A novel approach to Reusable Time-economized STIL based pattern development

by

Rahul Malhotra

Under the Supervision of
Dr. Sujay Deb
Fabio Carlucci

Indraprastha Institute of Information Technology Delhi
June, 2015

A novel approach to Reusable Time-economized STIL based pattern development

by

Rahul Malhotra

Submitted
in partial fulfillment of the requirements for the degree of
Master of Technology in Electronics & Communication
Engineering with specialization in VLSI & Embedded Systems

to

Indraprastha Institute of Information Technology Delhi
June, 2015

Certificate

This is to certify that the thesis titled **A novel approach to Reusable Time-economized STIL based pattern development** being submitted by Rahul Malhotra to the Indraprastha Institute of Information Technology Delhi, for the award of the Master of Technology, is an original research work carried out by him under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

June, 2015

Dr. Sujay Deb
Department of _____
Indraprastha Institute of Information Technology Delhi
New Delhi 110 020

Abstract

State of the art automotive micro-controllers (MCUs) implementing complex system-on-chip (SoC) architectures requires often additional functional patterns to achieve high degree of reliability. Functional pattern family includes test patterns checking internal device functionality under nominal condition. The development of these patterns is required to augment structural tests to achieve high test coverage. Moreover, it should be planned to minimize test time since it has direct impact on time to market. This thesis proposes a reusable and time-economized approach for functional test pattern development aimed for production tests using Test Information Model (TIM) discussed. The flow is also automated to exclude the potential source of error in the pattern generation. In applying this methodology, there is approximately 60%-70% reduction in pattern time as compared to conventional simulation based pattern development which required cyclization of simulation data to be compatible to testers used for production tests. Moreover, proposed automated production test pattern development activity is smooth as compared to manual approach and its availability is also aligned with the first verified RTL. Also, the results show the efficiency of this approach in terms of pattern re-usability by further reducing man-hours of the test engineer.

Acknowledgements

I am most grateful to my thesis guide, *Dr. Sujay Deb*, for his guidance and support throughout the time of my research. He constantly showered his rich experience and knowledge contributing to the acceptance of my work in one of the top conference.

I am thankful to *Fabio Carlucci*, project manager from ST Microelectronics for having me in his SoC team and acquainting me with diverse DFT concepts that ultimately were the stepping stones of the final outcome. Furthermore, I am obliged to *Ajay Kr. Dimri* from same team for keeping me motivated from the very beginning of blossoming of ideas to the actual implementation of them.

To the ever-close friends, *Ramandeep Kaur* and *Abhishek Jain*, from the same batch to string along on occasions of hangouts countless times during up and down moments of our togetherness.

Finally, I would like to acknowledge people who mean world to me, my parents, for their love and support throughout my life.

Contents

List of Figures

List of Tables

List of Abbreviations

ADC Analog to Digital Convertor. 2

ATE Automatic Test Equipment. 4, 5, 23, 25, 26, 29, 31, 32, 37

ATPG Automatic Test Pattern Generation. 14, 26, 29, 39–41, 43

BIST Built in Self Test. 2, 9, 14, 16–18

BSR Boundary Scan Register. 11, 13

CAD Computer Aided Design. 5

CoT Cost of Test. 29

CPU Central Processing Unit. 16

CUT Circuit Under Test. 3, 16

DAC Digital to Analog Convertor. 2

DFT Design For Testability. 2, 5, 8, 9

DL Defect Level. 7

DUT Device Under Test. 15, 23, 25, 26, 35, 37, 39

EDA Electronic Design Automation. 22

FC Fault Coverage. 6, 7

FMA Failure Mode Analysis. 4

GPIO General Purpose Input Output. 18

HDL Hardware Description Language. 35

IC Integrated Circuit. 21–23

IP Intellectual Property. 1, 2, 16, 18, 31, 33, 37, 39, 40, 42, 43

JTAG Joint Test Action Group. 11–14, 18, 33, 41

MUX Multiplexer. 8

OTP One Time Programmable. 33

PCB Printed Circuit Board. 11

PI Primary Input. 15

PLI Programming Language Interface. 29, 39

PLL Phase Locked Loop. 2, 18, 36

PO Primary Output. 15

RTL Register Transfer Language. 35, 37, 40, 43

SoC System on Chip. 2, 31–33, 37, 39

STIL Standard Test Interface Language. 1, 2, 14, 22, 23, 26, 27, 29, 31–33, 35, 37, 39, 40, 42–44

TAP Test Access Port. 12, 13, 33

TCK Test Clock. 12

TDI Test Data In. 12

TDL Test Description Language. 22

TDR Test Data Register. 33, 35

TIM Test Information Model. 32, 37, 40

TMS Test Mode Select. 12, 13

TRST Test Reset. 12

TTM Time to Market. 29

UUT Unit Under Test. 11

VCD Value Change Dump. 14, 22, 23, 25, 26, 29, 32, 39, 43

VLSI Very Large Scale Integration. 4, 5

WGL Waveform Generation Language. 14, 22

1

Introduction

1.1 Motivation

The development cycle time is continually decreasing to meet very aggressive time-to-market requirements, while the product quality is expected to reach levels currently only known for military or aerospace products [1]. All this has to be achieved while minimizing the total cost for a device, where test costs are contributing an increasing portion. Also, there is a necessity to meet schedules with less engineering staff and also to increase reuse in designs, code and tools. A crucial aspect in this area is the generation of test patterns which are used for different objectives including qualification, burn-in, and production test. The outcome of these tests plays a critical role in deeming the systems readiness for production.

Functional pattern family includes test patterns checking internal device functionality under nominal condition. The development of these patterns is required to augment structural tests to achieve high test coverage. Moreover, it should be planned to minimize test time since it has direct impact on time to market. Historically, these patterns were handcrafted or derived from simulations with little communication between design and test/product engineering[1]. Moreover, the traditional approach had its own drawbacks such as long test to time and others listed in Chapter 3. Standard Test Interface Language (STIL) based pattern generation was introduced to remove pattern generation dependency on simulation[2]. However, writing STIL pattern manually can be time consuming and can introduce potential source of errors.

1.2 Problem Statement

To overcome the above said limitations, this thesis proposes an automated flow that aims to generate these functional patterns in an industry accepted standard STIL (IEEE Std 1450). The flow also deals with re-usability of the test case for an Intellectual Property (IP) in different products. The aim is to reduce the test effort for same test in different verification environment. Test patterns are de- veloped for IPs such as Phase Locked Loop (PLL), Analog to Digital Convertor (ADC), Digital to Analog Convertor (DAC), Thermal Sensors etc. to verify their intended functionality by validating the test cases in system level environment.

1.3 Organization of Thesis

The chapter organisation of the thesis is in the following manner. Chapter 2 deals with the elemental introduction to **System on Chip (SoC)** testing, testability measures and how to improve them. Then the chapter advances to **Design For Testability (DFT)** concepts, **DFT** approaches namely **Ad-Hoc DFT** and **Structured DFT**. Peculiar Structured DFT concepts for instance **Scan Design**, **Built in Self Test (BIST)** and **Boundary Scan** are touched upon. It also focusses on types of system level testing such as using Automatic Test Pattern Generation (ATPG) scan patterns, Memory BIST and standalone test modes of analog macros. Then the organisation moves to Chapter 3 dealing with the conventional approach to test pattern development using Value Change Dump (VCD) file generated from simulation environment and conversion of this file to cyclic format i.e. STIL. Also, the shift from simulation based pattern generation to writing test patterns directly in cyclic format is touched upon. Chapter 4 elaborates the proposed approach to pattern development which serve to depreciate the test pattern development time significantly. Chapter 5 discusses the results comparing the test development time on application of the proposed approach and the conventional approach to different products. It also focusses on the comparison of automated approach and the manual approach to STIL based pattern development in terms of effort involved in generating different blocks of a test pattern. Finally, chapter 6 concludes the thesis and discusses the future work.

2

Theoretical Concepts

This section deals with some of the basic SoC testing concepts, how to improve testability i.e controllability and observability by inserting appropriate circuitry in the design broadly known as DFT. The test patterns for different blocks of an SoC such as ATPG scan patterns are used for digital modules, memory BIST is used for testing of memories and standalone test mode is used for testing the functionality of analog or mixed signal macros such as PLL, ADC, DAC. Test mode access for these standalone modes is through Test Access Port (TAP) signals at top level directly controllable using Joint Test Action Group (JTAG). The functional testing of such macros after integration is a crucial aspect to achieve utmost reliability. For this, a novel approach is proposed covered in subsequent chapters. This approach can also be used for running BIST on memories present in the design.

2.1 SoC Testing

Testing determines whether or not a circuit under test (Circuit Under Test (CUT)) operates correctly by comparing known output to the CUT response. The correctness and effectiveness of testing is most important for quality products. Quality and Economy are the two utmost benefits associated with testing. Figure 2.1 shows the process of realizing chips[3]. *Requirements* are the needs of the user satisfied by the chips derived from the function of a particular application. *Specifications* includes input/output characteristics, operating characteristics (Ex. power, frequency), environmental characteristics (temperature, humidity, reliability). Design has several stages. Firstly, system-level

structure of realizable blocks is implemented using *architectural design*. *Logic Design* decompose block into logic gates. These gates are implemented using physical devices like transistors and a chip layout is obtained using *physical design*.

The physical layout is converted into photo masks that are directly used in the fabrication of silicon Very Large Scale Integration (VLSI) chips. Impurities and defects in materials, equipment malfunctions, and human errors are some causes of defects.

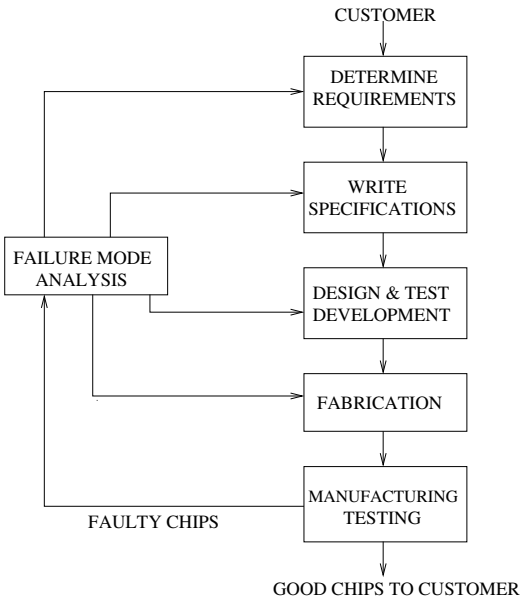

Figure 2.1: VLSI Realization Process[3]

The likelihood and consequences of defects are the main reasons for testing. Diagnosis is another important aspect of testing in which the reason for failure of the chip is determined, be it in fabrication, design or in testing. The faulty chip analysis is called failure mode analysis (Failure Mode Analysis (FMA)). FMA uses many different test types, including examination through optical and electron microscopes, to determine the failure cause and fix the process.

