ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
MESTRADOEM EM CIÊNCIA DA COMPUTAÇÃO

LEONARDO REZENDE JURACY

TESTING THE BLADE RESILIENT ASYNCHRONOUS TEMPLATE: A STRUCTURAL APPROACH

Porto Alegre
2018

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

TESTING THE BLADE RESILIENT ASYNCHRONOUS TEMPLATE: A STRUCTURAL APPROACH

LEONARDO REZENDE JURACY

Dissertation submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Prof. Dr. Alexandre de Morais Amory
Co-Advisor: Dr. Matheus Trevisan Moreira

Porto Alegre
2018

Ficha Catalográfica

J95t Juracy, Leonardo Rezende

Testing The Blade Resilient Asynchronous Template : A Structural Approach / Leonardo Rezende Juracy . – 2018.
117 p.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Alexandre de Morais Amory.
Co-orientador: Prof. Dr. Matheus Trevisan Moreira.

1. Resilient design. 2. Asynchronous design. 3. Design for Testability. 4. Cell design. I. Amory, Alexandre de Morais. II. Moreira, Matheus Trevisan. III. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Salete Maria Sartori CRB-10/1363

Leonardo Rezende Juracy

Testing The Blade Resilient Asynchronous Template: A Structural Approach

This Dissertation/Thesis has been submitted in partial fulfillment of the requirements for the degree of Doctor/Master of Computer Science, of the Graduate Program in Computer Science, School of Technology of the Pontifícia Universidade Católica do Rio Grande do Sul.

Sanctioned on March 21st, 2018.

COMMITTEE MEMBERS:

TESTANDO O ARCABOUÇO ASSÍNCRONO RESILIENTE BLADE: UMA ABORDAGEM ESTRUTURAL

RESUMO

Atualmente, a abordagem síncrona é a mais utilizada em projeto de circuitos integrados por ser altamente automatizado pelas ferramentas comerciais e por incorporar margens de tempo para garantir o funcionamento correto nos piores cenários de variações de processo e ambiente, limitando otimizações no período do relógio e aumentando o consumo de potência. Por um lado, circuitos assíncronos apresentam algumas vantagens em potencial quando comparados com os circuitos síncronos, como menor consumo de potência e maior vazão de dados, mas também podem sofrer com variações de processo e ambiente. Por outro lado, circuitos resilientes são uma alternative para manter o circuito funcionando na presença de efeitos de variação. Sendo assim, foi proposto o circuito Blade que combina as vantagens de circuitos assíncronos com circuitos resilientes. Blade utiliza latches em sua implementação e mantém seu desempenho em cenários de caso médio. Independentemente do estilo de projeto (síncrono ou assíncrono), durante o processo de fabricação de circuitos integrados, algumas imperfeições podem acontecer, causando defeitos que reduzem o rendimento de fabricação. Circuitos defeituosos podem apresentar um comportamento falho, gerando uma saída diferente da esperada, devendo ser identificados antes de sua comercialização. Metodologias de teste podem ajudar na identificação e diagnóstico desse comportamento falho. Projeto visando testabilidade (do inglês, *Design for Testability* - DfT) aumenta a testabilidade do circuito adicionando um grau de controlabilidade e observabilidade através de diferentes técnicas. *Scan* é uma técnica de DfT que fornece para um equipamento de teste externo acesso aos elementos de memória internos do circuito, permitindo inserção de padrões de teste e comparação da resposta. O objetivo deste trabalho é propor uma abordagem de DfT estrutural, completamente automática e integrada com as ferramentas comerciais de projeto de circuitos, incluindo uma série de métodos para lidar com os desafios relacionados ao teste de circuitos assíncronos e resilientes, com foco no Blade. O fluxo de DfT proposto é avaliado usando um módulo criptográfico e um microprocessador. Os resultados obtidos para o módulo criptográfico mostram uma cobertura de falha de 98,17% para falhas do tipo *stuck-at* e 89,37% para falhas do tipo *path-delay*, com um acréscimo de área de 112,16%. Os resultados obtidos para o microprocessador mostram uma cobertura de 96,04% para falhas do tipo *stuck-at* e 99,00% para falhas do tipo *path-delay*, com um acréscimo de área de 50,57%.

Palavras-Chave: Circuitos resilientes, Circuitos assíncronos, Projeto visando testabilidade, Projeto de células.

TESTING THE BLADE RESILIENT ASYNCHRONOUS TEMPLATE: A STRUCTURAL APPROACH

ABSTRACT

Nowadays, the synchronous circuits design approach is the most used design method since it is highly automated by commercial computer-aided design (CAD) tools. Synchronous designs incorporate timing margins to ensure the correct behavior under the worst-case scenario of process and environmental variations, limiting its clock period optimization and increasing power consumption. On one hand, asynchronous designs present some potential advantages when compared to synchronous ones, such as less power consumption and more data throughput, but they may also suffer with the process and environmental variations. On the other hand, resilient circuits techniques are an alternative to keep the design working in presence of effects of variability. Thus, Blade template has been proposed, combining the advantages of both asynchronous and resilient circuits. The Blade template employs latches in its implementation and supports average-case circuit performance. Independently of the design style (synchronous or asynchronous), during the fabrication process of integrated circuits, some imperfections can occur, causing defects that reduce the fabrication yield. These defective ICs can present a faulty behavior, which produces an output different from the expected, and it must be identified before the circuit commercialization. Test methodologies help to find and diagnose this faulty behavior. Design for Testability (DfT) increases circuit testability by adding a degree of controllability and observability through different test techniques. Scan design is a DfT technique that provides for an external test equipment the access to the internal memory elements of a circuit, allowing test pattern insertion and response comparison. The goal of this work is to propose a fully integrated and automated structural DfT approach using commercial EDA tools and to propose a series of design methods to address the challenges related to testing asynchronous and resilient designs, with focus on Blade template. The proposed DfT flow is evaluated with a criptocore module and a microprocessor. The obtained results for the criptocore module show a fault coverage of 98.17% for stuck-at fault model and 89.37% for path-delay fault model, with an area overhead of 112.16%. The obtained results for the microprocessor show a fault coverage of 96.04% for stuck-at fault model and 99.00% for path-delay fault model, with an area overhead of 50.57%.

Keywords: Resilient design, Asynchronous design, Design for Testability, Cell design.

LIST OF FIGURES

Figure 1.1 – Blade synthesis flow where the red dashed square shows the proposed contribution to improve the Blade testability. The numbers in the Figure represent the three contributions of this work. 27

Figure 2.1 – Muxed-D cell [WWW06]. 30

Figure 2.2 – Muxed-D waveform [WWW06]. 30

Figure 2.3 – Scan chain with Muxed-D cells [WWW06]. 30

Figure 2.4 – Time borrowing example [Kue18]. 32

Figure 2.5 – Backward retiming [SYN15a]. 33

Figure 2.6 – Forward retiming [SYN15a]. 33

Figure 2.7 – The LSSD cell block diagram [EW77]. 34

Figure 2.8 – The LSSD cell gate level diagram [EW77]. 34

Figure 2.9 – A typical LSSD cell waveform [WWW06]. 35

Figure 2.10 – Scan chain using single-latch LSSD approach [WWW06]. 36

Figure 2.11 – Scan chain using double-latch LSSD approach [WWW06]. 37

Figure 2.12 – LSSD cell using L2* [ABF94]. 37

Figure 2.13 – Logic gate diagram of the LSSD cell using L2* [ABF94]. 38

Figure 2.14 – Scan chain using L2* LSSD cells [ABF94]. The red dashed line represents the scan-chain path. 38

Figure 2.15 – LSSD test protocol for edge-triggered sequential cells (Clocked-LSSD). The "R" means that the system data is registered at clock rising edge. The "*" at the clocks **A** and **B** means that clock **A** must pulse before clock **B** [SYN15b]. 41

Figure 2.16 – Bundled-data block diagram [Spa01]. 43

Figure 2.17 – 4-phase handshake protocol waveform [Spa01]. 43

Figure 2.18 – 2-phase handshake protocol waveform [Spa01]. 44

Figure 2.19 – Blade Architecture [HTMH+15]. 45

Figure 2.20 – Blade EDL [HTMH+15]. 45

Figure 2.21 – Q-Flop cell [RMCF88]. 47

Figure 3.1 – Razor flip-flop [EKD+03]. 49

Figure 3.2 – RazorII flip-flop [DTP+09]. 51

Figure 3.3 – Razor-Lite flip-flop [KKF+14]. 51

Figure 3.4 – Bubble Razor flip-flop [FFK+13]. 53

Figure 3.5 – SafeRazor circuit blocks [CBC+15]. 54

Figure 3.6 – Scan Razor flip-flop [ATA15]. 55
Figure 3.7 – Timed flip-flop [FTK08]. 57
Figure 3.8 – Timed flip-flop [VFT+14]. 57
Figure 3.9 – Scan cell proposed by Kuppuswamy et al. [KDF+04]. 59
Figure 3.10 – Scan flip-flop with single-event upset detection. [DKM05]. 60
Figure 3.11 – XSEUFF scan cells [JOC07]. 62
Figure 3.12 – Error detection sequential scan approaches [HGJX13]. 62
Figure 4.1 – Blade sequential elements. 65
Figure 4.2 – Testable Transition Detector. This diagram is an adaptation of the Blade original paper [HTMH+15]. 67
Figure 4.3 – Static-logic Q-Flop implementation [MM07]. 68
Figure 4.4 – Proposed testable Q-Flop cell. 68
Figure 4.5 – Testable Q-Flop physical synthesis evaluation scenario. 70
Figure 4.6 – Testable Q-Flop physical synthesis results. 71
Figure 4.7 – LSSD protocol violation in Yurash's [Yur95] clocked-LSSD cell. 72
Figure 4.8 – Proposed optimized Clocked-LSSD schematic. 73
Figure 5.1 – Blade synthesis flow where the red dashed square shows the proposed contribution to improve the Blade testability. The numbers in the Figure represent the three contributions of this work. 77
Figure 5.2 – Proposed Error Detection Logic implementation. The TD was split into a latch and a transition detector circuit, and the Q-Flop was split into a flip-flop and a metastability filter circuit. The red dashed squares highlight these modifications. 78
Figure 5.3 – Controller modification to support scan chain insertion. MUX components were added to allow controllability for the clocks. The red dashed squares highlight the MUXs. 78
Figure 5.4 – Detailed proposed Blade DfT flow. 80
Figure 5.5 – XTEA pipeline design. This design is based in the Speed XTEA version presented in [Kap08]. 81
Figure 5.6 – Plasma microprcessor block diagram. 82
Figure 5.7 – Schematic sample of the scan chain. This Figure was extracted from DC Compiler tool by Synopsys. 82
Figure A.1 – LSSD using 28 nm technology node. 103
Figure A.2 – LSSD using 28 nm technology node with reset. 104
Figure A.3 – LSSD using 28 nm technology waveform. 104

Figure A.4 – LSSD using 28 nm technology setup characterization time arcs definition. 105
Figure A.5 – LSSD using 28 nm technology delay characterization time arcs definition. 105
Figure A.6 – LSSD liberty code sample presenting the manual modification necessary to the synthesis tool recognize the LSSD function. The *statetable* and the *test_cell* functions are added manually after the characterization. . . 106
Figure B.1 – Verilog code to Transition Detector cell. 107
Figure B.2 – Transition Detector liberty code sample. 107
Figure B.3 – Spice code to Transition Detector cell. 108
Figure C.1 – Verilog User Defined Primitives code to Metastability Filter cell. 109
Figure C.2 – Metastability Filter liberty code sample. 110
Figure C.3 – Spice code for Metastability Filter cell. 110
Figure D.1 – Verilog User Defined Primitives code to Clocked-LSSD cell. 111
Figure D.2 – Proposed Clocked-LSSD liberty code sample. 112
Figure D.3 – Spice code for the proposed Clocked-LSSD cell. 113
Figure D.4 – Proposed Clocked-LSSD liberty code sample. 114
Figure E.1 – DfT insertion signals declaration. 115
Figure E.2 – DfT insertion test protocol declaration. 116
Figure E.3 – DfT insertion using Q-Flop test setup insertion. 116
Figure E.4 – ATPG tool initialization commands. 117
Figure E.5 – Specifying the stuck-at fault model simulation for the ATPG process. . 117
Figure E.6 – ATPG commands to remove the internal faults of the Transition Detector cell. 117
Figure E.7 – Specifying the path-delay fault model simulation for the ATPG process.117

LIST OF TABLES

Table 2.1 – Comparison between the Muxed-D cell and the LSSD cells 39
Table 2.2 – Q-Flop truth table 47
Table 3.1 – Razor family characteristic regarding test insertion and register components. 63
Table 3.2 – Scan approaches limitations regarding DfT automated insertion and ATPG. 64
Table 4.1 – Comparison of original Q-Flop against testable Q-Flop. 70
Table 4.2 – Area and power evaluation of the synchronous XTEA with full scan and clocked-LSSD scan cells. 74
Table 4.3 – Area estimation for test cells 75
Table 5.1 – Synthesis results for the XTEA design regarding the flop-based synchronous version, flop-based synchronous version with DfT, Blade version and Blade version with DfT. 84
Table 5.2 – Cell proportion of the XTEA synchronous version and XTEA Blade version. 85
Table 5.3 – Area detailing of extra components used in test insertion and Blade conversion. 85
Table 5.4 – ATPG stuck-at fault model results for XTEA synchronous version and XTEA Blade version. 86
Table 5.5 – Details of the XTEA Blade undetected faults. 86
Table 5.6 – Comparison of the ATPG results for stuck-at fault model between the Blade with and without scannable Q-Flop (Clocked-LSSD). 86
Table 5.7 – Comparison of the undetected faults between the Blade with and without scannable Q-Flop (Clocked-LSSD). The column % represents the percentage of the faults detected with the Q-Flop in the scan chain. 87
Table 5.8 – ATPG path-delay fault model results for XTEA synchronous version and XTEA Blade version. 87
Table 5.9 – Throughput results for XTEA synchronous version and XTEA Blade version. 88
Table 5.10 – Synthesis results for the XTEA normalized design regarding the flop-based synchronous version, flop-based synchronous version with DfT, Blade version and Blade version with DfT. 89
Table 5.11 – ATPG stuck-at fault model results for XTEA normalized synchronous version and Plasma Blade version. 89

Table 5.12 – ATPG path-delay fault model results for XTEA normalized synchronous version and XTEA Blade version. 90
Table 5.13 – Synthesis results for the Plasma normalized design regarding the flop-based synchronous version, flop-based synchronous version with DfT, Blade version and Blade version with DfT. 91
Table 5.14 – ATPG stuck-at fault model results for Plasma normalized synchronous version and Plasma Blade version. 92
Table 5.15 – ATPG path-delay fault model results for Plasma normalized synchronous version and Plasma Blade version. 92
Table 6.1 – Implemented levels of the proposed cells. 95

LIST OF ACRONYMS

ATPG – Automatic Test Pattern Generation

BD – Bundled-Data

BIST – Built-in Self-Test

CAD – Computer-aided Design

DfT – Design for Testability

DSM – Deep Submicron

EDL – Error Detection Logic

IC – Integrated Circuit

LSSD – Level Sensitive Scan-based Design

PVT – Process, Voltage and Temperature

QDI – Quasi-delay-insensitive

SDF – Standard Delay File

SoC – System-on-Chip

SPF – Standard Test Interface Language Procedure File

STA – Static Time Analyses

TCL – Tool Command Language

TD – Transition Detector

UDP – User-Defined Primitives

VHDL – Very-High-Speed Integrated Circuit Hardware Description Language

CONTENTS

1 INTRODUCTION **23**
1.1 GOALS 25
1.2 RESEARCH SCOPE 25
1.3 CONTRIBUTION 26
1.4 DOCUMENT ORGANIZATION 27

2 CONCEPTS **29**
2.1 STUCK-AT AND DELAY FAULT MODELS 29
2.2 MUXED-D SCAN CHAIN 29
2.3 TIMING OPTIMIZATIONS FOR LATCH-BASED DESIGNS 31
2.3.1 CLOCK SKEW AND JITTER 31
2.3.2 TIME BORROWING 32
2.3.3 REGISTER RETIMING 33
2.4 LEVEL SENSITIVE SCAN-BASED DESIGN 34
2.4.1 LSSD SINGLE-LATCH SCAN 35
2.4.2 LSSD DOUBLE-LATCH SCAN 36
2.4.3 LSSD L2* SCAN 37
2.4.4 COMPARISON BETWEEN LSSD CELLS AND LSSD APPROACHES 39
2.5 LSSD SCAN PROTOCOL FOR FLIP-FLOP MEMORY ELEMENTS 40
2.6 TIMING RESILIENT CIRCUITS 40
2.7 ASYNCHRONOUS CIRCUITS 41
2.8 BLADE TEMPLATE 44
2.9 Q-FLOP 46

