A digital tester architecture for a system-on-chip implementation.

Rashid. Rashidzadeh

University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/598
A DIGITAL TESTER ARCHITECTURE FOR A SYSTEM-ON-CHIP IMPLEMENTATION

by

Rashid Rashidzadeh

A Thesis
Submitted to the Faculty of Graduate Studies and Research through Electrical and Computer Engineering in Partial Fulfillment of the Requirements for the Degree of Master of Applied Science at the University of Windsor

Windsor, Ontario, Canada
2003

Copyright © 2003 by Rashid Rashidzadeh
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.
This thesis presents the development of an Intellectual Property (IP) core for a System-on-Chip (SoC) implementation of an integrated circuit tester. The resulting realization is called a Tester-on Chip (ToC). The ToC IP core is used in conjunction with a microelectromechanical (MEMS) interface that provides the necessary connectivity between the tester circuitry and the Device Under Test (DUT). Instead of using traditional Automatic Test Equipment (ATE) that includes a complex external test head, the DUT is placed in a MEMS fixture or socket and spring loaded MEMS contacts are used to probe the DUT as required. The ToC implementation can generate and apply a comprehensive set of test vectors at-speed. The resulting test response information is analyzed by the ToC and the corresponding test results are sent via a Universal Serial Bus (USB) interface to a host computer, such as a laptop computer, for visualization and decision making. A scalable vector RAM is used to store the test vectors and it is held as a separate module in the MEMS interface socket. Both the DUT and scalable vector RAM are easily inserted into or removed from the MEMS socket using robotic or structured automation. The short MEMS-based connectivity paths between the ToC and the DUT reduce the transmission line effects normally associated with conventional test heads. By combining MEMS interfacing technology with SoC concepts it has been possible to develop the first CMOS intellectual property core with a capability that provides an alternative to conventional ATEs. A new tester architecture that works in conjunction with a MEMS test head has been developed in the form of a ToC IP core. Both a soft and firm realization of the
ToC IP core has been developed using Verilog-XL and Synopsys. A ToC implementation using 0.18 m CMOS technology has also been developed. Simulation studies have shown that the ToC can support 256 bi-directional test channels, each with a maximum test clock cycle of 100 MHz.
To my family for supporting me in my educational pursuits and for their encouragement
ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Dr. W. C. Miller, whose expertise, understanding, and patience, added considerably to my graduate experience. I appreciate his vast knowledge and skill in many areas and his assistance in writing reports. I am particularly grateful for his continual positive and his ever direct guidance in my quest to explore and analyse the topic of my thesis.

Further gratitude goes toward the committee members Dr. Majid Ahamadi of Electrical and computer Engineering and Dr. Reza Lashkari of Industrial Engineering, for the assistance they provided at all levels of the research project and their patience in reading and commenting on the thesis. I must also acknowledge Till Kuendiger of the Research Center for Integrated Microsystems(RCIM) for all of his suggestions, computer and technical assistance throughout my graduate program.

The author would like to acknowledge the financial support provided by the Natural Sciences and Engineering Research Council (NSERC), the Micronet Network of Centers of Excellence\(^1\) and the Gennum Corporation\(^2\) which formed the basis for developing this research project.

\(^1\)www.micronetrd.ca
\(^2\)www.gennum.ca
# TABLE OF CONTENTS

ABSTRACT iii

DEDICATION v

ACKNOWLEDGEMENTS vi

LIST OF FIGURES ix

I INTRODUCTION 1
  1.1 Statement of the problem .......................... 1
  1.2 Thesis Objective and Challenges .................. 3
  1.3 Outline of the thesis ............................. 4

II BACKGROUND 6
  2.1 Basic Concepts .................................. 6
  2.2 Economical Importance of Testing ................ 7
  2.3 Fault Models .................................... 8
  2.4 Test Evaluation and Fault Detection .............. 15
    2.4.1 Test Set Generation .......................... 18

III DESIGN FOR TESTABILITY 22
  3.1 Overview ...................................... 22
  3.2 Built-in Self-Test .............................. 22
  3.3 Linear Feedback Shift Register(LFSR) ............. 25
    3.3.1 LFSR Structure ............................. 25
    3.3.2 Polynomial Multiplier and Divider ............ 28
  3.4 Response Analysis .............................. 29

IV BOUNDARY SCAN 33
  4.1 IEEE 1149 Standard ............................. 33
    4.1.1 Bus Protocol ................................ 35
    4.1.2 Boundary Scan Cell .......................... 36
4.1.3 TAP Controller ................................. 37

V AUTOMATIC TEST EQUIPMENT 43
5.1 Overview ....................................... 43
5.2 Types of ATE .................................. 43

VI A DIGITAL TESTER ARCHITECTURE 47
6.1 Overview ....................................... 47
6.2 Core based SoC testing challenges .............. 47
6.3 Test Scheme .................................... 49
6.4 Tester-on-Chip .................................. 52

VII TESTER-ON-CHIP STRUCTURAL DESIGN 53
7.1 Tester-on-Chip Design Flow ....................... 53
7.2 Tester-on-Chip building blocks .................. 57
7.2.1 Pin-Electronics .............................. 57
7.2.2 High Speed Test Management: ................. 72
7.2.3 Interface .................................... 78

VIII SIMULATION AND RESULTS 80
8.1 Simulation of the ToC using Cadence and Synopsys tools ... 80
8.2 Test Flow ....................................... 82
8.3 Analysis Reports ................................ 83

IX CONCLUSIONS 86
9.1 Conclusions ..................................... 86
9.2 Future Works ................................... 86

REFERENCES 88
APPENDIX A — VERILOG CODES FOR THE TOC 94
APPENDIX B — LIST OF ABBREVIATIONS 104
APPENDIX C — VERILOG CODES FOR USB IP CORE 111
VITA AUCTORIS 124
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A Typical SoC Tester</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Stuck at Faults</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Stuck-on and Stuck-open</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>Functional Faults</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>Delay Fault</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>Memory Faults</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>AND Gate Test Evaluation System</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>General Test Evaluation System</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>Fault Detection</td>
<td>17</td>
</tr>
<tr>
<td>10</td>
<td>Stuck at Fault Detection</td>
<td>17</td>
</tr>
<tr>
<td>11</td>
<td>Fault Detectibility</td>
<td>18</td>
</tr>
<tr>
<td>12</td>
<td>Test Vector Generation</td>
<td>20</td>
</tr>
<tr>
<td>13</td>
<td>Typical Test Generation Flow</td>
<td>21</td>
</tr>
<tr>
<td>14</td>
<td>BIST Architecture</td>
<td>23</td>
</tr>
<tr>
<td>15</td>
<td>Exhaustive Test Hardware Structurer</td>
<td>24</td>
</tr>
<tr>
<td>16</td>
<td>Pseudoexhaustive Test Pattern Generation Circuit Structure</td>
<td>25</td>
</tr>
<tr>
<td>17</td>
<td>Two Types of LFSRs</td>
<td>26</td>
</tr>
<tr>
<td>18</td>
<td>LFSR with Characteristic Polynomial</td>
<td>26</td>
</tr>
<tr>
<td>19</td>
<td>LFSR Pattern Generation</td>
<td>27</td>
</tr>
<tr>
<td>20</td>
<td>Structure of a Polynomial Multiplier</td>
<td>28</td>
</tr>
<tr>
<td>21</td>
<td>LFSR as a Multiplier</td>
<td>29</td>
</tr>
<tr>
<td>22</td>
<td>Structure of a Polynomial Divider</td>
<td>30</td>
</tr>
<tr>
<td>23</td>
<td>LFSR as a Divider</td>
<td>31</td>
</tr>
<tr>
<td>24</td>
<td>Signature Analysis Circuit Structure</td>
<td>31</td>
</tr>
<tr>
<td>25</td>
<td>MISR - Multiple Input Signature Register</td>
<td>32</td>
</tr>
<tr>
<td>26</td>
<td>Boundary Scan Architecture</td>
<td>34</td>
</tr>
<tr>
<td>27</td>
<td>Boundary Scan Circuitry in a Chip</td>
<td>34</td>
</tr>
<tr>
<td>Page</td>
<td>Section Title</td>
<td>Page</td>
</tr>
<tr>
<td>------</td>
<td>---------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>28</td>
<td>A Typical BS Cell</td>
<td>36</td>
</tr>
<tr>
<td>29</td>
<td>State Diagram of TAP Controller</td>
<td>38</td>
</tr>
<tr>
<td>30</td>
<td>Execution of EXTEST Instruction</td>
<td>40</td>
</tr>
<tr>
<td>31</td>
<td>Execution of INTEST Instruction</td>
<td>41</td>
</tr>
<tr>
<td>32</td>
<td>Execution of SAMPLE and PRELOAD</td>
<td>42</td>
</tr>
<tr>
<td>33</td>
<td>Typical In-Circuit Tester</td>
<td>44</td>
</tr>
<tr>
<td>34</td>
<td>MEMS Socket Based Test Scheme</td>
<td>50</td>
</tr>
<tr>
<td>35</td>
<td>Driver-Receiver for MEMS Based Test Head</td>
<td>51</td>
</tr>
<tr>
<td>36</td>
<td>Design Flow of Teser-on-Chip</td>
<td>53</td>
</tr>
<tr>
<td>37</td>
<td>Block Diagram of Teser-on-Chip</td>
<td>55</td>
</tr>
<tr>
<td>38</td>
<td>An Example of Testing</td>
<td>58</td>
</tr>
<tr>
<td>39</td>
<td>Pin-Electronics Block Diagram</td>
<td>59</td>
</tr>
<tr>
<td>40</td>
<td>Block Diagram of Timing Circuit</td>
<td>61</td>
</tr>
<tr>
<td>41</td>
<td>Stimulus Waveforms</td>
<td>62</td>
</tr>
<tr>
<td>42</td>
<td>Capture and Comparison Block Diagarm</td>
<td>64</td>
</tr>
<tr>
<td>43</td>
<td>High Speed Test Management</td>
<td>73</td>
</tr>
<tr>
<td>44</td>
<td>ToC Selftest and Boundary Scan Architecture</td>
<td>75</td>
</tr>
<tr>
<td>45</td>
<td>Input and outputs signals of the Interface</td>
<td>79</td>
</tr>
<tr>
<td>46</td>
<td>ToC Simulation Results</td>
<td>81</td>
</tr>
<tr>
<td>47</td>
<td>Layout of the ToC</td>
<td>81</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

1.1 Statement of the problem

Very large scale integration (VLSI) developed during the 1980s, has vastly increased the circuit density. According to the Semiconductor Industry Association (SIA) roadmap, feature size of chips reduces 30 percent every three years, the overall chip size increase is 12 percent per year and we have 58 percent more devices per year. The reducing chip size leads to the increase in the number of physical faults, this significantly increases the problem of testing to make sure that the logic is correct and there are no fabrication errors. The decrease in chip size is leading to the rapid increase in the adoption of system-on-chip (SoC) designs where Intellectual Property (IP) cores are integrated on a single chip.

The SoC testing problems facing traditional Automatic Tester Equipment (ATE) include:

1. Chip Density

   Chip densities increase rapidly and result in higher manufacturing defects and lower fault observability which further complicate the SoC testing process.

2. High Cost of ATE Equipment

   General purpose high-speed pin-electronics is actually the heart of ATE and its quality has major impact on ATE’s ability to test advanced devices. A good quality pin-electronics meeting SoC test requirements, dissipate a considerable amount of power and they can not be highly integrated. These limitations forced ATE designers to resort to costly test heads which are quite often cooled
down using advanced liquid cooling systems resulting in a high cost ATEs. A
typical SoC tester such as T6682(Figure 1) of Advantest\textsuperscript{1} costs more than two
million dollars and this amount increases by 3000USD for every additional test
pin.

3. Exploding Test Data

The test data per gate is approaching 1 kilobyte and million gate designs require
1 Gigabyte of test data. The patterns intended to test a particular device-under-
test(DUT) need to be passed through a number of levels in order to reach the
DUT and the results have to again propagate back to an observable point. The
volume of test data and the capacity of Random Access Memory(RAM) to store
them are becoming huge and lead to excessively long test application time.

\textbf{Figure 1:} A Typical SoC Tester

All these bring forward the limitations of traditional ATE and point to the need
for having a different approach. The answer to SoC testing is not bigger testers,
the key to testing system-on-chip technology will be new architectures that provide

\textsuperscript{1}www.advantest.com
great flexibility. The next generation of tester architectures needs to be able to resolve several problems, such as testing high speed mixed signal devices, high pin count of device under test and very limited observability and controllability. As chip technology becomes more complex, it is believed the ATE industry will need to make functional protocol testing a major goal of system-on-chip testing. The new generation of semiconductor testers has to be sophisticated enough to do functional protocol testing of complex integrated ICs while continuing to drive down the cost of test. A fresh approach is needed to explore new solutions to the problems of system-on-chip testing.

1.2 Thesis Objective and Challenges

This thesis propose a new architecture for SoC testing based on a tester-on-chip (ToC) and a microelectromechanical (MEMS) fixture. With this new architecture two of the main challenges of SoC testing are addressed. First, the proposal eliminates the necessity of advanced, high price ATE to carry out SoC testing and replaces it with an ordinary controller. Second, the architecture supports flexible Vector RAM therefore high volume test data is supported without limitations.

The main objective of this project was to design a digital tester-on-chip IP core performing typical test operations on digital microelectronics, meeting SoC test requirements associated with hearing instruments. The ToC IP core can be part of a system-on-chip integrated solution or it can function in an autonomous manner. The main challenges of the project were:

- **Efficient Pin Electronics**
  
  To perform a successful SoC testing, high-performance pin electronics are needed to apply stimulus data to the DUT and to capture its response meeting timing constraints of device under test. This requirements should be met under severe area constraints. Any area overhead at this part is multiplied by the
total number of test channels and results in a considerable area overhead of final outcome.

- **Timing and Format Requirements**
  It was obvious from the very beginning of this project that timing part of the ToC should be precisely programmable to meet complicated timing requirements of advanced IP cores. This part of the ToC is actually the heart of the ToC system and should be designed very carefully.

- **Control and Management Circuits**
  How to design a controller to supervise at-speed SoC testing while generating necessary signals to activate or disable different modules of ToC, was a major challenge. There were different option such as using a microprocessor, sequencer, RISC processor, etc. each one with its own advantages and disadvantages.

### 1.3 Outline of the thesis

This thesis is organized as follows. Chapter 2 introduces the basic concepts of VLSI device testing, starting with the definition of test and describing various test methods and fault models and going through Automatic Test Pattern Generator (ATPG) for VLSI circuits. Chapter 3 begins with Design For Test and build-in self-test methods and proceeds with linear feedback shift register (LFSR) and multiple input shift register (MISR). Chapter 4 will focus on Boundary Scan and continue with test access port (TAP). Chapter 5 will present various automatic test equipment (ATE) and the their method of testing Chapter 6 proposes a new architecture for testing a core based system-on-chip (SoC). The novel feature of the approach and advantages of using MEMS fixture is presented. Chapter 7 will focus on the design flow of the ToC and its modules and submodules. Register
Transfer Level(RTL) codes representing various ToC modules are presented in this chapter. Chapter 8 begins with simulation results, proceeds with test flow and ends with Synopsys analysis reports.
CHAPTER II

BACKGROUND

2.1 Basic Concepts

Testing a system means evaluating whether it works correctly. This is normally done by applying some input stimuli, and observing the produced outputs. The tester records the actual responses measured at circuit outputs, and any differences between the observed responses and the expected responses are recorded as a fault. If a fault is found, diagnosis aims at locating the cause of the fault.

An Error is an incorrect behaviour of the system with respect to the reference one. Errors can be classified according to their cause:

- Design errors such as incorrect design specifications, violation of design rules, wrong implementation.

- Fabrication errors: they can vary depending on the kind of system considered (e.g. ASIC, board, FPGA).

- Physical Failures, which alter the system behaviour during its lifetime.

Depending on their characteristics, physical faults can be grouped in the following categories:

- **Permanent**: the fault appears at a given time and from that time on, it permanently affects the system or device.

- **Intermittent**: the fault appears at a given time, then disappears after a while, possibly reappears later, and so on.
• **Transient**: the fault appears, lasts for a given time, and then disappears. Physical faults are connected with the characteristics of the system they affect, with the process adopted for its production, and with the environment it works in. In order to simplify fault analysis, logical faults are introduced which models the effects of one or more physical faults on the system behaviour, and is normally more easily managed.

VLSI testing is a difficult and costly process therefore some methods of Design For Test (DFT) should be taken into account during the design phase of VLSI devices, otherwise test coverage and cost of testing would approach an unacceptable level. For instance, to test a common combinational circuit with 50 inputs, we need around $10^{15}$ test patterns to test the circuit. Assume that each test application takes $10^{-10}$ second, it will take 3.57 years to test such a circuit. Test generation of sequential circuits are much harder because of lower controllability and observability of Flip-flops or latches. Complete test of a simple sequential circuit becomes almost impossible. On the other hand, the designers have functional vectors for each circuit to verify their designs against the specification. However, these functional test sets may not be able to detect the physical faults caused in the manufacturing processes.

