University of Cincinnati

Date: 11/14/2011

I, Giridhar R Jaggannagari, hereby submit this original work as part of the requirements for the degree of Master of Science in Electrical Engineering.

It is entitled:
Bridging And Open Faults Detection In A Two Flip-Flop Synchronizer

Student's name: Giridhar R Jaggannagari

This work and its defense approved by:

Committee chair: Wen Ben Jone, PhD
Committee member: Carla Purdy, PhD
Committee member: Ranganarha Vemuri, PhD
Bridging And Open Faults Detection In A Two Flip-Flop Synchronizer

A thesis submitted to the faculty of
University Of Cincinnati,OH
in partial fulfillment of the requirements for the degree of
Master of Science
Department Of Electrical Engineering
University Of Cincinnati
January 30, 2012
by
Giridhar R Jaggannagari

Bachelor of Engineering (Electrical and Electronics Engineering)
Osmania University,India
June 2006

Thesis Advisor and Committee Chair: Dr. Wen Ben Jone
ABSTRACT

As the device size is shrinking and designs have become increasingly complex, multimillion transistor systems have emerged running with multiple asynchronous clocks with frequencies as high as multiple gigahertz. SoC systems have multiple interfaces and cores which run on different frequencies, some of which can be totally asynchronous to the other cores. Several modern serial interfaces are inherently asynchronous from the rest of the chip [1]. Thus one of the most challenging issues is to provide reliable communication between mutually asynchronous intellectual property (IP) cores in an SoC [2].

In this research, we implement a handshake protocol between two asynchronous clock domains using two flip flop synchronizers. All the bridging and open defects that can possibly occur in these synchronizers have been explained in detail in [8] and [9]. We then develop a testing methodology using Launch-on-Capture method to test for all the structural, functional and delay faults caused by bridging and open defects in the synchronizer. We have also extended this method to a SOC where each core is wrapped by an IEEE 1500 standard compliant wrapper using Turbo1500 tool provided by Syntest Technologies.
ACKNOWLEDGMENTS

I wish to express my sincere thanks to my advisor, Dr. Wen-ben Jone, for the guidance and constructive criticism that he provided throughout the whole work. He was always ready to provide guidance, and to encourage me to do better work. Thank you Dr. Jone, for all the great help and sincere concern.

I would like to thank the members of my committee, Dr. Ranga Vemuri and Prof. Carla Purdy for spending their valuable time reviewing this work. I would like to offer special thanks to Hyoung-Kook Kim for his help and guidance. Last, but not the least, I would like to express my gratitude to my parents and my brother, for their constant support and love.
# Contents

## Table of Contents

v

## List of Figures

ix

## List of Tables

xi

## 1 Background

1.1 Why Handshaking? .................................................. 2
1.2 Types of Synchronizers ........................................... 2
1.3 Faults in a synchronizer ................................. 5
  1.3.1 Bridging Faults ........................................ 5
  1.3.2 Open Faults ........................................ 6
1.4 Testing Strategy ........................................ 6

## 2 Interface Design Between Asynchronous Clock Domains

2.1 Control Logic .................................................. 12
2.2 S27 Benchmark Circuit ....................................... 14
2.3 Handshake Source Module ............................ 17
2.4 Handshake Destination Module ........................... 20

## 3 Testable Interface Design

3.1 Scan Design In Source Module .................................. 23
3.2 Scan Design In Destination Module .......................... 25
3.3 Test Clock Generator ....................................... 26
3.4 Programmable Delay Generator ............................. 30
3.5 Delay Equations for Test Clock Generator ............... 30
3.6 Structural Fault Testing Of Synchronizers ................. 36
  3.6.1 Capture Window For Domain-2 .................. 36
3.6.2 Stuck-At ‘0’ Fault In Domain-2 ........................................ 37  
3.6.3 Stuck-At ‘1’ Fault In Domain-2 ........................................ 37  
3.6.4 Capture Window for Domain-1 ........................................ 38  
3.6.5 Stuck-At ‘0’ Fault In Domain-1 ........................................ 38  
3.6.6 Stuck-At ‘1’ Fault In Domain-1 ........................................ 39  

3.7 Delay Fault Testing Of Synchronizer .................................... 39  
3.7.1 Detecting Late Transition Fault ....................................... 40  
3.7.2 Detecting Early Transition Fault ...................................... 41  
3.7.3 Detecting Inverted Output Fault ...................................... 42  
3.7.4 Detecting Input Dependent Pulses ..................................... 42  
3.7.5 Detecting Input Independent Pulses ................................... 44  
3.7.6 Detecting Internal Oscillation Fault .................................. 44  
3.7.7 Detecting One Time Pulse Fault ....................................... 44  
3.7.8 Metastable Output .......................................................... 46  
3.7.9 Undefined Output Logic .................................................. 47  

3.8 Detecting Open Faults In A Synchronizer ............................ 48  
3.8.1 Detecting Stuck-at Fault(SA) ........................................... 48  
3.8.2 Detecting Pulse fault ..................................................... 49  
3.8.3 Detecting One Time Pulse fault (OTPF) ............................. 49  

4 Wrapper Design For The Cores ........................................... 51  
4.1 The IEEE 1500 Wrapper Architecture .................................. 51  
4.2 IEEE 1500 Standard Instructions ....................................... 53  
4.3 Wrapper Boundary Register (WBR) ...................................... 55  
4.3.1 Parallel configuration of the WBR .................................... 55  
4.4 Wrapper Bypass Register (WBY) ......................................... 58  
4.5 Wrapper Instruction Register (WIR) .................................... 58  
4.5.1 WIR Operation ............................................................. 60  
4.6 Using Turbo1500 to generate the wrapper ............................ 61  
4.6.1 Embedded-Core-Interface -Test Control File.(cfg) .............. 62  
4.6.2 Library File (.lib) ......................................................... 68  
4.6.3 Wrapper Chain File (.wpso) ............................................. 69  
4.6.4 Scan Chain File (.pso) .................................................... 71  
4.6.5 Pin File (.pin) ............................................................ 73  
4.7 Testing of the wrapped cores ............................................. 77  
4.7.1 Test Patterns for wrapped cores ...................................... 78  

5 Conclusion ................................................................. 81  

Bibliography ................................................................. 83  

A Faults in a synchronizer .................................................. 85  
A.1 Bridging Faults ......................................................... 86
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B  Verilog Code</td>
<td>93</td>
</tr>
<tr>
<td>C  Turbo1500 Files</td>
<td>121</td>
</tr>
<tr>
<td>C.1 Input files for s27 receiver core</td>
<td>121</td>
</tr>
</tbody>
</table>
# List of Figures

1.1 Edge-detecting Synchronizer ............................................... 4
1.2 Pulse Synchronizer .......................................................... 5
1.3 Schematic of Synchronizer ..................................................... 6
1.4 Capture Window Schemes ....................................................... 9

2.1 Block Diagram ................................................................. 12
2.2 START-GEN ................................................................. 12
2.3 HS-DONE-GEN .............................................................. 13
2.4 EXECUTE-GEN ............................................................... 13
2.5 Waveform of Control Logic ................................................... 15
2.6 S27 Source Circuit ........................................................... 15
2.7 S27 Destination Circuit ...................................................... 16
2.8 Handshake Source Module .................................................. 18
2.9 Waveform of Handshake Source Module .................................. 19
2.10 Handshake Destination Module ........................................... 20
2.11 Source Side ................................................................. 21
2.12 Destination Side ............................................................ 22

3.1 Scan Chain In Source ......................................................... 24
3.2 Modified HS-DONE-GEN .................................................. 25
3.3 Scan Chain In Destination ................................................... 26
3.4 Test Clock Generator ....................................................... 27
3.5 Implementation Of Launch Clock Generator .............................. 28
3.6 Implementation Of Capture Clock Generator ............................ 29
3.7 Waveform Of Capture Clock Generator ................................... 29
3.8 Programmable Delay Generator .......................................... 31
3.9 Block diagram for Delay .................................................... 32
3.10 Waveform to calculate ‘d’ .................................................. 32
<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.11</td>
<td>Waveform of Fig 4.1</td>
<td>33</td>
</tr>
<tr>
<td>3.12</td>
<td>Clock Selector Logic</td>
<td>34</td>
</tr>
<tr>
<td>3.13</td>
<td>Domain-Sel = ‘0’</td>
<td>34</td>
</tr>
<tr>
<td>3.14</td>
<td>Domain-Sel = ‘1’</td>
<td>34</td>
</tr>
<tr>
<td>3.15</td>
<td>Complete Design</td>
<td>35</td>
</tr>
<tr>
<td>3.16</td>
<td>Capture Window For Domain-2</td>
<td>36</td>
</tr>
<tr>
<td>3.17</td>
<td>Capture Window For Domain-1</td>
<td>38</td>
</tr>
<tr>
<td>3.18</td>
<td>Inverted Capture Window For Domain-2</td>
<td>41</td>
</tr>
<tr>
<td>3.19</td>
<td>Pulse Fault whose frequency is $f_{clk}$</td>
<td>43</td>
</tr>
<tr>
<td>3.20</td>
<td>OTP0 by bridging defect</td>
<td>45</td>
</tr>
<tr>
<td>3.21</td>
<td>OTP1 by bridging defect</td>
<td>46</td>
</tr>
<tr>
<td>3.22</td>
<td>Voltage Window Detector</td>
<td>47</td>
</tr>
<tr>
<td>3.23</td>
<td>Pulse Fault by open defect whose frequency is $f_{clk}$</td>
<td>49</td>
</tr>
<tr>
<td>3.24</td>
<td>One Time Pulse Fault by open defect</td>
<td>50</td>
</tr>
<tr>
<td>4.1</td>
<td>Top level representation of the cores</td>
<td>52</td>
</tr>
<tr>
<td>4.2</td>
<td>Architecture of IEEE 1500 wrapped core</td>
<td>53</td>
</tr>
<tr>
<td>4.3</td>
<td>WC_SD1_CII</td>
<td>55</td>
</tr>
<tr>
<td>4.4</td>
<td>Example parallel configuration of core chains and WBR</td>
<td>57</td>
</tr>
<tr>
<td>4.5</td>
<td>Wrapper Instruction Register</td>
<td>59</td>
</tr>
<tr>
<td>4.6</td>
<td>Figure for <code>.cfg</code> file</td>
<td>67</td>
</tr>
<tr>
<td>4.7</td>
<td>Figure for <code>.wpso</code> file</td>
<td>71</td>
</tr>
<tr>
<td>4.8</td>
<td>Wrapped Source and Destination Cores</td>
<td>77</td>
</tr>
<tr>
<td>4.9</td>
<td>Timing diagram for fault detection in synchronizer using wrapper design</td>
<td>80</td>
</tr>
<tr>
<td>A.1</td>
<td>Schematic of Synchronizer</td>
<td>85</td>
</tr>
</tbody>
</table>
1.1 Category of all bridging faults ................................. 7
1.2 Category of all Open faults ........................................ 8

3.1 Category of all open faults ......................................... 48

4.1 IEEE 1500 Mandatory Instruction List ............................ 54
4.2 Control values for wrapper configuration in parallel mode ........ 57
4.3 Instruction and Opcodes ............................................. 59
4.4 Notations of PIN_TYPE field ....................................... 73
4.5 Notations of signal status ........................................... 73

A.1 Category of all bridging faults ................................... 87
A.2 Fault behaviors with bridging defects in DFF1 ........................ 88
A.3 Fault behaviors with bridging defects in DFF2 ........................ 89
A.4 Fault behaviors with bridging defects between DFF1 & DFF2 .... 90
The most important thing in an SOC is to provide proper synchronization between clock domain crossing signals. The basic step in using multiclock designs is to understand the problem of signal stability. When a signal crosses a clock domain, it appears to the circuitry in the new clock domain as an asynchronous signal. Successful data transfer between mutually asynchronous clock domains needs safe synchronization [3]. Synchronization prevents the metastable state of the first storage element (flip-flop) in the new clock domain from propagating through the circuit. Metastability is the inability of a flip-flop to arrive at a known state in a specific amount of time. For a flip-flop in metastable state, we can predict neither its output voltage level nor the time required for the output to settle to a correct voltage level. During this settling time, the flip-flop’s output is at some intermediate voltage level or may oscillate, and can cascade the invalid output level to flip-flops farther down the signal path. Synchronizer reliability is typically expressed in terms of Mean Time Between Failures [4]:

\[
MTBF = \frac{e^{T/\tau}}{T_W f_A f_B}
\]  

(1.1)
where $T$ represents the settling window, $\tau$ is the settling time constant of the flop, $T_W$ is a parameter related to its time window of susceptibility, $f_A$ is the synchronizer’s clock frequency, and $f_B$ is the frequency of pushing data across the clock domain boundary.

### 1.1 Why Handshaking?

As we have already discussed before, synchronizers are used to transfer data between two mutually asynchronous clock domains. Let us assume that data has to be transferred between two asynchronous cores over a bus, so an obvious solution that comes to mind is to use a synchronizer on each of the data lines in the bus. But this scheme is not a correct solution because, each of the several data synchronizers may end up doing something different. Some may sample the new data, others may skip it and retain the old data, while the remaining may enter metastability. Of those which enter metastability, some may settle to ‘1’ while others may settle to ‘0’. There is no way of telling which is which, as all four options are equally probable [6]. This problem can be solved using methods like handshaking protocols and FIFO design. In the handshaking method, request and acknowledge signals are used for data transfer between the sending and receiving domains. So it is sufficient if we synchronize these two signals instead of using synchronizers on all data lines of the bus.

### 1.2 Types of Synchronizers

Several types of synchronizers to avoid metastability are introduced in [5] depending on the level of asynchrony between the data signal and the local clock. Ginosar [6] has presented several cases of preventing safe synchronization, analyzed the causes
of errors and provided the correct synchronization circuits. Typically, industries provide a synchronizer cell for signal synchronization. It comprises of a flip-flop with a very high gain that consumes more power and is larger in size than a standard flip-flop. Such a flip-flop has reduced setup and hold time requirements for the input signal and is resistant to oscillation when the input signal causes a metastable condition. Another type of a simple synchronizer comprises two flip-flops in series with no combinational circuitry between them. This ensures that the first flip-flop exits its metastable state and its output settles before the second flip-flop samples it. Also the flip flops have to be placed close to each other to ensure the smallest possible clock skew between them [7].

Synchronizers basically fall into three categories :

**Level synchronizer:** This is the basic two flip-flop synchronizer discussed above and it is shown as part of Fig.1.1. In a level synchronizer, the signal crossing a clock domain stays high and stays low for more than two clock cycles in the new clock domain. The circuit requires that the signal to change to its invalid state before it can become valid again. Each time the signal goes valid, the receiving logic considers it a single event, no matter how long the signal remains valid.

**Edge-detecting synchronizer:** The edge-detecting synchronizer circuit adds a flip-flop to the output of the level synchronizer as shown in Fig.1.1. The output of the additional flip-flop is inverted and ANDed with the output of the level synchronizer. This circuit detects the rising edge of the input to the synchronizer and generates a clockwise, active-high pulse. Switching the inverter on the AND gate inputs creates a synchronizer that detects the falling edge of the input signal. Changing the AND gate to a NAND gate results in a circuit that generates an active-low pulse. The edge-detecting synchronizer works well at
synchronizing a pulse going to a faster clock domain. This circuit produces a pulse that indicates the rising or falling edge of the input signal. One restriction of this synchronizer is that the width of the input pulse must be greater than the period of the synchronizer clock plus the required hold time of the first synchronizer flip-flop. The safest pulse width is twice the synchronizer clock period. This synchronizer does not work if the input is a single clockwide pulse entering a slower clock domain; however, the pulse synchronizer solves this problem.

![Figure 1.1: Edge-detecting Synchronizer](image)

**Pulse Synchronizer:** The input signal of a pulse synchronizer is a single clockwide pulse that triggers a toggle circuit in the originating clock domain as shown in Fig.1.2. The output of the toggle circuit switches from high to low and vice versa each time it receives a pulse and passes through the level synchronizer to arrive at one input of the XOR gate, while a one clock cycle delayed version goes to the other input of the XOR. For one clock cycle, each time the toggle circuit changes state, the output of this synchronizer generates a single clockwide pulse. The basic function of a pulse synchronizer is to take a single clockwide pulse from one clock domain and create a single clockwide pulse in the new domain. One restriction of a pulse synchronizer is that input pulses must have a minimum spacing between pulses equal to two synchronizer clock periods. If the input
pulses are closer, the output pulses in the new clock domain are adjacent to each other, resulting in an output pulse that is wider than one clock cycle. This problem is more severe when the clock period of input pulse is greater than twice the synchronizer clock period. In this case, if the input pulses are too close, the synchronizer does not detect every one [7].

![Figure 1.2: Pulse Synchronizer](image)

The other types of synchronizers include one flop synchronizer, global reset synchronizer, conservative synchronizer and several others which are discussed in [6].

### 1.3 Faults in a synchronizer

This section provides a brief introduction to different kinds of faults in a two flip-flop synchronizer [8]. The faults can be categorized into two types:

- Bridging Faults
- Open Faults

#### 1.3.1 Bridging Faults

Fig 1.3 below shows the schematic of a synchronizer used in the handshake protocol. The synchronizer consists of two positive edge-triggered D flip flops, each
Chapter 1 Background

consisting of four transmission gates and four inverters. Table 1.1 shows all the different kinds of bridging faults that can occur in a two flip flop synchronizer. The first column gives the categories of all fault behaviors. The second column shows more detailed fault types belonging to each category, and the detailed fault behavior of each type is described in the third column. All the faults have been explained in detail in [8].

![Schematic of Synchronizer](image)

**Figure 1.3: Schematic of Synchronizer**

### 1.3.2 Open Faults

The synchronizer used for the analysis of open faults is same as the one used in bridging faults. All cases of open defects in the synchronizer: open defects in a transmission gate, and open defects in an inverter are considered. All the open faults listed in Table 1.2 are explained in detail in [9].

Table 1.2 shows the categories of all fault behaviors observed for the synchronizer.

### 1.4 Testing Strategy

There are two basic capture-clocking schemes for at-speed testing of multiple clock domains:

**I Skewed load:** which is now commonly called launch-on-shift (LOS) [10].
<table>
<thead>
<tr>
<th>Category</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stuck-At Fault</td>
<td>SA0</td>
<td>stuck-at-0</td>
</tr>
<tr>
<td></td>
<td>SA1</td>
<td>stuck-at-1</td>
</tr>
<tr>
<td>Functional Faults</td>
<td>TFT1</td>
<td>Output observed one clock cycle earlier than expected</td>
</tr>
<tr>
<td></td>
<td>TFT2</td>
<td>Inverted output observed one clock cycle earlier than expected</td>
</tr>
<tr>
<td></td>
<td>TFT3</td>
<td>Output observed one clock cycle later than expected</td>
</tr>
<tr>
<td></td>
<td>TFT4</td>
<td>Inverted output observed one clock cycle later than expected</td>
</tr>
<tr>
<td></td>
<td>DO</td>
<td>Output with delay of the synchronizer</td>
</tr>
<tr>
<td></td>
<td>IDO</td>
<td>Inverted output with delay of the synchronizer</td>
</tr>
<tr>
<td>Pulse Test Output</td>
<td>PTO1</td>
<td>Synchronizer generates pulses depending on the input signal</td>
</tr>
<tr>
<td></td>
<td>PTO2</td>
<td>Synchronizer generates pulses independent of the input signal</td>
</tr>
<tr>
<td>One Time Pulse</td>
<td>OTP0</td>
<td>Synchronizer generates 1-0-1 one time pulse</td>
</tr>
<tr>
<td></td>
<td>OTP1</td>
<td>Synchronizer generates 0-1-0 one time pulse</td>
</tr>
<tr>
<td>Metastable Output</td>
<td>MO</td>
<td>The output of the synchronizer appears at unpredictable time</td>
</tr>
<tr>
<td>Internal Oscillation Fault</td>
<td>IOF</td>
<td>A signal oscillation occurs in the synchronizer by a loop, and the output of a synchronizer depends on the applied clock frequency.</td>
</tr>
<tr>
<td>Undefined Value</td>
<td>UD</td>
<td>The output voltage of the synchronizer is in the range of $V_{DD} \times 0.3$ to $V_{DD} \times 0.7$.</td>
</tr>
</tbody>
</table>
Table 1.2: Category of all Open faults

<table>
<thead>
<tr>
<th>Category</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stuck-At Fault</td>
<td>SA0</td>
<td>stuck-at-0</td>
</tr>
<tr>
<td></td>
<td>SA1</td>
<td>stuck-at-1</td>
</tr>
<tr>
<td>Delay Fault</td>
<td>DF</td>
<td>Open defects cause delay faults</td>
</tr>
<tr>
<td>Delay Fault by resistive open defect</td>
<td>DFR</td>
<td>A synchronizer with resistive open defect generates more than one clock cycle later than expected.</td>
</tr>
<tr>
<td>Pulse Fault</td>
<td>PF</td>
<td>Open defects cause the faulty synchronizer faults to generate pulses.</td>
</tr>
<tr>
<td>One Time Pulse Fault</td>
<td>OTPF</td>
<td>The faulty synchronizer shows one time pulse before its valid output is generated.</td>
</tr>
<tr>
<td>Undefined Logic Value</td>
<td>ULV</td>
<td>The output voltage of the synchronizer is in the range of $V_{DD} \times 0.3$ to $V_{DD} \times 0.7$.</td>
</tr>
</tbody>
</table>

**II Double Capture:** which was called broad-side in [11] but is now commonly called launch-on-capture (LOC).

In [12] and [13], Wang et al. proposed several capture-window schemes to test intra-clock domain and inter-clock domain delay faults and structural faults for multicycle designs. Some of them are one-hot launch-on-shift, simultaneous launch-on-shift etc, but each method has its own advantages and disadvantages. The staggered double-capture scheme as shown in Fig. 1.4(a) is applicable to test asynchronous inter-clock domains in a scan-based BIST design. In [12], it is assumed that the flip-flops of a synchronizer are scanable. So, test patterns are applied in the shift window and then two capture pulses are generated for each clock domain in the capture window. By adjusting ‘d’, this approach can provide at-speed testing of inter-clock domain delay faults. However, synchronizers are very sensitive to the timing of clock
and data, it may not be appropriate to make them scanable. Hence we have modified the staggered double capture scheme and is shown in Fig 1.4(b). This modified capture window will be used to detect the bridging and open faults in a non-scanable synchronizer.

![Diagrams](image)

(a) Staggered Double Capture

(b) Modified Double Capture

Figure 1.4: Capture Window Schemes

This work consists of detecting all the bridging and open faults in a two flip-flop synchronizer used in the implementation of handshake protocol between two mutually asynchronous cores. The thesis is organized as follows:

**Chapter 2** explains the gate level implementation of a two way handshaking protocol between two asynchronous clock domains.
Chapter 3 explains how the interface design is made testable and test methods for detecting the bridging and open faults in each synchronizer.

Chapter 4 extends the testing method when asynchronous cores are used in an SoC using an IEEE 1500 wrapper.

