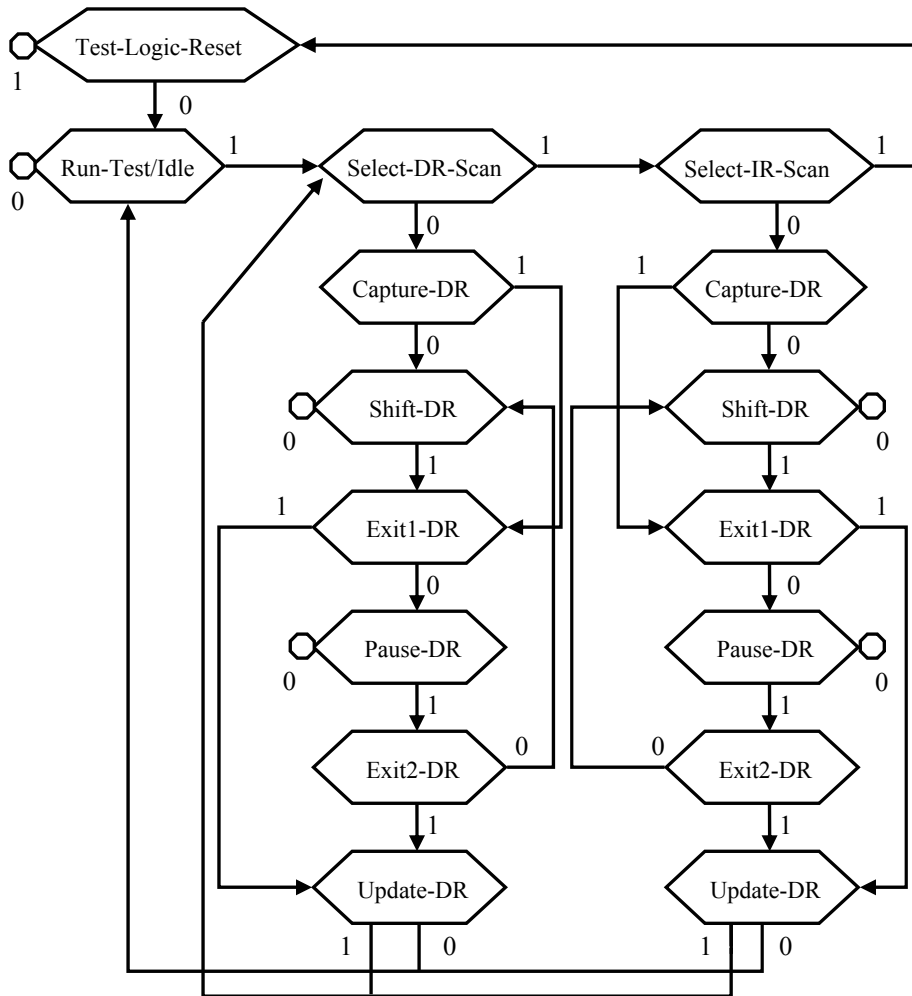


Figure 6: TAP FSM

A TAP controller according to the IEEE 1149.1 std works as a 16 state FSM. This FSM is responsible for loading instructions to the IR, applying control signals for loading and shifting test data and to perform test actions.

The main state diagram (figure 6) consists of six steady states: Test-Logic-Reset, Run-Test/Idle, Shift-DR, Pause-DR, Shift-IR, and Pause-IR. A unique feature of this protocol is that only one steady state exists for the condition when TMS is set high: the Test-Logic-Reset state. This means that a reset of the test logic can be achieved within five TCKs or less by setting the TMS input high (see figure 6). At power up, or during normal operation of the circuit, the TAP is forced into the Test-Logic-Reset state by driving TMS high and applying five or more TCKs. In this state, the TAP issues a reset signal that places all test logic in a condition that does not impede with the normal operation of the circuit. When test access is required, a protocol is applied via the TMS and TCK inputs, causing the TAP to exit the Test-Logic-Reset state and to move through the appropriate states.

2. Literature review

From the Run-Test/Idle state, an instruction register scan or a data register scan can be issued to transition the TAP, through the appropriate states shown in figure 6. The states of the data register scan and instruction register scan blocks are mirror images of each other.

To speed up testing it is possible to use multiple TAPs and BSRs. However, in order to conform to the standard it is not allowed to use separate control signals to the individual TAPs. This means that one master TAP has to be used to control the others, which has some major drawbacks as controlling the subordinate TAPs requires stepping through the states of the TAP FSM. It would be possible to connect the different components (controlled by individual TAPs) serially with common TMS a TCK signals, but as stated this causes a violation of the standard.

2.7.1.3 Boundary scan cells

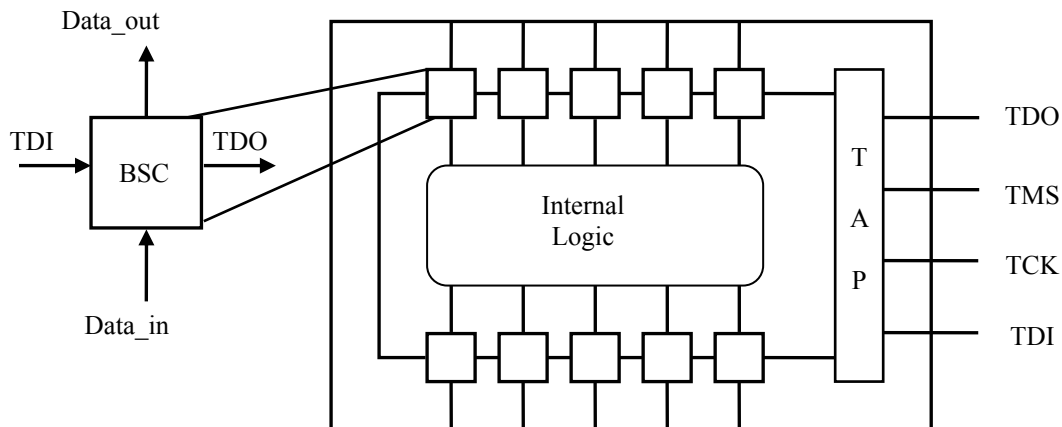


Figure 7 Boundary scan cell

The input or output signals all enter or leave the chip through the BSR, which is constituted by boundary scan cells (BSCs), as seen in figure 7. These are controlled by the instruction register as well as TMS. The composition of these cells varies depending on the use of the individual cell. The cells can be either input, output, bidirectional or control cells. The configuration of the cells is defined by the BSDL file. A BSDL file, or boundary scan description language file, describes the boundary scan architecture that is to be implemented in the DUT. [6]

2.7.1.4 Testing

There are various functions inherent in the 1149.1 std, many instructions have been defined and implemented either as public or proprietary by individual companies or groups. Some of these have been adopted into the IEEE 1149.1 standard as optional instructions. Only three of these instructions are mandatory, EXTEST, Sample/Preload and Bypass.

Sample enables a snapshot of the BSR to be taken without interfering with normal operation. Another useful function (not mandatory) is INTEST. This function allows test data to be applied bit per bit at a rate set by TCK. If a device is equipped with built in self-test (BIST) circuitry the optional RUNBIST instruction can be executed. To perform this test, the BIST instruction that tells the device to run its self-test is shifted through the TAP to the device under test. After the self-test is executed, the result is written into a device register, shifted out through the boundary-scan chain, and compared with the expected value. More on BIST in chapter 2.10. [7]

These techniques are performed by sending instructions and stimulus through the scan-path and shifting responses back through the test access port (TAP). INTEST, internal scan, and RUNBIST require device-level test patterns. Patterns can be generated by automatic test pattern generation (ATPG, see 2.9) tools for internal scan devices. BIST is mainly aimed towards fast testing of mass produced circuits.

To test a device using the INTEST instruction, data is shifted into the boundary register and applied to the device logic. Because patterns are shifted serially through the scan chain, there are practical limits on the data-application rate and on the size of the pattern set that can be applied.

To test a device equipped with internal scan capability, a special instruction needs to be designed into the device's boundary scan architecture that initiates the internal scan test mode and concatenates the internal scan register to the boundary register. To perform this test, a test vector is shifted through the boundary register to the device's concatenated internal scan register. The device is clocked, and the results are shifted out of the internal scan path through the boundary register for examination. [1, 4, 5]

To summarize JTAG:

The JTAG interface is straightforward and easy to use. Its popularity has grown immensely since the introduction in 1990. The IEEE 1149.1 is best suited for boundary scan testing aimed towards connectivity and hardware fault testing. The serial path is used both for data exchange and interface controlling.

Limiting factors

For any testing procedure that involves exchange of large volumes of data, in form of test vectors and test responses, the JTAG standard suffers some problems. The JTAG TAP offers only a one-bit wide serial test access mechanism. For large volumes of test data this renders unacceptable test application times.

Another problem arises when the DUT contains embedded cores that need to be tested in a block-by-block structure (testing the entire system directly would be very tedious). The TAP and the TAP state machine is responsible for both test control and test application. Access to the individual cores demands one TAP per core and the TAPs are connected serially. Each

TAPs FSM moves through its states in response to its TMS. Hence, serial access to the registers through TDI and TDO is only available as the FSM is in the correct predefined state. In a system with embedded cores and several TAPs, one TAP controller provides the TMS to another cores controller. This is very inefficient because signal transitions in the TMS coming from the controlling TAP require state transitions in the controlling TAPs FSM.

2.8 Automatic test equipment (ATE)

The following segments will discuss some common on-chip self-testing instances.

Fault list

The traditional ATE (or ATPG) approach requires a netlist, or gate-level description, of the circuit. To generate tests for the DUT, an ATPG tool (software) is provided with a netlist, which it uses to create a list of all faults to detect. This list is reduced in different steps until all faults are detected. Test patterns are pre-generated by an ATPG-tool and are typically stored in a tester memory and scanned into the circuit during testing.

Pseudo random pattern generator (PRPG)

Pseudo random pattern generators are used to generate random test patterns so that no pattern occurs more than once. The by far most common PRPG is called linear feedback shift register (LFSR). The LFSR (figure 8) is constituted by flip-flops that can be fed back from any stage and “x:ored” at the start of the chain. The register is clocked until the initial values are reproduced. This initial value is chosen arbitrarily. The operation of, and different incarnations of LFSRs, will not be discussed in this thesis, for a more detailed description please refer to [1].

The maximal length sequence of a LFSR has a period of $2^N - 1$, where N is the length of the LFSR (number of flip-flops). The sequence is fed to the DUT through the LFSR. Applying this sequence gives good, but not complete, fault coverage. Faults that are impossible to detect using pseudo random test patterns are called random pattern resistant. Such faults will have to be covered through other means.