### 2.2 **Economical Importance of Testing**

The rapid growth in the complexity of digital circuits and systems, and the demand for an always higher product quality raised the cost of testing. Developing an effective test strategy is now part of the design process and performing a suitable test, from the point of view of both its cost and its effectiveness, is a key issue in the success of any electronic system.

For instance, the average cost for testing ASICs rose from about 30 percent of the total production cost in the early 90s, to about 50 percent in 1998. This cost includes:

• Finding a test strategy which is normally done at the very beginning, and during
the design process

- Evaluating the effectiveness of the selected strategy, which is done during, and at the end of the design process

- Performing the test, which is done during the manufacturing process, and sometimes during the life of the electronic system.

The cost of the test process includes:

- The cost in terms of designer time, when the test strategy includes some design for testability technique.

- The cost in terms of tools for generating test stimuli and verifying their effectiveness, and the time for their use.

- The cost in terms of equipment required to apply test stimuli and verify the device behaviour.

### 2.3 Fault Models

The choice of the best fault model is crucial, since it strongly affects both the cost and the coverage of the test process. The choice depends on the abstraction level the system is described at, when the test is developed. The level ranges from the lowest (electrical, or transistor) to the highest (behavioural).

Fault, Failure, and Error are among the terms that are frequently used in test environments. Although sometimes they are used instead of each other they have different meanings. A fault is a physical defect of one or more components. A fault may or may not cause a system failure right away. An error is a sign of some faults that result in incorrect circuit outputs or states which can be observed. A failure is a variation of a circuit or system from its behavior which means that the circuit or system fails to do what it should do. Basically, some physical faults can cause errors and some
errors will cause the failure of circuits or systems.

There are many possible physical defects during the manufacturing process. Some examples of physical defects are: Silicon defects, mask contamination, process variations and defective oxide. Physical defects can cause electrical effects such as Shorts, opens, transistor stuck-on/open, resistive shorts and opens, threshold voltages variation. The effects on the electrical level are easier to measure compared to those on the physical level. Physical defects can also cause logic effects, such as logic stuck-at-0/1, slow transition, bridging.

Some representative fault models are introduced as follow.

1. **Single Stuck-at Fault**

   It is the most popular fault model for digital circuits when they are modelled as a netlist of gates. It corresponds to a line whose status is stuck at 0 or 1. A single stuck-at fault models several possible physical faults. The stuck-at fault model is easily manageable by algorithms and CAD tools.

   Figure 2 shows that there is a stuck-at-0 fault with one of the inputs of the OR gate. By applying an appropriate input vector such as 0111, we can trigger the fault at the faulty line. In the fault-free situation, the value of that line should be 1, while in the faulty situation, the value of that line should be 0. To observe the fault, we have to set the other input of the OR gate to 0 to allow the faulty effect propagating to the output terminal. The true response at the output should be 1, while the faulty response at the output should be 0. This fault model has the following advantages:

   - Many different physical defects may be modeled by the same single stuck-at faults so that the complexity of test generation is greatly reduced.
   - Single stuck-at fault test set can detect a large percentage of multiple
Figure 2: Stuck at Faults

- It is independent of technology and design style. Therefore it can be applied to different technologies such as TTL, ECL, CMOS and also it can be applied to various design styles such as Gate Arrays, Standard Cells, VLSI ICs.

2. Multiple Stuck-At Faults
The multiple stuck-at fault models assume that there are several stuck-at faults occurring at the same time. It actually models the real faulty situation especially for VLSI designs. The number of multiple stuck-at faults of any real circuits will be extremely large.

3. Bridging Faults
The bridging faults occur when two or more normally separate points or lines are shorted together. The logic effects of these faults depend on the technology the circuits used. In TTL, the logic effect is wired-AND. In ECL, the logic effect is wired-OR.

4. Stuck-on and Stuck-open
This fault model is widely used when dealing with circuits described at the
transistor level. A single transistor is stuck at the on or open status under this model. In static CMOS technology, when a transistor sticks on, the logic value of the output could be unknown. As shown in Figure(3a), when the n transistor sticks ON and the input is 0, there is an electrical path going from VDD to ground through both the p transistor and the n transistor. The logic value of

![Figure 3: Stuck-on and Stuck-open](image)

this situation depends on the relative impedances of the pull-up and pull-down networks. Because both the p and n transistors are conducting, this will cause increased quiescent current, called IDDQ fault.

In static CMOS networks, the transistor stuck-open may cause output floating. When the p transistor sticks open and the input is low, the output floats and keeps the previous value. Therefore, the transistor stuck-open fault can turn the circuit into a sequential circuit. In order to detect this kind of faults, two-vector tests are required. To detect the stuck-open fault of the p transistor shown in Figure(3b) the first vector is to initialize the output to the opposite value 0 through the normal n transistors. The second vector is to trigger the fault by forcing the faulty p transistor to conduct. If the p transistor fails to conduct, the faulty output will be 0 which is different from the fault-free output
value 1, thus the stuck-open fault of the p transistor is detected. If we care about both the stuck-at and stuck-open faults in CMOS circuits, the random patterns may not achieve enough fault coverage. The Stuck-at faults happen more often than the stuck-open faults. 91% of chips with stuck-open faults may also have stuck-at faults. In other words, if we use the test set of stuck-at faults and ignore the stuck-open faults, we still can detect those faulty chips. There are only about 0.12% chips with stuck-open faults escape the tests for stuck-at faults. Based on these figures, we don’t really need to worry about stuck-open faults in CMOS even though we know that this kind fault does exist.

5. **Single State Transition Fault**

This fault model is sometimes used when dealing with circuits implementing Finite State Machines (FSMs) and only the state transition graph is known and the gate-level implementation is not yet known. This fault model is an example of functional fault model. The fault causes a transition to a wrong state by changing either the origin or destination state, or the associated input value.

6. **Functional Faults**

Functional faults is to model the faults at a higher level than logic for function modules such as decoders, multiplexers, adders, counters, RAMs and ROMs. Functional test is usually developed in the early stage of a design, before the final gate level or transistor level is implemented. Basically the idea of functional test is based on taking advantage of the functional information so that some physical defects could be detected. Assume a 2-to-4 decoder shown in Figure(4) as an example. There are three typical functional faults.

(a) Instead of ith line, jth line is selected.

(b) In addition of ith line, jth line is selected.

(c) None of the lines are selected.
7. Delay Faults

There are defects on the chip which allows it to pass the DC or low frequency stuck-at fault testing, but causes it to fail when operated at system speed. For example, a chip may pass testing under 1 MHz operation but not under 10 MHz. There are typically two kinds of delay fault models used to generate the test set. Gate-Delay-Fault is to model defects which cause some gate to rise or fall too slow. An example shown in Figure(5) indicates that an unusual high resistance can cause a gate output to rise too slow. The Gate-Delay-Fault model has an obvious disadvantage. That is the delay faults resulting from the sum of several small incremental delay defects may not be detected. However, the failure of the circuit in terms of performance is usually due to the long delay
on some paths. The Path-Delay-Fault is to model the propagation delay of the path that exceeds the clock interval. However, the disadvantage of this model is that the number of path-delay faults grows rapidly with the number of gates.

8. Memory Faults

The testing of memories is not only to detect the functional faults, but also to detect the parametric faults. The parametric faults are to model the faulty operations in terms of the following concerns: output level, power consumption, noise margin etc. There are different kinds of functional faults associated with memory:

(a) the stuck-at faults in address registers, data registers, and address decoders
(b) the cell stuck-at faults
(c) the adjacent cell coupling faults
(d) the pattern-sensitive faults

The last two kinds of faults are unique to the memory structures and are very different from the other functional faults. As shown in Figure(6)

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & d & b \\
0 & a & 0 \\
\end{array}
\]

\[a=b=0 \Rightarrow d=0\]
\[a=b=1 \Rightarrow d=1\]

**Figure 6:** Memory Faults

The value of d depends on the values of its neighbors a and b. The rest of the neighbors of d should have the values of 0. The presence of a faulty signal depending on the signal values of the nearby points is called the pattern-sensitive faults. The test sequences of the memory testing can be derived without much
difficulty for each specific fault. However, the length of the test sequences can be unacceptable. For an n-bit RAM, the length of the test set for pattern-sensitive faults is \(5n^2\). Assume the test application time per test is 10ns, it will takes 14 hours to test a 1-M bit memory. For a 64-M bit chip, it will take 6 years.

9. **Specific Fault Models**

Specific fault models can be used in many cases to address the characteristics of the device or system which is under consideration.

For example, when testing RAM circuits, the coupling fault model assumes that the value of a bit depends on the values of the bits in the surrounding memory cells.

As an example, when considering microprocessor circuits, and only their behaviour is known, functional fault models are assumed, according to which the different units behave in some incorrect way for instance the decoding unit decodes one instruction instead of another.

### 2.4 Test Evaluation and Fault Detection

Once the fault model has been chosen, a common problem is the evaluation of how good a given input sequence is in detecting faults.

As an example, consider a simple circuit composed of a single AND gate. There are six possible stuck-at faults for this circuit, a/0 (stuck-at 0), b/0, U/0, a/1, b/1, and U/1. Four combinations can be applied to the circuit inputs a and b. The table above records for each input combination the value assumed by the circuit output U when no fault exists in the circuit (Ug), as well as the value assumed when one of the six possible faults exists (Uf). As can be seen, the input combination 00 causes the circuit output to be different than the expected one when the stuck-at-1 fault affects the line U; in this case, U/1 is detected by the input vector 00. Similarly, the input vector 01 detects the faults a/1 and U/1. It can be verified that each fault is detected
by at least one vector among the four. In the example above, the Fault Coverage is clearly 100%. The tools for the evaluation of the behaviour of a faulty system when a given set of input patterns are applied are known as Fault Simulators. The block diagram of a typical test evaluation system is presented in Figure 8.

A test is said to detect a fault in a circuit if we apply the input test vector to the circuit and the output of the circuit is different from that of fault free circuit. In Figure 9, there is a stuck-at-1 fault appearing at one of the inputs of the AND
gate. When we apply the test 001, the outputs of fault-free and faulty circuits are

\[ \begin{align*}
x_1 & \quad z_1 \\
x_2 & \quad z_2 \\
x_3 & \quad z_2 \end{align*} \]

\[ \begin{align*}
z_1 &= x_1 \cdot x_2 \\
z_2 &= x_2 \cdot x_3 \\
z_1 &= x_1 \\
z_2 &= x_2 \cdot x_3 \end{align*} \]

**Figure 9:** Fault Detection

different. So the test 001 detects the fault. A test vector must satisfy two conditions to detect a fault. In Figure 10, there is a stuck-at-1 fault which can be detected by 1011 pattern as input vector, since it activates the fault by creating different values than that of fault free situation.

**Figure 10:** Stuck at Fault Detection

A fault is said to be detectable if there exists a test to activate it and to propagate its effect to an observable point. For an undetectable fault, the fault-free and the
faulty values remain the same under any input vector. In other words, there is not an input vector which can activate fault and create a sensitized path to at least a primary output. The stuck-at-0 fault shown in the example in Figure 11 is undetectable since,

![Diagram](image)

**Figure 11:** Fault Detectibility

there is no input vector to force the output of the circuit to have different values under faulty and fault-free conditions. This means that the existence of this fault does not change the function of the circuit. So the related circuit is redundant and can be deleted to simplify the circuit. It can reduce the circuit area, improve the performance and testability of the circuits.

2.4.1 Test Set Generation

The complete detection test set is defined as a set of tests that detect any detectable faults in a class of faults. The quality of a test set is measured by its fault coverage. The fault coverage is usually determined by fault simulation. The fault coverage is defined as the fraction of the modeled faults detected by a test set. Typically, fault coverage of better than 95% is required in many semiconductor companies in terms of
stuck-at fault model. The test set with higher fault coverage will on average guarantee a better quality of the products.

When a given Fault Coverage is required, there is often the need for a method, which can generate a test to detect all the considered faults. As an example, a simple combinational circuit (Figure 12) is considered and described at the gate level. In order to detect a given stuck-at fault, an input vector must be able to generate on at least one of the circuit output lines a value different than the one existing on the same line in the good circuit. This means that the vector must satisfy two conditions:

- First, it must be able to put the circuit in a state, in which the value of the line where the fault is located is opposite to the one imposed by the fault itself. In this way, a difference is created in the circuit between the fault-free and the faulty case.

- Secondly, it must be able to propagate the difference towards at least an output line, so that the difference can be seen from the outputs.

If a test vector for the fault y/0 has to be found for the circuit below, one should first look for the input vectors able to force a 1 on the line y. These input vectors correspond to the cube abc = XX0. The propagation phase requires that the line x holds the value 0, which means that the condition is satisfied by the cubes abc = 0XX and abc = X0X. Therefore, the vectors that satisfy both conditions are 000, 100, and 010, which represent the set of input vectors able to detect the fault y/0.

The problem of generating a test vector for a stuck-at fault in a gate-level netlist is a serious problem whose complexity grows exponentially with the size of the circuit. The software tools addressing this problem are known as ATPGs (Automatic Test Pattern Generators). ATPG systems are normally composed of several modules:

- A module for the generation of the Fault List
Figure 12: Test Vector Generation

- A module for the generation of a test vector (or sequence) for a given fault
- A module performing the Fault Simulation.

The usual behaviour of an ATPG system can be summarised in the following way:

1. The fault list is generated; all the faults are marked as untested
2. A fault F is selected from the set of untested faults in the fault list
3. A test pattern T is generated, able to detect F
4. Simulate the fault with the test to determine whether there are other faults that are detected. This process can reduce the test generation time because the number of target faults in step 1 will be reduced. If a test pattern T has been generated, it is fault simulated, and all the faults it detects are marked as tested
5. If the stopping condition is not verified, the process repeats from step 2.

The stopping condition can be one of the following:

- item A maximum CPU time has been spent.
• item A given fault coverage has been reached.

Figure 13 shows the typical test generation flow. Sometime, the fault simulation is performed after we collect a set of related tests rather than after each test at the same time.

Figure 13: Typical Test Generation Flow
CHAPTER III

DESIGN FOR TESTABILITY

3.1 Overview

The complexity of current circuits often makes impossible the straightforward approach of dealing with test generation only at the end of the design process. In fact, for most circuits the Automatic Test Pattern Generator (ATPG) systems which are currently available are simply not able to generate a test set with a reasonable fault coverage and with reasonable CPU time requirements.

Therefore, an alternative approach became popular in the last few years, based on design for testability (DFT) during the design phase. According to this approach, test issues must be considered early in the design process, and some techniques have to be applied to simplify the test generation phase at the end of the design process.

During the early design steps of the ToC several DFT methods were employed. All flip-flops were replaced with scannable flip-flops, boundary scan chain created and a LFSR designed to generated on chip test pattern.

3.2 Built-in Self-Test

The basic BIST architecture is composed of three hardware modules in addition to the device under test. The architecture is shown in Figure 14.

The functions of these blocks are as follows. The test pattern generator generates the test patterns for the DUT. The response analyzer compresses and analyzes the test responses to determine correctness of the DUT. The BIST controller is the central unit to control all the BIST operations. The design of a test generator is determined by the test strategy being deployed. The test strategy being selected is determined
by the fault coverage, test hardware overhead, and testing time. The commonly seen test strategies include the followings.

1. **Stored-pattern**

   Stored-pattern approach stores the pregenerated test patterns to achieve certain test goals. It is often found in system level testing such as the power-on self test of a computer and microprocessor testing using microprogram. The test procedure is as follows. We pergenerate the test pattern by an Automatic Test Pattern Generator (ATPG) and store the pattern on chip or board. When BIST is activated, we apply them to the device under test and compare the responses with the corresponding stored responses. Because of the stored data size is limited, this method is attractive only in limited situation. This include the testing of regular circuits and the handle of hard to detect faults. For regular circuits like Programmable Logic Arrays (PLAs), only a very small amount of test patterns are required to achieve a very high fault coverage. For stored pattern approach, read only memories (ROM) are used to store test patterns and test responses.

2. **Exhaustive Testing**
Exhaustive testing applies all possible input combinations to the DUT. It guarantees that all detectable faults that do not produce sequential behavior will be detected. The faults that produce sequential property may not be detected because the test sequence may not contain the test sequence for that particular faults. Since all possible inputs combinations are applied, it is also regarded as the complete function testing. The hardware structure for exhaustive testing is very simple, as shown in Figure 15. The order of LFSR must be the same as the number of inputs of the DUT. The test cycle include all possible input combinations.

![Exhaustive Test Hardware Structurer](image)

**Figure 15:** Exhaustive Test Hardware Structurer

3. **Pseudoreexhaustive Testing**

Pseudoreexhaustive testing partitions the DUT into several smaller subcircuits and tests each of them exhaustively. The main goal of pseudoreexhaustive test is to obtain the same fault coverage as the exhaustive testing and, at the same time, minimize the testing time. This method requires extra design effort to partition the circuits into pseudoreexhaustive testable subcircuits. Circuit partitioning for pseudoreexhaustive testing can be done by cone segmentation. To generate pseudoreexhaustive test, as shown in Figure 16, we can use a LFSR and a shift register. The length of LFSR is usually greater than the size of the largest cone. Usually, at least two seeds are required. The number of test patterns generated is near minimal when the size of the cone is much smaller than the total number
of inputs. If LFSR has the shift mode, the seeds can be shifted in through scan chain.