Testing should be placed closest to the point of error. Test procedures should ensure that design meets all functional and other specifications and is manufacturable, testable and repairable.

VLSI testing is done in different phases of chip development. During chip design, testing is done to verify the correctness of design. This requires the involvement of the

design engineer and is called *verification testing*. This signals the beginning of production or large scale manufacturing. Fabricated chips being tested is called *manufacturing testing*. The user after having received the chip, tests it to ensure quality, a process known as *acceptance testing*.

2.2 Testability Measure

Testability measure is used to get the approximate measure of difficulty of setting internal circuit point to 0 or 1 by driving values on primary inputs or difficulty in observing internal circuit nodes by observing primary outputs. Such a measure provides an estimate of fault coverage. **Controllability** and **Observability** are the two testability measures that quantify testing. Controllability is defined as the difficulty of setting a particular circuit node to a 0 or a 1. Its value range from 1 to ∞. Observability is defined as the difficulty of observing the state of a logic signal. Its value ranges from 0 to ∞. High values means difficulty in controlling or observing circuit nodes.

There can be a number of points in the design that are difficult to observe or control. By adding special circuitry in design called test points, one can increase the testability of the design.

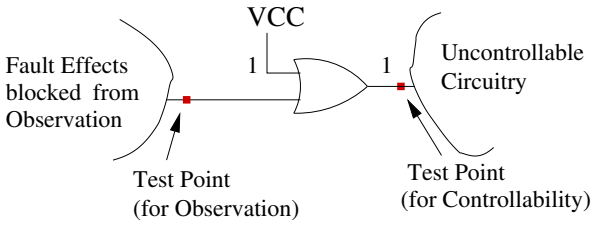

Figure 2.2: Uncontrollable and Unobservable Circuitry

For example, figure 2.2 shows a circuit node with controllability and observability problem. One input of this OR gate is tied to logic '1'. This blocks the ability to propagate any fault effect in circuitry feeding the other input. Thus, it is a test point to improve observation. Tied input causes logic '1' at the output and any circuitry downstream from the output uncontrollable. This output point becomes a test point to improve controllability.

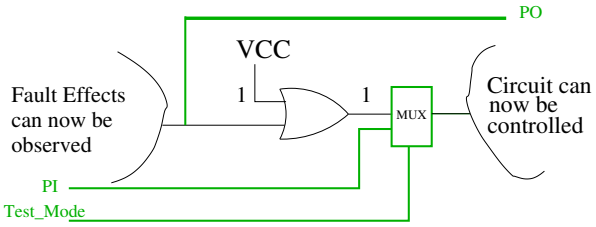

Figure 2.3: Testability benefits from test point circuitry

An added primary output at the observability test point as shown in figure 2.3 provides direct observation of this signal value. While an added Multiplexer (MUX) at controllability test point with test_mode control signal and primary input controls the value fed to the associated circuitry.

2.3 DFT

DFT is a technique, which facilitates a design to become testable after production. It is the added logic which is put in the design, during the design phase, which helps in its post-production testing. Post-production testing is essential because, the process of manufacturing is not 100% error free. There are defects in silicon which contribute towards the errors introduced in the physical device. If there are any errors introduced in the production process, a chip will not work as per the specifications. Diagnosis of the fault is a very important aspect. Since, to run all the functional tests on each of millions of physical devices produced or manufactured, is very time consuming, there was a need to device some method to ensure that the device has been manufactured correctly without running full exhaustive tests on the physical device. DFT is a technique that only detects if a device is faulty or not. After the post-production test, if it is found faulty, it is trashed and not shipped to the customers. Since it is a production fault, there is assumed to be no cure. So, the intended purpose of the DFT, is just a detection, and not the localization of the fault. For the end customer, the DFT logic present on the device is a redundant logic. To further justify the need of DFT logic, consider an example where a company needs to provide 1 Million chips to its customer. If there isnt any DFT logic in the chip, and it takes for example, 10 seconds (usually takes larger than that) to test a physical device, then it will take approximately three and a half months just to test the devices before shipping. So the DFT is all about reducing

three and a half months to may be three and a half days. Of course practically many testers will be employed to test the chips in parallel to help reduce the test time.

Before the device is sent to production, possible sources of errors during the production process are thought of and how will they be diagnosed after the device is manufactured. It can be a short circuit or an open circuit. There is a need to model the possible faults that might occur during the fabrication process. This is called ***fault modelling***. The most widely used fault model is referred to as single stuck-at fault model.

DFT is a technique aimed to simplify testing by alternating a design to improve the controllability and observability.

There are two main approaches to DFT: Ad-hoc and Structured

- **Ad-Hoc DFT** : Ad-Hoc DFT implies good design practices to enhance a design's testability without making major changes to the design. Ex:
  - Minimizing redundant logic
  - Minimizing asynchronous logic
  - Isolating clocks from logic
- **Structured DFT** : Structured DFT goal is to increase the controllability and observability of a circuit. To accomplish this, the most common methods are:
  - ***Scan Design*** technique, which modifies the internal sequential circuitry of the design.
  - ***BIST*** method, which inserts a device's testing function within the device itself.
  - ***Boundary Scan***, which increases board testability by adding circuitry to a chip.

2.3.1 Scan Design Overview

The objective of scan design is to improve the testability of sequential circuit so as to make it behave like an easier-to-test combinational circuit. To achieve this, sequential or memory elements are substituted with scannable sequential elements called *scan cells* and then stitching the scan cells together into scan registers, or scan chains. This

serially-connected scan cells can be utilized to shift data in and out when the design is in scan mode. The design in Figure 2.4 has both sequential and combinational logic. With the addition of scan design, two inputs, **sc_in** and **sc_en** and an output, **sc_out** are added to the already existing inputs **A**, **B**, **C** and outputs, **OUT1**, **OUT2**. This "Before Scan" design is difficult to initialize to a known state, making it difficult to both control the internal circuitry and observe its behaviour using the primary inputs and outputs of the design. Scan memory elements replace the original memory elements so that when shifting is enabled (**sc_en** is active), scan data is read from **sc_in** line.

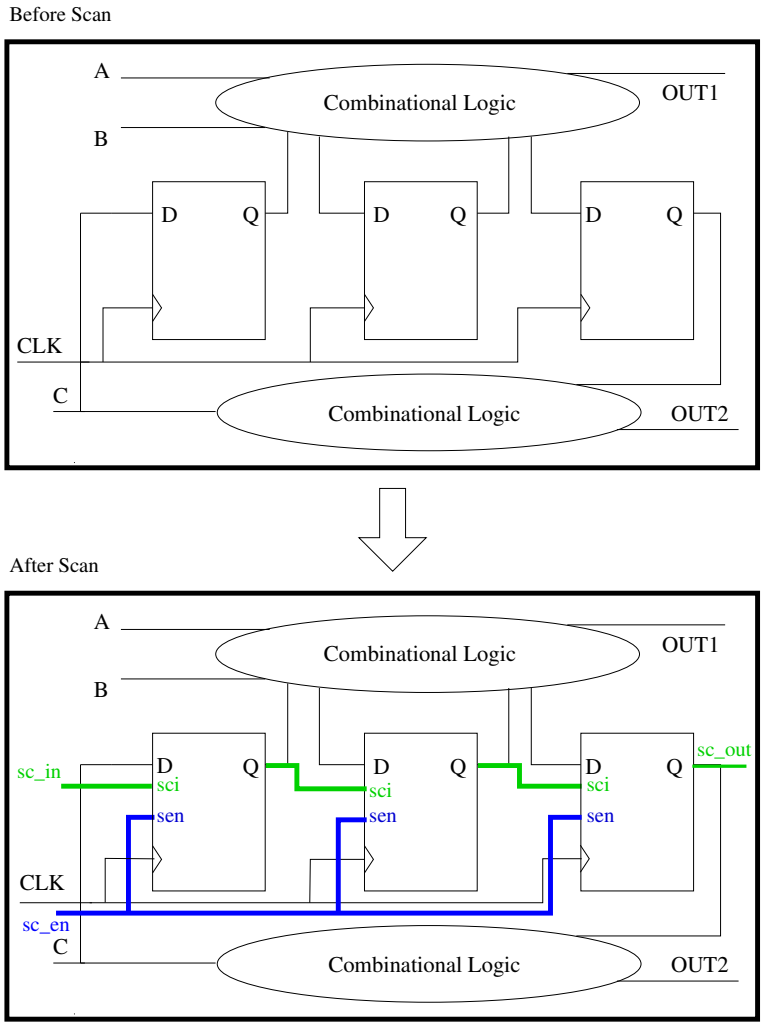

Figure 2.4: Design before and after adding scan

The operating procedure of scan is as follows:

- Enable the scan operation to allow shifting (to initialize scan cells)
- After loading the scan cells, hold the scan clocks off and then apply stimulus to the primary inputs.
- Measure the outputs.
- Pulse the clock to capture new values into scan cells.
- Enable the scan operation to unload and measure the captured values while simultaneously loading in new values via the shifting procedure

2.3.2 Boundary Scan

JTAG/Boundary Scan is another form of scan circuitry and was the first test methodology to become an IEEE standard (IEEE 1149.1). This standard defines features to be designed into an integrated circuit that provides access to its digital I/O pins from the inside of the device. This allows circuit nodes on the Printed Circuit Board (PCB) to be accessed with device internal test features, rather than with a bed-of-nails fixture. With these built-in test features, Boundary Scan helps us to access circuit nodes that may not be accessible when using external physical access methods with probes. Boundary scan tests can be developed very rapidly and early in the design cycle, as soon as the schematic design of the Unit Under Test (UUT) is available, even prior to having the layout of the PCB finished.