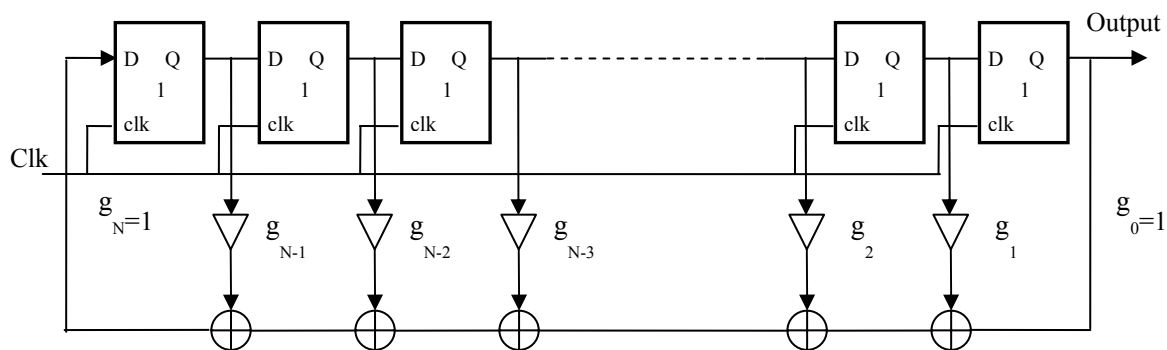


Figure 8 Fibonacci LFSR

Linear feedback shift registers make extremely good pseudo random pattern generators. When the outputs of the flip-flops are loaded with a seed value (anything except all 0s, which would cause the LFSR to produce all 0 patterns) and the LFSR is clocked, it will generate a pseudo random pattern of 1s and 0s. The only signal that is needed to generate the patterns is the clock.

The length of the sequence depends on two things, the feedback taps and the initial state. An LFSR of any given size N (number of flip-flops) is capable of producing every possible state during the period, but will do so only if proper feedback taps have been chosen. Such a sequence is called a *maximal length sequence*. The maximal length sequence of a LFSR has a period of $2^N - 1$.

Response compactor - Multiple input signature register (MISR)

A response compactor is used to compress an entire output response into a single signature for signature analysis. A very common compactor is an LFSR with several parallel input stages, called multiple input signature register. An MISR is a multi-input device that compresses a series of input patterns into a (pseudo) unique signature. The compressed response that comes out of the MISR is called the signature. MISRs are used to compare responses at the outputs of the DUT (the signature is compared to that of a good response, see 2.9.3).

2.9 Automatic test pattern generation (ATPG)

Automatic test pattern generation is a very common testing attribute. Its purpose is to generate tests for circuits after they are produced. In traditional ATPG a netlist of the design is fed to an electronic design automation (EDA) tool. This tool generates test patterns that are later fed to the DUT externally through a tester. Another approach is to let embedded test objects generate random test data on-chip.

2.9.1 Self Test methodology

Self-testing approaches vary in configuration. Test patterns can be exhaustive, random, pseudo random (no pattern repetition) or deterministic. The basic structure of any test methodology can be divided into three parts:

- Test pattern generation.
- Test application.
- Response evaluation.

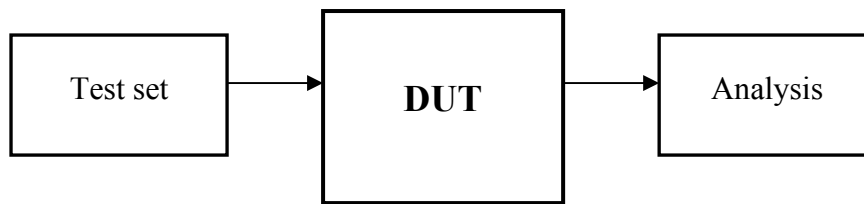


Figure 9 Generic test process

The first and final of these three steps (illustrated in figure 9) can be performed in several ways.

The test pattern generation can be done either off- or on chip. Off-chip test pattern generation means storing the test patterns in e.g. a ROM that is sequentially feeding the DUT. On-chip generation means that the circuit is augmented with extra equipment that generates the test patterns. These patterns are then applied to the DUT and the response may be compacted and evaluated by other circuit augmentations.

2.9.2 Test pattern generation

Test patterns are generated differently based on what they are meant to test and how they are meant to test it. They are generally divided into the following categories:

- Exhaustive
- Pseudo random
- Deterministic

Exhaustive tests means applying every possible input combination. This type of test grows very large with larger circuits and it is ill suited for sequential circuits, as the patterns are not applied with consideration to sequential behaviour.

Pseudo random test patterns (PRP) are generated either off-chip by a software program or on-chip by e.g. a LFSR. The patterns are generated in a fashion that prevents the same pattern from occurring twice (as opposed to completely random, where the same pattern may occur many times). Pseudo random patterns are typically used early on in the test procedure to remove easily detected faults from the fault list. Fault coverage increases with the number of test vectors, but there is a cut-off point where a larger amount of vectors is pointless. Faults that are undetectable by PRP are called random pattern resistant (RPR).

Deterministic: These RPRs can be detected with deterministic tests. Such tests target a specific fault, and due to their complexity the patterns are often generated with the help of *heuristics*. A heuristic is an algorithm that aims at speeding up the test pattern generation without guaranteeing an optimal solution.

2.9.3 Compaction and response evaluation

After the test patterns have been applied by e.g. an LFSR the response needs to be validated. In general, collecting each output response and loading it off the DUT is too impractical.

2. Literature review

Instead it is common to compress the output stream into a signature for signature analysis. This can be done in a number of ways, e.g. one-counting, parity check, transition counting etc. The most common method is to use an MISR for data compaction and signature analysis. The purpose of the compaction is to detect faults by comparing faulty and fault-free signatures. The maximum-length sequence from an LFSR gives the best fault coverage.

Aliasing

When using signature analysis the output stream is compressed to a single value. The basic principal is that the input stream is divided by the characteristic polynomial of the LFSR, resulting in a quotient (output stream) and a remainder. This is a lossy compression scheme, one specific signature can be generated by more than one input stream. The occurrence of an erroneous input stream that generates a correct signature is called *aliasing*. The probability of data loss as a result of compaction decreases as the test-length- and LFSR and MISR size-increases.

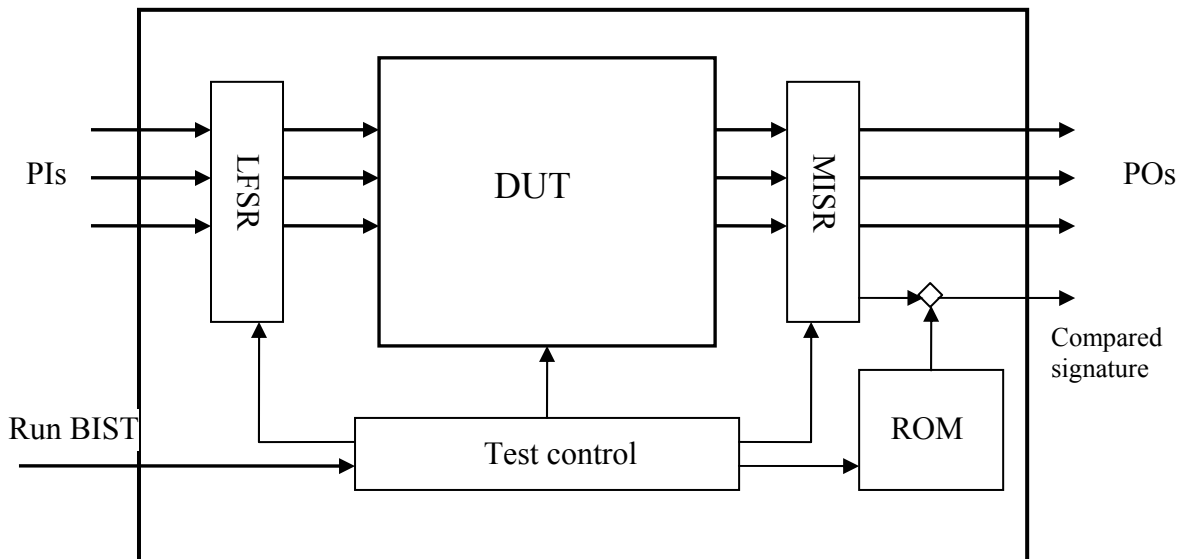


Figure 10 Traditional BIST architecture

Figure 10 illustrates a typical use of LFSRs and MISRs (where PI stands for primary input and PO for primary output). It shows the block diagram for a circuit using the JTAG interface and RUNBIST instruction to instantiate the DUT. An LFSR is used for pattern generation and the signature from the MISR is compared to a good response signature stored in a ROM. Both the test generator (LFSR) and response compactor (MISR) are built into the DUT. Typically, BIST is used in combination with Boundary-Scan. Under normal operation the extra hardware is simply bypassed.

2.9.4 ATPG algorithms

As mentioned in 2.9.2, ATPG approaches sometimes make use of heuristics. A well-known heuristic is the D-algorithm. This was the first algorithm to guarantee that a test for a fault will be generated if such a test vector exists. It introduces the “D notation”, where D is “1” in

2. Literature review

the good circuit and “0” in the faulty, and D' is “0” in the good circuit and “1” in the faulty. For example a stuck at zero fault is denoted by D , and a stuck at one fault is denoted by D' .

In order to understand the D-algorithm, and other ATPG algorithms, it is necessary to be familiar with certain operations:

- **PDCF:** The set of values on the inputs and outputs of a gate that cause the fault is called the primitive D-cube for failure (PDCF).
- **Propagation:** The process of transporting the faulty value (D or D') from input to an observable output.
- **Justification:** To place an otherwise unaffected node/line at a value that is required in order to propagate the fault is called to justify the values. This means backtracking from output to input.
- **Inconsistency:** An inconsistency occurs when the propagation and justification requires the same line to have different values.