![Figure 16: Pseudoexhaustive Test Pattern Generation Circuit Structure](image)

### 3.3 Linear Feedback Shift Register (LFSR)

LFSR is the most important and popular BIST hardware module. It has the following advantages. First it has a very simple and regular structure which minimize the hardware overhead and design effort. Second, it is able to generate random test patterns. Third, it can also be used to compress test responses with a very low aliasing probability. Fourth, its shift property is easily integratable in a DFT scan environment hence the hardware overhead is minimum when upgrading from a scan DFT design.

#### 3.3.1 LFSR Structure

Two examples of LFSRs are shown in Figure 17, both versions use D type flip-flops and logic gates (exclusive-or gates) to realize LFSRs.

The basic differences in these two structures are as follows. The external type LFSR puts XOR gates outside the shift path. It is also called type 1 LFSR. The internal type LFSRs, also called type 2 LFSR, puts XOR gates in between the flip-flops. These two types are equivalent in the sense that one-to-one correspondence...
between these two structure can be derived. The location of the XOR gates in Figure 17 is determined by characteristic polynomial. The general form of a characteristic polynomial is shown in equation 1.

![Two Types of LFSRs](image)

**Figure 17:** Two Types of LFSRs

$$g(x) = g_nx^n + g_{n-1}x^{n-1} + g_0x^0$$  \hspace{1cm} (1)

The relationship between the characteristic polynomial and the circuit structure of both types is shown in Figure 18 which shows a LFSR with characteristic polynomial $g(x) = g_nx^n + g_{n-1}x^{n-1} + K + g_0x^0$

![LFSR with Characteristic Polynomial](image)

**Figure 18:** LFSR with Characteristic Polynomial

The basic building blocks of LFSRs are D-type flip-flop, XOR gates, and connections. For the characteristic polynomial of $x^4 + x^3 + 1$, the LFSRs are shown in
One of the most important properties of LFSRs is their recurrence relationship. The recurrence relation guarantees that the states of a LFSR are repeated in a certain order.

\[
G(x) = \frac{\sum_{i=1}^{N} g_i x^i (a_{-i} x^{-i} + \ldots + a_{-1} x^{-1})}{g_x}
\]  

(2)

\(G(x)\) is the pattern being generated, \(a_{-i}\) is the initial state of the flip-flop \(D_i\). If the initial state is (0001), equation 2 generates \(G_x = \frac{1}{g_x}\). An example of the recurrence relation is shown in Figure 19. The initial state of the LFSR is (1000). The initial state repeats after \(2^4 - 1\) cycles. The recurrence cycle contains all possible combinations except all zeros (0000).

\[
g(x) = x^4 + x^3 + 1
\]

\(D_3\) \(D_2\) \(D_1\) \(D_0\)
\[\begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 \\
\end{array}\]

**Figure 19:** LFSR Pattern Generation

The sequence generated by an n-stage LFSR has a period of \(2^n - 1\) which is called a maximum-length sequence (M-sequence). A M-sequence contains all possible combinations except all zeros. The characteristic polynomial associated with the M-sequence is called a primitive polynomial. The patterns generated by a primitive polynomial satisfies the following randomness properties. First, the number of 1s in a M-sequence differs from the number of 0s by one. Second, a M-sequence produces
an equal number of runs of 1s and 0s. In every M-sequence, one half the runs have length 1, one forth have length 2, one eighth have length 3, and so forth. Here, a run of 1s of length 3 is (111). For instance, there are two runs of (111) in the serial patterns generated at D3, D2, D1, or D0 locations. The run is not limited to the consecutive ones. It can be any combinations.

### 3.3.2 Polynomial Multiplier and Divider

A LFSR can be used as a polynomial multiplier. Figure 20 shows the circuit diagram of a LFSR being used as a polynomial multiplier.

![Circuit Diagram](image)

\[
\begin{align*}
e(x) &= \alpha x^n + \alpha_{n-1} x^{n-1} + \ldots + \alpha_0 x^0 \\
\text{Input} &= f(x) \\
\text{Output} &= h(x) \\
h(x) &= e(x) \cdot f(x)
\end{align*}
\]

**Figure 20:** Structure of a Polynomial Multiplier

Initially, the state of D flip-flops are set to 0s. The presence of the input bit \( f_k \) is multiplied by the characteristic polynomial \( g(x) \). The result \( f_k g(x) \) is added into the register. When the clock comes, the LFSR is shifted one position to multiple current result by \( x \). At the moment, the coefficient whose calculation has completed is shifted to the output. Such a shift and add operations repeats until all the input bits are processed. Figure 21 shows a polynomial multiplication done by a LFSR. As can be seen, the process is the same as the long multiplication by hand.

A LFSR can be used as a polynomial divider. Figure 22 shows the circuit diagram of a LFSR being used as a polynomial divider.
(x^4 + x^3 + 1) \cdot (x^3 + x + 1) = x^7 + x^6 + x^4 + x^2 + 1

Figure 21: LFSR as a Multiplier

The presence of the input polynomial M(x) is divided by the characteristic polynomial g(x). The initial state of D flip-flops are set to 0s. After n shifts, the quotient bit q_i appear at the output. The q_i is multiplied by g(x) and subtracted from the register. When the clock comes, the LFSR is shifted one position to produce a quotient bit and lower the order by one. Such a shift and subtract operations repeats until all the input bits are processed. Figure 23 shows an example of the polynomial division done by a LFSR.

3.4 Response Analysis

Depending on how the responses are compressed, the techniques are classified as follows.

- **Signature Analysis:** Signature analysis is a compression technique based on LFSR. The circuit structure for the signature analysis is shown in Figure 24.

  The output sequence (polynomial) is divided by the characteristic polynomial.
\[ \varepsilon(x) = \varepsilon_n x^n + \varepsilon_{n-1} x^{n-1} + \ldots + \varepsilon_0 x^0 \]

Input = \( M(x) \)  \hspace{1cm} Remainder = \( R(x) \)

Output = \( O(x) \)  \hspace{1cm} \( M(x) = P(x) \varepsilon(x) + R(x) \)

**Figure 22:** Structure of a Polynomial Divider

The remainder of the division is called the signature as shown in Figure 23. The input sequence (110110110) is compressed into a signature of (11001). For an output sequence of length \( m \), there are a total of \( 2^m - 1 \) faulty sequence. Suppose the input sequence \( P(x) \) is represented by equation 3

\[ P(x) = Q(x)G(x) + R(x) \]  \hspace{1cm} (3)

\( G(x) \) is the characteristic polynomial; \( Q(x) \) is the quotient; and \( R(x) \) is the remainder or signature. For those aliasing faulty sequence, the remainder \( R(x) \) will be the same as the fault-free one. Since, \( P(x) \) is of order \( m \) and \( G(x) \) is of order \( n \), hence \( Q(x) \) has an order of \( m-n \). Hence, there are \( 2^{m-n} \) possible \( Q(x) \) or \( P(x) \). One of them is fault-free. Therefore, the aliasing probability is shown as follows.

\[ P_{sa} = \frac{2^{m-n} - 1}{2^m - 1} \]  \hspace{1cm} (4)

The aliasing probability of signature analysis is independent of the test responses. The aliasing probability can be reduced by increase the length of LFSR. According to the characteristics of polynomial field, signature analysis by LFSR has the following properties. Figure 24 shows the hardware structure for a single-output LFSR. For multiple output circuits, there is a multiple-input signature register or MISR. Figure shows the circuit structure of two MISRs.
The multiple input bits are from the top of the MISRs. MISR share the same properties as LFSR for single-input signature analysis.

- **Ones Count**: Ones count counts the number of 1s in the test response sequence to determine the correctness of the circuits. This is a very simply way of compression. It only requires a simple counter to accomplish the goal. The order of the test patterns being applied can be changed.

- **Transition Count**: Transition count counts the number of transitions in the
test response sequence to determine the correctness of the circuits. It is compatible in complexity with the ones count method. However, the test sequence cannot be changed.

- **Syndrome Count:** Syndrome count counts the probability of 1s in the test responses to determine the correctness of the circuits. It is very similar to the ones count except that the patterns need not be fixed.
CHAPTER IV

BOUNDARY SCAN

Boundary Scan (BS) may be the most successful test methodology in the 90’s. Initially motivated by the board or system level testing, this methodology is now widely accepted in industry and has been considered as an industry standard in most large IC system designs. BS has been successfully incorporated into designs and has greatly simplified the test or even diagnosis problems in advanced CPUs, military systems, consumer electronics, satellite or broadcasting systems, etc. The available computer aided design (CAD) tools were used to create boundary scan chain and test access port (TAP) for the tester-on-chip.

4.1 IEEE 1149 Standard

Boundary scan is in fact a family of test methodologies aiming at resolving many test problems: from chip level to system level, from logic cores to interconnects between cores, and from digital circuits to analog or mixed-mode circuits. The IEEE 1149.1 Standard, usually referred to as the Digital Boundary Scan, is the one which has been used most. It can be divided into two parts: 1149.1a, or the digital Boundary Scan Standard, and 1149.1b, or the Boundary Scan Description Language (BSDL). The IEEE 1149.1 Standard defines the chip level test architecture for digital circuits, and 1149.1b is a hardware description language used to describe boundary scan architecture.

The testing of mixed-mode circuits at the various levels of integration will be a critical test issue when the system-on-a-chip design becomes a reality. Therefore there is a demand to combine all the boundary scan standards into an integrated one.
The IEEE 1149.1 Standard, which mainly concerns how to incorporate some control logic and test registers into a chip to support the testing for the internal logic as well as the interconnects among chips. Figure 26 shows a chip containing the boundary scan architecture. The internal logic is the original circuit before the 1149.1a architecture is inserted. The name of boundary scan is due to the insertion of a boundary scan cell to each I/O pin of the original circuit and these cells are chained into a boundary scan path. In Figure 27 the boundary scan circuitry in a chip is extracted and we can divide the IEEE 1149.1a into four main hardware elements:

- a set of 4 mandatory (TDI, TDO, TMS and TCK) and 1 optional (TRST) Test
Access Ports (TAPs),

• a TAP controller

• an Instruction Register (IR) and its associated decoder,

• Test Data Registers (BS registers, Bypass register, Device-ID register).

The test access ports, which define the bus protocol of boundary scan, are the additional I/O pins needed for each chip employing Std.1149.1a. The TAP controller is a 16-state final state machine that controls each step of the operations of boundary scan. Each instruction to be carried out by the boundary scan architecture is stored in the Instruction Register. The various control signals associated with the instruction are then provided by a decoder. Several Test Data Registers are used to stored test data or some system related information such as the chip ID, company name, etc.

4.1.1 Bus Protocol

The Test Access Ports (TAPs) provide access to the test function of the IC between the application circuit and the chip’s I/O pads. The TAPs contain four mandatory pins and one optional pin as described below.

• **Test Clock Input (TCK):** a clock independent of the system clock for the chip so that test operations can be synchronized between the various parts of a chip. It also synchronizes the operations between the various chips on a printed circuit board. As a convention, the test instructions and data are loaded from system input pins on the rising edge of TCK and driven through system output pins on its falling edge.

• **Test Data Input (TDI):** an input to allow the test instruction and test data to be loaded into the instruction register and the various test data registers, respectively.
- **Test Data Output (TDO):** an output to allow test data be driven out.

- **Test Mode Selector (TMS):** the test control input to the TAP controller. The test operations are controlled by the sequence of 1s and 0s applied to this input. Usually this is the most important input that has to be controlled by external testers or the on-board test controller.

- **Test Reset Input (TRST):** The optional TRST pin is used to initialize the TAP controller, that is, if the TRST pin is used, then the TAP controller can be asynchronously reset to a Test-Logic-Reset state (to be discussed later) when a 0 is applied at TRST. This pin can also be used to reset the circuit under test.

4.1.2 **Boundary Scan Cell**

The IEEE Std. 1149.1a specifies the design of four test data registers as shown in Figure 27. Two mandatory test data registers, the bypass and the boundary-scan resistors, must be included in any boundary scan architecture. The boundary scan register, though may be a little confusing by its name, refers to the collection of the boundary scan cells. The other registers, such as the device identification register and the design-specific test data registers, can be added optionally. A typical BS cell is shown in Figure 28.

![Figure 28: A Typical BS Cell](image-url)
This cell can be used as either an input or output cell. As an input BS cell, IN corresponds to a chip input pad, and OUT is tied to a normal input to the application logic. As an output cell IN corresponds to the output of the application logic, and OUT is tied to an output pad. The operation of the test data registers is controlled according to the output of the TAP controller. During the normal operation, data passes from port IN to port OUT and the cell is transparent to the application logic. During the test operation, in the capture action, \( \text{ShiftDR} = 0 \) and one clock pulse is applied to ClockDR. Thus the test data will be captured into QA D-FF. In the scan action, \( \text{ShiftDR} = 1 \) and clock pulses are applied to ClockDR such that test data can be shifted in from SIN. At the same time, test response can be scanned out through SOUT. In the update mode, the data stored in QA D-FF can be applied to the OUT port by setting Mode control to 1 and applying a clock pulse to UpdateDR to transfer the data in QA to QB.

### 4.1.3 TAP Controller

The TAP controller is a 16-state finite state machine that operates according to the state diagram shown in Figure 29. The TAP controller can change state only at the rising edge of TCK and the next state is determined by the logic level of TMS. In other words, the state transition in Figure 29 follows the edge with label 1 when the TMS line is set to 1, otherwise the edge with label 0 is followed. The output signals of the TAP controller corresponding to a subset of the labels associated with the various states. As shown, 9 mandatory (ClockDR, ShiftDR, UpdateDR, ClockIR, ShiftIR, UpdateIR, Select, TCK, and Enable) and one optional Reset signals are defined. The main functions of the TAP controller are:

- To reset the boundary scan architecture,
- To select the output of instruction or test data to shift out to TDO,
- To provide control signals to load instructions into Instruction Register,
• To provide signals to shift test data from TDI and test response to TDO, and

• To provide signals to perform test functions such as capture and application of test data.

![State Diagram of TAP Controller](image)

**Figure 29:** State Diagram of TAP Controller

The 16 states can be divided into three parts. The first part contains the reset and idle states, the second and third parts control the operations of the data and instruction registers, respectively. Since the only difference between the second and the third parts are on the registers they deal with, in the following only the states in the first and third parts are described. Similar description on the second part can be applied to the third part.

1. **Test-Logic-Reset:** In this state, the boundary scan circuitry is disabled and the system is in its normal function. Whenever a Reset signal is applied to the BS circuit, it also goes back to this state. One should also notice that whatever
state the TAP controller is at, it will go back to this state if 5 consecutive 1's are applied through TMS to the TAP controller.

2. **Run-Test/Idle:** This is a state at which the boundary scan circuitry is waiting for some test operations such as BIST operations to complete. One typical example is that if a BIST operation requires 216 cycles to complete, then after setting up the initial condition for the BIST operation, the TAP controller will go back to this state and wait for 216 cycles before it starts to shift out the test results.

3. **Select-DR-Scan:** This is a temporary state to allow the test data sequence for the selected test-data register to be initiated.

4. **Capture-DR:** In this state, data can be loaded in parallel to the data registers selected by the current instruction.

5. **Shift-DR:** In this state, test data are scanned in series through the data registers selected by the current instruction. The TAP controller may stay at this state as long as TMS=0. For each clock cycle, one data bit is shifted into the selected data register through TDI.

6. **Exit-DR:** All parallel-loaded (from the Capture-DR state) or shifted (from the Shift-DR state) data are held in the selected data register in this state.

7. **Pause-DR:** The BS pauses its function here to wait for some external operations. For example, when a long test data is to be loaded to the chip(s) under test, the external tester may need to reload the data from time to time. The Pause-DR is a state that allows the boundary scan architecture to wait for more data to shift in.

8. **Exit2-DR:** This state represents the end of the Pause-DR operation, allows the TAP controller to go back to Shift DR state for more data to shift in.
9. **Update-DR**: The test data stored in the first stage of boundary scan cells, (QA) is loaded to the second stage (QB) in this state.

There are three mandatory boundary scan test instructions, namely EXTEST, SAMPLE/PRELOAD and BYPASS in addition to a commonly used instruction, namely the INTEST instruction. There exist some other useful instructions, e.g., RUNBIST, RUNSCAN, and IDCODE. We will only briefly describe these instructions. The EXTEST instruction is used to test the interconnect between two chips. It’s execution can be divided into three steps. Refer to Figure 30. Assume that an interconnect line from Chip1 to Chip2 as shown is to be tested. First, the test pattern

![Figure 30: Execution of EXTEST Instruction](image)

is shifted into the "driving terminals" of Chip1 through its TDI pin under the ShiftDR state of the TAP controller. Second, an Update operation in Chip1 is executed such that the shifted test data bit is loaded to the corresponding output pin of Chip1. At the same time, a Capture operation is executed in Chip2 and the test data bit is captured at the "driven terminal" of Chip2. In the third step, the ShiftDR operation is executed in Chip2 and the received test response can be scanned out through the TDO of Chip2 for examination. By this instruction, the interconnect between chips can be easily tested. Note that in the above description we only examine one interconnect. Clearly several interconnects can be examined simultaneously. The only
requirement is to shifted the required data bits to the appropriate driving positions and then capture these data at the driven positions simultaneously. Then the test responses can be scanned out through the TDO of Chip2 for further examination. In Figure 31 we show the steps of the INTEST instruction. At the first step, test data

![Figure 31: Execution of INTEST Instruction](image)

are shifted into the boundary scan cells of the driving terminals of the circuit under test, i.e., the internal logic. In the second step, and update operation is executed and the shifted data are loaded to the second stage of the boundary scan cell. At the same time the data is applied to the internal logic. The TAP controller then goes back to the Capture-DR state to capture the test result at the "receiving" terminals. Finally the ShiftDR operation shift the test results for observation. The reader may compare the execution of this instruction with that of the EXTEST instruction. It should be understood that the execution of INTEST requires two passes through the control of data registers part of the TAP controller, while the EXTEST only requires one pass of execution. In Figure 32 we show the execution of the SAMPLE/PRELOAD instruction.

