NTNU

Norwegian University of
Science and Technology

Delay-Fault BIST in Low-Power CMOS Devices

Tor Erik Leistad

Master of Science in Electronics

Submission date:	June 2008
Supervisor:	Einar Johan Aas, IET
Co-supervisor:	Frode Pedersen, Atmel Norway AS

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

Current test methodologies for CMOS devices assume that manufacturing defects in CMOS devices, such as stuck-at faults or delay faults, are invariant of operating voltage and temperature. Current low-power devices may operate at very low voltages, close to the threshold voltage, and need to operate across a wide temperature range. Because of this, delay faults are now becoming more dominant, and at-speed testing is necessary. Current designs use scan architecture for fault testing, but testing equipment is often slow and the time to scan in/out test-vectors/responses contribute to a large proportion of the test time. A study is proposed to investigate how delay fault test time can be reduced through a built-in selftest of a low-power device. The suggested solution is then to be implemented in a test chip design, which can be used for fault coverage measurements and fault model verification.

Assignment given: 15. January 2008
Supervisor: Einar Johan Aas, IET

Abstract

Devices such as microcontrollers are often required to operate across a wide range of voltage and temperature. Delay variation in different temperature and voltage corners can be large, and for deep submicron geometries delay faults are more likely than for larger geometries. This has made delay fault testing necessary. Scan testing is widely used as a method for testing, but it is slow due to time spent on shifting test vectors and responses, and it also needs modification to support delay testing.

This assignment is divided into three parts. The first part investigates some of the effects in deep submicron technologies, then it looks at different fault models, and at last different techniques for delay testing and BIST approaches are investigated. The second part suggests a design for a test chip, including a circuit under test (CUT) and BIST logic. The final part investigates how the selected BIST logic can be used to reduce test time and what considerations needs to be made to get a optimal solution.

The suggested design is a co-processor with SPI slave interface. Since scan based testing is commonly used today, STUMPS was selected as the BIST solution to use. Assuming that scan already is used, STUMPS will have little impact on the performance of the CUT since it is based on scan testing. During analysis it was found that several aspects of the CUT design affects the maximum obtainable delay fault coverage. It was also found that careful design of the BIST logic is necessary to get the best fault coverage and a solution that will reduce the overall cost.

The results shows that a large amount of time can be saved during test by using BIST, but since the area of the circuit increases due to the BIST logic it necessarily don't mean that one will reduce cost on the overall design. Whether or not a BIST solution will result in reduced cost will depend on the complexity of the circuit that is tested, how well the BIST logic fits this circuit, how many internal scan chains can be used, and how fast scan vectors can be applied under BIST. In this case it looks like the BIST logic is not well suited to detect the random hard to detect faults. This results in a large amount of top up patterns. This combined with the large area of the BIST logic makes it unlikely that BIST will reduce cost of this design.

Preface

This Master's thesis is the result of work done at Atmel Norway AS during the first half of 2008. The work started as a continuation of a previous project assignment done for Atmel, and the aim was to make a circuit design that could be used for fault model analysis. To make things more interesting the design of a BIST solution was added to the assignment, with the aim to use the BIST for delay fault detection and to reduce test time. The design of the circuit and delay fault BIST would be the main part of the assignment.

The assignment was made possible with the help from Frode Pedersen, who also was my project supervisor at Atmel. I was very exited to do this assignment, not only because of its goals, but also due to the fact that it would give me great experience in circuit design and experience in working with different tools used for circuit design.

Although my experience in advance circuit design was somewhat limited in the beginning, I feel that this work has given me a lot of experience that I would not have been able to get in any other way. This became very clear in the last few weeks of the assignment, where I felt that I could have done several things different regarding the circuit design. A lot of this was probably due to the fact that I now was more familiar with the tools and how they could be used.

I wish to thank Professor Einar J. Aas, from the department of electronics and telecommunications at NTNU, for acting as my mentor and supervisor during this project. I also wish to thank him for all discussions and lectures the past few years, as they have been a great inspiration for working with circuit testing. I also wish to thank Frode Pedersen for making this assignment possible and for his inputs. My gratitude also goes to Kai Kristian Amundsen and Are Årset for their help with software tools used during the assignment.

Contents

List of Figures

List of Tables

List of Abbreviation

Abbreviation	Meaning
ALU	Arithmetic Logic Unit
ATE	Automated Test Equipment
ATPG	Automatic Test Pattern Generation
BIST	Built In Self Test
CA	Cellular Automata
CUT	Circuit Under Test
CLK	Clock
FC	Fault Coverage
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GCD	Greatest Common Divisor
HDL	Hardware Description Language
LFSR	Linear Feedback Shift Register
LOC	Launch Of Capture
LOS	Launch Of Shift
MAC	Multiply Accumulate
MISR	Multiple Input Shift Register
ORA	Output Response Analyser
PI	Primary Input
PO	Primary Output
PS	Phase Shifter
RTL	Register Transfer Level
SCLK	SPI Clock
SDI	Serial Data Input
SDO	Serial Data Output
SPI	Serial Peripheral Interface
SS_N	Slave Select (Negative active state)
STUMPS	Scan Test Using MISR and Parallel Shift
TAP	Test Access Port
TPG	Test Pattern Generator

1 Introduction

Testing of integrated circuits is a challenging task, as any NP-complete problem is. Today's integrated circuits may contain billions of transistors, several miles of narrow metal interconnections and billions of metal vias and connections. Testing for defects are necessary to obtain good product quality, and with today's integrated circuit with geometries in the nanometre range this is becoming a challenging task. For older technologies it was sufficient to use stuck-at testing and a few functional test vectors for delay testing. In newer technologies delay faults are of more concern, and for this reason functional test patterns for delay testing is no longer an option. For devices that are required to operate across a large range of voltage and temperature delay testing is very important as these parameters will affect the amount of delay.

New test approaches are necessary for delay fault testing, as functional test patterns no longer are adequate. One of the reasons for functional test patterns to be inadequate is that they are hard to develop and will not give sufficient fault coverage. Delay faults have to be tested at a structural level to achieve sufficient fault coverage. This work looks on how delay fault test can be done, and how BIST can be used to reduce the time required for test. A test chip design is developed to investigate different aspects of BIST.

Chapter 2 takes a brief look at low-level CMOS circuitry and how delay is affected by external parameters. It then looks at scan-based testing, as this is a commonly used test approach in the industry. Then it looks into BIST and how it can be used to reduce test cost. Chapter 3 describes the design of the test chip, including both the test logic and the circuit under test. Chapter 4 describes how the chip design was developed and verified. Chapter 5 explains the test approach used to gather simulation results and the test setup used. Chapter 6 presents the results as a set of graphs. Chapter 7 discusses the circuit design based on observations and the results presented in chapter 6. Finally chapter 8 concludes the report with a short summary of the discussion and suggestion for further work.

2 Theory

Production testing is necessary to eliminate defective circuits from reaching the customer. Each device produced must be tested, and when millions of circuits are produced each day the time required to test the circuits becomes important. The time used to test a circuit contributes to the circuits cost, and as circuits get more complex and geometries becomes smaller are in fact becoming the most dominant cost of the circuit. For this reason, design for test are becoming more important as a means to reduce test time and cost.

With new technologies, in deep sub micron geometries, the types of defects that are most likely to occur are different from those in lager geometries. Delay fault are becoming more dominant, and this requires other testing approaches than stuck-at testing. Another problem with deep sub micron technologies is that external parameters might affect delay in other ways than in lager geometries. This means that testing might have to be done under other conditions, and with other patterns to get the best fault coverage.

The following sections will take a closer look at the different aspects of submicron CMOS, fault models, testing, and how BIST can be used to reduce test cost. A large portion of the theory is taken from [Stroud, 2002], [Crouch, 1999], [Segura] and [Johns and Martin, 1997].

2.1 Effects of deep-sub micron technologies

The newest technology today operates on 45nm geometries. This is very deep-sub micron, and circuits made in these geometries are subject to other effects than in larger geometries (greater than 180nm).

There are two important external parameters that affect the performance of circuit after it has been made, the supply voltage and temperature. Supply voltage is probably the most important as it directly affects the speed of the circuit. This can be shown with a simple equation.

$$t = \frac{C * V}{I} \tag{1}$$

Here t is the delay in seconds, C is the load capacitance in farad, which consists of gate and trace capacitances and other parasitic capacitances connected to the driver. V is the supply voltage and I is the current delivered by the driving transistor. Replacing I by an equation for the current, [Leistad, 2007], one gets an equation for the delay that contains both voltage and temperature.

$$t = \frac{C * V_{DD}}{\mu_{n0}(\frac{T}{300})^{\beta} * \frac{C_{ox}}{2} * \frac{W}{L} * (V_{DD} - (V_{th0} - 0.002T))^{\alpha}} \tag{2}$$

Here V_{DD} is the supply voltage, α is a constant somewhere between 1 and 2, β is somewhere between -1 and -2 , V_{th0} is the threshold voltage of the driving transistor at 300 Kelvin and μ_{n0} is the mobility at 300 Kelvin. T is the absolute temperature in Kelvin, and W and L is the width and length of the driving transistor.

From this equation one can see that the delay is affected by several parameters. Transistor size, mobility, threshold voltage, supply voltage and temperature all affect the performance. In addition the resistance in the interconnect will also affect the delay, but this model does not account for this effect. One model that looks at interconnect resistance is the Elmore delay model, which considers the RC delay in different nets. Considering the routing delay is important as it can contribute to a large portion of the total delay in a circuit.

The equation for the transistor current can be used to show how delay will vary with supply voltage and temperature when not considering routing delay. Figure 1 shows a 3D graph with voltage and temperature as parameters.

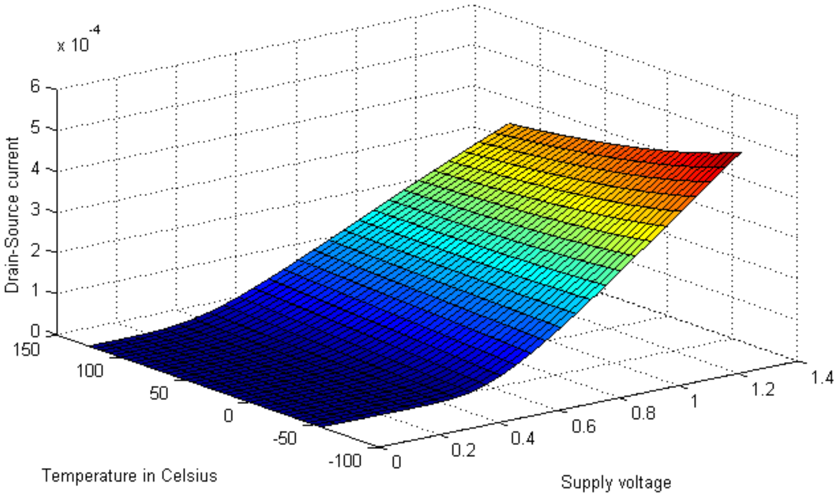

Figure 1: Vdd, temperature versus Id

From this figure one can clearly see that there is a change as the supply voltage decreases. For supply voltages greater than the threshold voltage, 0.7V, the drive strength decrease with increasing temperature, which will result in a larger delay. When the supply voltage decreases and becomes close to or less than the threshold voltage this changes. Now the drive strength increases with increasing temperature and this will give lower delay. The reason for this is that both the threshold voltage and the carrier mobility are affected by temperature. They both decrease with temperature but works in opposite directions. For large supply voltages the current is mostly affected by the carrier mobility. The variation in threshold voltage is small compared to the variation in mobility, and mobility variations dominate the equation. For small supply voltages the opposite happens.

Taking into account that today's circuit are required to operate across a large voltage and temperature range one can see that delay testing is important, but also difficult. The characteristics of a circuit may vary greatly with voltage and temperature and this may result in a circuit that operating correctly under one condition but not under another. Selecting the right voltage and temperature for testing is important, and to get sufficient fault coverage several settings might be necessary depending on the technology used.

2.2 Fault models

Several types of defects can occur in a circuit. These defects are mapped to a set of fault models that describes how a defect will affect the circuit. These fault models are then used in fault simulations of the circuit to determine which fault coverage can be obtain. Fault coverage is a measure of how good the production test is, and if this value is to give a reliable result it is important that there is a good relationship between the defects and the faults model. For this reason there are several commonly used fault models. The next sections will take a brief look at the most common models, [Stroud, 2002], [Crouch, 1999].

2.2.1 Gate stuck-at 1/0 faults

The gate stuck-at model assumes that one of the inputs or outputs of a gate are tied to logic one or zero. This means that a two input gate can have six different modelled faults. Several defects can lead to this type of behaviour. For instance can a low resistance bride between one of the power supply rails and one of the nodes tie the node to the power supply rail, making the input or output to always have the same logic value.

In the example in figure 2 one of the nodes are tied to logic one. To detect that this node is tied to one, the opposite value has to be assigned to that node. This can be done by setting S to one and A to zero. Since S is one it doesn't matter what value is assigned to B in this case. In a correct functional circuit the output should now be zero, but since there is a stuck-at one fault, the output remains at one.

2.2.2 Transistor stuck-on/off faults

Gate level stuck-at can accurately reflect the behaviour of transistor faults in NMOS circuitry, but the advantage of CMOS brought with it the need for transistor fault models. Transistors can be stuck-on or stuck-off due to different defects such as bridges and opens. If a transistor is stuck-on there might be a conducting path between the power supply rails for some

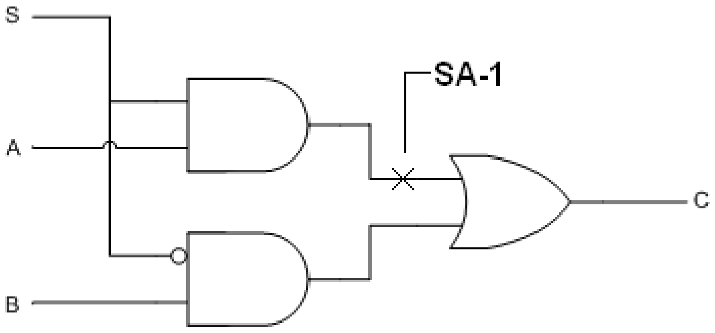

Figure 2: Stuck-at one fault.

input vectors. For gates with such faults the output value can be something between the two supply voltages. The input vector and the drive strength of the transistors will determine the output value of the gate.

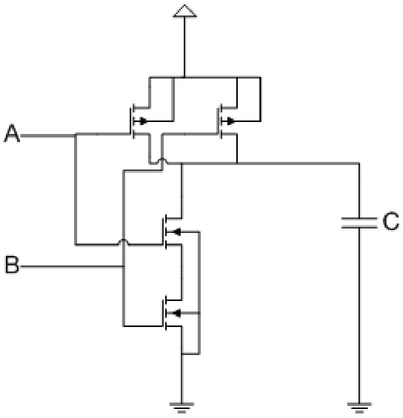

Figure 3: Transistor representation of NAND-gate.

One problem with transistor stuck-on and stuck-off fault is that they may have memory effects, meaning that several vectors are needed to detect them. Looking at figure 3 and assuming that one of the NMOS transistors are always off, one can notice that one first has to charge the load capacitance to logic one and then apply the vector for discharging it through the NMOS transistors. If the output value stays at one it is clear that one of the NMOS transistors are always off and the defects is detected.

2.2.3 Delay faults

Delay fault are different from other faults. This is because they are only detected if the test is done at-speed. Some defects, such as bridges and opens, may have characteristics that make the circuit function correctly at low speeds, but fail at high speeds. The object of delay fault testing is to test the paths between flip-flops, between primary inputs and flip-flops, and flip-flops and primary outputs. Delay fault testing requires two vectors, the first sets up the path and the second tests the path by producing a transition in the path for delay fault detection. The second vector also needs to be captured at-speed. There are two types of delay faults, transition delay and path delay fault, [Gjermundnes, 2006].

Transition delay faults, also known as slow-to-rise and slow-to-fall faults, are the type of defect that causes a single gate or transistor to be significantly slower that normal. This is also referred to as spot defects. Every gate has the possibility of having delay defects. Testing all possible faults gives long testing time, as two vectors are needed to test each possible fault location.

A path delay fault is what happens if the overall circuitry is too slow. Reasons for such faults are variation in the manufacturing process that affects all or several transistors in the circuit, such as over etching or insufficient doping. The result is that some paths through the circuitry may not meet the required performance specifications. To detect this type of fault one can apply delay tests to the longest paths through a circuit to see if they can operate at the given speed.

2.2.4 Bridge faults

Bridge faults occur when there is a short or resistive coupling between two or more nodes that should not be connected. There are two commonly used models for bridge faults, wired-AND and wired-OR. A wired-AND fault is also said to be zero dominant and a wired-OR is said to be one dominant. Bridge fault are complex faults. The effect of the bridge does not only depend on the bride resistance, but also one what gates are driving the different sides of the bridge and what vector is applied to these gates. Bridge faults can only be detected if opposite values are applied to the two different sides of the bridge. Since there is opposite values on the two sides there will be a conducting path between the power supply rails and in some cases the increase in current consumption can be detected.

One of the biggest problems with bridges is that one does not know which interconnections are likely to make bridges before the layout of the chip is done. To determine which wires are most likely to make a bridge one can extract the capacitive coupling between the different wires as it is likely that the wires with largest capacitive coupling is the wires that will make a bridge.

2.2.5 Open faults

An open fault is a break in the interconnection between to places. An open defect can be conducting or not depending on the size of the open. In the cases where it is conducting it can be modelled as a series resistance, and in cases where it doesn't conduct it can be modelled as an infinite series resistance. Delay testing is required to detect open faults, as the series resistance will increase the delay of the path by an unknown amount.

2.3 Scan testing

Testing of a circuit can be done in several ways. Functional testing uses the circuit without any additional logic to test it. This is not a very practical approach as a circuit can have several inputs, and if it is sequential a large amount of vectors are required to sufficiently test it. The most common approach to solve this problem is to use scan testing. If the design is a full scan design it is reduced to a combinational circuit as all sequential stages are removed. A partial scan design will have some sequential stages, but the number of them is reduced. When the circuit is reduced to a combinational circuit it reduces the number of test vectors needed, which in turn also reduces test time. It also reduces the time needed for generation of test vectors as the ATPG now only needs to consider a combinational circuit instead of a sequential one.

The reduction to a combinational circuit is obtained by connecting all registers in the circuit into a long shift register. This is done by adding a multiplexer in front of the input to each flip-flop. The multiplexers are controlled by a scan enable signal. When the scan enable signal is active the registers are connected into a shift register and the input and output are used to shift in and read out data from the circuit. When the scan enable isn't active the circuit is connected to its normal operational mode. The multiplexer adds additional area and delay, but this can be minimized by using and optimised scan cell, as seen in figure 4.

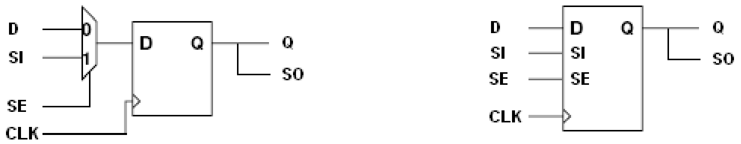

Figure 4: A muxed scan flip-flop.

To test the circuit one first sets it into scan mode by setting scan enable active. Then the scan chain is filled with known data so that the state of the circuit is known. The scan enable is then set to its inactive state, and one clock cycle is applied. The circuit is brought back to scan mode and the data is shifted out at the same time as new data is shifted in. The output from the scan chain is compared to the expected values, and if the output is different, one knows that there is a defect in the circuit. This procedure is repeated until all faults are covered.

The major problem with this approach is the time needed to shift the test vectors and the responses. One approach to reduce shifting time is to use several scan chains, but one is often limited by the amount of pins available for this purpose. The second problem is that shifting often has to be done at a slower speed than the maximum frequency of the circuit, as the testing equipment is slower than the circuit.

2.4 Delay testing

Delay testing is becoming more important as defects in smaller geometries are more likely to result in excessive delay. Delay testing can only be achieved by running the circuit at the maximum operational speed during test. This is not always an option. Testing equipment may be slow, and not able to run at these speeds. This leaves just two options; run the circuit at slower speed, but change to the maximum speed during the launch and capture of test data, or use some BIST logic that works independently from the external testing equipment.

Since scan testing is a widely used test method, modifying it to support delay testing would be a big advantage. This can be done in two ways, referred to as launch of shift (LOS) and launch of capture (LOC), [Xijiiang Lin; Press, Sept.-Oct. 2003], [Saxena, 2002], [Pateras, Sept.-Oct. 2003], [Vemula]. There is a big difference in these two methods. Launch of shift requires the scan enable signal to be an at-speed signal, that is, it needs to be able to operate at the maximum frequency of the circuit. This means that the scan enable signal has to be routed as a clock signal and this will result in additional area overhead. Both method has a launch and capture cycle. First the circuit is brought to a known state, and then the launch cycle applies a transition to the circuit which is captured in the capture cycle.

In launch of shift the last shift is used as the launch cycle, and the scan enable signal need to be inactive before the capture cycle. This is the reason for the need for an at-speed scan enable in this approach. The timing for a LOS cycle is shown in figure 5.

Instead of making the scan enable signal as an at-speed signal, other approaches can be used. The scan enable signal can be pipelined into several pipelines or local scan enable generators can be made, as described in [Ahmed, 1-5 May 2005]. These can then connect to a small amount of flip-flops, making the signals able to run at-speed.

The second approach, LOC, gives the ability to use a slow scan enable signal by inserting a delay before the launch cycle, as seen in figure 6.

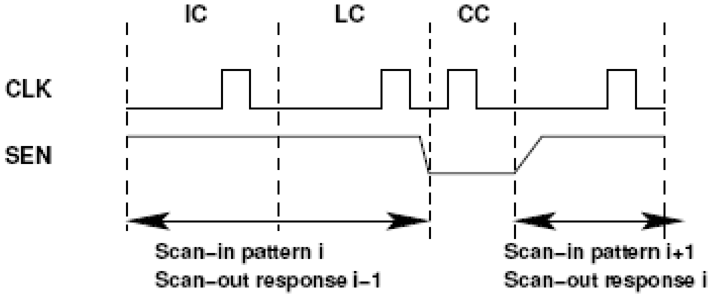

Figure 5: Timing diagram for LOS.

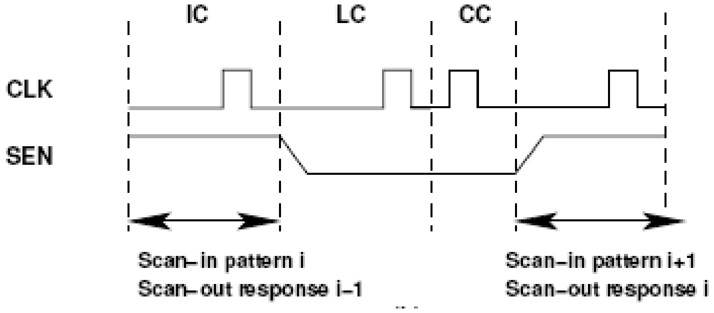

Figure 6: Timing diagram for LOC.

Both methods have some limitations. Delay testing requires two test vectors, one to set up the initial condition, and one to make some transition in the circuit. In LOS the second vector is just the first vector shifted by one bit. The means that the two vectors are very alike, and this may in some cases limit the fault coverage. This can easily be seen in an example with a tree of AND-gates as in at figure 7. Assuming that you want to test for a slow-to-rise at a gate input along the longest path in the figure. The initial value has to be zero and a transition to one must be applied. If LOS is used the initial zero will just be shifted one bit to the right, and the output of the AND-tree will still be zero. This means that it is impossible it test for a slow-to-rise along some paths, and this will reduce the fault coverage.

Launch from capture also has some limitations. The second vector is the response due to the first vector. This means that not all possible combinations of inputs can be applied during the launch, but all functional ones can be. Sequential ATPG will also be necessary for LOC since the second vector is the response of the first one. Since not all possible patterns can be applied there will be a reduction in fault coverage also in this case.

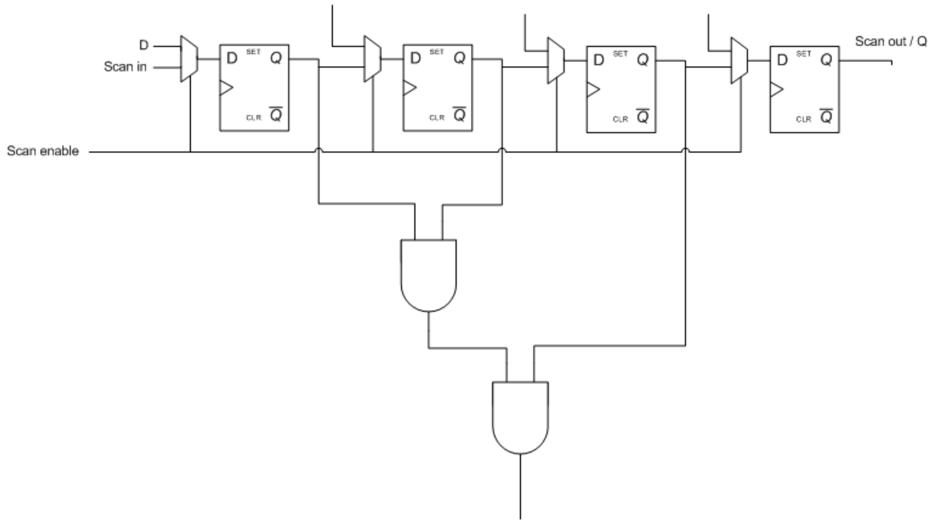

Figure 7: AND-tree attached to scan chain.

2.5 Fault coverage

Fault coverage is a measurement of how effective a set of test vectors are in detecting possible defects in a circuit. A fault simulator takes a circuit description and test vectors as input. It generates a fault list for the circuit, or takes it as input, and simulates the test vectors against the fault list. After simulation is completed the simulator will have a list with the detected faults, possibly detectable faults, undetectable faults, and undetected faults that remains in the fault list. Using these values, the fault coverage can be calculated as described in [Stroud, 2002]:

$$FC = \frac{D + xP}{T - U} \tag{3}$$

Here D is the number of detected fault, P is the number of potentially detected faults and x is the probability of it being detected. T is the total number of faults, and U is the number of undetectable faults.

2.6 BIST

Build In Self Test (BIST) is a means for reducing the cost of testing. This is accomplished by adding additional logic to the circuit to be tested to ease the testing procedure. There are several approaches to BIST, but they all have certain things in common, such as test pattern generators (TPG) and output response analysers (ORA), [Stroud, 2002].

The main problem for all BIST approaches is the total number of flip-flops in the circuit. The number of flip-flops can easily range in the number of hundreds of thousands, requiring large TPGs, resulting in that it is not possible to run the BIST logic for maximum length. For long test sequences higher fault coverage is obtained, but the cost of test also increases.