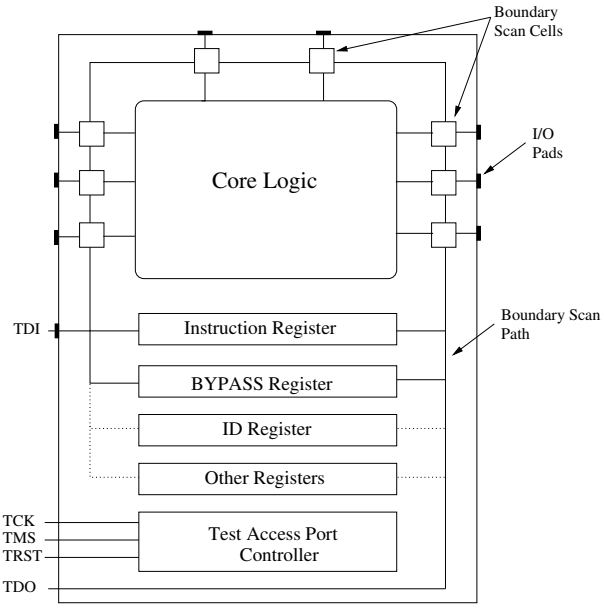

Figure 2.5: JTAG enabled device

The principle of boundary scan can be understood from the figure 2.5. Boundary Scan Register (BSR), which is a serial scan path intercepts the device's core logic and the 'pins'[4].

2.3.2.1 Interface Signals

The JTAG interface, collectively TAP uses the following signals to support the boundary scan operation.

- **Test Clock (TCK)** - This synchronizes the internal state machines operations.
- **Test Mode Select (TMS)** - This is sampled at the rising edge of TCK to determine the next state.
- **Test Data In (TDI)** - This signal represents the data shifted into the device's test or programming logic and is sampled at the rising edge of TCK when the internal state machine is in the correct state
- **TCK** - This signal represents the data shifted out of the device's test or programming logic and is valid on the falling edge of TCK when the internal machine is in the correct state
- **Test Reset (TRST)** - This is an optional pin, when available, can reset the TAP controller's state machine

These signals are present at the top level of the design and are directly accessible. By applying appropriate logic values on these signals, test mode entry can be achieved. Also, shifting in data into the *JTAG_REG* is done through TDI signal while clocking test clock i.e TCK in ShiftDR TAP state. Programming of such registers allows to control IPs to test them independently by configuring them or mapping the inputs or outputs to top level General Purpose Input Output (GPIO) pads.

2.3.2.2 TAP Controller

The TAP controller is a state machine whose transitions are controlled by TMS signal, controls the behaviour of the JTAG system. Figure 2.6 shows the state-transition diagram. TAP is used to toggle through these state to program the test data registers in standalone test modes.

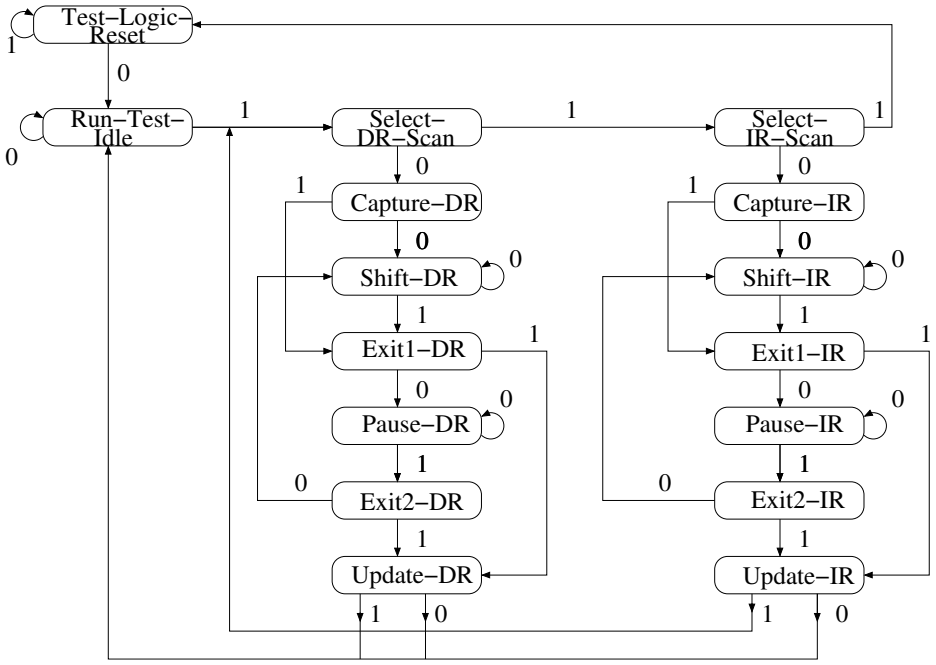

Figure 2.6: TAP State Diagram

2.4 Test Pattern

The goal of test generation is to produce a complete fault detection test set(vector), a set of tests that detect any detectable faults. Test Patterns are generated before the production of the device. These test patterns are used in a pre-production simulation called ***fault-simulation***. A test plan normally comprises of different types of tests to achieve high test coverage. Many devices utilize structural scan tests to achieve thorough test coverage. BIST methods are also used extensively, along with JTAG (IEEE 1149). Functional patterns still persist as a way to round out the test suite. Scan pattern formats are typically STIL or Waveform Generation Language (WGL). Functional patterns are typically created from a Verilog simulation generated VCD file. If the observed output fails to match the expected output for a particular set of input value or pattern, we generally say that this pattern has failed. Note that a test-pattern will never fail during a pre-production simulation, or fault-simulation. These so called test patterns will be re-used after the production to run what is called ***production-test***.

2.4.1 Types of test patterns

- ATPG Scan Pattern
- Memory Testing patterns using BIST
- Standalone testing of analog and mixed signal macros.

2.4.1.1 ATPG Scan Patterns

A scan pattern contains the events which include forcing a single set of values to all the flops in a scan chain and primary inputs, followed by observation of the subsequent responses at all primary outputs and scan cells. The basic scan pattern is made up of following events:

- Load values into scan chains.
- Force values on all non-clock primary inputs.
- Measure all primary outputs.
- Pulse a capture clock.
- Unload value from scan chain.

Figure 2.7 shows the scan operation to detect the stuck-at fault in the Device Under Test (DUT) whose memory elements are chained together as shown. The test vector '100101011' is shifted in the scan chain through the serial input **Scan In** with **Scan Enable=1** using the scan shift clock which is much slower than functional clock of the circuit. Shifted-in test vector is currently applied to the combinational logic that are driven by the scan flip-flops. Thus 2^{nd}, 3^{rd} and 4^{th} combinational logic block are already forced test inputs.

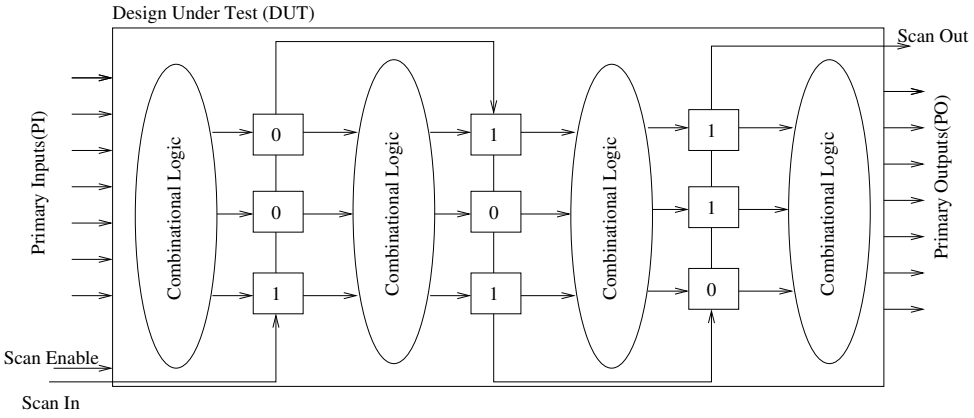

Figure 2.7: Scan Chain Operation for stuck-at fault

Forcing values to Primary Input (PI) to 1^{st} combinational block forces its output to be ready. Furthermore, the outputs of 4^{th} combinational block can now be observed from Primary Output (PO)s. In order to push the output values of combinational blocks 1,2 and 3 into scan flip-flops, the system clock needs to be toggled. On applying the system clock, D flip-flops will capture the value at their inputs. Then captured values are then shifted out using shift clock serially through the **Scan Out** while simultaneously shifting-in the next test vector.

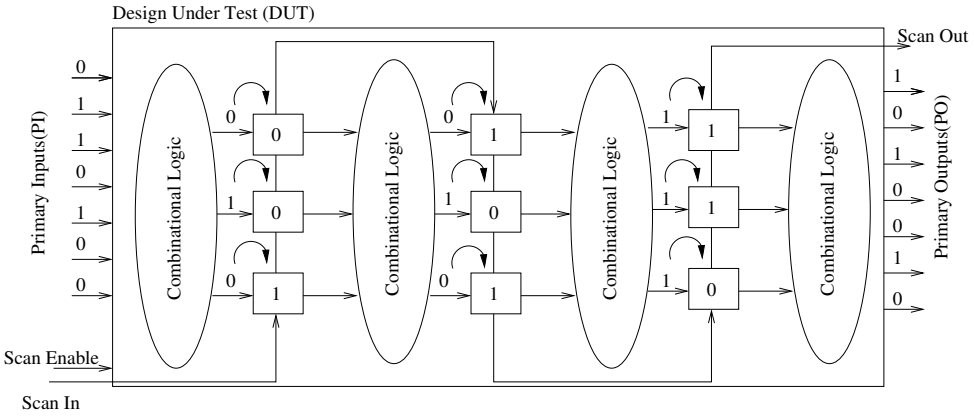

Figure 2.8: Application of test vector to detect stuck-at fault

2.4.1.2 Memory Testing Using BIST

Memory cores usually represent a significant portion of the chip area such that the yield of memory dominates the yield of chips. Memory density is higher than logic

thus, chance of having a defect is usual in embedded memory. According to the International Technology Roadmap for Semiconductors, 2007 [5], the percentage of memory in Central Processing Unit (CPU) cores ranges from 65% to 75%. A BIST block is an off-line verification of the CUT implementing an algorithm to test it. Integration of Memory BIST allows a direct access to the memories from the top level and hence, reduces the design complexity.