The SAMPLE operation can be completed by simply executing the Capture operation such that the required test data can be loaded in parallel to the selected data registers, while the PRELOAD operation simply means that the test data can be shifted into the selected data register. At the first glance, these two instructions
Figure 32: Execution of SAMPLE and PRELOAD

appear to be independent. However due to the execution procedure of the TAP controller, one should be able to find that both of these two operations require one pass of ejection of the control of the data registers. Therefore in Std. 1149.1, these two operations are merged into one instruction.
CHAPTER V

AUTOMATIC TEST EQUIPMENT

5.1 Overview

In this chapter we look at different types of Automatic Test Equipment implementing various methods of testing. The purpose of testing methods and their advantages and disadvantages are covered in this chapter. The results was helpful to figure out what sort of test methods should be supported by the tester-on-chip and what category of ATEs should be focused on to enhance the ToC effectiveness.

5.2 Types of ATE

A class of equipment used to test die, chip, printed circuit board or system is generally called ATE. An ATE may be used to test a final packaged product, or it may be used in conjunction with other equipment such as wafer probes to conduct test on raw die. Based on the economic requirements and technical benefits, different types of ATE are used. Among the most popular ATEs we can mention to:

- Functional Testers

Theses testers check for proper operation of the device under test. Functional test vectors are applied to DUT and its responses are captured and compared with expected data by functional tester. Any discrepancy between the captured and the expected data is reported as an error. Functional tests cover a very high percentage of modeled faults in logic circuits.

- Parametric Testers

This kind of testers are widely used in manufacturing facilities. There are two
types of parametric test, DC and AC parametric tests. DC parametric test detects short test, open test, maximum current test, leakage test, output drive current test, and threshold levels test. AC parametric test covers propagation delay test, setup and hold test, functional speed test, access time test, refresh and pause time test, and rise and fall time test. These tests are usually technology depended.

- **In-Circuit Testers (ICT)**
  
  This form of fixture based tester has been in use for many years and not only looks at short circuits, open circuits, component values, but it also checks the operation of ICs. Although a very powerful tool, In-Circuit(Figure ) testers are limited these days by lack of access to boards as a result of the high density. Pins for contact with the nodes have to be very accurately placed in view of the

![Figure 33: Typical In-Circuit Tester](image)

very fine pitches and may not always make good contact. In view of this and the increasing number of nodes being found on many boards today it is being used less than in previous years, although it is still widely used.
• **Automatic Optical Inspection (AOI) Systems**

This systems test DUT or printed circuirt boards by optical signal instead of widely used electrical signals. DUT is scanned via a laser beam to generate an image of DUT, then this image is compared with a reference image to locate possible faults. AOI systems have a unique ability to inspect a circuit board without experiencing the consequent errors which accompany human inspection. Along with this increased accuracy, AOI systems bolster inspection speed (under a minute for an assembly with up to 100 components and over 1000 solder joints). These systems have a high degree of fault coverage, yet do not need test fixtures, electrical sources, or measurements. The inability to perform electrical tests limits their usefulness, since a major aspect of product inspection involves testing functionality.

• **X-Ray Systems**

These systems are useful for inspecting ball-grid arrays and other set ups that mask solder connections on Printed Circuit Board (PCB). More specifically, they are very good at detecting badly placed or missing solders. While electrical testing will identify these defects as well, x-ray systems can detect them immediately. Companies can justify the purchase of what are typically very expensive x-ray systems by noting their versatility (they can be used to assist in failure analysis, inspect packaging, etc.). To overcome the lack of depth in x-ray images, systems that can produce 3-D images are now available. This type of systems suffers from the same disadvantages of AOI system which is inability to carry out electrical tests.

• **Flying-Probe Systems**

Increased density and complexity present an even greater challenge for manufacturers of low-volume production runs and prototype PCBs, who need to
conduct inspections more quickly and cheaply than higher volume manufacturers. The declining fault coverage of in-circuit testers which is a result of this increased complexity has opened the door to a vastly improved new generation of flying-probe systems. While slower than traditional in-circuit testing systems, flying-probe systems can provide nearly 100 percent fault coverage. They simultaneously analyze and record results and therefore reduce the costs, of testing. On the down side, they cannot provide the kind of comprehensive coverage of in-circuit testing.
CHAPTER VI

A DIGITAL TESTER ARCHITECTURE

6.1 Overview

In this chapter a new architecture for testing a core based system-on-chip is proposed. The novel feature of the approach is the use of a SoC implementation of a tester-on-chip together with a microelectromechanical systems (MEMS) test socket instead of traditional test head. This method greatly reduces transmission line effects and supports high speed bi-directional testing. A tester-on-chip is installed in a fixed MEMS socket and connects to the die-under-test via a removable MEMS interface. At speed tests are carried out inside the ToC while test results and control signals are transferred between the automatic test equipment and ToC at low speed. This approach eliminates the necessity of using matched impedance interfaces, costly test heads and high-speed ATE for SoC testing and dramatically decreases the cost of SoC testing.

6.2 Core based SoC testing challenges

Pre-designed reusable cores have fundamentally changed the way digital systems are designed. Core based system-on-chip design methodologies have reduced aggressive time to market and cost requirements of today’s new products. However, testing core-based systems is a major challenge. With the cost of testing nearly matching the cost of production, researchers are struggling to integrate design tools and testers to reduce the cost of testing. Built in self test (BIST) and design for test (DFT) have dominated discussions of how to overcome the astronomical costs of automated test equipment.
However, the problem of economically testing a SoC remains unsolved. This problem arises from the difficulty of constructing high-speed, general-purpose connections between the automatic tester equipment and the device under test. The traditional approach to increase the speed is to precisely match the impedance interfaces and reduce transmission line effects as much as possible by placing pin-electronics as close as possible to the device under test, but this method faces severe practical limitations. Specifically, the best general purpose pin-electronics dissipate a considerable amount of power and they can not be highly integrated. These limitations forced ATE designers to resort to costly test heads which are quite often cooled down using advanced liquid cooling systems. The packaging density and power consumption continues to worsen as the pin-count increases. The result is a very costly test system, typically a few million dollars for SoC testing. Many types of SoC testing methods have been proposed in the literature to minimize the cost of testing. In [10] a method of using an embedded ATE is proposed which is successful in lowering transmission line effects by decreasing the rate of data between the external ATE and the DUT, but suffers from the drawback of relatively large test area overhead. In [11,12] an alternative approach to test a SoC is proposed which is based on the fact that most system-on-chip designs have some sort of microprocessor in them that can be employed to perform test for the other cores. This method successfully minimizes the area overhead, but it is limited to just microprocessor based SoC systems which is less efficient than a general SoC test solution. We proposed a new method which has the potential to dramatically change the way that an ATE is designed. In our test scheme the test head and its associated pin-electronics are replaced with a MEMS socket together with a tester-on-chip. The design method of the MEMS socket system to establish the high density interconnection of a die-under-test with a tester-on-chip has already been published by our research group in [9]. In this paper the architecture of a tester-on-chip is proposed which connects to the DUT via a MEMS socket interface. The
advantages of this method can be summarized as: 1- Minimizes the transmission line effect problem for high speed SoC testing 2- Lowers the cost of testing significantly by reducing the required performance of the ATE. 3- Reduces the packaging cost by detecting faults at the die level and removing faulty dies before the packaging process starts.

6.3 Test Scheme

The concept of MEMS socket based SoC testing is illustrated in Figure 34. The fixed MEMS socket and tester-on-chip are packaged together to form a test head, while a removable MEMS socket acts as die-specific carrier for the die-under-test. The DUT is placed in a removable carrier socket and then a load mass is positioned on the DUT to keep it tightly pressed against the carrier to realize the connection between the DUT and the ToC.

The aim of the tester-on-chip is to integrate the high-speed portions of the external ATE directly into the ToC which is located just a few hundred micrometers away from the die-under-test. With the DUT and pin-electronics separation reduced to micrometers, the roundtrip delay time could be as little as 4.7ps (i.e. 700 m). As a result, the transmission line effects can be ignored and the ATE resources required for matched impedance lines can be eliminated at high speed data rates (i.e. 2Gbps). Moreover, by restricting the range of the drivers and receivers to only a single technology the power and the size of the pin-electronics is significantly reduced. The ToC can be fabricated in the same technology as the die-under-test to exactly match the I/O characteristic of the DUT. As a result as illustrated in Figure 35 the pin-electronics could be replaced with simple line drivers and receivers.

The ToC integrates the linear feedback signature register (LFSR), multiple input signature register (MISR), precision timing, real-time comparison, self-test and 256
Figure 34: MEMS Socket Based Test Scheme
Figure 35: Driver-Receiver for MEMS Based Test Head

pin-electronics. Vector memory resides outside of tester-on-chip and is connected to the ToC via the removable MEMS socket so that the amount of vector ram can be easily reconfigured depending on the die-under-test. This arrangement not only saves considerable area on the ToC die which can now be used for advanced high speed timing and control circuits, but also enables the tester-on-chip to carry out at-speed tests for devices that require a test pattern depth of higher than a 4M of vector memory per pin.

In the design of the ToC, one of the critical decisions was how to implement the on-chip timing generators for full flexibility in shaping the output waveform. The timing control circuit covers the maximum possible range of duty cycle and edge placement within the cycle boundaries, allowing the user to place edges near the beginning or end of the test cycle. Each designed I/O module contains eight I/O pins and 35 separate timing generators for stimulus and response edge placement. Each output pin can select from 7 timing generators to define the leading and trailing edges placement of the stimulus pin, and each group of eight output pins shares an independent set of 5 timing generators for a total of 35 stimulus timing generators per module.
6.4 Tester-on-Chip

Dominating testers in the market such as systems designed by Advantest\footnote{www.advantest.com} and Agilent\footnote{www.agilent.com} companies contain numerous boards and hundreds of integrated circuits. They are installed in an isolated area under special conditions (ie air conditioning). All these sophisticated systems and special environment increase the cost of testing dramatically and now it is obvious that the traditional approach to tackle SoC testing will face much more serious problems in the near future and may proceed to deadlock.

That is one of the main reasons that IEEE has formed a group of best researcher in this filed to find out an economic and practical solution for SoC testing based on a new scalable architecture for testing embedded cores which is called IEEE P1500, these new standard has not been finalized yet.

MEMS fixture based plan has a very good potential to handle high speed testing which is one of the major problems in the area of SoC testing. The heart of this new architecture is a Tester-on-Chip and its performance would have a major impact on the success of the proposed test architecture.

One of the main challenges during the design process of ToC was to design a tester just on one chip meeting major requirements for SoC testing. To do it we had to come up with a new approach to minimize building blocks of the ToC. To fulfill this requirement Pin-electronics which are the main parts of most of the automatic tester equipment were optimized for area overhead. The method of optimization and design details of Pin-electronics is explained in the next chapter.
CHAPTER VII

TESTER-ON-CHIP STRUCTURAL DESIGN

7.1 Tester-on-Chip Design Flow

There are various ways and methodologies of doing VLSI digital design. The main aim of this methodologies is to get from concept to chip quickly and efficiently. But most of the time the way that digital flow goes is determined by the software tools for doing the design. Based on the available tools at the VSLI lab and CMC guidelines, a design flow illustrated in Figure 36 was chosen and the following steps were taken to design the tester on chip.

![Design Flow of Teser-on-Chip](image)

**Figure 36:** Design Flow of Teser-on-Chip
1. Based on the proposed test architecture and SoC test requirements the block diagram of the ToC (Figure 37) was determined.

2. The ToC hardware is described using Verilog and the first step of simulations to verify functionality was done in cadence environment. The behavioral description of the hardware using RTL (Register Transfer Level) was generated, these codes were synthesizable and they could be mapped into their gate level descriptions. These codes were compiled in cadence environment by Verilog compiler.

3. A test bench was prepared in Verilog to generate all necessary signals to test the ToC and to make sure that the design requirements are met.

4. At this steps synopsys was used to synthesize the RTL codes. Synopsys needs inputs of the RTLS, a target library (CMOS 18μm for the ToC), link library and a set of constraints. The costraints are based on three aspects of the design: Timing, Power and Chip Area. There are other types of constraints as well such as pin drives, capacitances and transition times. How to set constraints are one of the most important phase of design, they should be very carefully set to guarantee a functioning outcome at the end. The constraints were set based on 10ns clock period to allow the ToC to carry out the test with 100 Mhz clock rate.

5. To make the ToC testable, all flip flop changed to scannable flip flops and other test structures were integrated in the design automatically by Test Tools then test patterns for testing the ToC circuits were generated by the ATPG (Automatic Test Pattern Generation)

6. The outputs of the synthesis are a netlist in HDL and timing information file in SDF (standard delay format) which can be used for timing analysis. The
Figure 37: Block Diagram of Tester-on-Chip
Qplace timing engines were used to do Static Timing Analysis. After doing iterations and running incremental compilation finally the timing specifications were met.

7. Post Synthesis simulation were done on the design to revalidate timing and functionality. The ToC's test bench were modified and employed to generate output waveforms to ensure that design has not been broken during synthesis process.

At this stage physical implementation of ToC begun and the result of synthesis were exported to areaPdp and then silicon ensemble for floorplanning and routing. The steps are summarized as follow:

- Loading in the initial design data(library information, Verilog netlist, and synthesis constraints)
- Initial die size estimation and automatic floorplan creation.
- I/O placement.
- Automatic timing driven placement of block macros.
- Initial power routing(final power routing is done later).
- Timing driven placement and concurrent placement based optimization through gate resizing and buffer insertion/deletion for the standard cells.
- Clock tree synthesis and placement.
- Inserting Filler cells.
- Routing the design (final power routing, clock routing and signal routing).
- Extracting parasitics and post clock tree path optimization through gate resizing and buffer insertion.
• Incremental final routing to repair optimized nets and post Routing Timing Analysis.

• Verification- connectivity, antenna and geometry.

• Output of final Verilog, DEF, SDF and GDSII. At this stage some other checks were done to prepare the ToC for fabrication such as Design Rule Checks(DRC), Layout Versus Schematic(LVS), ANTIENNA CHECK and Layer Density Checks on the design gds2.

### 7.2 Tester-on-Chip building blocks

As shown in Figure(37) the ToC has four major blocks as follow each of them containing several different submodules.

#### 7.2.1 Pin-Electronics

The stimulus data are applied to the device under test and DUT’s responses are captured by pin-electronics(PE). Basically, the characteristics of PE are key points to differentiate high and low quality testers. The circuit of pin-electronic is a part which is repeated for every IO pins and its area optimization, would result in a considerable area saving at the end.

To design an efficient pin-electronic, we assumed that the ToC is going to be used for testing CMOS 0.18m devices and the transmission line effect can be ignored at high frequency due to the MEMS based architecture. As a result the ToC’s pin-electronics was substantially simplified and all circuits to overcome high frequency problems were eliminated. This analysis did allow us to design pin-electronics for ToC using tri-state buffer drivers.

pin-electronics produces test cycle and supports edge placement. During the process of SoC testing most of the time an advance timing should be supported to fulfill necessary requirements to enable DUTs internal modules. Even for a very common
device such as D flip-flops, counter, latches, etc. some timing conditions should precisely be met to have a valid test results.

To clarify how the PE was designed, assume that DUT is a single D flip flop with just two inputs and one output as shown in Figure 38. We start the test process step by step, in each stage a problem facing the test is described and the solution is explained. As a result, the basic conditions that should be supported to have a functioning tester are determined.

![Figure 38: An Example of Testing](image)

1. To carry out the test, test channels have to be connected to the DUTs input and output and they have to be assigned as output and input respectively.

   • We have no information about DUT’s IO pins configuration but we do know what ever they are, the assigned test channel to the DUT pin should be either input or output. Therefore we have to consider bidirectional instead of unidirectional IO channels for the ToC to handle all possible DUT pin configurations.

2. As shown in Figure 38 IO1 should generate clock signal.

   • To handle these type of situations, a reference oscillator and a network to apply it to the IO pins should be considered.
3. IO2 has to provide data and we have to make sure that the data remains stable during rising edge of the clock.

4. IO3 has to capture the information on the flip flop’s output. To have a valid test, the data should be captured when it is valid on the flip flop’s output.