3 STATE OF THE ART **49**
3.1 RESILIENT TEMPLATES 49
3.1.1 RAZOR 49
3.1.2 RAZORII 50
3.1.3 RAZOR-LITE 51
3.1.4 BUBBLE RAZOR 52
3.1.5 SAFERAZOR 52
3.2 TESTING TIMING RESILIENT TEMPLATES 54

3.2.1 SCAN RAZOR FLIP-FLOP 54
3.2.2 TIMED FLIP-FLOP 56
3.3 TESTING SOFT-ERROR RESILIENT TEMPLATES 57
3.3.1 SCAN APPROACH TO BUILT-IN SOFT-ERROR RESILIENCE 58
3.3.2 SCAN APPROACH TO SINGLE-EVENT UPSET DETECTION, CORRECTION, AND MONITORING 59
3.3.3 XSEUFF SCAN CELLS 60
3.3.4 ERROR DETECTION SEQUENTIAL SCAN APPROACHES 61
3.4 STATE OF THE ART CONCLUSION 63

4 PROPOSED CELLS DESIGN 65
4.1 REPLACING LATCHES 66
4.2 TESTABLE TRANSITION DETECTOR 66
4.3 TESTABLE Q-FLOP 67
4.3.1 COMPARISON WITH THE ORIGINAL Q-FLOP 69
4.3.2 PLACE AND ROUTE METHODOLOGY 70
4.4 CLOCKED-LSSD OPTIMIZATION 72
4.4.1 RESULTS 73
4.4.2 CELL AREA ESTIMATION 75

5 BLADE DFT SYNTHESIS FLOW 77
5.1 DETAILED BLADE DFT FLOW 79
5.2 CASE STUDIES 80
5.2.1 XTEA 80
5.2.2 PLASMA 81
5.3 GENERAL RESULTS SETUP 81
5.4 XTEA EVALUATION 82
5.4.1 SILICON AREA RESULTS 83
5.4.2 STUCK-AT FAULT MODEL RESULTS 85
5.4.3 PATH-DELAY FAULT MODEL RESULTS 87
5.5 OPTIMIZED XTEA FOR NORMALIZED THROUGHPUT EVALUATION 87
5.5.1 NORMALIZED SILICON AREA RESULTS 88
5.5.2 NORMALIZED ATPG RESULTS 89
5.6 OPTIMIZED PLASMA FOR NORMALIZED THROUGHPUT EVALUATION 90
5.6.1 NORMALIZED SILICON AREA RESULTS 90

5.6.2 NORMALIZED ATPG RESULTS 91

6 CONCLUSION **93**

6.1 CONTRIBUTIONS 94

6.2 CURRENT LIMITATIONS AND FUTURE WORK 94

REFERENCES **97**

APPENDIX A – LSSD Design and Implementation **103**

APPENDIX B – Transition Detector Codes **107**

APPENDIX C – Metastability Filter Codes **109**

APPENDIX D – Proposed Clocked-LSSD Codes and Scripts **111**

APPENDIX E – DfT Automation Scripts **115**

1. INTRODUCTION

Nowadays, the most used paradigm in circuits design and implementation is the synchronous one, due to the automation provided by the commercial computer-aided design (CAD) tools. The memory elements employed in this approach use a global clock signal to synchronize the circuit components. Flip-flops and latches are the most common memory elements used to implement synchronous designs.

Latches are level-sensitive memory elements [RCN03], and according to the literature, they are employed in circuits designed for high-performance applications. Flip-flops are edge-triggered memory elements and simplify timing analysis when compared to latches, as they only depend on the clock period and do not allow complex timing scenarios such as timing borrowing. However, flip-flops are more susceptible to problems, such as clock skew and jitter. A major problem in synchronous designs is the necessity to incorporate timing margins to ensure correct operation under worst-case conditions, which limit the clock period optimization potential and the performance gains and also increase the power dissipation. Resilient circuits techniques are proposed to avoid this kind of problem, reducing these timing margins [EKD+03] [DTP+09] [KKF+14] [BTK+09].

Another class of design styles relies on the asynchronous paradigm. Asynchronous circuits do not employ a global synchronization signal. Instead, synchronization occurs using handshake protocols between individual components. Asynchronous designs present some potential advantages when compared to synchronous ones, such as power consumption and performance. Also, latches are more usual in asynchronous designs [CKLS06,SN07,NS15], benefiting from latches advantage as clock skew and jitter tolerances. The Blade [HTMH+15] template is an example of the use of latches in asynchronous circuits. It is an asynchronous resilient template that supports average-case circuit performance. A case study using the Blade template [HTMH+15] showed a 30% power reduction at the same performance, when compared to a similar synchronous circuit, for an area overhead of about 10%.

Independently of the design style, during the fabrication process of integrated circuits (ICs), some imperfections can occur, causing defects that reduce fabrication yield [WWW06]. These defective ICs can present a faulty behavior, which must be identified before their commercialization. Among the defects that can occur, there are the oxide break, parasitic transistors, contact degradation and impurities at the surface. Test methodologies help to find and diagnose faults, which is important to improve the manufacturing yield [ABF94].

Faults must be abstracted to make it viable to apply test methodologies to ease simulating the behavior of the faults and perform diagnosis. A fault model is a computational abstraction to represent in software a hardware defect [BA02]. Stuck-at is the most employed fault model because it can represent different kinds of faults independently of the technology [WWW06]. Another existing model is the path-delay fault model [BA02], used to verify if the output fails to reach the correct value of a predefined time constraint. Still, many times it can be difficult to test all the defects that can occur in a circuit, once the test time is limited.

Testability is the parameter that determines how much a circuit is testable, which is influenced by its logic size and complexity [BA02]. Testability also allows to determine the circuit state (which can be *normal*, *inoperable* or *degraded*) and to determine the effective test cost [ABF94]. The test of a design is one of the major components of the manufacturing cost. A microprocessor circuit test can cost thousand times more than the fabrication cost. This cost is associated with factors like test generation, testing time and automatic test generation cost [ABF94].

Two important attributes related to testability are controllability and observability. *Controllability* is the capacity to change the internal nodes of a circuit, setting the values from its primary input pins [ABF94]. *Observability* is the capacity to determine the value of the internal nodes of a circuit controlling the inputs and observing the value at the circuit primary outputs pins [ABF94]. The higher the controllability and observability, the higher is the testability.

Design for testability (DfT) increases circuit testability by adding a degree of controllability and observability through different test techniques. *Fault coverage* is the percentage of faults detected during the test process [ABF94]. DfT can reduce the cost of test generation and guarantee quality by increasing the fault coverage from the test [WWW06]. One DfT approach is to use ad hoc techniques, which include test point insertion, circuit partitioning, and logic redundancy. However, to use these techniques, the circuit must be designed for test purposes at the start of implementation. It is not possible to insert the ad hoc techniques generically.

Another approach is the scan design. It provides external access to the internal memory elements of a circuit by interconnecting memory elements of the design, forming a scan-chain. The scan-chain starts at the primary inputs and ends at the primary outputs of the design. Thus, it is possible to set a determined memory element with a value and observe its output, increasing controllability and observability. The use of scans permits a more generic test approach, this technique modifies just the memory elements of the design and does not need to be incorporated at the start of implementation. Among the architectures used to form a scan-chain, there is the Muxed-D cell to replace D flip-flops and the Level-Sensitive Scan Design (LSSD) cell to replace D latches. When compared to the literature [WWW06] [EW77], LSSD presents some advantages over Muxed-D. It does not use a MUX in its input, which makes its performance higher than the Muxed-D, once the data just pass through a latch. Furthermore, the LSSD is a race-free approach, which is not the case of Muxed-D. However, LSSD presents a big area overhead once each latch is replaced by two latches.

Specifically for asynchronous designs, the test can present some challenges for DfT approache. The absence of a global clock and the usage of non-standard sequential components makes it difficult to create a scan-chain with its memory elements. However, as mentioned before, some asynchronous designs such as Blade use latches, making the LSSD an alternative, since latches are level-sensitive memory elements. Considering the state of the art, although power reduction and lower area overhead were observed, the proposed Blade template [HTMH+15] does not address testability issues, which means that the Blade can present a low fabrication yield. Kuentzer [Kue18] proposed an approach to turn Blade's error detection logic (EDL) testable, but that work does not consider the others sequential elements present in the data-path and the circuit controller testability.

1.1 Goals

Considering the above discussion, the strategic goal of this work is to propose a fully integrated and automated DfT approach for the Blade template using commercial EDA tools and to check the viability of a fully structural test approach based on scan for Blade. To reach this strategic goal, the following specific objectives are needed:

- **Propose scan cells to perform the test in the Blade template**: Blade has unconventional sequential cells such as latches with error transition detector and Q-Flops. Those cells need to be scannable to improve the fault coverage. Thus, this goal is to convert these cells to scannable version that is compatible with standard EDA to allow DfT insertion at Blade;
- **Modify the synthesis flow to allow DfT insertion**: The original Blade synthesis flow does not support automated DfT insertion. Thus, one of the goals of this work is modifying the synthesis flow to support automated DfT insertion using DfT commercial tools;
- **Evaluate testability using commercial ATPG tools**: The DfT insertion is evaluated using ATPG tools to extract fault coverage, number of patterns and number of cycles used to perform the test;
- **Evaluate the proposed test costs**: The test overhead is evaluated regarding area and number of cells.

1.2 Research Scope

The scope of this work includes:

- **Only manufacturing test, no online test**: this work focuses just on manufacturing test, with focus on the scan chain DfT technique;
- **Only logic synthesis level for Blade**: The original Blade synthesis flow does not address physical synthesis. Specific cell layouts are needed to perform physical synthesis, such as C-elements and latches with error transition detector, besides the test cells. These layouts require a great amount of time, which were not possible to include these tasks in the master degree course. Only the Q-Flop evaluation addresses place and route constraints;
- **Only Stuck-at and path-delay fault models**: To evaluate the DfT insertion, only stuck-at and path-delay fault models are used. Others fault-models as gate-delay are out of the scope of this work;

1.3 Contribution

The main contributions of this work are:

1. **A test library to latch-based/mix designs**: LSSD and Clocked-LSSD test cells were implemented. The LSSD cell design and validation was published in [JMKA17];
2. **A new set of cells for Blade template**: A more test friendly version of error transition detector cell and a metastability filter cell were implemented;
3. **A test flow for Blade circuits**: As mentioned before, the original Blade synthesis flow does not support automated DfT insertion. Additional scripts to include the DfT were included into the Blade's design flow.

Figure 1.1 presents the proposed Blade synthesis flow with DfT insertion. The red dashed square shows the contributions of this work. The **Test Cells** block represents the test library to latch-based and mix designs. The **New Blade Cells** block represents the new set of testable cells to implement Blade template functions. In addition, the following papers were derived from this dissertation:

- As first author:
  - *Optimized Design of an LSSD Scan Cell*: published on IEEE Transaction on VLSI (Qualis A1);
  - *Testable Q-Flop: A Scannable Metastability-free Memory Element*: submitted to IEEE Transaction on VLSI (Qualis A1);
  - *An LSSD Compliant Scan Cell for Flip-Flops*: submitted to IEEE ISCAS (Qualis A1).

- As co-author author:
  - *On the Reuse of Timing Resilient Architecture for Testing Path Delay Faults in Critical Paths*: published on DATE (Qualis A1);
  - *Testable Error Detection Logic Design Applied to an Asynchronous Timing Resilient Template*: submitted to IEEE ISCAS (Qualis A1);

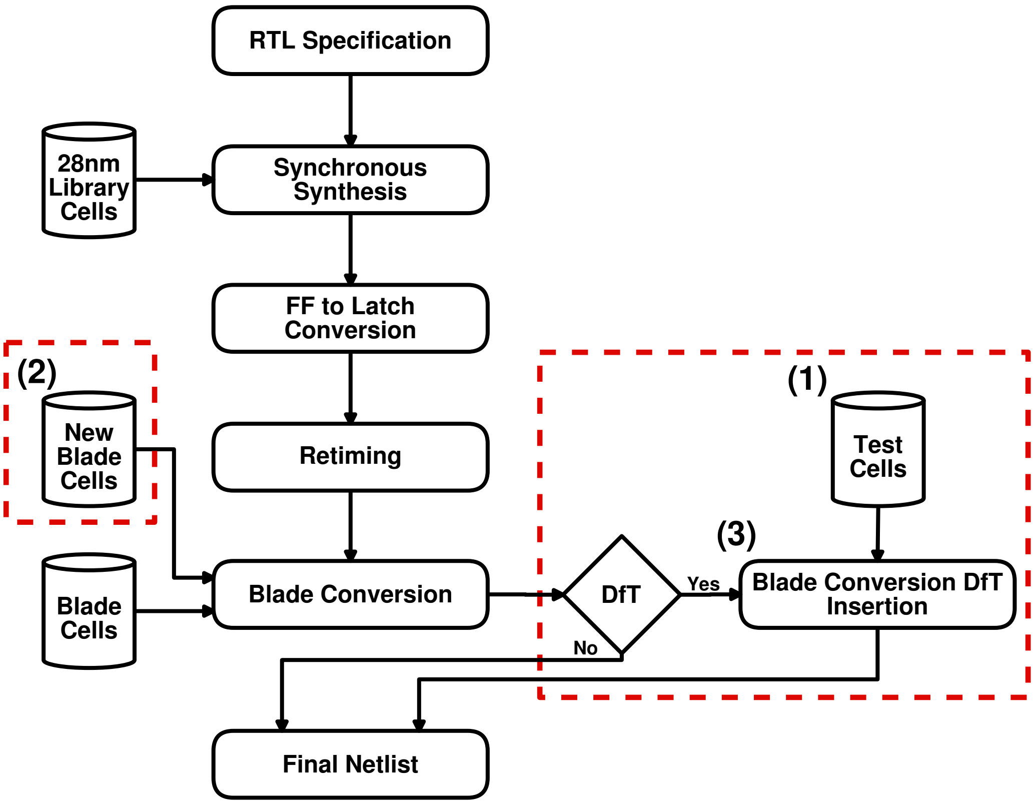


Figure 1.1 – Blade synthesis flow where the red dashed square shows the proposed contribution to improve the Blade testability. The numbers in the Figure represent the three contributions of this work.

1.4 Document Organization

The rest of this dissertation comprises the following sections: Section 2 provides relevant background information; Section 3 presents the state of the art in latch-based designs, LSSD approaches, resilient circuits and DfT to asynchronous and resilient designs; Section 4 describes the implementation of the test cells used to insert DfT in the Blade template; Section 5 describes the necessary modification in the Blade flow to add DfT logic; finally Section 6 presents the conclusion of this work.

2. CONCEPTS

This Chapter presents basic concepts about fault models, sequential cells, design-for-testability, asynchronous circuits and resilient architectures.

2.1 Stuck-at and Delay Fault Models

A fault model is a computational abstraction to represent a defect as software [BA02]. The stuck-at fault model is the most employed, due to its capability to represent different kind of faults and for being independent of technology. For example, a circuit node fixed in zero (stuck-at 0), like a short circuit with the ground, or one (stuck-at 1), like a short circuit with the supply voltage [WWW06].

Another model is the delay fault model [BA02]. This model uses a pair of test vectors, one to initialize the circuit in a known state and other to stimulate the logical paths. Logical path is a stretch of combinational logic from a start point to an end point. These points can be even Inputs and outputs or memory elements. If the delay of a logical path exceeds the design timing constraints, invalid values are captured by sequential cells or primary outputs, which characterize a fault.

2.2 Muxed-D Scan Chain

Standard D flip-flops are the most used memory element to implement synchronous designs. D flip-flops are registers that, when occurs a clock edge event (either rising or falling) the input data is sampled and copied to the output. Thus, one of the most used cells to implement scan-chains are the Muxed-D cells. These cells are composed of a D flip-flop and a MUX, as illustrated in Figure 2.1. This cell has two additional input pins, **SI** that is used to input test data, and **SE** that is used to choose if the cell is operating in normal mode or test mode. This cell uses the output pin both as normal output (**Q**) and test output (**SO**) [ABF94].

Figure 2.2 shows the time behavior of the Muxed-D cell. When **SE** is at logic level low, the circuit operates in normal mode, which means that the **DI** input value is transmitted to **Q**/**SO** in the clock edge event. When **SE** is in logic level high, the circuit is operating in test mode, which means that the **SI** input value is transmitted to **Q**/**SO** in the clock edge event.