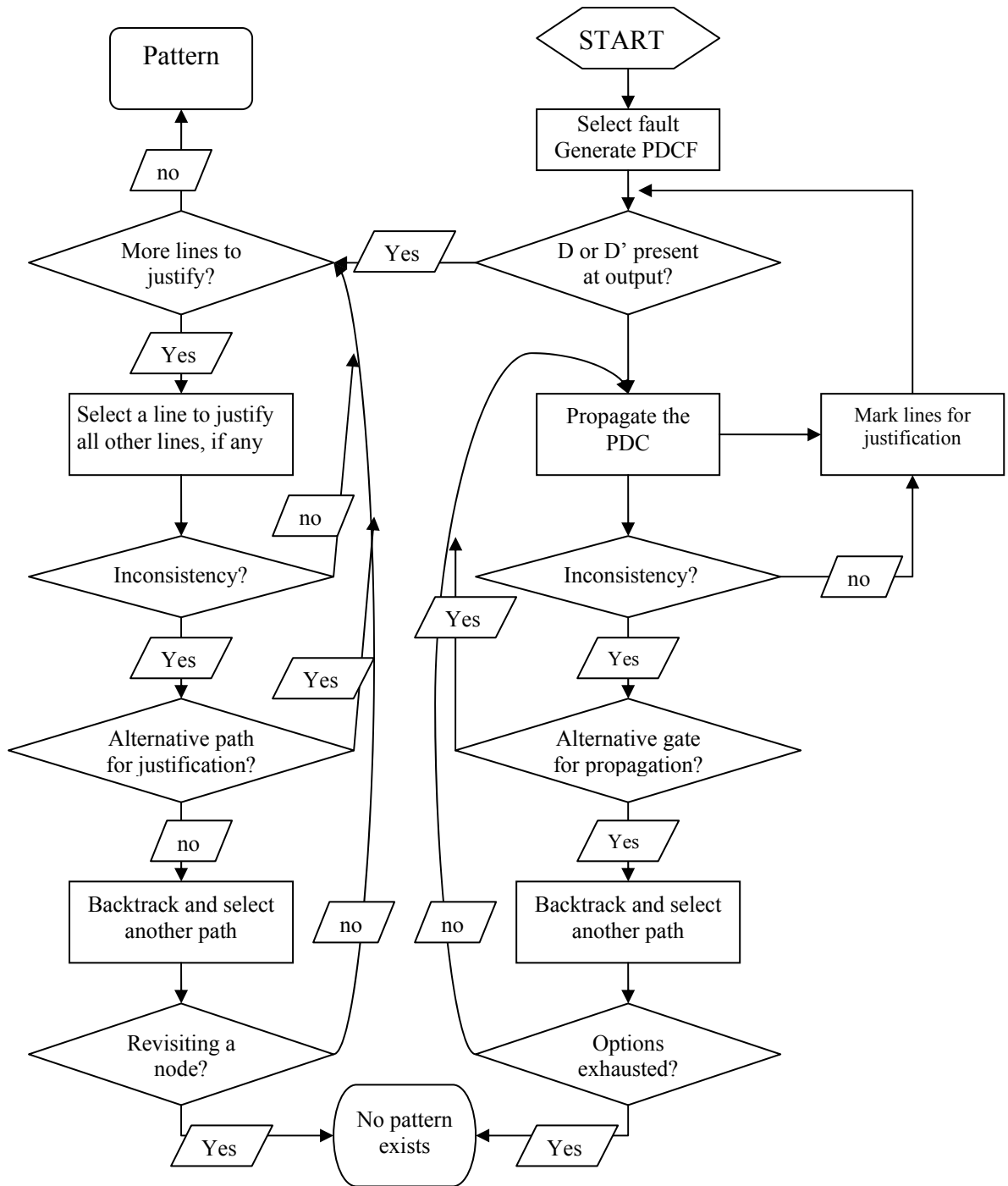


Figure 11 D-algorithm flowchart

The goal of the D-algorithm is to find an assignment of input values that will allow the detection of an internal fault at the outputs. The D-algorithm provides a means for systematically assigning input values so that a particular discrepancy is driven to an output, where it may be observed and thus detected. In other words, the D-algorithm provides a test input that is applied to a design. The output values are compared to the expected output values and any discrepancy indicates the presence of the particular internal fault that the test input was specifically designed to find.

2. Literature review

Two other types of ATPG algorithms are PODEM (Path Oriented Decision-Making) and FAN. PODEM differs from the D-algorithm in that it starts from the inputs instead of the faults. It also introduces backtracing, which reduces the number of redundant cubes.

These algorithms however, use the gate-level description of the DUT, and such a technique is not really feasible in the case of an advanced circuit. For such circuits it is necessary to find an algorithm that generates test sets from a functional description or a netlist of the circuit. The D-algorithm itself can be adapted for use on a functional model instead of a structural one (implication, propagation and justification performed on functional blocks instead of gates or lines). A functional approach suggested by Abadir-Reghbati [2] does exactly this.

Early algorithms for the functional approach used modified versions of the structural algorithms, and any loops in the circuitry was cut off so that iterative combinatorial circuits represented sequential behavior. These types of sequential ATPG algorithms are called *topological*. There are two other types of approaches aiming towards the same goal, these are called *simulation-based* and *symbolic*.

A simulation-based approach utilizes the results from a fault simulator. An algorithm is applied to the result of the fault simulation, with the help of an evaluation function and a fitness value that is assigned to each test sequence. Random sequences of vectors are fed to the fault simulator, and the evaluation function assigns a fitness value to each sequence. The fitness value is a measurement of how close the sequence is to optimal, with regards to detecting a given fault. The fitness value increases with increased observability of the detected fault. Then a new set of random sequences undergo the same procedure and, after a fixed set of iterations, an optimal set of sequences is acquired.

A symbolic approach makes use of the knowledge of both the output and the next-state function of the circuit.

In all though, very few algorithmic techniques exist for functional testing. [2, 8]

2.10 Built-in self-test (BIST)

The increased test data volume and increased effort required to generate useful tests (i.e. increased test cost), served as motivation for the development of a class of DFT methods known as built-in self-test (BIST). In BIST hardware support for test generation, test application and test result analysis is embedded as test objects within the DUT.

BIST falls under the automatic test equipment category because in its most common incarnations it uses pseudo random test generation (usually an LFSR).

Virtually all BIST implementations are based on a full scan architecture. This means that all the storage elements in the DUT (flip-flops or latches) are concatenated to form several scan chains. This way test patterns can be serially shifted in and out of the storage elements.

The test vectors may either be stored in a ROM, or generated on chip. Storing test vectors in a ROM limits the flexibility and demands large volumes of data to be stored (large ROM area). The more common approach is to generate the tests on chip. An LFSR may be used to generate pseudo random test patterns.

2. Literature review

The most widely used BIST architecture uses an LFSR as parallel shift register pattern generator (figure 12). The feedback register is supplied with a starting pattern (seed) and then cycles through a randomized (but predictable) sequence of states. The seed is the only external data required for testing, thus eliminating the time consuming external pattern application of normal scan approaches. The contents of the LFSR are shifted into the scan channels through a spreader network. The large volumes of data produced by the LFSR demands the response to be compacted, to reduce the response data to analyse. The response is captured and compressed using an MISR. The signature produced by the MISR after all test patterns have been applied is compared to a stored good signature (predicted by a simulator). Any deviance from a good signature indicates a faulty circuit. [1]

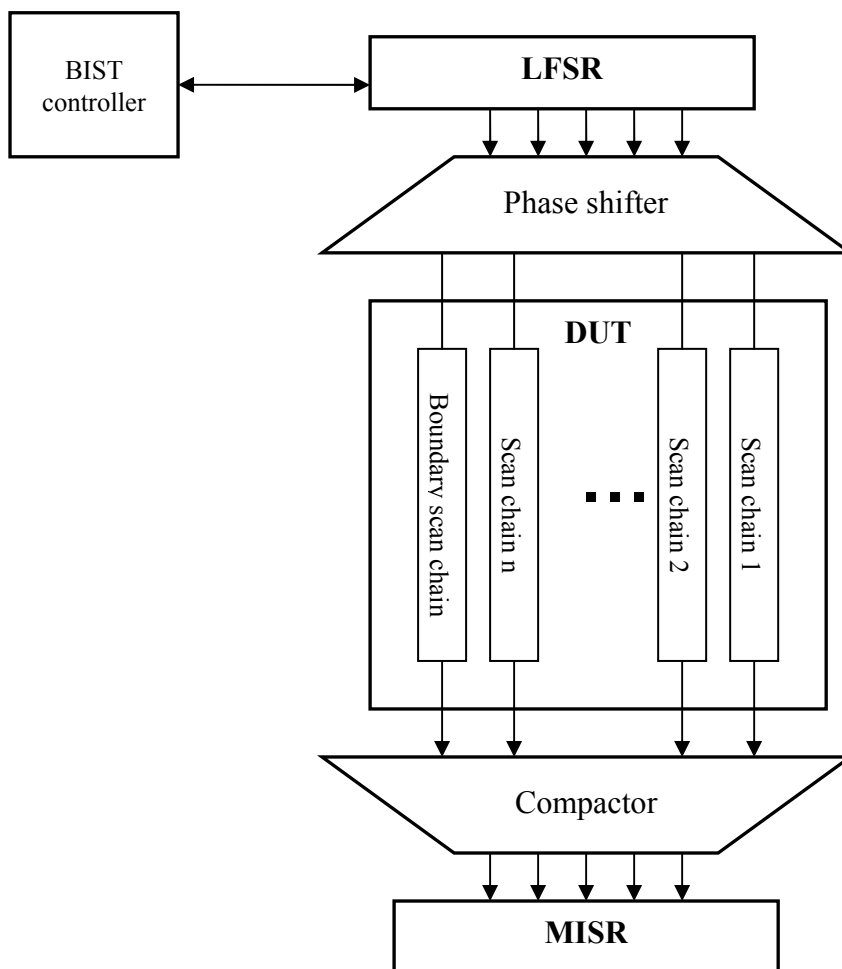


Figure 12 – Standard BIST architecture.

The BIST controller in figure 12 is used to configure the circuit for normal and test modes. During test mode the patterns are applied to the DUT from the LFSR. The spreader network, labelled phase shifter and compactor in figure 12, is just simple circuitry consisting of XORs, or other simple combinatorial circuitry, that connects the scan chains to the LFSR and the MISR. The purpose of the phase shifter is to alleviate any linear dependency in the patterns to

2. Literature review

the different scan chains (so that they are random). The fact that test patterns are generated and applied in the DUT enables at-speed testing.

BIST technology has been in use for decades. However, the initial adoption of BIST was deterred by its high extra area demands, its inability to diagnose errors (detection but not determination) and its poor fault coverage. In addition, most BIST tools are such that they are not integrated with a synthesis solution, which complicates the design flow. Also, as a result of the random patterns, the fault coverage is hard to predict.

Many advances have been made in BIST technology that improves aliasing, fault coverage and diagnostics. Traditional BIST implementations produce pass/fail verdicts only. Advanced BIST implementations produce more detailed diagnostic data as well, at the expense of additional silicon area. Solutions and methods that have improved BISTs efficiency with reasonable test patterns include test points, cellular automata implementations of the LFSR, weighting the bits produced by the LFSR and reseeding methods. In terms of improved fault coverage, the most effective method has been found to be reseeding of the pattern generator. Reseeding is a way of applying deterministic test patterns and was first introduced in 1991 [9]. In a sense, reseeding is a way of compressing input data, the test patterns are compressed into merely the seeds [10]. It can be done either by algorithmically feeding the LFSR back into a seed buffer, or by external application of the seeds. Test sequences are compacted into test signatures that constitute the seeds to the PRPG. A new approach that uses BIST for decompression/compression and is capable of applying deterministic patterns was introduced in [11]. In this approach test patterns are encoded as seeds and applied externally. Compared to traditional ATPG testing this method offers a significant reduction of test data volume and test application time. One drawback of such an approach is the complexity of finding a good balance between seeds and patterns generated from a seed.

A BIST method that decreases the extra area is the Built-in logic observer (BILBO, figure 13). BILBO utilizes existing registers and storage elements in the DUT to configure them as pattern generators and compactors, thus eliminating the need of explicit insertion of pattern generator and compactor.