Chapter 5 explains the conclusion and possible future work.
INTRODUCTION

The interface design presented below implements a two way handshake protocol between two S27 benchmark circuits. When the S27 circuit in the source module is ready to send data to the destination module, we give a pulse on the DATA-READY pin to the Control Logic unit which generates a START pulse. While the START signal is being asserted, the S27 circuit in the source module will be frozen to keep the data to send until the Handshake Source Module fetches the data from the S27 circuit. De-asserting the START signal makes the S27 circuit resume for next data transition by asserting the EXECUTE signal from the Control Logic. The Handshake Source Module takes the START pulse as an input and generates a request signal (TAKE-IT-TG). This request signal is fed to the Handshake Destination Module and an acknowledge signal (GOT-IT-TG) is generated, once the data is successfully transferred between the source and the destination. Once the acknowledge signal for the previous data is received by the Handshake Source Module, a new data transition will be executed as soon as the data to send is ready. The signal GOT-IT-PL informs the Control Logic that data transition is done. Fig 2.1 shows the block diagram of
Chapter 2  Interface Design Between Asynchronous Clock Domains

the design. All the blocks are explained in detail in the following section.

![Figure 2.1: Block Diagram](image)

The source module is operated at 100 MHz and destination module is operated at 151.15 MHz.

### 2.1 Control Logic

Control logic can be divided into three logic blocks namely START-GEN, HS-DONE-GEN and EXECUTE-GEN. The START-GEN logic block shown in Fig 2.2 generates the START pulse of width one clock cycle if the signals HS-DONE and DATA-READY are high.

![Figure 2.2: START-GEN](image)

The HS-DONE-GEN logic block shown in Fig 2.3 generates the signal HS-
DONE when the data transition between the source and destination is completed. The initial state of HS-DONE is logic-1 and the START pulse from START-GEN makes it de-asserted. The signal GOT-IT-PL from the Handshake Source Module resets the HS-DONE signal to logic-1. This HS-DONE-GEN logic block is coupled with a multiplexer in test mode to make the interface design testable. See Fig 3.2.

![Diagram](image)

**Figure 2.3: HS-DONE-GEN**

The EXECUTE-GEN logic block shown in Fig 2.4 generates the EXECUTE signal to control the S27 benchmark circuit. De-asserting the EXECUTE signal makes the S27 circuit frozen and asserting EXECUTE signal resumes the S27 circuit to prepare a new data for the next transition. The RESET signal resets the EXECUTE-GEN logic block as a result of which EXECUTE signal stays high for exactly one clock cycle.

![Diagram](image)

**Figure 2.4: EXECUTE-GEN**

Fig 2.5 shows the waveform of whole control logic circuit. When we connect all
the three logic blocks i.e. START-GEN, HSDONE-GEN and EXECUTE-GEN shown in Fig 2.11, they work in the following manner.

First, all the DFF’s are reset to have initial value ‘0’. This sets the initial value of HS-DONE to ‘1’ and causes the execute signal to go high for one clock cycle. At the falling edge of the first clock pulse, EXECUTE will become ‘0’ and thus only one clock cycle is applied to the S27 benchmark circuit to generate the data to be transferred over the data bus. We now apply a pulse of width one clock cycle on the DATA-READY pin. Since HS-DONE is already ‘1’, and now DATA-READY has become ‘1’, so START signal will be generated and it will be a pulse of one clock cycle width. The same START pulse will make the HS-DONE signal to ‘0’ at the next rising edge of the clock. When the handshaking is done and the receiving domain sends the acknowledge signal, the Handshake Source Module will generate GOT-IT-PL and this will make the HS-DONE signal back to ‘1’ indicating that the handshaking is successfully completed and the next data transmission can be started.

To start the next data transmission, first we reset all the flip flops to zero and the above process will be repeated.

2.2 S27 Benchmark Circuit

Fig 2.6 shows the S27 benchmark circuit in the source side modified to send data to the other S27 benchmark circuit in the receiving side such that G10 goes to G0, G13 to G1, G16 to G2 and G17 to G3 respectively. The S27 circuit in the source is given two clock cycles to prepare new data for transition. The input to the S27 circuit is 4-bit data and output is also 4 bits.

Fig 2.7 shows the S27 benchmark circuit in the destination side. The input is 4-bit data and output is only one bit.
Figure 2.5: Waveform of Control Logic

Figure 2.6: S27 Source Circuit
Figure 2.7: S27 Destination Circuit
2.3 Handshake Source Module

Fig 2.8 shows the implementation of the source module of handshake communication. The source module consists of a rising edge detector, toggle generator, pulse generator and a synchronizer together with some additional logic for data transfer on the bus.

Fig 2.9 shows the waveform of the Handshake Source Module. As explained in the previous section, when the data to send is ready, the Control Logic will assert a START pulse. The source module takes the START pulse as an input. The rising edge detector detects the rising edge of the START pulse and generates a pulse called FETCH-PL. The FETCH-PL is then given as an input to the toggle circuit which generates the TAKE-IT-TG (request) signal for the receiving domain at the next clock pulse to indicate to destination module that the source module is sending data to destination module. Once the data is received, the receiving domain sends out the GOT-IT-TG (acknowledge) signal. This signal is asynchronous with respect to the clock in sender domain. After passing through the synchronizer, the GOT-IT-TG signal becomes synchronized with the sending clock and is fed to a pulse generator circuit which generates GOT-IT-PL signal. As explained in Section 2.1, the GOT-IT-PL signal resets the HS-DONE signal to logic ‘1’ indicating that the handshaking has been successfully done. The START pulse and FETCH-PL go hand in hand but they are not redundant. The primary use of FETCH-PL is data transfer on the bus, whereas the START pulse is used to initiate the handshaking process. The TAKE-IT-TG signal toggles each time when both the sender and receiver are communicating.
Figure 2.8: Handshake Source Module
Figure 2.9: Waveform of Handshake Source Module
2.4 Handshake Destination Module

Fig 2.10 shows the implementation of the destination module for handshake communication. The destination module consists of a pulse generator, toggle generator and a synchronizer with some additional logic gates.

Whenever the state of the TAKE-IT-TG (request) signal is changed, it generates a pulse TAKE-IT-PL (after TAKE-IT-TG is synchronized). This TAKE-IT-PL pulse is used to transfer the data on the bus onto the input of the receiving s27 benchmark circuit. TAKE-IT-PL also toggles the GOT-IT-TG (acknowledge) signal to inform the source module that the data is transferred successfully.

![Diagram of Handshake Destination Module](image)

Figure 2.10: Handshake Destination Module
Fig 2.11 and Fig 2.12 show the complete circuit after all the logic blocks on the source side and destination side are stitched together.

Figure 2.11: Source Side
Figure 2.12: Destination Side
The Interface design presented in the previous chapter is made testable by adding some additional logic and making the design scanable by replacing the original flip flops by scanable flip flops. However, the synchronizers are not scanable because they are timing sensitive.

### 3.1 Scan Design In Source Module

The length of the scan chain in source module is 15 as shown in Fig 3.1. As explained in the above sections, the flip flops in EXECUTE-GEN block are falling edge triggered. Whenever there are rising edge and falling edge flip flops in a single scan chain, the general rule is to place all the falling edge DFF’s at the beginning of the scan chain. So the scan chain starts from EXECUTE-GEN logic block.
Figure 3.1: Scan Chain In Source
Also additional logic was added to HS-DONE-GEN logic block to ensure that the START pulse is generated only because of the scan-in data but not due to the initial state of HS-DONE signal. Also a multiplexer was added to choose between the functional clock and test clock in the Test mode (not shown in this figure). Fig 3.2 shows the new HS-DONE-GEN logic block.

Also a multiplexer is added to ensure that continuous clock is applied to the s27 source circuit in the test mode. (Note that in normal mode they are fed gated clock). See Fig 3.1.

The scan clock SCK should be very slow. So we have used 200 MHz clock for SCK.

![Figure 3.2: Modified HS-DONE-GEN](image)

### 3.2 Scan Design In Destination Module

A multiplexer was added to choose between the functional clock and test clock in destination module in the Test mode (not shown in this figure). The length of the scan chain is 12 as shown in Fig 3.3 below. The three additional scanable flip flops shown inside the blue box are added to detect some of the delay faults in the synchronizer. The flip flop colored in light red is a dual edge triggered DFF which is
used to detect a particular set of delay faults. More details will be provided in the section explaining delay faults.

Figure 3.3: Scan Chain In Destination

### 3.3 Test Clock Generator

This section explains the functioning of the circuit which generates test clocks for domain-1 and domain-2. Fig 4.1 below shows the block diagram of the test clock generator. The implementation of each of the blocks is explained in the later part of this section.

The at-speed launch and capture pulses are generated by the Test Clock Generator based on the signal DOMAIN-SEL. When DOMAIN-SEL is ‘0’, we test the
synchronizer in destination module and when DOMAIN-SEL is ‘1’, we test the synchronizer in source module.

When DOMAIN-SEL is ‘0’, the Test Clock Generator circuit generates one launch pulse in domain-1 followed by three capture pulses in domain-2. Similarly, when DOMAIN-SEL is ‘1’, it generates one launch pulse in domain-2 followed by three capture pulses in domain-1.

The signal FAULT-SEL is used to select the inverted capture window which will be used to test some of the delay faults in the synchronizer.

![Diagram of Test Clock Generator](image)

**Figure 3.4: Test Clock Generator**

The Test Clock Generator consists of three components. 1-Pulse generator which generates the launch clock. The generated clock will be propagated to 3-Pulse generator through channel delay element and a Programmable Delay Generator shown in Figure 3.8. The capture clock generator circuit generates three capture pulses for testing synchronizers. First of all, all D flip-flops used in the implementation are reset asynchronously, which means that if reset is deasserted, the output of the D flip-flop immediately changes to logic 0 without synchronizing with a clock edge. Fig.3.5 shows the detailed 1-Pulse Generator logic. SE signal is latched through the two D-flipflop synchronizer by the falling edge of the functional clock depending on the DOMAIN-SEL signal. Then, the synchronizer is followed by the falling edge detector, which
generates EN-FCK. When SE is deasserted, EN-FCK whose period is one clock cycle period of functional clock is generated and is used to pass only one clock cycle of either CLOCK1 or CLOCK2.

![Implementation Of Launch Clock Generator](image)

**Figure 3.5: Implementation Of Launch Clock Generator**

Fig 3.6 shows the detailed implementation of 3-Pulse generator logic. First, DFF1 is reset by RST = 0 or SE = 1. So, before launch-clk(1-pulse) arrives at 3-Pulse generator, DFF1 is initialized to logic 0. When launch-clk arrives at DFF1, DFF1 latches logic 1 and it becomes the rising edge of the first clock cycle of capture-clk. The logic 1 is propagated through another path which consists of a delay element of ‘delta’ approximately half period of clock cycle in receiving domain and an inverter. This delay inverted signal generates the falling edge of the first clock cycle of capture-clk. This first clock cycle propagate through a delay element of a period of clock cycle in receiving domain and arrives at DFF4 which is a double-edge-triggered D flip-flop. DFF4 is reset by RST = 0, SE =1, or when two falling edges are detected. So, DFF4 is initialized to logic 0 before the first clock cycle arrives. When the rising edge of the first clock cycle arrives after delay T, the output of DFF4 becomes logic 1 which is the rising edge of the second clock cycle. The upcoming falling edge of the first clock cycle resets the output of DFF4 to logic 0 which is the falling edge of the second clock cycle. This second clock cycle makes DFF2 latch logic 1. Like above, the second clock cycle
make the third clock cycle generated. The third clock cycle results in propagating
the output of DFF2 to DFF3 and resetting DFF4. The FAULT-SEL signal is used
to choose between the original and inverted capture signal. The delays $\Delta$ and $T$ can
be programmed using the Programmable Delay Generator circuit shown in Fig3.8.

![Implementation Of Capture Clock Generator](image)

Figure 3.6: Implementation Of Capture Clock Generator

<table>
<thead>
<tr>
<th>Current Simulation Time: 305 ns</th>
<th>0 ns</th>
<th>25 ns</th>
<th>50 ns</th>
<th>75 ns</th>
<th>100 ns</th>
<th>125 ns</th>
<th>150 ns</th>
<th>175 ns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>launch_clk</code></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>p0</code></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>p1</code></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>p2</code></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>capture_clk</code></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.7: Waveform Of Capture Clock Generator
Fig 3.7 shows the waveform of the Capture Clock Generator Circuit with FAULT-SEL as ‘0’. The points A and B have been marked to show the working of the circuit.

### 3.4 Programmable Delay Generator

The schematic of the Programmable Delay Generator circuit [14] is shown in Fig. 3.8. The circuit is triggered by a rising transition on the input signal. This rising transition will appear at the output after a fixed delay. The circuit consists of two delay lines, one using coarse delay buffers and the other using fine delay buffers. The required delay can be programmed using a series of scannable flip flops that need to be set to a one-hot code during scan mode. As long as input is 0, the output will not be affected due to the changing values in registers during the scan mode, and the output signal will remain stable at 0. When input is changed to 1, the NMOS transistor corresponding to the flip flop which has the value 1 stored, gets a rising signal after a certain delay through the delay line. The node Y drops to 0, which acts as the trigger to the fine delay controller. The fine control operates exactly the same as the coarse delay part, so that the output signal has a rising transition after the programmed delay.

The Programmable Delay circuit is used to generate the $T/2$ and $T$ delays in capture clock generator circuit. It is also used to generate different values of $t_{dist}$ [8].

### 3.5 Delay Equations for Test Clock Generator

This section explains the equations for ‘d’ in Fig 3.11 and ‘channel propagation delay’ in Fig 4.1. ‘d’ is the distance between the rising edge of the launch pulse in
Figure 3.8: Programmable Delay Generator

domain-1 to the rising edge of the first capture pulse in domain-2.

In Fig 3.9, t1 is the delay from Test Clock Generator to the handshake module which generates the request signal. t2 is the delay from Test Clock Generator to the receiving handshake module. The delay on the request line is α.

From Fig 3.10, we can derive the following equation,

\[ d = \alpha + t_{\text{dist}} + t_1 - t_2 \]  \hspace{1cm} (3.1)

where \( t_{\text{dist}} \) is the distance between the rising edge of the request signal and the rising edge of the first capture pulse at the receiving domain.

If the capture window is as in Fig 3.17, the above equation will be

\[ d = \beta + t_{\text{dist}} + t_2 - t_1 \]  \hspace{1cm} (3.2)

where \( \beta \) is the delay on the acknowledge line.

In Fig 3.6, ‘x’ is the delay of the AND gate, ‘y’ is the delay of the OR gate and ‘z’ is the delay of the MUX.
Chapter 3  Testable Interface Design

Figure 3.9: Block diagram for Delay

Figure 3.10: Waveform to calculate ‘d’

From Fig 3.11, we can derive the following equation,

\[
\text{Programmable Delay} + \text{Channel Propagation Delay} = d - (x + y + z) \quad (3.3)
\]
Equation 3.3 can be rewritten as,

\[
\text{Programmable Delay} = d - (x + y + z) - \text{Channel Propagation Delay} \quad (3.4)
\]

The fault behavior of the synchronizer depends on the value of \( t_{dist} \) as shown in Tables A.2, A.3 and A.4. So we need to generate request signal at different times in order to activate a particular fault.

Equations 3.1 and 3.3 can be used to generate request signal with different \( t_{dist} \) values. First we calculate the value of ‘\( d \)’ from Equation 3.1 depending upon the \( t_{dist} \) value that is required. Then we calculate the value of Programmable Delay from Equation 3.4. The Programmable Delay Generator is then used to generate this delay.

Also from Fig 3.6, we can derive the following equation,

\[
\Delta + x + y = T/2 \quad (3.5)
\]

where \( T \) is the clock period of the receiving clock domain.

Fig 3.12 shows the Clock Selector Logic which selects the appropriate launch and capture pulses based on the signal DOMAIN-SEL.

Fig 3.13 below shows the simulated waveform when DOMAIN-SEL is ‘0’.

Fig 3.14 below shows the simulated waveform when DOMAIN-SEL is ‘1’.

Fig 3.15 below shows the block diagram of the entire design with the Test Clock
Chapter 3  Testable Interface Design

Figure 3.12: Clock Selector Logic

Figure 3.13: Domain-Sel = ‘0’

Figure 3.14: Domain-Sel = ‘1’
generator. Multiplexers are used to choose between the Functional clock and Test clock for both the domains.

Figure 3.15: Complete Design
3.6 Structural Fault Testing Of Synchronizers

Test Scheme

The scan chain in source side is used to create $0 \rightarrow 1$ and $1 \rightarrow 0$ transition on TAKE-IT-TG signal to test for SA-0 and SA-1 fault respectively in the synchronizer of domain-2. Similarly the scan chain in destination side is used to create $0 \rightarrow 1$ and $1 \rightarrow 0$ transition on GOT-IT-TG signal to test for stuck-at faults in synchronizer in domain-1.

3.6.1 Capture Window For Domain-2

After we scan in test data in SHIFT MODE to create a transition on TAKE-IT-TG signal, we make SE low. This is CAPTURE MODE. We apply one at-speed capture pulse from Test Clock Generator in domain-1 followed by three at-speed capture pulses in domain-2, two pulses to synchronize the request signal and third pulse to capture the output of the synchronizer into the scan chain in domain-2. The capture window is shown in Fig 3.16 below.

![Capture Window For Domain-2](image)

Figure 3.16: Capture Window For Domain-2
3.6.2 Stuck-At ‘0’ Fault In Domain-2

To test for SA-0 fault in the synchronizer in destination module, we need a $0 \rightarrow 1$ transition on TAKE-IT-TG signal. So we scan in test data “0000100000000000” for 15 clock cycles of Scan clock (SCK) when SE is high. Simultaneously we scan in all 1’s in domain-2. At the last clock cycle in SHIFT mode, START pulse will be generated. When SE goes low (CAPTURE MODE), we apply one at-speed capture pulse in domain-1 to generate a $0 \rightarrow 1$ transition on TAKE-IT-TG. Then we apply three capture pulses in domain-2, two pulses to synchronize TAKE-IT-TG signal and third pulse to capture the response into the scan chain. Finally we make SE high and apply SCK to shift out the response of the synchronizer. After eleven shift cycles we observe the output of the synchronizer in domain-2 at scan out pin SO2. The output at the scan out pin is valid only after eleven clock cycles. All the scan out data before this must be ignored. In case of SA-0 fault, we observe ‘0’ after eleven SCK cycles at SO2 pin.

3.6.3 Stuck-At ‘1’ Fault In Domain-2

To test for SA1 fault in domain-2, we need a $1 \rightarrow 0$ transition on TAKE-IT-TG signal. For this we scan in all 1’s in domain-1 for 15 SCK cycles. Simultaneously we scan in all 0’s in domain-2. So, the initial state of TAKE-IT-TG is logic ‘1’. When we make SE low and apply the at-speed capture pulse, TAKE-IT-TG will toggle making a $1 \rightarrow 0$ transition. This will test for SA1 fault in domain-2. We then apply three at-speed capture pulses in domain-2 and make SE high. We observe logic ‘1’ after eleven clock SCK cycles at SO2 pin.
3.6.4 Capture Window for Domain-1

After we scan in test data in SHIFT MODE to create a transition on GOT-IT-TG signal, we make SE low to enter into CAPTURE MODE. We apply one at-speed capture pulse from Test Clock Generator in domain-2 followed by three at-speed capture pulses in domain-1, two pulses to synchronize the acknowledge signal and third pulse to capture the output of the synchronizer into the scan chain in domain-1. The capture window is shown in Fig 3.17 below.

![Capture Window Diagram](image)

Figure 3.17: Capture Window For Domain-1

3.6.5 Stuck-At ‘0’ Fault In Domain-1

To test for SA-0 fault in the synchronizer in source module, we need a $0 \rightarrow 1$ transition on GOT-IT-TG signal. We have to make sure that the output line of the synchronizer in domain-2 has logic’0’ to create a transition on GOT-IT-TG. Hence we use 17 clock cycles during SHIFT mode instead of 15. The two extra clock cycles are used to initialize the output line of the synchronizer to logic’0’. We scan in test data “0011111111111111” for 17 SCK cycles during SHIFT mode in domain-1 “0000000000000000” for 17 clock cycles of Scan clock (SCK) when SE is high in domain-2. When SE goes low (CAPTURE MODE), we apply one at-speed capture
pulse in domain-2 to generate a $0 \rightarrow 1$ transition on GOT-IT-TG. Then we apply three capture pulses in domain-1, two pulses to synchronize GOT-IT-TG signal and third pulse to capture the response of the synchronizer into the scan chain. Finally we make SE high and apply SCK to shift out the response of the synchronizer. After one shift cycle we observe the output of the synchronizer in domain-1 at scan out pin SO1. In case of SA-0 fault, we observe ‘0’ after one SCK cycles at SO1 pin.

3.6.6 Stuck-At ‘1’ Fault In Domain-1

To test for SA1 fault in domain-1, we need a $1 \rightarrow 0$ transition on GOT-IT-TG signal. For this we scan in “0000000000000011” for 17 SCK cycles in domain-2. Simultaneously we scan in all 0’s for 17 SCK cycles in domain-1. So, the initial state of GOT-IT-TG is logic ‘1’. When we make SE low and apply one at-speed capture pulse, GOT-IT-TG will toggle making a $1 \rightarrow 0$ transition. This will test for SA1 fault in domain-1. We then apply three at-speed capture pulses in domain-1 and make SE high. We observe logic ‘1’ after one clock SCK cycle at SO1 pin.

3.7 Delay Fault Testing Of Synchronizer

The different types of delay faults in the synchronizer are explained below.

**Late Transition Fault:** The output of the synchronizer appears one clock cycle later than expected. There are two types of such faults:

- Normal Late Transition
- Inverted Late Transition

**Early Transition Fault:** The output of the synchronizer appears one clock cycle earlier than expected. There are two types of such faults:
Chapter 3  Testable Interface Design

- Normal Early Transition
- Inverted Early Transition

**Inverted Output Fault:** In this fault, the output of the synchronizer is inverted with respect to the input logic but appears at the right time.

**Input Dependent Pulses:** The synchronizer generates pulses at the output. The frequency of these pulses depend on the input logic value.

**Input Independent Pulses:** The synchronizer generates pulses at the output whose frequency is independent of the input logic.

**Internal Oscillation Fault:** The synchronizer generates pulses at the output if the frequency of the clock applied to it is very high. For smaller frequencies, the synchronizer behaves as stuck at ‘1’.

**One Time Pulse:** The synchronizer generates only one pulse, either 0-1-0 or 1-0-1 depending on the resistance of the bridging.

### 3.7.1 Detecting Late Transition Fault

The same capture window as in stuck-at fault detection is used to detect this fault. In this fault, the output of the synchronizer appears one clock cycle later than expected i.e. after three clock cycles to the synchronizer. We give a $0 \rightarrow 1$ transition on TAKE-IT-TG signal with the same test pattern mentioned for the stuck-at fault testing and we scan in all 0’s in the domain-2. Since the capture window has three at-speed clock pulses, the output of the synchronizer cannot be latched into the scan chain in domain 2. Hence if we do not see a $0 \rightarrow 1$ transition after eleven clock cycles in the shift mode, we can say that there is a Late Transition Fault in the synchronizer. The same explanation can be given in case of Inverted Late Transition Fault.
3.7.2 Detecting Early Transition Fault

In this fault, the output of the synchronizer appears one clock cycle earlier than expected i.e. the synchronizer will take only one clock cycle to give the output. According to [8], this fault can be activated only if the input is applied to the synchronizer when clock is high. So the inverted capture window is used to trigger this fault. The inverted capture window shown in Fig 3.18 can be selected by making the FAULT-SEL signal to logic ‘1’.