Figure 2.3 shows a scan-chain formed by Muxed-D cells. The output **Q**/**SO** is connected to the next scan cell **SI** input, making the connection between the scan elements. When in normal mode, **SE** is at logic level low and the registers store the **DI** input value. When in test mode, **SE** is at logic level high and the circuit stores the **SI** input. The register

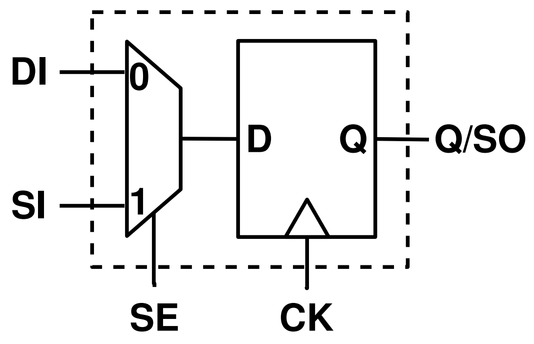


Figure 2.1 – Muxed-D cell [WWW06].

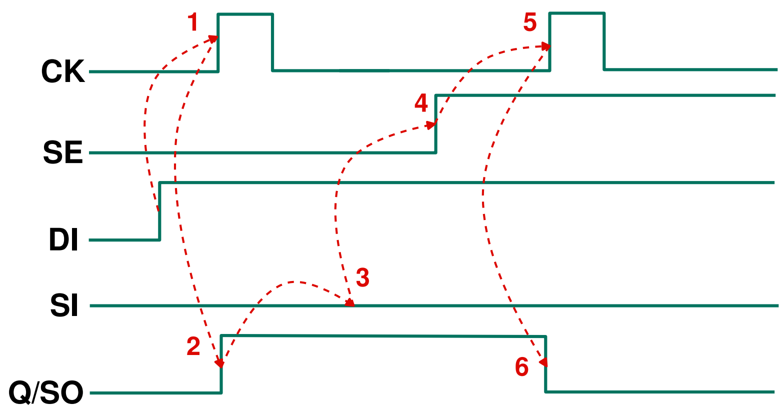


Figure 2.2 – Muxed-D waveform [WWW06].

stored values are shifted to the register on the right side at each clock cycle, which means that it takes *n* clock cycles to initialize a scan-chain with *n* registers. After the initialization of all registers, **SE** is set to logic level low, and the circuit goes back to operate as in normal mode, now with the initialized values through **SI** capturing the response of the combinational logic in the **DI** input. The test results are observed in the output **SO** after *n* clock cycles.

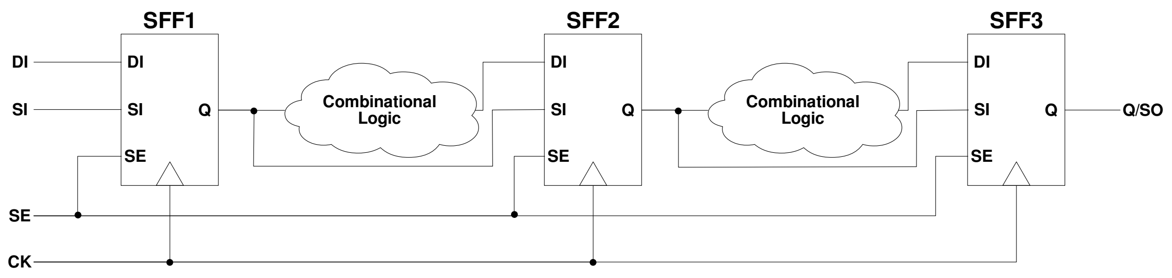


Figure 2.3 – Scan chain with Muxed-D cells [WWW06].

The scanning approach permits two ways to insert scanable cells [MZ00]:

- Full-scan insertion: All the memory elements of the design are replaced by scannable elements;

- Partial-scan insertion: A subset of memory elements of the design are replaced by scannable elements;

As an advantage, partial-scan allows a smaller test overhead than full-scan and reduce application time, once the number of elements of the scan-chain is reduced. However, the processing time of the ATPG algorithm may increase using partial-scan insertion. Partial-scan insertion can need a number of test vectors greater than full-scan insertion to keep the fault coverage of the full-scan insertion, once some part of the circuit cannot be covered due the reduction of the number of scan elements.

2.3 Timing optimizations for Latch-Based Designs

Latch-based designs present advantages compared to flop-based designs, for example, the capacity to store data in each half clock cycle. Besides, the level-sensitive characteristic enables latches to use a full cycle even in the presence of clock skew [Har00]. Nevertheless, depending on the path delay of the design, some latches need more than half clock cycle to process the data. Latches support the time borrowing characteristic that permits work around this problem. Register retiming is another technique that works both for latches and flip-flops. Time borrowing and register retiming are described next because they are used in the proposed testable flow. These processes are supported by EDA tool vendors, like Synopsys [SYN15a] and Cadence [Cad11].

2.3.1 Clock Skew and Jitter

Flip-flop-based designs can present timing related issues such as clock skew and jitter, which increase the sequential logic overhead. Clock skew is a phenomenon that occurs when the clock signal does not reach all the memory elements at the same time, which may affect the circuit correctness. Jitter is a statistical measure of the time difference between the clock edges of the expected clock signal and the real clock behavior [YK04].

The above phenomena are aggravated by the fabrication process and can influence the circuit operating frequency and the flip-flop setup and hold times. Thus, they must be considered to specify the clock period, which can decrease the performance of flop-based designs, once the clock period may be subject to additional margins to avoid these problems. As advantages, latches are more robust to skew and jitter, which means that the sequential overhead decreases when compared to flip-flop-based designs, improving the performance of the circuit [YK04].

2.3.2 Time Borrowing

Time borrowing [Har00] is a latch-based design style where time is borrowed from a previous logic stage by following logic stages in order to meet timing constraints. Figure 2.4 shows an example of time borrowing where the **C1** and **C2** are outdated and have a period of 10 ns, with setup and hold times equal to zero. The duty cycle of these clocks is 50%, which means that 50% of the period the clocks are high, and the other 50% are low. Thus, when **c1** is high, the latch **L1** and **L3** are transparent and the path between **L1** and **L2** have 5 ns to process the data from the input. When **c2** is high the latch, **L2** is transparent and the path between **L2** and **L3** have 5 ns to process the data from **L2**.

However, the path **L1** and **L2** have a delay of 7 ns, and it is not possible to process the data when **c1** is high. The path between **L2** and **L3** has a delay of 3 ns, which means that this path has a slack of 2 ns relative to the **c2** duty cycle. So, when **c2** is high, **L2** is transparent, making possible to process the 2 ns remaining of the path between **L1** and **L2** and pass the value through **L2** to the path between **L2** and **L3** process the data, avoiding additional margins in the system clock periods to work around this situation. Time borrowing is automated by EDA tool vendors, like Synopsys [SYN15a] and Cadence [Cad11].

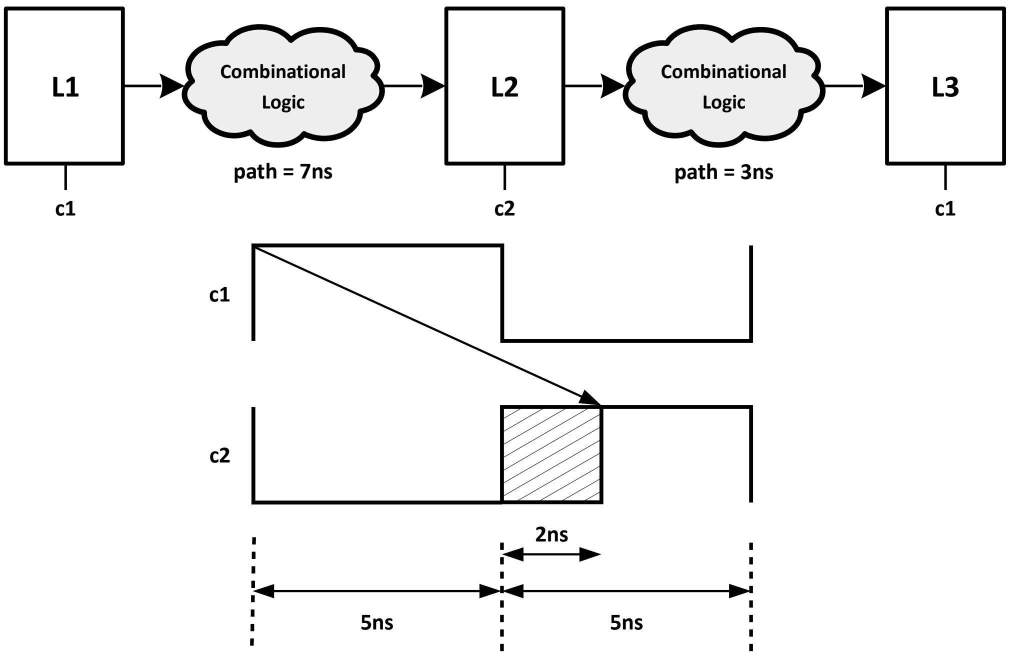


Figure 2.4 – Time borrowing example [Kue18].

2.3.3 Register Retiming

Register retiming [SYN15a] is a sequential optimization technique that consists of moving memory elements (flip-flops or latches) through the combinational logic gates to reduce the area and timing of a given circuit. This process is supported by EDA tool vendors, like Synopsys [SYN15a], which makes the process easier and faster to apply in specific circuits, due to the automation of the registers location and adjustment. The technique moves registers forward or backward through the combinational logic, balances the delay, and optimizes the design based on a given clock period. The modified circuit design keeps the same functional behavior. In Figure 2.5 and Figure 2.6, the registers are examples of backward and forward retiming, respectively.

Backward retiming is when the memory element is connected to the cells fanout and moved to its input. In Figure 2.5, the register **Flop 8** is moved to the OR gate inputs, and **Flop 7** is moved to AND gate input. Forward retiming is when the memory element is connected directly in the fanin of the cell and moved to the cell output. In Figure 2.6, the registers **Flop 2** and **Flop 4** are moved to the OR gate output, and **Flop 6** is removed from the circuit.

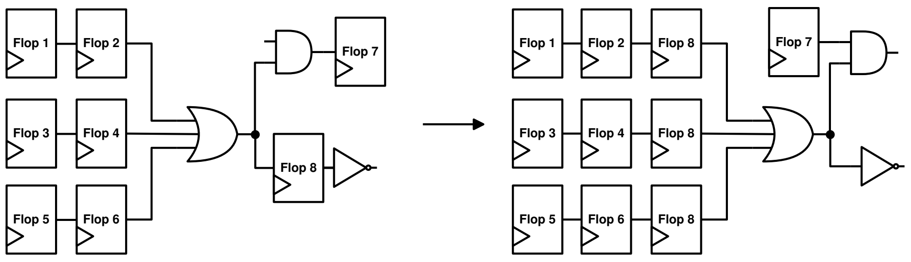


Figure 2.5 – Backward retiming [SYN15a].

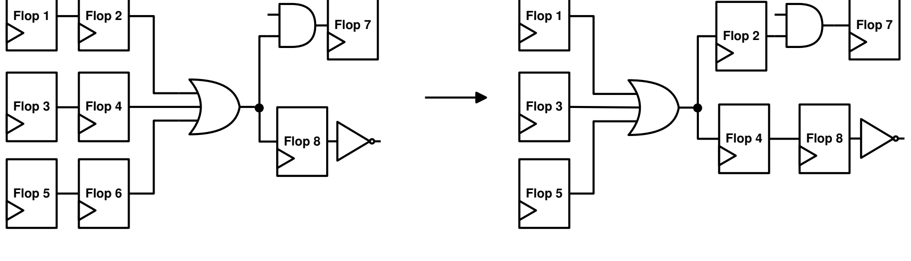


Figure 2.6 – Forward retiming [SYN15a].

2.4 Level Sensitive Scan-based Design

While some scan cells use the clock edge to process data, as the Muxed-D mentioned before, other cells assume the use of the clock level. It is the case of the Level Sensitive Scan-based Design (LSSD), proposed by Eichelberger and Williams [EW77].

An LSSD cell, as shown in Figure 2.7, is a shift register composed by two latches, one type D master with two inputs and other type D slave with one input. This cell has three clock signals: **C** and **A**, used to select between **D** data input and **I** scan input to be copied to the output **L1**, and clock **B**, used to propagate the **L1** node value to the **L2** output. These three clocks signals must be non-overlapping to avoid race condition [WWW06].

Figure 2.7 shows the original LSSD block diagram, where "*" in **L1** is used to separate test pins from normal pins [EW77]. Figure 2.8 shows the originally proposed gate level diagram for this circuit.

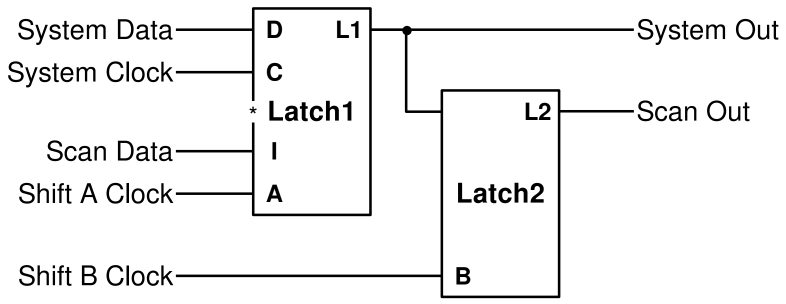


Figure 2.7 – The LSSD cell block diagram [EW77].

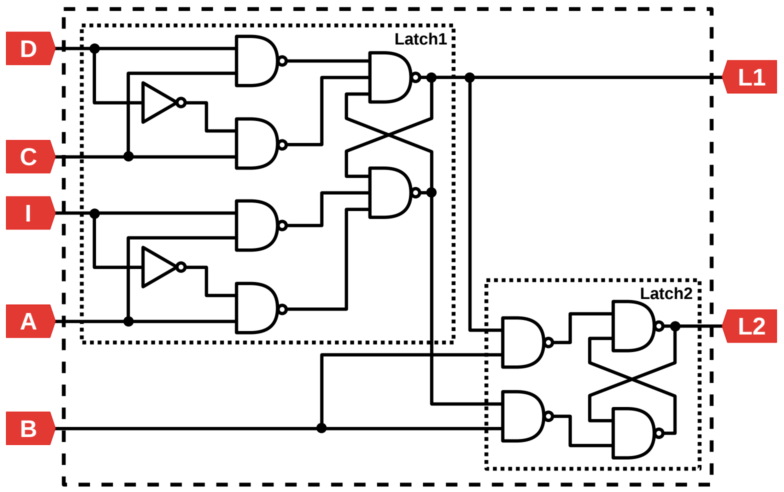


Figure 2.8 – The LSSD cell gate level diagram [EW77].

In normal mode, the clock signal **C** enables the normal input **D** to store its value in output **L1**. In shift mode, the clock signal **A** is used to enable the test input **I** to store its value in **L1** output. In test mode, the clock signal **B** is used to store the **L1** value in output **L2** [WWW06]. Figure 2.9 shows the time behavior of the LSSD cell. When **C** is in logic level high, latch **L1** is transparent. The same behavior occurs when **A** is at logic level high. When **B** is at logic level high, **L2** is transparent. The literature shows some proposed variations that address to power, area and delay reduction [DPH+12, SS03, Sav86, Sav97a, ZM01, Sav97b]. Also, the literature shows the application of the LSSD in high-performance processors from IBM and Motorola [HBB+05, WKP+02, PAG+99, BMS+02], asynchronous designs [EBE05] and others works that rely on LSSD to apply and reduce the overhead of test techniques, as boundary-scan [ZUC01]. The following sections discuss some variations of LSSD-based scan chains.

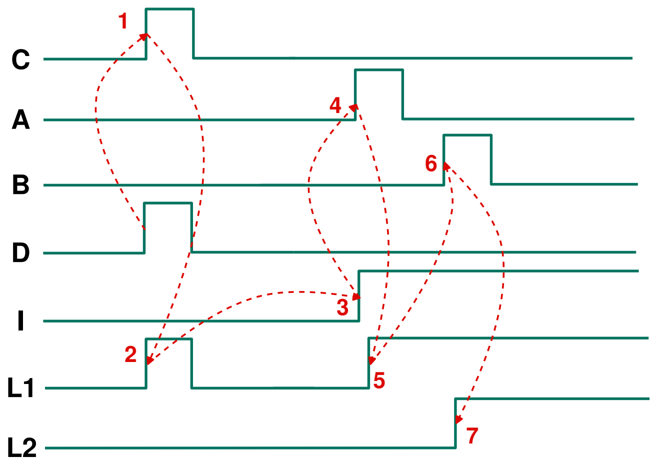


Figure 2.9 – A typical LSSD cell waveform [WWW06].