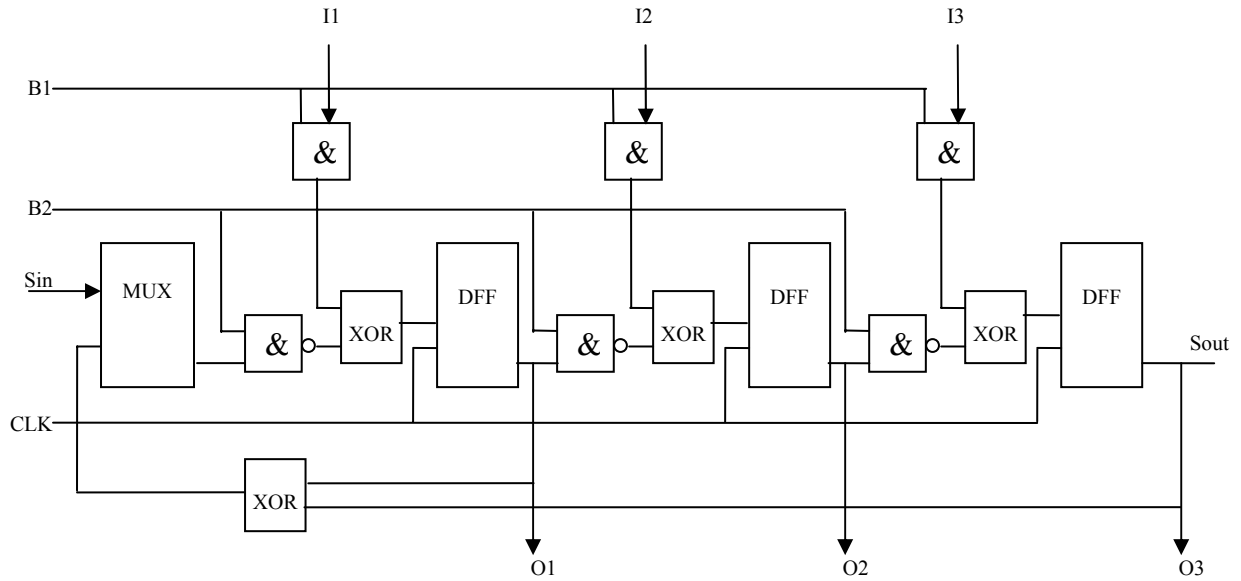


Figure 13 – Three flip-flop BILBO register.

B1	B2	BILBO function
0	0	Pattern generation
0	1	Reset
1	0	Signature analysis
1	1	Nontest

Table 1 – BILBO control

Figure 13 depicts a small BILBO register with three parallel inputs and outputs. Inputs B1 and B2 provide control of the register according to table 1, where reset means that the flip-flops are set to zero. By alternating which storage elements constitute pattern generator and compactor, BILBO allows testing of different blocks within the DUT.

The positive sides of BIST are the following:

- It enhances the speed and the efficiency of testing compared to traditional ATPG. For pseudo random tests, the test data that needs to be shifted in from an external tester is just a seed, or sometimes just a launch signal. In traditional ATPG, deterministic test patterns are pre-generated using a gate-level representation of the design netlist. These patterns are then stored in a tester memory and scanned into the circuit. The ability to encode deterministic test data as seeds renders very small test sets even for such an approach.
- A normal scan design demands two extra pins per scan chain. Using a parallel pattern generator enables insertion of an arbitrary number of scan chains using only two pins. Hence test application and execution time is significantly decreased compared to e.g. scan-path.
- BIST requires no interaction with a large, expensive external test system. The testing is all built-in, and only a small tester is needed to start the test.
- Off-speed testing, which is the case for any approach that uses an external tester to apply test patterns, causes great difficulties in handling delay faults. Delay faults are caused by signals propagating too slowly through the various instances of the circuit. Such faults may be missed by an off-speed tester. The problem is that the test patterns cannot be loaded into the DUT fast enough from an external tester. This is never a

problem in BIST testing, since at-speed testing is inherent when the test generation and application are built-in. [12]

- In terms of design changes there are two sides. Once the BIST has been implemented, a designer just makes the required changes and re-runs the random patterns. The result is a new signature, different from the previous one. On the other hand, insertion of BIST into a complex design is time consuming and complicated.

On the negative side, increasing the number of scan chains has a negative impact on routing. Other downsides are the extra area required, implementation effort required and fault diagnosis problems. Fault coverage may be poor for a complex sequential circuit such as a DSP core.

2.11 Software based self-testing

A recent trend in self-testing is an alternative to the traditional hardware based built in self-test that is (partially) software based. This reduces the extra area and performance loss of a strict hardware self test methodology.

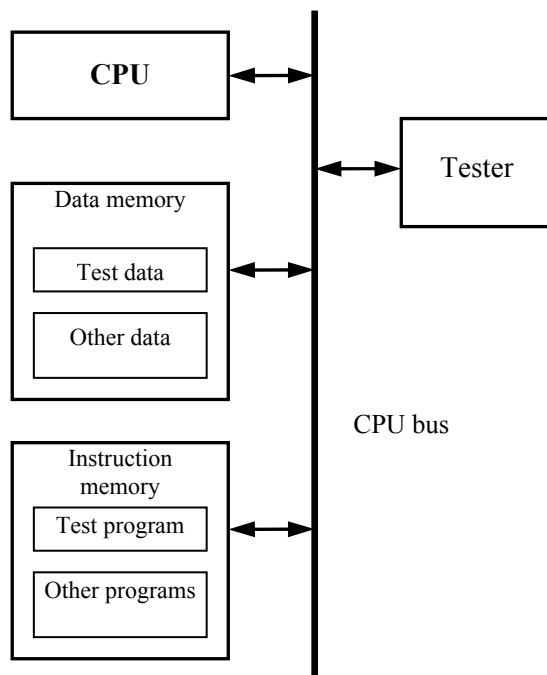


Figure 14 – Generic Software based self-testing outline.

Figure 14 shows the basic outline of software-based self-testing. The only test resources needed are test program/data storage in respective memory and the targeted processors instruction set and behavioral description.

In [13] an alternative that uses the targeted processors existing functionality and instruction set in order to perform self-test is introduced. This approach does not require any additional internal augmentations to the design. The limiting factor in this approach is the time it takes to download the test set from an external tester, which might be (is) slow. That being the case, a main objective, aside from fault coverage, is finding small test sequences. It is further claimed

in [13] that the proposed methodology implicates both low test development cost and low test application cost.

If the self-test methodology is based on the RT-level description of the target and the targets instruction set, it becomes technology independent. Self-test methodologies normally take on an abstract approach, using pseudo-random instruction sequences. This leads to large test sets and poor structural fault coverage. A more interesting approach, as in [13], uses small deterministic test sequences. It has been proven to display better fault coverage than the generic approach when applied to the same target processor.

2.11.1 A software based self-test methodology

The proposed methodology is divided into three stages.

1. Classification of components.
2. Ordering of components.
3. Test routine development.

Classification and ordering of components

During the classification stage each processor component is assigned one of three classes: *functional*, *control* or *hidden* component. Functional components are those that perform operations on data, such as e.g. the ALU, register-file or a multiplier. Control components are those that administer the flow of data, e.g. the program counter. Hidden components are added to increase performance but are not visible to the assembly language programmer, e.g. the pipeline logic.

During the ordering of components stage each component is given a priority based on the test priority criteria. This criteria is based on component size and the components controllability/observability. A larger component will have a larger contribution to the overall fault coverage. The functional components are normally larger than the control- and hidden components.

For a processor component, controllability is defined as the shortest instruction sequence required to apply a test pattern to the component inputs. Subsequently, observability for a processor component is defined as the shortest instruction sequence required to propagate a components outputs to observable outputs of the processor. Functional components also have the highest controllability/observability since their inputs/outputs come from internal registers or external data ports, whereas control components inputs/outputs come from communication between functional and control components. The test procedure then takes on the form illustrated in figure 15.

2. Literature review

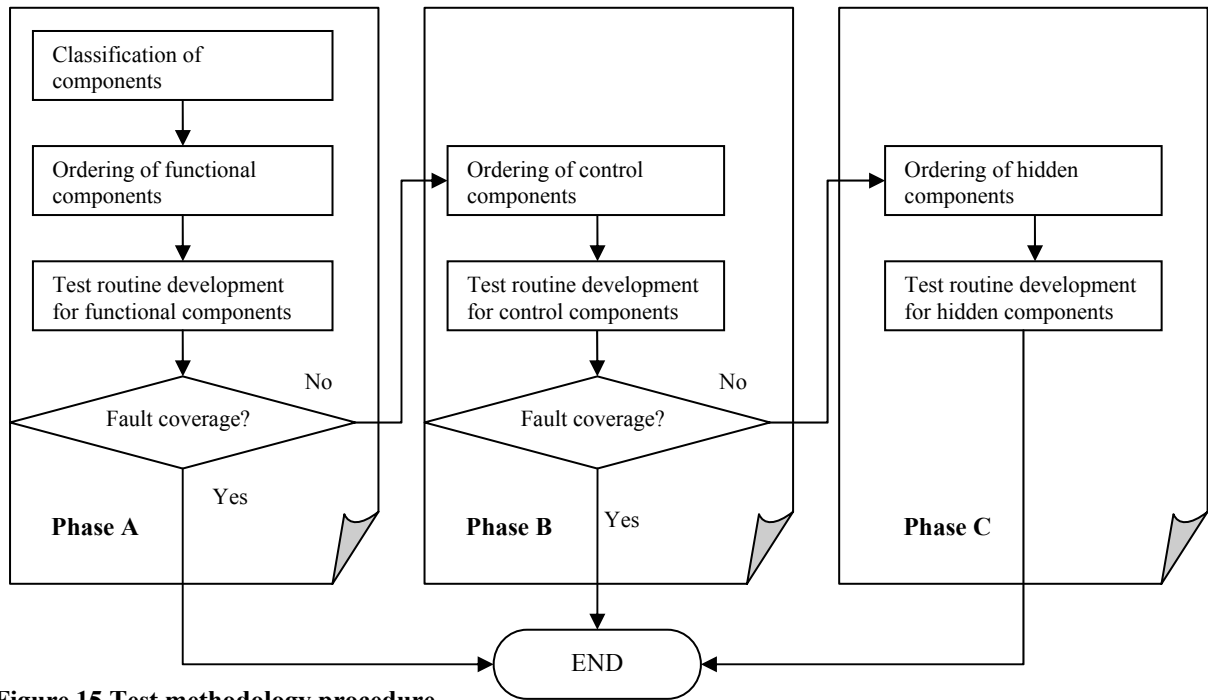


Figure 15 Test methodology procedure

Where Phase B and C are not performed if Phase A alone provides sufficient fault coverage. The functional components of phase A are easily accessible and thus the required test development effort for phase A alone is small. The high controllability/observability of the functional components also implicates small test program sizes. As such, if phase A provides sufficiently high fault coverage then the test effort and test cost are minimal.