- To fulfill the requirements of the last two steps, a programmable delay line in needed for the purpose of clock edge placement.

A block diagram of a single channel of our PE architecture is shown in Figure(39). It consists of five portions: timing, formatter, driver, capture and comparison. The timing produces test cycle and apply it to the formatter. The formatter section takes the stimulus information of the test vector, combines it with timing information. The driver, takes the formatted signals and apply them to the input pins of the DUT. The

![Figure 39: Pin-Electronics Block Diagram](image)

capture section samples the output pins of the DUT at a specified time and the comparison part, compares the results with the information supplied by the test vectors to generate PASS/FAIL signal. In the design of the PE chip, one of the important
design decisions was how to implement the on-chip timing generators. There were
two basic approaches to this problem: either build high-accuracy delay generators
using techniques such as PLL delay lines or use circuits which are stable and can be
easily calibrated, but which have low absolute accuracy. The latter approach was
chosen because of better expected die size.

Timing Module:
The timing generator provides a timing control pulse which determines where the
dge transitions will occur in the DUT waveform. This pulse determines the tran-
sition times of the waveform created by the formatter. The pin driver buffers the
low-level internal signal up to the external drive level.

For full flexibility in shaping the output waveform, the timing control pulse must
cover the maximum possible range of duty cycle and edge placement within cycle
boundaries. Allowing the user to place edges near the beginning or end of the tester
cycle makes the design of the pulse generator difficult because of the natural delays
of the logic. In addition, producing pulses with very small or very large duty cycles
is also hard since delay lines tend to deform narrow pulses. Both of these problems
become more serious when a relatively low-speed base technology like CMOS is used.
The diagram of the timing generator in Figure(40) shows that a programmable delay
line was used to overcome timing problems. The delay line consist of two different
delay elements: a shift register for low resolution delay adjustment, an inverter chain
for finer resolution. The input to both delay lines is the square-wave clock whose
period is one-half that of the tester cycle period. To form the timing pulse, the delay
lines provides a method of placing transitions near the tester clock edges.

Each of the two delay types employs a different technique to achieve the desired delay
and resolution required. In order to obtain relatively long but stable delays, a shift
register is used with an external clock to supply stable reference. The final output
stage of the shift register allows selection of either a normal rising edge-triggered D-type flip-flop output, or a falling edge-triggered master-only stage. This provides an overall resolution of one-half of the clock period. The inverter chain stage uses the minimum delay through a pair of inverters as a compact means of obtaining high resolution.

In order to accurately probe the frequency limits of the DUT, it is necessary for the PE to provide continuous operation over a wide range of frequencies. This presents a problem in the multistage delay generator chain. For the scheme to work, the range of delay provided by each of the delay types must be greater than the resolution of the previous stage. A problem occurs in the shift-register delay stage as the tester frequency is reduced from the maximum; the resolution of the shift register decreases with the clock frequency, but the range provided by the following stage, the inverter chain, remains constant since it is not tied to the clock frequency. To solve this problem, the shift-register clock is selected to be a multiple of either of one, two, four, or eight times the cycle clock frequency depending on the cycle frequency. By doing this, the resolution of the shift register is always constrained to be within the range of the inverter chain.
Formatter Module

The formatter combines the stimulus data of the force vector with the edge placement information of the timing generator. This data/timing mixing is performed in one of five selectable modes: nonreturn to ZERO (NRZ), return to ONE (RO), return to ZERO (RZ), return to tri-state (RT), and return to complement (RC). Figure 41 shows different types of signals, supported by formatter. The NRZ format is a non-pulsed mode; that is, a transition is made to the level specified by the input vector after a given delay and that level is maintained until the next tester cycle. There is at most one data transition per cycle. This mode is useful for simulating edge-triggered events such as register or counter outputs. All of the other four formats are pulsed modes: they start at a default level (ZERO, ONE, or tri-state for RZ, RO, and RT, respectively), transition to the desired level after a given delay, maintain that level for a set width, then return to the starting level. In operation, these modes allow the tester to simulate events at rates higher than the basic tester cycle rate would allow.

![Stimulus Waveforms](image)

**Figure 41:** Stimulus Waveforms
The RO format could also be used in such an example to generate the complementary clock. The RT mode doubles the PE channel’s I/O bandwidth in much the same manner as the RZ and RO modes are used to increase the output event rate. The main use of the RC mode is for making setup and hold measurements. In the RC mode the default level is the complement of the data to be driven. After the specified delay the output transitions to the value specified by force data, maintains that value for the specified width, and then returns to the complementary value. By adjusting the delay and width of the drive pulse to exactly meet the specified setup and hold requirements of the DUT, the device will fail if its specifications are exceeded. The RC mode differs from each of the other four modes in that it can cause up to three data transitions per cycle (RT can have three or even more transitions per cycle but only two are caused by the PE driver). One transition occurs at the start of each cycle to assume the complementary drive value, and two more occur when the output pulses to the drive value and back to the complementary value. When used as intended for making setup and hold measurements, this edge should fall outside of the capture window of the DUT. The stimulus waveform generated by the formatter must be amplified to drive the potentially high-capacitance load of the DUT.

**Capture and Comparison Modules:**
Response evaluation is done at Capture and Comparison(Figure ??) portion of the PE circuitry which is responsible for capturing data from the DUT and determining if it matches the desired response. Both the data value and output voltage level must be checked. In the acquire timing generator only control of the sample delay timing is necessary. In the data pipeline the acquired value is compared with the expect data from the test vector. If the comparison fails and the mask bit from the test vector is not set.
A stability checking circuit is used to determine if the DUT output(s) fluctuate during
Figure 42: Capture and Comparison Block Diagram

the clock period. The value of the DUT output at the leading edge is stored. This stored value is logically compared with the DUT output over the period of one test clock cycle. If the comparison changes during the clock period then a stability check flag is set. The occurrence of a stability flag in the absence of a corresponding comparison failure indicates a fault being masked by a hazard.

The following Verilog codes present RTL description of pin-electronics(Figur 39).

*************** RTL Codes representing Pin-Electronics ******

module Pin_Electronics
(Data,Address,Read,Write,Enable_VRAM,Reset,Cnk,Test_Channel,Fail);
inout[15:0] Data;
inout[7:0] Address;\;
inout Test_Channel;\;
input Reset,Read,Write,Enable_VRAM,Cnk;\;
wire Enable,Cap_Drv_Cnk,Stimulus,E_data,Mask;\;
wire [2:0] type,Lo_delay,Hi_delay;\;

64
wire[15:0] Vector_Data;
output Fail;
reg phi;

always @(posedge Clk) phi=!phi;
Timing u1(Lo_delay,Hi_delay,phi,Reset,Clk,CapDrv_Clk);
Formatter u2(CapDrv_Clk,Stimulus,type,Test_Channel,Enable,
             NRZ,RZ,RZI,R0,ROI,RINH,RC);
Capture u3(CapDrv_Clk,Test_Channel,Cap_data,Enable);
Comparison u4(CapDrv_Clk,Cap_data,E_data,Mask,Fail);
Vector_RAM u5(Vector_Data,Data,Address,Read,Write,
               Enable_VRAM,Clk); // outside of ToC

assign Enable=Vector_Data[0];
assign type[2:0]=Vector_Data[3:1];
assign Hi_delay[2:0]=Vector_Data[7:5];
assign Lo_delay[2:0]=Vector_Data[10:8];
assign Stimulus=Vector_Data[11];
assign E_data=Vector_Data[12];
assign Mask=Vector_Data[13];

endmodule
module Capture(CapDrv_Clk, Test_Channel, Cap_data, Enable);

inout Test_Channel; input CapDrv_Clk, Enable; \
output Cap_data; \\
reg Cap_data;

always@ (posedge CapDrv_Clk)
if(Enable==0) Cap_data=Test_Channel;
endmodule

module Comparison(CapDrv_Clk, Cap_data, E_data, Mask, Fail);

input Cap_data, E_data, Mask, CapDrv_Clk; \\
wire[0:6] Output_of_delay_line; \\
wire CapDrv_Clk; \\
output Fail; \\
reg Fail; \\
hr_delay uc1(CapDrv_Clk, Output_of_delay_line);
mux uc2(3'b000, {Output_of_delay_line, CapDrv_Clk}, CapDrv_Clk_Delayed);
always@ (posedge Cap_Drv_Clk_Delayed)
begin if (Mask==0) Fail=0 ;
else if (Cap_data==E_data) Fail=0;
else Fail=1; // Faults set Fail=1 end endmodule

/************ Timing Module *************

module Timing(Lo_delay,Hi_delay,phi,Reset,Clk,Cap_Drv_Clk);

//phi test clock
//Clk delay clock
//adr delay definition
//Reset Reset
//out output

input phi,Reset,Clk;\
input[2:0] Lo_delay,Hi_delay;\
output Cap_Drv_Clk;  \ 
wire [6:0] hr_dly_0_to_6,q;\ 
wire mux1out,Clk_bar;

assign Clk_bar=~Clk;

dff u0(phi,Clk,Reset,q[0]);\ 
dff u1(q[0],Clk,Reset,q[1]);\ 
dff u2(q[1],Clk,Reset,q[2]);\ 

dff     u3(q[2],Clk,Reset,q[3]);\

dff     u4(q[3],Clk,Reset,q[4]);\

dff     u5(q[4],Clk,Reset,q[5]);\

dff     u6(q[5],Clk,Reset,q[6]);\
mux     mux1(Lo_delay,{q[6:0],phi},mux1out);\
h delays u8(mux1out,hr_dly_0_to_6);\
mux     mux2(Hi_delay,{hr_dly_0_to_6,mux1out},CapDrv_Clk);

d endmodule

// **************** D flip flop ******************

module dff(d,Clk,Reset,q);\
input     Clk,d,Reset;\
output     q;\
reg       q;\
wire      Clk1,Reset_bar;\

assign Reset_bar=Reset;  \\  
assign     Clk1=Reset & Clk;  \\  
    always@ (posedge Clk1) if(Reset_bar)
        q=0 ;else q=d;
endmodule

// **************** Eight to one multiplexer ***********
module mux(adr,in,out); input [2:0] adr; input [7:0] in; output out; assign out=
    adr==3’b000 ? in[0]:
    adr==3’b001 ? in[1]:
    adr==3’b010 ? in[2]:
    adr==3’b011 ? in[3]:
    adr==3’b100 ? in[4]:
    adr==3’b101 ? in[5]:
    adr==3’b110 ? in[6]:
    adr==3’b111 ? in[7]:
    0;
endmodule

// *************** high resolution delay line ***************

module hr_delay(in,out);

input in; \ \
output[6:0] out; \ \
wire[6:0] wir;

not#(1)
    g00(wir[0],in),
    g0(out[0],wir[0]),
    g11(wir[1],out[0]),

69
g1(out[1],wir[1]),
g22(wir[2],out[1]),
g2(out[2],wir[2]),
g33(wir[3],out[2]),
g3(out[3],wir[3]),
g44(wir[4],out[3]),
g4(out[4],wir[4]),
g55(wir[5],out[4]),
g5(out[5],wir[5]),
g66(wir[6],out[5]),
g6(out[6],wir[6]);
endmodule

>Title

Formatter //********************************************

module Formatter(Cap_Drv_Clk,Stimulus,type,Test_Channel,Enable,
NRZ,RZ,RZI,RO,ROI,RINH,RC);

// Cap_Drv_Clk input of timing
// Stimulus input data from Vector Ram
// Test_Channel final output
// type determine the type of output
// if type=0 then Test_Channel=RZ
// if type=1 then Test_Channel=RZI
// if type=2 then Test_Channel=RO
// if type=3 then Test_Channel=ROI
// if type=4 then Test_Channel=RC

70
if type=5 then Test_Channel=RINH
if type=6 then Test_Channel=1
if type=7 then Test_Channel=0

input Cap_Drv_Clk, Stimulus, Enable;
input[2:0] type;
output Test_Channel, RZ, RZI, RO, ROI, RINH, RC, NRZ;
wire Test_Channel_wire, Enable_wire;
reg NRZ, RC_bar;

always @(negedge Cap_Drv_Clk) RC_bar=¬RC;
always @(posedge Cap_Drv_Clk) NRZ=Stimulus;

assign RZ=Cap_Drv_Clk ? Stimulus : 0;
assign RO=Cap_Drv_Clk ? Stimulus : 1;
assign RC=Cap_Drv_Clk ? Stimulus : RC_bar;
assign ROI=¬RO;
assign RZI=¬RZ;

bufif1 RINH_BUFF(RINH, Stimulus, Cap_Drv_Clk);

assign Test_Channel_wire=
    type==0 ? RZ:
    type==1 ? RZI:
    type==2 ? RO:
    type==3 ? ROI:
    type==4 ? RC:
type==5 ? RINH:

    type==6 ? 1:

    type==7 ? 0:

    0;

assign Enable_wire=

    type==5 ? Cap_Drv_Clk: // HighZ at the Test_Channel output

    Enable;

bufif1 driver(Test_Channel,Test_Channel_wire,Enable_wire);

e ndmodule

7.2.2 High Speed Test Management:

To supervise at-speed VLSI testing and perform selftest high speed test manage-
ment(Figure 45) part was designed. It has three submodules.

1. Processing Circuit:

This submodule receives control signals from external controller and generates
high speed address bus for Vector RAM. Initially, the test vectors are stored in
the Vector RAM. To apply these test data at the rated speed to the DUT, a
programable counter was considered to provide high speed address bus. First,
the start and stop address of the counter is loaded then the test mode is defined.
Three different mode of test are supported.

A- Normal Mode : In this mode counter runs just one time from the start
address to the stop address. As a result test vectors are applied to the DUT
just one time and the test result is saved. Later the result is sent to the external
controller for evaluation.

B- Pass Loop : In this mode the test runs as long as the test result is PASS and it stops at the first FAIL. This mode is helpful for fault diagnosis.

C- Loop Mode : In this mode counter runs as long as the status has not changed regardless of the test result. This mode is useful for IC manufacturing purposes.

The following Verilog codes represent the processing circuit:

Control(Data,Cntrl-Address,Enable-ERAM,Read,Write,Clk, Status,Reset,Fail,HILO,Counter); inout[15:0] Data;
input Clk,Reset,Read;
input[2:0] Fail,Enable-ERAM,Status;
input[4:0] Cntrl-Address;Error RAM address generator
input[1:0] HILO;// Control to read high or low part of the address bus
output[23:0] Counter;// Vector RAM address generator
reg Fail-reg;// Register to capture fails
reg[23:0] Counter,Stop-addr,Start-addr;
reg[15:0] data-reg1;
reg[7:0] data-reg2;

always @(posedge Clk) begin
  if(Reset)
    begin
      Stop-addr=24'hFFFFFFF; // vector RAM's end address
      Start-addr=0; // vector ram start address
      Counter=0; // vector ram address counter
      Fail-reg=0;
    end
  else
    begin
      if(Fail==1) Fail-reg=1;
      case(Status)
        0: data-reg1=Data; // load start address
        1: data-reg2=Data[7:0];
        2: Start-addr=data-reg1,data-reg2;
        3: Stop-addr=data-reg1,data-reg2;
        4: Counter=Start-addr;
        5: begin
          counter=Counter+1;
          if(Counter > Stop-addr)
            Counter=Start-addr;
        end
        6: if(!Fail-reg) // Pass-loop test mode
      endcase
    end
end
begin
  Counter=Counter+1;
  if(Counter > Stop-addr)
    Counter=Start-addr;
  end
endcase
end
endmodule

2. LFSR and MISR

The ToC are going to be employed to test SoC devices, the first we have to make sure that the ToC itself is fault free. To meet the requirement, build in self test(BIST) circuit based on LFSR and MISR were add to the ToC. The architecture is shown in Figure 45. The LFSR generates the test patterns for the ToC. The MISR compresses and analyzes the test responses to determine correctness of the DUT.

![Figure 44: ToC Selftest and Boundary Scan Architecture](image)
Our self test scheme aims at reducing the hardware overhead as much as possible. Instead of using LFSR and MISR for every input/output pins, our approach combine LFSR/MISR with shift register to minimize the hardware overhead. Figure 45 shows the structure of a build in self test for the ToC. LFSR generates test vectors and shift them to the inputs of the DUT via scan registers. At the same time, the response are scanned in and compressed by the LFSR. Instead of writing codes for this part, it was created directly by Synopsys Test Tools, scan and built in self test circuitry were inserted automatically.

3. Error-RAM

To enable the ToC to perform test at the rated speed. A circuitry should be considered to save at speed test results. This circuit can be as simple as register just to report PASS or FAIL, or it can be a huge memory, saving the result of all test steps separately. This approach is very helpful for diagnosis and repair purposes but it has considerable area overhead. We employed and Error-RAM which has just 32 memory locations, which means that in each run just the first 32 faults of each test channel are saved and the other ignored.

This module consist on a 32 registers each of them 24 bits and a simple control circuitry to detect the FAIL signal and to save the address of the faulty test step.

This module receives FAIL signal which is generated by comparison circuit as input and save the test step address in the assigned memory.