![Figure 3.18: Inverted Capture Window For Domain-2](image)

The three additional scan flip flops shown in Fig 3.3 are used to detect this fault. The control signal PTO is ‘0’ while detecting this fault. We give a $0 \rightarrow 1$ transition on TAKE-IT-TG signal by scanning in the appropriate test pattern during the shift mode. At the same time we scan in all 0’s in domain-2. In the capture mode, the at-speed launch pulse on TCK1 makes a $0 \rightarrow 1$ transition on TAKE-IT-TG. After three rising edges in the capture mode in TCK2, the output of the synchronizer will be latched into the first of the three additional scan flip flops. When SE goes high again in the shift out mode, we can observe the output after two SCK cycles at SO2 pin. Thus Normal Early Transition Fault can be detected.
To detect Inverted early Transition Fault, we scan in test pattern to make a $0 \rightarrow 1$ transition on TAKE-IT-TG and we scan in "$00011000000001$" in domain-2 so that the first scan flip flop and the three additional scan flip flops hold logic ‘1’ during the capture mode. At the first rising edge in TCK2 in capture mode, the output of the synchronizer will make a $1 \rightarrow 0$ transition. By the end of the capture mode, logic ‘0’ will be latched in the first of the three additional scan flip flops. Then in the shift out mode, we can observe logic ‘0’ at SO2 after two SCK cycles therefore detecting Inverted Early Transition Fault.

### 3.7.3 Detecting Inverted Output Fault

In this presence of this fault, the output of the synchronizer is inverted with respect to its input transition but appears at the right time. This fault will not affect the functioning of the handshake protocol as the handshake mechanism works based only on transition on the request and acknowledge signals. It does not matter if the transition is a $0 \rightarrow 1$ or a $1 \rightarrow 0$ transition. However, this fault can be easily detected with the same capture window. We make a $0 \rightarrow 1$ transition on TAKE-IT-TG signal and scan in all 1’s in domain-2. The synchronizer will give a $1 \rightarrow 0$ transition after two capture pulses and at the third capture pulse, logic ‘0’ will be latched into the scan chain. We can observe the output after eleven clock cycles in the shift out mode.

### 3.7.4 Detecting Input Dependent Pulses

In this fault, the synchronizer will generate pulses at its output depending on the input logic applied. If input is ‘0’, the synchronizer will generate pulses of frequency $f_{clk}/2$ where $f_{clk}$ is the frequency of the clock applied to the synchronizer. If input is ‘1’, the synchronizer will generate pulses of frequency $f_{clk}$. 
Figure 3.19 below shows the output of the synchronizer when a $0 \rightarrow 1$ input transition is applied.

![Diagram](image)

Figure 3.19: Pulse Fault whose frequency is $f_{clk}$

The control signal PTO is used to detect the faults which generate pulses at the output of the synchronizer such as Input Dependent pulses, Input Independent Pulses and Internal Oscillation Fault. PTO is set to logic ‘1’ in these cases. In Fig 3.3, we can see that these pulses will be applied to the clock of the additional scan flip flops only in the CAPTURE mode i.e. after SE goes low. This will ensure that in the SHIFT mode, we can scan in either 0’s or 1’s in the additional flip flops as required.

To detect this fault, initially we scan in all 0’s in domain-2. When SE goes low, the output of the synchronizer is applied to the clock of the three additional scan flip flops. Also the first flip flop always gets logic ‘0’ as input. So when the pulses generated by the synchronizer are applied to the clock of the additional flip flops in the CAPTURE mode, logic ‘1’ will be latched into these flip flops. So if we observe logic ‘1’ at the SO2 pin, we can say that the synchronizer is faulty. The SE must go high after we allow enough time for the logic ‘1’ to appear at SO2 pin. This period depends on the frequency of the pulses that are generated.
3.7.5 Detecting Input Independent Pulses

In this fault, the synchronizer will generate pulses of frequency \( f_{clk}/2 \) at its output irrespective of the input applied. The same method used to detect Input Dependent Pulses can be applied to detect this fault.

3.7.6 Detecting Internal Oscillation Fault

The Internal Oscillation Fault depends on the frequency of the clock applied to the synchronizer. When very high frequency clock of the order of \( f_{clk} = 500 \) MHz is applied, the synchronizer will generate pulses of frequency \( f_{clk}/2 \) regardless of the input logic value. However, if lower clock frequencies are applied to the synchronizer, it will behave as stuck at ‘1’.

If we assume that the clock frequency of domain-2 is very high, the synchronizer will generate pulses at its output and this fault can be detected using the same method as in Input Dependent Pulses.

If we assume that the clock frequency of domain-2 is low, the synchronizer will behave as stuck at ‘1’ and we already know how to detect stuck at faults. Thus Internal Oscillation Fault can be detected.

3.7.7 Detecting One Time Pulse Fault

As mentioned earlier, there are two types of One Time Pulse faults, OTP0 and OTP1. To detect a single pulse, we have to detect both the rising edge and also the falling edge of the pulse. Hence, we use a dual edge triggered DFF shown in Fig 3.3 to detect these faults. This dual edge triggered DFF must be reset to ‘0’ initially. So the initial value at the pin OTP will be ‘0’.
Detecting OTP0 Fault

This fault generates a single 1-0-1 pulse at the output of the synchronizer when a 0 → 1 transition is applied in the presence of a bridging fault with high resistance. Figure 3.20 below shows the output of the synchronizer in the presence of such fault.

In the SHIFT mode we scan in test data to make a 0 → 1 transition on TAKE-IT-TG. In the CAPTURE mode, after two capture clock cycles, a 1-0-1 pulse will be generated at the output of the synchronizer. This pulse will be applied to the clock of the dual edge triggered DFF. At the falling edge of the 1-0-1 pulse, the output of the dual edge triggered DFF (OTP) will become ‘1’ and at the rising edge of the 1-0-1 pulse, the output (OTP) becomes ‘0’. Hence if we observe a 0-1-0 pulse at the OTP pin, we can say that there is a OTP0 fault in the synchronizer.

![Diagram](image)

Figure 3.20: OTP0 by bridging defect

Detecting OTP1 Fault

This fault generates a single 0-1-0 pulse at the output of the synchronizer when a 1 → 0 transition is applied in the presence of a bridging fault with high resistance. Figure 3.21 below shows the output of the synchronizer in the presence of such fault.

In the SHIFT mode we scan in test data to make a 1 → 0 transition on TAKE-IT-TG. In the CAPTURE mode, after two capture clock cycles, a 0-1-0 pulse will be
generated at the output of the synchronizer. This pulse will be applied to the clock of the dual edge triggered DFF. At the rising edge of the 0-1-0 pulse, the output of the dual edge triggered DFF (OTP) will become ‘1’ and at the falling edge of the 0-1-0 pulse, the output (OTP) becomes ‘0’. Hence if we observe a 0-1-0 pulse at the OTP pin, we can say that there is a OTP fault in the synchronizer.

![Figure 3.21: OTP1 by bridging defect](image)

3.7.8 Metastable Output

A bridging defect can drive a D flip-flop of a synchronizer into metastable state regardless of the input arrival time. This kind of bridging defect causes the synchronizer to give the output one clock cycle early. So when we apply a $0 \rightarrow 1$ transition at the input of the synchronizer, we expect the output to show a rising transition one clock cycle earlier than expected. This means that one of the DFF’s of the synchronizer is being bypassed. The purpose of using a synchronizer is to give enough time to an asynchronous signal coming from another clock domain to synchronize with the clock of the receiving clock domain. In other words, a synchronizer is used to reduce the probability of occurrence of metastability. So when one of the DFF’s is bypassed, the probability of occurrence of metastability automatically goes high. If metastability occurs, the signal will settle to the correct logic value after some delay or at the next
clock cycle. If the output of the synchronizer settles down at the next clock cycle, then it behaves as a fault free synchronizer. So metastability due to the bridging defect can cause the synchronizer to behave sometimes as early transition fault or even fault free. Since metastability is very probabilistic, it cannot be detected.

3.7.9 Undefined Output Logic

We use a voltage window detector shown in Fig 3.22 to detect the undefined output of the synchronizer. Note that A and B are operational amplifiers (op-amps), and we have $V_{ref1} = \frac{R_3}{R_1+R_2+R_3}$ and $V_{ref2} = \frac{R_2+R_3}{R_1+R_2+R_3}$. $V_{in}$ will be the output of the synchronizer. The output $V_{out}$ is HIGH when $V_{ref1} < V_{in} < V_{ref2}$ and is LOW otherwise. As a result, by configuring R1, R2 and R3 properly, this circuit can detect the undefined state of the synchronizer. We can set $V_{ref1}$ to $0.3V_{DD}$ and $V_{ref2}$ to $0.7V_{DD}$.

Figure 3.22: Voltage Window Detector
3.8 Detecting Open Faults In A Synchronizer

Table 3.1 shows the different fault behaviors of the synchronizer in the presence of open faults. Though the terminology for One time Pulse fault and Pulse fault is the same in bridging faults and open faults, their behaviors are different.

Table 3.1: Category of all open faults

<table>
<thead>
<tr>
<th>Category</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stuck-At Fault</td>
<td>SA0</td>
<td>stuck-at-0</td>
</tr>
<tr>
<td></td>
<td>SA1</td>
<td>stuck-at-1</td>
</tr>
<tr>
<td>Delay Fault</td>
<td>DF</td>
<td>Open defects cause delay faults</td>
</tr>
<tr>
<td>Pulse Fault</td>
<td>POF</td>
<td>Open defects cause the synchronizer to generate pulses at its output.</td>
</tr>
<tr>
<td>One Time Pulse Fault</td>
<td>OTPF</td>
<td>The faulty synchronizer shows an one-time pulse before its valid output is generated.</td>
</tr>
<tr>
<td>Undefined Value</td>
<td>UD</td>
<td>The output voltage of the synchronizer is in the range of $V_{DD} \times 0.3$ to $V_{DD} \times 0.7$.</td>
</tr>
<tr>
<td>Delay Fault with</td>
<td>DFR</td>
<td>Synchronizer with resistive open defect generates its output more than one clock cycle later than expected.</td>
</tr>
<tr>
<td>resistive open defect</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.8.1 Detecting Stuck-at Fault (SA)

The stuck-at faults in case of open defects are detected in the same way as the stuck-at faults in case of bridging defects.
3.8.2 Detecting Pulse fault

The Pulse fault with open defect has a different behavior when compared to pulse fault with bridging defect. The difference can be seen when we look at the output of the synchronizer in the presence of these faults. Fig 3.19 shows the output of the synchronizer with bridging defect. Fig 3.23 below shows the output of the synchronizer when a $0 \rightarrow 1$ input transition is applied.

The Pulse fault with open defect is explained in detail in [9]. Even though the output waveform of the synchronizer (during pulse fault) is different in the presence of a bridging fault and open fault, the same method can be used to detect it.

![Figure 3.23: Pulse Fault by open defect whose frequency is $f_{clk}$](image)

3.8.3 Detecting One Time Pulse fault (OTPF)

Similar to Pulse fault, the One time pulse fault also has a different behavior in the presence of bridging and open defects. Figures 3.20 and 3.21 show the output of the synchronizer in case of bridging defects. Fig 3.24 shows the output of the synchronizer with OTPF in case of open defect. As seen in the Fig 3.24, this fault generates a single pulse before generating the correct output. So to detect this fault, we need to detect the rising and falling edge of the pulse. So we use the dual-edge
triggered DFF shwn in Fig 3.3. This dual edge triggered DFF must be reset to ‘0’ initially. So the initial value at the pin OTP will be ‘0’.

In the SHIFT mode we scan in test data to make a $0 \rightarrow 1$ transition on TAKE-IT-TG. The faulty synchronizer generates a pulse which is applied to the clock of the dual edge triggered DFF. At the rising edge of the 0-1-0 pulse, the output of the dual edge triggered DFF (OTP) will become ‘1’ and at the falling edge of the 0-1-0 pulse, the output (OTP) becomes ‘0’. Later when the synchronizer generates the correct output, the OTP pin again becomes ‘1’. Hence if we observe a 0-1-0 pulse at the OTP pin followed by $0 \rightarrow 1$ transition, we can say that there is a One Time Pulse fault in the synchronizer.

![Figure 3.24: One Time Pulse Fault by open defect](image)

The remaining open faults which include delay fault (DF), delay fault with resistive open defect (DFR) and the undefined logic value can be detected the same way as the corresponding bridging faults were detected.
This chapter explains how the sending and receiving domains together with the handshaking protocol have been wrapped with IEEE 1500 standard compliant wrapper to incorporate the cores in an SOC. The details of the wrapper design will be explained in the next sections. Turbo1500 provided by Syntest Technologies is used to generate the wrapper.

Figure 4.1 shows the top level representation of the sending and receiving domains without wrapper.

4.1 The IEEE 1500 Wrapper Architecture

A high level understanding of the 1500 wrapper architecture requires understanding of the functions of a wrapper. A wrapper is DFT logic inserted on a core to provide an interface to the SOC level TAM after the core is integrated into an SOC. A wrapper is also an isolation mechanism, not only for the embedded core, but also for logic surrounding the core. Lastly, the variety in core test needs requires support of the application of different types of test data through the wrapper, as well as a control mechanism that directs the type and sequence of tests applied to the embedded core. This translates into a variety of configurations or test modes in which
Figure 4.1: Top level representation of the cores

the wrapped core must be placed in order for the test corresponding to these modes to be applied. The various wrapper functions are supported through 1500 wrapper hardware components that enable embedded core testing. As shown in Figure 4.2, these wrapper components are:

- The wrapper interface port which provides an interface between the wrapper and SOC level TAMs. This interface has the flexibility of supporting various test bandwidth requirements.

- The Wrapper Instruction Register (WIR), which configures the wrapper into test modes and initiates all test activities inside the wrapped core.
- The Wrapper Boundary Register (WBR), which serves as an isolation mechanism with a data serialization and de-serialization scan chain. Various types of test data can be applied to the embedded core terminals through the WBR.

- The Wrapper Bypass register (WBY), which provides a short path through a wrapped core. The WBY is utilized when data meant for logic outside the core needs to traverse the core in as few clock cycles as possible so as to not incur unnecessary test time increase.

![Figure 4.2: Architecture of IEEE 1500 wrapped core](image)

### 4.2 IEEE 1500 Standard Instructions

The 1500-defined instruction set is composed of:

- **Serial instructions** - where the data uses only a serial path (WSI to WSO) and the wrapper serial controls (WSC).
Chapter 4  Wrapper Design For The Cores

- **Parallel instructions** - where the data may use a Test Access Mechanism (TAM) of 0 to n bits width in conjunction with user-defined controls.

- **Hybrid instructions** - where a combination of the terminals for serial and parallel instructions can be utilized.

There are three mandatory instructions in the 1500 standard. These three mandatory instructions cover the basic needs of a core that will be placed in an SOC. A functional mode, an external test mode and an internal test mode. A core can have as many instructions as required to have a complete test, but must always have the three mandatory instructions included in that set. Table 4.1 lists the three mandatory instructions.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Mandate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS_BYPASS</td>
<td>Required</td>
<td>Allows functional mode to occur and puts the wrapper into bypass mode.</td>
</tr>
<tr>
<td>WS_EXTEST</td>
<td>Required</td>
<td>Allows external test using a single chain configuration in the WBR.</td>
</tr>
<tr>
<td>Wx_INTEST</td>
<td>Required</td>
<td>Allows freedom to drive the internal testing needed for a core. Encompasses all of the INTEST instructions described in the 1500 standard.</td>
</tr>
</tbody>
</table>

Table 4.1: IEEE 1500 Mandatory Instruction List

All the instructions that we have defined to test the synchronizer will be explained in detail in the coming sections.
4.3 Wrapper Boundary Register (WBR)

The WBR enables the separation of core internal testing from external interconnect or logic testing. This wrapper register provides an isolation mechanism that allows test stimuli to propagate from wrapper test inputs to core inputs, and test response to propagate from core outputs to wrapper test outputs. The presence of the WBR also enables testing of logic external to a wrapped core, as stimuli can be launched from the WBR of a given wrapper, traverse logic external to the wrapper, and be captured into the WBR of another wrapped core. Another role of the WBR is to shield embedded cores when SOC level activity is deemed harmful to the wrapped core or to data inside the wrapped core. Conversely, the WBR can provide protection to logic outside the wrapped core from data coming out of this core.

Syntest Turbo 1500 supports four types of wrapper cells compliant with 1500 standard. The wrapper cell used in our design is called WC_SD1_CII and is shown in the Fig 4.3 below.

![Figure 4.3: WC_SD1_CII](image.png)

4.3.1 Parallel configuration of the WBR

Parallel configuration of the WBR is enabled by the WIR in response to a parallel or hybrid instruction. During these instructions, WBR segments can still use WRCK
but are otherwise fully under control of WPC terminals. All parallel instructions are also expected to configure the WBR so that it uses data from WPI and WPO exclusively. Figure 4.4 shows an example wrapper with a WBR chain that is configurable. The configuration is done by two sets of multiplexers.

The first set is controlled by the Extest enable signal. These multiplexers are used to concatenate internal scan chain segments with wrapper chain segments. The IF (Inward facing) mode test requires access to core registers and wrapper registers. The OF (Outward facing) mode test requires access to wrapper registers. For this reason, the internal scan chain elements are bypassed when the Extest enable is asserted.

The second set of multiplexers is controlled by the WPP enable signal. These multiplexers are used to concatenate the WBR chain or allow it to be divided into multiple segments.

Table 4.2 shows instructions and accompanying control signal values for the wrapper that is shown in Figure 4.4. During the mandatory WS_EXTEST instruction, all WBR cells are connected in a single chain between WSI-WSO for serial access. The wrapper also supports the WP_EXTEST instruction which makes use of a WPP that consists of two scan input terminals (WPI) and two scan output terminals (WPO). During the WP_EXTEST instruction, all WBR cells are part of a wrapper chain segment that can be accessed either via WPI[0]-WPO[0] or via WPI[1]-WPO[1] terminals. Core registers are bypassed during this instruction. These terminals also provide high data bandwidth in IF mode scan tests in order to limit test execution time. For this purpose, during WP_INTEST, all WBR cells and internal scan chains are part of chain segments that make use of the full WPP bandwidth in a manner similar to the WP_EXTEST configuration.
Figure 4.4: Example parallel configuration of core chains and WBR

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Extest enable</th>
<th>WPP enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS__EXTEST</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>WP__EXTEST</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>WP__INTEST</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.2: Control values for wrapper configuration in parallel mode
4.4 Wrapper Bypass Register (WBY)

The Wrapper Bypass Register (WBY) is utilized to bypass other Wrapper Data Registers (WDRs) and is a required element of the 1500 wrapper. The WBY is a mandatory dedicated test register and can be accessed only through the WSP signals. This register must be clocked by WRCK.

In an SOC, the WBRs of multiple cores can be concatenated. This may be useful in order to implement a minimal number of test ports. Rather than one or more scan inputs and outputs going to and coming from each WBR to SOC pins, a single scan in (WSI) and scan out (WSO) can be used if all of the WBRs are concatenated. However, this could create a very long scan chain. Any WBR that is not required during a specific test can be bypassed with the WBY so that the concatenated wrapper chain becomes shorter, thus reducing test time.

Turbo 1500 simply uses a flip flop as Bypass register.

4.5 Wrapper Instruction Register (WIR)

The 1500 wrapper contains a standard mechanism for handling test control. This mechanism is called the Wrapper Instruction Register (WIR) and is primarily meant to configure the WBR. However, it also controls the mode of the core embedded within the 1500 wrapper. Wrapper instructions are serially loaded through the standard WSP into the WIR.

Turbo 1500 implements a 8-bit shift register to hold the instruction opcode. The basic structure of the WIR is shown in Figure 4.5. This consists of a shift register and an update register with instruction decoding logic.

The WDR control signals and the core’s active test mode are driven by the update register content. The update register is typically built of memory elements
that are asynchronously reset when the active low WRSTN signal is asserted. As a result of this assertion, the wrapper goes into a disabled state which enables functional operation of the embedded core. This means that when WRSTN signal is made low, WS_BYPASS instruction is pushed into the WIR. The shift register allows the loading of a new instruction without disturbing the active test mode of the wrapper.

Table 4.3 shows the instructions defined and their corresponding opcodes. The opcode for WS_BYPASS has to be all 0’s.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS_BYPASS</td>
<td>00000000</td>
</tr>
<tr>
<td>WS_EXTEST</td>
<td>00000001</td>
</tr>
<tr>
<td>WP_EXTEST</td>
<td>00000101</td>
</tr>
<tr>
<td>WS_INTEST_SCAN</td>
<td>00000010</td>
</tr>
<tr>
<td>WP_INTEST_SCAN</td>
<td>00000011</td>
</tr>
</tbody>
</table>

Table 4.3: Instruction and Opcodes
4.5.1 WIR Operation

Operation of the WIR is controlled by the WSC terminals of the wrapper. While loading instructions into the wrapper, a standard protocol that activates WIR operations, one at a time and in the correct order, needs to be followed. The protocol is as follows:

1. Reset is activated i.e. WRSTN is made low, the wrapper is disabled and the core is put into functional mode.

2. Reset is de-activated, the wrapper is enabled(WRSTN is made high).

3. If SelectWIR is not enabled, it must be enabled next.

4. Assert ShiftWR to enable the shift of instruction opcodes into the WIR shift register. Instruction opcode data is loaded from WSI to WSO on the rising edge of WRCK.

5. Once the instruction opcode is loaded, the shift operation is disabled by releasing (setting to 0) the ShiftWR signal. The instruction opcode is stored in the shift register and does not influence the current wrapper mode until it is made active by the WIR update operation. The shift register holds its content when the ShiftWR is de-asserted.

6. Assert UpdateWR. The instruction in the shift register becomes active at the rising edge of WRCK.

7. De-assert SelectWIR to enable WDR operation.

The WIR circuitry retains its current output if the WRCK clock signal is stopped. The WIR output state can only be changed by a reset or an update operation.
4.6 Using Turbo1500 to generate the wrapper

Turbo1500 creates IEEE1500 standard test interface for users to test IP cores to improve core testing when the core is embedded in the system chip. The following are the features of Turbo1500.

- The capability to create synthesizable Verilog RTL codes.
- The capability to support instructions suggested by IEEE CTAG (Core Test Access Group).
- The capability to create CTL (Core Test Language) file.
- The capability to support user-defined instructions.
- The capability to create Verilog test bench used to verify IEEE1500 implementation.
- The capability to create test patterns to verify correctness of wrapper chain connectivity.
- The capability to support multiple wrapper chains (including scan chains).

Turbo1500 tool suite consists of two programs ctagsyn and ctag2ctl. ctagsyn is used for IEEE 1500 synthesis that:

- Inserts wrapper cells.
- Connects the functional ports of wrapper cells to the corresponding ports of the core.
- Concatenates the wrapper cells to create a wrapper chain.
- Inserts the Instruction controller into the circuit.
• Connects the control signals to the corresponding pins of wrapper cells.