The goal is to test the circuit for all possible faults. ATPG will make deterministic test vectors, which is possible since the ATPG tool knows how the circuit is constructed. When BIST is used, another approach has to be used. The BIST logic has to be made so that it effectively can apply a high number of test vectors to get sufficiently high fault coverage. The straight-ahead approach is to exhaustively test the circuit. This means that all possible combinations of ones and zeros are used at the inputs to combinational logic. The number of possible combinations in today's circuits is astronomical, and it is not an option to do this. The solution is to try to make the BIST logic make pseudo exhaustive test vectors. This means that all logic cones are exhaustively tested. The result is that the entire chip is exhaustively tested, and 100% fault coverage is guarantied to stuck-at and bridging faults.

BIST can be divided into two main categories, intrusive and non-intrusive BIST. The main difference between these two are the way the TPG and ORA are implemented in the circuit. For a solution using intrusive BIST, already existing flip-flops are used in the generation of TPG and ORA. This means that there will be a performance penalty for the circuit as additional logic, which might affect the critical path, is added to the circuit. A non-intrusive BIST solution does not use existing flip-flops in the circuit to make TPG and ORA. The CUT is unaffected as TPG and ORA are added as external modules, which connect to the CUTs inputs and outputs.

Examples of intrusive BIST solutions are BILBO and circular BIST B. Koenemann and Zwiehoff [1979], [Stroud, 2002]. In BILBO all registers can be TPG and ORA. Since a register can't function as both TPG and ORA at the same time, BILBO configures some registers as TPG and some as ORA. Then the logic between them is tested. When the logic has been tested, the registers are reconfigures so that untested logic can be tested, by changing which registers operates as TPG and ORA. Several test cycles might be necessary as it is likely that more registers has to operate as TPG than ORA. This is because a logic cone can have inputs coming from different registers, which all has to be TPGs to test the logic cone. This leads to the new optimization problem of minimizing the number of test cycles needed by selecting which registers to be TPG and ORA in the different test cycles. A thing to notice is that BILBO works well for pipelined architectures. Another thing is that scan is also needed to retrieve the responses from the test cycles. Circular BIST also uses existing flip-flops for TPG and ORA. One thing that is different from BILBO is that the responses from the CUT here are used as new test patterns. This means the pseudo exhaustive testing isn't guarantied.

BIST is rarely used to test an entire chip. This is due to different types of logic and structures used within the chip, such as RAM and general sequential logic. Often several different BIST circuits exist in a circuit, some optimized for testing RAMs and other for testing generic logic. For a device containing several BIST circuits, one can save test time by running more than one at a time. One of the main problems is to determine which to run simultaneously, as one need to consider the total dissipated power during test.

One thing that makes BIST very useful is the fact that it makes it possible to test the circuit in the field. Usually circuits are tested in a test machine. This means that it can't be connected to any other logic. If a circuit has BIST and it is implemented with input and output isolation, the circuit can still be tested on a circuit board if the BIST logic supports start and stop by JTAG, software, or by any other method.

2.6.1 Test pattern generation

Pattern generation can be done in several ways. Counters, accumulators, MAC-units and similar can be used as pattern generators. Their performance differs, and one will be better than another in some cases. The types of patterns made are different from the deterministic patterns made by ATPG, which target specific faults. Random patterns are of little value since one needs to know the patterns to know what signature to expect. For this reason pseudo random patterns are often used when random patterns are preferred. Counters do not generate random patterns, but still they can generate all possible patterns given enough time. For large counters this is impractical as the most significant bits switches very slow compared to the least significant bits. This result in that logic connected to the most significant bits needs very long time before they are tested.

On of the most common pattern generator used is the linear feedback shift register (LFSR). It is a pseudo random pattern generator, as its output have random characteristics. A LFSR consists of a shift register with feedbacks using XOR-gates. The location of the feedback taps is determined from primitive polynomials. The feedback can be made in two ways, internal and external. Figure 8 shows a 3 bit LFSR with external feedback and figure 9 shows a 3 bit LFSR with internal feedback.

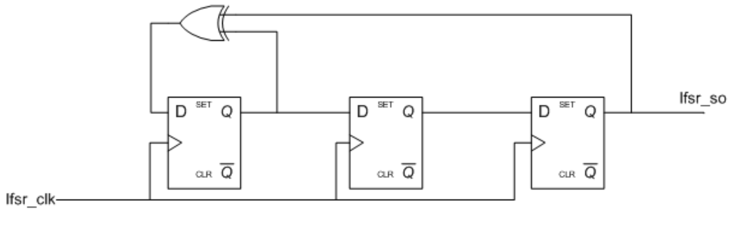

Figure 8: LFSR with external feedback.

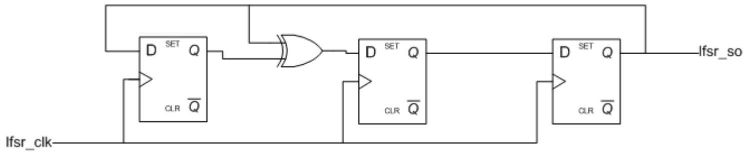

Figure 9: LFSR with internal feedback.

The external LFSR is just a shift register with feedback into the first register. If the LFSR is used to feed parallel scan chains, with one scan chain connected to each register output, the data in the different scan chains will be highly correlated as only one bit differ between them. The internal feedback performs better in this case as the internal XOR-gates help in randomizing the data between the scan chains. For large LFSR with several feedbacks the external feedback will also result in a LFSR with longer critical path, making it slower than the internal feedback solution. For these two reason the internal feedback register is the preferred choice when designing a LFSR.

Several other aspects also need to be considered when designing a LFSR. The most important is that it can't be initialized to the all zero state. If a LFSR is in the all zero state it will remain there, as there is no change in the feedback loop. This means that the all zero pattern can't be applied to a circuit using LFSR as TPG. To make sure that all other patterns can be applied one need to make sure that the LFSR is able to run for the maximum length. This is achieved by using a primitive polynomial to determine the construction of the feedback. The size of a LFSR is often referred to the degree of the polynomial used. Since the patterns made are determined by the initial state of the LFSR, and large LFSR makes it impractical to run it for the maximum length, it is often necessary to add additional logic to the LFSR making it possible to set the start value. This is referred to as reseeding.

As mentioned, there are some issues to be considered if a LFSR is used to feed parallel scan chains. One of them is structural dependency. If parallel output is used directly from the LFSR, parallel scan chains contains the same data just shifted by one bit. This means that not all possible vectors can be applied in an n-by-n array produced by the LFSR. Since not all possible patterns can be applied this will reduce the possible obtainable fault coverage. To reduce this problem internal feedback should be used, and using feedback with many XOR-gates is desirable.

Another problem is linear dependencies. Consider the example where a 4 bit LFSR is used to generate vectors for a 10bit scan chain. Since the all zero pattern can't be made by the LFSR, the length of the LFSR is a 15bit sequence before it starts to repeat. If exactly 10 bits are shifted into the scan chains for each test vector only 3 unique test vectors will be generated, and of course this is not enough to test all possible faults. The solution to this problem is to avoid cases where the length of the scan chains and the length of the LFSR have a common divisor other than 1.

$$1 = GCD(N_{FF}, 2^n - 1) \tag{4}$$

If GCD is a function which calculates the greatest common divisor the equality shown in equation 4 can be used as a requirement. Here N_{FF} is the number of flip-flops in the longest scan chain and n is the number of flip-flops in the LFSR. In the case of the 4bit LFSR and 10bit scan chain, one can see that they both have the common divisor 5. If one instead of shifting exactly 10bits shifts 11bit (setting N_{FF} to 11), one will see that 11 and 15 don't have any other common divisor than 1. This means that a maximum length test is possible by shifting 11 bits into the 10 bit scan chain for each test pattern that is applied.

Other considerations to make are the size of the LFSR and the characteristic polynomial. The size, n, should be greater than the span of any logic cone in the design driven by the scan chain. The span is the number of scan elements between the two inputs to a logic cone that are farthest away from each other in the scan chain. One other requirement for the size of the LFSR is that n are chosen so that equation 5 is satisfied.

$$2^n - 1 > N_{FF} * N_{TV} \tag{5}$$

There n is the size of the LFSR, N_{FF} is the number of flip-flops in the longest scan chain, and N_{TV} is the number of test vectors to be applied. Using this requirement helps in reduce linear dependencies problems.

Regarding the characteristic polynomial, it might affect the linear dependencies, as they can be a function of the polynomial and the location of the scan flip-flop driving a logic cone. For this reasons it might in some cases be a solution to have an LFSR with programmable polynomial.

It should be stated that LFSR isn't good for all circuits, and in these cases modifying the patterns or using other TPG might be necessary.

2.6.2 Output response analysers

Output response analysers are used to generate a single pass or fail indication as a result of the BIST sequence. Usually it is a multiple bit value, and it is only necessary to read this value to determine if the circuit is fault free or not. Since thousands of test vector responses are compacted into a single value there is a possibility of fault masking. Fault masking occurs when a fault masks another fault by changing the signature back to the value indicating a fault free circuit. Different ORAs have different properties and different likelihood of fault masking. In many ways ORAs are similar to

TPG. Accumulators, MAC-units, and counters can be used here with the response as input data.

If several identical circuits are tested at the same time a comparison-based analyser can be used. It only requires one XOR-gate per bit and one N-input OR/NOR gate. By comparing the outputs of the different circuits one can easily find if one of the circuits has a different output than the others. The comparison-based analyser has low area overhead, but its use is restricted. Also, aliasing will occur if both circuits connected to the same XOR-gate has the same defect. Another problem with it is that it is not guarantied that the ORA is completely tested.

A counting based output response analyser counts the number of ones or zeros that reaches the output of the circuit. Here one counter is required for each output, so the area overhead can be quite large. Aliasing can occur if the number of ones and zeros at the output are unchanged for a faulty circuit, or it the counter overflows and this is not controlled. The overflow of the counter can easily be fixed by using a large counter, or by using the overflow bit as an additional input.

Parity analysers are a form of counting analyser. It is very simple, as it is a one-bit counter and only checks whether there are an even or odd number of ones in the output. It works well for serial outputs, such as scan chain outputs, but since it is only one bit, any even number of faults will result in fault masking, and for this reason it is not very attractive as a fault analyser.

Instead of counting the number of ones and zeros one can use an accumulator based ORA. In this case all outputs are used as inputs to an accumulator. The signature is the value of the accumulator after the BIST sequence. This value can then be read by ATE and compared externally or compared to a value stored in ROM. The main problem with accumulator-based analysers is that they will overflow very fast if they are implemented as single precision accumulators (the size of the accumulator is the same as the number of outputs from the circuit). To solve this one can use double precision accumulators. In this approach the carry output from the single precision accumulator is used as input to a second adder so that no information is lost due to the carry from the first part. This approach uses more area as two adders are made. Another option is to feed the carry from the single precision accumulator to a flip-flop and then use the output from this flip-flop as carry input for the accumulator. Now the carry output information remains in the circuit, but it can still be masked by other faults.

One of the most common response analysers is the multiple input shift register, MISR. It is made in the same way as an LFSR, but in addition has one XOR-gate at each flip-flop input, which is connected to a signal that is to be compacted.

MISRs are also susceptible to fault masking. One way this can happen is when a fault from one scan chain enters the MISR, making a bit flip value.

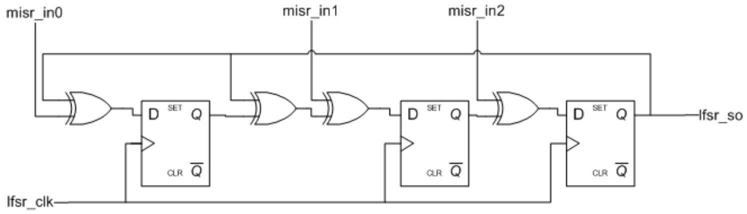

Figure 10: MISR with internal feedback.

Then in the next clock cycle another fault enters the next bit in the MISR, and as the first bit is shifted down the MISR, this bit switches back the first bit, resulting in a signature for a fault free circuit. To reduce the likelihood of fault masking one can use large MISRs, as the probably of aliasing is approximately 2^{-n}, where n is the degree of the characteristic polynomial used for the MISR. Using primitive polynomials and internal feedback also helps in reducing the probability of fault masking. One last requirement for the use of MISR as fault analyser is that the signature must be non-zero if the MISR is initialized to zero. The reason for this is that, as for LFSR, the MISR has the property that it will remain in that state if all inputs are zero.

There is one other source of fault masking. Most circuits will have a large amount of outputs, and it is not desirable to make very large MISRs or accumulators. To reduce the size of the needed response analysers, concentrator logic is added between the output and the analyser. The concentrator is made of XOR-trees. This means that any even number of faults entering the same XOR cone will result in fault masking.

There is one approach that can be used to always reduce the likelihood of fault masking. By periodically monitoring the signature to check for fault detection one can detect that a fault has occurred before it is masked, and one can also abort the testing before it is finished, and save test time.

2.6.3 Test points

One of the main problems with testing is to control and observe internal nodes in the circuit. ATPG tools can make special test vectors that set specific values to specific nodes in the circuit. This can be done to propagate the value of a specific node to one of the outputs. For BIST this is harder as the patterns are random, and for this reason the fault coverage will be lower than for ATPG. One way to improve fault coverage, both for ATPG and BIST, is to insert test points. A test point is a point that is easily controllable, observable or both. Only a few test points can increase the fault coverage of the circuit significantly.

A control point can be made in several ways. If the node only needs to be controlled to a specific value, zero or one, an AND or OR-gate can be used. If it needs to be controlled to different values during test a multiplexer can be used. The multiplexer will then select between the normal input and a test input, and it can be controlled by a test enable signal.

An observation point can be made of just additional wiring connecting the point to an additional output or an existing output via a multiplexer. Observation points will for this reason have less impact on the circuit, compared to control points.

For a BIST solution test points have an additional important task. The inputs of the circuit need to be controlled to prevent unknown data from reaching the response analyser. One way to achieve this is to add multiplexers in front of the inputs and supply data to these multiplexers from the TPG. The outputs are also often controlled to a known state, and at the same time the outputs are observed by connecting them to the inputs of the ORA.

The two main problems with test points are area overhead and additional delay. Test points ads additional area due to the additional logic, but also the additional routing contributes to the area overhead. If few test points are added, the additional area overhead might not be an issue, but the additional delay might be. If the test point is inserted in a critical path, the additional delay of a multiplexer or other gate will affect the circuit's performance, which might not be acceptable. Since observations points don't need additional logic to be inserted they are preferred over control points as the performance penalty are less for observation points.

2.6.4 Increasing the fault coverage

Due to the different problems described, it might be necessary to use large LFSR compared to the span of a logic cone. This might lead to LFSR with 20-25 bits or more, making it impossible to apply the maximum test length due to test time cost. To further increase the fault coverage one need to add some other approaches. One can be to support reseeding of the LFSR, or the support of changing the characteristic polynomial of the LFSR. Both of these can be supported by a test interface such as boundary scan.

In some cases it is not enough to use just a pattern generator and a response analyser to get sufficient fault coverage. Even if the pattern generator can make all possible patterns, the time required to do so may be excessive, and other means for applying the required patterns are necessary. There are some options to consider. The patterns can be applied from an external tester through scan or by some other method. This means that the external tester is required, and that a complete test can't be done without it. Another solution is to store the patterns internally in an ROM, but this will require additional area, and may be to costly if the required number of stored patterns is high. Another approach is to add logic that can modify the vectors generated by the pattern generator such that they will cover the hard to detect faults faster. Example of additional logic is phase-shifters, weighting logic and bit flipping logic.

2.6.5 Phase shifters

If a LFSR is used as TPG for a scan design, and several scan chains are connected to adjacent bits of the LFSR, neighbouring scan chains will have highly correlated data. The test data seen by the CUT will not be pseudo random; as the data in neighbouring scan chains have only one bit shift in difference. The worst case is for the LFSR with external feedback. A solution to this problem is to add logic that shifts data in neighbouring scan chains by more than one bit. The straight-ahead approach is to add shift registers with different length between all of the LFSR outputs and scan chains inputs. This approach result in large area overhead and is not a good choice. Instead one can construct a phase shifter from XOR-gates. A phase shifter is constructed by taking different outputs from the LFSR and XORing them together. This produces the same output sequence as the LFSR, but shifted by a number of clock cycles. An algorithm for calculation of which LFSR outputs to XOR together to generate a given number of shift delays is described in [Rajski and Tyszer, 26-30 Apr 1998].

A phase shifter has a number of interesting properties. It has low area overhead, only one additional XOR-gate per scan chain on average. This means that the LFSR and phase shifter has approximately the same area overhead as a TPG generated from CA-registers, but the LFSR and phase shifter performs better. The performance penalty is a delay of two XOR-gates between the LFSR and scan chain inputs on average. One of the most interesting properties of the phase shifter is that it allows one to generate outputs to more scan chains that there are outputs from the LFSR. This means that one can save additional area by making the LFSR smaller, but at the same time having a large amount of scan chains. The only requirement is that the size of the LFSR satisfies the equation 6.

$$2^n - 1 \geq N_{FF} * N_{SC} \tag{6}$$

Here n is the size of the LFSR, N_{FF} is the number of flop-flops in the longest scan chain, and N_{SC} is the number of scan chains.

2.6.6 Weighting logic

When using LFSR as TPG, and the output is random, there is an equal probability of a one or a zero entering the scan chain. Some circuits have random hard to detect faults, meaning that there is a very little chance for a random pattern to detect the faults. In these cases, additional logic might be added to the TPG to make it able to generate vectors that are more likely to detect these faults.

Weighting logic is used to change the distribution of ones and zeros in the test patterns. Weighting logic can be made up of AND-gates or OR-gates. Using AND-gates will increase the likelihood of zeros and using OR-gates will increase the likelihood of ones in the test vectors. To generate a set of different weights several outputs can be ANDed or ORed together. Multiplexers can then be used to select the different weights.

The area overhead of a weighting logic can be significant compared to the LFSR and phase shifter if several weights are supported. Several AND- or OR-gates are required, and large multiplexer have to be used for selection of the different weights. The fault coverage only increases with a few percents when weighting is used, and it might require a huge number of test vectors. For this reason, weighting is usually used when external test cant be used, and BIST is used for a complete onchip test [Kusko et al., 2001]

2.6.7 Bit flipping

Bit flipping logic is a more advanced method of changing the test vectors to increase the fault coverage. Individual bits in the test vectors are flipped, and a bit flipping function controls the flipping. The bit flipping function can uses the state of the BIST logic as inputs, for instance pattern counters, bit counters and any other counters that are available. The goal of bit flipping is to match deterministic test vectors to the ones made by the TPG by modifying the test vectors generated by the TPG [Wunderlich and Kiefer, 10-14 Nov 1996], [Gherman et al., May 2007].

To determine which bits should be flipped one need to do fault simulation. First the vectors from the BIST logic is fault simulated to determine which faults are undetected and which test vectors that don't detect faults. One then selects one of the undetected faults, and finds the one test vector, which previously didn't detect a fault, that most closely matches the test vectors that detect this fault. Logic is made to modify this test vector. This is then done for all remaining faults, and then the new BIST logic is simulated to control that all faults are detected.

This method have low area overhead, and gives high fault coverage. Since delay faults have lower random testability, this method might help significantly in increasing the fault coverage. The main problem with bit flipping is that it can take a long time to generate the bit flipping function, and if the CUT is changed in some way it all have to be redone as the test vectors might have changed.

2.7 Scan based BIST

Scan is one of the more common approaches to testing. ATPG are use together with ATE to performs testing on circuits. Scan based BIST moves the pattern generator inside the circuit and connects it to the scan chains. A response analyser is also connected to the scan outputs, and compacts the responses into a single value. A scan based BIST design allows several CUTs to share a common TPG and ORA. If all the CUTs have their own scan chains multiplexers can be used to connect the different scan chains to the TPG and ORA, allowing them to be tested separately. This can help in reducing the size of the ORA and TPG.

In addition to the TPG and ORA, a BIST controller is needed. The controller is used to control how many bits are shifted into the scan chains, and how many patters have been applied. A simple BIST controller can be made of an N + M bit counter, where the N least significant bits are used to count the number of bits shifted into the scan chains, an the M most significant bits are used to count the number of patterns applied. This means that the maximum length of a test using this approach is approximately 2^{N+M} clock cycles.

One of the problems with scan based BIST, and other BIST solutions, is that random patterns are less effective on detecting faults than deterministic patterns. According to [Stroud, 2002] approximately ten times more patterns are needed to obtain the same fault coverage with a scan based BIST design compared to ATPG scan. This can of course vary greatly between different designs. This means that if the same scan speed and the same number of scan chains are used, as for external scan testing, the testing time can be quite large. But since a BIST design isn't restricted by the number of input and output pins on the circuit, one can have a large amount of internal scan chains connected to a TPG, and by this reducing the number of clock cycles needs to shift new test vectors into the scan chains. The shifting speed might also be increased since the speed is no longer limited by the speed of the input and output pins, or by external testing equipment. Applying these two steps can significantly reduce testing time for a scan based BIST design, and should always be considered.

One limitation that might affect the scan speed is the total power dissipation of the circuit. Scan is one of the most power demanding operations since almost the entire logic will be switching every clock cycle. Due to this, scan based BIST might not be operated at the maximum frequency of the circuit during shifting of test vectors. If this is the case it will need a clock controller to generate the appropriate clock signals, as at-speed is necessary during the capture cycle to perform delay testing.

STUMPS, or Logic BIST, is one of the most common forms for scan based BIST. It uses an LFSR for TPG, and MISR for ORA. Additional logic such as phase shifters, bit flipping and weighting are often used. STUMPS are usually not intended as the only means of testing, so it is usually followed by test data from an external tester. The main reasons for using such approach is to reduce test cost. This is achieved by shorter test time or by reducing the amount of tester memory needed, or both. Fault coverage for LBIST can vary from 65% to 80% for large industrial designs. The large variation is due to the fact that different circuits can have different amount of hard to detect faults. Inserting additional test points in the circuit can significantly increase the fault coverage by reducing the amount of hard to detect faults.

One other scan based BIST design is the hybrid design [Das and Touba, 2000]. It combines external test patterns with internal test patterns at the same time. The idea is that merging random data with deterministic data will shorten the test time, compared to the STUMPS approach, and at the same time save tester memory. One of the problems is that the test can't run faster than the external testing equipment. This means that this approach is a good choice if tester memory is the problem, and this approach is used to reduce the amount of data stored in the testers memory.

2.8 Cost of test

The cost of testing contributes to a large portion of the total cost of manufacturing. In state of the art circuits the test cost is close to half the cost of the device. The test cost can be modelled in several ways, depending on what type of test approach is used. The two most common parameters to model test cost on are time used and memory footprint of the test vectors. One can say that these two are almost the same, as at a given test speed the amount of memory used will correspond to a given amount of time. Still there is a small difference, as the argument often is that a cheaper tester can be used if the memory footprint of the test vectors is small. For circuits using BIST additional parameters needs to be considered. The area overhead of the BIST logic will add cost to the circuit. The development cost of the BIST logic should also be considered, as this can be large in some cases. The development cost is much harder to model as this cost can be distributed over every manufactured device, but as this number is unknown, the cost of BIST development for each device is also an unknown. Since it is an unknown, the development cost is usually left out of the equation.

Looking at the STUMPS testing approach, one can easily see that the equation for the chip cost can be quite complicated as both BIST and ATPG are used. The problem with this approach is to find the optimal number of BIST and ATPG patterns. To few BIST patterns means that to many ATPG patterns are needed, but they might require a long time to be applied, which can increase test time and cost. To many BIST patterns will also increase test cost due to longer BIST testing time. In other words, one needs to find the optimal ratio between BIST and ATPG vectors. Since the number of

ATPG top up patterns only can be determined after the BIST patterns are simulated, it can be a tedious job to fine the optimal number of BIST and ATPG patterns as several simulations are required.

If one wants to find out how much can be saved in this case, one need to calculate the total cost for both cases. Starting with the pure ATPG testing scenario with a scan design, one can easily calculate the cost of the device. Knowing the area, A_{CUT}, of the circuit, area unit cost C_{Area}, the length of the longest scan chain, N_{FF}, and the number of test vectors, N_{TV}, the tester speed TS_{ATPG}, the required time for launch and capture cycles T_{LC} and the unit time cost C_{Time}, one can find a formula for the cost, C_{ATPG}, as:

$$C_{ATPG} = A_{CUT} * C_{Area} + (\frac{N_{FF} * N_{TV}}{TS_{ATPG}} + N_{TV} * T_{LC}) * C_{Time} \tag{7}$$

The equation for the STUMPS approach is much the same, but additional parameters are used. The parameters have the same meaning, and the suffix ATPG and BIST are used to separate BIST parameters from ATPG parameters. For instance A_{BIST} is the area of the BIST logic.

$$\begin{aligned} C_{BIST+ATPG} = (A_{CUT} + A_{BIST}) * C_{Area} + (\frac{N_{FFBIST} * N_{TVBIST}}{TS_{BIST}} + \\ \frac{N_{FFATPG} * N_{TVATPG}}{TS_{ATPG}} + (N_{TVBIST} + N_{TVATPG}) * T_{LC}) * C_{Time} \end{aligned} \tag{8}$$

When subtracting the two cost functions from each other one will get the test cost saving, as in equation 9. The cost due to the area of the CUT will disappear, as it is part of both equations.

$$C_{reduction} = C_{ATPG} - C_{BIST+ATPG} \tag{9}$$

One can also easily calculate the amount of saved test time by removing the area cost from the equations and just looking at the time aspect. Doing this one get equations 10 and 12 showing the required time for each of the two cases.

$$T_{ATPG} = \frac{N_{FF} * N_{TV}}{TS_{ATPG}} + N_{TV} * T_{LC} \tag{10}$$

$$\begin{aligned} T_{BIST+ATPG} = \frac{N_{FFBIST} * N_{TVBIST}}{TS_{BIST}} + \frac{N_{FFATPG} * N_{TVATPG}}{TS_{ATPG}} + \\ (N_{TVBIST} + N_{TVATPG}) * T_{LC} \end{aligned} \tag{11}$$

Subtracting these two and dividing by the time for ATPG test one get the percentage of saved time. Changing the values for test speed, length of scan chains and number of test vectors one can investigate how the amount of saved time will change.

$$T_{Saved\%} = \frac{T_{ATPG} - T_{BIST+ATPG}}{T_{ATPG}} \tag{12}$$

2.9 Clock source for use in delay fault testing

One challenge in delay testing is to know which speed to use as the maximum speed of the circuit. Process variations are a known fact, and can significantly affect the maximum frequency of a device. One can select the worst case parameters and calculate the maximum frequency for the worst temperature and voltage corner and use this as the maximum frequency for all devices. Although this would work, it also mean that a large portion of the circuits actually are able to perform much better than the frequency they are tested at. A possible solution to this is to add an oscillator to the design and use this oscillator as clock source for the launch and capture cycles during delay testing. There is one important restriction if this is going to work. The oscillator speed must track the variations of the maximum frequency of the critical path in the circuit in different process variation corners and for different voltage and temperature corners. Another problem is that it is hard to know the maximum frequency of the device before it is finished due to the fact that routing delay contributes significantly to the total delay. For this reason such an oscillator will need some means for calibration to match the critical path after the finial layout is done.