2.4.1 LSSD Single-Latch Scan

Figure 2.10 shows the scan connection using the LSSD single-latch approach. This approach uses **L1** to connect to the combinational logic and **L2** to connect to the **I** test input. This approach uses two non-overlapping system clocks (**C1** and **C2**) to avoid combinational loop in the design [WWW06].

In normal mode, the **D** input is used as the system input, and **L1** as a functional output, and the system clocks are **C1** and **C2**. In test mode, **I** is used as test input, and **A** is used as selector of **I** input. **L2** is connected to **I**, and this output is activated when the **B** clock is is high, copying the **L1** value to **L2** output of the next cell.

In Figure 2.10, **X** is a primary input and **Y** is a primary output, the input **SI** is connected in the test input **I** of the first scan element of the chain and **SO** is connected with the output **L2** of the last scan element of the chain. The test protocol works as following to initialize the scan-chain:

- Input **SI** is set to the test value;
- Clock **A** is set at one and clock **B** must be zero;

- Clock **B** is set at one and clock **A** must be zero;

- Repeat the above steps *n* times, where *n* is the number of scan elements.

After the scan is initialized, clocks **C1** and **C2** are set to one to execute the test values. Finally, the same protocol to initialize the scan-chain is employed, but now to shift and observe the test values at the test output **SO**. Some logic synthesis tools like Cadence RTL Compiler [Cad11] does not support the LSSD test protocol. DC Compiler from Synopsys [SYN14] is an example that supports LSSD test protocol and can replace the latches registers by the LSSD cell.

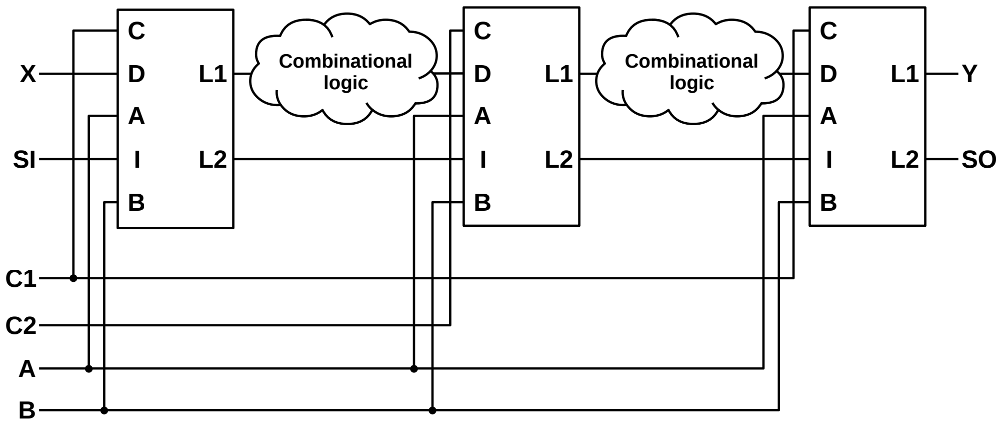


Figure 2.10 – Scan chain using single-latch LSSD approach [WWW06].

2.4.2 LSSD Double-Latch Scan

Another approach to connect the LSSD cell in a scan-chain is the double-latch. Double-latch is an approach used to replace master-slave registers by the LSSD cell, and also is supported by DC Compiler from Synopsys [SYN14]. Figure 2.11 shows the connections used to form the chain using this approach. As in the single-latch approach, **C1** and **C2** are non-overlapping, but in this case, **C2** and **B** can be the same [WWW06], once even in normal mode or test mode, the design has a master-slave behavior.

The inputs **D**, **C1** and **C2** are used in normal mode, and the inputs **I**, **A** and **C2** (the same as **B**) are used in test mode. The **L1** output is not used, the data always pass through **L2**, making the behavior of this approach the same of a master-slave flip-flop. The test protocol is the same of the single-latch approach, described in Section 2.4.1.

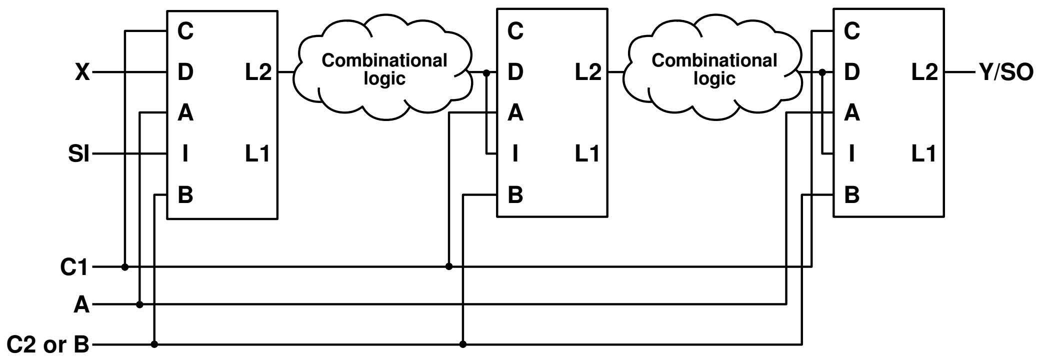


Figure 2.11 – Scan chain using double-latch LSSD approach [WWW06].

2.4.3 LSSD L2* Scan

This approach decreases the area overhead of the LSSD. The main difference of this approach compared to the others two is that latch **L2** (now called **L2***) has an additional port enabled by a clock signal **D*** and an additional clock signal **C***. Figure 2.12 shows the proposed LSSD cell using **L2*** in a register level and Figure 2.13 shows the gate level diagram of an LSSD using **L2*** [ABF94].

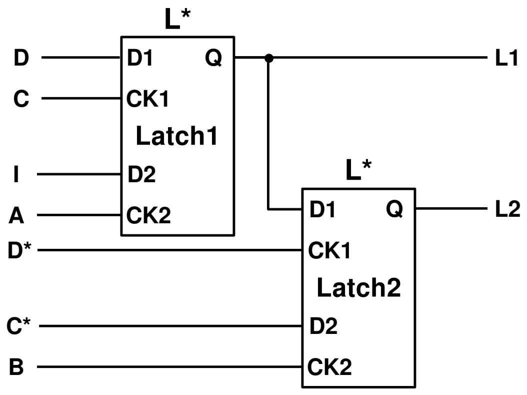


Figure 2.12 – LSSD cell using L2* [ABF94].

In this approach, **A** and **B** are test clocks, as in the previous one. **C** and **C*** are system clocks, **I** is the test input, and **D** and **D*** are functional inputs. **D** is enabled by **C**, **I** is enabled by **A**, **B** shifts the **L1** value to **L2** and **D*** is enabled by **C***.

Figure 2.14 shows the connection of the scan-chain with the **L2*** cell using a single-latch approach. **D** is connected to **N1** output and **D*** is connected to **N2**. As a disadvantage, it is not possible to test combinational logic **N1** and **N2** at same time since it is possible to load just one test vector for each logic at a time.

In normal mode, the latches **L*1** and **L*2** are enabled by the clock **C**, while **L*3** and **L*4** are enabled by the clock **C***. Thus, when **C** is at logic level high, the data in **L*1** and **L*2** are transmitted to the logic **N2**. When **C*** is at logic level high, the data in **L*3** and **L*4** are transmitted to the logic **N1**.

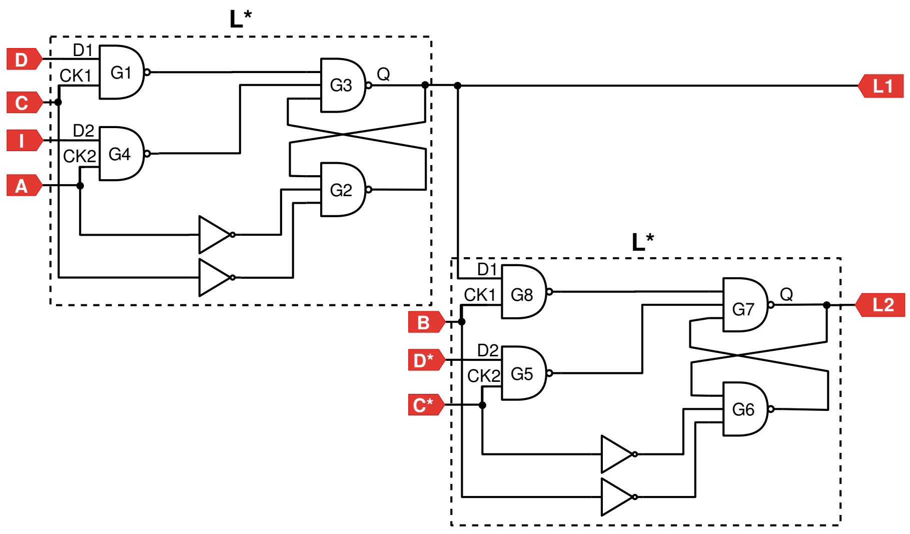


Figure 2.13 – Logic gate diagram of the LSSD cell using L2* [ABF94].

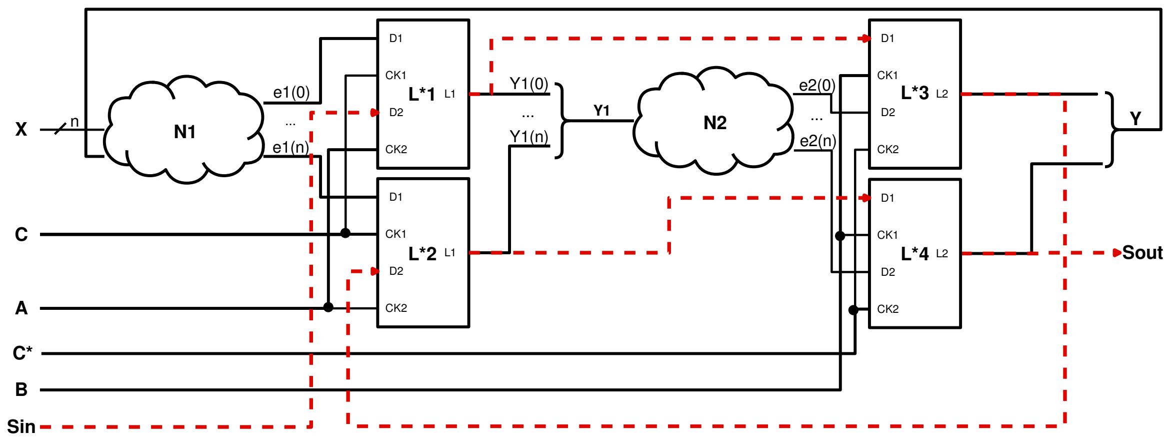


Figure 2.14 – Scan chain using L2* LSSD cells [ABF94]. The red dashed line represents the scan-chain path.

In test mode, **L*1** and **L*2** are enabled by clock signal **A**, and **L*3** and **L*4** are enabled by clock signal **B**. The test protocol works as follow:

- **SI** is connected to input **I** and is set to the test value;
- Clock **A** is set at one and clock **B** must be zero. Thus, **SI** input value is copied from **L*1** to **L*3**, and the **L*2** value is copied to **L*4**;
- Clock **B** is set at one and clock **A** must be zero. Thus, the **L*3** value is shifted to **L*2**, and the **L*4** value is shifted to output **Sout**;
- Repeat the above steps $n/2$ times, where n is the number of scan elements.

After the initialization, the clocks **C** and **C*** must be started to execute the test with the test values. Finally, the same protocol to initialize the scan-chain is used to observe the test values at the test output **Sout**.

2.4.4 Comparison between LSSD Cells and LSSD Approaches

This Section presents a comparison, proposed by the author, between the Muxed-D cell and the LSSD cells, since the literature does not present a similar comparison. Regarding area overhead, the Muxed-D cell has just a MUX as overhead (two ANDs, one NOR, and an inverter). The LSSD has an extra latch (four NANDs) and the overhead control between test input and normal input (two NANDs and an inverter), reaching seven extra gates.

The single latch approach has the biggest overhead compared to the others approaches using LSSD [WWW06]. In normal mode, the single-latch uses only **L1** and **L2** stays in idle mode. In test mode, both latches are used. Thus, the **L2*** approach rises as a potential alternative to reduce the area overhead since it uses just one latch in both normal and test modes.

About pin overhead, the LSSD has more pins than Muxed-D. The LSSD cell adds two clock pins plus two pins to test input and test output, summing up four extra pins. Once the Muxed-D cell shares the output pin to test, it adds the test input and the input selector, summing up two extra pins.

The single-latch and **L2*** approaches have the same number of clock signals, one more than double-latch. The double-latch approach does not need an additional clock to avoid combinational loops, which facilitates the clock tree routing. Table 2.1 resume this information about pin and area overhead. The column **Extra Gates** inform the number of extra gates necessary to implement each test approach. Column **Extra Pins** inform the number of necessary extra pins in each approach.

Table 2.1 – Comparison between the Muxed-D cell and the LSSD cells

	Extra Gates	**Extra Pins**
Muxed-D	3	2
LSSD Single-latch	7	4
LSSD Double-latch	7	3
LSSD L2*	6	6

About performance, the test occurs in the same way in both Muxed-D and LSSD cells, once the initialize, shift and capture operations are the same. The difference is in the control of these operations, once Muxed-D uses an additional signal to control the data selection, while the LSSD uses extra clocks.

The Muxed-D cell has a longer delay than LSSD, once the extra MUX increases the number of logic levels to compute the data, increasing propagation delay. Besides, the LSSD uses latches instead of flip-flops. Latches are memory elements potentially faster than flip-flops, once latches have fewer logic levels in their implementation. Furthermore, in normal mode, the path to the LSSD output uses just the (**L1** latch, as shown in Figure 2.7, while the Muxed-D cell always includes the MUX and the flip-flop delay.

The fanout of the Muxed-D cell is bigger than that of the LSSD, once the Muxed-D shares the output pin to functional and test outputs. The LSSD cell has separate pins to functional output (**L1+** pin) and test output (**L2+** pin), which makes the LSSD potentially faster than Muxed-D, due to the smaller fanout in the output pins.

According to the state of the art [WWW06] [BA02], the single-latch approach has a potential advantage regarding performance when compared to the double-latch approach. In normal mode, the single latch approach uses just one latch in the data-path, which does not occur with the double-latch approach, making single-latch faster than double-latch in normal mode.

The use of **L2*** can be an alternative to increase performance. The data does not need to pass through two latches in its path in normal mode, unlike the double-latch approach. This characteristic makes the **L2*** approach faster than the other approaches in both normal and test modes, except the single-latch approach. However, unlike the approaches single-latch e double-latch, EDA tools do not support automatic **L2*** insertion.

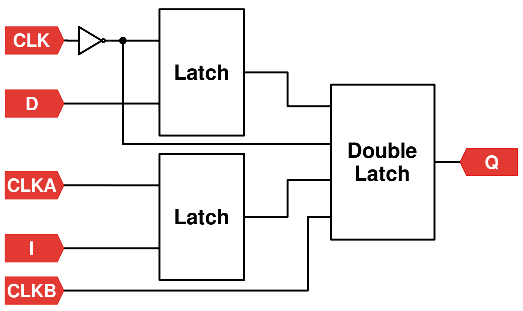
2.5 LSSD Scan Protocol for Flip-flop Memory Elements

As mentioned before, Eichelberger and Williams [EW77] proposed a test protocol for latch-based designs called LSSD. However, when a design mixes latches and flip-flops, it is necessary to have an LSSD compliant scan cell to replace both types of cells. An LSSD compliant cell for flip-flop is illustrated in Figure 2.15(a), called Clocked-LSSD.

The Clocked-LSSD has a D flip-flop behavior in normal mode, which means that the memory elements are edge-triggered. In test mode, the Clocked-LSSD has the standard LSSD behavior mentioned before, which means that the memory elements are level sensitive. Figure 2.15(b) shows the Clocked-LSSD truth table. The "R" means that the system data is registered at clock rising edge. The "*" at the clocks **A** and **B** means that clock **A** must pulse before clock **B**.

2.6 Timing Resilient Circuits

As silicon technology advances into the deep submicron (DSM), it allows performance improvements and energy efficiency, but it also brings concerns such as variations in process, voltage, and temperature (PVT), and reliability issues due to the use of lower voltages and transistors degradation, which can cause a premature failure of the sys-



(a) Clocked-LSSD cell [SYN15b].

	D	CLK	I	CLKA	CLKB	Q
Normal Mode	0	R	x	0	0	0
Normal Mode	1	R	x	0	0	1
Normal Mode	x	x	0	0	0	Q
Test Mode	x	0	0	1*	1*	0
Test Mode	x	0	1	1*	1*	1

(b) Clocked-LSSD truth table [SYN15b].