BIST controller acts as an interface between the collar and the user. It controls various testing and reporting activities of the memory test. The *BIST collar* is a wrapper close to the memory responsible for executing instructions received from the controller in the form of march elements to test the memory. It runs at speed for the memory under test. The collar also supports testing multiple memories in parallel or sequentially. They are soft IPs and occupy about 2% to 7% of the memory size.

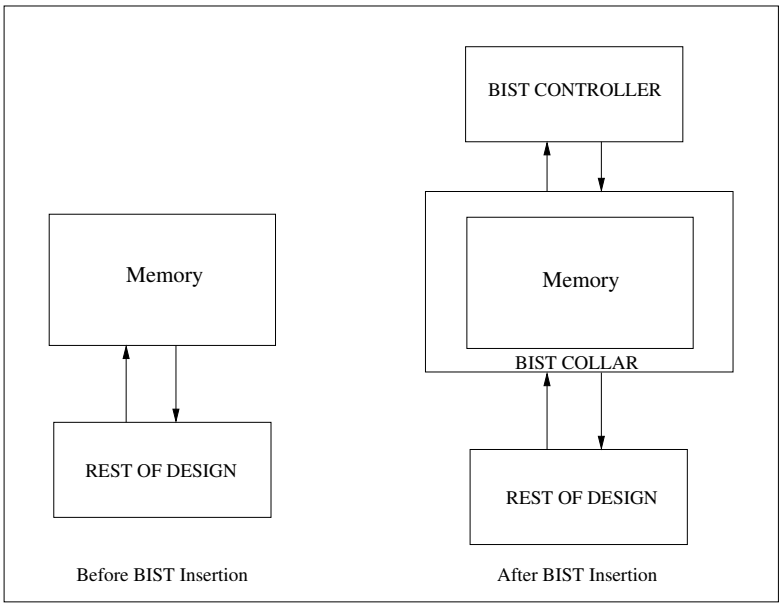

Figure 2.9: Memory Testing using BIST

A BIST is made of one controller associated with one or more collars. There are 3 types of collars that are supported in the BIST. They are dedicated collar, parallel shared collar and sequential shared collar as represented in figure 2.10.

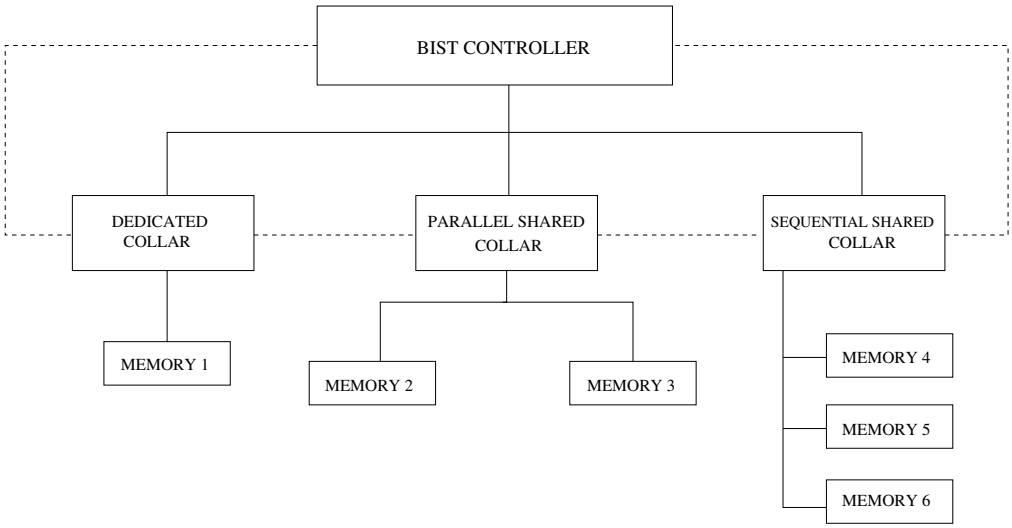

Figure 2.10: Controller and collar configuration

A BIST uses a built-in algorithm usually called march algorithm to perform a series of read and write operation with the objective to detect one or more defective rows or columns. If the memories have redundant rows/columns and is repairable, these defective row or column is replaced by a redundant row/column. The algorithmic elements are shifted serially from the controller to the collar. All data transfer between the controller and collar are done at a low clock frequency(tester clock), while the algorithm execution at collar level is done at memory speed. Thus, each collar has two clocks(tester clock and memory clock), while controller operates at only tester clock frequency.

2.4.1.3 Standalone Mode testing

Standalone mode are used primarily for analog macro testing. In standalone mode, all the necessary inputs and outputs of the IP under test are mapped to the primary pins of the device, so that test engineering can test various parameters of the IP for the given specifications. Test strategy for some of the macros for standalone mode is discussed in the next section that are implemented using the proposed work.

2.4.2 PLL standalone test

Figure 2.11 depicts the connections around a PLL block. Functional or test input is determined from the enable input of the multiplexer that comes from a test data register bit (*JTAG_REG_BIT*). Test input may come from a GPIO pin of the device or JTAG_REG_BIT. Similarly, outputs are mapped to the GPIO pins of the device. PLLs on the chip can be activated with individual control through JTAG A sample PLL standalone test case has the following sequence:

- Program the JTAG_REG to set the *pll_standalone_bit*, power down the PLL and program the values of NDIV, IDF, ODF of PLL.
- Wait for certain μs for PLL power down.
- Program the JTAG_REG again to power up the PLL.
- Wait to allow PLL to lock.

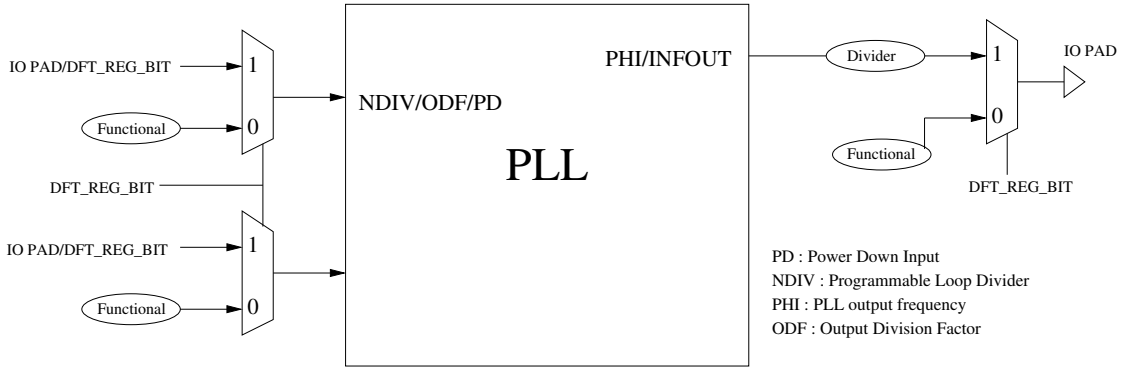

Figure 2.11: Standalone mode connections

2.4.3 18-bit delta-sigma ADC standalone test

The following sequence is used to test this block:

- Test mode entry into the standalone test of a ADC is done through *JTAG_REG_BIT* and other ADCs and DACs are powered down by setting appropriate bits. This gives the control of inputs and outputs to the GPIOs as shown in the figure
- Channel selection is done by GPIO pin at top level.

- Sinusoidal signal of 48 KHz is then applied at the selected channel.
- Clock input of Fs i.e. 48 KHz is applied on GPIO1
- The converted output is available on GPIO3 which is sampled for 32 pulses of SCK which is the clock for outputting serial data.

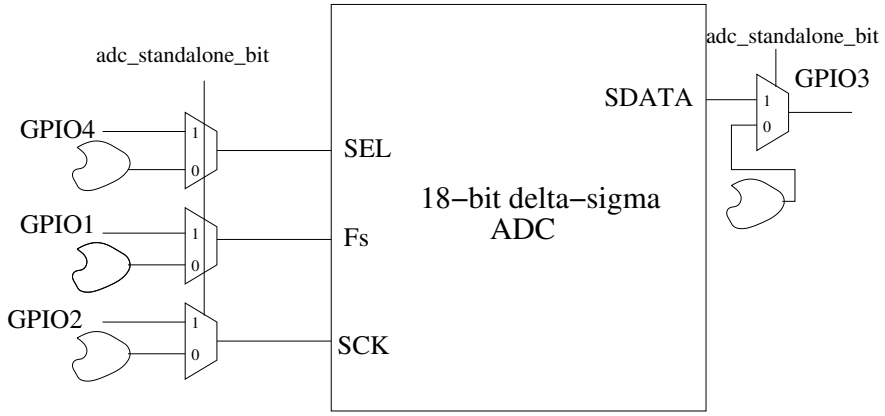

Figure 2.12: ADC standalone test

3

State of the art

To test semiconductor Integrated Circuit (IC) devices using a semiconductor IC test system, the basic principle of functional testing in a semiconductor IC test system as shown in the 3.1 involves generating input stimulus for the IC device, applying these stimulus to the IC device and comparing the output of the IC device with the expected results by strobing the output at predetermined times. Such input stimulus and strobes are collectively called a test pattern or test vector and are traditionally generated based on test data in a cyclized form. This conventional test system is called a cycle based test system or a cyclized tester where data for producing the input stimulus and strobes is defined in reference to corresponding test cycles (tester rates or time-sets). This cycle based test systems are generally used in mass production of IC devices[6].

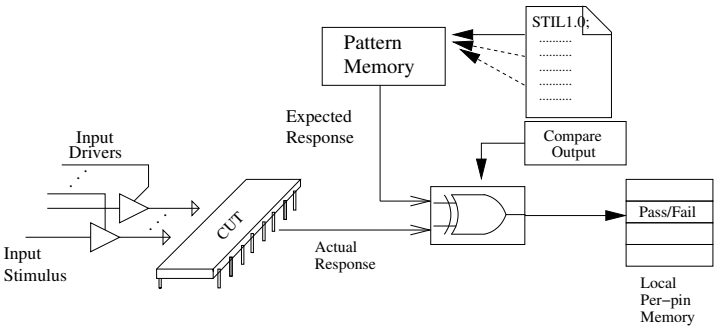

Figure 3.1: Principle of functional testing of a semiconductor device

The cyclized testers are generally used in the production process of the IC devices, because they have serious constraints that the design data cannot be used directly for testing. Thus, they are not considered fit for development process of IC designs.