Test routine development

During test routine development each component is dealt with separately. Based on the DUTs instruction set architecture and RT-level description, the components operations and the instructions necessary to excite them are determined. Based on this, test sets are developed for each component. The final step is to combine these sets into self-test routines, while taking into consideration component regularities that can be utilized to decrease the test size. The procedure is illustrated in figure 16.

2. Literature review

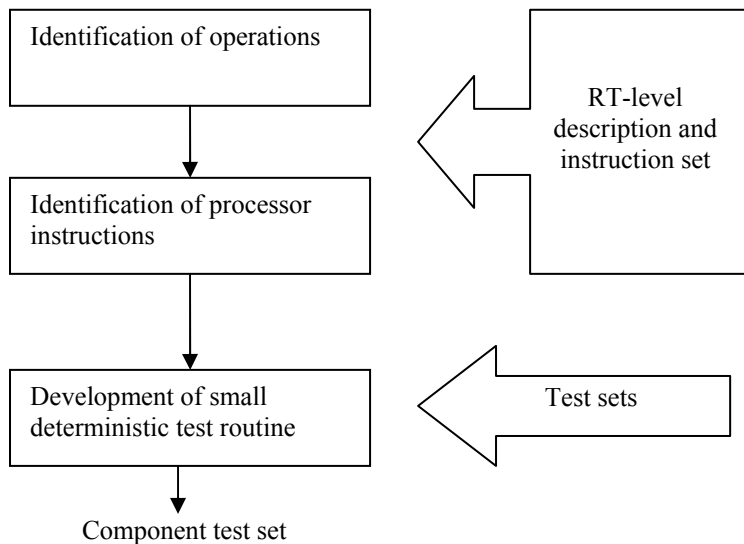


Figure 16 Component test set development procedure

The test procedure can be realized with e.g. the MentorGraphics suite, using Leonardo, ModelSim and FlexTest for synthesis, functional- and fault simulation. When the methodology was applied to a RISC processor with a 3-stage pipeline Phase A alone resulted in a 91.1 percent fault coverage. Adding Phase B gave an increase of 1.1 percent.

The positive sides of this kind of approach are:

- No, or virtually no, modifications required to the DUT.
- No extra area or performance penalties implicated.
- Only requires a behavioral description and an instruction set architecture.
- High fault coverage for complex circuits compared to ATPG or BIST.

2.12 Testing of embedded cores

Traditional IC test development is carried out by one person, or a closely cooperating group of persons within an organization. In traditional IC design everything is built from scratch. In an attempt to avoid this time consuming task, a design style based on large reusable modules, embedded cores, has developed over the last years. This design paradigm is known as System-on-Chip (SoC). The biggest advantage of using embedded cores is the shortened development time that results from design reuse. Core-based testing might occasionally lead to a situation of distributed test development. Multiple groups of people, spread over different companies, time zones, and continents, are jointly responsible for the test development of large, highly complex circuits.

Although design reuse is very attractive, the techniques to apply it are still in their infancy, and hence the practical implementation of the core-based design scenario is fraught with many unresolved issues. One of the biggest challenges of this emerging discipline is testing.

2. Literature review

The new domains that need to be issued are; core test knowledge transfer (in the case of distributed development), test access to embedded cores and chip-level SoC tests and their optimization.

An SoC typically consists of several types of cores, such as processors, random logic, RAMs and analog blocks. Traditional testing methodologies, such as ATE, lack the capability to take this block structure into consideration. Such methodologies are not scalable with the increasing complexity inherent in SoCs.

Any test methodology dedicated for SoCs invoke a scalable block-by-block based testing scheme. One way to achieve this is by embedding test objects and coupling them with corresponding cores. Typically an embedded test solution for SoCs consists of a set of embedded test objects (for logic, analog blocks, RAMs etc), that are distributed over the SoC through dedicated test access mechanisms (TAMs). These test objects could be accessed (although not ideally, see 2.7.1.4) through e.g. a JTAG TAP.

Testing of systems containing embedded cores typically involves three generic elements, these are:

1. Test data source and sink.
2. Test access mechanism (TAM).
3. Core wrapper.

These generic elements are illustrated in figure 17.

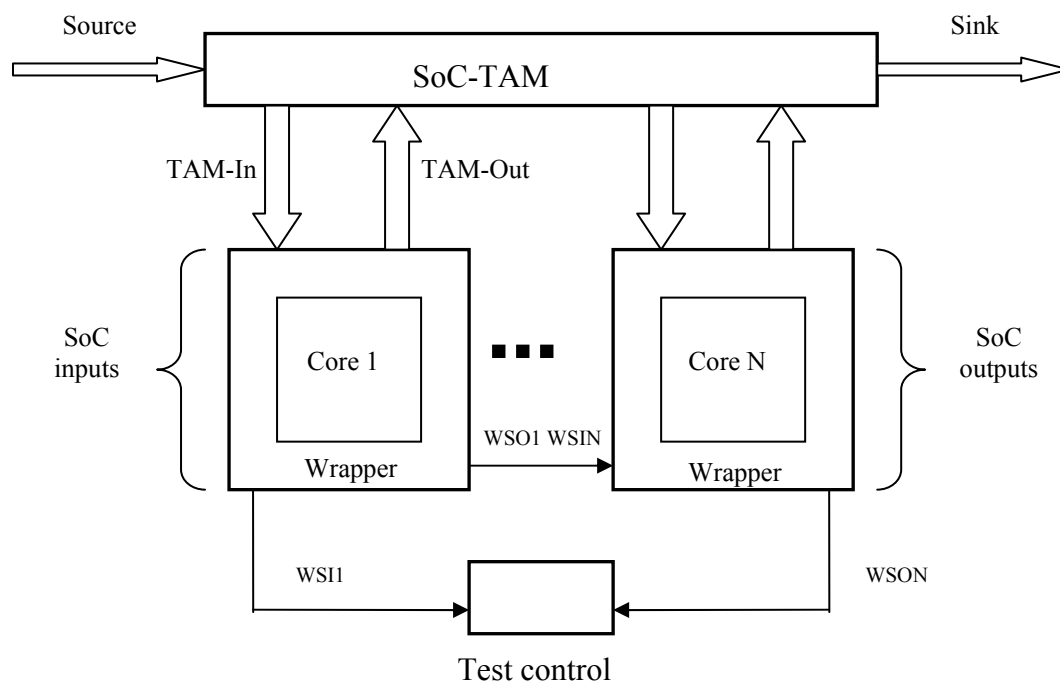


Figure 17 – Core testing outline.

1. Test data source and sink

The test source is responsible for generation of test stimuli and the test sink is responsible for reception and evaluation of test responses. Source and sink can be off-chip, on-chip or a combination of both. Traditional ATE would be implemented as off-chip source and sink.

2. Literature review

Test data generators and response evaluators, as in BIST, would mean on-chip source and sink implementation.

External ATE requires TAMs from pins to core inputs (terminals). Inherent in this, especially for cores of which testing requires large volumes of high-speed data (that would require wide TAMs), is significant extra chip area.

2. Test access mechanism

The test access mechanisms responsibility is to transport test data from source to core under test to sink. A single IC may contain different TAMs. The key TAM parameters are its width and length. The width determines the bandwidth, which should always meet the minimum requirement set by the data rate of the core under test. The length of a TAM is the distance between what it interconnects. Shorter length means less extra area. On-chip source and sink and sharing of TAM between multiple cores reduce the total TAM length.

For off-chip sink and source, the TAMs must connect pins to core terminals, so that the cores are accessible. One common and direct way to achieve this is to multiplex the core terminals to the IC pins. This way a core can be tested as if it was a standalone IC. There are two disadvantages to this method though. It is not scalable, meaning that testing becomes infeasible if the core has more terminals than the IC has pins. Also, for an SoC with many cores, multiplexing all cores to IC pins renders high extra area.

A more sensible solution would be to reuse existing resources as TAMs. One example that has been used on RISC processors is to let the system bus transport the test data from external pins to the core under test via the external bus interface. Another proposal is to impose a scan chain around the cores and use JTAG as TAM. JTAG however only allows a one-bit wide TAM, which has to tend to both test control and test data application. This could possibly lead to impossibly large test application times. Another idea is to use a dedicated test bus as TAM.

3. Core wrapper

The core wrapper constitutes the interface between the core and the chip environment. It can be set to three standard modes:

- Normal operation: The wrapper is transparent and the core is connected to the SoC.
- Core internal test mode: The TAM is connected to the core, allowing test application and response observation.
- Core external test mode: The TAM is connected to interconnect wiring and logic, allowing test data to be applied at one cores outputs and responses to be observed at the next cores inputs.

Other than these three generic modes, modes allowing individual cores to be detached from the chip environment, or bypassed through the TAM might be useful.

Since a single TAM is often used for several cores, the TAM width does not necessarily match the number of core terminals. The cores function determines the number of terminals, whereas the TAM width is determined by the minimum bandwidth required, as well as how much extra area is afforded. When the number of terminals does not match the TAM width, width adaptation is done in the wrapper through serial-to parallel at the inputs and parallel-to serial at the outputs. [14, 15]

2.12.1 IEEE P1500 CTAG

There is currently no standard for testing of circuits containing embedded cores (that may originate from multiple sources). This served as a motivation for the development of a new standard, currently being crafted by the IEEE P1500 group. [16]. The standard, entitled P1500 or CTAG (from Core Test Action Group), will be independent of the underlying functionality of the circuit or the individual cores. The method will allow automatic identification and configuration of testability features in circuits containing embedded cores (SoC:s). P1500 does not standardize the core's internal test methods or DFT, nor system-chip test integration and optimization issues, such as the type of source, sink, or TAM.

The P1500 group is working to meet the following goals for the standard:

- To standardize a test architecture for embedded cores.
- This architecture defines a (core) test interface between embedded cores and system chip.
- Define testability measures for SoCs.
- Facilitate test reuse for embedded cores through core access and isolation mechanisms.

Test facilitation means defining standards for core test mechanisms, such as protocols and test control implementation. This means that the standard will support various test methods that the individual cores might be designed for (BIST, scan etc.). P1500 encompasses a standard language, called core test language (CTL) that is intended to describe any test-related information about an embedded core. The information conveyed through the CTL includes data on internal DFT, test stimuli, expected responses, fault coverage, test modes etc.

As mentioned one goal of the standard is to provide a method for testing cores individually. This way reuse of tests and test sets is possible for cores that have been tested successfully in another environment.

P1500 Architecture

A wrapper as in figure 18 encapsulates each core, or component. The TAM-in in figure 18 comes from the SoC TAM, and is defined by the SoC designers. Depicted in figure 18 is a single core with wrapper. All such cores are connected to the SoC TAM, and can be isolated for access during testing.

The source to the SoC TAM may come from chip I/O, test-bus, test-port, BIST etc. The wrapper interface port may be controlled via chip I/O, internal TAP controllers or possibly other.