The Verilog codes to generate this part of the circuit are as follow.

```verbatim
//************************************************************************** ERROR RAM **********

module

Error_RAM(Data,Address,Read,Write,Reset,fail,Clk,Counter,Enable,HIL0);
```
input Read, Write, Reset, Clk, Enable; \\ninput fail; \\ninput[23:0] Counter; \\ninout[15:0] Data; \ \ninput[4:0] Address; \\ninout[1:0] HILO; \\nreg Stop_reg; \ 
reg [4:0] error_cntr; \ 
reg[23:0] memory[0:31], outreg; \ 
wire Stop_reg_bar, read_cycle; \ 
reg[5:0] i;

assign read_cycle= Enable & Read; \ \nassign Stop_reg_bar=~Stop_reg;

//when ERROR RAM is not enable it bus should be highZ
// The ToC’s Data bus is 16 bits but the address bus is 24 bits
// to read the address of faulty steps which are saved in the
// Error_RAM, HILO flag considered to read HI and LO part of
// the address separately

assign Data= Enable==0 ? 16’bZZZZZZZZZZZZZZZZ:

  (Enable==1 & HILO==0 & Read==1) ? outreg[23:8]:
  (Enable==1 & HILO==1 & Read==1) ? outreg[7:0]:
  (Enable==1 & HILO==2 & Read==1) ? 16’bZZZZZZZZZZZZZZZZ:
16'bZZZZZZZZZZZZZZZZ;

// save address of faulty steps
always @(posedge Clk) begin
    if(Reset)
        begin
            error_cntr=0;  // error counter
            Stop_reg=0;
            for(i=0;i<32;i=i+1) memory[i]=0;
        end
    else if(Stop_reg_bar)
        begin
            error_cntr = error_cntr+1;  // Freeze counting if there
                                          // are more than 32 faults
            if(error_cntr==5'b11111) Stop_reg=1;
            if(fail) memory[error_cntr]=Counter;
        end

    if(read_cycle) outreg=memory[Address];
end
endmodule

7.2.3 Interface

The Interface circuit provides all control signals and power supply for the ToC. There were several option for this portion of the ToC. We could use PCI bus, Serial or Parallel ports to control the ToC. Using PCI bus could simplify the ToC's internal circuitry but it was preferred to use Standard Universal Bus(USB) which is now more
popular and available on almost every types of PCs.

![Diagram of Interface](image)

**Figure 45:** Input and outputs signals of the Interface

There are lots of available USB IP cores free of charge for designers, so instead of redesigning this part, an already tested USB IP Core chosen. To make it a functioning module of the ToC, Verilog codes were added to make it compatible with the other portions. The codes representing USB port is attached.
CHAPTER VIII

SIMULATION AND RESULTS

8.1 Simulation of the ToC using Cadence and Synopsys tools

Verilog-XL is a great tool in Cadence to perform digital logic design. It lets you perform logic design at the functional level. To simulate the Tester-on-Chip a testbench was prepared to provide the input signal waveforms. All input signals were generated in the testbench environment and then applied to the Tester-on-Chip. The ToC’s responses were captured for design analysis by Signal-Scan which is a powerful analysis tool. Figure 46 shows the results and indicates the output signals generated by the ToC. All requirements were met at this level and the desired formats of output waveforms were successfully created. This simulation was repeated after synthesis and routing to ensure that the design has not been broken during numerous design steps. First, it was repeated after doing synthesis by synopsys and after creation of golden Verilog netlist by areaPdp the process of testing the ToC using testbench was repeated again. The results were not acceptable at the beginning but after doing some changes in design constraints the ToC design requirements were met.

The ToC was designed to meet general VLSI test requirements. To carry out test on microelectronics associated with hearing aid instruments the ToC could be simplified since hearing aid instruments are working at relatively low frequency and they have a very limited number of pins. A modified ToC with 32 IO channels was prepared for fabrication(Figure 47).

The design of each portion of the ToC was synthesized and checked to ensure the gate level design and the prepared firm core on CMOSP .18μm meets all the
Figure 46: ToC Simulation Results

Figure 47: Layout of the ToC
design goals. Synopsys optimizes the design based on the defined constraints and apply advanced methods to minimize area, power, etc. The process of optimization should be carefully supervised by the designer since the outcome does not necessarily meet the expectation and gate level Verilog codes have to be simulated again to ensure functioning design. Sometimes, optimization process of Synopsys damages the design, for example our low resolution delay line which was an array of NOT gates was replaced by a simple wire and all NOT gates were deleted by Synopsys automatically. Synopsys optimizes circuits based on their truth table, since the function of a wire is the same as the functions of even number of NOT gates connected serially, the whole delay line was replaced with a wire which was not acceptable in our design.

8.2 Test Flow

The process of SoC testing is performed in four different phases. In the first and last phases the external computer takes the control of the system to send test data and capture test results. In the other two phases the external controller is standby and all test processes are performed at the rated speed under the supervision of ToC's "High Speed Test Management" module.

1. Loading Phase:
   
   Stimulus Test Data or Test Vectors, which have already been prepared by SoC designer is loaded into the external controller or computer. The format of test data is modified based on the ToC requirements, then these new stimulus data are loaded into the vector RAM.

2. Initial Phase:
   
   Control words to initialize Test-Mode, Test-Speed, Start and Stop registers are loaded into the ToC by external controller.

3. Test Phase: Test vectors are applied at the rated speed to the DUT under
supervision of test management circuit and the DUT responses are captured and saved in the error-RAM.

4. **Evaluation Phase:**

The test results which have been captured by the ToC are imported by the external controller and any errors are reported to the system operator.

### 8.3 Analysis Reports

Design Compiler is the core of the Synopsys synthesis software products. It provides constraint-driven optimization and supports a wide range of design styles. Design Compiler was used to constraint and to optimize the ToC for speed, area, and power. The generated report indicates that the ToC timing requirements were met. Column path shows rising and falling delay of each cell in the circuit generated by Synopsys.

*************** Timing Analysis Report***************

Path Group: Clk
Path Type: max

<table>
<thead>
<tr>
<th>Des/Clust/Port</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToC256</td>
<td>TSMCSK_Conservative</td>
<td>tpz973gwc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Point</th>
<th>Incr</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock Clk (rise edge)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>clock network delay (propagated)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Component Description</td>
<td>Value</td>
<td>Type</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>Control/Counter_reg_1_/CK (DFEPQ1)</td>
<td>0.00</td>
<td>#</td>
</tr>
<tr>
<td>Control/Counter_reg_1_/Q (DFEPQ1)</td>
<td>0.47</td>
<td>f</td>
</tr>
<tr>
<td>U2880/Z (INVD0)</td>
<td>0.25</td>
<td>r</td>
</tr>
<tr>
<td>U2882/Z (INVD0)</td>
<td>0.29</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_1/CO (ADHALFD1)</td>
<td>0.28</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_2/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_3/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_4/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_5/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_6/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_7/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_8/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_9/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_10/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_11/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_12/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_13/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_14/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_15/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_16/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_17/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_18/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_19/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_20/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_21/CO (ADHALFD1)</td>
<td>0.23</td>
<td>f</td>
</tr>
<tr>
<td>r6084/U1_1_22/S (ADHALFD1)</td>
<td>0.38</td>
<td>r</td>
</tr>
<tr>
<td>U1303/Z (INVD1)</td>
<td>0.09</td>
<td>f</td>
</tr>
</tbody>
</table>

84
U78/Z (NAN2D1) 0.12 6.44 r
U112/Z (NAN2D1) 0.09 6.52 f
U110/Z (NOR2D1) 0.15 6.68 r
U587/Z (NOR2D1) 0.09 6.76 f
U585/Z (NOR2D1) 0.14 6.91 r
U1784/Z (OR2D0) 0.16 7.07 r
U685/Z (OR2D1) 0.17 7.24 r
U1313/Z (INVD1) 0.06 7.31 f
U1897/Z (OR2D1) 0.23 7.53 f
U1896/Z (INVD1) 0.52 8.06 r
U4852/Z (OAI22M22D1) 0.26 8.32 r
Control/Counter_reg_0_/D (DFEPQ1) 0.00 8.32 r
data arrival time 8.32

clock Clk (rise edge) 10.00 10.00
clock network delay (propagated) 0.00 10.00
clock uncertainty -0.30 9.70
Control/Counter_reg_0_/CK (DFEPQ1) 0.00 9.70 r
library setup time -0.21 9.49
data required time 9.49

-----------------------------
data required time 9.49
data arrival time -8.32

-----------------------------
slack (MET) 1.17
CHAPTER IX

CONCLUSIONS

9.1 Conclusions

An Intellectual Property (IP) core for a System-on-Chip (SoC) implementation of an integrated circuit tester or Tester-on-Chip (ToC) has been developed. The Tester-on-Chip IP core is used in conjunction with a microelectromechanical (MEMS) test fixture that provides connectivity to the device under test. The short MEMS-based connectivity paths between the ToC and the DUT reduce the transmission line effects normally associated with conventional test heads. A new tester architecture that works in conjunction with a MEMS test head has been developed in the form of a ToC IP core. Both a soft and firm realization of the ToC IP core has been developed using Verilog-XL and Synopsys. A ToC implementation using 0.18μm CMOS technology has also been developed. A scalable vector RAM is used to store the test vectors and it is held as a separate module in the MEMS interface socket. Both the DUT and scalable vector RAM are easily inserted into or removed from the MEMS socket using robotic or structured automation. Simulation studies have shown that the ToC can support 256 bi-directional test channels, each with a maximum test clock cycle of 100 MHz. The resulting test response information is analyzed by the ToC and the corresponding test results are sent via a Universal Serial Bus (USB) interface to a host computer, such as a laptop computer, for visualization and decision making.

9.2 Future Works

(A) Design a test software environment to upload test vectors into the Vector RAM and to download the test results for evaluation

86
(B) Prepare physical samples of MEMS socket, ToC and DUT to prove simulation results
REFERENCES


91


[38] Stannett M., "An abstract approach to VLSI test IEE Colloquium on Design for Testability, 16 May 1991, pp. 10/1 -10/3


APPENDIX A

VERILOG CODES FOR THE TOC

`timescale 1ns/100ps

module ToC256(Data, Address, Clk, Reset, Read, Write, Enable, Status, Test_Channel, HILO);

input Reset, Read, Write, Clk;
input[2:0] Status;
input[1:0] HILO; // Data bus is 16bits while counter is 24bits, read is done in two phase HIhg and Low
inout[15:0] Data;
wire [4:0] Cntrl_Address;

input[9:0] Enable; //if Enable[8]=1 then Enable Vector RAM - if Enable[9]=1 then Enable Error RAM

inout[255:0] Test_Channel;
wire [2:0] Enable_VRAM, Enable_ERAM;
wire Fail;

input[23:0] Address;
wire[4:0] Err_Address; // Provide address bus for Error RAM
wire[2:0] Hi_delay, Lo_delay, Type; // Hi resolution delay & low resolution delay registers - Type of output signal NRZ, RZ, ....
wire[23:0] Counter; //Provide high speed address bus for or Vector RAM
wire[7:0] VRAM_Address;

assign Cntrl_Address = Address[4:0];
assign VRAM_Address = Address[7:0];
assign Enable_VRAM = (Enable[9]!=1 & Enable[8]==0) ? 256’b1<<Enable[7:0] : 0;
assign Enable_ERAM = (Enable[9]==0 & Enable[8]==1) ? 256’b1 <<Enable[7:0]:0;

Control Cntrl(Data, Cntrl_Address, Enable_ERAM, Read);
module Pin_Electronics (Data, Address, Read, Write, Enable_VRAM, Reset, Clk, Test_Channel, Fail);

inout[15:0] Data; \
input[7:0] Address; \
inout Test_Channel; \
input Reset, Read, Write, Enable_VRAM, Clk; \
wire Enable, Cap_Drv_Clk, Stimulus, E_data, Mask; reg phi;
wire [2:0] type, Lo_delay, Hi_delay; \
output Fail;
wire[15:0] Vector_Data;

always @(posedge Clk) phi=~phi; \

Timing u1(Lo_delay, Hi_delay, phi, Reset, Clk, Cap_Drv_Clk); \
Formatter u2(Cap_Drv_Clk, Stimulus, type, Test_Channel, \
Enable, NRZ, RZI, RO, ROI, RINH, RC);
Capture u3(Cap_Drv_Clk, Test_Channel, Cap_data, Enable);
Comparison u4(Cap_Drv_Clk, Cap_data, E_data, Mask, Fail); \
Vector_RAM u5(Vector_Data, Data, Address, Read, Write, 
    //Enable_VRAM, Clk); // outside of ToC
assign Enable=Vector_Data[0]; \
assign type[2:0]=Vector_Data[3:1]; \
assign Hi_delay[2:0]=Vector_Data[7:5]; \
assign Lo_delay[2:0]=Vector_Data[10:8]; \
assign Stimulus=Vector_Data[11]; \
assign E_data=Vector_Data[12]; \
assign Mask=Vector_Data[13];
endmodule

/****************** Capture Module ******************
module Capture(CapDrv_Clk, Test_Channel, CapData, Enable);

  input  Test_Channel; \\
  input  CapDrv_Clk, Enable; \\
  output CapData; \\
  reg    CapData;

  always@ (posedge CapDrv_Clk) \\
  if (Enable == 0)
    CapData = Test_Channel; \\
  endmodule

//************************* Comparison Module ******************

module Comparison(CapDrv_Clk, CapData, E_data, Mask, Fail);

  input  CapData, E_data, Mask, CapDrv_Clk; \\
  wire[6:0] Output_of_delay_line; \\
  wire    CapDrv_Clk; \\
  output  Fail; reg Fail;

  hr_delay    uc1(CapDrv_Clk, Output_of_delay_line); \\
  mux         uc2(3'b000, {Output_of_delay_line, CapDrv_Clk}, CapDrv_Clk_Delayed); \\
  always@ (posedge CapDrv_Clk_Delayed)
    begin if (Mask == 0) Fail = 0;
      else if (CapData == E_data) Fail = 0;
      else Fail = 1;
    end

//************************* Timing Module ******************

module Timing(Lo_delay, Hi_delay, phi, Reset, Clk, CapDrv_Clk);

  //phi    test clock
  //Clk    delay clock
  //adr    delay definition
  //Reset  Reset
  //out    output
  input  phi, Reset, Clk; \\
  input  [2:0] Lo_delay, Hi_delay; \\
  output CapDrv_Clk; \\
  wire   [6:0] hr_dly_0_to_6, q; \\
  wire   mux1out, Clk_bar; \\

  assign Clk_bar = ~Clk; \\
module dff(d, Clk, Reset, q); input Clk, d, Reset; output q; reg q;
wire Clk1, Reset_bar; //
assign Reset_bar = Reset; //
assign Clk1 = Reset & Clk; //
always@ (posedge Clk1) if(Reset_bar)
  q = 0; //
  else q = d;
endmodule

module mux adr, in, out); input [2:0] adr; input [7:0] in; output out; assign out =
  adr == 3'b000 ? in[0]:
  adr == 3'b001 ? in[1]:
  adr == 3'b010 ? in[2]:
  adr == 3'b011 ? in[3]:
  adr == 3'b100 ? in[4]:
  adr == 3'b101 ? in[5]:
  adr == 3'b110 ? in[6]:
  adr == 3'b111 ? in[7]:
  0;
endmodule

module hr_delay(in, out);

input in; //
output[6:0]  out;  \nwire[6:0]  wir;  not#(1)
g00(wir[0],in),
g0(out[0],wir[0]),
g11(wir[1],out[0]),
g1(out[1],wir[1]),
g22(wir[2],out[1]),
g2(out[2],wir[2]),
g33(wir[3],out[2]),
g3(out[3],wir[3]),
g44(wir[4],out[3]),
g4(out[4],wir[4]),
g55(wir[5],out[4]),
g5(out[5],wir[5]),
g66(wir[6],out[5]),
g6(out[6],wir[6]);
endmodule

/***********************************************************
Formatter **********************************************************/

module Formatter(Cap_Drv_Clk,Stimulus,type,Test_Channel,
Enable,NRZ, RZ,RZI,RO,ROI,RINH,RC);

// Cap_Drv_Clk          input timing
// Stimulus             input data from Vector Ram
// Test_Channel          final output
// type                  determine the type of output
//                       if type=0 then Test_Channel=RZ
//                       if type=1 then Test_Channel=RZI
//                       if type=2 then Test_Channel=RO
//                       if type=3 then Test_Channel=ROI
//                       if type=4 then Test_Channel=RC
//                       if type=5 then Test_Channel=RINH
//                       if type=6 then Test_Channel=1
//                       if type=7 then Test_Channel=0

input  Cap_Drv_Clk,Stimulus,Enable;  \ninput[2:0]  type;  \noutput  Test_Channel,RZ,RZI,RO,ROI,RINH,RC,NRZ;  \nwire  Test_Channel_wire,Enable_wire;  \nreg  NRZ,RC_bar;

always @(negedge Cap_Drv_Clk)  RC_bar='RC;  \nalways @(posedge Cap_Drv_Clk)  NRZ=Stimulus;  \nassign  RZ=Cap_Drv_Clk  ?  Stimulus  :  0;