Electronic Design Automation (EDA) environment is used for IC design where designers use high-level language such as Verilog or VHDL and simulate the design by behavioural, gate-level Verilog simulators producing input/output event data of the design, i.e. VCD. Such simulation is intended to verify the functionality and performance before the design is sent for fabrication. This design simulation data needs to be converted to cyclized form such as WGL or STIL, in order to test the device during the production stage of the IC device.

Figure 3.2 shows the simulation dumped VCD file description and the event based description of waveforms which is set/reset type event based format showing the changes in the input and output of the designed IC. As can be seen, the design simulation data (VCD) is similar to the event based description.

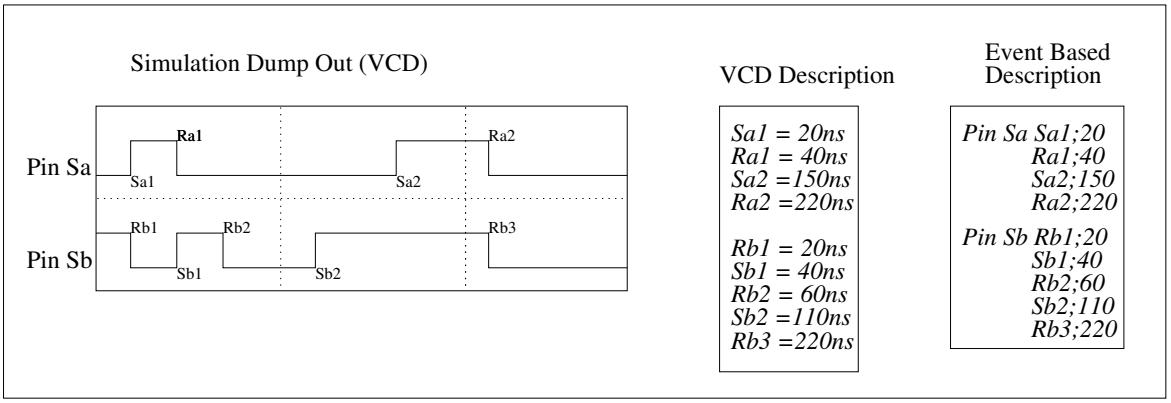

Figure 3.2: VCD and event based description

Since, the operating environment of test systems are different from the IC design environment, vector conversion consumes extensive time, requires larger server and disk capacities, and is susceptible to errors. Moreover, every tester has it's own unique and proprietary language and vector format (ex. Test Description Language (TDL) by Advantest). Test environment requires engineers to reformat simulation testbenches and re-run simulation to capture the cyclized vectors that are required for testing. Figure 3.3 is a schematic block diagram showing an architecture of the event based test system. An event generator is used to generate events based on event data stored in event memory by converting them into actions for driver to apply a test vector. Event data indicates the timing of the event and type of event. The test vectors are applied to the DUT by the driver, and comparator is used for comparing the output of the

DUT with the expected data. The driver and comparator are collectively called as pin-electronics. The event based tester store timed events thus, they require a significantly large amount of memory as compared to the cyclized tester[7].

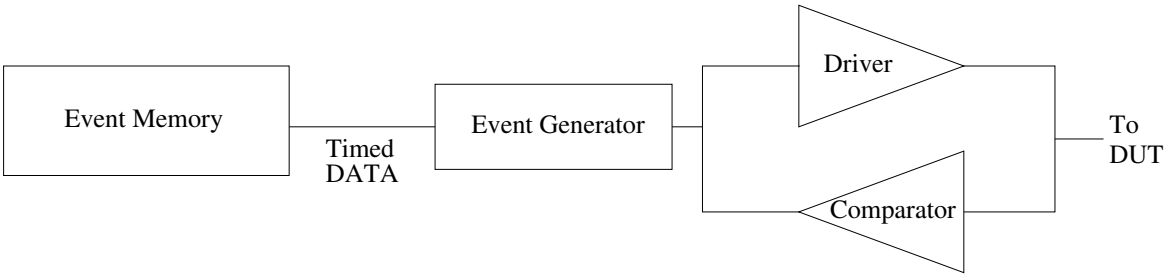

Figure 3.3: Architecture of event based test system

During the development of IC device, the IC testing environment required by event tester is the same as the original IC design environment. Conversion of simulation testbenches to conventional Automatic Test Equipment (ATE) cyclized form, and VCD to STIL or other formats is not required to be done by the engineers.

While the event testers are convenient in the IC development and debug, cyclized testers are used in the production stage of the IC devices. Large industrial ATE operate at fixed device cycle thus require test data in 'cycle-based' format[8].

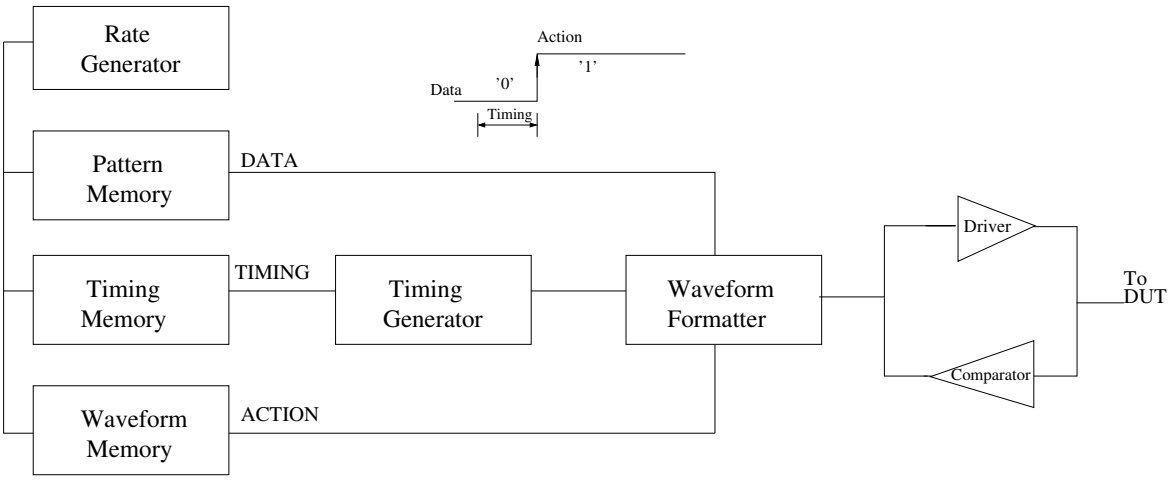

Figure 3.4: Architecture of cycle based test system

For semiconductor test system using the cycle format, test data must be divided into test cycles, waveforms and pattern values. The architecture of cycle based test

system is shown in figure 3.4. It is composed of a rate generator for producing test cycles. It has timing, pattern and waveform memories for storing timing, pattern and waveform data respectively. Timing generator is used to generate timing signals based on timing data. Waveform formatter develop a test pattern using the timing signals, pattern data and waveform data. Driver is used for supplying the test vectors to the DUT while, comparator compares an output of the DUT with the expected output[6]

3.1 Simulation based pattern generation for production tests

There is increasing concern today about the escalating cost of test and the ability of ATE to meet stringent timing/accuracy requirements for those test strategies which use at-speed functional tests. Such tests require not only high speed clocks but also accurate edge placement for a large number of pins. A conventional pattern generation flow is shown in the Figure 3.5. This flow starts with simulating the device behavior using an input stimulus usually derived from the Verification Database. This database consists of testbench environment and test cases that acts as input stimuli for the Design Under Test (DUT)[1]. These test cases are written in System Verilog or System C and are simulated using simulation tools whose output is a Value Change Dump (VCD) file that depicts the signal changes in the form of time related events. This event-based VCD

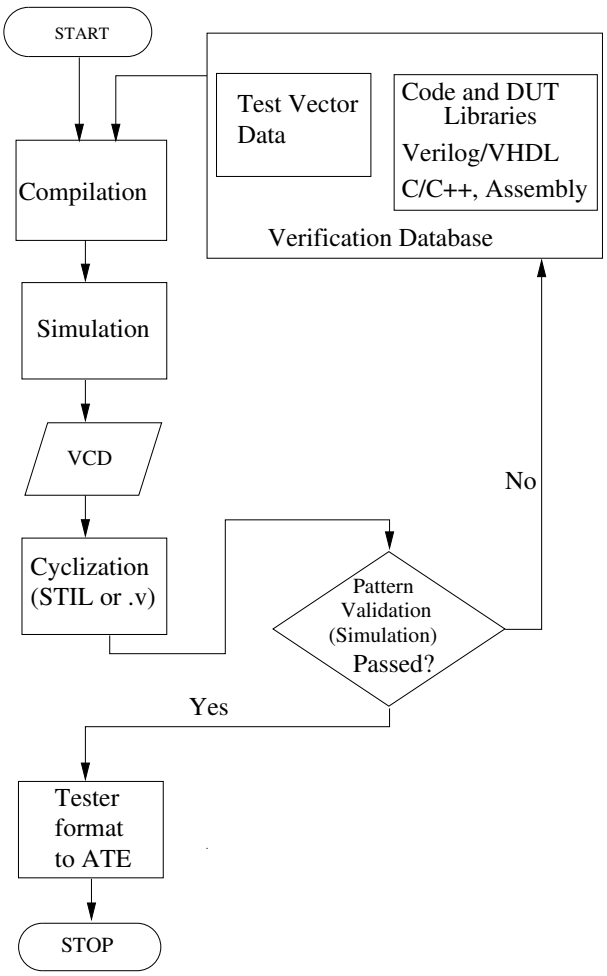

Figure 3.5: Simulation Based Pattern Generation Flow

file is translated to tester specific cycle-based force and expect values using some cyclization tools. Cyclization involves identifying the fastest signal period in the VCD, usually a clock signal, and using its period as the cyclized period. This cyclized period is used for all signals in the VCD to process all the event states and create drive/receive edge timings for all signals. These timings are all relative to the cyclized period. This ATE format file is run on silicon and in case of failures, the VCD for the specific failure needs to be regenerated. This take considerable time since changes must be fed back into the Verification Database and the entire flow has to be repeated every time a pattern update is necessary. To eliminate pattern conversion, an approach is proposed in [9] however, the focus is only on enhancement of Memory BIST pattern generation.