**ctag2ctl** is Core Test Language file generator and a verification pattern generator that:

• Generates test patterns to verify the operation of instruction decoder.

• Verifies the correctness of wrapper chain connection per IEEE 1500 standard.

• Generates CTL description file for the cores wrapped by wrapper cells.

### 4.6.1 Embedded-Core-Interface - Test Control File (.cfg)

The **.cfg** file controls the actions and specifies constraints. The **.cfg** file provides the following information:

• Port names in the top module for IEEE1500 WIP. (%GLOBAL\_SIGNAL)

• Wrapper cell information used for input or output ports. (%WRAPPER\_CELL)

• Wrapper chain information. (%WRAPPER\_CHAIN\_FILE)

• Port list and skip ports that users do not wish to connect to a wrapper cell. (%SKIP\_SIGNAL\_NAME)

• Instruction information. (%INSTRUCTION\_TABLE)

All the fields mentioned above are explained in detail below.

**%GLOBAL\_SIGNAL Field**

Users must define port name for IEEE1500 WIP. In order to provide parallel data’s interface communicating with TAM bus, two optional fields are needed. One field (%PARALLEL\_IN) defines a input port (that is “WPI[0]” of Figure 4.4) and another
%GLOBAL_SIGNAL  
{  
%WRCK <port name>;  
%WRSTN <port name>;  
%SHIFTWR <port name>;  
%CAPTUREWR <port name>;  
%UPDATEWR <port name>;  
%SELECTWIR <port name>;  
%WP_SI <port name>;  
%WP_SO <port name>;  
%PARALLEL_IN <port name>;/optional field  
%PARALLEL_OUT <port name>;/optional field  
}  

(%PARALLEL_OUT) defines a port (that is “WPO[0]” of Figure 1-1) that outputs data to TAM bus. These two fields must be specified only in the case of multiple wrapper chains.

%WRAPPER_CELL Field

This field is used to request IEEE1500 synthesizer to select a particular module as IEEE1500 wrapper cell. All <module name> specified in this field also must be found in the .lib file. This field only specifies module name and each port’s usage of this module is defined in the .lib file.
Chapter 4 Wrapper Design For The Cores

\%WRAPPER_CELL

\%
\%INPUT_WRAPPER
\%OUTPUT_WRAPPER
\%INPUT_SAFE0_WRAPPER
\%OUTPUT_SAFE0_WRAPPER
\%INPUT_SAFE1_WRAPPER
\%OUTPUT_SAFE1_WRAPPER

\%

\%WRAPPER_CHAIN_FILE Field

This field specifies directory path of the .wpso file. The details of .wpso file are explained in coming sections.

\%WRAPPER_CHAIN_FILE {\%FILE_NAME <file name>; }

\%SKIP_SIGNAL_NAME Field (Optional)

The ports specified in this field will be skipped by IEEE1500 wrapper cell. For scan chain’s test mode, scan mode and clocks, the SynTest IEEE 1500 synthesizer will not wrap these ports by default. We do not need to specify them in this field.

\%SKIP_SIGNAL_NAME{<port name list> ;}

\%INSTRUCTION_TABLE Field

From \%INSTRUCTION_TABLE field, we can determine IEEE1500 instruction we want to use. Except for user-defined instructions, other IEEE1500 instructions names follow the name suggested by IEEE 1500. From this field, users can tell instructions supported by the Turbo1500. The syntax for user defined instructions has not been explained here as we have not used it.
%INSTRUCTION_TABLE

%INSTRUCTION WS_BYPASS = <instruction binary code>; //mandatory
%INSTRUCTION WS_EXTTEST = <instruction binary code>; //mandatory
%INSTRUCTION WS_INTTEST_RING = <instruction binary code>; //mandatory
%INSTRUCTION WS_INTTEST_SCAN = <instruction binary code>; 
%INSTRUCTION WP_EXTTEST = <instruction binary code>; 
%INSTRUCTION WP_INTTEST_RING = <instruction binary code>; 
%INSTRUCTION WP_INTTEST_SCAN = <instruction binary code>; 
%INSTRUCTION WS_PRELOAD = <instruction binary code>; 
%INSTRUCTION WP_PRELOAD = <instruction binary code>; 
%INSTRUCTION W_CLAMP = <instruction binary code>; 
%INSTRUCTION W_SAFE = <instruction binary code>; 

Now let us take our sender core i.e s27 circuit with the handshaking source module and write the .cfg file.

.cfg file for sender core

We have used the standard naming convention for wrapper interface ports i.e. WRCK, WRSTN, CAPTUREWR etc. as mentioned by IEEE 1500 standard. The input and output wrapper cells are of the same kind. Among the available wrapper cells, we have chosen STI_ONE_FF_CELL because of its simplicity. The pin order list will specify the name of .pin file. In our case it is HS_S27_SRC_top_port.pin which is explained in later sections. The name of wrapper chain file is HS_S27_SRC.wpso. Fig 4.6 is used to understand the .cfg file in a better way. All the input and output ports shown in the figure will be under %GLOBAL_SIGNAL field. The pins Clk, TAKE_IT_TG and GOT_IT_TG of the sender core that do not require wrapper
cells and are mentioned under %SKIP_SIGNAL_NAME field. The .cfg file for the sender core is given below.

```plaintext
%GLOBAL_SIGNAL
{
%WRCK WRCK;
%WRSTN WRSTN;
%SHIFTWR SHIFTWR;
%UPDATEWR UPDATEWR;
%SELECTWIR SELECTWIR;
%CAPTUREWR CAPTUREWR;
%WP_SI WP_SI;
%WP_SO WP_SO;
%PARALLEL_IN TAM_in;
%PARALLEL_OUT TAM_out;
}

%WRAPPER_CELL
{
%INPUT_WRAPPER STI_ONE_FF_CELL;
%OUTPUT_WRAPPER STI_ONE_FF_CELL;
}

%PIN_ORDER_LIST
{
%FILE_NAME HS_S27_SRC_top_port.pin;
}

%WRAPPER_CHAIN_FILE
{
%FILE_NAME HS_S27_SRC.wpso;
}

%SKIP_SIGNAL_NAME
{
Clk,TAKE_IT_TG,GOT_IT_TG;
}

%INSTRUCTION_TABLE
{
%INSTRUCTION WS_BYPASS = (00000000); %INSTRUCTION WS_EXTEST = (00000001); %INSTRUCTION WP_EXTEST = (00000101); %INSTRUCTION WS_INTEST_SCAN = (00000010); %INSTRUCTION WP_INTEST_SCAN = (00000011); %INSTRUCTION WP_INTEST_SCAN = (00000011);
}
```
Figure 4.6: Figure for .cfg file
4.6.2 Library File (.lib)

This file defines the I/O pin names of the Wrapper cells. This information is used during Wrapper cells insertion. In order for the SynTest IEEE1500 synthesizer to correctly synthesize wrapper cells, the SynTest IEEE1500 synthesizer must know the function (role) of each port within a wrapper cell. This library file will provide such information for the SynTest IEEE1500 synthesizer.

**NOTE:**

1. %WRAPPER_CELL field specified in the configuration file is used to request IEEE1500 synthesizer to select a module and a wrapper cell. %WRAPPER_CELL field specified in the library file provides the information about the function of each port of wrapper cell’s module.

2. If module “A” is specified in the library file, but not in the configuration file, the SynTest IEEE1500 synthesizer will not use this module “A” as a wrapper cell.

3. If module “A” is specified in the configuration file, but not in the library file, the SynTest IEEE1500 synthesizer will report error message and request that users must specify the function of each port of module “A” in the library file.

Now let us write the .lib file for the sender core i.e s27 circuit with the hand-shaking source module.

**.lib file for sender core**

Since we have used STI_ONE_FF_CELL as the wrapper cell for both input and output ports, we need to specify the function of each port of this cell in .lib file. Though we have not used STI_ONE_FF_CELL_WITH_SAFE_0 module, it
has been described just as an example. The verilog files of the wrapper cells will be included when we run the `ctagsyn` synthesizer along with the verilog files of sender core. The `.lib` file for the sender core is given below.

```verbatim
%WRAPPER_CELL STI_ONE_FF_CELL
{
%DATA_IN CFI;
%DATA_OUT CFO;
%SCAN_IN CTI;
%SCAN_OUT CTO;
%WP_SHIFT shift;
%APPLY apply;
%CAPTURE_CLK capture_clk;
}

%WRAPPER_CELL STI_ONE_FF_CELL_WITH_SAFE_0
{
%DATA_IN CFI;
%DATA_OUT CFO;
%SCAN_IN CTI;
%SCAN_OUT CTO;
%WP_SHIFT shift;
%APPLY apply;
%CAPTURE_CLK capture_clk;
%SAFE safe;
}
```

### 4.6.3 Wrapper Chain File (.wpso)

This section explains how to specify wrapper chain file, which determines the number of wrapper chains and the order of pins within one wrapper chain.

1. **Syntax if core has no scanchain**:

   ```verbatim
   %WRAPPER_CHAIN <chain name>{
   port name0(1|0), port name1(1|0), port name2(1|0), ... port nameN(1|0);
   }
   ```

   **Note**: ‘0|1’ means safe value 0 or 1.

2. **Syntax with scanchain**:

   ```verbatim
   %WRAPPER_CHAIN <chain name>{
   #<scan chain name>, port name0(1|0), ... port nameN(1|0);
   ```
Chapter 4 Wrapper Design For The Cores

Note: Item “#<scan chain name>” can be inserted in any location.

We must specify .wpso file to tell Turbo1500 information about the number of wrapper chains and port’s position within one wrapper chain. If we want to wrap ports with wrapper cell with safe value, we must specify ‘safe’ for those ports. Under IEEE1500 architecture supported by SynTest, we can also stitch wrapper cells with scan chains. We can only specify scan chains in the one %WRAPPER_CHAIN field. Each wrapper chain occupies one bit of TAM bus. %SCAN_CHAIN_FILE specifies the information about scan chains. About how to specify this file, see Section 4.6.4.

Now let us write the .wpso file for the sender core i.e s27 circuit with the handshaking source module.

.wpso file for sender core

We have decided to use the TAM ports as shown in the Fig 4.6. So now we have two scan chains, one with all the input and output wrapper cells stitched together and the other with the internal scanchain including the SI and SO wrapper cells. The length of wp_chain0 is 8 and length of wp_chain1 is 17 (internal scan chain length 15 + SI wrapper cell + SO wrapper cell). The order in which the wrapper cells are stitched in wp_chain0 is the same as order in which the port names are specified in the .wpso file shown below. The details of scan_chain_0 will be given in ‘HS_S27_SRC.pso’ file. Fig 4.7 shows the scan chains in detail.

%WRAPPER_CHAIN wp_chain0
{
  SRC_DATA_IN[3];
  SRC_DATA_IN[2];
  SRC_DATA_IN[1];
  SRC_DATA_IN[0];
  SRC_DATA_OUT[0];
  SRC_DATA_OUT[1];
4.6.4 Scan Chain File (.pso)