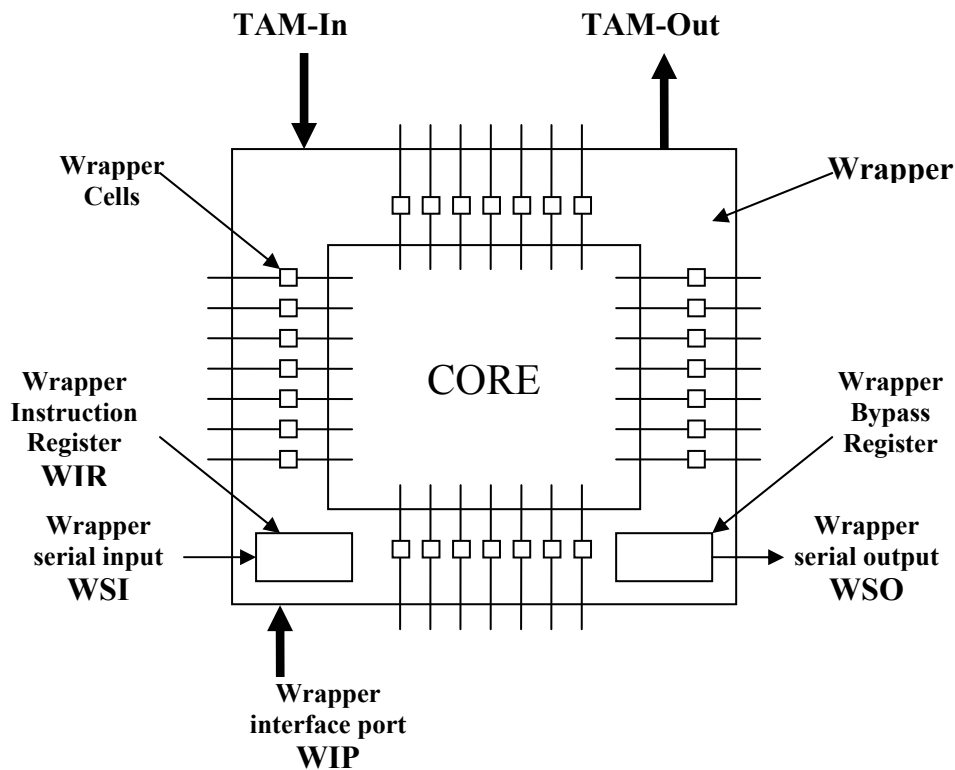


Figure 18 - P1500 Wrapper

The wrapper allows easy integration of the core into a system chip design. It is equipped with:

- A Wrapper Instruction Register (WIR), for controlling wrapper mode.
- A Wrapper Boundary Register (WBR), for providing test vectors.
- A Bypass register.
- A Wrapper Interface Port (WIP), for serial control.

Test control is provided by the WIP, which is used to access the WIR, Bypass, Data and other registers. Actions are categorized either as wrapper data register (WDR) or wrapper instruction register (WIR) instructions. These instructions correspond to the two branches in the JTAG FSM.

CTAG is basically an alternative version of JTAG that has been proposed as a standard to provide test access for embedded cores. It has two major departures from the 1149.1 standard. Firstly the TAP FSM is replaced by independent control signals to enable IR and DR instructions, thus it more efficiently supports multi TAP systems through a serial interface layer (SIL). Secondly it allows using shared I/O-cells in the boundary scan registers, which can be used to implement either boundary scan- or normal system functions.

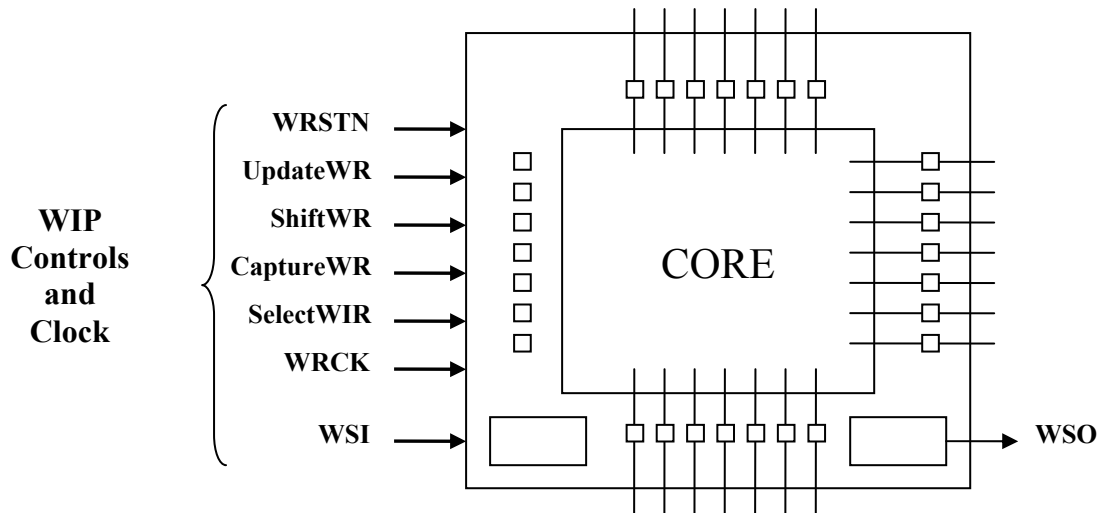


Figure 19 - Wrapper interface port

WIP

The CTAG Wrapper interface port (WIP, figure 19) contains test signals CaptureWR, ShiftWR, UpdateWR and SelectWIR. These correspond to the internal states of the JTAG TAP FSM. [16]

To summarize CTAG:

CTAG is simply put JTAG with the FSM replaced by separate state signals. This way it is intended to be more apt for embedded cores- and hierarchical test access. If a design is made up largely by cores that can be tested by reuse of already verified tests, then CTAG will cut testing time and test effort significantly. Much like the SoC paradigm has cut design time and effort significantly.

3. Simple case study

The widely accepted System-on-Chip (SoC) design usually consists of an embedded processor core with surrounding cores for program and data storage, communication etc. SoC designs in general lead to a richer functionality and reduced development time. One thing that is in no way facilitated by SoC design though, is testing. In the case of external testing the test data volume required is excessive. Besides, the gap between external tester frequencies and chip operating frequencies makes at-speed testing impossible. The opposite extreme, BIST, makes at-speed testing possible. However, hardware self-test for SoCs implicates significant extra area and performance reduction that might not be affordable. A more sensible approach for testing of SoC designs seems to be a software-based self-testing methodology.

As this thesis is mainly meant to evaluate testing schemes applicable on DSPs or SoCs, it seemed pertinent to take a closer look at a methodology that has been successfully implemented on a real DSP core. This method is described in this thesis to illustrate the process of developing and executing a deterministic test methodology. This to give a more tangible illustration of some of the subjects that might have seemed a bit abstract in previous chapters.

3.1 An instruction-level based processor testing approach

In [17] a test program design method intended for DSP cores, that requires no insertion of scan chains, is introduced. The methodology is an alternative (self proclaimed favourable one) to traditional ATPG and BIST techniques. Like the technique in chapter 2.11 its testability metrics (controllability, observability, fault coverage) are based on the targets instruction set. The method uses a program template on the instruction set to generate the test program. The test program is constituted by a sequence of instructions, which are executed as a loop a number of times, with different random data (LFSR generated) stored in the DSP core. The test architecture is depicted in figure 20.

3. Simple case study

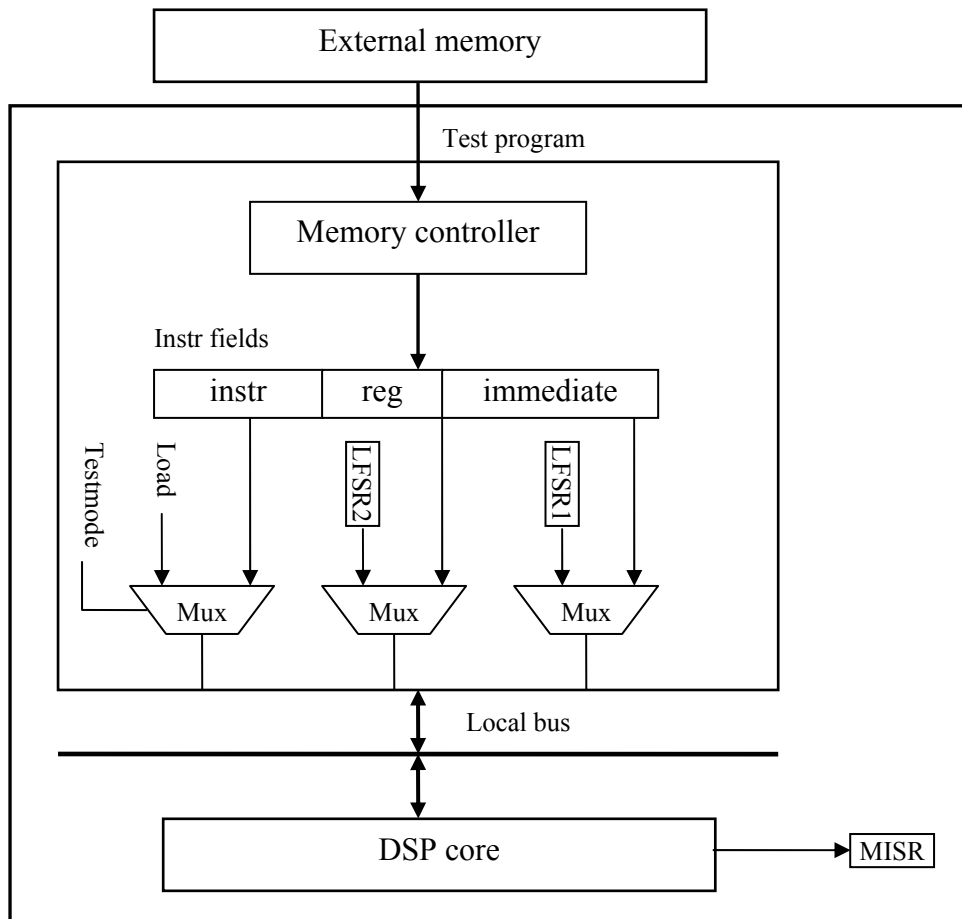


Figure 20 – Test architecture.

Figure 20 shows the test insertion approach. The instruction stream between memory and DSP core is modified for insertion of random data from LFSR1 and LFSR2. The instructions immediate field is filled with data from LFSR1. This pseudorandom data is transferred to the DSP core by transforming unused opcodes from the DSP architecture to normal **load** instructions. Feeding LFSR2 (by “X-Oring” it) to the instructions register field is used to alternate the registers being used by the test program. Thereby exercising a different group of registers each loop of the test program and hence covering each of the registers that make up the register file. The core output is fed into an MISR for response analysis.

The objective of the test program is to exercise core components classed as datapath components (like the functional classification in 2.11), and to propagate possible errors to core outputs (registers). The test patterns in this methodology are not the instructions derived from the template, but rather the random data that is produced by LFSRs and is loaded to the core in between loops.