3.1.1 Limitation of this approach

- $PatternVerification$: Any change to the test patterns-regardless of how simple-may necessitate verification. For each pattern, code compilation takes minutes to hours while simulation time takes hours to days thus affecting Cost of Test.
- $LargeVCDfiles$: VCD formats have been uniformly adopted way of capturing simulation output. Every ATE has limitation like vector memory, number of permitted drive and receive edges.VCD files can grow quite large for larger designs, or even for long simulation time in case of smaller designs.
- ATE debug time can increase exponentially due to re-simulation, re-conversion, and more ATE debug activities if VCDs are not created and converted properly.

3.2 IEEE 1450 : Standard Test Interface Language (STIL)

The STIL environment supports transferring tester-independent test programs to a specific automated testing equipment (ATE) systems. A direct link between the ATPG tools and ATE environment can be established using STIL[10]. It allows to write the timing and pattern information for the application of digital test vector to a DUT. Thus, translation of VCD format into tester specific format is eliminated. STIL test pattern is structured using following modules also represented in figure 3.6.

3.2.1 STIL Constructs

- **Signal**: This block defines all the top level pin information along with the direction. There is one such block allowed in a STIL file.
- **SignalGroups**: This is used to create a named reference for one or more signal. A group of one signal can be used as an alias name for that signal. This property can be used to create generic procedures.
- **PatternBurst**: This specify list of patterns to be executed in a single execution.
- **Timing**: This defines the placement of timing edges and cyclized waveforms format that is referenced by the WaveformChars in the vector statements.
- **WaveformTable**: This contains a set of waveforms for each signal. Its name is used in the pattern block to reference the waveform.
- **Waveforms**: This is defined in WaveformTable block. Each waveform is identified by the signals that are being defined.
- **Pattern**: This block has following main statements:
  - **Vector**(V) Statement: This statement defines the stimulus and response for that particular test cycle.
  - **Condition**(C) Statement : This defines the stimulus and/or response to be set up. This is used in scan testing when defining the background states for signals before applying the scan data.
  - **Call** Statement: This statement makes a call to a named procedure in **Procedures** block and transfers the execution to that block. Such a procedural STIL file is shown in figure 4.3.
- **Procedures**: This has named set of procedures that are called from a pattern block. Procedures may be defined in a separate **.stil** file and this file can be included in the main STIL file as shown in the figure 3.6.

```
STIL 1.0;
Include "./stil_procedures.stil";
Signals {
   DIR In;
   OE_In;  JTAG_TDI In; JTAG_TMS In; JTAG_TCK In;  JTAG_TRSTn In;
    A0 InOut; A1 InOut; A2 InOut; A3 InOut;
    A4 InOut; A5 InOut; A6 InOut; A7 InOut;
    B0 InOut; B1 InOut; B2 InOut; B3 InOut;
    B4 InOut; B5 InOut; B6 InOut; B7 InOut;
}

SignalGroups {
    ABUS = 'A7+A6+A5+A4+A3+A2+A1+A0'     ;
    BBUS = 'B7+B6+B5+B4+B3+B2+B1+B0'    ;
    BUSES = 'ABUS + BBUS';
    ALL = 'DIR + OE_ + BUSES + JTAG_TCK +
              JTAG_TDI + JTAG_TMS + JTAG_TRSTn';
    tck = 'JTAG_TCK';
    tdi = 'JTAG_TDI  ';
    tms = 'JTAG_TDO';
    trstn = 'JTAG_TRSTn';
 }
Timing T1 {
     WaveformTable _default_ {
      Period '500ns';
      Waveforms {
      DIR { 01 { '0ns' D/U; }}
     OE_ { 01 { '0ns' D/U ; }}
     ABUS { XH LT  { '0ns' Z; '260ns' X/H/L/T; }}

}

PatternBurst "burst" {
   PatList { "pat1"; }
}

PatternExec {
    PatternBurst ¨burst" ;
}

Pattern "pat1" {
    W _default_;
    C { ALL = \r 18 Z; }
    Call "load_test_data_reg" { tdi = 0001010111010 };
}
```

Figure 3.6: STIL pattern structure

3.3 STIL Based Pattern Generation

Reducing Cost of Test (Cost of Test (CoT)) is being more demanding which requires reducing test time to meet aggressive Time to Market (Time to Market (TTM)). CoT and TTM can be hindered by factors like long vector simulation times, large VCD files, VCD to ATE binary conversion, ATE test time per hour cost. Thus, there is a need to remove pattern generation dependency on simulation.

STIL based pattern generation eliminates translation into tester specific formats by establishing direct link between ATPG tools and ATE[11]. ATPG tools like Mentor Graphics Tessent®, Synopsys TetraMax™ typically provide scan test pattern in STIL format which requires manual handling and knowledge of the standard to generate the test pattern. A suitable method for verifying the test patterns is achieved by creating a relatively simple testbench that accesses the patterns directly during simulation through a Verilog Programming Language Interface (PLI)[12]. Such a method avoids creating a large testbench which includes all the test vector. Using this method, test data can be modified without recompiling the design. However, manually writing a STIL pattern can be time consuming and error prone since pattern has to be written vector to vector. This may lead to larger time to test thus affecting time to market.

4

Proposed Work

As a result of long test development cycles due to cyclization of simulation based data to make it compatible for production tests on ATE, test vectors in STIL can be adopted to write test vectors in cyclized form directly. While STIL is accepted as an IEEE standard for writing cyclized pattern for production testing, it is very challenging to write test vectors cycle-by-cycle. Moreover, manual pattern generation can be error-prone requiring strong hold of STIL syntax. Thus a novel approach to re-usable, time-economized pattern development is proposed.

4.1 Proposed Flow of Pattern Generation

The proposed approach for pattern generation has been shown in Figure 4.1. A library of STIL procedures or templates are created that are specific to IPs whose pattern are created. This is a one time investment since these templates can be reused for the same IP in a different SoC. A database is also created containing pattern specific information. Using these two information, an ATE format pattern i.e STIL pattern is created. This pattern is then validated by simulation. The main focus of dynamic validation is to verify whether the patterns are functionally right (produce the correct and desired results) and do not harm the device or the ATE. The dynamic validation can only be performed through simulation. The ability to simulate the vectors in ATE formats (like STIL) flags off many tester failures early and reduces cost of tester debug. If the pattern fails the schema can be manually edited for debug purposes. This speeds up

debug and reduces pattern regeneration time as simulation is not required to generate VCD file.

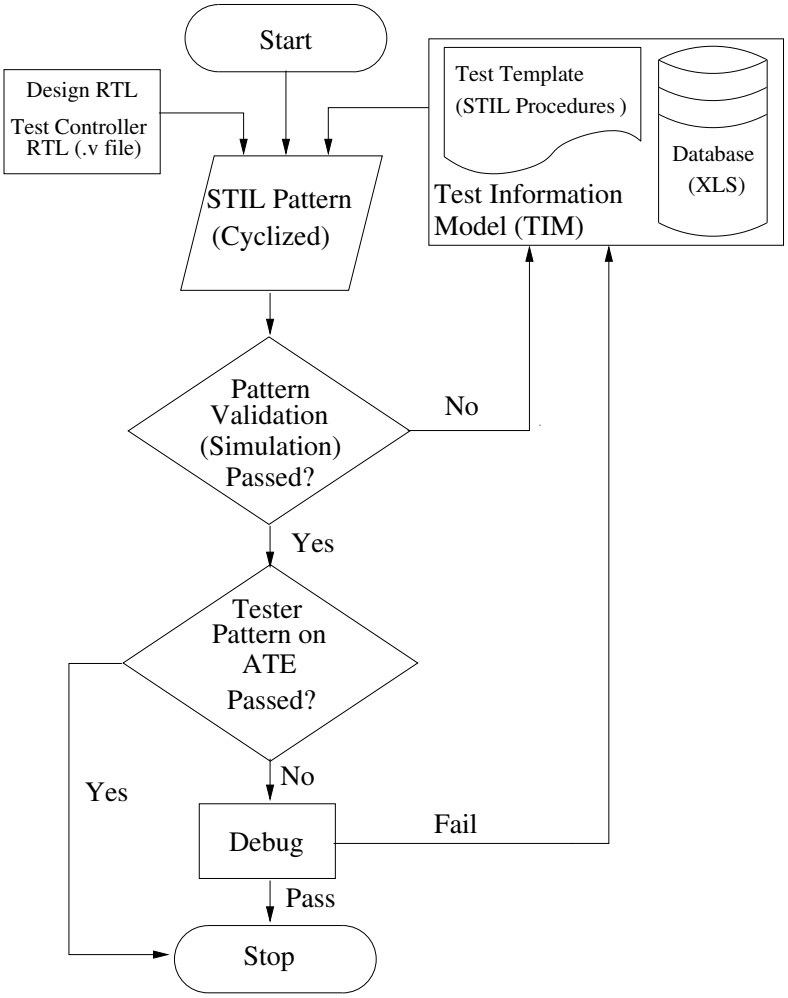

Figure 4.1: Proposed Pattern Generation Flow

The entire data flow for pattern generation is shown in figure 4.2. DUT Register Transfer Language (RTL) and test controller RTL act as inputs to the flow. DUT RTL in Verilog Hardware Description Language (HDL) is used to extract the top level pin information along with their direction. This information is used to generate the STIL Signals construct. Test controller RTL is utilized to obtain Test Data Register (TDR) bits information. This information is used as a reference to create functional test pattern from a sequence of call to *load_test_data_reg* procedures by programming appropriate bits of TDRs in database.