Figure 2.15 – LSSD test protocol for edge-triggered sequential cells (Clocked-LSSD). The "R" means that the system data is registered at clock rising edge. The "*" at the clocks **A** and **B** means that clock **A** must pulse before clock **B** [SYN15b].

tem [TBW$^+$09]. Delay margins are added to deal with PVT problems and achieve a good yield. However, these additional margins affect system performance. An alternative to minimize these delay margins is the timing resilient design technique.

Timing resilient design techniques allow the circuit to operate with relaxed timing constraints that eventually cause a timing violation. This technique has the capability to recover its normal operating status, even after some internal failure or a fault caused by these violations. For such cases, these techniques use some timing error detection logic and a timing violation recovery mechanism. Besides, timing resilient circuits benefit from the average-case performance. They rely on the fact that errors have a low probability of occurrence and thus the recovery penalties have a small effect on performance. Thus, the error rate is a critical parameter of this kind of design, once timing resilient circuits present a trade-off between allowed error rate and the performance, power and reliability of the circuit. The Razor family is one of the first resilient approaches, and is described in Sections 3.1.1 to 3.1.4.

2.7 Asynchronous Circuits

A synchronous circuit is a design style where the circuit operations are synchronized by a global clock signal, which simplifies the implementation of circuits. This style is used in most of the digital circuits fabricated nowadays.

Asynchronous is a class of design styles that does not have a global signal to synchronization. Instead, the synchronization occurs using local handshaking protocols between circuit blocks. Different from the synchronous style that always processes data, when no handshaking protocol occurs, the circuit remains in idle mode, which means that the component process data just when requested. Asynchronous circuits present potential advantages when compared to synchronous circuits [SF01]:

- Low power consumption, once the circuit only processes data when and where requested;
- High operation speed, once the circuit speed is determined by local latencies, instead of by a global worst-case period signal;
- Reduction of the power for clock tree distribution.

However, asynchronous circuits may also present as drawbacks [BOF10]:

- Can presents more gates than a functionally equivalent synchronous circuit;
- Difficult to test and debug, once exist a lack of test cells and test tools for asynchronous design.
- There is a lack of CAD tools and cell libraries compared to the synchronous approach.

Asynchronous circuits can be designed using different data-encoding schemes and handshake protocols. The choice of handshake protocol and of the data-encoding scheme is called a template [BOF10]. Currently, there are two main families of asynchronous templates, Quasi-delay-insensitive (QDI) and Bundled-data (BD) [BOF10].

QDI uses the multi-rail data encoding, where a completion signal is embedded into the data representation. One example is the dual-rail protocol that uses two wires to represent one bit of data and the corresponding encoded request signal. QDI provides relaxed timing constraints, but the circuits from this family of templates are usually larger and have higher power consumption when compared to BD, and can have an area overhead 3.45 times greater when compared to a synchronous approach [CVG07].

BD templates use standard Boolean encoding for data, and separate request and acknowledge wires are bundled with respective data signals to provide synchronization, as illustrated in Figure 2.16. A BD template can employ either 2-phase or 4-phase handshake protocols. Figure 2.17 shows the waveform for a 4-phase protocol. This protocol behaves as follows:

1. The sender generates the data and sets the request to high;
2. The receiver stores the data and sets the acknowledge to high;

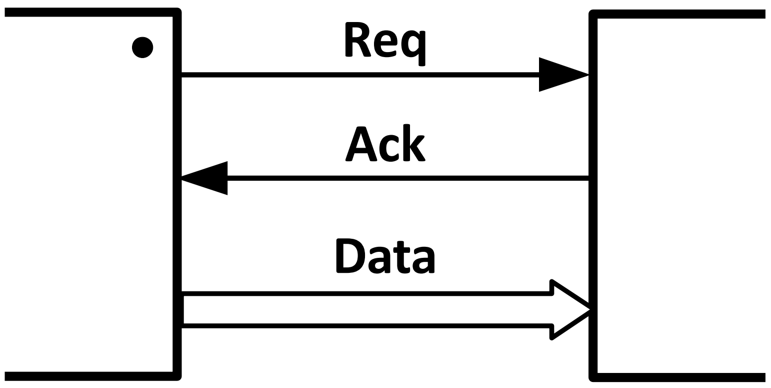


Figure 2.16 – Bundled-data block diagram [Spa01].

3. The sender sets the request to low;
4. The receiver sets the acknowledge to low, allowing the sender to transmit new data.

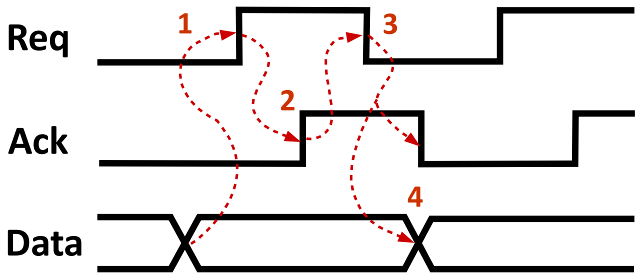


Figure 2.17 – 4-phase handshake protocol waveform [Spa01].

The 4-phase protocol has the disadvantage of the additional return-to-zero (RTZ) transitions, that cost unnecessary time and energy, and potentially reduce the performance of the circuit. The 2-phase protocol is shown in Figure 2.18. This protocol waits for an event in the request and acknowledge signals, and behaves as follows:

1. The sender generates data and switches the request signal;
2. The receiver stores the data and switches the acknowledge signal, allowing the sender to transmit new data.

This protocol avoids the RTZ transitions and improves the performance of the circuit. However, the 4-phase requires less complex control circuitry.

BD control blocks can be implemented with different design styles. The main templates available in the state of the art use 2-phase handshake protocols. The concept of micropipeline was introduced in [Sut89], which uses special capture-pass latches that are event controlled and capable of sensing transitions at their inputs. Another approach is the latch-based design Mousetrap for high-performance asynchronous designs, leveraging

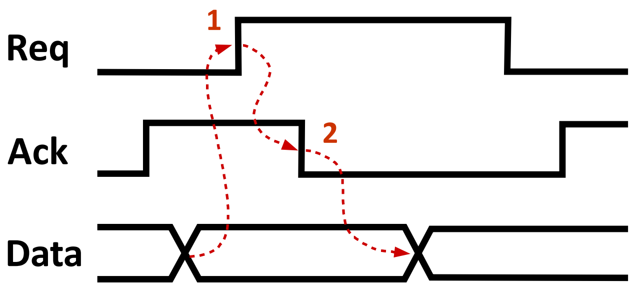


Figure 2.18 – 2-phase handshake protocol waveform [Spa01].

the fact that latches present less logic complexity than flops and are level sensitive. This template implements a 2-phase BD circuit using level-sensitive latches and XOR gates.

A more recent template is Click [PtBdWM10] that instead of using latches and C-elements, conventional asynchronous circuits components, they proposed a 2-phase BD template that only uses edge-triggered cells. This approach, due to the use of edge-triggered, has support to scan insertion using the Muxed-D cell, making this template a possible choice for DfT insertion in asynchronous BD design.

2.8 Blade Template

Synchronous resilient circuits are an alternative to eliminate worst-case safety margins, as mentioned in Section 2.6. However, this kind of circuits are prone to failure, if metastability occurs [BCC+14], or it presents a high recovery penalty. Blade [HTMH+15] is an asynchronous template for 2-phase bundled-data (BD) circuits that helps to overcome some problems in current synchronous timing resilient architectures. Figure 2.19 illustrates the basic Blade architecture. It consists of a controller, two configurable delay lines (δ and Δ), and an error detection logic (EDL). The controller uses a BD channel **L**/**R** to communicate with others pipeline stages.

The value δ is the delay used to control the moment that data at the output of the combinational logic can be sampled and propagated through the EDL. The Δ is a delay value used to define the amount of time that the latch is transparent and the timing resilience window (TRW), which is a period where errors are allowed to happen. The values of δ and Δ must be designed such that their sum is sufficiently large to cover the longest critical path in the stage.

The signal **Err** is used to flag a time violation. When this signal is set to high, the controller then communicates with its right neighbor using a speculative handshake through the error channel **RE**/**LE**. To recover from the timing violation, the next stage uses the time borrowing property of the latch, keeping it transparent until the correct data is propagated through the combinational logic.

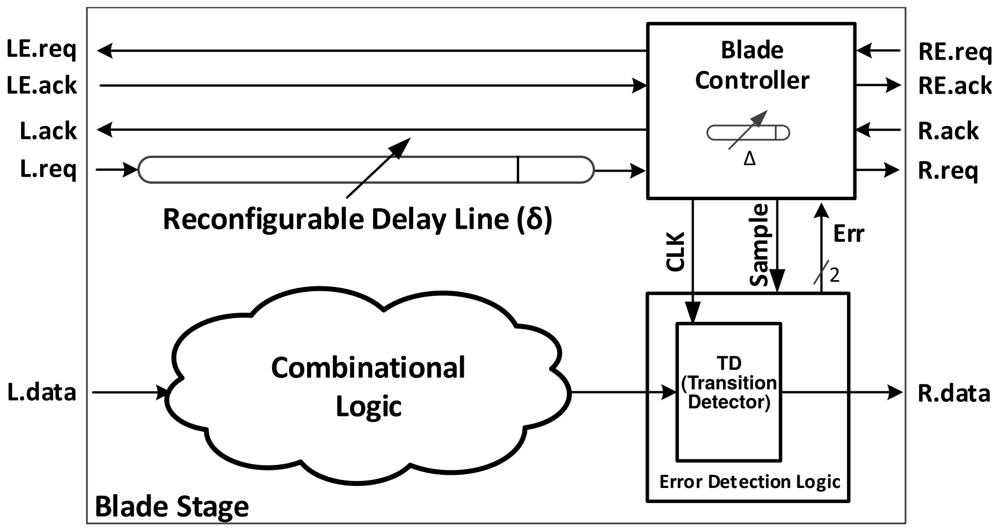


Figure 2.19 – Blade Architecture [HTMH+15].

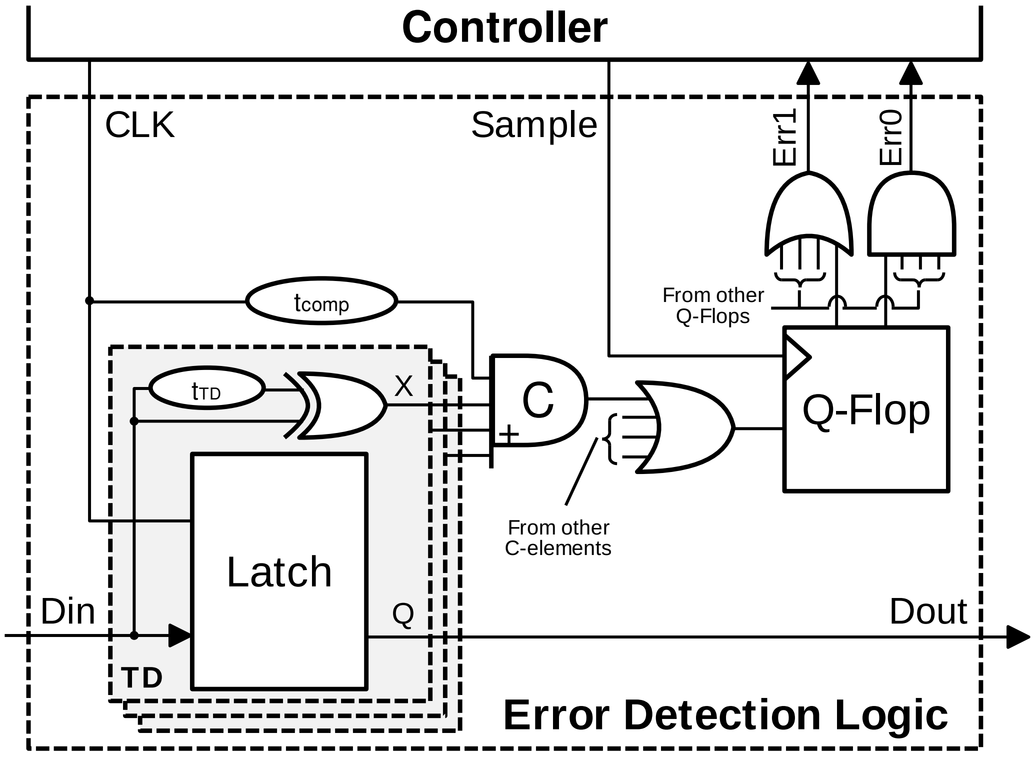


Figure 2.20 – Blade EDL [HTMH+15].

Figure 2.20 details the error detection logic. The design consists of a latch with error transition detector, called Transition Detector (TD), an asymmetric C-element and a Q-Flop [RMCF88]. The asymmetric C-element stores any violation detected when **CLK** is high. This C-element switches to logic level low when **CLK** is low and to logic level high only when **CLK** and the XOR output are high. The output of the C-element is sampled at the end of the TRW by the Q-Flop. The Q-Flop uses a metastability filter that ensures safe operation against metastability.

The signal Err is a dual-rail signal that stalls the controller until the TDs outputs are stable. The delay element t_{TD} defines the transition detector pulse width, while t_{comp} is the compensation delay added to ensure that a transition before the rising edge of **CLK** is not considered a violation. The error detection logic is only used in the critical paths to minimize the area overhead since these paths are more susceptible to timing violations than other paths of the design.

The Blade template has a synthesis flow that converts a synchronous Flip-flop based design into an asynchronous design [HTMH+15]. First, this flow converts the Flip-flop based design into a latch-based version by changing the edge-triggered flip-flops into two latches. After this conversion, the retiming technique is applied to reduce the critical path, which leads potentially to a reduction in the number of inserted EDLs. The last step of this flow is to convert the latch-based retimed circuit to an asynchronous circuit with the Blade template.

A 3-stage version of the Plasma microprocessor[1] (based o MIPS-I instruction set) targeting a 28nm FD-SOI technology is used as case study. Results show that the overall area overhead of the Blade version is 8.4% when compared to the original synchronous design. The standard benchmark CoreMark was used to evaluate its performance. The performance of the Blade version increased 19%, with an average frequency of 793MHz.

2.9 Q-Flop

Metastability is a phenomenon where a circuit element assumes an undetermined value between logic zero and one. This value can be propagated through the entire design, making the circuit produce a wrong value at its output. Register elements can easily be made metastable just toggling the clock and the input data at the same time [Gin11]. Thus, it is necessary to avoid the propagation of this wrong value through the entire design.

The original Q-Flop architecture, illustrated in Figure 2.21, was proposed by Rosenberger et al. [RMCF88] as a metastability-free memory element. Table 2.2 shows the logic behavior of the Q-Flop in a truth table. As described by the authors, when the clock input is at logic level 0, the Q-Flop outputs are also at logic level 0. At the rising edge of the clock input, the Q-Flop has a type D flip-flop behavior, sampling data at the clock rising edge. For its memory element, the Q-Flop relies on a pair of cross-coupled NANDs (as in a traditional SR latch) driven by two control NANDs that ensure the data is only sampled at the rising edge. It has a single-rail input, as in conventional digital circuits, but the output is dual-rail, as commonly used in non-synchronous digital circuits, and guaranteed to be metastability-free by a metastability filter.

[1]https://opencores.org/project,plasma

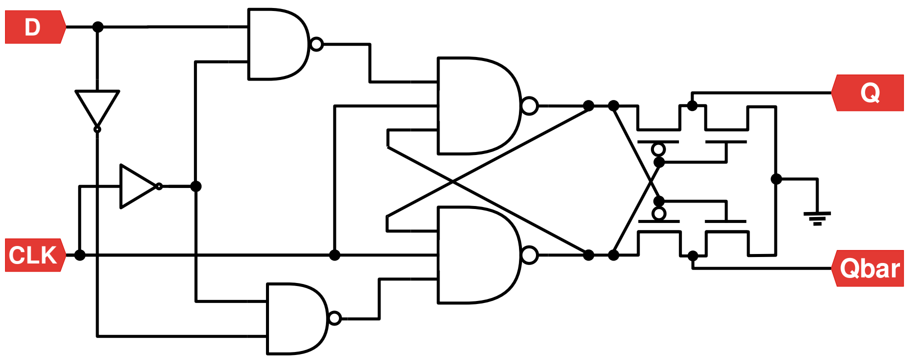


Figure 2.21 – Q-Flop cell [RMCF88].