98
assign RO=CapDrvClk ? Stimulus : 1;
assign RC=CapDrvClk ? Stimulus : RC_bar;
assign ROI="RO;
assign RZI="RZ;

bufi1 RINH_BUFF(RINH,Stimulus,CapDrvClk);

assign Test_Channel_wire=
    type==0 ? RZ;
    type==1 ? RZI:
    type==2 ? RO:
    type==3 ? ROI:
    type==4 ? RC:
    type==5 ? RINH:
    type==6 ? 1:
    type==7 ? 0:
    0;

assign Enable_wire=
    type==5 ? CapDrvClk: // HighZ at the Test_Channel output
    Enable;
bufi1 driver(Test_Channel,Test_Channel_wire,Enable_wire);
endmodule

 (;;) High Speed Test Management *******
module Control(Data,Cntr1_Address,Enable_ERAM,Read,Write
    ,Clk,Status,Reset,Fail,HIL0,Counter);

inout[15:0] Data;
input
    Clk,Reset,Read;
input[2:0] Fail,Enable_ERAM;
input[2:0] Status;
input[4:0] Cntr1_Address;
in[1:0] HIL0;
output[23:0] Counter;
reg
    Fail_reg;
reg[23:0] Counter,Stop_addr,Start_addr;
reg[15:0] data_reg1;
reg[7:0] data_reg2;

Error_RAM u[255:0](Data,Cntr1_Address,Read,Write,Reset,Fail,Clk,
    Counter,Enable_ERAM,HIL0);

always @(posedge Clk)
begin
  if(Reset)
  begin
    Stop_addr=24'hFFFFFFF; // vector ram end address 4M
    Start_addr=0; // vector ram start address
    Counter=0; // vector ram address counter
    Fail_reg=0;
  end
else
  begin
    if(Fail==1) Fail_reg=1;
    case(Status)
      0: data_reg1=Data; // load start address
      1: data_reg2=Data[7:0];
      2: Start_addr={data_reg1,data_reg2};
      3: Stop_addr={data_reg1,data_reg2}; //load stop address
      4: Counter=Start_addr;
      5: begin
          Counter=Counter+1; // Loop_Test mode or single_step
          if(Counter > Stop_addr) Counter=Start_addr;
        end
      6: if(!Fail_reg) // Pass_loop test mode
        begin
          Counter=Counter+1; // Loop_Test mode or single_step
          if(Counter > Stop_addr) Counter=Start_addr;
        end
    endcase
  end
endmodule

//******************************************************** ERROR RAM *************

module Error_RAM(Data,Address,Read,Write,Reset,fail,\
                   Clk,Counter,Enable,HIL0);

  input             Read,Write,Reset,Clk,Enable;\   
  input            fail;\
  input[23:0]      Counter;\
  inout[15:0]      Data;\
  input[4:0]       Address;\
  input[1:0]       HIL0;\
  reg             Stop_reg;\
  reg              [4:0] error_cntr;\
  reg[23:0]       memory[0:31],outreg; \

100
wire Stop_reg_bar, read_cycle; \\
reg[5:0] i;

assign read_cycle= Enable & Read; \\
assign Stop_reg_bar=¬Stop_reg;\

assign Data=
 Enable==0 ? 16'bZZZZZZZZZZZZZZZ:
 (Enable==1 & HILO==0 & Read==1) ? outreg[23:8]:
 (Enable==1 & HILO==1 & Read==1) ? outreg[7:0]:
 (Enable==1 & HILO==2 & Read==1) ? 16'bZZZZZZZZZZZZZZZZ:
 16'bZZZZZZZZZZZZZZZ;

always @(posedge Clk) begin
  if(Reset)
    begin
      error_cntr=0; // error counter
      Stop_reg=0;
      for(i=0;i<32;i=i+1) memory[i]=0;
    end
  else if(Stop_reg_bar)
    begin
      error_cntr= error_cntr+1;
      if(error_cntr==5'b11111) Stop_reg=1;
      if(fail) memory[error_cntr]=Counter;
    end
  if(read_cycle) outreg=memory[Address];
end
endmodule

//************* VECTOR RAM *************
module Vector_RAM(Vector_Data, data, address,
  read, Write, enable, Clk); // outside of ToC

input read, Write, Clk, enable; \\
input[7:0] address; inout [15:0] data; \\
output[15:0] Vector_Data; \\
reg[15:0] memory; \\
wire read_bar;

//initial
//begin
//memory=16’h5;
//end

assign read_bar=¬read; \\

101
assign Vector_Data = (Write==1 & enable==1) ? memory : 0 ;
assign data = (read==1 & enable==1) ? memory+1 : 16’hZ;

always@(posedge Clk) if (enable & read_bar) memory=data;
endmodule

***************************************************************************** Test Bench *****************************
'timescale 1ns/100ps

module ToC128_tb(Data,Address,Clk,Reset,Read,Enable_Err_RAM,
                    Enable_Vector_RAM,Status,Test_Channel,HIL0);

output Reset,Read,Clk;
output[2:0] Status; \
output[1:0] HIL0; // Data bus is 16bits while counter is 24bits,
                // read is done in two phase High and Low
reg Reset,Read,Clk;\nreg[2:0] Status;\nreg[1:0] HIL0;\nreg[1:0] Enable_Vector_RAM,Enable_Err_RAM; \\
reg [23:0] Address;

assign Data=
(Read==0) ? 16’b00000000000000101 : 16’bZZZZZZZZZZZZZZZZZZ;
ToC256
    u1(Data,Address,Clk,Reset,Read,Enable_Err_RAM,
       Enable_Vector_RAM,Status,Test_Channel,HIL0);
initial begin $shm_open("ToC256.db");
$shm_probe(Data,Address,Clk,Reset,Read,Enable_Err_RAM,Enable_Vector_RAM,
            Status,Test_Channel,HIL0,u1.IO[1].Data,u1.Cntrl1.Status);
Address=0; \\
Reset=0; \\
Read=0; \\
HIL0=0;\nEnable_Vector_RAM=2;\nEnable_Err_RAM=1;\nStatus=1; \\
Clk=0;\n#100 Read=1; \\

102
#500 $finish;
end

always #3 Clk=#5 ~Clk;
always #3 Address=#20 Address+1;
always @(posedge Clk) #1type=#160 type+1;
endmodule
APPENDIX B

LIST OF ABBREVIATIONS

ATE - Automatic Test Equipment. An automated, usually computer-driven, approach to testing semiconductors, electronic circuits, and printed circuit board assemblies.

ATPG- Automatic Test Pattern Generator - Tool-based approach to generate test pattern that relies heavily on the design database and netlist.

At-Speed Scan - Form of scan where both the data shift and sample occur at the rated frequency of operation. Structure and timing performance can both be verified with this kind of scan test.

BILBO - A BILBO is a multitalented logic circuit that can be a state register, a scan register, an LFSR, or a MISR depending on the state of it's mode pins. BILBOs are sometimes used to cascade large combinational logic blocks in a BIST engine.

BIST - (Built in Self Test) BIST essentially builds tiny tester models onto the integrated circuit so that it can test itself.

Boundary Scan - Generic term for IEEE 1149.1. It is a methodology allowing complete controllability and observability of the boundary (I/O) pins via a standard interface. (AKA JTAG)

Catastrophic fault - These are faults such as open and short circuits that cause sudden and large variation in component values.

Channel - The tester functions and the path through a pin-group card and DUT fixture dedicated to one DUT pin.

CAD - Computer Aided Design

Crosstalk - A phenomenon in which one or more signals interfere with another signal.

Cyclic redundancy check (CRC) - A mathematically generated number
that data receivers use to verify the proper bit arrangement in a bit stream.

Defect(s) - Term used to reference specific flaw(s); physical or chemical imperfection, on a manufactured device. Most defects can be detected and measured by a Failure Analysis group. Specific devices that do not perform as expected contain defect(s) or have Design flaws.

Detectability - The degree to which it is possible, within the constraints of test equipment, to observe faulty output behavior of a faulty component. In other words,

DFT - (Design For Test) Design For Test is the practice of adding hardware hooks to integrated circuits in order to facilitate effective, inexpensive testing.

Diagnostic tests - Tests intended to determine the cause of a possible malfunction and to suggest a repair strategy.

Die - A piece of semiconductor with circuitry fabricated on it; one location on a wafer; compare chip.

DUT - Device Under Test (pronounced "dut") This is the target device being tested. Less frequently referred to as "CUT" (circuit under test).

EDA - electronic design automation. EDA refers to the design tools and environment utilized to render the logic, schematics, insert scan, insert BIST, etc. for a new chip design.

Edge - An abrupt voltage or current change.

Edge detection - A technique that locates an edge by examining an image for abrupt changes in pixel values. See also image processing.

Edge-placement accuracy - The precision within which an edge can be placed relative to a reference.

Fault(s) - This term is used in reference to classes, or concepts,
of defect types. The most common of these is the stuck at type, or fault class. In the EDA and academic worlds, a fault is a software model of a defect, or class of defects.

Fault Coverage - Quality measure for a test or set of tests, based on the percentage of actually detected faults (defects) versus the total number of theoretically detectable faults, on a particular fault model. A coverage figure should be given for each model type tested. As the operator defines the nodes to be evaluated (in some cases this is done by defining lists of nodes not to be used in the task) the raw number has little meaning without a full analysis of the set up.

Fault dictionary - A fault dictionary contains the same information contained in a fault list, but include information about how the fault manifests itself including the error location and its effect on circuit components. It may also include suggested test conditions.

Fault dominance - Dominance is the condition in which the deviation at the output caused by one fault is negligible compare with the deviation at the same output, but caused by a different fault.

Fault equivalence - Two faults are said to be equivalent if their effects on the output cannot be distinguished from each other.

Fault list- A fault list that contains every possible fault for consideration, but no information bout how the fault manifests itself under various circuit conditions and test conditions.

Fault masking - Two faults are said to be mask each other if their sensitivities are equal and opposite in sign.

Firmware - A program permanently recorded in ROM; it is effectively a piece of hardware that performs software functions.

Force - To stimulate DUT pins with formatted pattern data.

Format - To produce a waveform from pattern data and timing information in accordance with a format mode.

Full Scan - Scan architecture implementation where all of the
memory elements (flip-flops) are scannable in the design.

Functional test - A process that applies pattern vectors to a device and checks the output to determine that the device is operating according to its truth table.

Go/no-go test - A test with minimum and maximum limits that stops on the first error without performing any diagnostics, characterization, or actual measurements other than limit checking.

Hard fault - A hard fault is a potential open or short circuit in the design modeling the effects of manufacturing defects on the network connects.

IDDQ - Quiescent Supply Current (IDD). Theoretically, static CMOS logic should have nearly zero current when the clock is stopped. This methodology puts the chip into various logic states, stops the clock and measures the power supply current.

I/O channel - An input/output tester port that is capable of both stimulating a device pin and monitoring a response from the same pin.

JTAG - (Joint Test Action Group) Originally, the name of the team, through a not uncommon twist of fate, the term has come to be associated with the output of the team. JTAG is now essentially synonymous with the IEEE 1149.1 standard for Test Access Port and Boundary Scan.

LFSR - (Linear Feedback Shift Register) LFSRs are shift registers with exclusive-OR gates that allow some bits in the register (usually referred to as a polynomial) to feed back into selected points within the register. LFSRs are often used in BISTed designs to form PRPGs and MISRs.

MBIST - (Memory Built In Self Test) BIST approach that is specific to memory testing.

MISR - (Multiple Input Shift Register) Also known as Multiple Input Signature Register. MISRs are simply LFSRs configured as signature analyzers. MISRs are often used on the back end of BIST
engines to capture and compress output sequences from a circuit under test.

P1500 - Emerging IEEE standard for test wrappers (common -scan like test structures "wrapped around SOC cores).

Parametric faults - These faults are caused by variations in component parameter values produced by process or environmental changes.

Parametric test - The measurement and verification of terminal voltage and current characteristics at a device pin.

Partial Scan - Scan architecture (implementation) where only some of the storage elements (flip-flops) are scannable.

Path Delay - Fault characterized by a particular logic path being too slow to meet the overall timing requirements of a circuit.

Pattern - The binary data applied to and expected from a DUT during a functional test.

Pattern vector - The pattern data applied to and expected from a DUT during one test cycle.

Pin-Electronics- One of the main section of automatic test equipment, where the stimulus data are applied to the device under test and its responses are captured

Primary Input - Physical input from the outside world to a device, can be a signal input, a scan chain input, etc. (Note: In the case of cores in an SOC, the outside world may still be inside the chip).

Primary Output - Physical output to the outside world from a device, can be a signal output, a scan chain output, etc. (Note: In the case of cores in an SOC, the outside world may still be inside the chip).

PRPG - (Pseudo- Random Pattern Generator) PRPGs are LFSRs that are sometimes used on the front end of BIST engines to generate pseudo-random patterns to be presented to a circuit under test. Scan - DFT technique where traditional functional logic is reconfigured into "chains" for direct test access to internal
nodes.

Scan Chain - Serial organization of scan elements such that the first element of the chain is at a device input and the last element of the chain is at a device output. Devices may use single or multiple scan chains to 'capture' all of the scannable nodes.

SoC - (System on a Chip) Practice of integrating one or more processor cores, embedded memories, peripheral interfaces, and sometimes mixed signal circuits onto a single chip to form a complete (or nearly complete) system.

Soft fault (parametric fault) - Any fault caused by a parametric or process variation outside of tolerance or nominal values is called a soft fault.

STDF - (Standard Test Data Format) STDF is a standard output format for test results. There are numerous tools for post processing STDF generated files and performing statistical analysis on a population of tested devices.

STIL - (Standard Test Interface Language) STIL is an emerging standard for test development. It is supported as an output format by the ATPGs of most EDA toolsets.

Stimulus - An input signal which imitates action or reaction in a circuit, such as voltage or current.

Structural Testing - Strategy of testing integrated circuits that focuses on detecting manufacturing defects. Unlike functional or behavioral testing, defects are targeted directly.

STUMPS - (Self Test Using MISR and Parallel SRS) STUMPS is a common BIST architecture that combines a PRPG (or multiple SRSGs), multiple scan chains, and a MISR.

TAP - (Test Access Port - pronounced "tap") Part of the JTAG standard, the TAP is a 4 (or optionally 5) pin port to enable boundary scan.

Test protocol - A sequence of control operations required to perform a test. At the lowest level a test protocol is just a series of logic 0 and 1 applied to specified test control ports. It will typically also contain symbolic references to the test data that is to be applied to or observed at specified test data

109
or system data ports. Test protocols involve the activation of one or more test modes and may also contain pre-conditioning and post-conditioning functions or sequences.

Test vector - A test vector or test setup condition is a set of stimuli applied to the device-under-test (DUT) to elicit a known output at a given test measurement node.

Tester - An assembly of apparatus used for evaluating semiconductor devices.

USB - Universal Standard Port
APPENDIX C

VERILOG CODES FOR USB IP CORE

///////////////////////////////////////////////////////////////////////////
///
/// USB 1.1 function IP core
///
/// Author: Rudolf Usselmann
///     rudi@asics.ws
///
/// Downloaded from: http://www.opencores.org/cores/usbl_function/
///
///////////////////////////////////////////////////////////////////////////
///
/// Copyright (C) 2000-2002 Rudolf Usselmann
///     www.asics.ws
///     rudi@asics.ws
///
/// This source file may be used and distributed without
/// restriction provided that this copyright statement is not
/// removed from the file and that any derivative work contains
/// the original copyright notice and the associated disclaimer.
///
/// THIS SOFTWARE IS PROVIDED ‘‘AS IS’’ AND WITHOUT ANY
/// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
/// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
/// FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR
/// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
/// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
/// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
/// GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
/// BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
/// LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
/// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
/// OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
/// POSSIBILITY OF SUCH DAMAGE.
///
///////////////////////////////////////////////////////////////////////////
`include "usb1_defines.v"

/*
   // USB PHY Interface
   tx_dp, tx_dn, tx_oe,
   rx_d, rx_dp, rx_dn,
   These pins are a semi-standard interface to USB 1.1 transceivers.
   Just match up the signal names with the I/Os of the transceiver.
   // USB Misc
   phy_tx_mode, usb_rst,
   The PHY supports single ended and differential output to the
   transceiver. Depending on which device you are using, you have to
   tie the phy_tx_mode high or low. usb_rst is asserted whenever the
   host signals reset on the USB bus. The USB core will internally
   reset itself automatically. This output is provided for external
   */
logic that needs to be reset when the USB bus is reset.

    // Interrupts
dropped_frame, misaligned_frame,
crc16_err,
dropped_frame, misaligned_frame are interrupt to indicate error
conditions in Block Frame mode. crc16_err, indicates when a crc 16
error was detected on the payload of a USB packet.

    // Vendor Features
v_set_int, v_set_feature, wValue,
wIndex, vendor_data,
This signals allow to control vendor specific registers and logic
that can be manipulated and monitored via the control endpoint
through vendor defined commands.

    // USB Status
usb_busy, ep_sel,
usb_busy is asserted when the USB core is busy transferring data
ep_sel indicated the endpoint that is currently busy. This
information might be useful if one desires to reset/clear the
attached FIFOs and want to do this when the endpoint is idle.