A benefit of this approach is that one is able to measure the actual maximum frequency of the device. This again can be used to do speed binning of the manufactured devices. Speed binning can be important for several reasons. First, one can sell devices, which are able to operate at very high frequencies at a higher price, but more important, one can sell devices that work at lower speed than what the actual specification specifies at a lower price instead of scrapping them.

3 Test chip design

The goal is to make a design for a chip that can be used in research on delay fault testing. As scan-testing is one of the most used test approaches today, and is likely to be a part of the design flow of many companies, it can be interesting to find a solution that uses this with some modifications to achieve delay testing. In addition, BIST is becoming more and more popular as a means to reduce test cost, and for this reason a BIST solution will also be added to this design to compare BIST to regular ATPG-tools.

Given the requirements just mentioned it seems that STUMPS/LBIST is the best-suited solution for this design. This means that the CUT and BIST logic can be developed somewhat separately as this BIST architecture is non-intrusive, except for the scan chains that is.

The circuit, which is to be tested, should be made so that it can be scaled in size to see which effect it has on the fault coverage. At the same time is must contain sufficient amount of registers so that several scan chains can be inserted into it.

Since both external scan and delay fault testing are requirements for the design a clock controller is also needed. The reason for this is that the shifting in and out of scan vectors and responses may have to be done at slower speed than the maximum speed of the design. For this reason the clock controller must be designed to support switching between two different clock frequencies during test. This switching must be glitch free as a glitch may result in X values propagating into the MISR, which will destroy the signature.

The next subsections will take a closer look at the different modules of the design.

3.1 CUT design

One of the challenges was to come up with a design that was easy to make but at the same time complex enough so that delay fault testing can be done and give meaningful results. The choice ended on a simple co-processor with SPI interface. The SPI interface was chosen due to its low pin count. It also meant that it would be easy to interface the circuit to an AVR circuit, which would be handy when verifying the design. The co-processor consists of an ALU, a register file and logic connecting it to the SPI interface. A simple overview of the design is shown in figure 11.

The SPI interface is the only means of communicating with the co-processor. Three different data package formats exist. The simplest is the one containing only an instruction for the co-processor. It is a 16bit package consisting of a 7 bit opcode and three address fields of 3 bits each as shown in figure 12. The opcode is the most significant bits, and are sent first. The opcode also tells the FSM what format the package is, and therefore controls

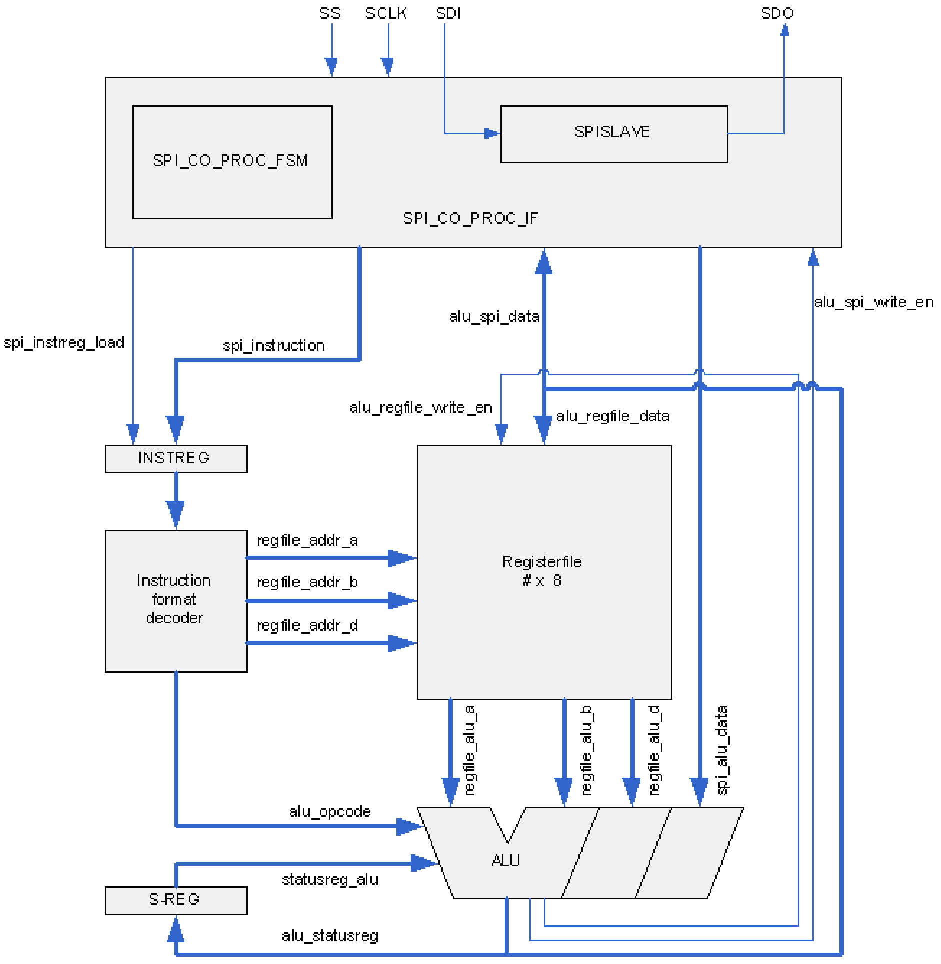

Figure 11: Co-processor overview.

parts of the FSM.

15 Opcode 9	8 Addr D 6	5 Addr A 3	2 Addr B 0

Figure 12: Instruction package

The two other formats also have and additional data field with the same size as the bit width of the processor. In these cases the 16 most significant bits correspond to the first package described, then the data follows with the most significant bits first. Table below shows the package format with data.

The difference between these two packages is who is supplying the data. If the opcode is a write command the SPI circuit knows that it is suppose to

47 Opcode 41	40 Addr D 38	37 Addr A 35	34 Addr B 32	31 Data 0

Figure 13: Instruction with data package

receive data, and goes into receive mode. If the opcode is a read command is knows that it is suppose to supply data, and it goes into the transmit state and takes data from a buffer register and sends out.

SPI can send and receive data at the same time. This is not utilized in this design, and the host need to take this into consideration when reading the data it receives.

3.1.1 The ALU

The ALU is implemented with a case-statement in verilog, using the opcode as argument. The inputs are the status register, opcode and the data inputs. Outputs are the data out, the new value of the status register and write enable signals. The reason for making the ALU in this way is that it makes it easy to add or remove instructions from it without the need for much code editing. The ALU is a pure combinational logic block, and it therefore executes all instructions in a single cycle. Most of the instructions implemented uses one or two data inputs, with data pointed to by address a and b, and stores the result at the third address location, d. There are two exceptions, the MAC instruction uses three inputs (a,b and d), where a and b are multiplied and added to d. The result is then stored back to the location pointed to by address d. The second instruction that is different is the load instruction. It has its own data input, which is connected to the data out from the SPI module. This is used to store data received with the SPI interface to the register file. For a complete instruction set summary, refer to appendix A.

3.1.2 The registerfile

The register file is made up of a large array of registers. The inputs to the register file module are the three address fields from the instruction register, the data input and the write enable signal. It has three data outputs, and each output outputs the data pointed to by the corresponding address input. When the write enable signal is active, the data at the data input port is written to the register pointed to by address d at the rising edge of the clock.

3.1.3 The instruction format decoder

The instructions for the ALU are 16 bits wide. The seven most significant bits are the opcode that determines what the ALU is going to do. The

Opcode	Addr D	Addr A	Addr B

Figure 14: Instruction

nine least significant bits are the address bits for the operands. It is divided into tree address fields of 3 bits. This limits the register file size to only 8 registers.

The instruction format decoder was originally a block that only performed splitting of the instruction register signal into the opcode and the tree addresses. Later some masking was added to mask unused bits in the opcode to increase the fault coverage during delay test.

3.1.4 The SPI slave

The SPI slave is the only part of the circuit that communicates with external circuits during normal operation. The data size for the SPI is set to 8 bits, and the mode of operation is set to CPOL=0 and CPHA=0. That is, the base value of the clock signal sclk is zero, and data is shifted in on the rising edge of sclk. The most significant bits are sent first.

One of the main problems one will encounter with SPI is that it uses two different clock signals, clk and sclk. The main system clock is clk, and sclk is the SPI data clock, which is controlled by the SPI master. The SPI shift register can be directly driven by the SPI clock. The problem with this approach is that there will be two different clock domains in the system, and this needs extra care when designing the BIST logic to prevent X propagation due to meta-stable values. The solution to this is to use just one clock signal, the main system clock, clk, and handle the SPI clock as a regular input. By sampling the sclk signal one can detect the rising edge and capture the input data on the serial data input pin, SDI. The only requirement to this solution is that the main system clock is at least four times faster than the SPI clock. Figure 15 shows a simplified overview of the SPI slave module. The main clock signal is not shown, but is connected to all registers and modules. In the figure one can see the input sampling registers and the gate used to perform edge detection.

One other thing to notice about the SPI slave is that it has a tri-state output. This output is controlled by the synchronized chip select signal, ss_n_sync. The reason for adding a tri-state buffer is that several SPI slave devices can be connected to the same SPI data lines, SDI and SDO, and SPI clock signal, but only one of them is allowed to send data on the SDO line. The SPI master selects which slave is allowed to send data by pulling the desired chip select line low, and by only having one chip select line in the active state, there will only be on SPI slave driving the SDO line at any given time.

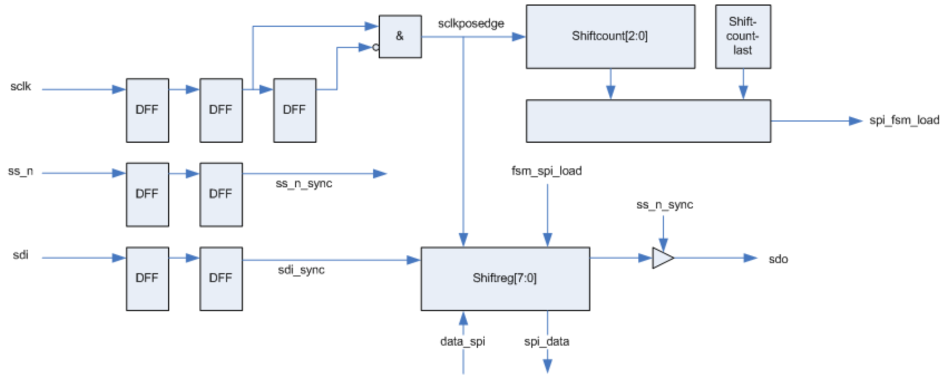

Figure 15: Simplified SPI slave overview.

3.1.5 The SPI interface and controller

Since the SPI slave operates on data of 8 bit size and the ALU operates on data of larger size, additional logic is needed to put several 8bit packages together and to split larger values from the ALU down to chunks of size 8 bits. This task if performed by the SPI_CO_PROC_IF module. It is made up of the SPI slave module a finite state machine, some counters and registers to store received values and values to be sent. The finite state machine is shown in figure 16.

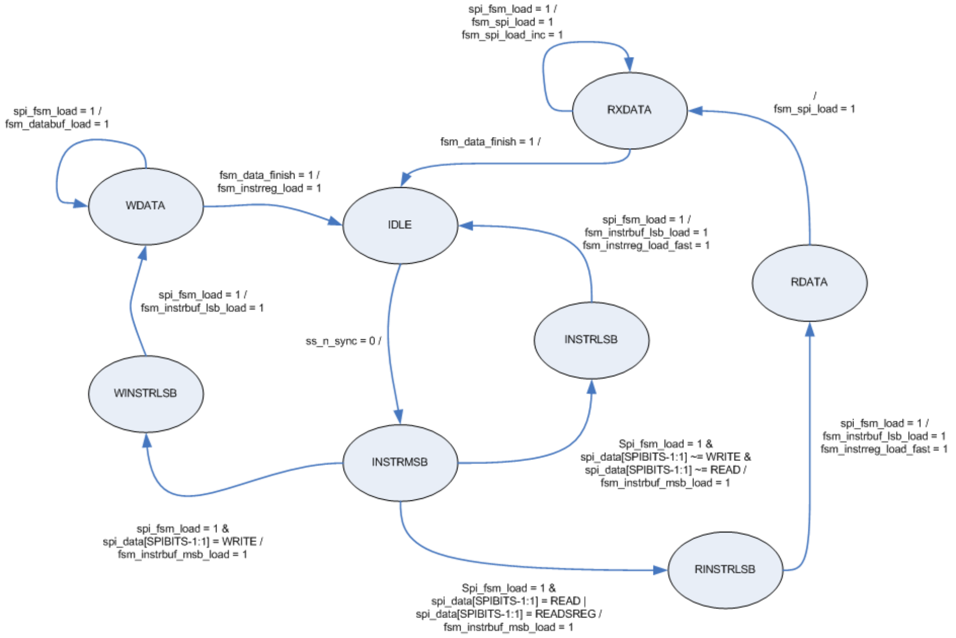
spi_fsm_load = 1 /
fsm_spi_load = 1
fsm_spi_load_inc = 1
RXDATA
/
fsm_spi_load = 1
spi_fsm_load = 1 /
fsm_databuf_load = 1
fsm_data_finish = 1 /
WDATA
fsm_data_finish = 1 /
fsm_instrreg_load = 1
IDLE
spi_fsm_load = 1 /
fsm_instrbuf_lsb_load = 1
fsm_instrreg_load_fast = 1
RDATA
spi_fsm_load = 1 /
fsm_instrbuf_lsb_load = 1
ss_n_sync = 0 /
INSTRLSB
WINSTRLSB
INSTRMSB
Spi_fsm_load = 1 &
spi_data[SPIBITS-1:1] ~= WRITE &
spi_data[SPIBITS-1:1] ~= READ /
fsm_instrbuf_msb_load = 1
spi_fsm_load = 1 /
fsm_instrbuf_lsb_load = 1
fsm_instrreg_load_fast = 1
spi_fsm_load = 1 &
spi_data[SPIBITS-1:1] = WRITE /
fsm_instrbuf_msb_load = 1
RINSTRLSB
Spi_fsm_load = 1 &
spi_data[SPIBITS-1:1] = READ |
spi_data[SPIBITS-1:1] = READSREG /
fsm_instrbuf_msb_load = 1

Figure 16: Finite state machine controlling SPI sending and receiving.

The default state after reset is the IDLE state. When the chip select goes low the FSM switches to the INSTRMSB state. Here it waits for the eight most significant bits of the instruction to be received. Since the opcode is the seven most significant bits in the instruction these are now available, and the FSM can decide if this is an write operation, meaning that data will follow the instruction, if it is an read, meaning that data should be sent, or if it is an instruction that not involves sending or receiving data. When spi_fsm_load goes high the SPI slave has new data ready, and if the opcode is the write instruction the FSM changes to the WINSTRLSB state where it waits for the last eight bits of the instruction (containing the address bits). It then goes to the WDATA states where it stays until all data is received. When the date is ready the FSM sets the fsm_instrreg_load signal active and the instruction is loaded to the ALUs instruction register. At the same time the FSM returns to the IDLE state.

If the opcode was one of the read instructions the FSM switches to the RINSTRLSB instead. Here it waits for the LSB part of the instruction, and when it is received it directly loads the instruction to the instruction register so it can be executed in the next clock cycle. The FSM moved to the RDATA state. It then goes to the RX data state where it stays until all 8bit packages are sent. It then moves to the IDLE state.

If the opcode is neither read or write it means that the ALU is going to performe some operation on the date it already has in the register file. There is no need to wait for additional data, so the FSM can move to INSTRLSB were it waits for the LSB portion of the instruction. When it is received it is directly loaded to the instruction register, and the FSM returns to the IDLE state.

3.2 BIST design

The design of the BIST logic can be divided into two several parts. The design of the pattern generator and response analyser, and the design of the BIST controller. Since one of the goals was to analyse different cases with the BIST logic, such as the effect of phase shifter, different weights, and different sizes for LFSR, it was necessary to make the system so that it could easily be modified to test the different cases.

To prevent excessive time going away in code rewriting and debugging of new code, a combination of a configurable BIST controller and HDL generator scripts was developed for generation of the BIST logic.

3.2.1 Design automation

Design automation is important as it eases the task of development of systems. Entire systems or parts of a system can be generated automatically and this saves a lot of time during development. Specifying certain rules and parameters for a system means that scripts can be made for automatic generation of code for the system. This has been used here as large portions of the BIST logic depend on rules and parameters.

The design of LFSR, phase shifters, weighting logic and other parts making up the pattern generator and analyser are most efficiently done by scripts that generate HDL code from a set of inputs. The reason for this is that the design of these logic blocks depends on several parameters and rules to give an optimal solution. By supplying these parameters in a configuration file the script can generate the code fast, and it makes it easy to test out different scenarios as just the parameters has to be changed and the scripts re-run to generate new code.

Here scripts are used to generate code for all modules of the BIST except for the BIST controller. The LFSR and MISR generator scripts take the size as arguments, it then searches through an additional file containing primitive polynomials to find a polynomial of matching size. From this it generates the code for the LFSR and MISR.

The generation of the phase shifter takes the number of scan chains as argument and compares this to the number of outputs from the LFSR. With the two additional arguments, minimum numbers of shifts between scan chains and maximum number of ones, it then generates logic which performs phase shifting. For large LFSRs and large number of scan chains, the generation of the code for the phase shifter can take significant amount of time. This is due to the maximum number of ones argument, which can make the script run for a long time before there is a hit.

The generators for weighting, flipping and concentrator are straightforward. The scripts takes arguments from the configuration file and generate code. There are no special requirements giving long runtime for the scripts, so they are very fast.

3.2.2 LFSR

The LFSR is made with a primitive polynomial and internal feedback. The choice of using internal feedback is that this will give shorter critical path for the LFSR, making it able to run faster. This is probably not that important in this circuit design as there almost certainly will be a path with worst delay anyway. One other reason for using internal feedback is that it removes some of the dependencies between the parallel outputs, which feed the scan chains. This is important, and in cases where phase shifters isn't used this is likely to increase the performance.

To increase the flexibility of the BIST circuitry the LFSR also has a serial input and serial output which could be used to scan in new seed values when the load signal is active. An example is shown in figure 17.

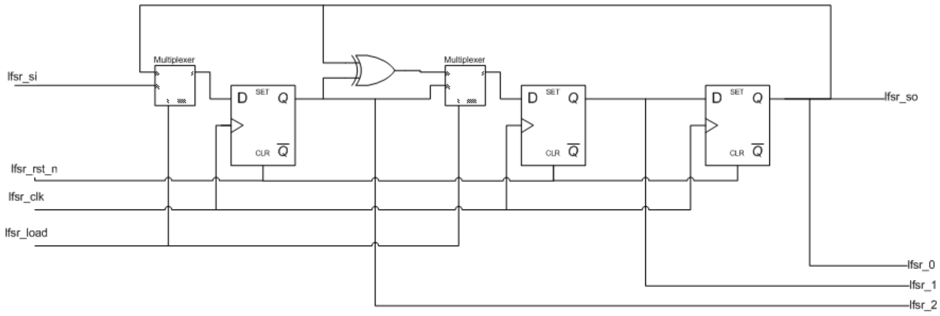

Figure 17: 3 bit LFSR with internal feedback and load capability.

3.2.3 MISR

The MISR is made in much the same way as the LFSR. It is made from a primitive polynomial, has internal feedback and a serial input and output port. The serial input and output is used for retrieving the signature from the test and can be used to set a seed value. One thing that is different from the LFSR is that the MISR has parallel inputs instead of outputs. These inputs are connected to the output from the concentrator or scan chain outputs. An example of such a MISR is shown in figure 18. Note that an actual MISR will be larger to prevent fault masking as the probability of this to happen is 2^{-n}.

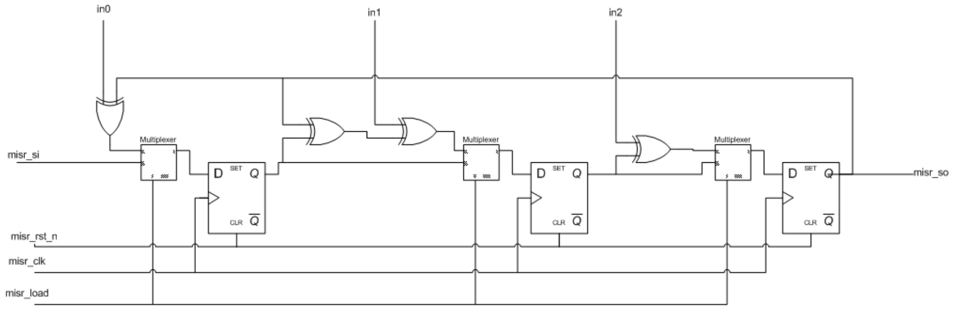

Figure 18: 3 bit MISR with internal feedback and shift in/out capability.

3.2.4 Phase shifter

Since the LFSR is going to feed several scan chains in parallel it is necessary to remove any correlation between the scan chains. A phase shifter is implemented and used for this. The phase shifter uses XOR-gates to generate new outputs, which are feed into the scan chains. The inputs to the XOR-gates are the parallel outputs from the LFSR. To limit the size of the phase shifter a maximum number of ones allowed to be XORed together is set. The result is a phase shifter that has an average of one XOR gate per scan chain.

3.2.5 Weighting logic

To increase the possibility of detecting random hard to detect faults weighting logic is added. This logic increases the number of zeros that is applied to the scan chains by AND-ing together neighbouring scan chains. The weight is selectable by multiplexers. See figure 19.

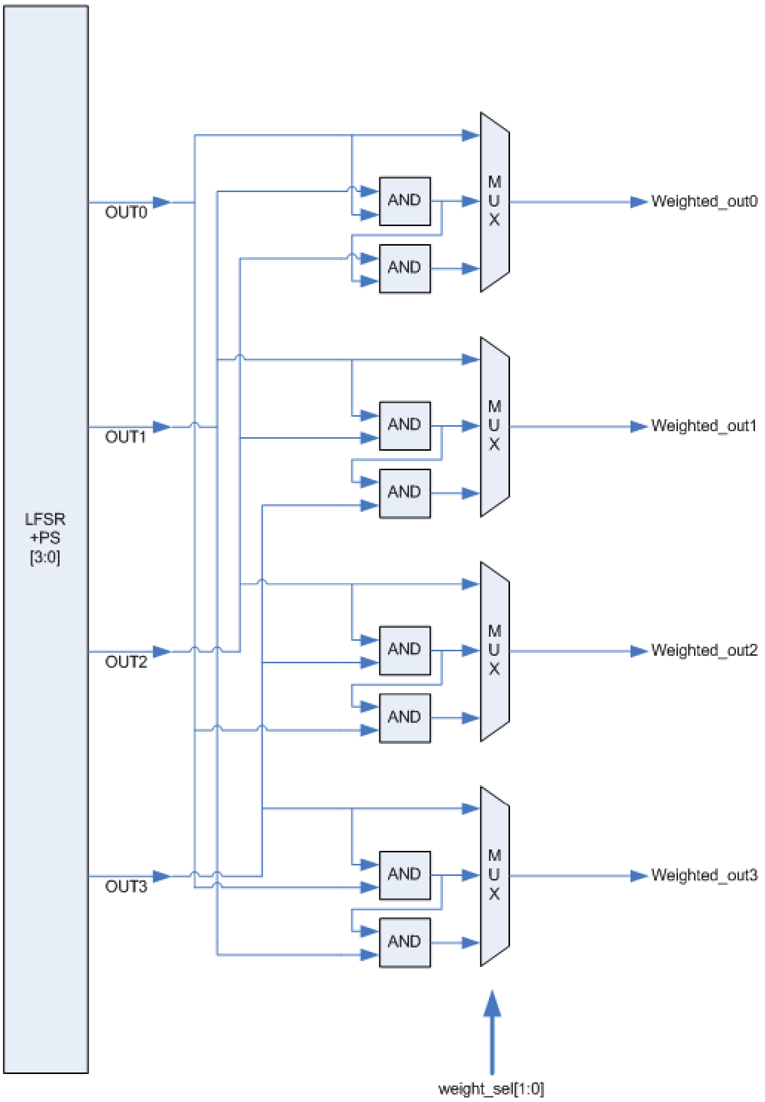

Figure 19: Connection for muxes and AND-gates for weight generation.

3.2.6 Flipping logic

Since the weighting logic increases the probability of a zero being scanned into the scan chains a flipping block is added. This can be used to flip the bits, and by doing this, one can increase the likely hood of a one being scanned in instead of a zero. The flipping block is made of XOR-gates, where each XOR has one separate control line. See figure 20.

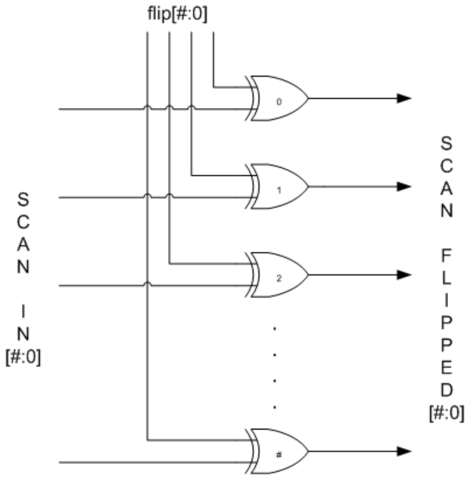

Figure 20: Logic for flipping the input data to the scan chains.

3.2.7 Concentrator

For designs with several scan chains it can be desirable to reduce the number of outputs before it is applied to the MISR. This is here done by building several small XOR-trees which feeds the inputs to the MISR as shown in figure 21.

3.2.8 BIST controller

The BIST controller is a state machine, shown in figure 22, which controls the shifting in and out of test vectors and test responses. In addition it controls which weights is applied and how many test vectors that are applied. Table 1 shows the input and output signals when going from one state to another.

The system stays in normal operational mode until the bist_start signal is activated, then the BIST controller takes over. The BIST finite state machine switches from IDLE to the LOAD_CONF state. In this state it checks different configuration registers and depending upon whether they are set or not loads a BIST configuration setting. It then goes to the SCAN_IN_OUT state. This state controls the shifting of scan vectors in to the scan chains

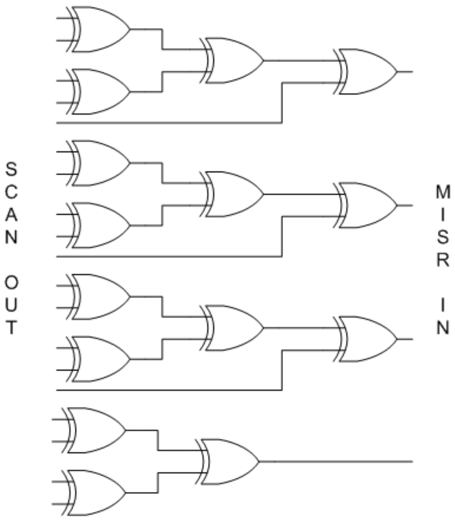

Figure 21: Concentrator logic between scan outputs and MISR inputs.

and the retrieval of the responses from the scan vectors. The first time the system is in SCAN_IN_OUT the MISR must be disabled to prevent unknown data from being shifted into it. This is achieved by and additional register, misr_enable_r, which is set active after the first test vector. The BIST controller has several wait states to support both stuck-at and delay testing. From the SCAN_IN_OUT state the finite state machine can go to WAIT0 or WAIT1. Which is chosen is determined by the conf_r[0] bit. The reasons for this is that different combinations for scan speed vs. capture speed and delay between end of scan and the capture cycle can be more optimised and thereby saving some test time. At the same time it is important that there is enough delay for the scan enable signal to settle before the launch and capture cycle.