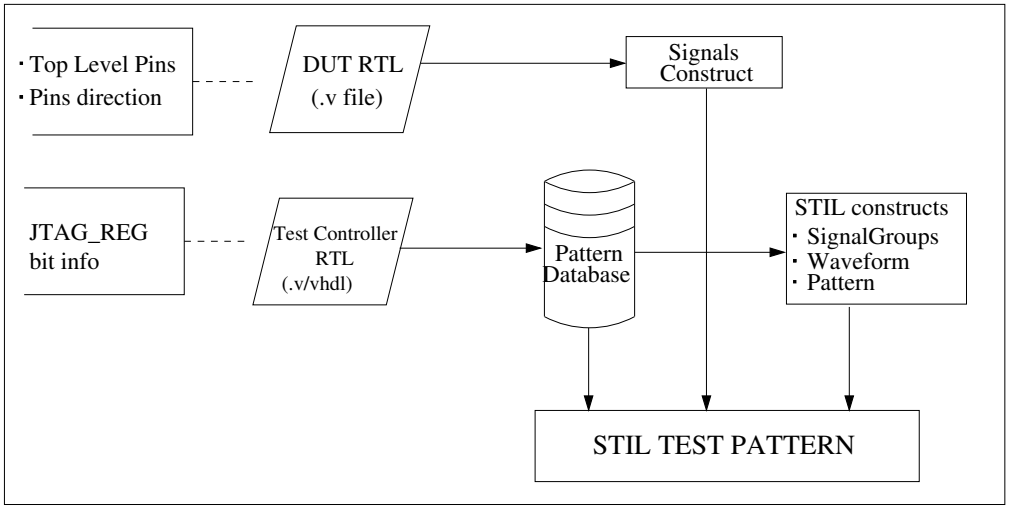

Figure 4.2: Data flow in pattern generation

4.1.1 Test Information Model

Test Information Model (Test Information Model (TIM)) contains database used to convey information about the test framework of an SoC or embedded core as well as to describe complete test programs that it can execute. TIMs serve the purpose of delivering test vectors generated in ATE format (STIL) to test engineering responsible for running the tests on ATEs. TIM is composed of :

4.1.1.1 STIL Library

This is composed of a set of STIL Procedures specific to IPs as well as procedures commonly used in different SoCs to configure their test data registers (TDRs). A STIL procedure is a set of patterns to be executed in an order to accomplish a task like precondition (load) the scan chain, and to observe (unload) the scan chain. The tool will make use of this library to generate a STIL file for a particular IP in a SoC. With a new IP to be functionally tested, a generic procedure to verify its functionality can be created which can be reused in different SoCs. Library of STIL procedures contains:

- Generic STIL procedures: Ex. *load_test_data_reg*–This includes sequence of patterns to configure the test data registers as shown in Figure 4.3 with Test Access

Port (TAP) states. These can be reused across SoCs even with different functional blocks but with same test controller.

- Specific STIL Procedures: These are specific to a functional block. For example, *program_fuse, read_fuse* to write and read memory locations respectively of One Time Programmable (OTP) fuses. These can be reused for SoCs with common functional blocks.

```
STIL 1.0
Procedure {
"load_test_data_reg" {
Shift{ V{"tck"=P; "tdi"=#; "tms"=0; "trstn"=1; }}   //ShiftDR
V{"tck"=P; "tdi"=#; "tms"=1; "trstn"=1; }            //Exit_1_DR
V{"tck"=P; "tdi"=1; "tms"=1; "trstn"=1; }            //UpdateDR
V{"tck"=P; "tdi"=0; "tms"=0; "trstn"=1; }            //Run_Test_Idle
}}
```

Figure 4.3: An abstract of STIL Library

4.1.1.2 Database

The development of test pattern by configuring long test data registers is very tedious as these registers can be ∼200 bits long. Shifting in desired values into them vector by vector by JTAG TAP can lead to long test development time. This is a small schema created for every pattern containing information about signals that are forced or expected in that test case. Table 4.1 is an abstract from such a database showing an example that defines information required to generate the **signal group** and **waveform** construct for a particular test case as shown in figure 4.4. Tester period is also provided and all the timings specified are in reference to the given tester period since the vectors written are in cyclized form. The database also contains the specifications of the test data registers as shown in the table 4.2 which are dumped using the test controller RTL. This information is used to generate the test case **pattern** construct.

	Default		
Tester Period	33ns		
Signal_group	Default State	Strobe Time	
all_bidirectionals	Z	15ns	
Pattern Specific Signal Group	Pins		
tdi tck trst_n tms tdo	JTAG_TDI JTAG_TCK JTAG_TRSTn JTAG_TMS JTAG_TDO		
Pattern_clock	Default State	T1	T2
tck	U	10ns	26ns

Table 4.1: Database Template

JTAG_REG1	Bit Location		JTAG_REG2	Bit Location
tst_enable_bit	1		en_ck_gated_bit	1
pll_standalone_bit	2		adc_byp_bit	2
:	:		en_top_bist_gated	3
pll_idf_2_bit	45		:	:
pll_idf_1_bit	46		bist_top_mode_sp	32
pll_idf_0_bit	47		:	:
pll_pd_bit	48		bist_shift_en_bit	123
pll_byp_bit	49			

Table 4.2: JTAG_REG specifications

```
SignalGroups {

"all_bidirectionals" = '"Pin1" + "Pin2" + "Pin3" + .... + "PinN"'    ;
"tdi" = '"JTAG_TDI"' ;
"tck"= '"JTAG_TCK"';
"trst_n" = '"JTAG_TRSTn"';
"tms" = ''JTAG_TMS"';
"tdo" = '"JTAG_TDO"';
}

Timing {
        Waveform Table "_Default_WFT_" {
                Period '33ns' ;
                  Waveforms  {
                                "all_bidirectionals" { X { '0ns' Z; '15ns' X; } }
                                  "all_bidirectionals" { H { '0ns' Z; '15ns' H; } }
                                  "all_bidirectionals" { 0 { '0ns' D; } }
                                  "all_bidirectionals" { 1 { '0ns' U;   } }
                                  "tck" { P { '0ns' U; '10ns' D; '26ns' U; } }

                    }
         }
}
```

Figure 4.4: Signal Groups and Timing Construct

4.1.2 An Automated Solution

Lack of automation in functional test pattern generation and long test development time have been cited as one of the reasons to move away from function testing to structural testing. However, functional tests are more effective in testing at speed, under realistic electrical noise[13]. Automation offers potential for both reducing test engineer workload during pattern development and risk of error. The source of automation adopted is scripting in Perl. The script has the STIL rules intelligence embed into it.

4.1.3 A PLL test case

An excerpt from functional test pattern of PLL is shown in Figure 4.5 in which first call to *load_test_data_reg* procedure in Figure 4.3, power down the PLL and configure it based on sequence provided by the user in the database as shown in table 4.3 using 4.2 as reference. Then, second call to the procedure power up the PLL after a certain wait period. Lastly, wait period is given with system clock running to allow PLL lock

signal to go high.

Register Programming Sequence	Bit Name	Value
JTAG_REG1	tst_enable_bit	1
	pll_standalone_bit	1
	pll_idf_2_bit : pll_idf_1_bit	0x03
	:	:
	pll_pd_bit	1
JTAG_REG1	tst_enable_bit	1
	pll_standalone_bit	1
	pll_idf_2_bit : pll_idf_1_bit	0x03
	:	:
	pll_pd_bit	0

Table 4.3: A PLL test case

```
Pattern "_pattern_" {
    W "_default_WFT_";                                   – pll_pd_bit
Ann {*PLL is powered down and configuration is done*}
Call "load_test_data_reg" {"tdi"=11000...011100110...1  ; }

Ann {*PLL should be in power down for minimum 10us*}
Loop 300 { V { "tck"=P; "tdi"=0; "tms"=0; "trstn"=1; "system_clk"=P; }}

Ann {*PLL is  powered up*}
Call "load_test_data_reg" {"tdi"=11000...011000110...1  ;}

Ann {*Wait for PLL to lock*}
Loop 20000 { V { "tck"=0; "system_clk"=P; }}
}
```

Figure 4.5: Pattern block for PLL test

The flow makes use of this TIM to generate STIL pattern for every block that requires verifying its functionality. The STIL patterns obtained are in tester format which are first validated by simulation tools like Cadence NC-Sim or Synopsys VCS® and then delivered to the test engineering for running the test cases on an ATE.

4.1.4 Test Generation Efficiency

The idea of proposing this methodology at the outset was to reduce test time and keep manual intervention as minimum as possible to reduce the potential of human error. Also, expectation was to buildup a library of procedures to facilitate re-usability. Thus, the test engineer only has to give a little information in database form. The proposed flow in addition to reduction in pattern development time, permits the generation activity to be initiated as soon as first verified RTL of the DUT is available.

4.1.5 Pattern Reuse

Signal Groups construct in STIL provide a mechanism to reference a signal using alias name. This property can be used to build up a library of STIL procedures with generic names. The tool will take care of assigning generic names to the actual signal names in the STIL pattern file by specifying it in the database. Thus, apart from Signals, Waveform constructs and power-up sequence, same pattern can be reused for an IP in a different SoC.

5

Results

The proposed approach of test pattern generation was used for analysis on SoC1 and SoC2. The complexity of these SoCs are provided in table 5.1. The flop count is obtained from the Tetramax ATPG. Plan to test an IP at system level is deduced from its functional specifications and its production pattern is developed in cyclized form i.e.STIL format using automated flow. However, IP reuse in a different product requires almost no time to develop its pattern. These production test patterns are validated by simulation using Cadence NC-Sim by creating a relatively simple testbench that instantiates the DUT and accesses the patterns directly during simulation through a Verilog PLI. The functionality of the macro is verified by strobing signals at appropriate times as given in the device specifications. Ex. Strobing a *pll_lock* signal within a defined time window to verify if it goes high in this time frame and thus, lock is successful.

To accept the benefits of the proposed approach, it is appropriate to not only compare the effort spent for pattern development, but also to consider reuse aspects. To demonstrate these capabilities, the typical effort spent for the tester pattern development process is compared with the traditional approach i.e VCD to STIL conversion especially for the benefits in case of device families. The comparison of the two approaches has been shown in table 5.1 in the form of pattern generation time.

Product		SoC1	SoC2
Application		An imaging vision processor	Modern automotive car radio system
Specifications		270 Memories, 4 PLLs, DDR PHY, 2 Thermal sensors, MIPS core	110 Memories, 3 PLLs, 10-bit SAR ADC, 18-bit MIC ADC, Audio DAC, Cortex ARM
Flop Count		∼5.5 lacs	∼4 lacs
Number of Patterns		15	10
Pattern Generation Time(weeks)	Traditional	8-10	6-8
	Proposed	2-3	1.5-2

Table 5.1: Comparison of Traditional and Proposed Approach

The environment for the implementation of the methodology has been shown in table 5.2.