    // Endpoint Interface
This implementation supports 8 endpoints. Endpoint 0 is the
control endpoint and used internally. Endpoints 1-7 are available
to the user. replace 'N' with the endpoint number.

    epN_cfg,
This is a constant input used to configure the endpoint by ORing
these defined together and adding the max packet size for this
endpoint: 'IN and 'OUT select the transfer direction for this
endpoint 'ISO, 'BULK and 'INT determine the endpoint type

Example: "'BULK | 'IN | 14'd064" defines a BULK IN endpoint with
max packet size of 64 bytes

    epN_din, epN_we, epN_full,
This is the OUT FIFO interface. If this is an IN endpoint, ground
all unused inputs and leave outputs unconnected.

    epN_dout, epN_re, epN_empty,
this is the IN FIFO interface. If this is an OUT endpoint ground
all unused inputs and leave outputs unconnected.
epN_{bf\_en}, epN_{bf\_size},
These two constant configure the Block Frame feature.
*/

define\_usb\_core(clk\_i, rst\_i,
    // USB PHY Interface
    tx\_dp, tx\_dn, tx\_oe,
    rx\_d, rx\_dp, rx\_dn,

    // USB Misc
    phy\_tx\_mode, usb\_rst,

    // Interrupts
    dropped\_frame, misaligned\_frame, crc16\_err,

    // Vendor Features
    v\_set\_int, v\_set\_feature, wValue,
    wIndex, vendor\_data,

    // USB Status
    usb\_busy, ep\_sel,

    // Endpoint Interface
    ep1\_cfg,
    ep1\_din, ep1\_we, ep1\_full,
    ep1\_dout, ep1\_re, ep1\_empty,
    ep1\_bf\_en, ep1\_bf\_size,

    ep2\_cfg,
    ep2\_din, ep2\_we, ep2\_full,
    ep2\_dout, ep2\_re, ep2\_empty,
    ep2\_bf\_en, ep2\_bf\_size,

    ep3\_cfg,
    ep3\_din, ep3\_we, ep3\_full,
    ep3\_dout, ep3\_re, ep3\_empty,
    ep3\_bf\_en, ep3\_bf\_size,

    ep4\_cfg,
    ep4\_din, ep4\_we, ep4\_full,
    ep4\_dout, ep4\_re, ep4\_empty,
ep4_bf_en, ep4_bf_size,

ep5_cfg,
ep5_din, ep5_we, ep5_full,
ep5_dout, ep5_re, ep5_empty,
ep5_bf_en, ep5_bf_size,

ep6_cfg,
ep6_din, ep6_we, ep6_full,
ep6_dout, ep6_re, ep6_empty,
ep6_bf_en, ep6_bf_size,

ep7_cfg,
ep7_din, ep7_we, ep7_full,
ep7_dout, ep7_re, ep7_empty,
ep7_bf_en, ep7_bf_size

);

input clk_i; input rst_i;

output tx_dp, tx_dn, tx_oe; input rx_d, rx_dp, rx_dn;

input phy_tx_mode; output usb_rst; output dropped_frame, misaligned_frame; output crc16_err;

output v_set_int; output v_set_feature; output [15:0] wValue; output [15:0] wIndex; input [15:0] vendor_data;

output usb_busy; output [3:0] ep_sel;

// Endpoint Interfaces
input [13:0] ep1_cfg; input [7:0] ep1_din; output [7:0] ep1_dout; output ep1_we, ep1_re; input ep1_empty,
ep1_full; input ep1_bf_en; input [6:0] ep1_bf_size;

input [13:0] ep2_cfg; input [7:0] ep2_din; output [7:0] ep2_dout; output ep2_we, ep2_re; input ep2_empty,
ep2_full; input ep2_bf_en; input [6:0] ep2_bf_size;

input [13:0] ep3_cfg; input [7:0] ep3_din; output [7:0] ep3_dout; output ep3_we, ep3_re; input ep3_empty,
ep3_full; input ep3_bf_en; input [6:0] ep3_bf_size;

input [13:0] ep4_cfg; input [7:0] ep4_din; output [7:0]
ep4_dout; output ep4_we, ep4_re; input ep4_empty,
ep4_full; input ep4_bf_en; input [6:0] ep4_bf_size;

input [13:0] ep5_cfg; input [7:0] ep5_din; output [7:0]
ep5_dout; output ep5_we, ep5_re; input ep5_empty,
ep5_full; input ep5_bf_en; input [6:0] ep5_bf_size;

input [13:0] ep6_cfg; input [7:0] ep6_din; output [7:0]
ep6_dout; output ep6_we, ep6_re; input ep6_empty,
ep6_full; input ep6_bf_en; input [6:0] ep6_bf_size;

input [13:0] ep7_cfg; input [7:0] ep7_din; output [7:0]
ep7_dout; output ep7_we, ep7_re; input ep7_empty,
ep7_full; input ep7_bf_en; input [6:0] ep7_bf_size;


// Local Wires and Registers

// UTMI Interface
wire [7:0] DataOut; wire TxValid; wire TxReady;
wire [7:0] DataIn; wire RxValid; wire RxActive;
wire RxError; wire [1:0] LineState;
wire [7:0] rx_data; wire rx_valid, rx_active, rx_err;
wire [7:0] tx_data; wire tx_valid; wire
tx_ready; wire tx_first; wire tx_valid_last;

// Internal Register File Interface
wire [6:0] funct_adr; // This functions address (set by
controller) wire [3:0] ep_sel; // Endpoint Number Input
wire crc16_err; // Set CRC16 error interrupt wire
int_to_set; // Set time out interrupt wire int_seqerr_set;
// Set PID sequence error interrupt wire [31:0] frm_nat; //
Frame Number and Time Register wire nse_err; // No Such
Endpoint Error wire pid_cs_err; // PID CS error wire
crc5_err; // CRC5 Error
reg [7:0] tx_data_st; wire [7:0] rx_data_st; reg [13:0]
cfg; reg ep_empty; reg ep_full; wire [7:0] rx_size;
wire rx_done;

wire [7:0] ep0_din; wire [7:0] ep0_dout; wire
ep0_re, ep0_we; wire [13:0] ep0_cfg; wire [7:0] ep0_size;
wire [7:0] ep0_ctrl dout, ep0_ctrl din; wire ep0_ctrl_re, ep0_ctrl_we; wire [3:0] ep0_ctrl_stat;
wire ctrl_setup, ctrl_in, ctrl_out; wire send_stall;
wire token_valid; reg rst_local; // internal reset
wire dropped_frame; wire misaligned_frame; wire v_set_int; wire v_set_feature; wire [15:0] wValue; wire [15:0] wIndex;
reg ep_bf_en; reg [6:0] ep_bf_size; wire [6:0] rom_adr;
wire [7:0] rom_data;

///////////////////////////////////////////////////////
//
// Misc Logic
//

// Endpoint type and Max transfer size
assign ep0_cfg = 'CTRL | ep0_size;

always @(posedge clk_i)
  rst_local <= #1 rst_i & ~usb_rst;

///////////////////////////////////////////////////////
//
// Module Instantiations
//

usb_phy phy(
  .clk(clk_i),
  .rst(rst_i), // ONLY external reset
  .phy_tx_mode(phy_tx_mode),
  .usb_rst(usb_rst),

  // Transceiver Interface
  .rx_d(),
  .rx_dp(),
  .rx_dn(),
  .tx_dp(),
  .tx_dn(),
  .tx oe(),

  // UTMI Interface
  .DataIn_o(DataIn),
  .RxValid_o(RxValid),

117
.RxActive_o( RxActive ),
.RxError_o( RxError ),
.DataOut_i( DataOut ),
.TxValid_i( TxValid ),
.TxReady_o( TxReady ),
.LineState_o( LineState );

// UTMII Interface
usb1_utmi_if u0(
  .phy_clk(    clk_i ),
  .rst(        rst_local ),
  .DataOut(    DataOut ),
  .TxValid(    TxValid ),
  .TxReady(    TxReady ),
  .RxValid(    RxValid ),
  .RxActive(   RxActive ),
  .RxError(    RxError ),
  .DataIn(     DataIn ),
  .rx_data(    rx_data ),
  .rx_valid(   rx_valid ),
  .rx_active(  rx_active ),
  .rx_err(     rx_err ),
  .tx_data(    tx_data ),
  .tx_valid(   tx_valid ),
  .tx_valid_last( tx_valid_last ),
  .tx_ready(   tx_ready ),
  .tx_first(   tx_first );

// Protocol Layer
usb1_pl    u1(    .clk(    clk_i ),
  .rst(      rst_local ),
  .rx_data(   rx_data ),
  .rx_valid(  rx_valid ),
  .rx_active( rx_active ),
  .rx_err(    rx_err ),
  .tx_data(   tx_data ),
  .tx_valid(  tx_valid ),
  .tx_valid_last( tx_valid_last ),
  .tx_ready(  tx_ready ),
  .tx_first(  tx_first ),
  .tx_valid_out( TxValid ),
  .token_valid( token_valid ),
  .fa(        funct adr );
.ep_sel( ep_sel ),
.x_busy( usb_busy ),
.int_crc16_set( crc16_err ),
.int_to_set( int_to_set ),
.int_seqerr_set( int_seqerr_set ),
.frm_nat( frm_nat ),
.pid_cs_err( pid_cs_err ),
.nse_err( nse_err ),
.crc5_err( crc5_err ),
.rx_size( rx_size ),
.rx_done( rx_done ),
.ctrl_setup( ctrl_setup ),
.ctrl_in( ctrl_in ),
.ctrl_out( ctrl_out ),
.ep_bf_en( ep_bf_en ),
.ep_bf_size( ep_bf_size ),
.dropped_frame( dropped_frame ),
.misaligned_frame( misaligned_frame ),
.csr( cfg ),
.tx_data_st( tx_data_st ),
.rx_data_st( rx_data_st ),
.idma_re( idma_re ),
.idma_we( idma_we ),
.ep_empty( ep_empty ),
.ep_full( ep_full ),
.send_stall( send_stall );

usb1_ctrl u4( .clk( clk_i ),
.rst( rst_local ),
.rom_adr( rom_adr ),
.rom_data( rom_data ),
.ctrl_setup( ctrl_setup ),
.ctrl_in( ctrl_in ),
.ctrl_out( ctrl_out ),
.ep0_din( ep0_ctrl_dout ),
.ep0_dout( ep0_ctrl_din ),
.ep0_re( ep0_ctrl_re ),
.ep0_we( ep0_ctrl_we ),
.ep0_stat( ep0_ctrl_stat ),
.ep0_size( ep0_size ),

119
.send_stall (    send_stall    ),
.frame_no (    frm_n[26:16]    ),
.funct_addr (    funct_addr    ),
.configured (    ),
.halt (    ),
.v_set_int (    v_set_int    ),
.v_set_feature (    v_set_feature    ),
wValue (    wValue    ),
wIndex (    wIndex    ),
.vendor_data (    vendor_data    );

usbi_rom1 rom1 (.clk (    clk_i    ),
.adr (    rom_adr    ),
.dout (    rom_data    );

// CTRL Endpoint FIFO
generic_fifo_sc_a #(8,6,0) u10 (  
    .clk (    clk_i    ),
    .rst (    rst_i    ),
    .clr (    usb_rst    ),
    .din (    rx_data_st    ),
    .we (    ep0_we    ),
    .dout (    ep0_ctrl_dout    ),
    .re (    ep0_ctrl_re    ),
    .full_r (    ),
    .empty_r (    ),
    .full (    ep0_full    ),
    .empty (    ep0_ctrl_stat[1]    ),
    .full_n (    ),
    .empty_n (    ),
    .full_n_r (    ),
    .empty_n_r (    ),
    .level (    );

generic_fifo_sc_a #(8,6,0) u11 (  
    .clk (    clk_i    ),
    .rst (    rst_i    ),
    .clr (    usb_rst    ),
    .din (    ep0_ctrl_din    ),
    .we (    ep0_ctrl_we    ),
    .
always @(ep_sel or ep0_cfg or ep1_cfg or ep2_cfg or ep3_cfg or ep4_cfg or ep5_cfg or ep6_cfg or ep7_cfg)
  case(ep_sel)  // synopsys full_case parallel_case
  4'h0:   cfg = ep0_cfg;
  4'h1:   cfg = ep1_cfg;
  4'h2:   cfg = ep2_cfg;
  4'h3:   cfg = ep3_cfg;
  4'h4:   cfg = ep4_cfg;
  4'h5:   cfg = ep5_cfg;
  4'h6:   cfg = ep6_cfg;
  4'h7:   cfg = ep7_cfg;
endcase

// In endpoints only
always @(posedge clk_i)
  case(ep_sel)  // synopsys full_case parallel_case
  4'h0:   tx_data_st <= #1 ep0_dout;
  4'h1:   tx_data_st <= #1 ep1_din;
  4'h2:   tx_data_st <= #1 ep2_din;
  4'h3:   tx_data_st <= #1 ep3_din;
  4'h4:   tx_data_st <= #1 ep4_din;
  4'h5:   tx_data_st <= #1 ep5_din;
  4'h6:   tx_data_st <= #1 ep6_din;
  4'h7:   tx_data_st <= #1 ep7_din;
endcase

// In endpoints only
always @(posedge clk_i)
  case(ep_sel)  // synopsys full_case parallel_case
    4'h0: ep_empty <= #1 ep0_empty;
    4'h1: ep_empty <= #1 ep1_empty;
    4'h2: ep_empty <= #1 ep2_empty;
    4'h3: ep_empty <= #1 ep3_empty;
    4'h4: ep_empty <= #1 ep4_empty;
    4'h5: ep_empty <= #1 ep5_empty;
    4'h6: ep_empty <= #1 ep6_empty;
    4'h7: ep_empty <= #1 ep7_empty;
  endcase

// OUT endpoints only
always @(ep_sel or ep0_full or ep1_full or ep2_full or ep3_full or ep4_full or ep5_full or ep6_full or ep7_full)
  case(ep_sel)  // synopsys full_case parallel_case
    4'h0: ep_full = ep0_full;
    4'h1: ep_full = ep1_full;
    4'h2: ep_full = ep2_full;
    4'h3: ep_full = ep3_full;
    4'h4: ep_full = ep4_full;
    4'h5: ep_full = ep5_full;
    4'h6: ep_full = ep6_full;
    4'h7: ep_full = ep7_full;
  endcase

always @(posedge clk_i)
  case(ep_sel)  // synopsys full_case parallel_case
    4'h0: ep_bf_en = 1'b0;
    4'h1: ep_bf_en = ep1_bf_en;
    4'h2: ep_bf_en = ep2_bf_en;
    4'h3: ep_bf_en = ep3_bf_en;
    4'h4: ep_bf_en = ep4_bf_en;
    4'h5: ep_bf_en = ep5_bf_en;
    4'h6: ep_bf_en = ep6_bf_en;
    4'h7: ep_bf_en = ep7_bf_en;
  endcase

always @(posedge clk_i)
  case(ep_sel)  // synopsys full_case parallel_case
    4'h1: ep_bf_size = ep1_bf_size;
    4'h2: ep_bf_size = ep2_bf_size;
    4'h3: ep_bf_size = ep3_bf_size;
    4'h4: ep_bf_size = ep4_bf_size;
    4'h5: ep_bf_size = ep5_bf_size;
4'h6:    ep_bf_size = ep6_bf_size;
4'h7:    ep_bf_size = ep7_bf_size;
endcase

assign ep1_dout = rx_data_st; assign ep2_dout = rx_data_st; assign
ep3_dout = rx_data_st; assign ep4_dout = rx_data_st; assign
ep5_dout = rx_data_st; assign ep6_dout = rx_data_st; assign
ep7_dout = rx_data_st;

assign ep0_re = idma_re & (ep_sel == 4'h00); assign ep1_re =
idma_re & (ep_sel == 4'h01) & !ep1_empty; assign ep2_re = idma_re
& (ep_sel == 4'h02) & !ep2_empty; assign ep3_re = idma_re &
(ep_sel == 4'h03) & !ep3_empty; assign ep4_re = idma_re & (ep_sel
== 4'h04) & !ep4_empty; assign ep5_re = idma_re & (ep_sel ==
4'h05) & !ep5_empty; assign ep6_re = idma_re & (ep_sel == 4'h06) &
!ep6_empty; assign ep7_re = idma_re & (ep_sel == 4'h07) &
!ep7_empty;

assign ep0_we = idma_we & (ep_sel == 4'h00); assign ep1_we =
idma_we & (ep_sel == 4'h01) & !ep1_full; assign ep2_we = idma_we &
(ep_sel == 4'h02) & !ep2_full; assign ep3_we = idma_we & (ep_sel
== 4'h03) & !ep3_full; assign ep4_we = idma_we & (ep_sel == 4'h04)
& !ep4_full; assign ep5_we = idma_we & (ep_sel == 4'h05) &
!ep5_full; assign ep6_we = idma_we & (ep_sel == 4'h06) &
!ep6_full; assign ep7_we = idma_we & (ep_sel == 4'h07) &
!ep7_full;
endmodule
VITA AUCTORIS

- Name: Rashid Rashidzadeh
- Place of Birth: Iran
- Year of Birth: 1963
- Education:
  - Lalehaye engelab, Tehran, Iran.
    1979-1982
  - Sharif University of Technology, Tehran, Iran.
    1984-1989 B.Sc.
  - University of Windsor, Ontario, Canada.
    2001-2003 M.A.Sc.