3.3 Clock controller design

The design includes a clock controller. This is necessary as the circuit might not be able to scan test vectors at full speed, due to a slow tester or due to power handling issues. The clock controller has two input clocks, an internal oscillator and the external test clock. In normal mode the internal oscillator is used as clock source. The clock from the oscillator is applied to a counter, which acts as a clock divider. There are seven taps from the clock divider in addition to the undivided clock. One of these clocks are selected as the main clock for the system.

The BIST circuitry also uses the internal oscillator as clock source. It uses the same clock divider circuitry, but has its own select multiplexer. The only requirement for the BIST clock is that it is equal or slower than the

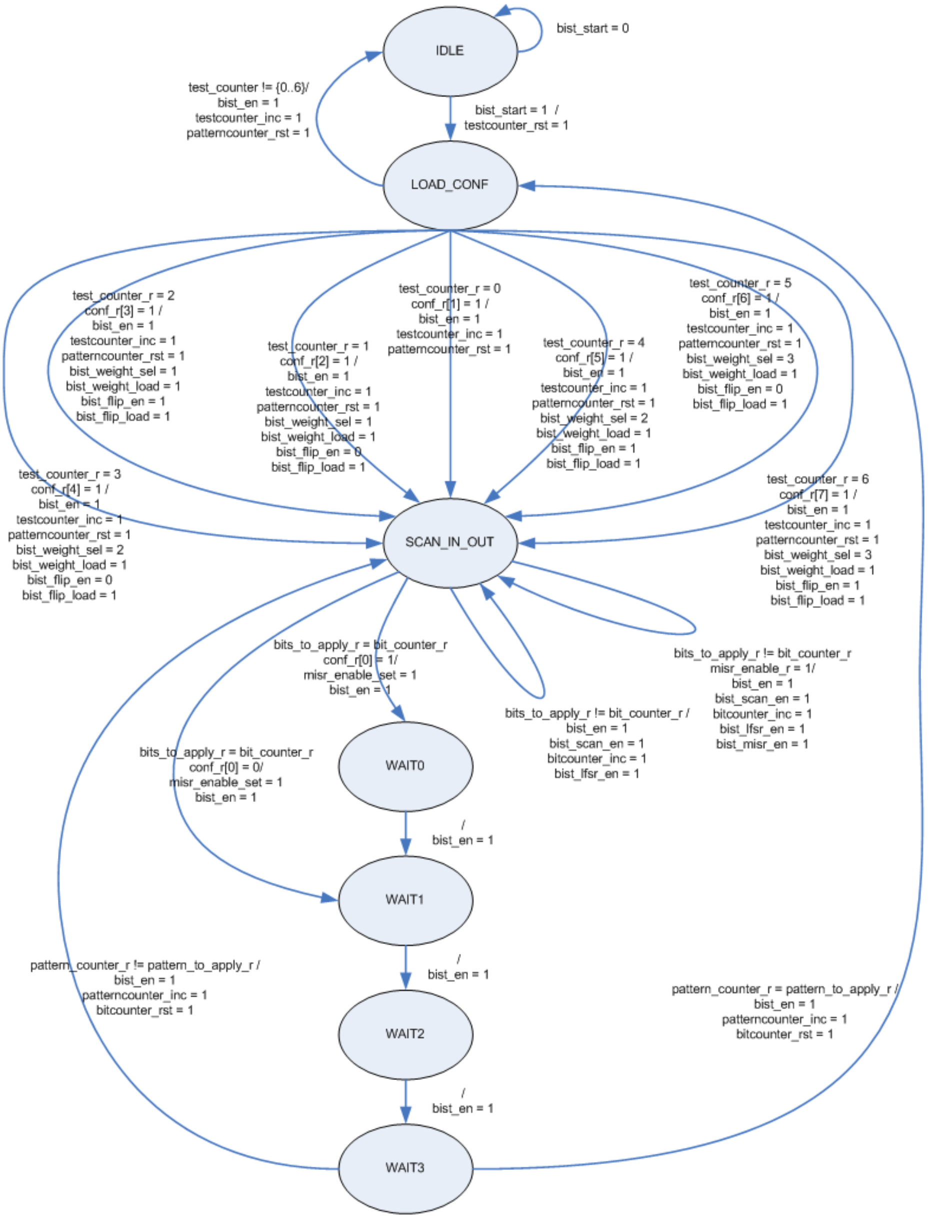

Figure 22: Finite state machine of the BIST controller.

main clock.

The external clock is used for external scan, but also during external test is the internal oscillator used for the launch and capture cycles for the main circuitry. The external clock is also used for scanning in new configuration to the clock controller and the BIST circuitry.

Current state	Input condition	Output signal	Next state
IDLE	bist_start = 1	testcounter_rst = 1	LOAD_CONF

LOAD_CONF	test_counter !=(0..6)	bist_en = 1
		testcounter_inc = 1
		patterncounter_rst = 1
			IDLE

LOAD_CONF	test_counter = 0
	conf_r[1] = 1
		bist_en = 1
		testcounter_inc = 1
		patterncounter_rst = 1
			SCAN_IN_OUT

LOAD_CONF	test_counter = 0
	conf_r[2] = 1
		bist_en = 1
		testcounter_inc = 1
		patterncounter_rst = 1
		bist_weight_sel = 1
		bist_weight_load = 1
		bist_flip_en = 0
		bist_flip_load = 1
			SCAN_IN_OUT

LOAD_CONF	test_counter = 0
	conf_r[3] = 1
		bist_en = 1
		testcounter_inc = 1
		patterncounter_rst = 1
		bist_weight_sel = 1
		bist_weight_load = 1
		bist_flip_en = 1
		bist_flip_load = 1
			SCAN_IN_OUT

LOAD_CONF	test_counter = 0
	conf_r[4] = 1
		bist_en = 1
		testcounter_inc = 1
		patterncounter_rst = 1
		bist_weight_sel = 2
		bist_weight_load = 1
		bist_flip_en = 0
		bist_flip_load = 1
			SCAN_IN_OUT

LOAD_CONF	test_counter = 0
	conf_r[5] = 1
		bist_en = 1
		testcounter_inc = 1
		patterncounter_rst = 1
		bist_weight_sel = 2
		bist_weight_load = 1
		bist_flip_en = 1
		bist_flip_load = 1
			SCAN_IN_OUT

LOAD_CONF	test_counter = 0
	conf_r[6] = 1
		bist_en = 1
		testcounter_inc = 1
		patterncounter_rst = 1
		bist_weight_sel = 3
		bist_weight_load = 1
		bist_flip_en = 0
		bist_flip_load = 1
			SCAN_IN_OUT

LOAD_CONF	test_counter = 0
	conf_r[7] = 3
		bist_en = 1
		testcounter_inc = 1
		patterncounter_rst = 1
		bist_weight_sel = 1
		bist_weight_load = 1
		bist_flip_en = 1
		bist_flip_load = 1
			SCAN_IN_OUT

SCAN_IN_OUT	bits_to_apply_r != bit_counter_r	bist_en = 1
		bist_scan_en = 1
		bitcounter_inc = 1
		bist_lfsr_en = 1
			SCAN_IN_OUT

SCAN_IN_OUT	bits_to_apply_r != bit_counter_r
	misr_enable_r = 1
		bist_en = 1
		bist_scan_en = 1
		bitcounter_inc = 1
		bist_lfsr_en = 1
		bist_misr_en = 1
			SCAN_IN_OUT

SCAN_IN_OUT	bits_to_apply_r = bit_counter_r
	conf_r[0] = 1
		misr_enable_set = 1
		bist_en = 1
			WAIT0

SCAN_IN_OUT	bits_to_apply_r = bit_counter_r
	conf_r[0] = 0
		misr_enable_set = 1
		bist_en = 1
			WAIT1

WAIT0		bist_en = 1	WAIT1
WAIT1		bist_en = 1	WAIT2
WAIT2		bist_en = 1	WAIT3

WAIT3	pattern_counter_r != pattern_to_apply_r	bist_en = 1
		patterncounter_inc = 1
		bitcounter_rst = 1
			SCAN_IN_OUT

WAIT3	pattern_counter_r = pattern_to_apply_r	bist_en = 1
		patterncounter_inc = 1
		bitcounter_rst = 1
			LOAD_CONF

Table 1: BIST controller FSM transition table.

3.3.1 Clock pipelining

The scan enable pipeline is used to ensure that enough time elapses for the scan enable to settle low before the launch or capture cycle is applied.

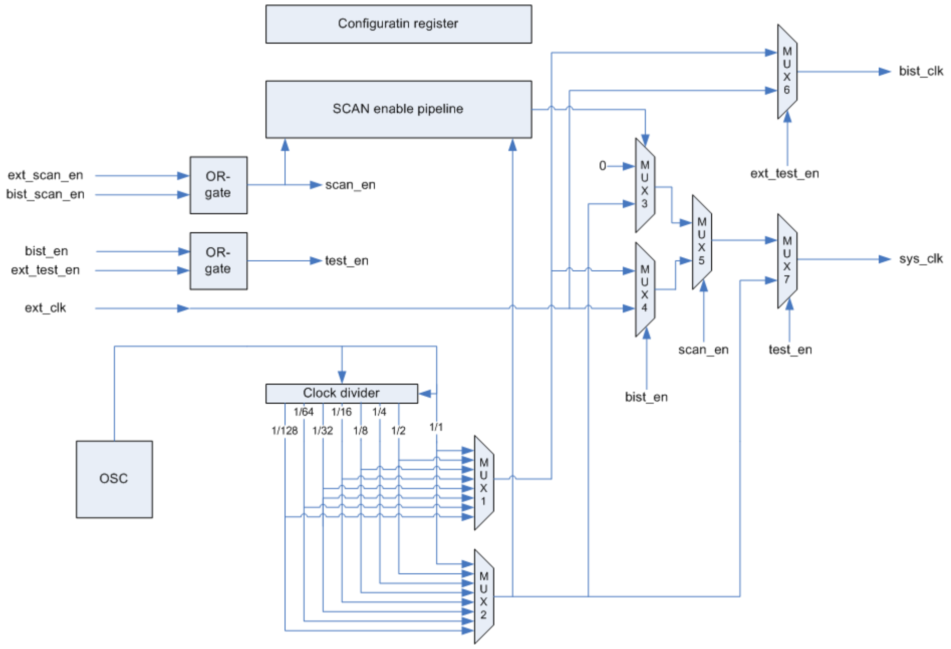

Figure 23: The clock controller.

Figure 24 shows the scan enable pipeline and logic making up the select signal for the multiplexer controlling the launch and capture clock, mux3 in figure 23. The two wires coming from the configuration registers are used to control whether one or two clock cycles should be applied during the time scan enable is inactive. In effect this controls whether a delay fault test or stuck-at test is performed.

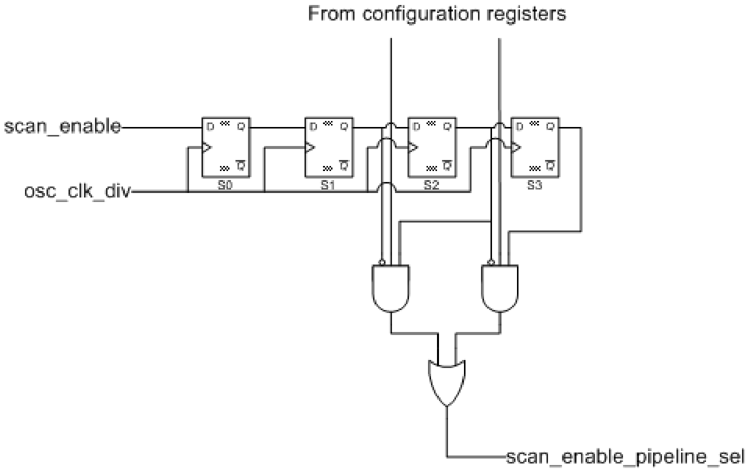

Figure 24: The scan enable pipeline.

3.3.2 Clock multiplexing

The clock controller contains a lot of multiplexers, which switches between different clock sources. Changing clock source may result in a glitch on the clock, and this can lead to unknown data and unpredictable behaviour. One way a glitch can occur is when the clock signal goes high and changes some value which in turn controls the multiplexer that applies the clock. If the two clock signals are in different state at that point the clock will return to zero, and unpredictable behaviour can occur. To prevent this from happening a multiplexer which only changes when both clock signals are in the same state are needed. Figure 25 shows an overview of the implemented clock multiplexer. The two clock signals, clk1 and clk2, have their separate enable signals, which controls the output from the AND-gates. If clk_sel is zero, the enable signal for the clk1 signal is high and clk1 is output. When clk_sel changes the logic and latch waits for clk1 to go low and then disables the enable signal for clk1. If clk2 is high at that point it waits for it to go low before the enable signal for clk2 is set high so that clk2 can be output.

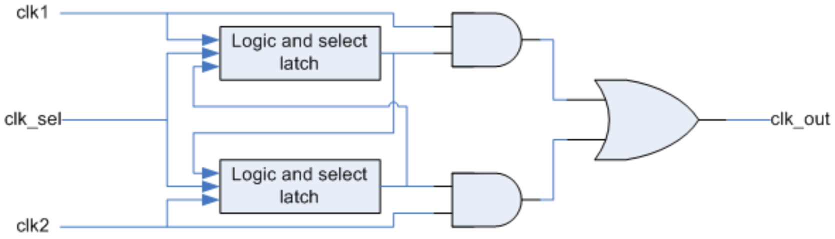

Figure 25: Clock multiplexer.

3.4 Oscillator design

In the theory it is suggested that an internal oscillator may be used as source for the launch and capture clock. The reason for wanting this is that an internal oscillator may track process variations, making it possible to test the device at the actual maximum frequency. To investigate if this is possible an oscillator consisting of inverters was made in SPICE. In addition multiplexers was added making it possible to select the length of the inverter chain to scale the frequency to some degree. A critical path from an existing device was reported using PrimeTime, and this path was also implemented in SPICE for simulations making it possible to investigate whether an oscillator made of inverts will be affected in the same way as an actual path in the circuit. [1]

[1] Spice files for the oscillator design are not attached as they also were used for another chip design, and for this reason are considered confidential.

3.5 Scan chains

The CUT is synthesised with several scan chains. Figure 26 shows how the internal scan chains are connected together for external scan, and how they are connected for BIST depending on a select signal controlling the multiplexers.

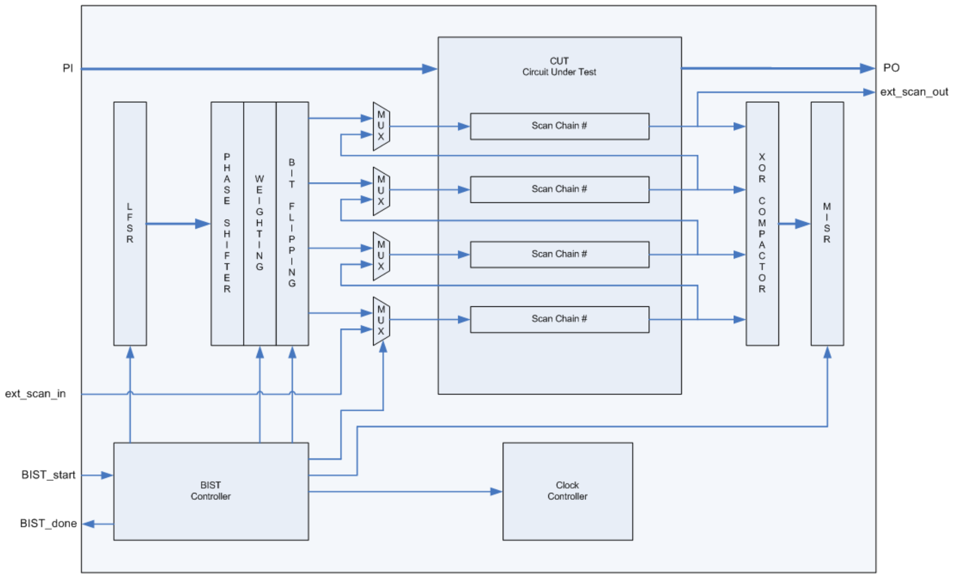

Figure 26: Scan chain overview.

In addition to the scan chains shown there is an additional scan chain. This chain goes through all configurations registers for the clock controller and BIST controller, and can be used for applying new configuration and for external scan test of these modules.

4 Chip verification

Verifying that a design works correctly is important. One of the most common ways to test a design is to make a testbench in HDL and instantiate the circuit, which is to be verified, into the testbench. One of the problems with this approach is that the testbench might not simulate real life signals, and can give results that are incorrect. The solution to this is to validate the design on an FPGA in a realistic environment.

Both methods have been used in the development of this design. The co-processor design was first verified using testbenches. Then it was synthesized for a FPGA and tested in a more realistic environment. The clock controller and BIST logic have only been verified using testbenches. The reason for this is that additional ATE would have been required to test it in a realistic setting.

4.1 CUT verification

The CUT, or co-processor, has a SPI interface, which is used to connect to other circuitry. A testbench, which simulates a SPI master, was developed and used to verify the operation of the co-processor. The testbench starts with resetting the circuit. Then it sends two write commands, writing data to the register file in the circuit. The two write commands writes to different registers and these registers are then use as arguments in an addition command to the ALU. To see if the ALU executed the instruction correctly a read commend is used to read back the result of the operation and one can then see if the ALU executed the instruction correctly. Figure 27 shows the waveform for this simulation. The bottom line is the chip select line, ss_n, and it is pulled down at the very start. Right after this one can see that the SPI clock starts toggling, and that SDI and SDO toggles as command (cmd) and data are sent to the circuit. At about 9990ns after start in the waveform the first write command is finished and one can see that the chip select line is deactivated. The serial output returns to the high impedance state and the SPI clock is stopped. Then the second write command is sent, and the same events occur with some changes to the data sent. Then the command is sent. One can see that this event is much shorter than the two previous events. This is due to the fact that no additional data is sent, just the command that is to be executed. At last the read command is sent and the events are much the same as in the first two cases. One thing that is different is that some small spikes occur on the SDO line. These are due to reloading of new data into the SPI shift register.

Simulation and waveform inspection was used a lot during the development of the co-processor and used to locate and remove design errors.

Since the circuit has a SPI interface it would be an easy task to validate it on an FPGA, which in turn was connected to an AVR circuit with SPI. The

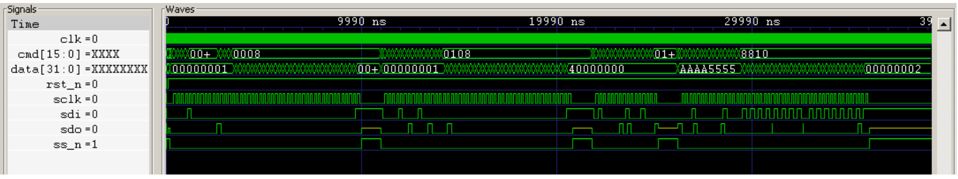

Figure 27: Waveform of SPI signals performing write, read and one addition.

co-processor was instantiated in a top module for the FPGA, this module only maps signals from the co-processor to the pins of the FPGA. A debug output was added from the SPI finite state machine, and connected to LEDs on the FPGA kit. The FPGA used was an Altera Cyclone II, mounted on Altera DE1 board [2]. This board was then connected to Atmel's STK500 board [3] with an ATMega8515 AVR micro controller. A C program was written for the AVR MCU and uploaded using AVR studio. The STK500 was then connected to a serial terminal on a computer so that debug output could be viewed.

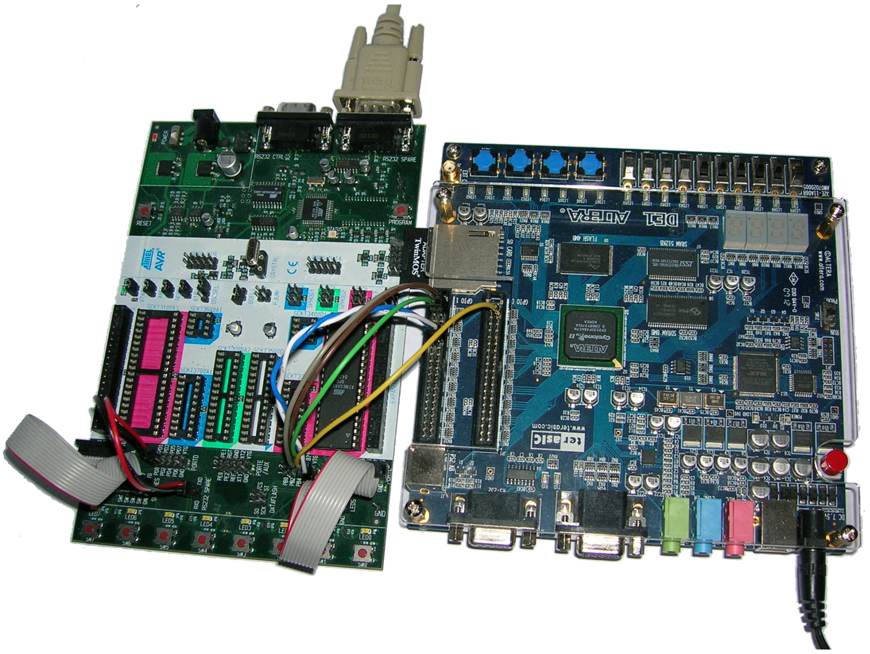

Figure 28: The validation setup with Altera DE1 and Atmel STK500.

The C program performs several tests. It first writes to all the registers, and the reads from all registers to check that they all can be read and written. It then tests several of the instructions of the ALU before it enters

[2] http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=83

[3] http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=2735

a loop that performs counting by reading, writing and addition instructions on the co-processor. The connection between STK500 and the FPGA board is described in appendix B.

One important design issue was found using the FPGA approach compared to the testbench approach. The testbench approach uses only two clocks, the SPI clock and the fast system clock. The problem with the testbench is that these clocks are triggered at the same time, making asynchronous signals synchronous. When using the FPGA kit and STK500 this will not happen as there in fact are two different clock signals. The AVR circuit, which act as the SPI master, makes the SPI clock, and the clock circuit on the FPGA kit makes the clock for the co-processor. When running the test program it was found that at some random time from start it crashed and the FPGA was locked up in an unknown state. Adding the debug output from the SPI finite state machine and running the SPI communication at very slow speed made it possible to locate the fault as something wrong with the values entering the SDI on the SPI module. After some research it was found that there was a problem with the sampling, most likely metastable values entered the circuit, but this was easily fixed by correcting the sampling of the signals.

4.2 BIST verification

The BIST and clock controller verification were done entirely with testbenches. Several testbenches were made to test the different modules. First the BIST controller was verified using a simple testbench and by studying the waveform. Second, the BIST controller and clock controller was simulated together. The reason for this is that the clock controller uses signals controlled by the BIST controller, and the easiest way to get correct signal assignment is to use the BIST controller. Next, the entire BIST logic and clock controller was simulated together. To verify that scan vectors and responses could be applied and retrieved correctly a very simple circuit was made, which was connected to the scan input and output from the BIST logic. The circuit consists of several adder circuits, which adds one to the current value at each rising edge of the clock. In addition they have the possibility to form shift registers when the scan enable signal is activated.

From figure 29 one can see that when the BIST start signal is set the system clock stops until scan starts.

From figure 30 one can see the waveform of the launch-of-capture cycles. One can see that the scan input value changes for every clock cycle of the system clock as long as the scan enable signal is set. When the scan enable signal is low the inputs are constant, but one can see that there are transitions in the circuit by looking on the scan output signal. When the scan enable signal is activated one can see that shifting in the scan chains are resumed, and new test vectors are shifted in as old are shifted out.

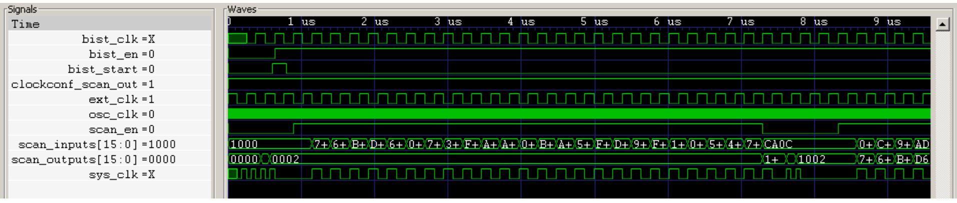

Figure 29: Wavform of BIST startup and first scan pattern.

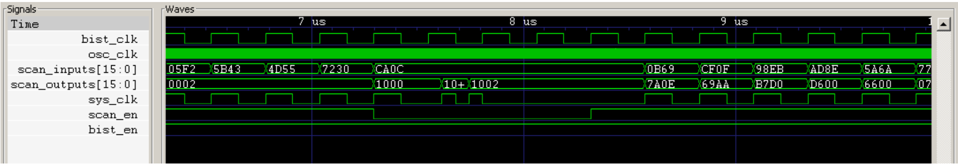

Figure 30: Waveform of a launch from capture cycle.

One other important property of the clock controller is that it can change the clock speed by selecting from several outputs of a clock divider. Figure 31 shows new configuration for the clock controller being scanned in through a separate scan chain going through the configuration registers. After the new configuration is scanned in one can se that the clock frequency is changes. The reason for the time delay between the scan of new configuration and the change in clock speed has to do with the fact that the clock multiplexers only changes which signal is output when the signals are low to prevent glitches on the clock signal.

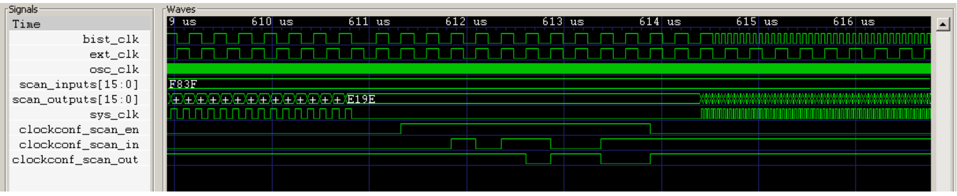

Figure 31: Wavefrom showing new configuration to the clock controller being applied.

5 Test approach

When developing a BIST solution one need do simulations to determine how well it performs. Fault simulations are necessary to determine which faults the BIST logic can detect, but also to generate the ATPG top up patterns. This chapter describes the setup for test pattern generation, fault simulation of the generated patterns, how to determine the amount of top up patterns needed, and how the circuit was synthesised.