This file provides information about the scan chains in the core for Turbo1500.

```plaintext
%SCAN_MODE <port name>;
%TEST_MODE <port name>;
%SCAN_CHAIN <scan chain name> {
    %SI <port name>;
    %SO <port name>;
}```
Chapter 4 Wrapper Design For The Cores

\%
\%LENGTH <integer>;
\%CLK <port name list>;
\}

Note: If no scan chain name is specified within %WRAPPER_CHAIN field, this file is unnecessary.

.pso file for sender core

Since we have specified scan chain scan_chain_0 in .wpso file of sender core, we need to give the details of this scan chain in the .pso file shown below.

\%
\%SCAN_MODE SE;
\%TEST_MODE T_N;
\%SCAN_CHAIN scan_chain_0
{
  \%SI SI;
  \%SO SO;
  \%LENGTH 15;
  \%CLK Clk;
}


4.6.5 Pin File (.pin)

The file is needed by `ctag2ctl` program to create CTL file to describe the circuitry with wrapper chains.

%PACKAGE\_NAME <top module name> <number of pins>

| PIN\_INDEX | The index of port |
| PIN\_NAME   | Port name of top module with wrapper chain |
| PIN\_TYPE   | Port type |
| ACTIVE\_STATUS | Available value of Signal connected to the port |
| DEFAULT\_STATUS | Disable value of Signal connected to the port. |

**Note:** There are 5 notations in PIN\_TYPE field and Signal status has 4 types given in tables below.

| C  | This is a clock port |
| I  | This is an input port |
| O  | This is an output port |
| E  | This is an enable port |
| B  | This is a bidirectional port |

Table 4.4: Notations of PIN\_TYPE field

| P  | clock |
| U  | high level signal available |
| N  | unknown value |
| D  | low level signal available |

Table 4.5: Notations of signal status
### .pin file for sender core

The .pin file for sender core is straightforward and is given below.

<table>
<thead>
<tr>
<th>%PACKAGE_TYPE</th>
<th>BS_S27_SRC</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>//PIN_INDEX</td>
<td>PIN_NAME</td>
<td>PIN_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SI</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>SE</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>T_N</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>SRC_DATA_IN[3:0]</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>GOT_IT_TG</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>TAKE_IT_TG</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>SRC_DATA_OUT[3:0]</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>SO</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>WRCK</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>WRSTN</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>UPDATEWR</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>SHIFTWR</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>CAPTUREWR</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>SELECTWR</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>WP_SI</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>WP_SO</td>
<td>O</td>
</tr>
</tbody>
</table>

All the files discussed above must also be written for receiver core. These files are given in Appendix C.

The input files for `ctagsyn` are:

- cfg file (.cfg)
- library file (.lib)
- wrapper chain file (.wpso)
- scan chain file (.pso)
- Verilog files which define the core (say top.v) and wrapper cells.

The output files generated are:
- CtagWrappedModule_of_<top>.v :: Synthesizable IEEE 1500 Verilog RTL Code. The top-level name of the netlist that is wrapped by 1500 interface logic.

- CtagCtrl_Of_<top>.v :: IEEE1500 Controller netlist.

- MUX Netlist File. This Mux netlist is needed during logic synthesis and verilog simulation

- Log File

The **ctag2ctl** program is to create a test bench to verify the result of Turbo1500-synthesis and create CTL description files about instructions defined by %INSTRUCTION_TABLE in .cfg file.

The test bench created has functions described as follows:

- Do flush test on wrapper chains to verify they are stitched correctly.

- Verify bypass mode to test the integrity of the wrapper chain and instruction controller.

- Verify instruction set to test the correctness of the instruction operations.

The input files for **ctag2ctl** are:

- CtagWrappedModule_of_<top>.v

- CtagCtrl_Of_<top>.v

- CtagWrappedModule_of_<top>.pin

- STI.WRAPPER_CELL.v :: This file includes wrapper cells RTL code.

- STI_LIB.v :: This file includes mux RTL code. This Mux netlist is needed during IEEE1500 logic synthesis and verilog simulation.
Chapter 4 Wrapper Design For The Cores

- cfg file (.cfg)
- library file (.lib)
- wrapper chain file (.wpso)
- scan chain file (.pso)

The output files generated are:

- CtagWrappedModule_of_<top>.tp :: This file contains the test patterns for bypass mode and wrapper chain flush verification.
- Testbench_of_CtagWrappedModule_of_<top>.v :: This file is a test bench file created by the program for IEEE 1500 verification.
- <Instruction name>.CTL :: A set of CTL files is used to generate test patterns.
- CtagWrappedModule_of_<top>.int :: The file contains the top-level port information.
- Log File :: This file records the result information of the tool’s execution.

Note: Turbo1500 creates CTL files according to the instructions of Instruction Table defined in .cfg file.

All the files created for s27 sender and receiver are shown in Appendix C.

Once the wrapped cores of s27 sender and receiver are generated by Turbo1500, we need to manually stitch them together along with the Test Clock Generator circuit as shown in Fig 4.8.
4.7 Testing of the wrapped cores

Figure 4.8 below shows the top level schematic after the source and destination cores are wrapped with IEEE 1500 wrappers using Turbo 1500.

![Diagram of wrapped cores]

**Figure 4.8: Wrapped Source and Destination Cores**

The source domain has 5 functional inputs and 5 functional outputs. All the functional ports are wrapped except the TAKE_IT_TG and GOT_IT_TG pins. So
Chapter 4 Wrapper Design For The Cores

the wrapper length of one scan chain in source core is 8. As per the IEEE 1500 standard, the test mode pins such as T_N, SE, RST etc and clock pins will not be wrapped. However SI and SO are wrapped. To make use of the parallel instruction such as WP_INTEST_SCAN and to reduce the test time, the wrapper and the internal scan chain are divided into two separate scan chains. The first scan chain consists of all the wrapper cells of functional ports (scan chain length is 8). The second scan chain consists of the internal scan chain of the core and the wrapper cells of SI and SO pins (scan chain length is 17). Similarly, the destination side has been divided into two scan chains, one of length 5 and other of length 14. The TAM ports WPI[0:1] and WPO[0:1] are used to access these scan chains. The TAM port WPI[0] is connected to the scan chain which connects all the wrapper cells (except SI and SO wrapper cells ) and the TAM port WPI[1] is connected to the internal scan chain via the wrapper cells of SI and SO pins.

**NOTE:** The scan clock SCK is no longer required when we use the wrapper. WRCK is used in the shift mode.

### 4.7.1 Test Patterns for wrapped cores

Test patterns to generate $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions on request and acknowledge signals are given below.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Transition</th>
<th>Test Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take_IT_TG</td>
<td>$0 \rightarrow 1$</td>
<td>“0000010000000000” in Core 1 for 17 WRCK cycles</td>
</tr>
<tr>
<td></td>
<td>$1 \rightarrow 0$</td>
<td>All 1’s in Core 1 for 17 WRCK cycles</td>
</tr>
<tr>
<td>GOT_IT_TG</td>
<td>$0 \rightarrow 1$</td>
<td>“000000000000000010” in Core 2 for 19 WRCK cycles</td>
</tr>
<tr>
<td></td>
<td>$1 \rightarrow 0$</td>
<td>“00000000000000110” in Core 2 for 19 WRCK cycles</td>
</tr>
</tbody>
</table>

**Note:** The reason for two extra clock cycles to create a transition on GOT_IT_TG
has already been explained in the previous chapter.

By following the procedure explained in Section 4.5.1, we scan in the opcode for WP_INTEST_SCAN instruction and depending on the fault that is being detected, we scan in the test patterns for rising or falling transition on request and acknowledge signal as required. The test patterns are applied at the WPI[1] TAM ports of the corresponding cores. Now we switch from SHIFT mode to CAPTURE mode. This is done by making WRSTN = 0 and SE = 0. When WRSTN is made ‘0’, the wrapper control circuitry pushes all 0’s into the WIR (which is the opcode for WS_BYPASS instruction). The WS_BYPASS instruction puts the cores in functional mode. During this period, three at-speed capture pulses are applied and the output of the synchronizer is captured into the scan chain. We then switch from CAPTURE mode to SHIFT mode to scan out the captured response of the synchronizer. This is done by making SELECTWIR = 1 and WRSTN = 1. When WRSTN is made high, the wrapper is enabled again. As SELECTWIR is already high, we now scan in the opcode for WP_INTEST_SCAN instruction into the WIR’s of both the cores through their respective WSI pins. We need to remember that the instruction opcode stored in the WIR does not influence the current wrapper mode until it is made active by the WIR update operation. Therefore we make UPDATEWR = 1 for one clock cycle to update the wrappers with the new instruction. The last step is to make SELECTWIR = 0 so that the WIR holds its current instruction. We should now make SHIFTWR = 1 and SE = 1 to observe the captured response of the synchronizer at WPO[1] port of the corresponding core. Figure 4.9 shows the timing diagram to load WP_INTEST_SCAN instruction and the capture window to detect faults in a synchronizer in a wrapper design.

The above procedure together with appropriate test patterns is followed to detect all the bridging and open faults in a synchronizer in an SOC environment.
The number of WRCK cycles after which the output can be observed at WPO[1] for a particular fault with the wrapper will be one more than the number of SCK cycles required while detecting the same fault without wrapper. This is because of the wrapper cell that has been stitched to the SO pin of the core.

Figure 4.9: Timing diagram for fault detection in synchronizer using wrapper design
In this research, we implemented a two-way handshaking protocol using two flip-flop synchronizers to transfer data from one clock domain to another asynchronous clock domain. The handshake interface design was made testable by using scanable flip-flops and some additional logic. Since the launch-on-capture scheme discussed in 1.4 is not an appropriate method to test inter-clock delay faults and structural faults, we modified the capture window to detect such faults. We also designed a Test Clock Generator which implements the modified capture window, generating one at-speed launch pulse for the source module followed by three at-speed capture pulses for the destination module. We then developed a detailed test strategy which uses this capture window to detect all the bridging and open faults in the synchronizer. The input test patterns required in the scan chains of source and destination modules to trigger the faults and to capture the faulty output are also derived. We have also concluded that metastable fault cannot be detected using this capture method due to its probabilistic nature.

We have shown that this method can be extended to an SoC design which can consist of a number of cores, each operating at a different frequency. In future, work can be done to improve the handshaking and test circuitry.


A

Faults in a synchronizer

This appendix is based on the work of Hyoung-Kook Kim. This chapter provides a brief introduction to the different kinds of faults in a two flip-flop synchronizer. The faults can be categorized into two types:

- Bridging Faults
- Open Faults

The Fig A.1 below shows the schematic of the synchronizer used in the handshake protocol. The synchronizer consists of two positive edge triggered D flip flops, each consisting of four transmission gates and four inverters.

Figure A.1: Schematic of Synchronizer
A.1 Bridging Faults

All the cases of bridging faults have been considered. The bridging faults within the D flip flop are represented as $bcxy$ and the bridging faults between the two D flip flops are represented as $bbxy$, where $x$ and $y$ denote the nodes that are bridged. The synchronizer shows different fault behaviors depending on input signal arrival time, input signal pattern and also the resistance of the bridging. All the bridging faults have been explained in detail in [8]. Tables A.1, A.2, A.3, give the summary of different cases of occurrences of these faults.
### Appendix A. Faults in a synchronizer

#### Table A.1: Category of all bridging faults

<table>
<thead>
<tr>
<th>Category</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stuck-At Fault</td>
<td>SA0</td>
<td>stuck-at-0</td>
</tr>
<tr>
<td></td>
<td>SA1</td>
<td>stuck-at-1</td>
</tr>
<tr>
<td>Functional Faults</td>
<td>TFT1</td>
<td>Output observed one clock cycle earlier than expected</td>
</tr>
<tr>
<td></td>
<td>TFT2</td>
<td>Inverted output observed one clock cycle earlier than expected</td>
</tr>
<tr>
<td></td>
<td>TFT3</td>
<td>Output observed one clock cycle later than expected</td>
</tr>
<tr>
<td></td>
<td>TFT4</td>
<td>Inverted output observed one clock cycle later than expected</td>
</tr>
<tr>
<td></td>
<td>DO</td>
<td>Output with delay of the synchronizer</td>
</tr>
<tr>
<td></td>
<td>IDO</td>
<td>Inverted output with delay of the synchronizer</td>
</tr>
<tr>
<td>Pulse Test Output</td>
<td>PTO1</td>
<td>Synchronizer generates pulses depending on the input signal</td>
</tr>
<tr>
<td></td>
<td>PTO2</td>
<td>Synchronizer generates pulses independent of the input signal</td>
</tr>
<tr>
<td>One Time Pulse</td>
<td>OTP0</td>
<td>Synchronizer generates 1-0-1 one time pulse</td>
</tr>
<tr>
<td></td>
<td>OTP1</td>
<td>Synchronizer generates 0-1-0 one time pulse</td>
</tr>
<tr>
<td>Metastable Output</td>
<td>MO</td>
<td>The output of the synchronizer appears at unpredictable time</td>
</tr>
<tr>
<td>Internal Oscillation Fault</td>
<td>IOF</td>
<td>A signal oscillation occurs in the synchronizer by a loop, and the output of a synchronizer depends on the applied clock frequency.</td>
</tr>
<tr>
<td>Undefined Value</td>
<td>UD</td>
<td>The output voltage of the synchronizer is in the range of $V_{DD} * 0.3$ to $V_{DD} * 0.7$.</td>
</tr>
</tbody>
</table>
## Appendix A. Faults in a synchronizer

### Chapter A. Faults in a synchronizer

Table A.2: Fault behaviors with bridging defects in DFF1

<table>
<thead>
<tr>
<th>Fault Name</th>
<th>Input pattern</th>
<th>$\Omega$</th>
<th>Sensitive Resistance</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b12$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>TFT1$^1$</td>
<td>2.910kΩ</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>TFT1$^1$</td>
<td>4.355kΩ</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>b13$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>IDO/TFT2$^2$</td>
<td>1.865kΩ</td>
<td>IDO</td>
<td>IDO</td>
<td>IDO</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>IDO/TFT2$^2$</td>
<td>1.738kΩ</td>
<td>IDO</td>
<td>IDO</td>
<td>IDO</td>
</tr>
<tr>
<td>b14$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>TFT1$^1$</td>
<td>254Ω</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>b15$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>TFT2$^3$</td>
<td>11.58kΩ</td>
<td>SA1</td>
<td>SA1$^{14}$</td>
<td>SA1$^{14}$</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>TFT2$^3$</td>
<td>11.6kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>b16$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>TFT1$^1$</td>
<td>527Ω</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>TFT1$^1$</td>
<td>254Ω</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>b17$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>TFT2$^3$</td>
<td>997Ω</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>TFT2$^3$</td>
<td>980Ω</td>
<td>TFT2</td>
<td>OTP1</td>
<td>OTP1</td>
</tr>
<tr>
<td>b22$_{DFF1}$</td>
<td>both</td>
<td>SA1</td>
<td>3.847kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>b24$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>SA0</td>
<td>6.697kΩ</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>SA0</td>
<td>6.308kΩ</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
<tr>
<td>b25$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>SA1</td>
<td>10.39kΩ</td>
<td>SA1</td>
<td>SA1/OTP1$^{14}$</td>
<td>SA1/OTP1$^{14}$</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>SA1</td>
<td>10.40kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1$^{15}$</td>
</tr>
<tr>
<td>b26$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>SA0</td>
<td>8.968kΩ</td>
<td>SA0</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>SA1</td>
<td>6.627kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>b27$_{DFF1}$</td>
<td>both</td>
<td>IOF</td>
<td>Not Available (N/A)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b33$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>SA1</td>
<td>1.792kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>SA1</td>
<td>1.972kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>b35$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>TFT1$^1$</td>
<td>1.153kΩ</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>TFT1$^1$</td>
<td>1.462kΩ</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>b36$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>IDO/TFT4$^4$</td>
<td>1.933kΩ</td>
<td>IDO</td>
<td>IDO</td>
<td>IDO</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>IDO/TFT4$^4$</td>
<td>1.816kΩ</td>
<td>IDO</td>
<td>IDO</td>
<td>IDO</td>
</tr>
<tr>
<td>b37$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>SA0</td>
<td>996Ω</td>
<td>SA0</td>
<td>SA0</td>
<td>TFT3</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>TFT1$^4$</td>
<td>253Ω</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>b45$_{DFF1}$</td>
<td>both</td>
<td>TFT2$^3$</td>
<td>11.601kΩ</td>
<td>IDO</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>b46$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>TFT1$^1$</td>
<td>371Ω</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>TFT1$^1$</td>
<td>73Ω</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>b47$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>IDO/TFT2$^{20}$</td>
<td>878Ω</td>
<td>TFT2</td>
<td>PTO1</td>
<td>PTO1</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>IDO/TFT2$^{21}$</td>
<td>687Ω</td>
<td>TFT2</td>
<td>PTO2</td>
<td>PTO2</td>
</tr>
<tr>
<td>b56$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>SA0</td>
<td>3.992kΩ</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>SA0</td>
<td>3.996kΩ</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
<tr>
<td>b57$_{DFF1}$</td>
<td>both</td>
<td>SA1</td>
<td>6.609kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>b67$_{DFF1}$</td>
<td>0 $\rightarrow$ 1</td>
<td>SA0</td>
<td>1.769kΩ</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
<tr>
<td></td>
<td>1 $\rightarrow$ 0</td>
<td>SA0</td>
<td>1.564kΩ</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
</tbody>
</table>

1. TFT1 in $t_{dist} = -0.1ns \sim 0.7ns$. 2. TFT2 in $t_{dist} = -0.1ns \sim 0.3ns$, and IDO otherwise. 3. TFT1 in $t_{dist} = -0.1ns \sim 0.3ns$. 4. Except $t_{dist} = -0.5ns \sim -0.3ns$. 5. Except $t_{dist} = -0.3ns$. 6. TFT4 in $t_{dist} = -0.7ns \sim -0.3ns$, and IDO otherwise. 7. IDO in $t_{dist} = -0.7ns \sim -0.3ns$, and TFT2 otherwise. 8. Except $t_{dist} = -0.7ns \sim -0.3ns$. 9. Except $t_{dist} = -0.9ns \sim -0.3ns$. 10. IDO in $t_{dist} = -0.7ns \sim -0.3ns$, and TFT2 otherwise. 11. IDO in $t_{dist} = -0.5ns \sim -0.3ns$, and TFT2 otherwise. 12. OTP0 if $11.528k\Omega \leq R \leq 11.597k\Omega$. 13. TFT3 at $R = 244\Omega$. 14. SA1 if $t_{dist} = -0.5ns$ and OTP0 if $t_{dist} = 0.5ns$. 15. TFT3 if $R = 10.39k\Omega$. 16. OTP0 if $R = 3.9795k\Omega$. 17. $t_{dist} = -0.3ns$, and TFT2 otherwise. 18. Except $t_{dist} = -0.3ns$.
### Appendix A. Faults in a synchronizer

#### Table A.3: Fault behaviors with bridging defects in DFF2

<table>
<thead>
<tr>
<th>Fault Name</th>
<th>Input pattern</th>
<th>0Ω</th>
<th>Sensitive Resistance</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>bc12_DFF1</td>
<td>0 → 1</td>
<td>TFT1</td>
<td>3.404Ω</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT1</td>
<td>6.087Ω</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>bc13_DFF2</td>
<td>0 → 1</td>
<td>TFT2</td>
<td>2.573Ω</td>
<td>DO</td>
<td>TFT2</td>
<td>DO</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>IDO</td>
<td>7.266Ω</td>
<td>DO</td>
<td>DO</td>
<td>DO</td>
</tr>
<tr>
<td>bc14_DFF2</td>
<td>0 → 1</td>
<td>TFT1</td>
<td>273Ω</td>
<td>TFT1</td>
<td>TFT2</td>
<td>TFT2</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1</td>
<td>459Ω</td>
<td>SA1</td>
<td>IDO</td>
<td>IDO</td>
</tr>
<tr>
<td>bc15_DFF2</td>
<td>0 → 1</td>
<td>TFT2</td>
<td>10.59kΩ</td>
<td>TFT2</td>
<td>OTP0</td>
<td>OTP0</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT2</td>
<td>20.082kΩ</td>
<td>SA1</td>
<td>PTO1</td>
<td>DO</td>
</tr>
<tr>
<td>bc16_DFF2</td>
<td>0 → 1</td>
<td>SA1</td>
<td>1.20Ω</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1</td>
<td>99Ω</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>bc17_DFF2</td>
<td>both</td>
<td>PTO1</td>
<td>99Ω</td>
<td>PTO1</td>
<td>PTO1</td>
<td>PTO1</td>
</tr>
<tr>
<td>bc23_DFF2</td>
<td>0 → 1</td>
<td>SA1</td>
<td>5.450KΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>DO</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1</td>
<td>5.318KΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>DO</td>
</tr>
<tr>
<td>bc24_DFF2</td>
<td>0 → 1</td>
<td>SA0</td>
<td>6.425kΩ</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA0</td>
<td>6.367kΩ</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
<tr>
<td>bc25_DFF2</td>
<td>0 → 1</td>
<td>SA1</td>
<td>9.32kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1</td>
<td>19.04kΩ</td>
<td>SA1</td>
<td>DO</td>
<td>DO</td>
</tr>
<tr>
<td>bc26_DFF2</td>
<td>0 → 1</td>
<td>SA1</td>
<td>8.116KΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1</td>
<td>6.622KΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>bc27_DFF2</td>
<td>both</td>
<td>IOF</td>
<td>Not Available(N/A)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bc34_DFF2</td>
<td>0 → 1</td>
<td>SA1</td>
<td>3.233kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>DO</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1</td>
<td>3.233kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>DO</td>
</tr>
<tr>
<td>bc35_DFF2</td>
<td>0 → 1</td>
<td>TFT1</td>
<td>8.083kΩ</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT1</td>
<td>5.264kΩ</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>bc36_DFF2</td>
<td>0 → 1</td>
<td>SA1</td>
<td>2.881kΩ</td>
<td>UD</td>
<td>UD</td>
<td>UD</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT2</td>
<td>2.597kΩ</td>
<td>TFT2</td>
<td>UD</td>
<td>UD</td>
</tr>
<tr>
<td>bc37_DFF2</td>
<td>0 → 1</td>
<td>SA0</td>
<td>620Ω</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT1</td>
<td>661Ω</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>bc45_DFF2</td>
<td>0 → 1</td>
<td>TFT2</td>
<td>10.59kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>DO</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT2</td>
<td>20.106kΩ</td>
<td>SA1</td>
<td>PTO1</td>
<td>DO</td>
</tr>
<tr>
<td>bc46_DFF2</td>
<td>0 → 1</td>
<td>TFT1</td>
<td>527kΩ</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT1</td>
<td>270Ω</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>bc47_DFF2</td>
<td>0 → 1</td>
<td>PTO1</td>
<td>910Ω</td>
<td>PTO1</td>
<td>PTO1</td>
<td>PTO1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>PTO1</td>
<td>878Ω</td>
<td>PTO1</td>
<td>PTO1</td>
<td>PTO1</td>
</tr>
<tr>
<td>bc56_DFF2</td>
<td>0 → 1</td>
<td>UD</td>
<td>3.418kΩ</td>
<td>UD</td>
<td>UD</td>
<td>UD</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>UD</td>
<td>4.981kΩ</td>
<td>UD</td>
<td>UD</td>
<td>UD</td>
</tr>
<tr>
<td>bc57_DFF2</td>
<td>0 → 1</td>
<td>SA1</td>
<td>7.304kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1</td>
<td>9.952kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>DO</td>
</tr>
<tr>
<td>bc67_DFF2</td>
<td>0 → 1</td>
<td>UD</td>
<td>2.832kΩ</td>
<td>UD</td>
<td>UD</td>
<td>UD</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>UD</td>
<td>2.656kΩ</td>
<td>UD</td>
<td>UD</td>
<td>UD</td>
</tr>
</tbody>
</table>
Table A.4: Fault behaviors with bridging defects between DFF1 & DFF2

<table>
<thead>
<tr>
<th>Fault Name</th>
<th>Input pattern</th>
<th>0Ω</th>
<th>Sensitive Resistance</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>6612</td>
<td>0 → 1</td>
<td>TFT1</td>
<td>5.190kΩ</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT1</td>
<td>7.136kΩ</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>6613</td>
<td>0 → 1</td>
<td>TFT2</td>
<td>2.324kΩ</td>
<td>TFT2</td>
<td>DO</td>
<td>DO</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT2</td>
<td>2.417kΩ</td>
<td>TFT2</td>
<td>DO</td>
<td>DO</td>
</tr>
<tr>
<td>6614</td>
<td>0 → 1</td>
<td>SA1</td>
<td>606Ω</td>
<td>OTP0</td>
<td>OTP0</td>
<td>OTP0</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1</td>
<td>410Ω</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>6615</td>
<td>0 → 1</td>
<td>TFT2</td>
<td>10.591kΩ</td>
<td>TFT2</td>
<td>OTP0</td>
<td>OTP0</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT2</td>
<td>12.921kΩ</td>
<td>TFT2</td>
<td>SA1</td>
<td>OTP0</td>
</tr>
<tr>
<td>6616</td>
<td>0 → 1</td>
<td>UD</td>
<td>1.079kΩ</td>
<td>UD</td>
<td>UD</td>
<td>UD</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>UD</td>
<td>1.931kΩ</td>
<td>UD</td>
<td>UD</td>
<td>UD</td>
</tr>
<tr>
<td>6617</td>
<td>0 → 1</td>
<td>PTO1</td>
<td>1.047kΩ</td>
<td>PTO1</td>
<td>PTO1</td>
<td>PTO1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>PTO1</td>
<td>1kΩ</td>
<td>PTO1</td>
<td>PTO1</td>
<td>PTO1</td>
</tr>
<tr>
<td>6622</td>
<td>0 → 1</td>
<td>SA0</td>
<td>7.344kΩ</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1</td>
<td>6.719kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>6623</td>
<td>both</td>
<td>PTO2</td>
<td>Not Available(N/A)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6624</td>
<td>0 → 1</td>
<td>SA1</td>
<td>9.170kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1</td>
<td>8.623kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>6625</td>
<td>0 → 1</td>
<td>SA1</td>
<td>28.174kΩ</td>
<td>SA1</td>
<td>DO</td>
<td>DO</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1</td>
<td>9.25kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>6626</td>
<td>0 → 1</td>
<td>SA1</td>
<td>9.182kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1</td>
<td>6.719kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>6627</td>
<td>both</td>
<td>PTO2</td>
<td>Not Available(N/A)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6632</td>
<td>0 → 1</td>
<td>TFT2</td>
<td>8.626kΩ</td>
<td>TFT2</td>
<td>TFT2</td>
<td>TFT2</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT2</td>
<td>7.431kΩ</td>
<td>TFT2</td>
<td>TFT2</td>
<td>TFT2</td>
</tr>
<tr>
<td>6633</td>
<td>0 → 1</td>
<td>SA0</td>
<td>1.162kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT1</td>
<td>1.162kΩ</td>
<td>SA1</td>
<td>SA1</td>
<td>F/F1</td>
</tr>
<tr>
<td>6634</td>
<td>0 → 1</td>
<td>PTO1</td>
<td>911Ω</td>
<td>PTO1</td>
<td>PTO1</td>
<td>PTO1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>PTO1</td>
<td>1.016kΩ</td>
<td>PTO1</td>
<td>PTO1</td>
<td>PTO1</td>
</tr>
<tr>
<td>6635</td>
<td>0 → 1</td>
<td>TFT1</td>
<td>8.506kΩ</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT1</td>
<td>6.785kΩ</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>6636</td>
<td>0 → 1</td>
<td>PTO1</td>
<td>2.559kΩ</td>
<td>UD</td>
<td>UD</td>
<td>UD</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>PTO1</td>
<td>1.529kΩ</td>
<td>UD</td>
<td>UD</td>
<td>UD</td>
</tr>
<tr>
<td>6637</td>
<td>0 → 1</td>
<td>PTO1</td>
<td>1.138kΩ</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>PTO1</td>
<td>1.069kΩ</td>
<td>TFT3</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>6641</td>
<td>0 → 1</td>
<td>TFT1</td>
<td>5.235kΩ</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT1</td>
<td>7.086kΩ</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>6643</td>
<td>0 → 1</td>
<td>TFT2</td>
<td>2.395kΩ</td>
<td>TFT2</td>
<td>DO</td>
<td>DO</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT2</td>
<td>7.030kΩ</td>
<td>DO</td>
<td>DO</td>
<td>DO</td>
</tr>
</tbody>
</table>

Continued on next page...
### Appendix A. Faults in a synchronizer

<table>
<thead>
<tr>
<th>Fault Name</th>
<th>Input pattern</th>
<th>$\Omega$ Sensitive Resistance</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta 44$</td>
<td>both</td>
<td>$958\Omega$</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>$\delta 45$</td>
<td>0 → 1</td>
<td>TFT2 $\rightarrow 10.725\Omega$</td>
<td>TFT2</td>
<td>OTP0</td>
<td>OTP0</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT2 $\rightarrow 20.100\Omega$</td>
<td>SA1</td>
<td>PTO1</td>
<td>DO</td>
</tr>
<tr>
<td>$\delta 46$</td>
<td>0 → 1</td>
<td>PTO1 $\rightarrow 900\Omega$</td>
<td>UD</td>
<td>UD</td>
<td>UD</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>PTO1 $\rightarrow 1.739\Omega$</td>
<td>UD</td>
<td>UD</td>
<td>UD</td>
</tr>
<tr>
<td>$\delta 47$</td>
<td>0 → 1</td>
<td>PTO1 $\rightarrow 962\Omega$</td>
<td>PTO1</td>
<td>PTO1</td>
<td>PTO1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>PTO1 $\rightarrow 770\Omega$</td>
<td>PTO1</td>
<td>PTO1</td>
<td>PTO1</td>
</tr>
<tr>
<td>$\delta 52$</td>
<td>0 → 1</td>
<td>SA0 $\rightarrow 6.506\Omega$</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA0 $\rightarrow 5.816\Omega$</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
<tr>
<td>$\delta 53$</td>
<td>0 → 1</td>
<td>SA0 $\rightarrow 10.665k\Omega$</td>
<td>SA0</td>
<td>F/F</td>
<td>SA1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA0 $\rightarrow 10.665k\Omega$</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>$\delta 54$</td>
<td>0 → 1</td>
<td>IOF $\rightarrow 7.925k\Omega$</td>
<td>SA0</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>IOF $\rightarrow 8k\Omega$</td>
<td>SA0</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>$\delta 55$</td>
<td>0 → 1</td>
<td>SA1 $\rightarrow 10.352k\Omega$</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1 $\rightarrow 10.352k\Omega$</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>$\delta 56$</td>
<td>0 → 1</td>
<td>PTO2 $\rightarrow 8.555k\Omega$</td>
<td>PTO2</td>
<td>PTO1</td>
<td>PTO1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>PTO2 $\rightarrow 7.898\Omega$</td>
<td>PTO2</td>
<td>PTO1</td>
<td>PTO1</td>
</tr>
<tr>
<td>$\delta 57$</td>
<td>0 → 1</td>
<td>SA1 $\rightarrow 11.619k\Omega$</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1 $\rightarrow 11.622k\Omega$</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
<tr>
<td>$\delta 72$</td>
<td>0 → 1</td>
<td>TFT2 $\rightarrow 8.632\Omega$</td>
<td>TFT2</td>
<td>TFT2</td>
<td>SA0</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT2 $\rightarrow 7.457k\Omega$</td>
<td>TFT2</td>
<td>F/F</td>
<td>SA0</td>
</tr>
<tr>
<td>$\delta 73$</td>
<td>0 → 1</td>
<td>MO $^1$ Not Available(N/A)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT1 $\rightarrow 253\Omega$</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>$\delta 74$</td>
<td>0 → 1</td>
<td>TFT2 $\rightarrow 877\Omega$</td>
<td>TFT2</td>
<td>PTO1</td>
<td>OTP0</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT2 $\rightarrow 901\Omega$</td>
<td>PTO1</td>
<td>PTO1</td>
<td>PTO1</td>
</tr>
<tr>
<td>$\delta 75$</td>
<td>0 → 1</td>
<td>TFT1 $\rightarrow 8.511k\Omega$</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>TFT1 $\rightarrow 4.483k\Omega$</td>
<td>TFT1</td>
<td>TFT1</td>
<td>TFT1</td>
</tr>
<tr>
<td>$\delta 76$</td>
<td>0 → 1</td>
<td>UD $\rightarrow 1.983k\Omega$</td>
<td>UD</td>
<td>UD</td>
<td>UD</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>UD $\rightarrow 1.319k\Omega$</td>
<td>UD</td>
<td>UD</td>
<td>UD</td>
</tr>
<tr>
<td>$\delta 77$</td>
<td>0 → 1</td>
<td>SA1 $\rightarrow 997\Omega$</td>
<td>SA0</td>
<td>SA0</td>
<td>SA0</td>
</tr>
<tr>
<td></td>
<td>1 → 0</td>
<td>SA1 $\rightarrow 186\Omega$</td>
<td>SA1</td>
<td>SA1</td>
<td>SA1</td>
</tr>
</tbody>
</table>

1. Except $t_{dist} = -0.3\text{ns}$. 2. Except $t_{dist} = -0.5\text{ns} \sim -0.3\text{ns}$. 3. TFT3 at $R = 1.157k\Omega$. 
Verilog Code

top.v

```
//timescale 1ns / 1ps
module top(  
  input T_N,  
  input SE,  
  input SI1,  
  input SI2,  
  output SO1,  
  output SO2,  
  input PTO,  
  output OTP,  
  output Q,  //port wasted  
  input RST,  
  input CLOCK1,  
  input CLOCK2,  
  input SCK,  
  input [3:0] PI,  
  output PO,  
  input DOMAIN_SEL,  
  input FAULT_SEL,  
  input WRCK,  
  input WRSTN,  
  input SELECTWIR,  
  input CAPTUREEWR,  
  input SHIFTWIR,  
  input UPDATEEWR,  
  input WP_SI1,  
  input WP_SI2,  
  output WP_SO1,  
  output WP_SO2,  
  input [1:0] TAM_IN1,  
  input [1:0] TAM_IN2,  
  output [1:0] TAM_OUT1,  
  output [1:0] TAM_OUT2  
);  
```

```
wire take_it_tg_sig,got_it_tg_sig,TCK1,TCK2,tck1_sig,tck2_sig,int_out_sig,pto_out_gated;  
wire temp,temp_clk,temp4,mux_en,so_sig,dual_q;  
wire [3:0] src_dst_data;  
defparam MUX1.wait_delay = 0.1;  
MUX MUX1 (  
  .in_0(CLOCK1),  
  .in_1(tck1_sig),  
  .sel(T_N),  
  .out1(TCK1)  
);  
defparam MUX2.wait_delay = 0.1;  
MUX MUX2 (  
  .in_0(CLOCK2),  
  .in_1(tck2_sig),  
  .sel(T_N),  
  .out1(TCK2)  
);  
TEST_CLK_GEN TEST_CLK_GEN1(  
  .CLOCK1(CLOCK1),  
  .CLOCK2(CLOCK2),  
  .OUT1(TCK1),  
  .OUT2(TCK2)  
);  
```
```verilog
module tb_top;

// Inputs
reg T_N;
reg SE;
reg SI1;
reg SI2;
reg PTO;
reg RST;
reg CLOCK1;
reg CLOCK2;
reg SCK;
reg [3:0] PI;
reg DOMAIN_SEL;
reg FAULT_SEL;
reg WRCK;
reg WRSTN;
reg SELECTWR;
reg CAPTUREWR;
reg SHIFTWR;
reg UPDATEWR;
reg WP_SI;
reg WP_SO;
reg [1:0] TAM_IN;
reg [1:0] TAM_OUT;