Operating System	Red Hat Enterprise Linux v5.9
# of processing cores	10
Processor Speed(GHz)	2.80
Simulation Tool	Cadence NC-Sim
Scripting Language	Perl v5.8.8

Table 5.2: Simulation Environment

Due to development of library of STIL procedures for IPs, the advantage of pattern reuse can be exploited as can be inferred from the table 5.3. Even more significant, when it comes to generating patterns for a product family (having slightly different architectures), the pattern generation shows the real power of reuse as compared to conventional flow.

Product(SoC)	Common IPs	First-time Pattern Development Time (man-hours)	Pattern Re-use Time (man-hours)
SoC1 & SoC2	OTP Fuse, PLL, Compensation Cell, ADC, DAC	10-15	2-4

Table 5.3: Pattern re-use results for SoC

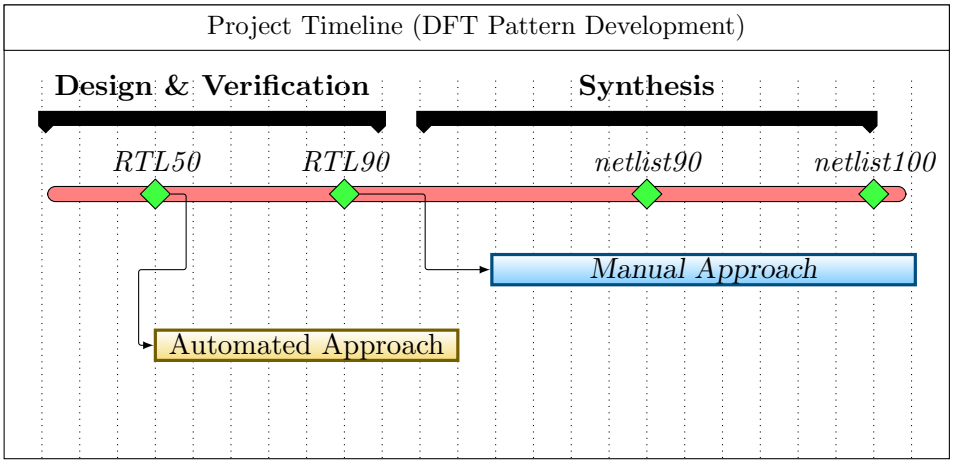

Figure 5.1: Pattern Development Timeline

Figure 5.1 shows the project timeline for the test pattern development. The manual approach of production pattern development activity could be commenced midway between RTL90 and netlist90 after Tetramax ATPG would generate scan patterns in STIL format. However, with the proposed automated TIM based flow, as soon as first verified RTL is obtained, pattern development activity can be initiated which will in-turn be consuming less time because of automation and re-usability.

Figure 5.2 shows a comparison of automated and manual approach in generating different sections of a test pattern in terms of manual effort involved. It can be inferred that pattern block takes the most time since in this block patterns are written vector-to-vector in cyclized form. This effort is reduced in the proposed approach since now most of the patterns in the block involve programming the JTAG registers which is done automatically after the test engineer enters the register configuration in the database.

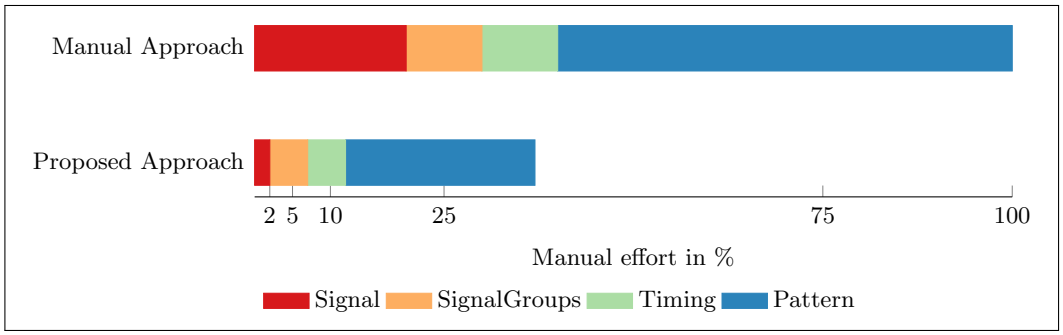

Figure 5.2: Distribution of different blocks in a test pattern

Signal block in conventional approach has been shown to exceed to that in proposed flow because of the waiting factor for the netlist to be generated for Tetramax ATPG to read and generate this block. Signal Groups and Timing blocks too have reduced manual effort in the proposed approach because test engineer need not spend time writing the appropriate syntax for these blocks. They can just enter the desired times with respect to the tester cycle in the database and tool will take care of the rest.

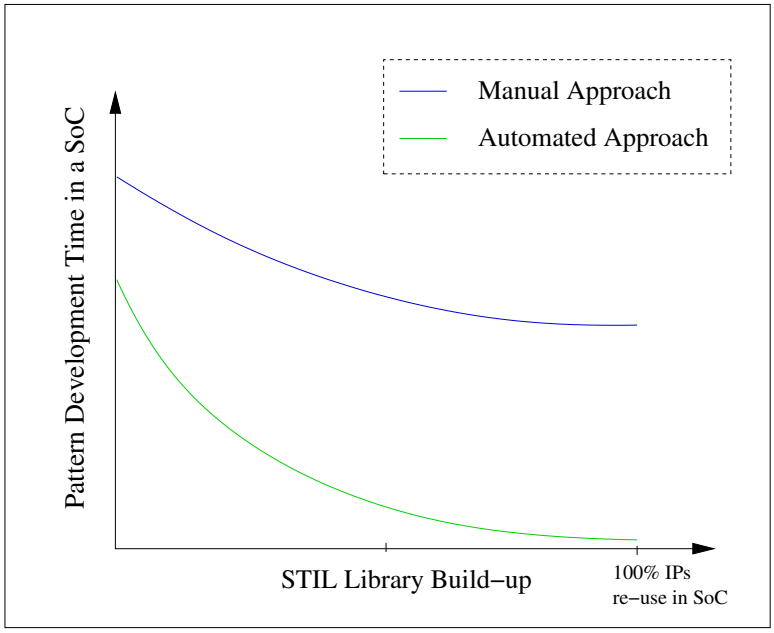

Figure 5.3: Development time with library build-up

Figure 5.3 shows the trend in declining pattern development time as the STIL procedure library is built with the addition of new IPs to be functionally tested. It can be observed that there is significant gain in test development time for automated over manual approach as the generic procedures for new IPs are developed. This time approaching to very little if all IPs in a SoC are being reused.

6

Conclusion and Future Work

6.1 Conclusion

As can be implied from the results, test pattern generation using the proposed flow reduces the test development time by approximately 60%-70% as compared to traditional simulation based flow. Also, test patterns are generated in cyclized format eliminating need of conversion tools for translating VCD obtained from event based testbenches to cyclized format. The flow is automated which makes the possibility of human error negligible consequently saving pattern verification time. When comparing with the manual approach of STIL based pattern development, this flow allows to create test patterns as soon as first verified RTL is available rather than waiting for the gate level design which is obtained after synthesis, required by ATPG tools to dump the initial STIL template consisting of Signal and SignalGroups construct. Thus, it can be concluded that there is a significant gain when we move from simulation based pattern development to STIL based pattern development for production tests. However, with the proposed automated flow, the pattern generation effort is smooth when compared with the manual handling of such and its availability also gets aligned with the first RTL maturity available for verification.

6.2 Future Work

The flow is limited to a set of devices having more or less common test architecture. There is a scope to support a wider range of products with different test architectures as well thereby making it more generic. Moreover, with new IPs to be functionally

tested, STIL procedure library has to be continuously updated to make this approach reusable for them.

Bibliography

[1] Ernst Aderholz, Heiko Ahrens, and Michael Rohleder. **Bridging the gap between Design and Test Engineering for Functional Pattern Development**. In *Test Conference, 2008. ITC 2008. IEEE International*, pages 1–10. IEEE, 2008. 1, 21

[2] Tony Taylor and Gregory A Maston. **Standard test interface language (STIL) a new language for patterns and waveforms**. In *2013 IEEE International Test Conference (ITC)*, pages 565–565. IEEE Computer Society, 1996. 1

[3] Michael Bushnell and Vishwani D Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, **17**. Springer Science & Business Media, 2000. vi, 3, 4

[4] **JTAG Technical Guide**. `http://www.xjtag.com/support-jtag/jtag-technical-guide.php`. 10

[5] **International Technology Roadmap for Semiconductors 2007**. `http://www.itrs.net/ITRSs/2007ITRS/2007_Chapters/2007_ERD.pdf`. ITRS, 2007. 14

[6] R. Rajsuman, R. Sauer, and H. Yamoto. **Architecture and design of universal IC test system**, November 20 2003. US Patent App. 10/228,845. 18, 21

[7] R. Rajsuman, S. Sugamori, R.F. Sauer, H. Yamoto, J.A. Turnquist, B.R. Parnas, and A. Le. **Event based IC test system**, August 8 2006. US Patent 7,089,135. 20

[8] R Raghuraman. **Simulation requirements for vectors in ATE formats**. In *null*, pages 1100–1107. IEEE, 2004. 20

[9] Prokash Ghosh, Celia John, Ajay Gupta, and Veerabhadrarao Siripurapu. **Enhancement of Production Pattern Development Methodology and Best Practices**. In *Electronic System Design (ISED), 2013 International Symposium on*, pages 153–157. IEEE, 2013. 22

[10] **Standard Test Interface Language**. `http://grouper.ieee.org/groups/1450/Flyer_frame.html`. 22

[11] Helmut Lang, Bhuwnesh Pande, and Heiko Ahrens. **Automating test program generation in STIL-expectations and experiences using IEEE 1450 [standard test interface language]**. In *Test Workshop, 2003. Proceedings. The Eight IEEE European*, pages 99–104. IEEE, 2003. 25

[12] Nastaran Nemati, Majid Namaki-Shoushtari, and Zainalabedin Navabi. **A mixed HDL/PLI test package**. In *Design & Test Symposium (EWDTS), 2010 East-West*, pages 518–523. IEEE, 2010. 25

[13] Sandip Kundu, TM Mak, and Rajesh Galivanche. **Trends in manufacturing test methods and their implications**. In *Test Conference, 2004. Proceedings. ITC 2004. International*, pages 679–687. IEEE, 2004. 31