5.1 Circuit synthesis

One of the challenges with the circuit made was how it should be synthesised. Since a LBIST architecture is implemented, one of the requirements is that the circuit under test has to have several scan chains. These scan chains are then to be connected to the BIST logic. Since several test are to be done, and this requires the CUT to be resynthesised with different settings it was decided that it would be easiest to synthesise the CUT separately from the rest of the circuit. Although it would be possible to synthesis the entire circuit and connect the scan chains to the appropriate nodes in the circuit in one run, this approach was not chosen as this would have required that the setup of the scan chains had to be done more thoroughly, as registers not in the CUT cant be in the scan chains connected to the BIST logic. Another problem that would have arisen if the circuit were synthesised in one run was how the BIST patterns should be made and stored.

Synthesis was done with Synopsys Design compiler / DFT compiler. Two synthesis scripts were made, one for the CUT and one for the BIST and clock controller logic. Both scripts generate a verilog netlist of the circuit, and these two netlist can then be connected together in a top module using only assign statements to generate the entire netlist. The synthesis also reports area and delay for the circuits.

5.2 BIST test vector generation

Generating the test vectors from the BIST logic is an easy task. This can simply be done by simulating the BIST logic in a testbench and monitoring and saving the output from the TPG. But of course this is not the case. The responses from the test vectors are also needed, and must be saved. So are the values of the primary input and output of the CUT, so these must also be monitored and saved. This complicates the test vector generation. The way this was solved was by simulating the entire circuit. The fact that the CUT was synthesised separately from the rest of the circuit would come in handy in this case. This made it easy to monitor the primary input and outputs of the circuit, and at the same time made it easy to capture the data shifted into the scan chains and responses shifted out of the scan chains.

A testbench was developed for this purpose. The testbench instantiates the synthesised CUT, the verilog models for the BIST logic and the clock controller circuitry. The reason for not using the synthesis logic for the BIST controller and clock controller is that these modules have control parameters that would be changed during different simulations and instead of having to resynthesis these modules it was easier to recompile the testbench as this had to be done in either case. In addition to these modules some additional logic was added to the testbench. To be able to save the test patterns and test responses additional shift registers was added to the output from the TPG and from the output of the CUT. The test vectors from the TPG are shifted into both the CUT and the additional shift register. At the same time the test responses are shifted into the MISR and the additional shift registers at the output. These additional shift registers are then used to store the test vectors and test responses to different files, which can be used by the fault simulator to determine the fault coverage of the test vectors. The values of the primary inputs and outputs are also saved to files, which also will be used by the fault simulator.

5.3 Fault simulation of BIST patterns

Fault simulation was done with Synopsys Tetramax. Tetramax is an ATPG tool, which is used to generate deterministic test patterns for a circuit. Because of this, additional work was necessary to generate the test patterns made by the BIST circuitry, as described in the previous section.

The first step was to use Tetramax to generate deterministic test vectors for the CUT and save them. After studying the format of the saved test patterns the testbench, which was used to simulate the circuit and to generate the BIST patterns, was developed. This testbench instantiates the synthesised CUT together with the BIST logic, clock controller and oscillator modules. When simulation is run the BIST patterns and responses are generated and stored to files. These patterns are then read by Tetramax and used in regular fault simulation on the CUT to determine the fault coverage of these patterns. Tetramax was only able to simulate on the CUT as it did not support simulation of the entire circuit including the BIST logic and clock controller.

5.4 ATPG top up patterns

One of the most interesting results when making a LBIST design is the amount of ATPG top up patterns needed to achieve the desired fault coverage. This was also determined by fault simulation with Tetramax. By first simulating the patterns generated by the BIST logic, then switching to Tetramax's internal pattern source, only the remaining faults are simulated and generated patterns for. The simulation was run until Tetramax was unable to detect more faults. The generated top up patterns was then saved, and both sets of test patterns were simulated and the coverage was reported and saved.

5.5 Analysing the results

Several fault simulations were done to see how different configurations affected the fault coverage. The output from Tetramax was saved in log files for the different cases. The logs contain a lot of information, but only a part of it is of interest. The data of interest is the part showing how the fault coverage increases with the number of applied test patterns. This data is used to generate graphs for easy comparison of the different scenarios. As this data is a part of a huge log file a script was made to search through all log files for this data and copy it into a new file as a comma separated list. This made it easy to import the data into Microsoft XL for graph generation.

6 Results

This chapter presents the results of the synthesis and fault simulations. The results are extracted from different log and report files and imported to XL to make grafts for easy comparison of the results. The results are divided into stuck-at fault simulations, transition fault simulations, and area reports from the synthesis. In addition there is a section showing how RTL-code can affect the fault coverage.

The reported fault coverage is for the CUT only and does not include fault coverage of the BIST logic and clock controller logic. Therefore it is likely that the actual fault coverage will be lower when the fault coverage of these modules is added to the results. The reason for not to included the BIST and clock controller is that the tools used could not analyse this with the setup used.

6.1 RTL code modifications

It was found that the CUT had very low fault coverage for delay fault testing using BIST. The reasons for this were two design issues. The instruction register resets to a default value at each rising edge of the clock when it is not written too, and only 21 of 128 possible opcodes are implemented. This issues were fixed by adding an additional register and by masking the opcode.

Figures in this section shows how ATPG and BIST fault coverage is affected by the way the design is made. Simulations are done for transition fault testing. For the BIST solution a 8bit LFSR was used together with phase shifters.

Figure 32 shows the results for patterns made by ATPG. Both cases are for a 32bit design. The non-code-fix run is from the first version of the design and the code-fix run is for the same design after changes was made to the instruction register and opcode to increase the fault coverage. One can see that these changes had very little affect when ATPG vectors were used.

Figure 33 shows the same circuits simulated with patterns made by the BIST circuit. One can see that the case with unmodified code, ”32 bit”, has very low fault coverage. The simulation ”32bit fixes instruction register” shows the effect of adding an additional register to pipeline the instruction. The last case, ”32bit fixed instr.reg and opcodes”, shows the finial circuit design with all modifications.

Figure 34 shows the same results as for figure 33, but the results are shown for the individual modules in the CUT.

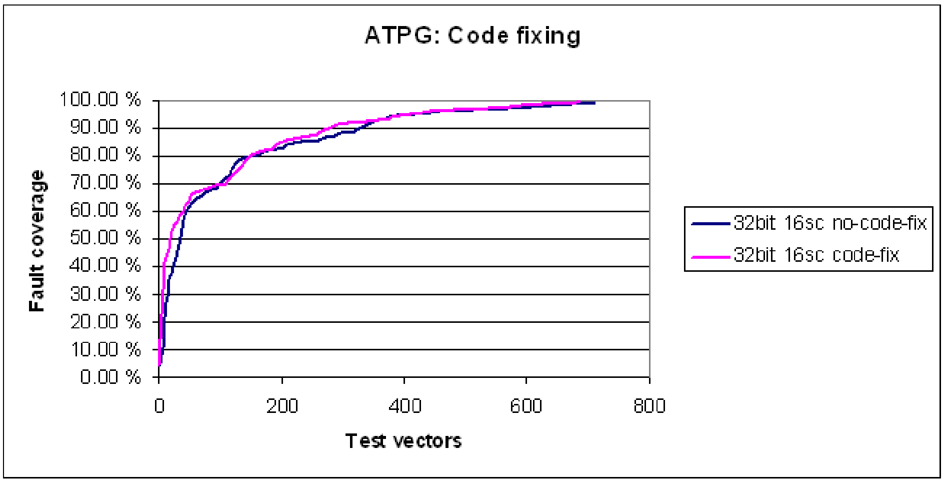

Figure 32: RTL-code fixes under ATPG simulation.

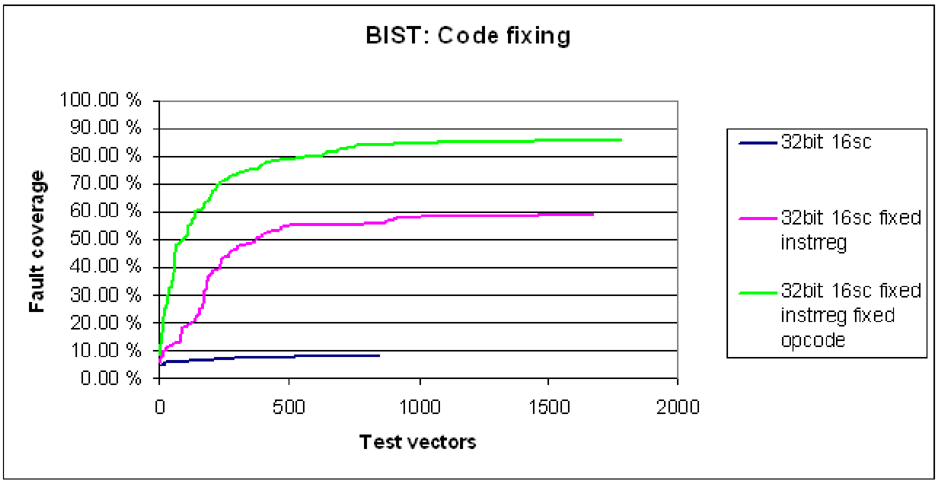

Figure 33: Simulation for RTL-code fixes under BIST.

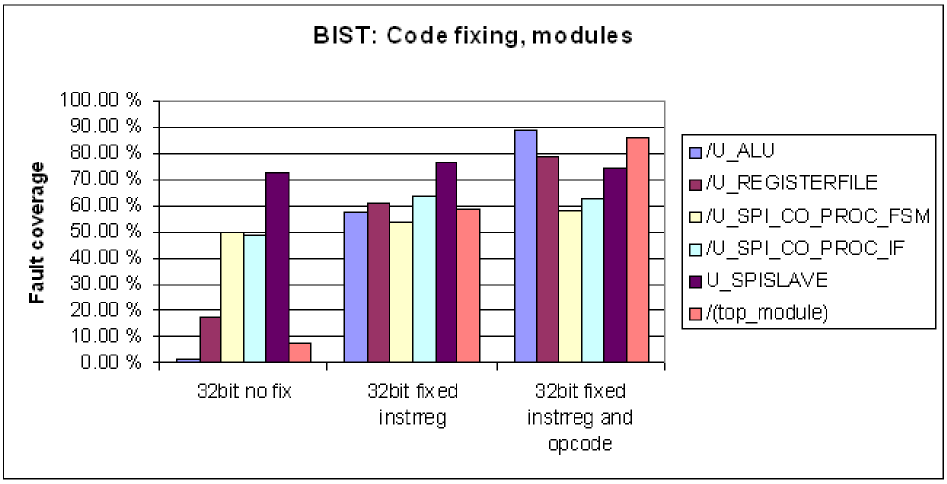

Figure 34: Module overview for RTL-code fixes.

6.2 Stuck-at fault simulations

This section shows the result from stuck-at fault simulations. Both ATPG and BIST vectors are simulated for different cases. All simulations are done with the design with fixed RTL-code for increased fault coverage.

6.2.1 ATPG stuck-at fault simulations

Figure 35 shows how the amount of scan chains will affect the amount of test vectors and the fault coverage for ATPG vectors. As one can see there is very little difference between 1 and 16 scan chains.

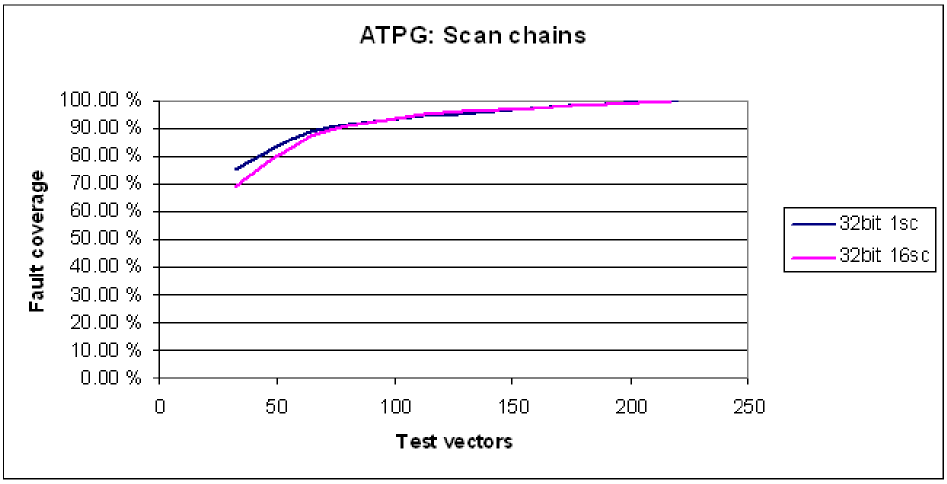

Figure 35: Simulation for different number of scan chains for ATPG.

Figure 36 shows how the CUT size will affect the fault coverage for ATPG vectors. The cases simulated are 32, 64 and 128 bit CUT size. All circuits was synthesised with 16 scan chains.

Figure 37 shows how the number of test vectors are affected for the same simulations. One can see that larger CUTs require more test vectors to achieve approximately the same fault coverage.

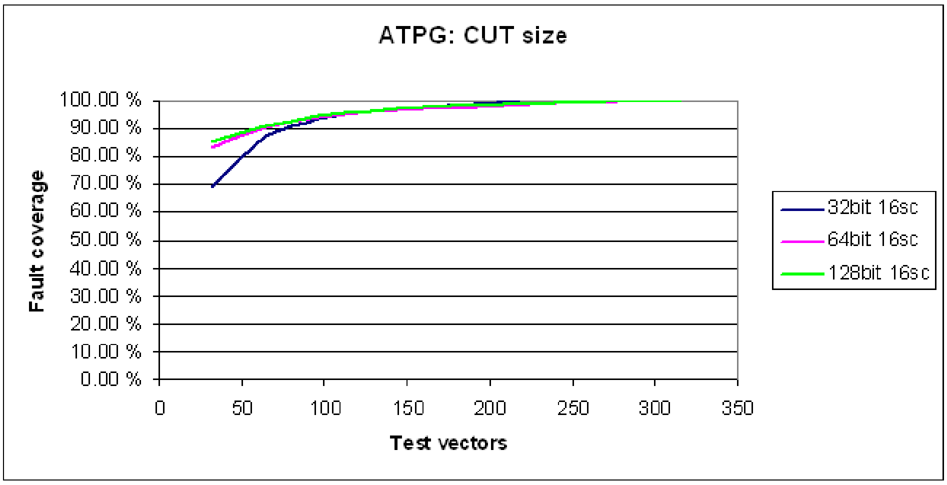

Figure 36: Simulation for different CUT sizes for ATPG.

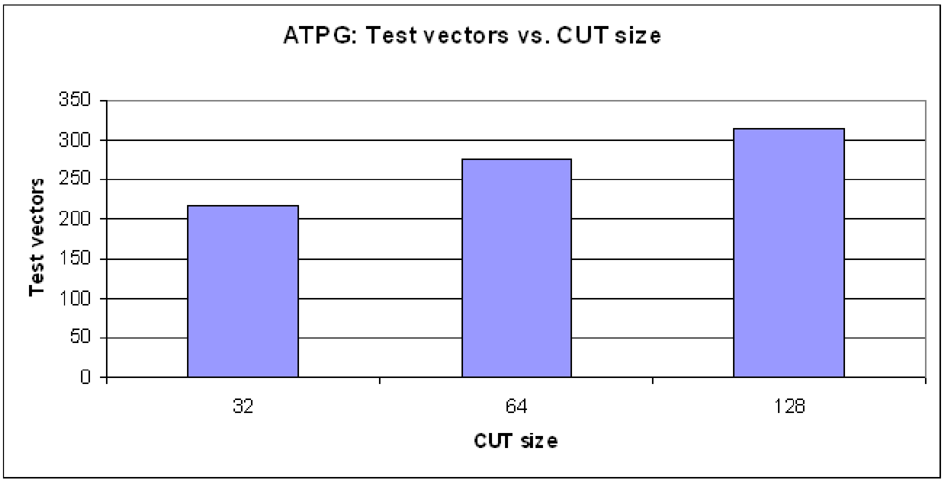

Figure 37: Simulation for different CUT sizes for ATPG.

6.2.2 BIST stuck-at fault simulations

Figure 38 shows how the fault coverage is affected by the size of the CUT when BIST patterns are used. The same BIST logic is used in all the cases. The LFSR size is 10 bits, and the CUTs were synthesised with 16 scan chains. Simulation was done for 1024 random patterns.

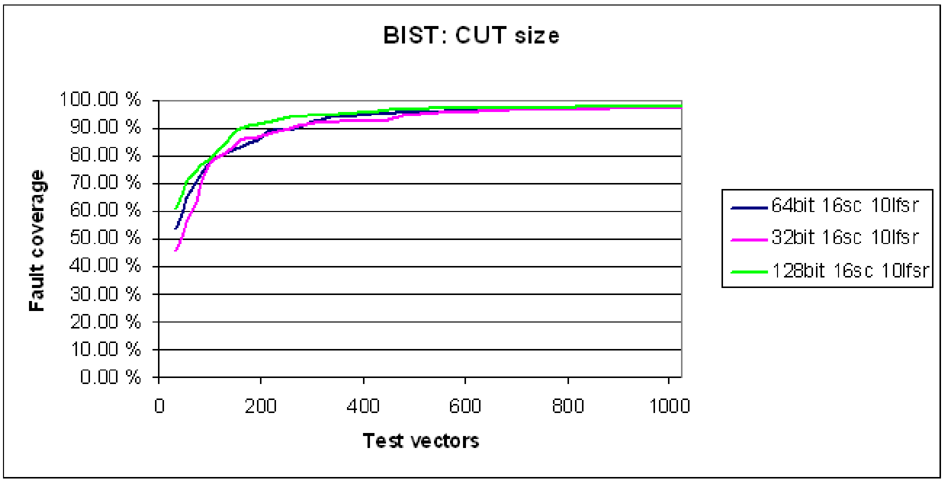

Figure 38: Simulation for different CUT sizes for BIST.

Figure 39 shows the obtained fault coverage for the same simulations when the same amount of test patterns are applied. The difference in fault coverage is very little.

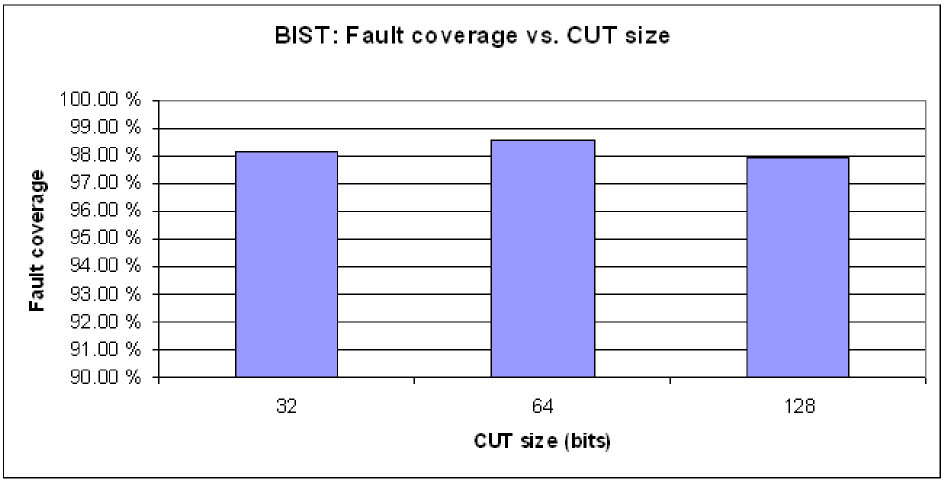

Figure 39: Simulation for different CUT sizes for BIST.

Figure 40 shows the effect of different LFSR sizes. All simulations are done on the same CUT with the same number of scan chains. The only difference is the size of the LFSR and the phase shifter, which is remade to fit the LFSR. All simulations are run for the maximum length and with all weights, but only the first 2000 patterns are shown in the figure, as there is almost no change after this.

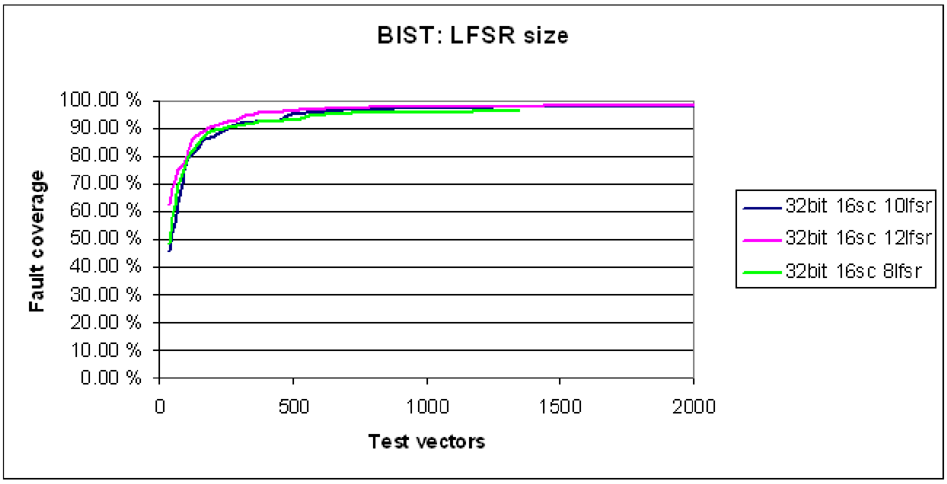

Figure 40: Simulation for different LFSR sizes for BIST.

Figure 41 shows the obtained fault coverage for the different LFSR sizes. One can see that the larger LFSR results in higher fault coverage.

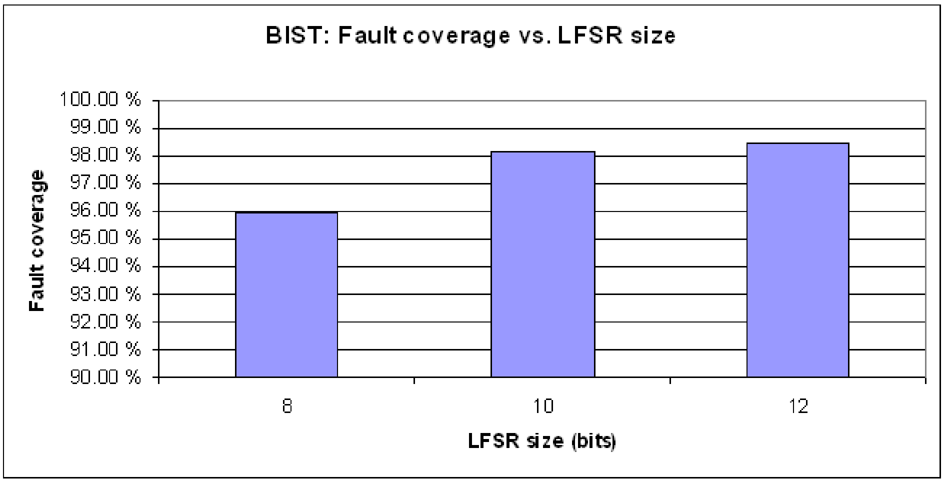

Figure 41: Simulation for different LFSR sizes for BIST.

Figure 42 shows how the number of scan chains can affect the fault coverage. The same size for LFSR is used in all cases, but the phase shifter is remade to fit the number of scan chains. The simulations are only run for 1024 patterns for each weight, but all weights are used. Figure 43 shows a close-up of the random patterns for the same simulation. The two cases have about the same fault coverage.

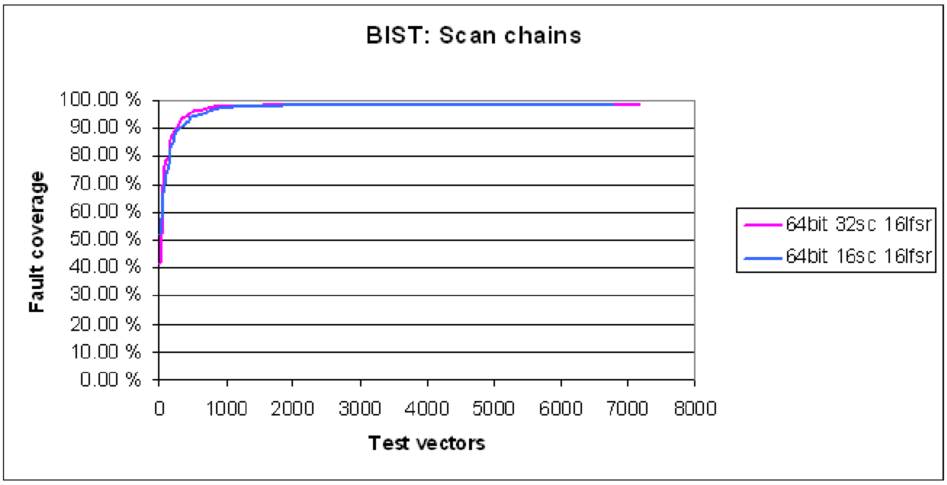

Figure 42: Simulation for different number of scan chains for BIST.

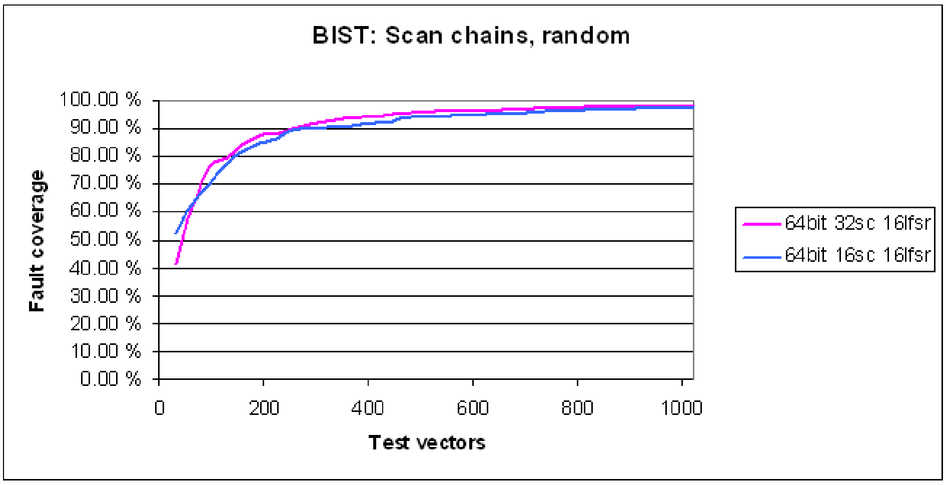

Figure 43: Simulation for different number of scan chains for BIST.

6.3 Transition delay fault simulations

This section will look at transition fault simulations for both ATPG and BIST patterns. All simulations are done on the CUT with fixed RTL-code for increased fault coverage.

6.3.1 ATPG transition fault simulations

Figure 44 shows how the number of scan chains affect the fault coverage for transition fault tests using ATPG vectors. As one can see there is very little difference.