// Outputs
endmodule
```

/*timescale 1ns / 1ns

module tb_top;

// Inputs
reg T_N;
reg SE;
reg SI1;
reg SI2;
reg PTO;
reg RST;
reg CLOCK1;
reg CLOCK2;
reg SCK;
reg [3:0] PI;
reg DOMAIN_SEL;
reg FAULT_SEL;
reg WRCK;
reg WRSTN;
reg SELECTWR;
reg CAPTUREWR;
reg SHIFTWR;
reg UPDATEWR;
reg WP_SI;
reg WP_SO;
reg [1:0] TAM_IN;
reg [1:0] TAM_OUT;

// Outputs
endmodule
*/
wire SO1;
wire SO2;
wire OTP;
wire Q;
wire PO;
wire WP_SO1;
wire WP_SO2;
wire [1:0] TAM_OUT1;
wire [1:0] TAM_OUT2;

// Instantiate the Unit Under Test (UUT)
top uut (
  .T_N(T_N), .SE(SE),
  .SI1(SI1),
  .SI2(SI2),
  .SO1(SO1),
  .SO2(SO2),
  .PTO(PTO),
  .OTP(OTP),
  .Q(Q),
  .RST(RST),
  .CLOCK1(CLOCK1),
  .CLOCK2(CLOCK2),
  .PO(PO),
  .PI(PI),
  .DOMAIN_SEL(DOMAIN_SEL),
  .FAULT_SEL(FAULT_SEL),
  .WRCK(WRCK),
  .WRSTN(WRSTN),
  .SELECTWIR(SELECTWIR),
  .UPDATEWR(UPDATEWR),
  .SHIFTWR(SHIFTWR),
  .WP_SI1(WP_SI1),
  .WP_SI2(WP_SI2),
  .WP_SO1(WP_SO1),
  .WP_SO2(WP_SO2),
  .TAM_IN1(TAM_IN1),
  .TAM_IN2(TAM_IN2),
  .TAM_OUT1(TAM_OUT1),
  .TAM_OUT2(TAM_OUT2)
);

always
  #5 CLOCK1 = CLOCK1; // 100 MHz
always
  #3.3 CLOCK2 = CLOCK2; // 151.51 MHz
always
  #10 WRCK = WRCK; // 50 MHz
always
  #20 SCK = SCK; // 25 MHz
initial begin
  // Initialize Inputs
  T_N = 0;
  SE = 0;
  SI1 = 0;
  SI2 = 0;
  PTO = 0;
  RST = 0;
  CLOCK1 = 0;
  CLOCK2 = 0;
  PO = 0;
  PI = 0;
  DOMAIN_SEL = 0;
  FAULT_SEL = 0;
  WRCK = 0;
  WRSTN = 0;
  SELECTWIR = 0;
  UPDATEWR = 0;
  SHIFTWR = 0;
  WP_SI1 = 0;
  WP_SI2 = 0;
  WP_SO1 = 0;
  WP_SO2 = 0;
  TAM_IN1 = 0;
  TAM_IN2 = 0;

  // Wait 100 ns for global reset to finish
  #100;
  WRSTN = 1; // Unreset WRSTN after 100 ns;
  T_N = 1;

  // Testing Synchronizer in Domain-2 (Destination)
  RST = 1;
  DOMAIN_SEL = 0;
  FAULT_SEL = 0;
  PTO = 0; // should be '1' for detecting pulse outputs...
  #25;
  SELECTWIR = 1;
  #20;
SHIFTWR=1;
WP_SI1=0;
WP_SI2=0;
#20;
WP_SI1=0;
WP_SI2=0;
#20;
WP_SI1=0;
WP_SI2=0;
#20;
WP_SI1=0;
WP_SI2=0; ///OPcode for instruction (WP_INTEST_SCAN) Parallel INTEST
#20;
WP_SI1=0;
WP_SI2=0;
#20;
WP_SI1=0;
WP_SI2=0;
#20;
WP_SI1=1;
WP_SI2=1;
#20;
WP_SI1=1;
WP_SI2=1;
#20;
WP_SI1=0;
WP_SI2=0; //making them 0 after scan in of op code
SHIFTWR=0;
UPDATEWR=1;
#20;
UPDATEWR=0;
SELECTWR=0; ///Deselect WIR to hold the instruction in WIR
SHIFTWR=1; ///To shift Test data into the core through the wrapper
#5;
SE = 1;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //1
#20;
TAM_IN1[1]=1;
TAM_IN2[1]=0; //2 1 to make a 1->0 transition on Take_it_tg
#20;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //3
#20;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //4
#20;
TAM_IN1[1]=1;
TAM_IN2[1]=0; //6 1 to make a 0 to 1 transition on Take_it_tg with Domain Sel=0
#20;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //7
#20;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //8
#20;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //9 //last 14 clock cycles for Scan chain in Destination 12+2(wrapper)
#20;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //10
#20;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //11
#20;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //12
#20;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //13
#20;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //14
#20;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //15
#20;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //16
#20;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //17
#20;
TAM_IN1[1]=0;
TAM_IN2[1]=0; //18
#20; TAM_IN1[1]=0;
#20; TAM_IN2[1]=0; //19
#20; TAM_IN1[1]=0;
#20; TAM_IN2[1]=0; //20
#20; TAM_IN1[1]=0;
#20; TAM_IN2[1]=0; //21
#20; TAM_IN1[1]=0;
#20; TAM_IN2[1]=0; //22
#20; TAM_IN2[1]=0;
WRSTN=0; // Reset WRSTN for normal[BYPASS] mode;
SE = 0;
SELECTWIR=1;
# 110;
WRSTN=1; // Set WRSTN for WP_INTEST_SCAN mode;
SELECTWIR=1;
#20; SHIPTWR=1;
WP_SI1=0;
WP_SI2=0;
#20; WP_SI1=0;
WP_SI2=0;
#20; WP_SI1=0;
WP_SI2=0;
WP_SI1=0;
WP_SI2=0;
WP_SI1=0;
WP_SI2=0; //OPcode for instruction (INTEST_SCAN)
#20; WP_SI1=0;
WP_SI2=0; //making them 0 after the opcode for WP_INTEST_SCAN
SHIPTWR=0;
UPDATEWR=1;
#20;
UPDATEWR=0;
SELECTWIR=0; //Deselect WIR to hold the instruction in WIR;
SHIPTWR=1; //To shift Test data into the core through the wrapper
SE = 1;
end
endmodule

Ctag wrapped destination module

```
timescale 1 ns / 10 ps
module CtagWrappedModule_Of_HS_S27_DST_NEW
(
    TAKE_IT_TG,
    GOT_IT_TG,
    CLK,
    SI,
    SE,
    RST,
    PTO,
    OTP,
    SO,
    DST_DATA_IN,
    DST_DATA_OUT,
    //WIP ports of CTAG
    WRCK,
    WRSTN,
    UPDATEWR,
    SHIFTWR,
    CAPTUREWR,
    SELECTWR,
    WP_SI,
    WP_SO,
    TAM_in,
    TAM_out);
input TAKE_IT_TG;
output GOT_IT_TG;
input CLK;
input SI;
input SE;
input RST;
input PTO;
output OTP;
output SO;
input [3:0] DST_DATA_IN;
output DST_DATA_OUT;
input WRCK;
input WRSTN;
input UPDATEWR;
input SHIFTWR;
input CAPTUREWR;
input SELECTWR;
input WP_SI;
output WP_SO;
input [1:0] TAM_in;
output [1:0] TAM_out;
//SYNSTEST_CTAGSYN_ADDED V1.1.0 instance
HS_S27_DST_NEW HS_S27_DST_NEW (
.TAKE_IT_TG (TAKE_IT_TG) ,
.GOT_IT_TG (GOT_IT_TG) ,
.CLK (SYNTEST_CTAGSYN_port_29) ,
.SI (SYNTEST_CTAGSYN_port_19) ,
.SE (SYNTEST_CTAGSYN_port_32) ,
.RST (RST) ,
.PTO (PTO) ,
.OTP (OTP) ,
.SO (SYNTEST_CTAGSYN_port_25) ,
.DST_DATA_IN (SYNTEST_CTAGSYN_port_0) ,
.DST_DATA_OUT (SYNTEST_CTAGSYN_port_7) ) ;
//SYNSTEST_CTAGSYN_ADDED V1.1.0 instance
CtagCtrl_Of_HS_S27_DST_NEW CtagCtrl_Of_HS_S27_DST_NEW (
.WRCK (WRCK) ,
.WRSTN (WRSTN) ,
.UPDATEWR (UPDATEWR) ,
.SHIFTWR (SHIFTWR) ,
.CAPTUREWR (CAPTUREWR) ,
.SELECTWR (SELECTWR) ,
.WP_SI (WP_SI) ,
.WP_SO (WP_SO) ,
.TAM_in (TAM_in[1]) ,
.TAM_in[0] ,
.TAM_out (TAM_out[1] ,
.TAM_out[0] ) ,
.Ctrl_ToWpShift (SYNTEST_CTAGSYN_port_1) ,
.Ctrl_ToOutWpCaptureclk (SYNTEST_CTAGSYN_port_2) ,
.Ctrl_ToOutWpCaptureclk (SYNTEST_CTAGSYN_port_8) ,
.Ctrl_ToOutWpApply (SYNTEST_CTAGSYN_port_3) ,
.Ctrl_ToWpParallel (SYNTEST_CTAGSYN_port_9) ,
.Ctrl_ToParallelIn0 (SYNTEST_CTAGSYN_port_10) ,
.Ctrl_ToParallelOut0 (SYNTEST_CTAGSYN_port_11) ,
.Ctrl_ToParallelIn1 (SYNTEST_CTAGSYN_port_13) ,
.Ctrl_ToParallelOut1 (SYNTEST_CTAGSYN_port_12) ,
.scan_mode (SYNTEST_CTAGSYN_port_14) ,
.scan_select_sys_clk (SYNTEST_CTAGSYN_port_15) ,
```

Appendix B. Verilog Code

```verilog
.comand_start_clk (SYNTEST_CTAGSYN_port_30),
.comand_instruction (SYNTEST_CTAGSYN_port_20),
.Ctrl_ToeSerialOut (SYNTEST_CTAGSYN_port_11),
.Ctrl_FromSerialOut (SYNTEST_CTAGSYN_port_28)
);

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_ONE_FF_CELL WpCell_0_Of_WpChain_0 (.
.CFI (DST_DATA_IN[3]),
.CFO (SYNTEST_CTAGSYN_port_0),
.CTI (SYNTEST_CTAGSYN_port_10),
.CTO (SYNTEST_CTAGSYN_port_14),
.shift (SYNTEST_CTAGSYN_port_1),
capture_clk (SYNTEST_CTAGSYN_port_2),
.apply (SYNTEST_CTAGSYN_port_3)
);

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_ONE_FF_CELL WpCell_1_Of_WpChain_0 (.
.CFI (DST_DATA_IN[2]),
.CFO (SYNTEST_CTAGSYN_port_4),
.CTI (SYNTEST_CTAGSYN_port_14),
.CTO (SYNTEST_CTAGSYN_port_15),
.shift (SYNTEST_CTAGSYN_port_1),
capture_clk (SYNTEST_CTAGSYN_port_2),
.apply (SYNTEST_CTAGSYN_port_3)
);

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_ONE_FF_CELL WpCell_2_Of_WpChain_0 (.
.CFI (DST_DATA_IN[1]),
.CFO (SYNTEST_CTAGSYN_port_5),
.CTI (SYNTEST_CTAGSYN_port_15),
.CTO (SYNTEST_CTAGSYN_port_16),
.shift (SYNTEST_CTAGSYN_port_1),
capture_clk (SYNTEST_CTAGSYN_port_2),
.apply (SYNTEST_CTAGSYN_port_3)
);

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_ONE_FF_CELL WpCell_3_Of_WpChain_0 (.
.CFI (DST_DATA_IN[0]),
.CFO (SYNTEST_CTAGSYN_port_6),
.CTI (SYNTEST_CTAGSYN_port_16),
.CTO (SYNTEST_CTAGSYN_port_18),
.shift (SYNTEST_CTAGSYN_port_1),
capture_clk (SYNTEST_CTAGSYN_port_2),
.apply (SYNTEST_CTAGSYN_port_3)
);

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_ONE_FF_CELL WpCell_4_Of_WpChain_0 (.
.CFI (SYNTEST_CTAGSYN_port_7),
.CFO (DST_DATA_OUT),
.CTI (SYNTEST_CTAGSYN_port_18),
.CTO (SYNTEST_CTAGSYN_port_17),
.shift (SYNTEST_CTAGSYN_port_1),
capture_clk (SYNTEST_CTAGSYN_port_2),
.apply (SYNTEST_CTAGSYN_port_3)
);

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_2to1_MUX ParallelMux_0 (.
.out (SYNTEST_CTAGSYN_port_10),
in0 (SYNTEST_CTAGSYN_port_11),
in1 (SYNTEST_CTAGSYN_port_12),
.select (SYNTEST_CTAGSYN_port_13)
);

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_2to1_MUX SiMux_0 (.
.out (SYNTEST_CTAGSYN_port_19),
in0 (SYNTEST_CTAGSYN_port_21),
in1 (SYNTEST_CTAGSYN_port_22),
.select (SYNTEST_CTAGSYN_port_20)
);

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_2to1_MUX SoMux_0 (.
.out (SYNTEST_CTAGSYN_port_19),
in0 (SYNTEST_CTAGSYN_port_21),
in1 (SYNTEST_CTAGSYN_port_22),
.select (SYNTEST_CTAGSYN_port_20)
);

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_ONE_FF_CELL WpCell_0_Of_WpChain_1 (.
.CFI (SI),
.CFO (SYNTEST_CTAGSYN_port_21),
.CTI (SYNTEST_CTAGSYN_port_24),
.CTO (SYNTEST_CTAGSYN_port_22),
.shift (SYNTEST_CTAGSYN_port_1),
capture_clk (SYNTEST_CTAGSYN_port_2),
.apply (SYNTEST_CTAGSYN_port_3)
);

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_2to1_MUX ParallelMux_1 (.
.out (SYNTEST_CTAGSYN_port_24),
in0 (SYNTEST_CTAGSYN_port_11),
in1 (SYNTEST_CTAGSYN_port_12),
.select (SYNTEST_CTAGSYN_port_13)
);

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_2to1_MUX SiMux_1 (.
.out (SYNTEST_CTAGSYN_port_27),
in0 (SYNTEST_CTAGSYN_port_22),
in1 (SYNTEST_CTAGSYN_port_25),
.select (SYNTEST_CTAGSYN_port_26)
);

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_2to1_MUX SoMux_1 (.
.out (SYNTEST_CTAGSYN_port_27),
in0 (SYNTEST_CTAGSYN_port_22),
in1 (SYNTEST_CTAGSYN_port_25),
.select (SYNTEST_CTAGSYN_port_26)
);
Ctag wrapped source module

*timescale 1 ns / 10 ps
module CtagWrappedModule_Of_HS_S27_SRC

(Clk,
SI,
SO,
SE,
T_N,
SRC_DATA_IN,
SRC_DATA_OUT,
TAKE_IT_TG,
GOT_IT_TG,
//WIP ports of CTAG
WRCK,
WRSTN,
UPDATEWR,
SHIFTWR,
CAPTUREWR,
SELECTWIR,
WP_SI,
WP_SO,
TAM_in,
TAM_out);
input Clk ;
input SI ;
output SO ;
input SE ;
input T_N;
input [3:0] SRC_DATA_IN;
output [3:0] SRC_DATA_OUT;
output TAKE_IT_TG;
input GOT_IT_TG;
input WRCK;
input WRSTN;
input UPDATEWR;
input SHIFTWR;
input CAPTUREWR;
input SELECTWIR;
input WP_SI;
output WP_SO;
output [1:0] TAM_in;
output [1:0] TAM_out;
// SYTest_CTAGSYN_ADDED V1.1.0 instance
HS_S27_SRC HS_S27_SRC (Clk (SYTest_CTAGSYN_port_35),
SI (SYTest_CTAGSYN_port_25),
SO (SYTest_CTAGSYN_port_31),
SE (SYTest_CTAGSYN_port_38),
T_N (SYTest_CTAGSYN_port_39),
SRC_DATA_IN ( SYTest_CTAGSYN_port_0 ),
SYTest_CTAGSYN_port_4,
SYTest_CTAGSYN_port_5,
SYTest_CTAGSYN_port_6),
SRC_DATA_OUT ( SYTest_CTAGSYN_port_12,
SYTest_CTAGSYN_port_11,
SYTest_CTAGSYN_port_10,
SYTest_CTAGSYN_port_7),
TAKE_IT_TG (TAKE_IT_TG),
GOT_IT_TG (GOT_IT_TG);
// SYTest_CTAGSYN_ADDED V1.1.0 instance
CtagCtrl_Of_HS_S27_SRC CtagCtrl_Of_HS_S27_SRC (WRCK (WRCK),
WRSTN (WRSTN),
UPDATEWR (UPDATEWR),
SHIFTWR (SHIFTWR),
CAPTUREWR (CAPTUREWR),
SELECTWIR (SELECTWIR),
WP_SI (WP_SI),
WP_SO (WP_SO),
// SYTest_CTAGSYN_ADDED V1.1.0 instance
STI_2to1_MUX STI_2to1_MUX (SYTest_CTAGSYN_port_30),
in0 (SYTest_CTAGSYN_port_30),
in1 (CLk),
select (SYTest_CTAGSYN_port_31); // SYTest_CTAGSYN_ADDED V1.1.0 instance

// SYTest_CTAGSYN_ADDED V1.1.0 instance
STI_2to1_MUX STI_2to1_MUX (SYTest_CTAGSYN_port_20),
in0 (SE),
in1 (SYTest_CTAGSYN_port_20),
select (SYTest_CTAGSYN_port_26);
endmodule // endmodule of CtagWrappedModule_Of_HS_S27_DST_NEW
Verilog Code

101

.TAM_in ( TAM_in[1] ,
TAM_in[0] ) ,
.TAM_out ( TAM_out[1] ,
TAM_out[0] ) ,
.Ctrl_ToWpShift (SYNTEST_CTAGSYN_port_1) ,
.Ctrl_ToWpCaptureclk (SYNTEST_CTAGSYN_port_2) ,
.Ctrl_ToOutWpCaptureclk (SYNTEST_CTAGSYN_port_8) ,
.Ctrl_ToWpApply (SYNTEST_CTAGSYN_port_3) ,
.Ctrl_ToOutWpApply (SYNTEST_CTAGSYN_port_9) ,
.Ctrl_ToWpParallel (SYNTEST_CTAGSYN_port_16) ,
.Ctrl_ToParallelIn0 (SYNTEST_CTAGSYN_port_15) ,
.Ctrl_ToParallelIn1 (SYNTEST_CTAGSYN_port_29) ,
.Ctrl_FromParallelOut0 (SYNTEST_CTAGSYN_port_23) ,
.scan_mode (SYNTEST_CTAGSYN_port_26) ,
.scan_shift_clk (SYNTEST_CTAGSYN_port_37) ,
.scan_test_mode (SYNTEST_CTAGSYN_port_36) ,
.test_mode (SYNTEST_CTAGSYN_port_40) ,
.scan_instruction (SYNTEST_CTAGSYN_port_32) ,
.Ctrl_ToSerialIn (SYNTEST_CTAGSYN_port_14) ,
.Ctrl_FromSerialOut (SYNTEST_CTAGSYN_port_34) ) ;

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_ONE_FF_CELL WpCell_0_Of_WpChain_0 (
.CFI (SRC_DATA_IN[3]) ,
.CFO (SYNTEST_CTAGSYN_port_0) ,
.CTI (SYNTEST_CTAGSYN_port_13) ,
.CTO (SYNTEST_CTAGSYN_port_17) ,
.shift (SYNTEST_CTAGSYN_port_1) ,
capture_clk (SYNTEST_CTAGSYN_port_2) ,
.apply (SYNTEST_CTAGSYN_port_3) ) ;

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_ONE_FF_CELL WpCell_1_Of_WpChain_0 (
.CFI (SRC_DATA_IN[2]) ,
.CFO (SYNTEST_CTAGSYN_port_4) ,
.CTI (SYNTEST_CTAGSYN_port_17) ,
.CTO (SYNTEST_CTAGSYN_port_18) ,
.shift (SYNTEST_CTAGSYN_port_1) ,
capture_clk (SYNTEST_CTAGSYN_port_2) ,
.apply (SYNTEST_CTAGSYN_port_3) ) ;

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_ONE_FF_CELL WpCell_2_Of_WpChain_0 (
.CFI (SRC_DATA_IN[1]) ,
.CFO (SYNTEST_CTAGSYN_port_5) ,
.CTI (SYNTEST_CTAGSYN_port_18) ,
.CTO (SYNTEST_CTAGSYN_port_19) ,
.shift (SYNTEST_CTAGSYN_port_1) ,
capture_clk (SYNTEST_CTAGSYN_port_2) ,
.apply (SYNTEST_CTAGSYN_port_3) ) ;

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_ONE_FF_CELL WpCell_3_Of_WpChain_0 (
.CFI (SRC_DATA_IN[0]) ,
.CFO (SYNTEST_CTAGSYN_port_6) ,
.CTI (SYNTEST_CTAGSYN_port_19) ,
.CTO (SYNTEST_CTAGSYN_port_20) ,
.shift (SYNTEST_CTAGSYN_port_1) ,
capture_clk (SYNTEST_CTAGSYN_port_2) ,
.apply (SYNTEST_CTAGSYN_port_3) ) ;

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_ONE_FF_CELL WpCell_4_Of_WpChain_0 (.
.CFI (SRC_DATA_OUT[0]) ,
.CFO (SYNTEST_CTAGSYN_port_7) ,
.CTI (SYNTEST_CTAGSYN_port_20) ,
.CTO (SYNTEST_CTAGSYN_port_21) ,
.capture_clk (SYNTEST_CTAGSYN_port_3) ,
.apply (SYNTEST_CTAGSYN_port_4) ) ;

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_ONE_FF_CELL WpCell_5_Of_WpChain_0 (.
.CFI (SRC_DATA_OUT[1]) ,
.CFO (SYNTEST_CTAGSYN_port_8) ,
.CTI (SYNTEST_CTAGSYN_port_21) ,
.CTO (SYNTEST_CTAGSYN_port_22) ,
.capture_clk (SYNTEST_CTAGSYN_port_3) ,
.apply (SYNTEST_CTAGSYN_port_4) ) ;

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_ONE_FF_CELL WpCell_6_Of_WpChain_0 (.
.CFI (SRC_DATA_OUT[2]) ,
.CFO (SYNTEST_CTAGSYN_port_9) ,
.CTI (SYNTEST_CTAGSYN_port_22) ,
.CTO (SYNTEST_CTAGSYN_port_23) ,
.capture_clk (SYNTEST_CTAGSYN_port_3) ,
.apply (SYNTEST_CTAGSYN_port_4) ) ;