Table 2.2 – Q-Flop truth table

CLK	D	Q	Qbar
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	0

The initial state of the filter circuit has the node **N1** at logic level 1, driving the output **Q** to 0 using the NMOS transistors **T1**. When **N1** changes to logic level 0, the PMOS transistor **T0** is activated and propagates a logic one value at the other input **N0** to the output **Q**. An equivalent behavior can be observed for output **Qbar**. Note that this behavior happens only if **D** respects its setup and hold timing constraints with respect to **CLK**, as in a conventional flop.

However, in the event of **D** switching too close to a transition in **CLK**, and violating its constraints, the element can go metastable. For example, let us assume that the setup time of a Q-Flop is 50 ps and that input **D** switches from 0 to 1 20 ps before input **CLK** switches from 0 to 1, causing a violation in the setup constraint. In this case, the component may sample a transitioning signal that will drive nodes **N0** and **N1** to an undefined voltage level, between the levels that define logic 0 and 1. For simplicity sake, let us assume that both nodes assume $V_{dd}/2$ V. This value is enough to partially activate NMOS transistors **t1** and **t3**, keeping the outputs at logic 0, even with the small current produced in this case. Besides, PMOS transistors **t0** and **t2** will present $V_{dd}/2$ V at their gate and source terminals, not activating the transistors and not driving current to the outputs.

Eventually, for some internal or external event, one of the internal nodes (say **N0**) will start to drive to a value greater than $V_{dd}/2$, and the PMOS **T0** will start to conduct. At the same time, the cross-coupled NANDs will drive node **N1** to a value smaller than $V_{dd}/2$ when the node **N0** has a value greater than the $V_{dd}/2$. Thus, the next circuit state has **N1** at 0 and **N0** at 1, switching the **Q** output to level logic 1 and keeping **Qbar** at 0 without glitches and errors. A similar scenario can happen if node **N1** pulls to a value greater than $V_{dd}/2$ while in a metastable state, in which case the Q-Flop will resolve to a **Qbar** at logic 1 and

Q at 0. The importance of this component relies on the fact that while it is resolving these metastable states, the outputs are kept at 0 by the metastability filter, preventing metastable signals to propagate.

3. STATE OF THE ART

This Chapter presents an overview of the current literature on resilient circuits (Section 3.1), test of resilient circuits (Section 3.2 and Section 3.3) and a conclusion about both topics (Section 3.4).

3.1 Resilient Templates

This Section describes some of the resilient templates present in the literature, more specifically the Razor family. This family of resilient circuit was chosen to show the presence of synchronous resilient templates in the literature and the techniques used in this kind of circuit, which are necessary to understand the scope of this work.

3.1.1 Razor

As mentioned before in Section 2.6, delay margins are added in synchronous designs to deal with PVT variations and achieve good yield, which affects the system performance. The Razor synchronous architecture [EKD+03] is a technique to eliminate worst-case safety margins in the clock period by using a novel voltage management technique, where the processor operates with dynamic voltage scaling (DVS). This technique replaces the flip-flops on critical paths of the design by Razor flip-flops, illustrated in Figure 3.1.

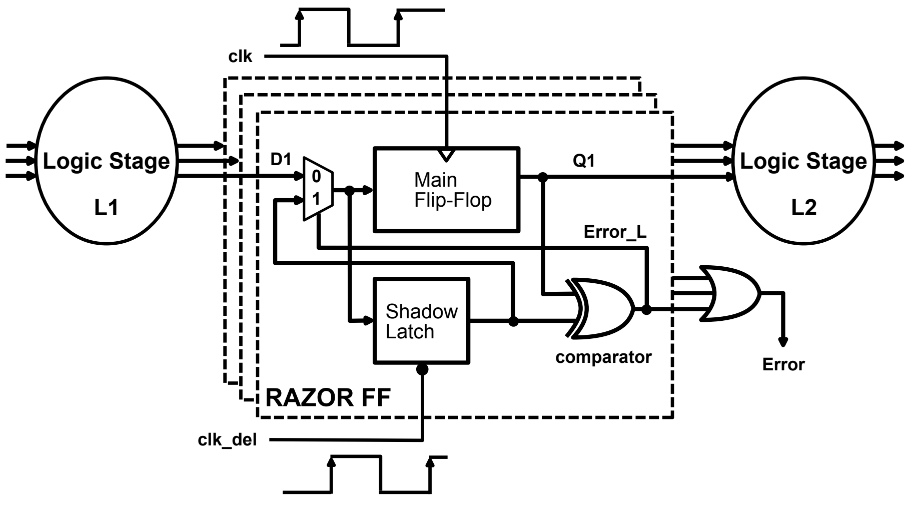


Figure 3.1 – Razor flip-flop [EKD+03].

The **clk** signal is designed with less pessimistic time margins and it controls the **Main Flip-flop**. The **Shadow Latch** is controlled by a delayed clock (**clk_del**), that is designed to meet the latch setup time in worst-case situations, guaranteeing its data validity even when the **Main Flip-flop** timing is violated. To detect a timing violation, the Razor flip-flop compares the values stored in the **Main Flip-flop** and in the **Shadow Latch**. If the values are different, then an error is detected, and the output **Error** changes its value to logic level high. The error signal causes the circuit to redirect the valid data stored in the latch to the flip-flop with a one cycle penalty. Besides, the previous stage must be stalled, and data at the following stage must be flushed.

By replacing the flip-flops on critical paths of the design by Razor flip-flops, the circuit can scale down the supply voltage to the point of the first failure for a given frequency, eliminating all margins due to global and local PVT variations, saving energy. The supply voltage can be scaled lower than the first failure point, achieving additional energy savings, if the application tolerates a low error rate. The error rate is the fraction of the input vectors that do not complete within the clock period.

A set of simulated benchmark experiments shows that an error rate of 1.5% allows an average energy savings of 41% with a maximum performance slowdown of 6%. A 64-bit Alpha processor Was manufactured in 0.18 µm technology using the Razor technique, that operates at 200MHz and a delayed clock for the shadow latch set at 1/2 from the system clock. 192 flip-flops Were replaced by Razor FFs out of a total of 2408. Razor presents a power overhead up to 3.1% of the total power due to the error detection and correction circuitry in error-free operation. Area overhead results are not presented in this work.

3.1.2 RazorII

RazorII [DTP+09] is a simplification of the original Razor described in Section 3.1.1. It removes the error recovery mechanism, once the error rate at the point of the first failure is in the order of 1 error in 10 million cycles, which makes error correction energy negligible. Therefore, RazorII FF performs only error detection and error recovery takes place through architectural replay.

The RazorII flip-flop uses a single latch combined with a transition detector. If a transition occurs after the rising edge of the clock, while the latch is transparent, the RazorII flip-flop detects and asserts the error signal. When an error occurs, all pipeline is flushed, and the failing instruction is re-executed. In the case of successive failures, the clock frequency is reduced by half during eight cycles. Figure 3.2 shows the RazorII flip-flop diagram.

A 64-bit 7-stage Alpha processor was manufactured in 0.13 µm technology featuring RazorII as a case study. The results show that a energy savings up to 37.4% were obtained with an error rate of 0.04% compared to worst-case, when the supply voltage is set to ensure correct operation. RazorII circuitry presents a power overhead up to 1.2% of the

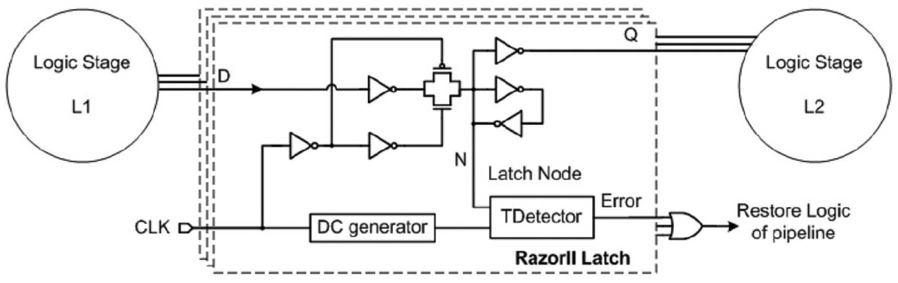


Figure 3.2 – RazorII flip-flop [DTP+09].

total power due to the error detection and correction circuitry in error-free operation. Area overhead results are not presented in this work.

3.1.3 Razor-Lite

The Razor-Lite approach [KKF+14] uses a side-channel detection strategy compatible with standard D flip-flops to reduce additional error detection circuitry. This approach is used in well-balanced pipelines implementations, which present a significant number of critical paths and need a large number of flip-flops with timing violation detectors.

Figure 3.3 illustrates a conventional flip-flop design and the added logic required by Razor-Lite. The added logic connected to the virtual VDD (**VVDD**) and virtual VSS (**VVSS**) rails of the flip-flop acts as a transition detector. Under normal operation, after the rising edge of the clock, **VVDD** and **VVSS** are kept charged and discharged, respectively, and no error is detected. A timing violation is detected if **D** changes after the rising edge of the clock. If **D** transitions to logic level low, **VVDD** is discharged through node **DN**. If **D** transitions to logic level high, **VVSS** is charged through node **DN**. **VVDD** and **VVSS** are restored to their original state after the falling edge of the clock.

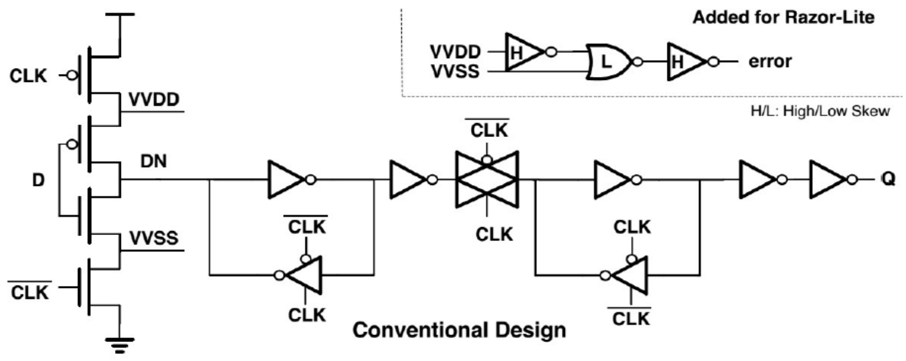


Figure 3.3 – Razor-Lite flip-flop [KKF+14].

A 64-bit 7-stage Alpha architecture pipeline demonstrates the use of Razor-Lite. The processor was prototyped in 45 nm SOI CMOS technology, and all 492 of decode and execute registers, which are the processor's critical path, were replaced by Razor-Lite registers. Error outputs are grouped via an OR-three and recorded at a pipeline register. After the rollback, the clock frequency is reduced by half during four cycles to allow the affected instruction to execute correctly. The total penalty for an error is 11 cycles. Razor-Lite circuitry increased the area in 4.42% and power by 0.3%. An 83% peak energy efficiency improvement was observed, compared to the baseline processor.

3.1.4 Bubble Razor

Bubble Razor [FFK+13] is an architecturally independent approach to timing error detection and correction. This approach uses a two-phase latch-based data-path instead of a flip-flop based data-path.

The error detection circuit presented in Figure 3.4 is similar to the original Razor, which uses a shadow latch to detect timing violations. Local stalling is used to perform error correction. When a timing violation is detected, an error signal (bubble) is propagated to neighboring latches, preventing them from becoming transparent and disabling the clock by a clock gating logic at each stage. This clock gating adds time for the correct data to arrive at the latch that identified the timing violation. Once a neighbor latch receives a bubble, it is propagated to the next stage. When loops are present, the authors propose a bubble propagation algorithm to avoid indefinite bubble propagation.

An ARM Cortex-M3 microcontroller with Bubble Razor was implemented using a 45nm SOI CMOS technology. A flop-based design was converted to a two-phase latch-based implementation. The error detection logic was added to all latches, resulting in 87% area overhead. When considering the combination of eliminating margins and running beyond the point of the first failure, besides replacement of flip-flops by latches in the design, this implementation enables 100% throughput increase or 60% energy reduction when compared to the microcontroller with worst case timing margins.

3.1.5 SafeRazor

Even with the advantages of the Razor family of synchronous design templates to deal with timing errors, all known implementations are prone to failures due to the non-deterministic timing behavior introduced by metastability. SafeRazor [BCC+14] [CBC+15] is a Razor-based circuit design technique that combines the Razor principle with stoppable clocks in a globally asynchronous locally synchronous (GALS) design. This approach avoids any timing failure due to metastability and does not require any checkpoint mechanism or pipeline restarting logic, other than the usual auxiliary latch to store the valid data.

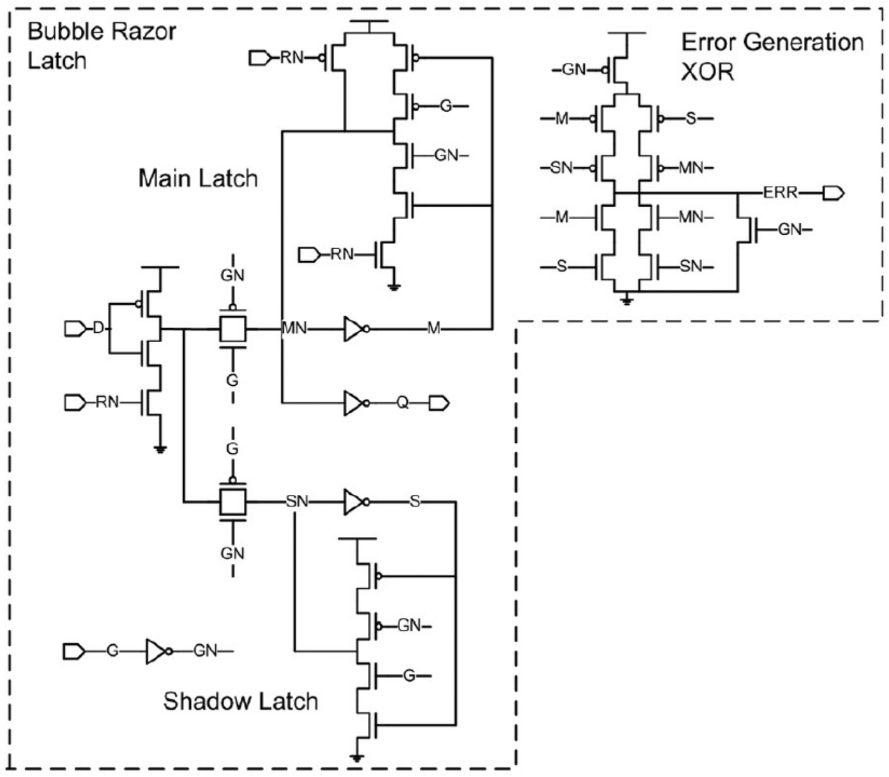


Figure 3.4 – Bubble Razor flip-flop [FFK+13].

Figure 3.5 shows the locally synchronous SafeRazor module. The **COMB LOGIC** block is identical to the logic blocks presents in conventional circuits, which means that it processes the combinational logic of the design. The **STORAGE** block has the memory elements and the logic to detect metastability and speculation errors. **RING OSCILLATOR** block contains the adaptable delays to generate the clock under different operating modes.

The operating modes are:

- **Normal mode**: In this mode, the **STORAGE** block behaves as a standard flip-flop. Setup and hold constraints are not violated, and the data at shadow latch is correct;
- **Operation with metastability**: In this mode, if metastability is detected, the **MD** component extends the cycle period until metastability is resolved;
- **Operation with error detection and correction**: In this mode, the error signal generated at the **error detecting FF** selects one of the delays in the MUX of the ring oscillator (**Err** or **No Err**). The cycle period is extended by the extra delay **d4** to allow the circuit to recover from metastability.