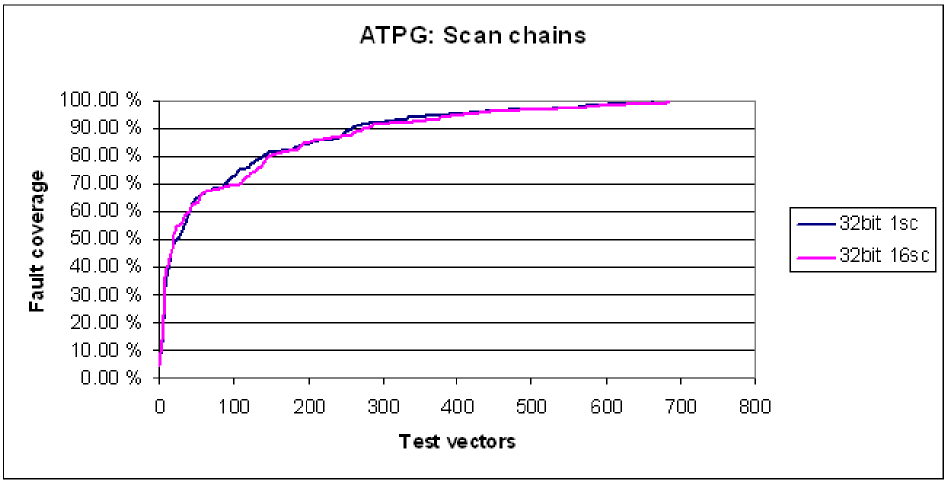

Figure 44: Simulation for different number of scan chains for ATPG.

Figure 45 shows how the CUT size affects the fault coverage. The simulations are done for 32, 64 and 128 bit CUT size. The same number of scan chains are used in all cases.

Figure 46 shows how the number of test patterns are affected by CUT size when about the same fault coverage is obtained. The graphs show that larger CUTs require more test vectors to obtain about the same fault coverage.

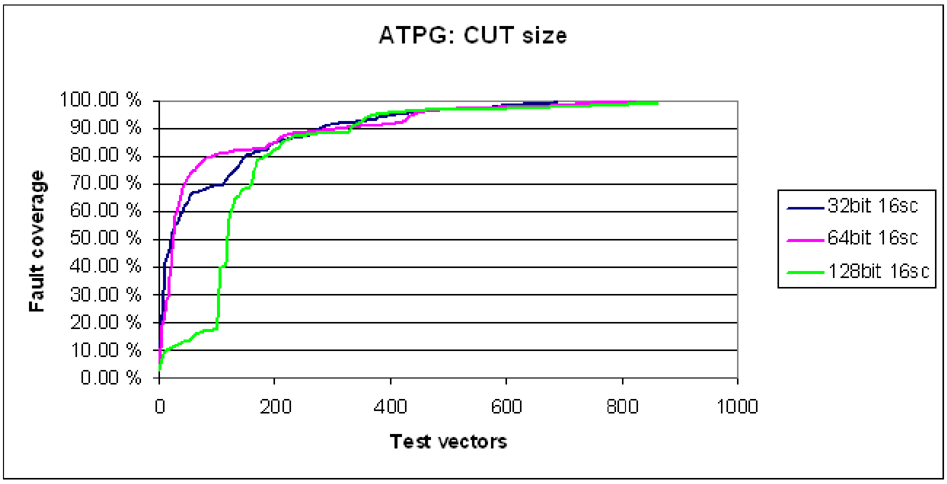

Figure 45: Simulation for different CUT sizes under ATPG.

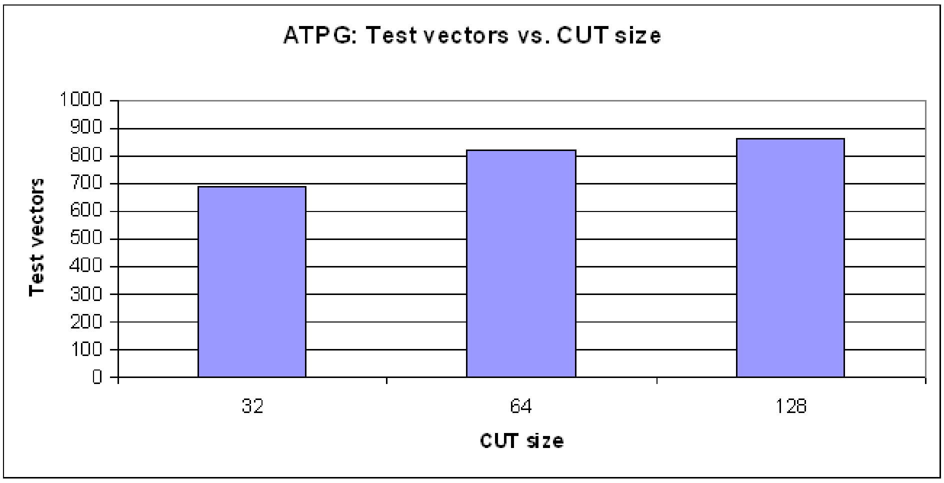

Figure 46: Simulation for different CUT sizes under ATPG.

6.3.2 BIST transition fault simulations

Figure 47 shows how CUT size affects the fault coverage when transitions testing is performed with BIST patterns. The same BIST logic is used for all circuits. The CUT has 16 scan chains and the LFSR size is 10 bits. The BIST logic is run for maximum length with all weights.

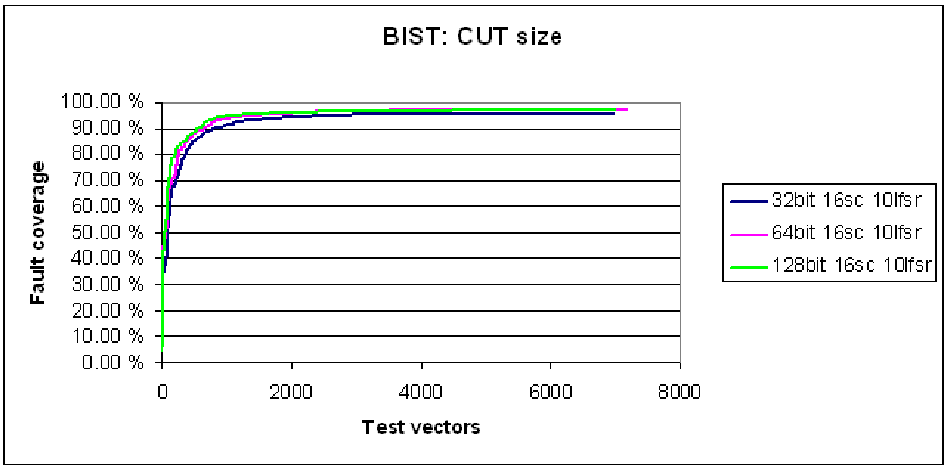

Figure 47: Simulation of different CUT sizes.

Figure 48 shows the obtained fault coverage for the same simulations.

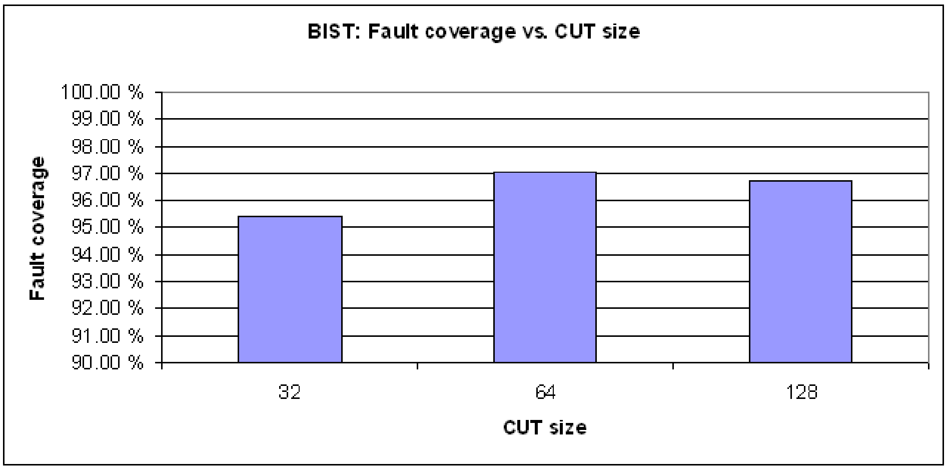

Figure 48: Cut size vs fault coverage.

Figure 49 shows how the LFSR size affects the fault coverage. Here the same CUT is used for all cases and the phase shifter is updated to match the LFSR to the number of scan chains. The figure shows the result when random patterns are applied.

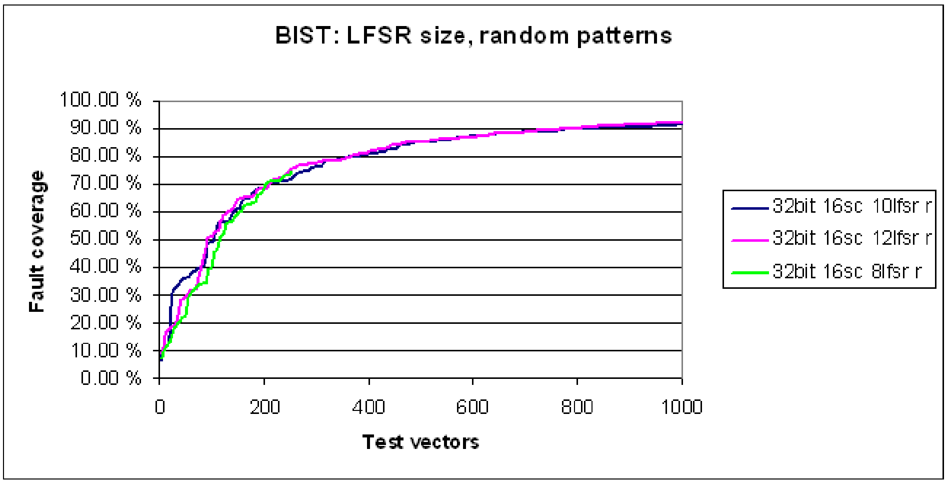

Figure 49: Simulation of different LFSR size and random patterns.

Figure 50 shows the same simulations but in this case the BIST is run for the maximum length with all weights. One can see that larger LFSR results in higher fault coverage.

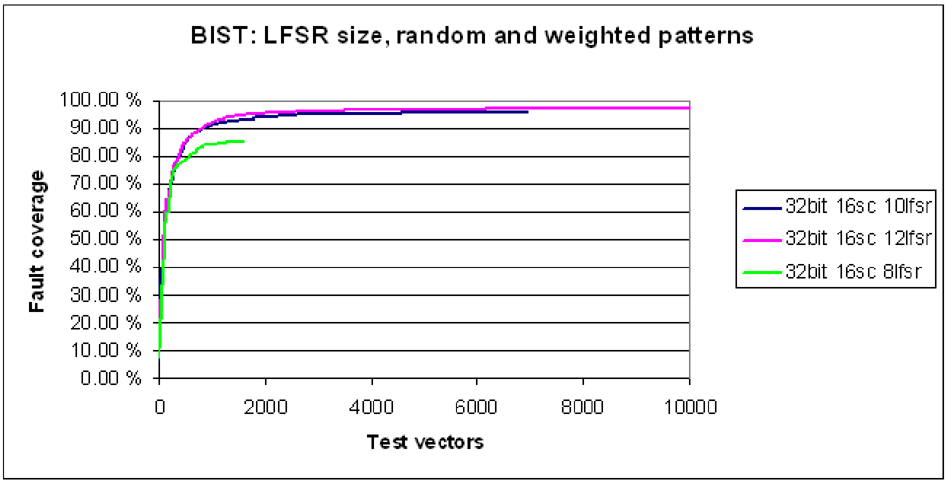

Figure 50: Simulation for different LFSR sizes.

Figure 51 shows how the number of scan chains affect the fault coverage for transition fault testing with BIST. One can see that there is very little difference. The simulation is run for 1024 patterns for all weights. Figure 52 shows a close-up of the random patterns.

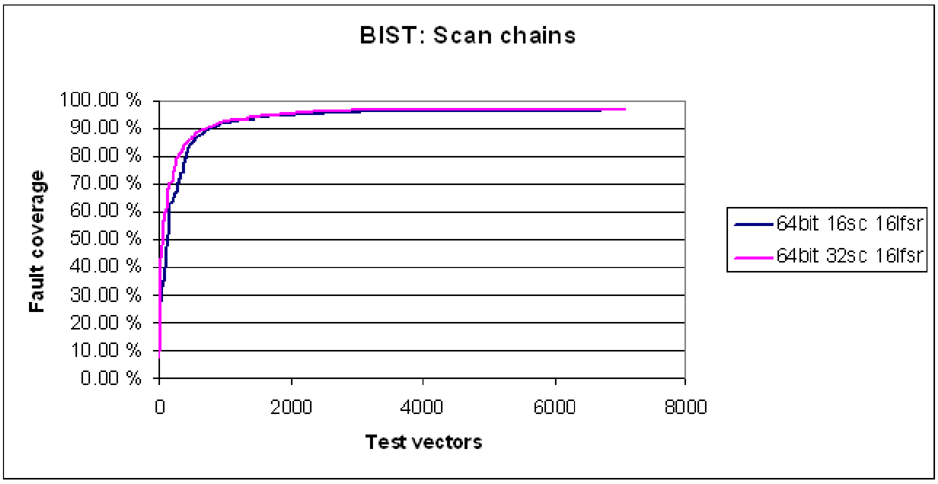

Figure 51: Simulation of different scan chains.

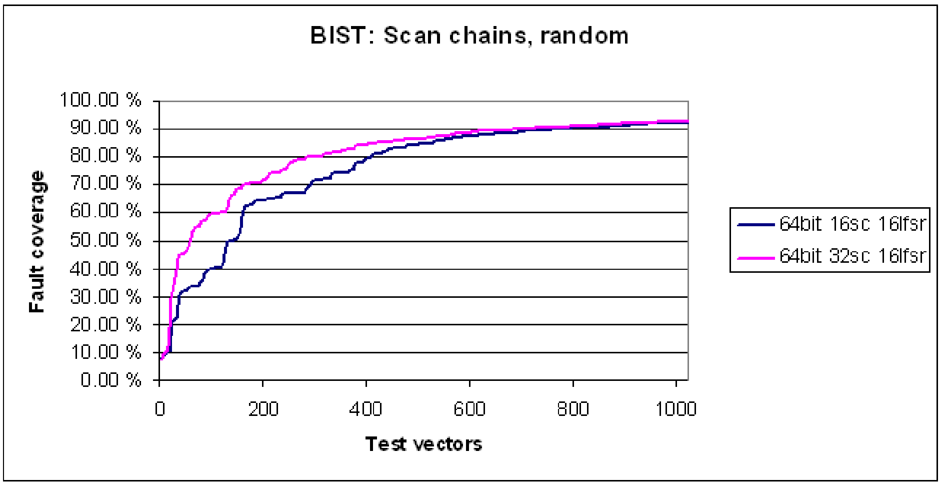

Figure 52: Simulation of different scan chains, random patterns.

Figure 53 shows how the weight applied to the patterns affect the fault coverage. The number after the w tells how many signals are ANDed together to create the weight. Those with an f at last are in addition flipped to create the opposite weight. The simulations are done with the same CUT and with the same BIST logic. The CUT has 16 scan chains and the size of the LFSR is 10 bits. One can see that the unweighted patter performs best.

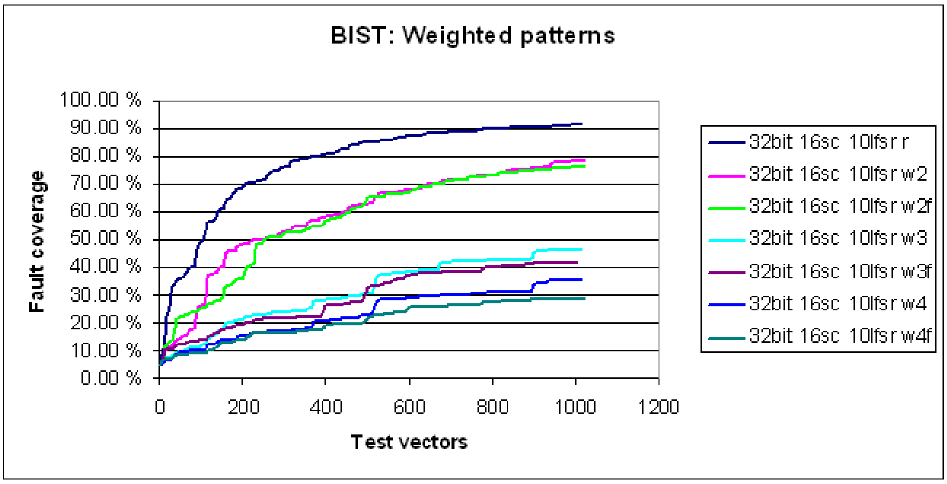

Figure 53: Simulation for different weights applied.

Figure 54 shows how the fault coverage increase with different weights when 1024 random patterns are applied before the weighted patterns are applied.

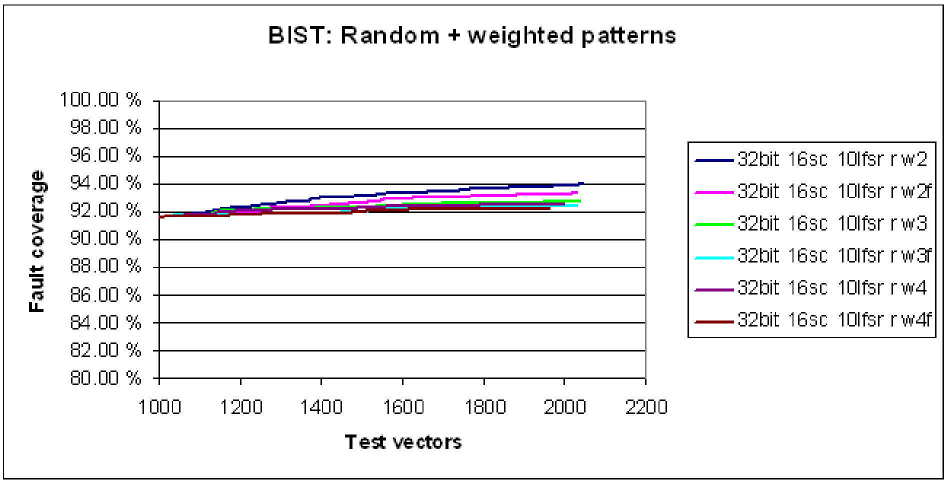

Figure 54: Simulation for different weights applied after random patterns.

The effect of a phase shifter is shown in figure 55. The same CUT is used and only the phase shifter is removed from the BIST logic. The simulations are run with 1024 patterns for all weights. Figure 56 shows a close-up of the random patterns. From the figures one can see that the effect of the phase shifter is very little.

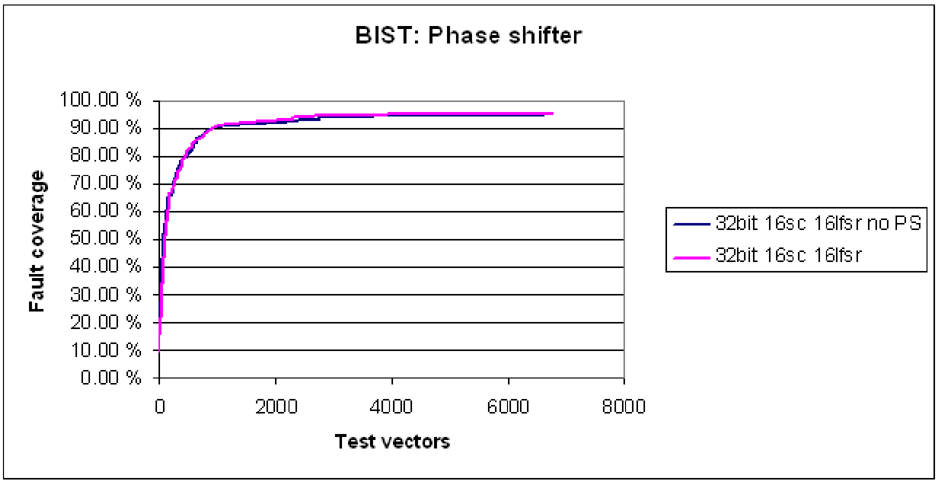

Figure 55: Simulation of BIST with phase shifter and without.

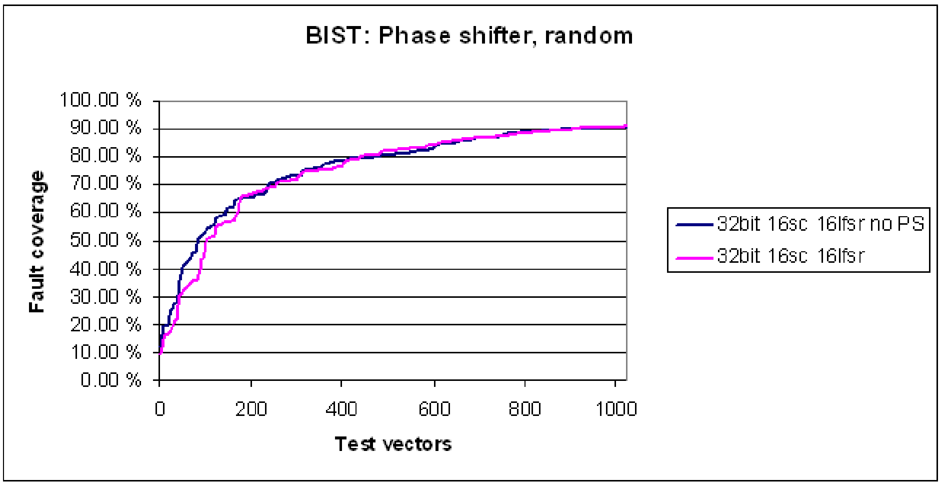

Figure 56: Simulation of BIST with phase shifter and without, random patterns.

Figure 57 shows the effect of adding primary input isolation to the circuit. In this case it has no effect on the fault coverage. This is probably because of the way the inputs are made in this design.

Figure 58 shows how each individual module of the CUT is affected by different settings.

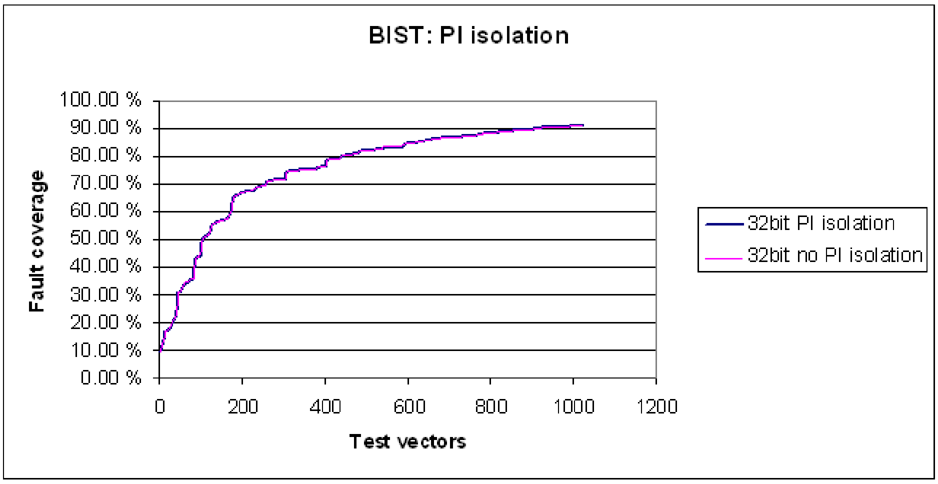

Figure 57: Simulation of BIST with PI isolation and without.

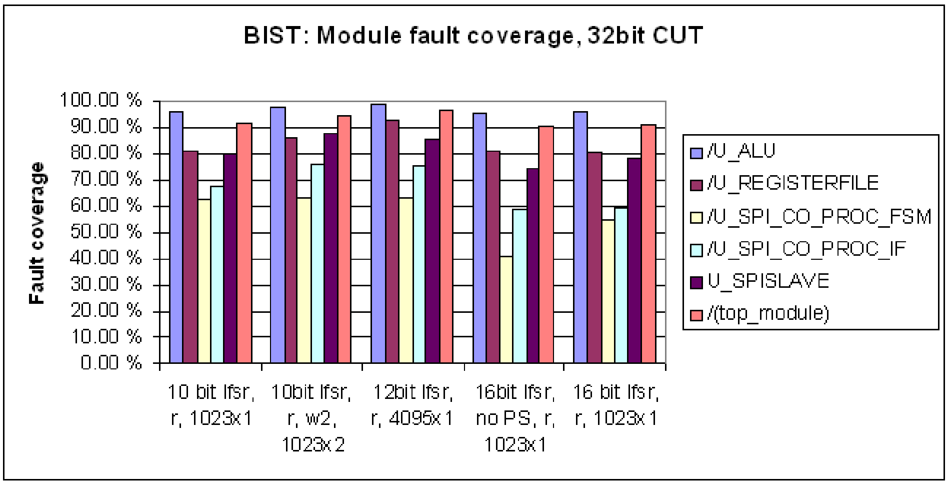

Figure 58: Fault coverage for different modules.

6.4 ATPG top up patterns

Figure 59 shows how the amount of test patters changes with different BIST settings. The first column is for pure ATPG test. This is for reference against the top up patterns. One can see that the amount of top up patterns decreases with increasing number of BIST patterns. For each simulation one additional weight is added to the BIST patterns. Each weight is simulated for 1024 patterns. The CUT size is 32bit and it has 16 scan chains. The LFSR size used is 10 bits.

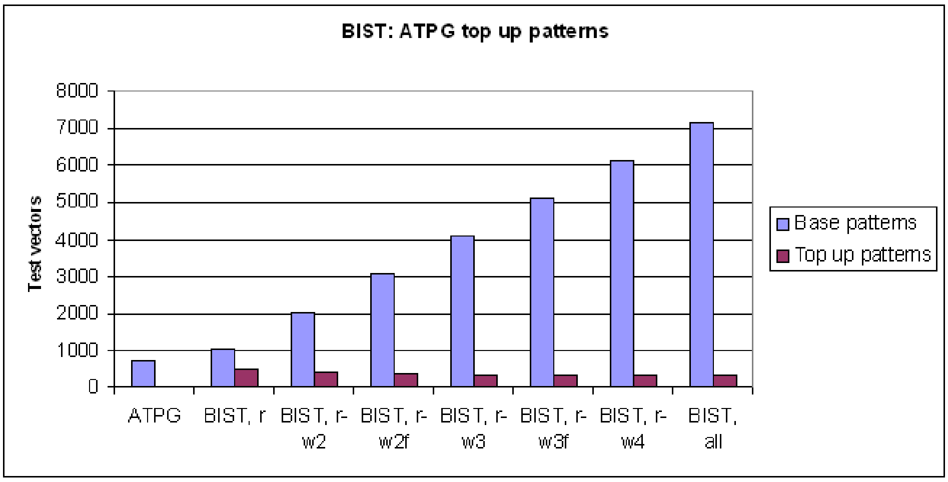

Figure 59: BIST test patterns and ATPG top up patterns.

Figure 60 shows how much time can be saved during test assuming that for external test there is only one scan chain available and that BIST can use 16 scan chains. The different BIST data series adds one additional weight each time in the same manner as in figure 59. The figure shows saved time in percentage versus the scan shift speed ratios between ATE scan and BIST scan speed.