During testing the core must be initialised to a known state. To get it into this state the instructions in the test program are preceded by a group of instructions. (These instructions are excluded from the loop and are only run once)

3. Simple case study

The test program generation flow is divided into two parts. Step 1 is the construction of a metrics table and step 2, which is the part when the instructions are chosen, based on the results of step 1.

Step 1 – Construction of testability metrics table:

The purpose of this step is to produce a table of the controllability and observability metrics that any given instruction renders for any component in the behavioral datapath. The controllability and observability are derived from behavioral simulation.

In this testing approach testing is not based directly on fault coverage. Instead coverage is determined by controllability and observability. A component is *covered* when an instruction (or a sequence of instructions) generates metrics greater than chosen threshold values for controllability and observability. These thresholds are normally initially set to 0.7 for controllability and 0.5 for observability.

The controllability is derived from the entropy, $H(X)$, which measures the uncertainty of a variable value (component X , the variable is the different input values) [18]. Each components entropy is measured by applying a sequence of instructions to the DSP cores inputs. Controllability for component X for a given instruction is then defined as

$$C(X) = \frac{H(X)}{n},$$

where n is the number of inputs to the component and $H(X)$ is the entropy for that component for a specific instruction, see table 2. Controllability can vary from 0 to 1. Controllability differs from instruction to instruction. A low controllability means it demands a greater effort by the LFSR to cover the component.

The observability metric describes the ability to propagate an error on the components outputs to observable outputs (DSP core output, register). The observability is calculated by dividing the number of errors detected at the cores output, n_d , by the number of purposely simulated errors at the components output, n_s .

$$O(X) = \frac{n_d}{n_s}.$$

Not all possible erroneous values are simulated, that would be exhaustive and take too long. Instead a heuristic is used that generates $2n$ simulated errors for every good simulation, where n is the number of outputs of the component. What this means is that for every good simulation $2n$ simulations are run with a random erroneous value replacing the actual output of the component. So if for example 1000 good simulations are run, and then $1000 * 2n$ faulty simulations, and if 8000 errors are detected at the output for an 8-bit signal, then the observability is

$$O(X) = \frac{8000}{1000 * 2 * 8} = 0.5.$$

The observability varies from 0 to 1, where 1 means that every erroneous value at the components outputs results in an error at the cores output.

3. Simple case study

		Components							
		comp 1	comp 2	comp 3	comp 4	comp 5	comp 6	comp 7	comp 8
Instructions	instr 1	0.77/0.80	0.77/0.88						0,85/0,96
	instr 2	0.72/0.92	0.71/0.66						0,88/0,96
	instr 3	0.70/0.99			0,75/0,88		0,77/0,88		0,72/0,66
	instr 4	0.81/0.96			0,78/0,88		0,77/0,88		0,74/0,66
	instr 5	0.78/0.66		0,88/0,66			0,82/0,90	0,00/0,88	0,82/0,60
	instr 6	0.74/0.58		0,78/0,99			0,84/0,90	0,00/0,78	0,84/0,60
	instr 7	0.84/0.40	0,40/0,12			0,68/0,55			0,64/0,99
	instr 8	0.47/0.70	0,34/0,32			0,72/0,60			0,64/0,99

Table 2 – Controllability/Observability metrics table.

The controllability and observability derived from simulation constitutes the metrics table (table 2). This table shows the controllability and observability each instruction (row) implicates on each component (column). The metrics table in table 2 is just there for illustration, an actual metrics table is much larger.

Step 2 – Instruction sequence selection:

The next step is to look at the metrics table and generate the test program accordingly. This is done in two phases. First off, initial controllability/observability threshold values are chosen. The purpose of the first phase is to step by step delete as many columns as possible in the metrics table by choosing instructions that cover them to be included in the test program. Since each test sequence is preceded by a **load** instruction (to insert random data into the core) and ends with an **out** instruction (to observe the outputs), the columns that are covered by **load** and **out** are the first to be deleted from the table. The same goes for any columns covered by instructions used to initialize the DSP to a known state. Then the instructions that cover the most columns are chosen in decreasing order and subsequent columns are deleted from the table. This process continues until all columns have been deleted or there are no more instructions to choose from. Columns remaining after the initial phase are dealt with in phase 2.

The objective of phase 2 is to find a sequence of instructions that covers each of the components remaining in the table after phase 1. The easiest way to do this is to find a single instruction that meets the controllability-, but not the observability, threshold for the uncovered component. Then to meet the observability threshold attempt to find a sequence of instructions that sufficiently propagates the outputs of the component to DSP core outputs. What this really means is that the components metrics are improved by using a sequence of instructions. This step also eliminates columns that lack controllability, i.e. their control bits are not set accordingly by any instruction.

For example, if table 2 is used with tresholds values 0.70/0.50 the test generation process would go as follows: After the columns that are covered by the **load** and **out** instructions have been deleted, the instructions that constitute the test program are chosen by a covering algorithm, choosing the instructions that cover the most columns in descending order. In the metrics table above the first instruction to be chosen would be instruction 5, since it covers four columns. As seen in table 2 component 5 is not sufficiently covered by any individual instruction (no instruction meets the observability threshold of 0.5). This means that a sequence of instructions has to be chosen to cover this, and any other component like this, until the final test program is completed. Table 2 also conveys that component 7 lacks

3. Simple case study

controllability at all. This shows the main drawback of an approach based solely on the targets instruction set architecture; such components are impossible to exercise with processor instructions and are deleted from the table.

Experimental results:

The methodology has been implemented on a DSP core with a 4-stage pipeline. The targeted DSP had a 17-bit instruction format illustrated in table 3:

16-12	11-4	3-0
Opcode	Datafield/Source registers	Destination Register

Table 3 – Instruction format

Where bits 11-4 correspond to the immediate field in figure 20, and function as either Datafield or the address of two source registers, depending on instruction.

An industrial compiler (Synopsis) was used on a VHDL description of the DSP to get a gate-level verilog netlist. This to enable fault-simulation (which requires gate-level knowledge) for evaluation purposes (the methodology otherwise does not require gate-level knowledge of the DUT). This netlist, along with test patterns acquired through simulation of the LFSRs, were fed to a fault simulator (Synopsis Tetramax) to compute the fault coverage. A test program of 37 instructions simulated through 6000 iterations gave a fault coverage of 98.14 %, which is significantly higher than a normal ATPG or BIST implementation would generate for the same DUT. When applying traditional ATPG testing using the same program suite, the ATPG tool could only manage to render a fault coverage of 8.51 %. This goes to show the drawbacks in automatic test equipment. ATE suffers severe difficulties in generating good test patterns for complex sequential circuits.

Perl scripts were used to compute the metrics table. The scripts modify the VHDL code and simulate it using a VHDL simulator, which results are used to compute the controllability and observability.

The pseudorandom data from the LFSRs is used as operands for other instructions.

Evaluation:

The advantages of a software-based self-testing technique compared to traditional ATPG with scan chains or BIST can be summarised as follows:

- Minimal insertion of intrusive extra hardware (no scan chain insertion).
- Minimal knowledge of the internal structure of the DUT is required.
- Higher fault coverage for complex sequential circuits.
- Instruction level testability metrics prevents accessibility problems to components that might pose problems for other approaches.
- Very small test program that operates iteratively with different random data in the DUT.
- No modifications of the core are required.
- Only instruction set and high-level description required.
- The very small test program virtually eliminates the problem with slow external testers. Test application becomes at speed since the test program is stored on-chip in the instruction memory.
- No performance- or area penalty.

3. Simple case study

On the negative side:

- Software-based self-testing is quite new and is not yet as established as traditional ATPG with scan chains or BIST. The first complete application of such a methodology on a “real” complex circuit was only achieved as late as 2003.
- One possible limitation of this type of approach is if the targeted processor has an architecture that is such that a significant number of components are not easily accessible by processor instructions.

4. Discussion

Design for test tool vendors are providing multiple solutions for reducing the increasing test costs substantially. Multiple solution options always pose tough choices for the design and test teams. Today there are two major options to choose from when confronted with test cost reduction problem. One is to incorporate test generation and response-capture circuitry on the chip and the other is generation of extremely compact test patterns.

4.1 IEEE standards and DFT techniques

JTAG:

The simplicity of the TAP interface as well as its extensibility has led to the great proliferation of the IEEE 1149.1 std. At the birth of the JTAG TAP it was primarily intended to provide access to the boundary scan features of the targeted circuit. Since then, additional features have been added so it can be used as a more general purpose test access interface.

Despite its popularity and capabilities to perform boundary scan testing, the IEEE 1149.1 std faces some problems with providing a more general purpose test interface. It is ill suited for testing of embedded cores, or testing procedures in which large volumes of test vectors and results are exchanged. The reason is that the standard uses a serial protocol that is used both for test data exchange and for providing test control signals to the TAP controller. This puts a limit on the volume of test data exchange.

The FSM also causes problems with testing of systems with several TAP controllers, e.g. systems with embedded cores. The FSM moves through its states in response to TMS. Hence the registers are only serially accessible through TDI and TDO at predefined states. So in a system with several TAPs (hierarchical systems) it is highly inefficient to use one TAP controller to provide the TMS signal for another TAP controller. Inefficient because new output signal transitions require state transitions in the controlling TAPs FSM.

CTAG:

The current CTAG proposal suffers some problems with hierarchical control of embedded cores (for which it is intended). These problems stem from the fact that the standard only supports serial connections of all SILs inside the DUT, and leaves it up to the tester to find a way of gaining access to the individual SILs. In [19] an addition to the standard, called the Hierarchical SIL (H-SIL), has been suggested. This addition would grant hierarchical access to the individual test controllers. This is yet to be adopted into the standard.

CTAG relies heavily on the existence of pretested cores and prevalidated test sets. As such, a design with little inherent DFT really cannot benefit from the standard. Because of the complexity of implementing CTAG, or any methodology that requires test objects to be embedded into the design, it seems ill suited for this project.

DFT integration:

The advantage of any scan technique, such as Scan-path, Boundary scan and BIST (2.5-2.10) is that access is granted to internal nodes, to which otherwise test data would have to be propagated from the primary inputs. With a scan chain, test data can simply be shifted serially into and out from any node included in the chain. Thus it also eliminates the need to propagate test results from internal nodes to primary outputs. All that is required is to shift the result out into the scan chain on one clock cycle, and then shift the result out through the test data output. Thus controllability and observability are significantly improved.