A SafeRazor circuit with 3 GALS islands, each containing a pipelined multiplier with four stages was implemented to show the results. Also, a standard flip-flop version and a Razor version were implemented to compare with the SafeRazor version. The designs were synthesized with Synopsys Design Compiler using a 90 nm standard cell library. The

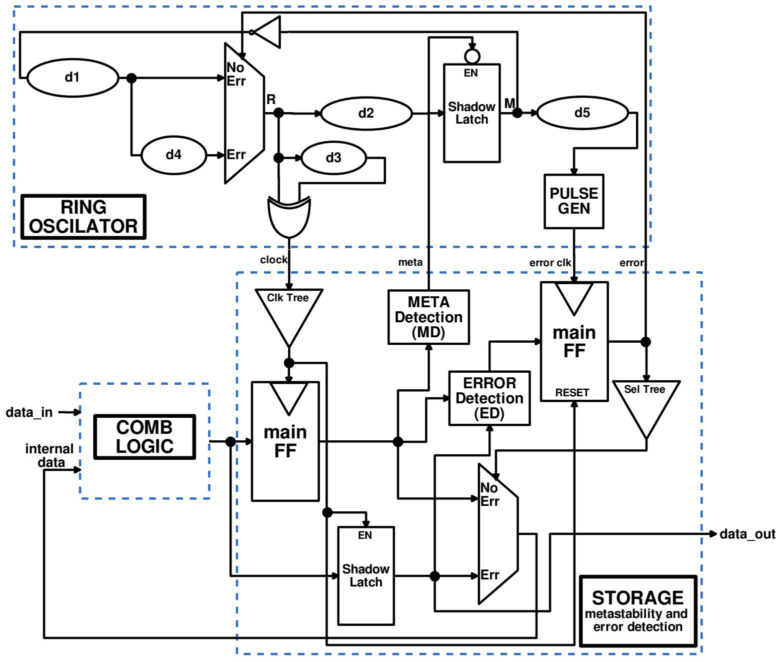


Figure 3.5 – SafeRazor circuit blocks [CBC+15].

physical layout of the netlist was performed using the Cadence Encounter tool. Compared to the synchronous circuit, the total area overhead for Razor is 27.7%, while in SafeRazor this overhead can increase more 15% due to the ring oscillator.

3.2 Testing Timing Resilient Templates

This Section describes approaches used to test timing resilient templates, with focus on scan cells/components used to apply the scan-chain technique at timing resilient architectures.

3.2.1 Scan Razor flip-flop

As mentioned in previous Sections, Razor is a well-known design technique used to deal with timing errors. However, as described in Section 3.1.5, the area overhead is high, and it makes difficult to insert DfT in the Razor design since it will increase the area overhead even further.

Anastasiou et al. [ATA15] propose to reuse the Razor topology to achieve low power scan testing operations, as illustrated in Figure 3.6. This technique is a viable solution off-line AND low-power testing requirements. The work also explores the ability to apply this technique to at-speed scan testing.

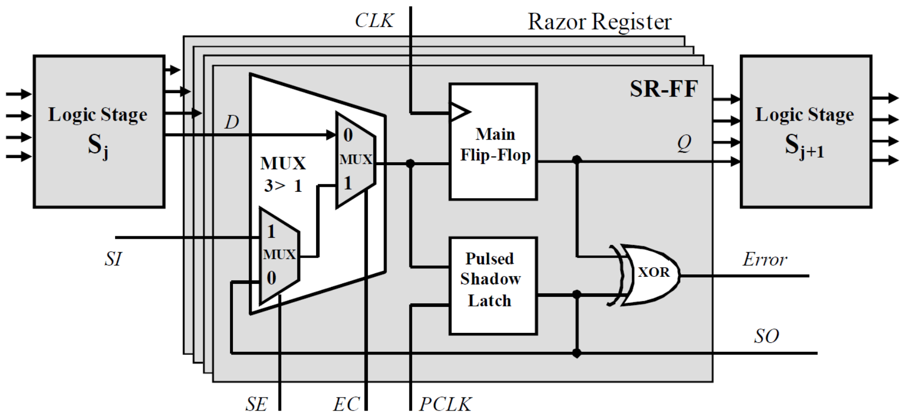


Figure 3.6 – Scan Razor flip-flop [ATA15].

The scan Razor flip-flop (SR-FF) topology has in the feedback path of the circuit an additional multiplexer controlled by the signal **SE**. The Shadow Latch is replaced by a pulsed latch synchronized by the signal **PCLK** to support the proposed scan operations. A pulser circuit [SP11] generates the **PCLK**, and its rising edge of the **PCLK** signal is delayed according to the corresponding edge of the **CLK** signal.

The SR-FF has three operation modes:

- **Normal mode**: In this mode, **SE** and **EC** must be at logic level low. The data at the **D** input are propagated to the Main flip-flop (**MFF**) and to the Pulsed Shadow Latch (**PSL**). This mode is the same as in the standard Razor approach;

- **Error correction mode**: To enable the error correction mechanism, it is necessary to set **SE** to logic level low and **CE** to logic level high, storing the value of the **PSL** in the **MFF**. This mode, as the normal mode, is the same as in the standard Razor approach;

- **Scan mode**: In this mode, **SE** and **EC** must be in logic level high. The test data input **SI** is propagated to the **MFF** and the **PSL** while the test data stored in the **PSL** is propagated through the test data output **SO**. For shift operation, the **PSL** is used for the scan-in/out of test data. Only the **PCLK** signal is active in the shift phase, and each pulse propagates the test data from the **SI** to **SO** output.

Experimental results using ISCAS89 benchmarks show that the scan power consumption of this approach is reduced compared to the classic scan-chain approach, since the signal transitions at the inputs of the combinational logic are eliminated during the scan testing operations, reducing the dynamic power of the circuit.

3.2.2 Timed flip-flop

Floros et al. [FTK08] proposed as scan flip-flop design that provides timing error detection/correction capabilities called Timed flip-flop. Besides, the work also presents a time dilation scan architecture that is suitable for concurrent error detection/correction and off-line testing. This architecture also offers concurrent multiple error detection and correction at the small penalty of one clock cycle delay at the normal circuit operation for each error correction.

Figure 3.7 shows the Timed flip-flop architecture, where **D** is the normal data input, **Scan_EN** is the test enable signal and **Q** is the data output. The XOR gate is used to compare the data at the **M** node and the **Q** output of the **Main Flip-Flop** for error detection. Also, the two MUXs gates and the feedback path from the **M** line to the input of the **MUX-B** forms a register element that holds the valid data for error correction, called **MUX-latch**. The **MUX-latch** is enabled by the signal **Memory**, which is controlled by the signal **Mem_CLK**. When **Memory** is at logic level high, the register **MUX-latch** is opaque. When **Memory** is at logic level low, the register **MUX-latch** is transparent.

The Timed flip-flop can be divided into three operation modes:

- **Normal mode**: In this mode, **Scan_EN** must be in logic level low. The test data input **D** is propagated to the **Main Flip-Flop**, and propagated to the output **Q** when the **CLK** is activated. This behavior is similar to the standard flip-flop;
- **Error detection/correction mode**: In this mode, **Scan_EN** must be in logic level low. When an error is detected, the output **error_F** change its value to a logic level one, paralyzing the circuit by one clock cycle, and the value registered in the **MUX-latch** fed the main flip-flop. When the circuit recoveries from the error, the output **error_F** change its value to logic level zero and the **Main Flip-Flop** still fed the **D** input.
- **Scan mode**: In this mode, **Scan_EN** must be in logic level high. The test data input **Scan_IN** is propagated to the **Main Flip-Flop**, and propagated to the scan output when the **CLK** is activated. This behavior is similar to the standard Muxed-D component presented in Section 2.2.

The Timed flip-flop has a performance penalty less than 4%. According to the authors, the design approach is characterized by low silicon area requirements, with a reduction of about 24% when compared to the Razor flip-flop area [EKD+03].

An extension of the Timed flip-flop was proposed by Valadimas et al. [VFT+14] that concerns with metastability, illustrated in Figure 3.8. The value at the node **M** and the **CLK** signal can transition at the same time, causing a setup violation. It produces a wrong value at the **Main Flip-Flop** output, making the XOR gate not capable of providing

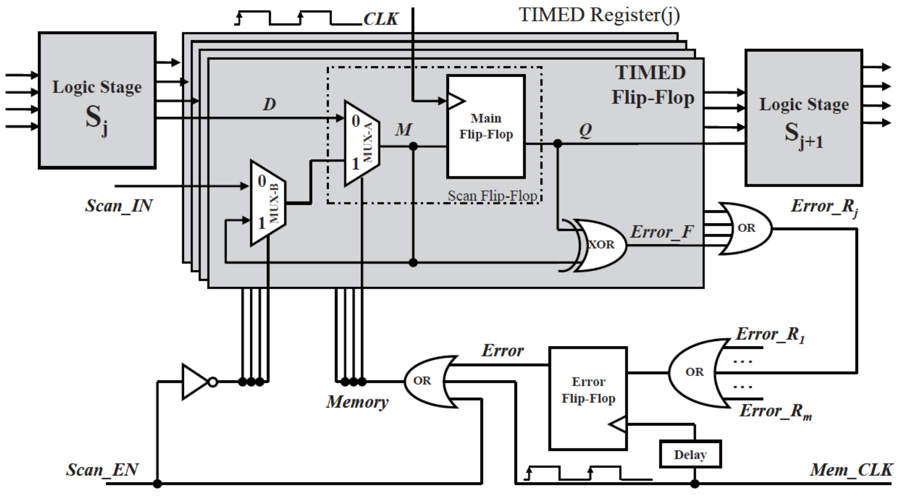


Figure 3.7 – Timed flip-flop [FTK08].

a reliable comparison between the data at the input and the output of the **Main Flip-Flop**, compromising timing errors identification. Thus, a metastability filter is added at the output to avoid the XOR gate comparison problem.

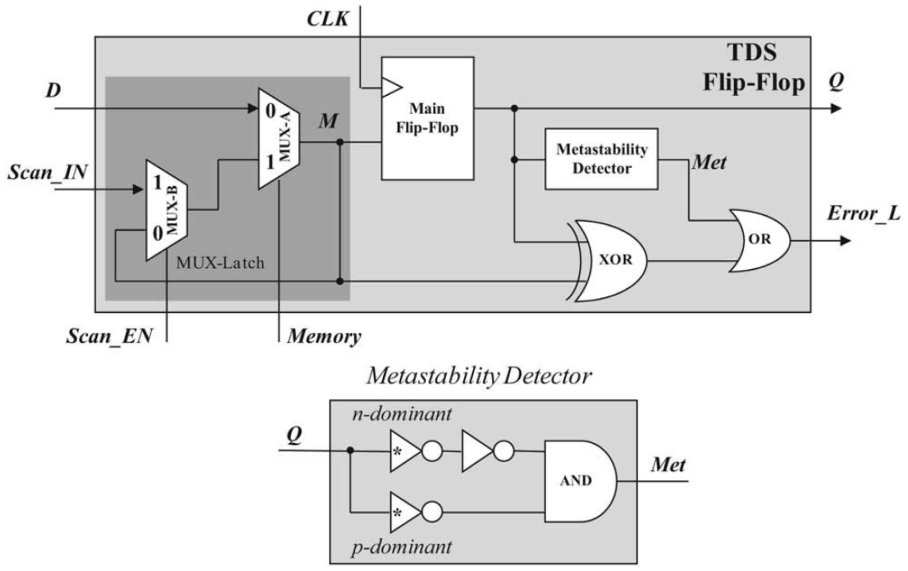


Figure 3.8 – Timed flip-flop [VFT+14].

3.3 Testing Soft-Error Resilient Templates

This Section describes approaches used to test soft-error resilient templates. Soft-errors can occur in a circuit when it is exposed to radiation, like cosmic rays an neutrons particles. This exposition can change the memory elements values, generating erroneous outputs [JOC07].

These works are evaluated in this dissertation due to the similarity with timing resilient approaches. Soft-error resilient circuits also use techniques as shadow logic and comparator components to identify and correct errors. It is possible to use some of these soft-errors resilient approaches for timing resilient operation adding buffers or delays at the components, allowing to capture timing violations. Also, it is possible to combine both soft-error and timing techniques to improve the circuit. As in Section 3.3, the focus of this Section is to present works based on scan cells/components.

3.3.1 Scan Approach To Built-In Soft-Error Resilience

Kuppuswamy et al. [KDF+04] present a scan cell that provides structural testing using automated test pattern generation tools, functional testing using signature analysis and efficient post-silicon debug. Figure 3.9 shows this scan cell, which is divided in two portions: **system flip-flop** and **scan portion**. This scan cell has two operation modes:

- **Normal mode**: In this mode **SCA**, **SCB**, **UPDATE**, and **CAPTURE** are at logic level low. The system data input **D** is propagated to the output **Q** when the **CLK** is activated, similar to the standard flip-flop;
- **Scan mode**: In this mode, clocks **SCA** and **SCB** are used to shift the test pattern into latches **LA** and **LB**. Next, the **UPDATE** clock is applied to move the contents of **LB** to **PH1**, to the test pattern be registered in the system flip-flop portion. After, **CLK** is applied to captures the system response from the test pattern, and the **CAPTURE** signal is applied to move the contents of **PH1** to LA. Finally, **SCA** and **SCB** are used to shift the test pattern result to the **SO**.

Besides, if the **CAPTURED** signal is forced to the logic level high, the **scan portion** is converted to a shadow register of the **system flip-flop**. Thus, it is possible to capture soft-error (which can be a single-event upset or a single-event transient) in the design comparing the outputs **Q** and **SO** adding some comparison circuitry.

Mitra et al. [MSZ+05] modified this scan cell approach to reduce the soft-error rate. This work adds an XOR gate to compare **Q** and **SO** and detect the soft-error. A scan-chain assuming that 25% of the flip-flops are protected from soft-errors was used to evaluate the system-level impact of soft-error of the proposed scan cell. The results show a reduction of more than 20 times compared to the error rate for an unprotected flip-flop, with an area overhead of 0.30% and a power overhead of 5%.

Elakkumanan et al. [EPS06] present a novel single-event transient mitigation scheme for flip-flops that also is based in the scan cell presented by Kuppuswamy et al. [KDF+04]. In this paper, the time redundancy principle is used to achieve single-event transient tolerance.

Time redundancy is a technique where a regular input of a flip-flop is delayed and applied as the second input to the same flip-flop. Simulation results for a stand-alone flip-flop and ISCAS benchmark circuits in 70 nm predictive technology show that the area overhead of the technique is 70.83% and the power overhead is 25%.

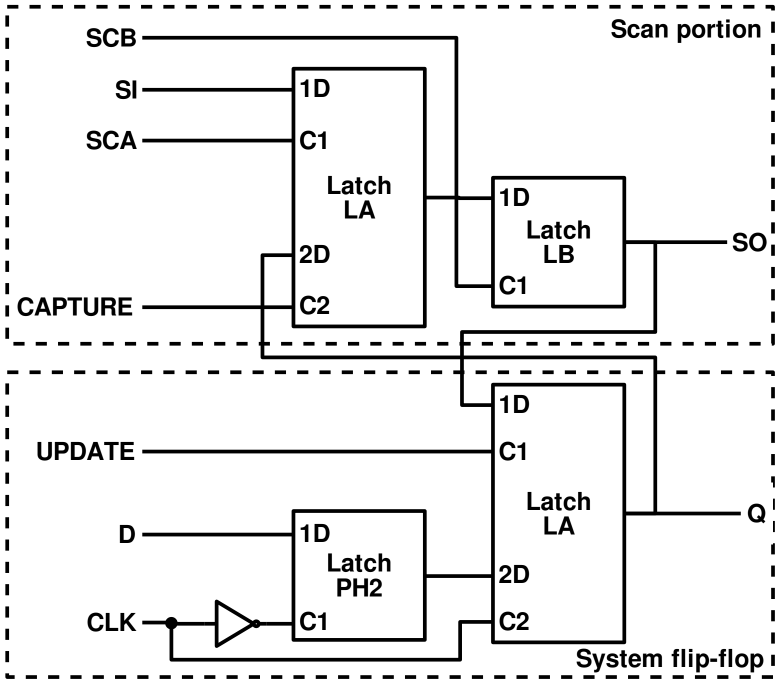


Figure 3.9 – Scan cell proposed by Kuppuswamy et al. [KDF+04].

3.3.2 Scan Approach To Single-Event Upset Detection, Correction, and Monitoring

Drake et al. [DKM05] proposed a scan flip-flop with single-event upset detection, correction, and monitoring capabilities. Figure 3.10 shows the proposed scan component. An extra latch **SL** is added in the master-slave register to form triple redundancy of the stored value. A voter on the output ensures correct data operation by analyzing the data stored at the **MA** and **SI** registers. The **Maj** component detects when one of the latch stages has changed its value. When an error occurs, the output of the voter is stored back into all of the latches, correcting the error.

This scan cell has three operation modes:

- **Normal mode**: In this mode the system data input **DIN** is propagated to the output **DOUT** using the clocks **C_MA** and **C_SL**;
- **Scan mode**: In this mode, **Ctrl** must be in logic level low. Clocks **C_SC** and **C_SL** are used to shift a test pattern into latches **MA** and **SL**. The **Maj** component determines the value to be passed to the **SOUT**.

- **Monitor mode**: This mode is enabled when the scan operation is finished, and the control signal gives the error detection circuit control of the clock inputs to the latches. The error detecting circuit monitors the three latches to analyze if they have the same stored data. If an error is detected, the signal **ERROR** change to logic level high, making the latches transparent and storing the **Maj** value back into the upset latch. When the error is corrected, **ERROR** is at logic level low, and the latches return to their opaque state.