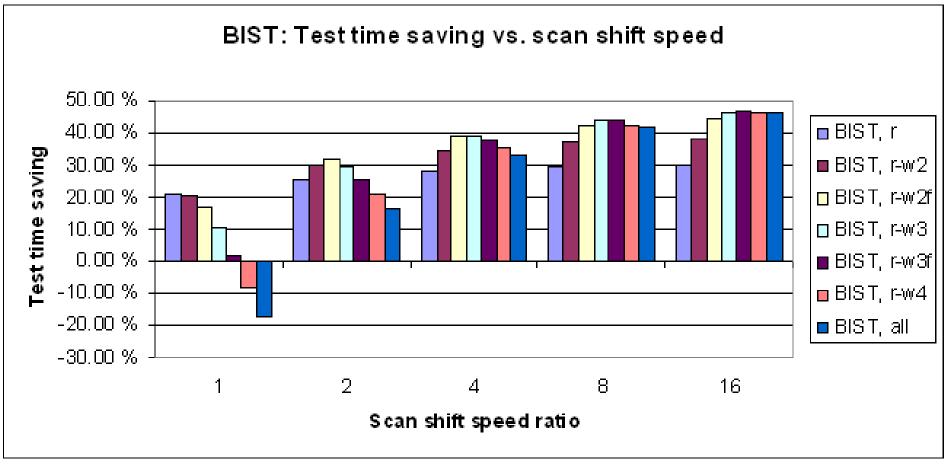

Figure 60: Test time saving for different scan speeds and number of BIST patterns.

6.5 Area

Figure 61 shows how the area of the BIST logic is affected by the size of the LFSR and MISR. One can see that the area for these modules increases but that they don't contribute significantly to the overall BIST logic.

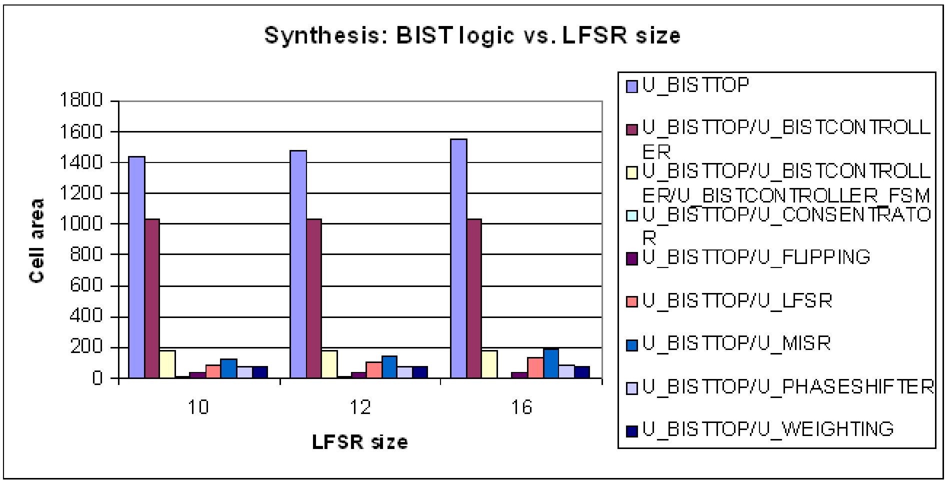

Figure 61: BIST area for different LFSR size.

Figure 62 shows how much the BIST area overhead is against different CUT sizes. One can clearly see that it decreases fast with the size of the CUT. The BIST logic used assumes one external scan chains, 16 internal scan chains, 10 bit LFSR and MISR, weighting logic and phase shifter. The CUT area used assumes 16 scan chains as part of the CUT area.

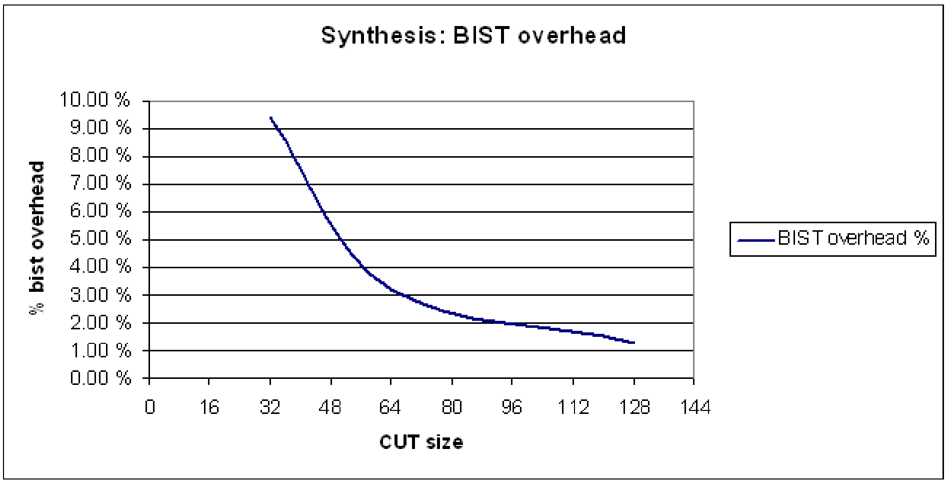

Figure 62: BIST area overhead.

Figure 63 shows how the BIST area is affected by the number of scan chains. In both cases 16 bit LFSR and MISR are used.

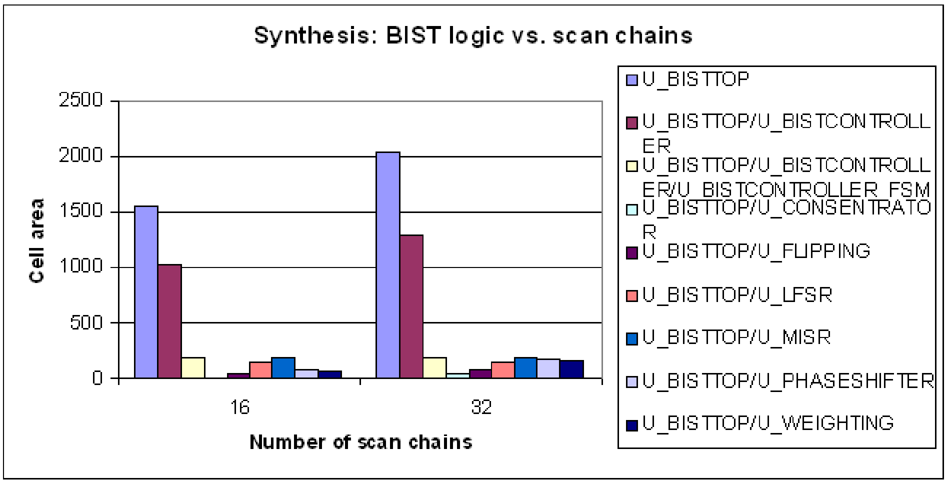

Figure 63: BIST area for different scan chains.

Figure 64 shows how the area is distributed between combinational and non-combinational logic in addition to show how much each module contribute. The results are shown for a 32bit CUT with 16 scan chains. Figure 65 shows the same for different CUT sizes, all with 16 scan chains.

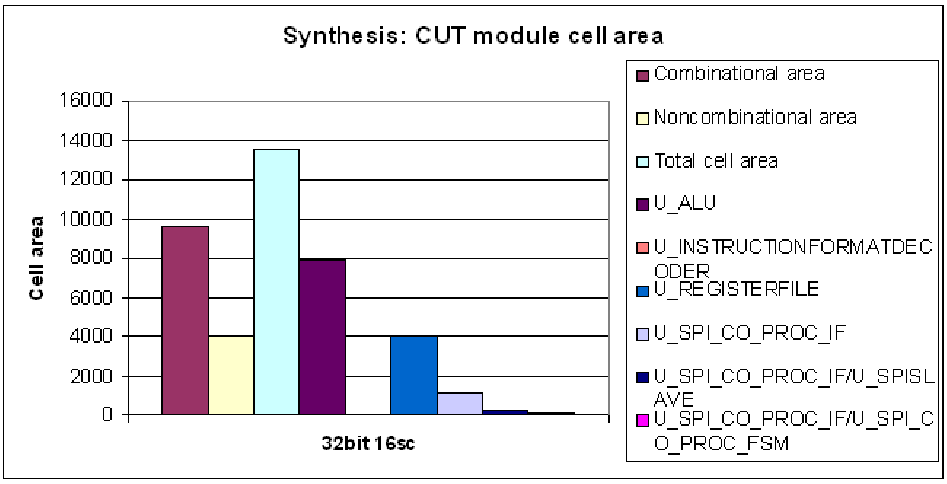

Figure 64: Area for different modules of the CUT.

Figure 66 shows how the total area, combinational area, and non-combinational area increases with CUT size.

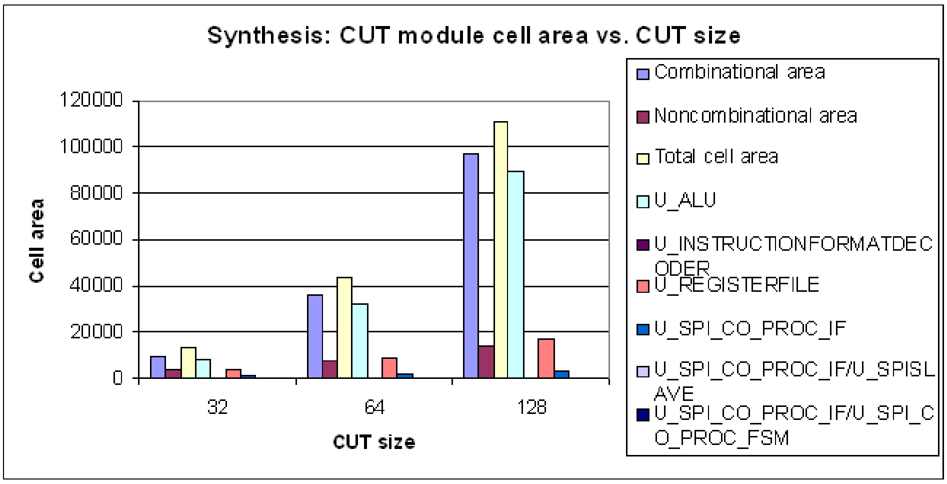

Figure 65: Area for different modules of the CUT for different CUT size.

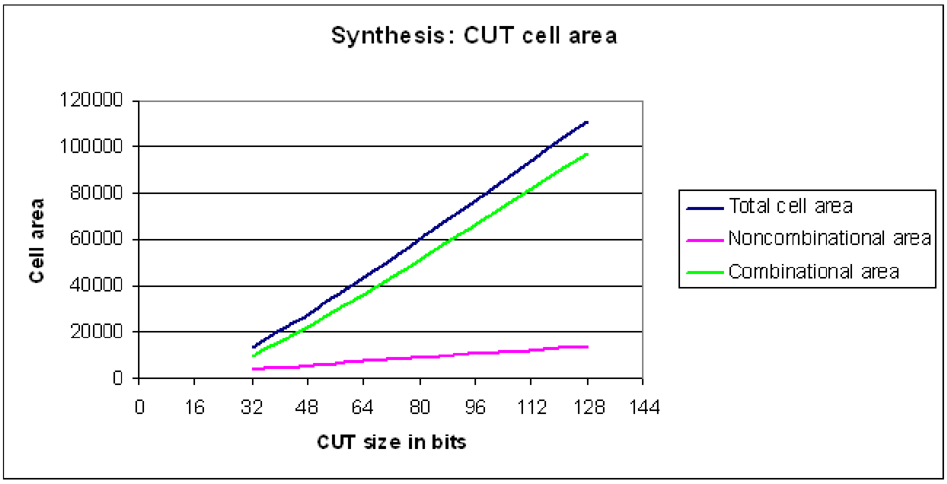

Figure 66: Area of the CUT for different CUT size.

6.6 Oscillator simulations

Figure 67 shows how the different oscillators are varying with voltage and temperature corners for the worst technology parameters. The typical temperature and voltage is set as baseline, and variations are calculated out from this. The typical case, TC, is a temperature of 25 Celsius and a supply voltage of 1,8V. The worst case, WC, is a temperature of 100 Celsius and a supply voltage of 1,6V. The best case, BC, is a supply voltage of 1,95V and a temperature of -40 Celsius. The PrimeTime data series is the one reported from the actual circuit path, and takes routing delay into account. Spice_Prim.path is the same path, but without the routing delay. The two last data series are the inverter oscillator with the two most extreme selections possible with the multiplexers.

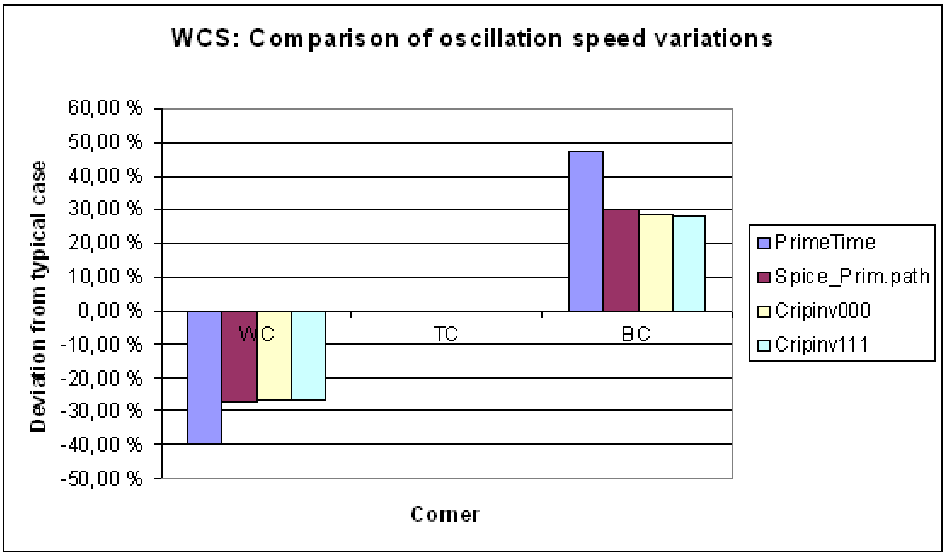

Figure 67: Inverter oscillator simulations.

7 Discussion

This section will take a closer look at the results presented in the previous section, and compare them against the theory. The circuit design will also be discussed. One of the main challenges was to obtain sufficient simulation results. The design contains several parameters that can be changed, and doing so will affect the obtained fault coverage. Due to time limitations it was not possible to investigate all aspects, so for this reason a base case was set and only a few parameters were changed at a time to see what effects they had. The base case selected was 32bit size for the CUT, a 10bit LFSR, and 16 internal scan chains.

7.1 Circuit design

The circuit design was divided into two parts, the CUT design and the BIST design. The reason for doing this was that it made the design task much easier as the CUT could be made first. Synthesizing the CUT with scan chains would be easy, and connecting it to the BIST logic would be straightforward. It would also be very easy to do regular ATPG and external scan testing on the CUT.

The BIST logic was made very generic so it could be used on several CUTs of different size. Several scripts were made for generation of different structures in the TPG and ORA to remove some code writing when trying out different combinations of parameters. The BIST controller contains several registers used to store configuration information. At reset these registers are reset to default values, and during development these default values can be changed and the circuit recompiled to use the new values. This is of course not possible after the circuit is made in silicon, so a method for changing these values is needed. This was obtained by connecting all of the configuration registers for the BIST logic and clock controller into a long scan chain. A shift enable signal was added to put these registers into shift mode so that new values can be shifted into the circuit. This scan chain can also be used to set seed values for the MISR and LFSR, and to retrieve the signature from the MISR. Another reason for doing this was that since nothing writes to these registers, some of them was optimized away during synthesis if they weren't connected into a shift register, but as this shift register now is added this is no longer a problem. If would probably be possible to write a synthesis script, which had connected these registers into a scan chain and added the scan enable signal, but this meant that the circuit had to be synthesized and then simulated to verify that putting this scan chain into shift mode did not destroy the operation of the circuit, by for instance changing the main clock of the circuit in an incorrect way. Making this scan chain manually meant that the different modules could be tested separately, and that all the shift registers could be connected at a higher level for final testing.

Another reason for making the BIST logic and CUT logic separately, and giving the BIST logic its own scan chain for configuration data, is that this removed all problems that could occur if the BIST logic was tightly integrated to the CUT. For instance, if the BIST logic was integrated into the CUT, additional care would have to be made so that the BIST parameters stored in registers couldn't be changed by the CUT during a test controlled by the BIST logic. Making the CUT and BIST logic separately also meant that the two parts could be synthesized separately, and the netlists could be connected in a top module. This made it easy to try out different combinations of BIST logic and CUT sizes, as only the top connection needs to be changed when analyzing different combinations of the circuit. This also reduced the time spent on synthesis since several BIST designs easily could be tested without the need for resynthesizing the entire circuit.

Although several problems were avoided by the selected approach, there are also arguments for doing this the other way around. If the BIST logic had been made with an interface against the CUT, the SPI co-processor, this would have made it much easier to change parameters in the BIST logic and clock controller logic as instructions could have been made that had written new parameters to these registers. It would also make it easy to start the BIST logic and retrieve the signature from the MISR, as instructions could have been made for this. It would also significantly reduce the amount of pads required for testing as most of the signals and data would have gone through the existing SPI interface. For a circuit that is to be manufactured on a large scale it is clear that these shortcuts can't be made, and it would be necessary to make the BIST logic with an interface towards for instance a TAP controller or other test interface. Since time was limited, and the main goal was to get a functional circuit and some measurements done, this was not done to save time. Other optimization could also have been done, such as using the already existing inputs and outputs, SDI and SDO, as input and outputs for the external scan chain. Doing so would reduce the amount of additional pads required, and help in reducing the area of the circuit. The reasons for not doing all kind of optimizations was to reduce the time spent on development, but also to save time used to learning how to use the software tools, as they are quite huge and have a lot of options for doing these kind of things.

7.2 DFT design issues

DFT design rules were followed during the design of the circuit. This is reflected in the stuck-at fault coverage, which is above 99% for ATPG. When transition fault tests were run the story was different. For ATPG it was above 99% but for BIST it was less than 10%. This was not expected. When the log files was inspected it was found that the ALU and register file had very low coverage, and since they contributes to the most of the faults in the circuit, the overall fault coverage was very low.

The reason for the low fault coverage was due to a single register in the design. The ALU executes the instruction stored in the instruction register. The instruction register defaults to the NOP instruction on each rising edge of the clock signal, and the only time it contains another instruction is when the SPI interface writes to it. For stuck-at testing this is not a problem since only one clock cycle is applied when the CUT is not in scan shift mode. This means that the instruction that is shifted into the instruction register is executed and the response from this execution is shifted out. In delay testing there is at least two clock cycles where the CUT is not in scan shift mode. In the first clock cycle the instruction that was shifted in is executed, and the instruction register resets to the NOP instruction if it is not written to by the SPI interface. Since the likelihood of the instruction register to be written is low when random patterns are used, this means that most of the delay test vectors will capture a NOP instruction, which will result in low fault coverage.

To fix this problem one additional register was placed between the instruction register and the SPI interface. This delays the instruction from the SPI interface by one clock cycle, and is for this reason to be considered as a performance penalty, as it is only placed there to increase testability. Now the instruction register always loads the data from this additional register, and the additional register is loaded with the NOP instruction when it is not written to. The result is that the ALU now executes two instructions during delay test, and both of these instructions are scanned in. This significantly increased the fault coverage for delay BIST, but it was still quite low. The reason for this was that 7 bits was reserved for the opcode, but they were not all used. Only 21 instructions were implemented, but 7 bits means that a total of 128 instructions can be implemented. The ALU considers all unimplemented opcodes as a NOP instruction and this means that the ALU still executes many NOP instructions during test. Since this has to do with the actual opcode, this design issue affects both delay faults and stuck-at faults. The problem was fixed by rearranging the opcodes so that masking could be performed on it. The result is that the ALU sees the opcode as 5 bits instead of 7 bits. This result is that the ALU now executes something else than the NOP instruction in 21 of 32 cases. The fault coverage is increase, but there is a slight performance penalty as the masking of the opcode adds delay. Figure 32, 33 and 34 shows how the fault coverage was affected by the fixes made to the design.

From this one can clearly see that good stuck-at coverage necessarily don't mean good delay coverage. One can also see that some small fixes can have significant affect on the fault coverage of the design, but that these fixes are likely to give some performance penalty as they are only used to increase fault coverage. The main problem seems to be case-statements with large amount of unused states, and registers that default to a given value and at the same time are unlikely to be written to.

Another issue to be aware of is that the reported fault coverage is for the CUT only, and the fault coverage of the BIST logic itself and clock controller logic are not considered. The reason for this is that the tools and setup used did not support analyzing the fault coverage of these modules. Still, one gets a good comparison of the CUT against BIST and ATPG.

7.3 LFSR size

From the simulations it is clear that the size of the LFSR will affect the fault coverage. Looking at figure 40 and 41 one can clearly see that the size of the LFSR will affect the stuck-at fault coverage. This is also true for transition faults, as seen from figure 49 and 50. For stuck at faults there is not a big difference. The 8 bit LFSR gives about 96% fault coverage, and the 10 bit and 12 bit LFSR gives about 98,5% fault coverage. In the case of transition faults the difference is greater. In this case the 8 bit LFSR gives only about 85% fault coverage when it is run for the maximum length. The 10 bit LFSR gives about 95% fault coverage, and the 12 bit LFSR gives slightly more. These results could be expected. First, the 8bit LFSR does not satisfy the requirements for the LFSR as described in the theory. The minimum size for an LFSR, according to equation 5 and equation 6, should be 9 bits for the CUT used in these cases. Also, an 8 bit LFSR will only generate 255 different test vectors, which is a low count of possible test vectors. It may also be to small to cover the greatest logic cones in the design. The 10 bit LFSR satisfies the requirements for the LFSR, and gives higher fault coverage. The total amount of test vectors is much higher than for the 8 bit design, but still small enough so that the BIST can be run for maximum length. Increasing the LFSR size further will increase the fault coverage, as seen for the 12 bit LFSR, but the increase is much less than compared to the increase from 8 to 10 bit LFSR. One of the reasons might be that the 12 bit LFSR design is not run for maximum length, as this is starting to be very long. Instead it is run for the same length as the 10 bit LFSR.

Another observation that can be done is that a small LFSR will go trough all possible combinations fast. This means that small logic cones will be exhaustively tested, and this might help in giving a fast increase in fault coverage. But since they are small and have few possible patterns, larger LFSR will pass and give higher fault coverage for larger number of patterns. From the simulations done, the different LFSR seems to have about the same increase in fault coverage in the start for random patterns. This might be because smaller LFSR is better in testing small logic cones, but that the larger LFSR is better in testing larger logic cones.

Since the ALU is made as a big case-statement, the size of the larges logic cone is unknown, but what is known is the size of the instruction register and opcode. Since the opcode is 7 bit, it is clear that the LFSR needs to be greater than this to make all possible opcodes. The instruction contains tree address fields in addition to the opcode. These fields are only 3 bits wide, and since this is less than the opcode it is also very likely that all possible combinations will exist here. The problem is that data also needs to be considered, and the data size is in excess of 32 bits. Having a LFSR of size larger than 32 bit will most likely make it larger than the larges logic cone, but it will still be impossible to run it for the maximum length, and for this reason a logic cone of this size cant be exhaustively tested in a reasonable amount of time.

7.4 Number of scan chains

The main reason for using several scan chains is to reduce the time spent on shifting test vectors in and shifting response out of the circuit. Considering stuck-at testing which only has one capture cycle, or delay testing which has one launch and one capture cycle, they only use one or two clock cycle to perform the actual test. Shifting of test vectors on the other hand uses as many clock cycles as the length of the longest scan chain. This means that a very little amount of the time used by test performs useful testing.

Looking at figure 35 one can see that for ATPG the number of scan chains has very little affect on how many test patterns are required and how fast the fault coverage increases versus the number of test vectors. This of course has to do with the fact that the ATPG tools can make test vectors that are optimised for the circuit, independent of how the scan chains are made. Still, using several scan chains will reduce the amount of time used by scanning test vectors and responses in and out, and for this reason several scan chains will normally be used. The main problem for external scan designs is that the number of scan chains may be limited by the number of pins available on the device for scan testing. This will set a limit on how short the scan chains can be, and finally how much time is spent on scanning data in and out.

Looking at figure 42 and 43, one can see that the number of scan chains has very little affect on the number of test vectors required to reach a given fault coverage for stuck-at BIST. This is also expected, since the TPG is basically the same in these cases.

Looking at figure 44 one can see that the case is the same for delay fault testing as for stuck-at testing when ATPG is used. Looking at figure 51 and 52 one can see that the same is the case for delay fault BIST also. From figure 52 one can see that the fault coverage increases faster in the case with 32 scan chains. One reason for this to happen might be that the phase shifter used in this case performs better than the one used in the design with 16 scan chains.

7.5 Weighted random patterns

Since some faults are hard to detect with random patterns, different weights was applied to see which weight gives the fastest increase in fault coverage. From figure 50 one can clearly see that random patterns are the most effective patterns to start with, and in this case they give over 90% fault coverage when the BIST logic is run for the maximum length. The next step was to find which weight should be used after the random patterns was applied. Simulations for random patterns followed by one of the weights were run, and from figure 50 one can see that w2, which has only 25% probability of a one and 75% probability of zero, is the most effective, giving an increase to the fault coverage of about 2%.

One of the reasons for this weight to perform well might be that the SPI interface has several small counters connected to the finite state machine. These counters are more likely to end up having a value triggering an event for patterns with large amount of zeros. At the same time, the test patterns still have some of the random characteristics, which result in several transitions, which again resulting in faster testing of the logic. For other weights with more zeros, the counters will have values resulting in transitions, but all other register values are now very likely to have a lot of zeros, making transition in the other logic unlikely. Since there are few transitions it is unlikely that faults will be detected, and the fault coverage increases very slowly.

One important observation is that applying weighted pattern will result in that the instruction register to the ALU in the design will be more likely to have some values than others. This again means that some opcodes are more likely than others, and that some addresses will be more likely than other. This will of course reduce the effectiveness of the weighted patterns, as not the entire circuit will be tested with weighted patterns, and by this reduce the increase in fault coverage. One way to avoid this is to place registers that should not have weighted patterns into a separate scan chain, and making the BIST controller able to set the weights for each individual scan chain. This will of course require a more advance BIST controller, and the BIST controller will increase in size. The choice of making such a BIST controller depends on whether the increase in fault coverage can defend the additional area cost of the BIST logic.

It is clear that the choice of weights to use is tightly connected to the logic, which is tested, and that this has to be traded against BIST area overhead for an optimal solution.

7.6 Phase shifter

Figure 55 and 56 shows the results for a design with phase shifter compared to one without. The difference is very small, and as seen from figure 55 the design with phase shifter has less than 1% more fault coverage. One possibility is that the phase shifter design used does not make enough shifts between the scan chains. Another possibility is that the CUT design does not have a large amount of defects that are hard to detect. This will result in that any benefits of the phase shifter is masked in the results. To investigate this further one can study the individual modules of the design. Figure 58 shows the fault coverage for the design for a number of different cases. Looking at the two last cases, 16 bit LFSR with and without phase shifter, one can see that the design with phase shifter has about 15% more fault coverage on the U_SPI_CO_PROC_FSM module, which is the finite state machine controlling the operation of the CUT. This indicates that phase shifters can have large impact on the fault coverage for some designs, and the reason for it to have an overall low impact on this design is that the finite state machine is small compared to the ALU and register file. Again, the choice of using a phase shifter or not depends on whether the increase in fault coverage can defend the increase in area overhead.