The drawbacks implicated by insertion of DFT into the circuit are costs in performance losses (in terms of area, power and timing) and design for test effort. Test effort can be minimized by using an electronic design automation tool that takes care of the DFT implementation and test pattern generation. Today's EDA tools implement advanced ATPG algorithms that are capable of covering large sequential circuits. But still, when compared to a deterministic self-test approach the fault coverage achieved using insertion of scan DFT and ATPG is low. Implementation of scan DFT and ATPG on a functionally complete netlist also comes with some problematic implications: Scan design rule violations require design modifications, synthesis design rule or constraint violations require chip-level re-optimization and ATPG problems carry similar consequences as scan DRC rule violations. The traditional scan DFT insertion flow is outlined in figure 21. [20]

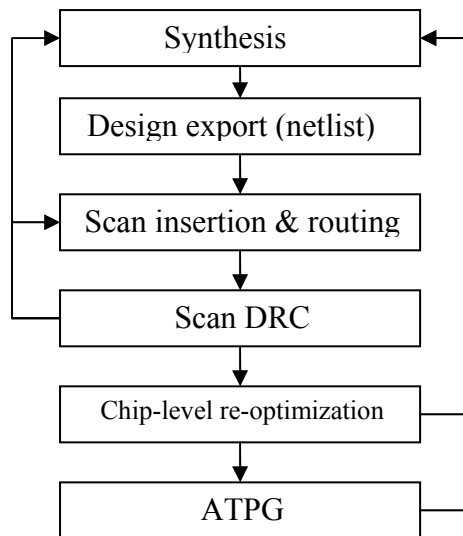


Figure 21 – Scan DFT insertion flow.

Augmenting a circuit with scan chains and then using ATPG to feed the circuit externally with enough test patterns to exercise each component is really not a viable approach on a complex design.

Embedding test generation into the design in the form of BIST does not seem to be the most pertinent solution either. Even though at-speed testing is achieved, design rule violations require painful design modification. BIST requires the design to be cleaned up from any signal integrity problems, and is in general an effort demanding implementation task. Also, experimental results have shown that poor fault coverage is achieved on e.g. a DSP.

4.2 Self-testing methodologies

Normally, self-testing using ATPG is performed by loading a gate-level description of the DUT into an ATPG tool. This methodology has two major drawbacks: Firstly, it requires a gate level description of the circuit. Secondly, gate level ATPG tools do not have the ability to take the intricacies of a complex sequential circuit into consideration, thus rendering poor fault coverage for e.g. a DSP. For example, when trying to propagate an error to observable outputs of a DSP core, the **output** instruction is very useful. An ATPG tool has no way of knowing this since it relies completely on the efficiency of randomness.

Unless exhaustive testing is used, and such testing is not realistic, test pattern generation is a complex problem. Pseudo random test patterns render smaller, but still large, test sets with fault coverage tradeoffs. Deterministic test patterns are fault oriented and effectively decrease test set size, but finding good patterns is demanding.

As mentioned, the difficulty of testing a circuit increases with size. So if a circuit is partitioned into subcircuits for testing, and these subcircuits can be tested individually, the test effort becomes less demanding. This can be done e.g. by supplying them with individual TAMs or by isolating them under test generation (as in 2.11 and 3).

A methodology based on existing functionality requires minimal insertion of intrusive extra hardware and thus minimizes the performance losses implicated by the test. Furthermore, such an approach to testing also alleviates any need for painful modifications to meet scan design rule check.

The DUT for which this thesis is intended is an SoC based DSP. Given that traditional ATPG and BIST schemes suffer difficulties with efficiently testing such circuits, a software-based self-testing approach, such as the one examined in chapter 3, seems to be the most interesting approach.

5. Conclusions

Testing has always been the tedious backside of IC-design. With the extensive embrace of the SoC paradigm, the challenges of testing have become even more of a hassle. Large SoC designs, comprising of different kinds of embedded cores, prevent old school test approaches from being sufficient. The increasing gate count is a factor in the increasing difficulties of testing. The predominant issue however, is the variety of functional blocks and the necessity of different test approaches for different blocks within a design.

At present time, there are no standard methodologies dedicated for testing of these complex systems, and the standards that are advocated for the future will not create a perfect fit for any given design. This is an innate quality of any standard, as standards are formed to be compatible with a broad range of designs.

The traditional ATPG approach, starting with a chip netlist, inserting scan-chains, and generating vectors is the most direct and effortless approach. The test effort and augmentations/alterations required on the circuit are surmountable. Very little knowledge of the DUT is required. Today's EDA tools are capable of deducing (from a netlist) how to partition a design into blocks, and to isolate them by scan-chains. Even so, without further enhancement of the ATPG algorithms, this kind of approach renders insufficient fault coverage. Also, ATE systems applying test patterns externally have inherent problems in meeting the high frequency demands of today's systems.

Many promote logic BIST as the best method for complex systems. BIST, once implemented, provides a simple and effective way to test blocks individually. However, implementation of BIST on a complex system, containing embedded cores, is time consuming and complicated compared to traditional ATPG testing. BIST implementation requires the design to be completely clean in terms of timing and signal integrity. This in order to prevent any unknown states from propagating into the signature analyzers.

Ideally, a testing methodology should impose as few restrictions as possible on the design. This would implicate inserting as little DFT logic as possible in order to minimize its impact. One way of achieving this with reasonable test design effort and efficiency is through a software-based self-test methodology as the one discussed in chapter 3. Such a deterministic block-by-block test approach demands more intricate knowledge of the circuit, and as such, it also requires a bit of test effort. But when that knowledge is available, it has been shown to render very good fault coverage. Minimal DFT insertion also has another advantage. In the world of physics there is a saying that goes "what we observe is not nature itself but nature exposed to our method of questioning" (Werner Heisenberg). This is also true for hardware testing. Insertion of DFT causes delays and thus timing problems, as well as problems with signal integrity.

The future will decide how testing of large, complex sequential circuits is best approached. Many advocate completely new standards, with dedicated test interfaces like IEEE P1500, while others recommend making existing methodologies more proficient. In that sense, perhaps the future of testing is an evolution, rather than a revolution, of present methodologies. Others still, advocate minimizing circuit augmentations and prefer testing through software-based self-tests. So perhaps less is more, and the proposed standard

5. Conclusions

methodologies for testing of complex SoCs, with insertion of embedded test objects, will not have a great impact on the testing community. Either way, it seems presently there is no suitable standard methodology for testing of SoCs.

Bibliography

- [1] - Samiha Mourad, Yervant Zorian, *Principles of Testing Electronic Systems*, Wiley-Interscience, New York, 2000, ISBN 0-471-31931-7.
- [2] – Hassan K.Reghbati, *VLSI Testing & Validation Techniques*, Simon Fraser University, Burnaby, British Columbia, Canada, 1985, ISBN 0 444 87947 1.
- [3] – Skahill, Kevin, *VHDL for programmable logic*, Menlo Park : Addison-Wesley Publishing Company, Inc., 1996, ISBN 0-201-89573-0.
- [4] - William D. Brown, *Advanced Electronic Packaging*, University of Arkansas, Institute of Electrical and Electronics Engineers, New York. 1999, IEEE Press Marketing, ISBN 0-7803-4700-5.
- [5] - IEEE 1149.1 (JTAG), *Boundary-Scan Testing in Altera Devices*, ver. 4.05 Application Note 39, Altera corporation, San Jose, September, 2000.
- [6] - Igor Mohor, *Boundary Scan Implementation*, (www.opencores.org), August 6, 2000.
- [7] - *VICTORY Technical overview*, Teradyne Inc. Assembly Test Division, 2625 Shadelands Dr., Walnut Creek, CA 94598-2597, 2000.
- [8] - P. Prinetto, M. Rebaudengo, M. Sonza Reorda, *An Automatic Test Pattern Generator for Large Sequential Circuits based on Genetic Algorithms*, Politecnico di Torino, Dipartimento di Automatica e Informatica Torino, Italy, 1994.
- [9] - B. Koenemann, *LFSR-Coded Test Patterns for Scan Design*, Proc. of ETC, April 1991.
- [10] - Erik H. Volkerink, Subhasish Mitra, *Efficient Seed Utilization for Reseeding based Compression*, Center for Reliable Computing (CRC) Stanford University, Stanford, CA, 2003.
- [11] - R. Kapur, R. Chandramouli, T. W. Williams, *New approach moves logic BIST into mainstream*, EEdesign.com, October 14, 2002.
- [12] - Ravi Apte, *The case for logic BIST*, EEdesign.com, Februari 14, 2003.
- [13] – N. Kranitis, G. Xenoulis, D.Gizopoulos, A.Paschalis, Y.Zorian, *Low-Cost Software-Based Self-Testing of RISC Processor Cores*, Department of Informatics & Telecom. University of Athens, Greece, 2003.
- [14] - Erik Jan Marinissen, Philips Research, Yervant Zorian, LogicVision, *Challenges in Testing Core-Based System Ics*, IEEE Communications Magazine, June 1999.
- [15] - Yervant Zorian, *Embedded test complicates SoC realm*, Chief Technology Adviser, LogicVision Inc., San Jose, Calif. EE Times December 22, 2000.

Bibliography

[16] - IEEE *P1500 Standard for Embedded Core Test (SECT)*
<http://grouper.ieee.org/groups/1500/>

[17] – Hani Rizk, Chris Papachristou, Francis Wolff, *Designing Self Test Programs for Embedded DSP Cores*, Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, Ohio, 2004.

[18] – F. Corno, P. Prinetto, M. Sonza, *Testability Analysis and ATPG on Behavioral RT-Level VHDL*, IEEE International Test Conference, Washington DC, 1997.

[19] - Bulent I. Dervisoglu, *Designing Hierarchical Test Access Controllers for Embedded Cores using IEEE P1500 and VSIA Compliant Architectures*, Cadence Design Systems 2655 Seely Avenue, San Jose, CA 95134, 2001.

[20] - TetraMAX ATPG, *High-Performance Automatic Test Pattern Generator Methodology Backgrounder*, Synopsys, Inc, May 1999.

APPENDIX A, Glossary

ATE	- Automatic test equipment.
ATPG	- Automatic test pattern generation.
BIST	- Built-in self-test.
BSR	- Boundary Scan Register.
Controllability	- Measurement of effort required to control an internal node.
DFT	- Design for testability, a collective term for design efforts meant to facilitate testing.
DUT	- Device under test.
FSM	- Finite state machine.
Fault coverage	- Measurement of test efficiency.
Heuristic	- Algorithm that speeds up time consuming processes.
IEEE	- Organization that establishes standards.
LFSR	- Linear feedback shift register, used for test pattern generation.
MISR	- Multiple input signature register, used for data compaction.
Observability	- Measurement of effort required to observe an internal node.
Overhaul	- Extra area required by test implementation.
PRGP	- Pseudo random pattern generator, e.g. LFSR.
Pseudo random	- Random but in such a way that the same number is not reproduced.
Rpr faults	- Random pattern resistant faults.
Seed	- Starting pattern that is supplied to a PRPG.
SIL	- Serial interface layer.
TAM	- Test access mechanism.
TAP	- Test access port, the JTAG standard control unit.
TMS	- Test mode select, the JTAG standard control signal.
Test-fixture	- Or test set, the set of test patterns that constitute the test stimuli.
WIP	- Wrapper interface port, the CTAG standard control unit.
Wrapper	- The interface between an embedded core and its chip environment.

På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: <http://www.ep.liu.se/>