// SYNTEST_CTAGSYN_ADDED V1.1.0 instance
STI_ONE_FF_CELL WpCell_7_Of_WpChain_0 (.
.CFI (SRC_DATA_OUT[3]) ,
module CtagCtrl_Of_HS_S27_DST_NEW(
    WRCK,
    WRSTN,
    UPDATEWR,
    SHIFTWR,
    CAPTUREWR,
    CFI0,
    CFO0,
    CFI1,
    CFO1,
    CFI2,
    CFO2,
    CFI3,
    CFO3,
    CFI4,
    CFO4,
    CFI5,
    CFO5,
    CFI6,
    CFO6,
    CFI7,
    CFO7,
    Clk,
    SE,
    T_N,
    WpCell_0_Of_WpChain_1,
    WpCell_1_Of_WpChain_1,
    WpCell_2_Of_WpChain_1,
    WpCell_3_Of_WpChain_1,
    WpCell_4_Of_WpChain_1,
    WpCell_5_Of_WpChain_1,
    WpCell_6_Of_WpChain_1,
    WpCell_7_Of_WpChain_1
    );

endmodule // endmodule of CtagWrappedModule_Of_HS_S27_SRC

CtagCtrl_Of_HS_S27_DST.v
APPENDIX B: Verilog Code

```
SELECTWIR,  
WP_SI, WP_SO,  
TAM_in,  
TAM_out,  
Ctrl_ToWpShift, Ctrl_ToInWpCaptureclk, Ctrl_ToOutWpCaptureclk, Ctrl_ToWpApply, Ctrl_ToParallel, Ctrl_ToParallelIn0, Ctrl_ToParallelIn1, Ctrl_FromParallelOut0, Ctrl_FromParallelOut1, scan_mode, scan_select_sys_clk, scan_shift_clk, scan_instruction, Ctrl_ToSerialIn, Ctrl_FromSerialOut ;  
input WRCK ;  
input WRSTN ;  
input UPDATEWR ;  
input SHIFTWIR ;  
input CAPTUREWIR ;  
input SELECTWIR ;  
input WP_SI;  
output WP_SO;  
input [1:0] TAM_in;  
output [1:0] TAM_out;  
output Ctrl_ToWpShift;  
output Ctrl_ToInWpCaptureclk; output Ctrl_ToOutWpCaptureclk; output Ctrl_ToWpApply; output Ctrl_ToParallel; output scan_mode; output scan_select_sys_clk; output scan_instruction; output Ctrl_ToSerialIn;  
input Ctrl_FromSerialOut;  
output Ctrl_ToParallelIn0;  
input Ctrl_FromParallelOut0;  
output Ctrl_ToParallelIn1;  
input Ctrl_FromParallelOut1;  
wire wir_so;  
wire bypass_en;  
wire from_bypass;  
wire from_wbr_or_user_reg; // a wire comes from wrapper boundary register // or user-defined register  
wire from_wdr; // a wire comes from wrapper data register(bypass register, // wrapper boundary register or user-defined register  
wire shift_en;  
wire in_capture_en; //capture enable of input cell  
wire out_capture_en; //capture enable of output cell  
wire in_apply_en;  
wire out_apply_en;  
wire parallel_en;  
Instruction_Decoder Instruction_Decoder(  
.shift_en(shift_en),  
in_capture_en(in_capture_en),  
out_capture_en(out_capture_en),  
in_apply_en(in_apply_en),  
out_apply_en(out_apply_en),  
bypass_en(bypass_en),  
.parallel_en(parallel_en),  
.scan_instruction(scan_instruction),  
.wrstn(WRSTN), .selectwir(SELECTWIR), .shiftwr(SHIFTWIR), .capturewir(CAPTUREWIR), .updatewr(UPDATEWR), .wrck(WRCK), .si(WP_SI), .so(wir_so));  
// "shift" control signal of wrapper cell  
assign Ctrl_ToWpShift = shift_en & SHIFTWIR & ( SELECTWIR) ;  
// "capture clock" of input wrapper cell  
assign Ctrl_ToInWpCaptureclk = (in_capture_en & (SHIFTWR | CAPTUREWIR)) & (SELECTWIR) & WRCK ;  
// "capture clock" of output wrapper cell  
assign Ctrl_ToOutWpCaptureclk = (out_capture_en & (SHIFTWR | CAPTUREWIR)) & (SELECTWIR) & WRCK ;  
// "apply" of input wrapper cell  
assign Ctrl_ToWpApply = in_apply_en & (SELECTWIR) ;  
// "apply" of output wrapper cell  
assign Ctrl_ToOutWpApply = out_apply_en & (SELECTWIR) ;  
// "parallel" signal of wrapper chain  
assign Ctrl_ToParallel = parallel_en ;  
assign scan_mode = scan_instruction & SHIFTWIR & (SELECTWIR) ;  
assign scan_shift_clk = scan_mode & WRCK ;  
assign scan_select_sys_clk = (scan_instruction) | (CAPTUREWIR) & (SELECTWIR)) ;  
assign Ctrl_ToSerialIn = WP_SI;  
assign Ctrl_ToParallelIn0 = TAM_in[0];  
assign Ctrl_ToParallelIn1 = TAM_in[1];  
```
assign TAM_out[1] = Ctrl_FromParallelOut1 ;
assign from_wbr_or_user_reg = Ctrl_FromSerialOut ;
assign from_wbr = (bypass_en) ? from_bypass : from_wbr_or_user_reg ;
assign WP_SO = (SELECTWIR) ? wir_so : from_wdr ;
endmodule // end of HS_S27_DST_NEW
module Instruction_Decoder(
    shift_en,
    in_capture_en,
    out_capture_en,
    in_apply_en,
    out_apply_en,
    bypass_en,
    parallel_en,
    scan_instruction,
    wrstn,
    selectwir,
    shiftwr,
    capturewr,
    updatewr,
    wrck,
    ai,
    so);
output shift_en;
output in_capture_en;
output out_capture_en;
output in_apply_en;
output out_apply_en;
output bypass_en;
output parallel_en;
output scan_instruction;
input wrstn;
input selectwir;
input shiftwr;
input capturewr;
input updatewr;
input wrck;
input ai;
output so;
wire [INSTRUCTION_LENGTH-1:0] instruction;
Instruction_Register Instruction_Register (.instruction(instruction),.so(so),.si(si),.wrstn (wrstn),.selectwir(selectwir), .shiftwr(shiftwr),
.updatewr(updatewr), .wrck(wrck));
always @(instruction or shiftwr or capturewr)
begin
case (instruction)
    WS_INTEST_SCAN:
        begin
            shift_en = 1'b1;
            in_capture_en = 1'b1;
            out_capture_en = 1'b1;
            in_apply_en = 1'b1;
            out_apply_en = 1'b0;
            bypass_en = 1'b0;
            parallel_en = 1'b0;
            scan_instruction = 1'b1;
        end
    WP_INTEST_SCAN:
        begin
            shift_en = 1'b1;
            in_capture_en = 1'b1;
            out_capture_en = 1'b1;
            in_apply_en = 1'b1;
            out_apply_en = 1'b0;
            bypass_en = 1'b0;
            parallel_en = 1'b0;
            scan_instruction = 1'b1;
        end
    WS_EXTEST:
        begin
            shift_en = 1'b1;
            in_capture_en = 1'b1;
            out_capture_en = 1'b1;
            in_apply_en = 1'b0;
            out_apply_en = 1'b1;
            bypass_en = 1'b0;
            parallel_en = 1'b1;
            scan_instruction = 1'b1;
        end
    WP_EXTEST:
        begin
            shift_en = 1'b1;
            in_capture_en = 1'b1;
            out_capture_en = 1'b1;
            in_apply_en = 1'b0;
            out_apply_en = 1'b1;
            bypass_en = 1'b0;
            parallel_en = 1'b1;
            scan_instruction = 1'b0;
        end
    end
endmodule // end of Instruction_Decoder

Appendix B. Verilog Code

module Instruction_Decoder (so, si, selectwir, wrstn, shiftwr, updatewr, wrck);
output [INSTRUCTION_LENGTH-1:0] instruction;
output so;
input si;
input wrstn;
input selectwir;
input shiftwr;
input updatewr;
input wrck;

always @(posedge wrck or negedge wrstn)
begin
if (wrstn)
begin
shift_stage_register = 0;
else
if (shiftwr & selectwir)
begin
shift_stage_register['INSTRUCTION_LENGTH-1:0] = shift_stage_register['INSTRUCTION_LENGTH-2:0],si;
end
assign instruction = (selectwir) ? update_stage_register : instruction;
end
endmodule // of Instruction_Register

module BYPASS_REG (in, out, clk);
input in;
output out;
input clk;
reg out;
always @(posedge clk)
begin
out = in;
end
endmodule // end of BYPASS_REG

CtagCtrl_Of_HS_S27_SRC.v

/** \* DESCRIPTION : P1500 controller netlist file \*/
/** \* Create by : CTagSyn V1.1.0 \*/
/** \* \*/

module CtagCtrl_Of_HS_S27_SRC (WRCK, WRSTN, SHIFTWR, CAPTUREWR, SELECTWR, WP_SI, WP_SO, TAM_in, TAM_out, Ctrl_ToWpShift, Ctrl_ToInWpCaptureclk, Ctrl_ToOutWpCaptureclk, Ctrl_ToInWpApply,
Appendix B. Verilog Code

Chapter B Verilog Code

module Instruction_Decoder(
    .shift_en(shift_en),
    .in_capture_en(in_capture_en),
    .out_capture_en(out_capture_en),
    .in_apply_en(in_apply_en),
    .out_apply_en(out_apply_en),
    .bypass_en(bypass_en),
    .parallel_en(parallel_en),
    .scan_instruction(scan_instruction),
    .wrck(WRCK), .selectwir(SELECTWIR), .shiftwr(SHIFTWR), .capturewr(CAPTUREWR), .updatewr (UPDATEWR), .wrstn(WRSTN), .selectwir(SELECTWIR), .shiftwr(SHIFTWR), .capturewr(CAPTUREWR), .updatewr (UPDATEWR), .wrck(WRCK), .si(WP_SI), .so(wir_so));

// "shift" control signal of wrapper cell
assign Ctrl_ToWpShift = shift_en & SHIFTWR & ( SELECTWIR) ;
// "capture clock" of output wrapper cell
assign Ctrl_ToOutWpCaptureclk = (in_capture_en) & (SHIFTWR | CAPTUREWR) & ( SELECTWIR) ;
// "apply" of output wrapper cell
assign Ctrl_ToOutWpApply = out_apply_en & ( SELECTWIR) ;
// "parallel" signal of wrapper chain
assign Ctrl_ToWpParallel = parallel_en ;
assign scan_mode = scan_instruction & & (SELECTWIR) ;
assign scan_shift_clk = scan_mode & WRCK ;
assign test_mode = scan_instruction & ( SELECTWIR) ;
assign from_bypass = Ctrl_FromSerialOut ;
assign from_parallel_in = Ctrl_FromParallelIn ;
assign from_wdr = from_wdr ;
assign from_wbr_or_user_reg = from_wbr_or_user_reg ;
assign WP_SO = (SELECTWIR) ? wir_so : from_wdr ;
endmodule // end of HS_S27_SRC
module Instruction_Decoder(

Ctrl_ToOutWpApply,
Ctrl_ToWpParallel,
Ctrl_ToParallelIn0,
Ctrl_FromParallelOut0,
Ctrl_ToParallelIn1,
Ctrl_FromParallelOut1,
scan_mode,
scan_select_sys_clk,
scan_shift_clk,
test_mode,
scan_instruction,
Ctrl_ToSerialIn,
Ctrl_FromSerialOut) ;
input WRCK ;
input WRSTN ;
input UPDATEWR ;
input SHIFTWR ;
input CAPTUREWR ;
input SELECTWIR ;
input WP_SI ;
output WP_SO ;
input [1:0] TAM_in ;
output [1:0] TAM_out ;
output Ctrl_ToWpShift ;
output Ctrl_ToOutWpCaptureclk ;
output Ctrl_ToOutWpApply ;
output Ctrl_ToWpParallel ;
output scan_mode ;
output scan_instruction ;
output Ctrl_ToParallelIn0 ;
input Ctrl_FromParallelOut0 ;
output Ctrl_ToParallelIn1 ;
input Ctrl_FromParallelOut1 ;
wire wir_so ;
wire bypass_en ;
wire from_bypass ;
wire from_wbr_or_user_reg ; // a wire comes from wrapper boundary register // or user-defined register
wire from_wdr ; // a wire comes from wrapper data register(bypass register, // wrapper boundary register or user-defined register
wire shift_en ;
wire in_capture_en ; //capture enable of input cell
wire out_capture_en ; //capture enable of output cell
wire in_apply_en ;
wire out_apply_en ;
wire parallel_en ;
Instruction_Decoder Instruction_Decoder(
    .shift_en(shift_en),
    .in_capture_en(in_capture_en),
    .out_capture_en(out_capture_en),
    .in_apply_en(in_apply_en),
    .out_apply_en(out_apply_en),
    .bypass_en(bypass_en),
    .parallel_en(parallel_en),
    .scan_instruction(scan_instruction),
    .wrck(WRCK), .selectwir(SELECTWIR), .shiftwr(SHIFTWR), .capturewr(CAPTUREWR), .updatewr (UPDATEWR), .wrck(WRCK), .si(WP_SI), .so(wir_so));
Appendix B. Verilog Code

```verilog
shift_en,
in_capture_en,
out_capture_en,
in_apply_en,
out_apply_en,
bypass_en,
parallel_en,
scan_instruction,
wrstn,
selectwir,
shiftwr,
capturewr,
updatewr,
wrck,
si;
output shift_en ;reg shift_en ;
output in_capture_en ;reg in_capture_en ;
output out_capture_en ;reg out_capture_en ;
output in_apply_en ;reg in_apply_en ;
output out_apply_en ;reg out_apply_en ;
output bypass_en ;reg bypass_en ;
output parallel_en ;reg parallel_en ;
output scan_instruction ;reg scan_instruction ;
input wrstn ;
input selectwir ;
input shiftwr ;
input capturewr ;
input updatewr ;
input wrck ;
input si ;
output so ;
wire [\INSTRUCTION_LENGTH-1:0] instruction ;
Instruction_Register Instruction_Register (.instruction(instruction),.so(so),.si(si),.wrstn (wrstn),.selectwir(selectwir), .shiftwr(shiftwr), .updatewr(updatewr), .wrck(wrck));
always @(instruction or shiftwr or capturewr)
begin
  case (instruction)
  'WS_INTEST_SCAN :
  begin
    shift_en = 1'b1 ;
in_capture_en = 1'b1 ;
out_capture_en = 1'b1 ;
in_apply_en = 1'b1 ;
out_apply_en = 1'b0 ;
bypass_en = 1'b0 ;
parallel_en = 1'b0 ;
scan_instruction = 1'b1 ;
  end
  'WP_INTEST_SCAN :
  begin
    shift_en = 1'b1 ;
in_capture_en = 1'b1 ;
out_capture_en = 1'b1 ;
in_apply_en = 1'b1 ;
out_apply_en = 1'b0 ;
bypass_en = 1'b0 ;
parallel_en = 1'b0 ;
scan_instruction = 1'b1 ;
  end
  'WS_EXTEST :
  begin
    shift_en = 1'b1 ;
in_capture_en = 1'b1 ;
out_capture_en = 1'b1 ;
in_apply_en = 1'b0 ;
out_apply_en = 1'b1 ;
bypass_en = 1'b0 ;
parallel_en = 1'b0 ;
scan_instruction = 1'b1 ;
  end
  'WP_EXTEST :
  begin
    shift_en = 1'b1 ;
in_capture_en = 1'b1 ;
out_capture_en = 1'b1 ;
in_apply_en = 1'b0 ;
out_apply_en = 1'b1 ;
bypass_en = 1'b0 ;
parallel_en = 1'b0 ;
scan_instruction = 1'b0 ;
  end
  'WS_BYPASS :
  begin
    shift_en = 1'b0 ;
in_capture_en = 1'b0 ;
out_capture_en = 1'b0 ;
in_apply_en = 1'b0 ;
out_apply_en = 1'b0 ;
bypass_en = 1'b0 ;
parallel_en = 1'b0 ;
scan_instruction = 1'b0 ;
  end
  end
end
```
out_apply_en = 1'b0;
bypass_en = 1'b1;
parallel_en = 1'b0;
scan_instruction = 1'b0;
cend
default:
begin
    shift_en = 1'b0;
in_capture_en = 1'b0;
out_capture_en = 1'b0;
in_apply_en = 1'b0;
out_apply_en = 1'b0;
bypass_en = 1'b1;
parallel_en = 1'b0;
scan_instruction = 1'b0;
cend
cendcase
cend
cndmodule //of Instruction_Decoder
module Instruction_Register(instruction, so, si, selectwir, wrstn, shiftwr, updatewr, wrck);
output ["INSTRUCTION_LENGTH-1:0"] instruction;
output so;
input si;
input wrstn;
input selectwir;
input shiftwr;
input updatewr;
input wrck;
reg ["INSTRUCTION_LENGTH-1:0"] shift_stage_register;
reg ["INSTRUCTION_LENGTH-1:0"] update_stage_register;
assign so = shift_stage_register["INSTRUCTION_LENGTH-1"]; always @(posedge wrck or negedge wrstn) begin
    if (wrstn)
        shift_stage_register = 0;
    else
        if (shiftwr & selectwir)
            shift_stage_register["INSTRUCTION_LENGTH-2:0"],si = shift_stage_register["INSTRUCTION_LENGTH-2:0"],si;
end
assign instruction = (selectwir) ? update_stage_register : instruction;
always @(negedge wrck or negedge wrstn)
begin
    if (wrstn)
        update_stage_register = 0;
    else
        if (updatewr)
            update_stage_register = shift_stage_register;
end
cndmodule //of Instruction_Register
module BYPASS_REG (in, out, clk);
input in;
output out;
input clk;
reg out;
always @(posedge clk)
begin
    out = in;
end
cndmodule //end of BYPASS_REG

HS_FINAL.v
	//timescale 1ns / 1ps
	////////////////////////////////////////////////////////////////////////////////////
	// Module Name: HS_FINAL_NEW
	////////////////////////////////////////////////////////////////////////////////////
module HS_FINAL_NEW(input T_N,
    input SI1,
    input SI2,
    input SE,
    output SO1,
    output SO2,
    input PTO,
    output OTP,
    output Q, //port wasted
    input RST,
    input CLOCK1,
    input CLOCK2,
    input SCK,
    input [3:0] PI,
    output PO,
    input DOMAIN_SEL,
    input FAULT_SEL);
    wire take_it_tg_sig,got_it_tg_sig,TCK1,TCK2,tck1_sig,tck2_sig,int_out_sig,pto_out_gated;
    wire temp,temp_clk,temp4,mux_en,mux_out,mux_en,so_out_sig,dual_q;
    wire [3:0] src_dst_data;
defparam MUX1.wait_delay = 0.1;
Mux MUX1 (.in_0(CLOCK1),
    .in_1(tck1_sig),
    .out(T_N),
    .en(0));
Mux MUX2 (input [3:0] PI,
    .out(CLOCK2),
    .en(1));
defparam MUX2.wait_delay = 0.1;
TEST_CLK_GEN TEST_CLK_GEN1( .CLOCK1(CLOCK1),
    .CLOCK2(CLOCK2),
    .SCK(SCK),
    .RST(RST),
    .SR(SE),
    .DOMAIN_SEL(DOMAIN_SEL),
    .FAULT_SEL(FAULT_SEL),
    .TCK1(tck1_sig),
    .TCK2(tck2_sig)
);

HS_S27_SRC HS_S27_SRC1( .Clk(TCK1),
    .SI(SI1),
    .SO(SO1),
    .SE(SE),
    .T_N(T_N),
    .SRC_DATA_IN(P0),
    .SRC_DATA_OUT(src_dst_data),
    .TAKE_IT_TG(take_it_tg_sig),
    .GOT_IT_TG(got_it_tg_sig)
);

HS_S27_DST_NEW HS_S27_DST_NEW1( .CLK(TCK2),
    .SI(SI2),
    .SO(SO2),
    .SE(SE),
    .DST_DATA_IN(src_dst_data),
    .DST_DATA_OUT(PO),
    .TAKE_IT_TG(take_it_tg_sig),
    .GOT_IT_TG(got_it_tg_sig),
    .RST(RST),
    .PTO(PTO),
    .OTP(OTP) );
module SRC_DATA;
  input [2:0] SRC_DATA[3];

endmodule

module SRC_DATA1;

  input [2:0] SRC_DATA);

  output [3:0] SRC_DATA_OUT;

  input SI, SO, SE, Clk;

  input FETCH_PL;

endmodule

module HS_SRC_CTRL1;

  input START, SI, SO, SE, Clk;

  input GOT_IT_TG, GOT_IT_PL;

  input TAKE_IT_TG;

  input FETCH_PL;

endmodule

module START_GEN1;

  input DATA_READY, HS_DONE, SI, SO, SE, Clk, START;

endmodule

module HS_DONE_GEN1;

  input GOT_IT_PL, HS_DONE, SI, SO, SE, Clk, START;

endmodule

module EXE_DATA_READY_GEN1;

  input DATA_READY, EXECUTE, SI, SO, SE, Clk, START;

endmodule

module HS_S27_DST;

  reg [2:0] temp1, temp2, temp3, temp4, temp5, temp6;

  wire [3:0] SRC_DATA_OUT;

  wire ClkB, Clk_to_Exec;

  defparam MUX2.wait_delay = 0;

  MUX MUX2 (.in_0(hs_done_sig), .in_1(0), .sel(T_N), .out1(hs_done_test));

  defparam INV1.wait_delay = 0.1;

  INV_G INV1 (.in_1(Clk), .out1(CLKB));

  defparam MUX3.wait_delay = 0.1;

  MUX MUX3 (.in_0(CLKB), .in_1(Clk), .sel(T_N), .out1(Clk_to_Exec));

  EXE_DATA READY GEN1 EXE_DATA READY GEN1 (.DATA READY (data_ready_sig), .EXECUTE (execute_sig), .SI (SI), .SO (temp), .SE (SE), .Clk (Clk_to_Exec), .START (start_sig));

  assign SO = temp6;

cendmodule
module HS_S27_DST_NEW(
    input TAKE_IT_TG,
    output GOT_IT_TG,
    input CLK,
    input SI,
    input SE,
    input RST,
    input PTO,
    output OTP,
    output SO,
    input [3:0] DST_DATA_IN,
    output DST_DATA_OUT
);  
wire [3:0] data;
wire take_it_pl,temp,temp1,temp2,temp3,int_out_sig,pto_out_sig,temp_clk,dual_q,mux_en;
DST_DATA DST_DATA1( .INP(DST_DATA_IN),
    .OUTP(data),
    .SI(temp),
    .SE(SE),
    .SO(temp1),
    .Clk(CLK),
    .TAKE_IT_PL(take_it_pl));