7.7 Primary input and output isolation

Figure 57 shows the difference in fault coverage when primary inputs are isolated and connected to the LFSR in test mode, and when they are not isolated, but only connected to a logic value in the testbench. As assumed there is no difference. The reason for this is that there is no logic between the input pins and the first register. Another reason is that all inputs to the CUT have at least two flip-flops in series as a shift register. The reason for this is that it is used to prevent meta stable data and to sample the SPI clock signal. Since there are at least two flip-flops in series transition fault testing is possible in the following logic, as two different values are present. The only paths that not are tested at-speed is the path between the input pin and the first flip-flop, and the path between the to series flip-flops at the input. Using input isolation would make it possible to test these paths, and the paths between the input isolation multiplexers and the first flip-flops, but still this would have left the path from the input pins to the isolation multiplexers. For this reason using input isolation on this design does not increase the fault coverage significantly, and the only reason for using it would be to make the circuit testable when the input pins can't be controlled by a tester. In cases where the inputs not are connected to flop-flops as in this case, input isolation would be required to test the logic between the input pins and the first registers. In a case like this it would also be important to make the BIST logic in a way that allows the TPG to run during the launch and capture cycles also, so that the inputs gets changing values, and by this obtaining delay testing of this logic.

The primary output of the CUT is a tri-state buffer. Since the ss_n_sync signal controls the tri-state buffer, this means that it can be in the Z state at any time during test due to the vectors shifted in. If it is in the Z state and at the same time is connected to the ORA, the ORA will interpret this as an X, and the signature will be destroyed. One way to prevent this is to add an additional control signal to the tri-state buffer, turning it always on during test. This would prevent X values going into the ORA. This is not done in this design. The reason for this is that there is no logic between the tri-state buffer and the register it takes its data from. This register is in one of the scan chains and is feed to the ORA. Adding an additional control signal will only add additional logic to the design, which also needs to be tested, and for this reason the output is not feed into the ORA. If there were logic between the last register and output a control signal would be necessary to control the tri-state buffer to always on and feed the output to the ORA, or the input to the tri-state buffer could be feed to the ORA to test this logic.

In some cases the output also needs to be isolated from the output pin. This would be necessary if the circuit is mounted on a card with other components connected to the output pin, and these components are tested at the same time assuming they don't have input isolation. One way to add this type of isolation would be to add boundary scan. This would add input and output isolation. It would also add a TAP controller to the circuit, which the BIST controller could be connected to.

7.8 CUT size

The size of the CUT is very important in the analysis of the fault coverage. Figure 36 and 37 shows how the circuit size affects the number of ATPG stuck-at patterns. One can see that there is an increase in the required number of test patterns, but that the increase is low compared to the increase in CUT size.

Figure 38 and 39 shows how fault coverage is affected by the circuit size when the same BIST logic is used. One can see that the fault coverage stays at about the same value for the different CUT sizes. For the 128bit case the fault coverage is a bit lower than for the first two cases. One of the reasons for this is that the CUT now have significantly more flip-flops than for the 32bit case, and the requirement for the size of LFSR is no longer satisfied. Increasing the size of the LFSR would probably make the 128bit case perform better than the two other cases as the ALU and registerfile get better tested and at the same time dominated the number of fault more than they did for the smaller CUT sizes.

Looking at the transition fault situation, the results are comparable to the stuck-at case. Figure 45 and 46 shows how the fault coverage and amount of test vectors are affected for the ATPG case. There is an increase in the number of test vectors, but the fault coverage stays about the same, as expected. Figure 47 and 48 shows the result for delay BIST. The situation here is the same as for the stuck-at fault case.

7.9 ATPG top up patterns

The main goal is to reduce the number of ATPG test vectors, and by this reducing the cost. Figure 59 shows how the number of ATPG test vectors is reduced for different BIST settings. One can clearly see that the number of ATPG top up patterns is reduced in the first few cases, but for BIST settings using more than 4000 test patterns the reduction in top up patterns is small. This is of course a reflection of that the increase in fault overage for the BIST solution is very slow after 4000 patterns are applied. Using a more advance BIST controller might help in further increasing the BIST fault coverage, and by this reducing the amount of top up vectors.

7.10 BIST area

The area of the BIST logic is of very high importance. The additional area adds cost to the overall design in form of silicon cost. This additional cost, and more, has to be saved by the reduction in test time, or else there will be no reason for using BIST in the first place. Figure 61 shows how the BIST logic is affected by the size of the LFSR and MISR, but for the same number of scan chains. One can see that the area for the LFSR and MISR increases, but that overall area does not increase very much. The reason for this is the BIST controller. It is very large compared to the TPG and ORA, but it does not change in size. Some optimization could have been done to reduce the BIST controller size, but since the controller is to be used to test different scenarios it was made generic enough to test the different scenarios without to much modifications. Some of the counters used in the BIST controller are quite large and can be reduced in size to more closely match the actual needed size.

Comparing the BIST area for different number of scan chains gives another result. Looking at figure 63 one can see that there is significant area difference be 16 and 32 scan chains. About half of the increase comes from the BIST controller, and about half of the increase from the increased size of the modules making up the TPG and ORA.

When comparing the size of the BIST logic against the CUT size, see figure 62, one can see that the area overhead decreases significantly with increasing size of the CUT. This can of course be explained by the fact the BIST controller makes up for most of the area in the BIST logic, but since this area stays almost constant, it will relatively take up less area as the size of the CUT increases.

Looking at the fault coverage obtained with the BIST logic and comparing it to the size of the modules in the TPG one can see that the area of the phase shifter and weighting logic are small compared to the overall

BIST area. They increase the fault coverage with several percents, and for this reason it is likely that they will contribute to reduce the overall cost of the design when they are used.

Looking at the figures showing the area of the different modules of the BIST logic one can see that the BIST controller contributes to about $\frac{2}{3}$ of the total BIST area. This is very large. The reason for this is the large amount of configurability added to the BIST controller. Removing this configurability and making a simple FSM that applies the most effective test vectors is likely to save a large amount of area, which will make the BIST solution more profitable to use as the area cost is reduced.

7.11 CUT area

Figure 64 shows the area usage of the different modules in the CUT. One can see that the area is dominated by the combinational logic of the ALU. The register file also makes significant contribution to the total area of the circuit. The instruction format decoder is invisible in the image. This is due to the fact that it just contains a few gates used to mask the opcode for the ALU. Looking at figure 65 one can see how the area is affected by the bit width of the CUT. Here one can see that the area of the CUT actually more than doubles each time the number of bits are doubled. One can also see that the ALU increases in size much faster than the other parts of the circuit. Looking at figure 66 one can see that the area increases linearly, but that the combinational area, which for the most part is the ALU, increases much faster than the non-combinational area. Since the combinational area increases much faster than the non-combinational area, this means that each bit stored in a register have to test more logic for the larger designs during test to obtain the same fault coverage.

7.12 Test cost

The test cost is modeled with test time as the main factor. The reason for using test time as the main factor and not the memory footprint of the test vectors is that the memory footprint can be translated into time assuming a constant time for each test vector when using an external tester. Also, if the memory footprint is so large that the tester requires reloading of test vectors, this will only be in favor of BIST, so not taking this into account is the same as a worst case analysis regarding the possible cost reduction when BIST is used.

Using equation 12 to calculate the saved test time when BIST is used for different ratios of scan speed between BIST and external testing equipment it is found that having a high ratio gives the most saved test time. This is expected as fast scan-shifting speed in BIST will significantly reduce the time used by the BIST logic for test. From figure 60 one can see that the amount of saved time increases fast when the scan shift speed ratio goes from 1 to 2 and from 2 to 4, but that it does not increase as fast when going from 4 to 8 and from 8 to 16. These calculations are based on the same test vectors, and since the time for the ATPG vectors stays constant and the time for the BIST vectors decreases fast, the equation is dominated by the ATPG test vectors when the ratio becomes large. This means that there is not much to save by using large ratios in this case. Using a ratio of 8 instead of 16 can have great impact on the power dissipation of the device while not making a great impact on the test cost.

Here the simulations are just done for a limited set of test vectors, but if high scan speed ratios are possible, a greater number of BIST test vectors can be used without using a large amount of time. Using a large amount of BIST test vectors can again reduce the number of ATPG top up vectors, which seems to dominate the time for high scan shift speed ratios. As seen from figure 60, at a ratio of 1 the case with the least BIST vectors gives the highest test time cost saving. As the scan shift speed ratio increases to 2 this is no longer the case. Now the case with two times as many BIST vectors is the one with the largest reduction. For the cases with ratios of 4, 8 and 16 one can see that each time the ratio increases, the BIST cases with larger amount of test vectors becomes more and more profitable to use.

Although only scan shift speeds have been simulated here, the same will be the case for the ratio of internal versus external scan chains, as internal scan chains will be shorter and therefore require less time to shift vectors in and responses out. There is one advantage of increasing the ratio of scan chains compared to increasing the shift speed. Increasing the number of scan chains while keeping the shift speed will not increase the power consumption of the device. For devices that have limited shift speed due to power dissipation issues this might be a solution that can be used to increase the benefits of using a BIST solution.

As seen from figure 60 it is possible to save about 40% test time when BIST is used. This can seem a lot, but one needs to consider the additional cost of the BIST area. If one for this design assumes that 2000 gates equals the cost of 1 second testing time one can see that the use of BIST actually increases the cost due to the large area used by the BIST logic. If BIST should be profitable, assuming the same vectors were generated, the area needs to be five times smaller. This is not possible as this is smaller than the size of the TPG and ORA used to generate the vectors [4].

7.13 Fault coverage

As mentioned, the main reason for using a BIST solution is to reduce the overall cost of the device. This is obtained by reducing the cost of test by

[4] The spreadsheet CostReduction_rev2.xls was used for these calculations.

using additional circuit area to make BIST logic. There are several aspects to consider when making the BIST logic. Should BIST be the only way to perform testing, or should other methods also be available? If BIST is the only method for testing this will set strict requirements for the BIST logic. For instance, if BIST is the only testing method, this means that the BIST solution have to yield high fault coverage. If the circuit has other methods for testing in addition to BIST, the BIST solution does not have to give 100% fault coverage. This eases the requirements a lot, as the fault not detected by BIST can be detected by other means. Since delay faults are harder to detect than stuck-at faults, it is more likely that the BIST solution wont be able to get as high fault coverage as required, and for this reason an approach using STUMPS is a good choice.

In the case of this design, the ALU and registerfile are both quite large, and contributes to most of the faults in the fault list of the device. This also means that the fault coverage obtained on these two modules will dominate the overall fault coverage, as seen from figure 58. The fault coverage of these two modules is in the range of 80-95% for random patterns, which is very high. The other modules have fault coverage in the range 60-80% for random patterns, and this is more similar to what is found in other papers with similar BIST design. This might indicate that the large combinational area of the ALU, and the fact that this dominates the total fault coverage together with the registerfile are not well suited to model what will happen in a more advance design. Looking at the figure, one can see that the SPI interface, with its sub modules, have results that are more comparable to what is found in other research. Changing the ALU and registerfile to some other logic might give results that are more comparable to other results. On the other hand, one should have in mind that the HDL code was modified to give higher fault coverage for the ALU and registerfile, this was not done for the other modules, and this is also one of the reasons for these modules to have lower fault coverage. Looking at the log files from the ATPG simulations one can see that the fault coverage for the finite state machine is only 77% for transition faults, but is 100% for stuck-at faults. From the experience with the instruction register for the ALU, it is likely that this module also contains similar logic, which resets at certain points preventing delay fault testing in achieving high fault coverage. In this case it is not possible to fix this as it was done for the ALU. The reason for this is that this logic requires strict timing in the number of clock cycles from one event to another, and it is therefore impossible to add additional registers, as this will change the timing.

One thing to consider is the fact that by using BIST one are able to apply a huge amount of test vectors. These test vectors might detect unmodeled faults, and by this reduce the amount of defective circuits that escapes testing. This is of course hard to predict, but should be considered as a benefit of using a BIST solution.

7.14 Oscillator design

From figure 67 one can see that the variation due to temperature and voltage is much larger for the reports from PrimeTime compared to the simulations done in spice which don't include routing delay. Simulations show that the spice model has a delay of approximately 11ns from input to output. PrimeTime reports in excess of 20ns, which means that about 50% of the delay is due to routing. This can be a problem. The delay in the critical path is made up of gate delay and routing delay. The routing delay is affected by total capacitance of the net, which again is a function of routing between the gates and the amount of gate-inputs a gate output has to drive. When making an oscillator with the same gates as in the critical path, the routing will still be different. Because the routing contributes so much to the delay it is very likely that the oscillator won't oscillate with the same speed as the actual critical path. One can hope that routing on average will be the same, and that this will make the oscillator oscillate at the correct frequency, but since the oscillator gates won't have the same fanout, this is very unlikely.

Looking at figure 67 one can also see that the spice simulations have about the same amount of variation in the different cases. This was also true for different technology corners. This means that the oscillator made of inverters is likely to behave the same way as the one made of other gates.

Since there is little variation between the inverter oscillator and the one using gates from the actual critical path, there is no reason for not using only inverters in the oscillator as long one stays within the voltage and temperature parameters for the library used. This is because the supply voltage in these cases is much larger than the threshold voltage for the transistors. If the supply voltage is scaled down more than this, as might be done for very low power devices, the supply voltage will become close to the threshold voltage of the gates and inverters. Since different gates have different threshold voltage due to stacked transistors, this can result in greater difference in delay between inverters and other gates, and this might result in that inverters can't be used for this purpose. More investigation of this will be necessary for very low power devices.

This leaves the routing problem. One can see that the variation is almost the same in any direction from the typical case in the figure. This means that the design of the oscillator can be done in the typical case when one adds the same margin in both directions. The question now becomes how much margin is needed. If the inverter oscillator is made to oscillate at the critical frequency in the typical voltage and temperature corner using the center tap of the oscillator calibration, this will give about the same margin in both directions. This can then be used to scale for small variation in temperature difference, and for large variation in routing delay. Making an oscillator this way will make it possible to do delay testing at the actual critical frequency of the circuit when the oscillator is calibrated to match difference in routing and temperature. This can possibly be done by the ATE if frequency measurement of the oscillator is possible. If one selects the tap that makes the oscillator oscillate at the slowest frequency and measure this frequency, one can use this value to look up the correct calibration value to use. This assumes that the oscillator scales equally to the critical path. The stored values that are looked up will in this case need to compensate for difference in routing delay between the oscillator and the critical path. These values can for instance be extracted from PrimeTime when the design is finished. If this is applied to a design it can also be used for speed binning of the circuits.

7.15 Future work and suggestions for improvement

From what is found it is clear that BIST can significantly reduce test time, but that this necessarily don't mean that cost will be reduced. Reducing cost and increasing profit is always the main goal, and by using the appropriate BIST solution this should be possible. The CUT design used here seems to be very random testable, but the few random hard to detect fault also seems to be very hard to detect, and result in a high number of ATPG top up vectors. One can also see that delay fault testing is much harder than stuck-at testing. To overcome these obstacles it is likely that Deterministic BIST is the only solution that will yield high fault coverage and at the same time keep BIST area to a minimum. Future work will be to make a new BIST solution that uses bit flipping or other methods to make deterministic test vectors in addition to random vectors. This BIST solution can then be tested on the first CUT design without the modifications for increased fault coverage and the one with modifications. One will then be able to investigate whether it is possible to make a BIST solution that can get sufficient fault coverage without modifying the CUT, and see if this BIST solution will have a area overhead that can be defended by the reduction in test time.

8 Conclusion

A circuit for delay testing with BIST logic have been developed. The design was made in two separate parts, the CUT and the BIST logic, as this made the work of analysing the circuit much easier. The main drawback of this solution is that it requires several external test pads, which are only used for testing purpose. Changing configuration of the clock controller or BIST controller is also more complex as all configuration registers are connected into a long shift register, requiring the entire configuration to be shifted in each time something is to be changed. In a BIST design attended for mass production, the BIST controller should be made with an interface against a TAP controller or similar for easy access to the configuration registers.

From the analysis if was found that some design issues will affect delay fault coverage, whereas stuck-at fault coverage is unaffected. Registers that reset to default values at each clock cycles, if they are not written to, and case statements with unused states will significantly reduce the fault coverage for delay testing. Sometimes these issues can be fixed in the RTL code, but this will probably give a performance penalty. In other cases it might not be possible to fix the code, and the fault coverage will for this reason be low. Low BIST fault coverage will result in a large amount of top up test vectors and reduce the effectiveness of the BIST solution. To overcome this, deterministic BIST is possibly a good candidate. Due to time limitations deterministic BIST have not been investigated here. Future work should try to analyse deterministic BIST against the solution used here to see if it can further reduce test time and cost.

The analysis have shown that effectiveness of BIST can greatly be increased by being able to do shifting of test vectors and responses faster than what is possible with external testing equipment. It has also shown that it doesn't have to be done at-speed, as top up patterns might dominate the time for high BIST scan speeds. Increasing the number of internal scan chains will have the same effect. Using more internal scan chains is mainly done to compensate for the fact that BIST requires far more test vectors than ATPG vectors.

The results show that it is possible to reduce test time with over 40%. This is not simulated with consideration to power dissipation of the device, and this needs to be considered when doing test on silicon. In addition it was found that the area of the BIST logic is too large in this case. It is therefore unlikely that the overall cost of this design will be reduced with the BIST solution used here. This of course don't mean that this will be the case always, but it is clear that a very configurable BIST controller will require a large amount of area, and for this reason might not be profitable.

References

C.P.; Tehranipoor M.; Plusquellic J. Ahmed, N.; Ravikumar. At-speed transition fault testing with low speed scan enable. *VLSI Test Symposium, 2005. Proceedings. 23rd IEEE*, pages 42–47, 1-5 May 2005. ISSN 1093-0167. doi: 10.1109/VTS.2005.31.

J. Mucha B. Koenemann and G. Zwiehoff. Built-in logic block observation technique. *Proc. IEEE International Test Conf.*, pages 37–41, 1979.

Alfred L. Crouch. *Design-For-Test For Digital IC's and Embedded Core Systems.* Prentice Hall PTR, 1999.

D. Das and N.A. Touba. Reducing test data volume using external/lbist hybrid test patterns. *Test Conference, 2000. Proceedings. International*, pages 115–122, 2000. doi: 10.1109/TEST.2000.894198.

V. Gherman, H.-J. Wunderlich, J. Schloeffel, and M. Garbers. Deterministic logic bist for transition fault testing. *Computers & Digital Techniques, IET*, 1(3):180–186, May 2007. ISSN 1751-861X.

Øystein Gjermundnes. *Exploiting Arithmetic Built-In Self-Test Techniques for Path Delay Fault Testing.* PhD thesis, Norwegian University of Science and Technology, Faculty of Information Technology, Mathematics and Electrical Engineering, 2006.

David A. Johns and Ken Martin. *Analog Integrated Circuit Design.* John Wiley and Sons, Inc, New York, 1997.

M.P. Kusko, B.J. Robbins, T.J. Koprowski, and W.V. Huott. 99% ac test coverage using only lbist on the 1 ghz ibm s/390 zseries 900 microprocessor. *Test Conference, 2001. Proceedings. International*, pages 586–592, 2001. doi: 10.1109/TEST.2001.966677.

T. E. Leistad. Verification of fault models for low-power cmos devices. Project assignment, Norwegian University of Science and Technology, Faculty of Information Technology, Mathematics and Electrical Engineering, 2007.

S. Pateras. Achieving at-speed structural test. *Design & Test of Computers, IEEE*, 20(5):26–33, Sept.-Oct. 2003. ISSN 0740-7475. doi: 10.1109/MDT.2003.1232253.

J. Rajski and J. Tyszer. Design of phase shifters for bist applications. *VLSI Test Symposium, 1998. Proceedings. 16th IEEE*, pages 218–224, 26-30 Apr 1998. doi: 10.1109/VTEST.1998.670871.

K.M.; Gatt J.; Raghuraman R.; Kumar S.P.; Basu S.; Campbell D.J.; Berech J. Saxena, J.; Butler. Scan-based transition fault testing - implementation and low cost test challenges. *Test Conference, 2002. Proceedings. International*, pages 1120–1129, 2002. ISSN 1089-3539. doi: 10.1109/TEST.2002.1041869.

C. F. Segura, J. Hawkins. How it works, how it fails. *CMOS Electronics*, pages 153–239.

Charles E. Stroud. *A Designer's Guide to Built-in Self-Test.* Springer, 2002.

Sudheer Vemula. Scan based delay testing. Technical report, Auburn University, Dept. of Electrical and Computer Engineering.

H.-J. Wunderlich and G. Kiefer. Bit-flipping bist. *Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on*, pages 337–343, 10-14 Nov 1996. doi: 10.1109/ICCAD.1996.569803.

J.; Reuter P.; Rinderknecht T.; Swanson B.; Tamarapalli N. Xijiiang Lin; Press, R.; Rajski. High-frequency, at-speed scan testing. *Design & Test of Computers, IEEE*, 20(5):17–25, Sept.-Oct. 2003. ISSN 0740-7475. doi: 10.1109/MDT.2003.1232252.

Appendices

A Instruction set summary

Table 2 shows the instruction set summary for the co-processor.

B FPGA connection to STK500

Table 3 shows how to connect the Altera DE1 FPGA development kit to connection to STK500 from Atmel for verification of the operation of the co-processor. Connect PORTD to LED on SDK500 for LED output debugging information

C Changes in verilog code for different simulations

Although a scan chain was made for changing parameters in the BIST logic and clock controller this was not used. Instead the parameters was changed in the verilog parameter files as the code had to be recompiled anyway. This chapter describes the necessary changes to code when changing some of the parameters.

In clockcontroller_params.v one need to change the configuration when changing between stuck-at testing and delay testing.

Bisttop.v needs updating when changing the size of the LFSR, MISR, any other part of the TPA and ORA, or the number of scan chains. Since the code is written in emacs with verilog mode this is a simple task, as one only needs to update modules that are instantiated with autoinst, so a simple refresh is only required.

Bistcontroller_params.v have parameters that need to be modified for different tests. These parameters control the number of patterns and the number of shifts to perform.

Chip_params.v is the main parameter file and are used with the code generator scripts. Changes are need for generation of new TPG and ORA. This file is also used for the parameter check script, which calculates the optimal number of shifts for BIST and checks the requirements for LFSR size.

Testbench.v needs to be modified when a new CUT is tested. Testing a new CUT will change scan chain length and possibly the amount of scan chains. This affects the functions writing test patterns to different files, and this needs to be updated. Modifications to the PO vector might also be necessary. A parameter stating whether stuck-at or transition fault testing

Mnemonic	Operands	Description	Operation	Flag
NOP		No operatin		
WRITE		Write data from SPI to register file	Rd = SPI DATA	
READ	Ra	Read data from register file	SPI DATA = Ra	
WRITESREG		Write data from SPI to status register	SREG = SPI DATA	
READSREG		Read data from statur register	SPI DATA = SREG	
MOV	Ra	Move data between registers	Rd = Ra	
ADD	Ra, Rb	Addition without carry	Rd = Ra + Rb	C
ADC	Ra, Rb	Addition with carry	Rd = Ra + Rb + C	C
SUB	Ra, Rb	Subtraction without carry	Rd = Ra - Rb	C
SBC	Ra, Rb	Subtraction with carry	Rd = Ra - Rb - C	C
MUL	Ra, Rb	Multiplication	Rd = Ra * Rb	C
MAC	Ra, Rb, Rd	Multiply and accumulate	Rd = (Ra * Rb) + Rd	C
LSL	Ra	Logic shift left	Rd = Ra(6:0):0	C
LSR	Ra	Logic shift right	Rd = 0:Ra(7:1)	C
ROL	Ra	Rotate left	Rd = Ra(6:0):C	C
ROR	Ra	Rotate right	Rd = C:Ra(7:1)	C
AND	Ra, Rb	Logic bitwise AND	Rd = Ra AND Rb	
OR	Ra, Rb	Logic bitwise OR	Rd = Ra OR Rb	
XOR	Ra, Rb	Logic bitwise XOR	Rd = Ra XOR Rb	
INV	Ra	Logic bitwise inversion	Rd = !Ra	

Table 2: Instruction set summary.

are performed must also be updated when changing between these two as this affects the generation of test patterns.

Signal name	STK500 with ATmega8515L	ATERA DE1	Hpin
ss_n	PB4	GPIO_1 pin 35 input	40
MOSI/sdi	PB5	GPIO_1 pin 34 input	39
MISO/sdo	PB6	GPIO_0 pin 32 output	37
sclk	PB7	GPIO_1 pin 33 input	38
GND	GND	GND GPIO_1-con	30

Table 3: FPGA connections.

D Attached files

This report is delivered with an attached zip file containing the source code, simulation and synthesis scripts, log files, and spreadsheet documents used for analysis of the results. This section describes which files can be found in the different directories.

The Zip-file's root folder contains two subfolders, chip and results. The chip folder contains the source code and log files, and the result folder contains the spreadsheets, CSV-files and the scripts used to search through the log files. The CSV-files and scripts used for the generation of the CSV-files are located under \results\scripts.

The chip folder has the subfolder source where all the files are stored. Under \chip\source\generator one can find the scripts used to generate the HDL source code for the TPG and ORA, and the parameter checking script. Running generate.sh form a shell runs all these scripts. The different generator scripts are located in subfolders under \chip\source\generator\python.

The folder \chip\source\rtl is the main folder for the HDL code of the circuit. All HDL code is located in the subfolder module. This folder contains several subfolders, each one for a different module from the design. Each module folder has several subfolders such as custom, import, include and verilog. This is done to comply with Atmel guidelines. Except for the chip module folder, the HDL code is located in \custom\verilog and parameter files in \custom\include.

The folder \chip\source\synth contains the synthesis files. The alteraDE1 folder contains the entire FPGA project, and the files CII_Starter_TOP.sof and CII_Starter_TOP.pof can be used to program the FPGA using Altera software. The folder chip_atmel contains the synthesis scripts and saved synthesis files for the synthesis of BIST logic and clock controller. The folder dut_atmel contains the synthesis scripts for the CUT and saved synthesis files. The log files from synthesis are not included as these contain confidential information regarding the process technology used.

The folder \chip\source\test contains two subfolders, dut_atpg and dut_bist. The folder dut_atpg contains scripts and result for ATPG simulations done with Tetramax. The folder dut_bist contains results and scripts used to do fault simulation on BIST patterns with Tetramax.

The folder \chip\source\validation contains the C program developed to run on an AVR device for communication with the FPGA kit and the developed co-processor for verification of its operation.

The folder \chip\source\verification has several folders containing testbenches for the different modules, and two additional folders for different simulators used, modelsim and vcs. The folders modelsim and vcs have subfolders for the different simulations done and these folders contains scripts for compiling source code, simulation and for viewing the waveforms from simulations. The testbench used to generate the BIST test patterns is located under \chip\source\verification\vcs\chip_top. The generated patterns are stored in the subfolder genpat when the testbench is simulated.