HS_DST_CTRL HS_DST_CTRL1(
    .TAKE_IT_TG(TAKE_IT_TG),
    .TAKE_IT_PL(take_it_pl),
    .GOT_IT_TG(GOT_IT_TG),
    .Clk(CLK),
    .SI(SI),
    .SE(SE),
    .SO(temp),
    .INT_OUT(int_out_sig),
    .PTO_OUT(pto_out_sig)
);
S27_DST S27_DST1(
    .G0(data[0]),
    .G1(data[1]),
    .G2(data[2]),
    .G3(data[3]),
    .Clk(CLK),
    .SI(temp1),
    .SE(SE),
    .SO(temp2),
    .G17(DST_DATA_OUT)
);
defparam MUX3.wait_delay = 0.1;
MUX MUX3 (  
    .in_0(int_out_sig),
    .in_1(PTE),
    .sel(PTO),
    .out1(temp3));
defparam AND1.wait_delay = 0;
AND_G AND1 (  
    .in_1(PTO),
    .in_2((SE)),
    .out1(mux_en));
defparam MUX4.wait_delay = 0.1;
MUX MUX4 (  
    .in_0(CLK),
    .in_1(pto_out_sig),
    .sel(mux_en),
    .out1(temp_clk));
//Three additional DFF's for testing few faults
SCAN_CHAIN3 SCAN_CHAIN3(
    .D(temp4),
    .SI(temp2),
    .SE(SE),
    .SO(SO),
    .CLK(temp_clk),
    .Q(Q)); //port wasted
defparam DUAL_DFF1.wait_delay = 0.1;
DUAL_DFF DUAL_DFF1 (  
    .D(dual_q),
    .Q(dual_q),
    .Clk((RST & (SE)))); // Check if RST is required
assign OTP = dual_q;
endmodule

HS_DST_CTRL.v

timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Module Name: HS_DST_CTRL
////////////////////////////////////////////////////////////////////////////////
module HS_DST_CTRL(
    input TAKE_IT_TG,
    output TAKE_IT_PL,
    
    output GOT_IT_TG,
    output PTO,
    output OTP,
    output SO,
output GOT_IT_TG,
input Clk,
input SI,
input SE,
output SO,
output INT_OUT,
output PTO_OUT
);
wire INTERP;
wire take_it_pl_temp,temp,temp1,sync_out_pl;
SYNC SYNCHRONIZER1 (  
.sync_in(TAKE_IT_TG),
sync_out(INTERP),
.Clk(Clk) );
/////////////////////////////////////////////////////////////////////////////
//defparam  
// buf1.wait_delay = 491.7;
//BUFFER buf1 (  
//.in_1( (INTERP)),  
//.in_1(INTERP), ///for pulses
// .out_1(sync_out_pl) );  
 //////////////////////////////////////////////////////////////////////////
//Pulse Gen broken up  
defparam  
SDFF1.wait_delay = 0;
SDFF SDFF1 (  
.D(INTERP), ///normal operation
//.D(sync_out_pl), ///for generating pulses at the output of sync
.SI(SI),
.SE(SE),
.Clk(Clk),
.Qout(temp));
defparam  
XOR1.wait_delay = 0;
XOR_G XOR1(  
.in_1(INTERP), /// normal operation
//.in_1(sync_out_pl), /// for generating pulses at the output of sync
.in_2(temp),
.out1(take_it_pl_temp) );
////////////////////////////////////
TOGGLE_GEN TOGGLE_GEN1(  
.PIN(take_it_pl_temp),  
.SI(temp),
.SE(SE),
.SO(SO),
.Clk(Clk),
.POUT(GOT_IT_TG) );
assign TAKE_IT_PL = take_it_pl_temp;
assign INT_OUT = temp;
assign PTO_OUT = INTERP; //normal mode
//assign PTO_OUT = sync_out_pl; // for pulses
endmodule

DST_DATA.v
/*timescale 1ns / 1ps */
/////////////////////////////////////////////////////////////////////////////
// Module Name: DST_DATA
/////////////////////////////////////////////////////////////////////////////
module DST_DATA(  
input [3:0] INP,  
output [3:0] OUTP,  
input SI,  
output SO,  
input SE,  
input Clk,  
input TAKE_IT_PL  
);
wire tempout0,tempout1,tempout2,tempout3,temp0,temp1,temp2,temp3;
// first data  
defparam MUX0.wait_delay = 0;
MUX MUX0(  
.in_0(tempout0),
.in_1(INP[0]),
.sel(TAKE_IT_PL),
.out1(temp0) );
defparam SDF0.wait_delay = 0;
SDFF SDF0 (  
.D(temp0),
.SI(SI),
.SE(SE),
.Clk(Clk),
.Qout(tempout0) );
// second data  
defparam MUX1.wait_delay = 0;
MUX MUX1(  
.in_0(tempout1),

Appendix B. Verilog Code

Verilog Code

```verilog
// Module Name: S27_DST
module S27_DST(
    input G0,
    input G1,
    input G2,
    input G3,
    input Clk,
    input SI,
    input SE,
    output SO,
    output G17
);
wire G5, G10, G6, G11, G7, G13, G14, G8, G15, G16, G9;
assign SO = G7;
INV_G NOT_0 ( .in_1(G0), .out1(G14) );
INV_G NOT_1 ( .in_1(G11), .out1(G17) );
AND_G AND2_0 ( .in_1(G14), .in_2(G6), .out1(G8) );
OR_G OR2_0 ( .in_1(G12), .in_2(G8), .out1(G15) );
OR_G OR2_1 ( .in_1(G3), .in_2(G8), .out1(G16) );
NAND_G NAND2_0 ( .in_1(G16), .in_2(G15), .out1(G9) );
NOR_G NOR2_0 ( .in_1(G14), .in_2(G8), .out1(G10) );
NOR_G NOR2_1 ( .in_1(G5), .in_2(G9), .out1(G11) );
NOR_G NOR2_2 ( .in_1(G1), .in_2(G7), .out1(G12) );
NOR_G NOR2_3 ( .in_1(G2), .in_2(G12), .out1(G13) );
SDFF DFF_2 ( .SI(G6), .SE(SE), .D(G15), .Clk(Clk), .Qout(G7) );
SDFF DFF_1 ( .SI(G5), .SE(SE), .D(G10), .Clk(Clk), .Qout(G6) );
SDFF DFF_0 ( .SI(SI), .SE(SE), .D(G11), .Clk(Clk), .Qout(G5) );
endmodule
```

```verilog
// Module Name: SCAN_CHAIN3
module SCAN_CHAIN3(
    input D,
    input G5, G10, G6, G11, G7, G13, G14, G8, G15, G16, G9;
assign SO = G7;
INV_G NOT_0 ( .in_1(G0), .out1(G14) );
INV_G NOT_1 ( .in_1(G11), .out1(G17) );
AND_G AND2_0 ( .in_1(G14), .in_2(G6), .out1(G8) );
OR_G OR2_0 ( .in_1(G12), .in_2(G8), .out1(G15) );
OR_G OR2_1 ( .in_1(G3), .in_2(G8), .out1(G16) );
NAND_G NAND2_0 ( .in_1(G16), .in_2(G15), .out1(G9) );
NOR_G NOR2_0 ( .in_1(G14), .in_2(G8), .out1(G10) );
NOR_G NOR2_1 ( .in_1(G5), .in_2(G9), .out1(G11) );
NOR_G NOR2_2 ( .in_1(G1), .in_2(G7), .out1(G12) );
NOR_G NOR2_3 ( .in_1(G2), .in_2(G12), .out1(G13) );
SDFF DFF_2 ( .SI(G6), .SE(SE), .D(G13), .Clk(Clk), .Qout(G7) );
SDFF DFF_1 ( .SI(G5), .SE(SE), .D(G11), .Clk(Clk), .Qout(G6) );
SDFF DFF_0 ( .SI(SI), .SE(SE), .D(G10), .Clk(Clk), .Qout(G5) );
```
Appendix B. Verilog Code

Chapter B  Verilog Code

src_input_ctrl.v

*timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////////////////////
// Module Name: SRC_INPUT_CTRL
/////////////////////////////////////////////////////////////////////////////////

module SRC_INPUT_CTRL(
  input [3:0] INP,
  output [3:0] OUTP,
  input SI,
  output SO,
  input SE,
  input Clk
);

defparam SDFF1.wait_delay = 0;
SDFF SDFF1 (D(INP[0]), SI(SI), SE(SE), Clk(Clk), Qout(OUTP[0]));
defparam SDFF2.wait_delay = 0;
SDFF SDFF2 (D(INP[1]), SI(OUTP[0]), SE(SE), Clk(Clk), Qout(OUTP[1]));
defparam SDFF3.wait_delay = 0;
SDFF SDFF3 (D(INP[2]), SI(OUTP[1]), SE(SE), Clk(Clk), Qout(OUTP[2]));
defparam SDFF4.wait_delay = 0;
SDFF SDFF4 (D(INP[3]), SI(OUTP[2]), SE(SE), Clk(Clk), Qout(OUTP[3]));
assign SO = OUTP[3];
endmodule

s27_src.v

*timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////////////////////
// Module Name: S27_SRC
/////////////////////////////////////////////////////////////////////////////////

module S27_SRC(
  input G0,
  input G1,
  input G2,
  input G3,
Verilog Code

```verilog
input Clk,
output G17,
output G16,
output G10,
output G13,
input SI,
input SE,
output SO
);
wire G5, G6, G11, G7, G14, G8, G15, G12, G9;
assign SO = G5;
INV_G NOT_0 ( .in_1(G0), .out1(G14) );
INV_G NOT_1 ( .in_1(G11), .out1(G17) );
AND_G AND2_0 ( .in_1(G14), .in_2(G6), .out1(G8) );
OR_G OR2_0 ( .in_1(G12), .in_2(G8), .out1(G15) );
OR_G OR2_1 ( .in_1(G3), .in_2(G8), .out1(G16) );
NAND_G NAND2_0 ( .in_1(G16), .in_2(G15), .out1(G9) );
NOR_G NOR2_0 ( .in_1(G14), .in_2(G11), .out1(G10) );
NOR_G NOR2_1 ( .in_1(G5), .in_2(G9), .out1(G11) );
NOR_G NOR2_2 ( .in_1(G1), .in_2(G7), .out1(G12) );
NOR_G NOR2_3 ( .in_1(G2), .in_2(G12), .out1(G13) );
SDFF DFF_2 ( .SI(SI), .SE(SE), .D(G13), .Clk(Clk), .Qout(G7) );
SDFF DFF_1 ( .SI(G7), .SE(SE), .D(G11), .Clk(Clk), .Qout(G6) );
SDFF DFF_0 ( .SI(G6), .SE(SE), .D(G10), .Clk(Clk), .Qout(G5) );
endmodule
```

`src_data.v`

```verilog
		 timescale 1ns / 1ps
		 // Module Name: SRC_DATA
		 module SRC_DATA( input [3:0] INP,
		 output [3:0] OUTP,
		 input SI,
		 output SO,
		 input SE,
		 input Clk,
		 input FETCH_PL)
		 );
		 wire tempout0,tempout1,tempout2,tempout3,temp0,temp1,temp2,temp3;
		 // first data
defparam MUX0.wait_delay = 0;
		 MUX MUX0( .in_0(tempout0),
		 .in_1(INP[0]),
		 .sel(FETCH_PL),
		 .out1(temp0) );
defparam SDFF0.wait_delay = 0;
		 SDFF SDFF0 ( .D(temp0),
		 .SI(tempout1),
		 .SE(SE),
		 .Clk(Clk),
		 .Qout(tempout0));
		 //second data
defparam MUX1.wait_delay = 0;
		 MUX MUX1( .in_0(tempout1),
		 .in_1(INP[1]),
		 .sel(FETCH_PL),
		 .out1(temp1) );
defparam SDFF1.wait_delay = 0;
		 SDFF SDFF1 ( .D(temp1),
		 .SI(tempout2),
		 .SE(SE),
		 .Clk(Clk),
		 .Qout(tempout1));
		 //third data
defparam MUX2.wait_delay = 0;
		 MUX MUX2( .in_0(tempout2),
		 .in_1(INP[2]),
		 .sel(FETCH_PL),
		 .out1(temp2) );
defparam SDFF2.wait_delay = 0;
		 SDFF SDFF2 ( .D(temp2),
		 .SI(tempout3),
		 .SE(SE),
		 .Clk(Clk),
		 .Qout(tempout2));
		 //fourth data
defparam MUX3.wait_delay = 0;
		 MUX MUX3(
```
Verilog Code

Appendix B

Verilog Code

`in_0(tempout3),
in_1(INP[3]),
.sel(FETCH_PL),
.out1(temp3));
defparam SDFF3.wait_delay = 0;
SDFF SDFF3 (D(temp3),
SI(SI),
SE(SE),
.Clk(Clk),
.Qout(tempout3));
assign SO = tempout0;
assign OUTP[0] = tempout0;
assign OUTP[1] = tempout1;
assign OUTP[2] = tempout2;
assign OUTP[3] = tempout3;
endmodule

hs_src_ctrl.v

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////////////////////
// Module Name: HS_SRC_CTRL
/////////////////////////////////////////////////////////////////////////////////
module HS_SRC_CTRL(
input START,
input SI ,
input SE,
input Clk,
output SO,
input GOT_IT_TG,
output GOT_IT_PL,
output TAKE_IT_TG,
output FETCH_PL
);
wire [0:2] INTERP;
wire fetch_pl_temp;
wire abc, temp2, temp, sdf
ff_out;
SYNC SYNCHRONIZER1 (sync_in(GOT_IT_TG),
.sync_out(INTERP[0]),
.Clk(Clk));
//PULSE_GEN PULSE_GEN1 ( .in_1(INTERP[0]),
// .SI(temp),
// .SE(SE),
// .SO(abc),
// .Clk(Clk),
// .out1(GOT_IT_PL) );
RISING_EDGE_DET RISING_DET1 (in_1(START),
SI(SI),
SE(SE),
SO(temp),
.Clk(Clk),
.out1(fetch_pl_temp));
/////////////////////////////////////////////////////////////////////////////////
//———Test Mode———
defparam DFF1.wait_delay = 0;
DFF DFF1 (D(temp),
.Clk(Clk),
.Q(sdf_out)); //PULSE_GEN broken up
/////////////////////////////////////////////////////////////////////////////////

TOGGLE_GEN TOGGLE_GEN1(PIN(fetch_pl_temp),
.SI(sdf_out),
.SE(SE),
.SO(temp2),
.Clk(Clk),
.POUT(TAKE_IT_TG));
assign FETCH_PL = fetch_pl_temp;
assign SO = temp2;
endmodule

start_gen.v

'define false 1'b 0 'define FALSE 1'b 0 'define true 1'b 1 'define TRUE 1'b 1
	timescale 1 ns / 1 ns // timescale for following modules

module START_GEN (DATA_READY, HS_DONE, SI, SO, SE, Clk, START);
input DATA_READY;
input HS_DONE;
input SI;
output SO;
input SE;
output Clk;
output START;
//reg SO; //reg START;
wire Q_HS_DONE;
wire SEND_READY;
wire Q_SEND_READY;
wire Q_Q_SEND_READY;
wire B_Q_SEND_READY;
wire temp;
//initial
// begin : process_6
// temp = 1'b 0;
// end
assign SO = Q_Q_SEND_READY;
defparam U1.wait_delay = 0;
SDFF U1 (.D(SEND_READY),.SI(SI),//.SO(temp),.SE(SE),.Clk(Clk),.Qout(Q_SEND_READY));
defparam U2.wait_delay = 0;
AND_G U2 (.in_1(HS_DONE),.in_2(DATA_READY),.out1(SEND_READY));
defparam U3.wait_delay = 0;
SDFF U3 (.D(Q_SEND_READY),.SI(Q_SEND_READY),//.SO(SO),.SE(SE),.Clk(Clk),.Qout(Q_Q_SEND_READY));
defparam U4.wait_delay = 0;
INV_G U4 (.in_1(Q_Q_SEND_READY),.out1(B_Q_SEND_READY));
defparam U5.wait_delay = 0;
AND_G U5 (.in_1(Q_SEND_READY),.in_2(B_Q_SEND READY),.out1(START));
endmodule // module START_GEN

hs_done_gen.v

	timescale 1ns / 1ps

module HS_DONE_GEN (START, GOT_IT_PL, SI, SE, Clk, SO, HS_DONE);
wire MUX_SEL;
wire Qsig;
wire Qb;
wire Dsig;
defparam XOR1.wait_delay = 0;
XOR_G XOR1(.in_1(START),
...
exec_data_ready_gen.v

defparam SDFFC1.wait_delay = 0.1;
SDFFC SDFFC1 (
    D(0), ///should be 1 in normal mode .SI(SI),
    .SE(SE), .Clk(Clk),
    Qout(XOR_IN_0));
defparam SDFFC2.wait_delay = 0.1;
SDFFC SDFFC2 (
    D(XOR_IN_0), .SI(XOR_IN_0),
    .SE(SE), .Clk(Clk),
    Qout(Q_XOR_IN_0));
defparam SDFFC3.wait_delay = 0.1;
SDFFC SDFFC3 (
    D(Q_XOR_IN_0), .SI(Q_XOR_IN_0),
    .SE(SE), .Clk(Clk),
    Qout(Q_XOR_IN_1));
defparam XOR1.wait_delay = 0.1;
XOR_G XOR1 (
    .in_1(XOR_IN_0),
    .in_2(XOR_IN_1), .out1(EXECUTE_BUF));
defparam INV2.wait_delay = 0.1;
INV_G INV2 (
    .in_1(EXECUTE_BUF), .out1(DATA_READY_BUF));
assign EXECUTE = EXECUTE_BUF;
assign DATA_READY = DATA READY_BUF;
assign SO = XOR_IN_1;
cendmodule

synchronizer.v

defparam SDFF1.wait_delay = 0;
SDFF SDFF1 (
    .D(Dsig), .SI(SI), .SE(SE),
    .Clk(Clk),
    Qout(Qsig));
defparam INV1.wait_delay = 0;
INV_G INV1 (
    .in_1(Qsig),
    .out1(Qb));
assign HS_DONE = Qb;
assign SO = Qsig;
cendmodule
module SYNC(
    input sync_in,
    input Clk,
    output sync_out
);
wire temp;
defparam DFF1.wait_delay = 0;
DFF DFF1 (.D(sync_in), .Clk(Clk), .Q(temp));
defparam DFF2.wait_delay = 0;
DFF DFF2 (.D(temp), .Clk(Clk), .Q(sync_out));
endmodule

rising_edge_detector.v
	timescale 1ns / 1ps

// Module Name: RISING_EDGE_DET

module RISING_EDGE_DET(
    input in_1,
    input SI,
    input SE,
    input Clk,
    output out1,
    output SO
);
wire temp,temp1;
defparam SDFF1.wait_delay = 0;
SDFF SDFF1 (.D(in_1), .SI(SI), .SO(temp), .SE(SE), .Clk(Clk), .Qout(temp));
defparam INV1.wait_delay = 0;
INV_G INV1 (.in_1(temp), .out1(temp1) );
defparam AND1.wait_delay = 0;
AND_G AND1 (.in_1(temp1), .in_2(in_1), .out1(out1) );
assign SO = temp;
endmodule
dual_dff.v
	timescale 1ns / 1ps

// Module Name: DUAL_DFF

module DUAL_DFF(
    D,
    Q,
    Clk,
    Clr
);
parameter wait_delay = 0;
input D;
input Clk;
input Clr;
output Q;
reg Q;
initial
begin : process_3
Q = 0;
end
always @(posedge Clk or posedge Clr)
begin : process_1
if (Clr == 1'b 1)
begin
Q <= #(wait_delay) 1'b 0;
end
else
begin
Q <= #(wait_delay) D;
end
endalways @(negedge Clk or posedge Clr)
begin : process_2

if (Clr === 1'b 1)
begin
    Q <= #(wait_delay) 1'b 0;
end
else
begin
    Q <= #(wait_delay) D;
end
endmodule
C.1 Input files for s27 receiver core

.cfg file

```plaintext
%GLOBAL_SIGNAL
{
%WRCK WRCK;
%WRSTN WRSTN;
%SHIFTWR SHIFTWR;
%UPDATEWR UPDATEWR;
%SELECTWR SELECTWR;
%CAPTUREWR CAPTUREWR;
%WP_SI WP_SI;
%WP_SO WP_SO;
%PARALLEL_IN TAM_in;
%PARALLEL_OUT TAM_out;
%
%WRAPPER_CELL
{
%INPUT_WRAPPER STI_ONE_FF_CELL;
%OUTPUT_WRAPPER STI_ONE_FF_CELL;
%
%PIN_ORDER_LIST
{
%FILE_NAME HS_S27_DST_NEW_top_port.pin;
%
%WRAPPER_CHAIN_FILE
{
%FILE_NAME HS_S27_DST_NEW.wpso;
%
%SKIP_SIGNAL_NAME
{
CLK,PTO,OTP,RST,TAKE_IT_TG,GOT_IT_TG;
}
%INSTRUCTION_TABLE
{
%INSTRUCTION WS_BYPASS = (00000000);
%INSTRUCTION WS_EXTEST = (00000001);
%INSTRUCTION WP_EXTEST = (00000101);
%INSTRUCTION WS_INTEST_SCAN = (00000010);
%INSTRUCTION WP_INTEST_SCAN = (00000011);
}
```
%WRAPPER_CELL STI_ONE_FF_CELL_WITH_SAFE_0
{
%DATA_IN CF1;
%DATA_OUT CFO;
%SCAN_IN CTI;
%SCAN_OUT CTO;
%WP_SHIFT shift;
%APPLY apply;
%CAPTURE_CLK capture_clk;
%SAFE safe;
}

.pso file

%SCAN_MODE SE;
%SCAN_CHAIN scan_chain_0
{
%SI SI;
%SO SO;
%LENGTH 12;
%CLK CLK;
}

.wpso file

%WRAPPER_CHAIN wp_chain0
{
DST_DATA_IN[3];
DST_DATA_IN[2];
DST_DATA_IN[1];
DST_DATA_IN[0];
DST_DATA_OUT;
}
%WRAPPER_CHAIN wp_chain1
{
#scan_chain_0;
}
%SCAN_CHAIN_FILE 'HS_S27_DST_NEW.pso';
### .pin file

<table>
<thead>
<tr>
<th>%PACKAGE_TYPE</th>
<th>HS_S27_DST_NEW</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>//PIN_INDEX</td>
<td>PIN_INDEX</td>
<td>PIN_NAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SI</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>SE</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>SO</td>
</tr>
<tr>
<td>2</td>
<td>DST_DATA_IN[30]</td>
<td>I</td>
</tr>
<tr>
<td>3</td>
<td>DST_DATA_IN[30]</td>
<td>I</td>
</tr>
<tr>
<td>4</td>
<td>DST_DATA_OUT</td>
<td>O</td>
</tr>
<tr>
<td>5</td>
<td>DST_DATA_OUT</td>
<td>O</td>
</tr>
<tr>
<td>6</td>
<td>WRCK</td>
<td>C</td>
</tr>
<tr>
<td>7</td>
<td>WRSTN</td>
<td>E</td>
</tr>
<tr>
<td>8</td>
<td>UPDATEWR</td>
<td>E</td>
</tr>
<tr>
<td>9</td>
<td>SHIFTWR</td>
<td>E</td>
</tr>
<tr>
<td>10</td>
<td>CAPTUREWR</td>
<td>E</td>
</tr>
<tr>
<td>11</td>
<td>SELECTWR</td>
<td>E</td>
</tr>
<tr>
<td>12</td>
<td>WP_SI</td>
<td>I</td>
</tr>
<tr>
<td>13</td>
<td>WP_SO</td>
<td>O</td>
</tr>
</tbody>
</table>