Simulations show that this scan flip-flop with single-event upset detection is 34% faster than the triple-redundant flip-flop, which is an approach that uses three flip-flops and a voter circuit, while dissipating equivalent power. However, the use of triple-redundant latches to achieve single-event upset tolerance results in an area 3.71 times larger than a standard flip-flop.

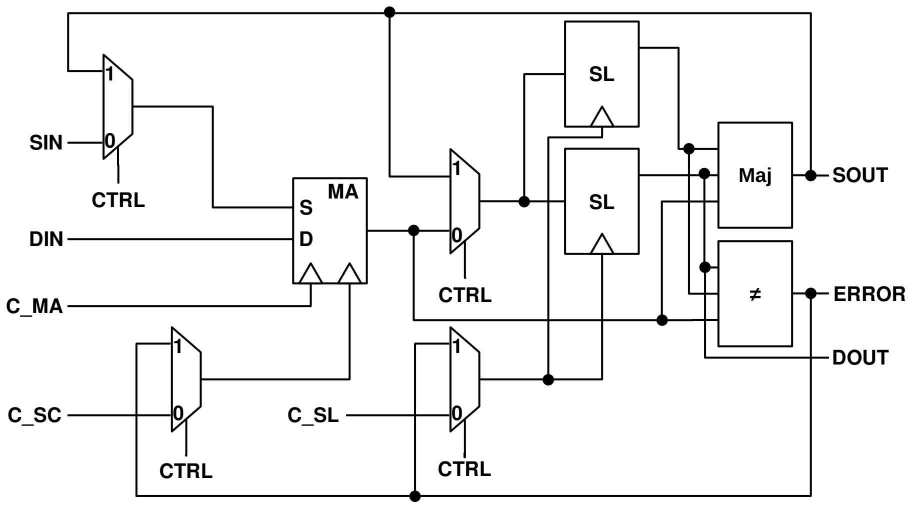


Figure 3.10 – Scan flip-flop with single-event upset detection. [DKM05].

3.3.3 XSEUFF Scan Cells

Jagirdar et al. [JOC07] proposed two register approaches that provide tolerance to single-event upset and single-event transient effects, called **XSEUFF 1** and **XSEUFF 2**. Figure 3.11 show these components.

XSEUFF 1 works as follows:

- **Normal mode**: In this mode, **TESTBAR** must be in logic level high. The master-slave flip-flop composed by **LA** and **LB** are activated when **CLK** is high, acting as the shadow logic. The master-slave flip-flop composed by **PH1** and **PH2** are activated when **SYS_CLK** is high. The voter votes on **LA**, **LB** and **PH1** when **SYS_CLK** is high, and the value of the voter is captured by the keeper at the output node. The voter votes on the keeper, **LB** and **PH1** when **SYS_CLK** is low.

- **Scan mode**: In this mode, **TESTBAR** must be in logic level low. The latches **LA** and **LB** forming a scan behavior and are activated by the scan clocks **SCA** and **SCB**, like the LSSD test protocol [EW77]. The input **SI** receives the test input. After **LA** and **LB** load the test input, the **UPDATE** signal is pulsed to store the test input and the system change to the normal mode operation. The test response is captured after one functional clock cycle, **PH2** loaded the test response and propagated this value to **PH1** at the end of the cycle. **CAPTURE** is activated, and the final output of the test is captured in **LA**. The system returns to scan mode operation, and the captured response is copied to the **SO** output.

XSEUFF 2 works as follows:

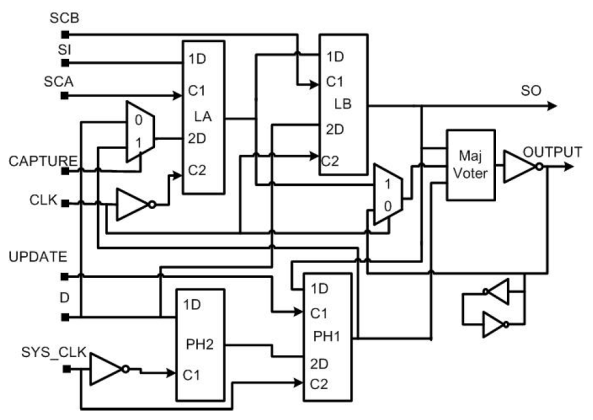
- **Normal mode**: **SyncLA** and **SyncLB** are used as gating signals for **LA** and **LB** respectively. Pulses **SyncLA** and **SyncLB** are at logic level low after the active clock, which allows **LA** and **LB** to sample copies of the expected data in **PH2**. When the clock is at logic level high, the voter circuit compares **PH2**, **LA** and **LB** to ensure the data correctness.

- **Scan mode**: In this mode, **ScanMode** must be in logic level high. The latches **LA** and **LB** form a scan behavior and are activated by the scan clocks **SCA** and **SCB**, like the LSSD test protocol protocol [EW77]. The input **SI** receives the test input. **UPDATE** and **CAPTURE** are used to apply the test input and capture the circuit response, respectively.

The results are extracted using the ISCAS'89 benchmark circuits. The registers of the benchmark circuits are replaced by the **XSEUFF** scan cells and compared using as reference the cell proposed by Kuppuswamy et al. presented in the Section 3.3.1. The results show an average transistor overhead of 28% and 20% for **XSEUFF 1** and **XSEUFF 2** respectively when compared with the Kuppuswamy et al. cell. According to the authors, the designs provide concurrent error correction with minimal degradation of system performance.

3.3.4 Error Detection Sequential Scan Approaches

Han et al. [HGJX13] analyze and compare three scannable error detection sequential approaches to replace the Intel resilient memory element [BTK+09]. Figure 3.12 shows the three scan designs. The Scan Shadow flip-flop approach (Figure 3.12(a)) adds a MUX gate to select between the system input **D** and the test input **S_I** controlled by the **Mode**


(a) XSEUFF 1 scan cell.

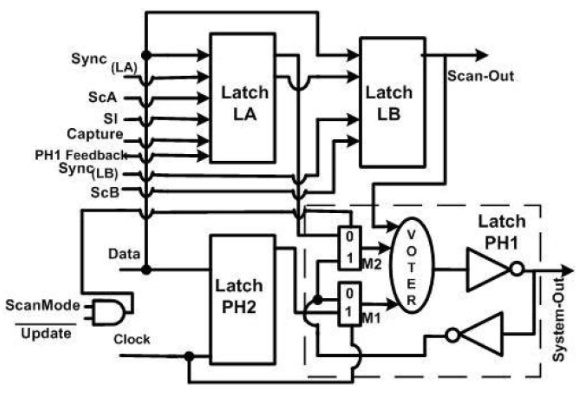
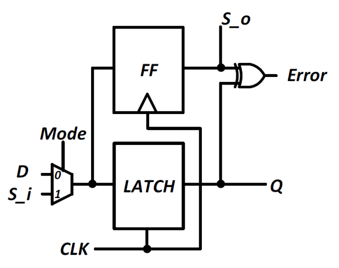
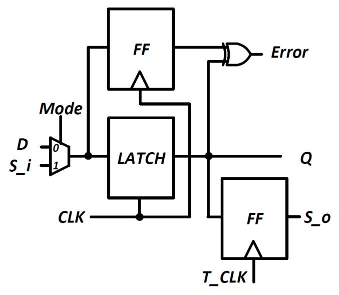

(b) XSEUFF 2 scan cell.

Figure 3.11 – XSEUFF scan cells [JOC07].

signal. The test output **S_O** is drived by the shadow flip-flop. The Scan tail flip-flop approach (Figure 3.12(b)) adds a new flip-flop at the latch output as the scan register. This version also adds a MUX gate to select between **D** and **S_I**. To perform the test, the **S_I** test input is copied to the latch activated by the signal **CLK** and to the flip-flop activated by the **T_CLK** signal. The third approach is based on the LSSD (Figure 3.12(c)). It uses a separate system and scan clocks to distinguish between normal and test mode. **CLK** is the system clock and controls the **D** input. The scan clocks are generated by the **T_CLK** signal, and controls the **S_I** input.


(a) Scan Shadow flip-flop cell.


(b) Scan tail flip-flop cell.

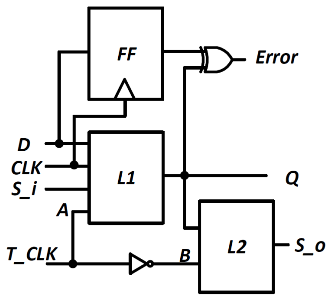

(c) Scan cell based on the Level Sensitive Scan-based Design.

Figure 3.12 – Error detection sequential scan approaches [HGJX13].

According to the authors, the Scan tail flip-flop approach reduces the scan-path switching power consumption when compared to the Scan Shadow flip-flop approach. However, the area overhead of The Scan tail flip-flop is large, once it adds a new register element. The Scan Shadow flip-flop has the smallest area overhead between the three approaches, but introduces the metastability problem on error-paths and adds a delay on data paths. LSSD has medium area overhead and does not present metastability and delay problems, which makes this approach a suitable design among all trade-offs.

3.4 State of the Art Conclusion

Despite the Razor family brings solutions to eliminate worst-case safety margins in the clock period, few architectures of this family concern with DfT. The Razor registers presented by this family are complex and can present test issues as area overhead. Another characteristic of this family is that some solutions present both latches and flip-flops in its implementation, and it may require a specific cell that allows test both types of registers simultaneously, once they have different test protocols. Also, the Razor techniques can present solutions that difficult the DfT insertion, like Bubble Razor that uses clock gating techniques. Clock gating technique increases the test area overhead, once extra logic in the clock gating is necessary to allow controllability to the clock during the test process.

Table 3.1 summarizes the test limitations of Razor family. The **DfT proposals** column informs if the template addressed DfT insertion. Only the Razor template [EKD+03] presents a scan cell approach: the Scan Razor flip-flop (described in Section 3.2.1). The Columns **Flip-flop** and **Latch** inform whether the template uses flip-flops and latches cells.

Table 3.1 – Razor family characteristic regarding test insertion and register components.

	DfT proposals	Flip-flop	Latch
Razor [EKD+03]	A [EKD+03]	Yes	Yes
RazorII [DTP+09]	NA	No	Yes
Razor-Lite [KKF+14]	NA	Yes	No
Bubble Razor [FFK+13]	NA	No	Yes
Saferazor [CBC+15]	NA	Yes	Yes

Section 3.2 and Section 3.3 present some alternatives to insert DfT in resilient circuits. However, few papers mention ATPG validation, and all the works do not present automated DfT insertion for these scan approaches. All the scan components presented in Section 3.2 are custom solutions to allow testability at resilient designs, and commercial DfT and ATPG tools may not support these solutions.

Table 3.2 summarizes the limitations of these scan approaches. The **Automated DfT insertion** column informs if the scan cell approach can be inserted automatically into the design by a DfT tool. The **ATPG Validation** column informs if the scan cell was or can be integrated ATPG tools. The Scan Razor flip-flop [ATA15] approach was validated using test vectors generated by an ATPG tool, while Kuppuswamy et al. [KDF+04] mention that its scan cell is compatible with ATPG tools.

The state of the art presents test solutions to resilient circuits regarding timing issues or soft-error issues. All works present custom test solution based on the scan chain technique. Most of these works concern with area overhead, and present approaches that reuse the existent components of the register elements to implement the scan function, as in the Scan Razor flip-flop [ATA15]. Other works describe solutions that have a small overhead

Table 3.2 – Scan approaches limitations regarding DfT automated insertion and ATPG.

	Automated DfT insertion	**ATPG Validation**
Scan Razor flip-flop [ATA15]	No	Yes
Timed flip-flop [FTK08]	No	No
Kuppuswamy et al.'s Scan cell [KDF+04]	No	Yes
Scan flip-flop with single-event upset detection [DKM05]	No	No
XSEUFF 1 [JOC07]	No	No
XSEUFF 2 [JOC07]	No	No
Scan Shadow flip-flop [HGJX13]	No	No
Scan tail flip-flop [HGJX13]	No	No
Scan cell based on the LSSD [HGJX13]	No	No

increase due to the addition of small components like MUXs used to allow the test, as the Scan Shadow flip-flop cell [HGJX13].

However, none of these works concern with DfT tools compatibility. The works do not mention if the proposed scan cell can be recognized by a DfT tool as a test cell and if the tool can replace the registers of the design by these test cells automatically. This limitation increases the time spent with the design for test, once the registers must be replaced by the test cells either manually or using custom TCL scripts that find and replace these registers. Therefore, according to the works presented in Section 3.2 and Section 3.3, test automation is a gap in resilient templates.

Besides, few works mention the use ATPG tool in its evaluation or mention fault coverage estimation. None of the works presented in Section 3.2 and Section 3.3 evaluate the fault coverage for fault models like stuck-at and path-delay. Thus, the support to ATPG is also a gap in resilient templates.

4. PROPOSED CELLS DESIGN

This Chapter presents contributions **1** and **2** (see Section 1.3) of this dissertation, which consists of the description and implementation of the cell implementations needed to add scan chain into the Blade design flow. As mentioned before, Blade uses different types of memory elements such as latches, EDLs and Q-Flops, as shown in Figure 2.20. These cells must be replaced by scannable cells to build scan chains. This section presents the proposed scan cells for the latches, EDLs and Q-Flops. The following cells were implemented to replace the Blade sequential elements:

- An active low LSSD;
- An active high LSSD;
- An active low LSSD with active low reset;
- An active high LSSD with active low reset;
- An active high Clocked-LSSD.

Also, the following cells were implemented to specific Blade functions:

- A transition detector;
- A metastability filter.

Figure 4.1 shows a generic example and where the sequential cell is localized in the Blade circuit. Sections 4.1, 4.2 and 4.3 show the scannable version of latches (identified by the number one), TDs (identified by the number two) and Q-Flops (identified by the number three), respectively. All of these cells are implemented targeting the 28 nm FD-SOI technology. Section 4.4 shows an option compatible with the LSSD test protocol to replace the Q-Flop flip-flop.

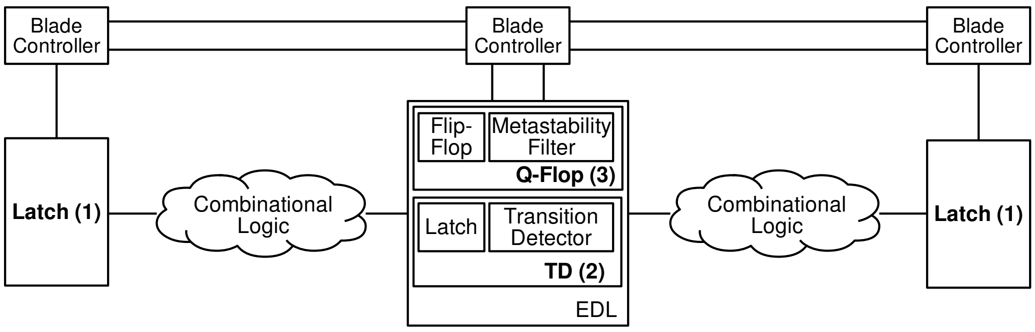


Figure 4.1 – Blade sequential elements.

4.1 Replacing Latches

This Section describes part of the contribution number **1** presented in Section 1.3 at page 26. The Blade synthesis flow has a phase where flip-flops memory elements are replaced by latches (described in Section 5). This phase uses different kinds of latches such as active low, active high, active high with active low reset and active high with active low reset latches. Thus, it is necessary the LSSD cell versions that are compatible with these latches. So, the following LSSD versions were implemented:

- An active low LSSD;
- An active high LSSD;
- An active low LSSD with active low reset;
- An active high LSSD with active low reset.

These cells were designed by the guideline presented in [JMKA17]. A Spice description, a Verilog description, and a Liberty file were generated for each LSSD cell. The Liberty file was generated using Liberate by Cadence [SYN14], and the Verilog description was generated using User-Defined Primitives (UDP). Appendix A details the design and implementation of these cells. The obtained results are presented in Section 5.3 that evaluates the case study XTEA criptocore [NW97] using full scan approach.

4.2 Testable Transition Detector

This Section describes part of the contribution number **2** presented in Section 1.3 at page 26. As mentioned before, the Blade template EDL has a latch with an error transition detector, called TD. This circuit consists of a D latch and a transition detector implemented as a single cell.

The Latch inside the TD must be replaced by an LSSD to allow the DfT insertion in the Blade flow. The solution to automate the DfT insertion into the TD cell is to split the TD cell into two different cells:

- A latch cell available in the 28 nm library;
- A transition detector cell;

Figure 4.2 shows the TD diagram with these two blocks. The split description allows the synthesis tool to recognize the latch as a simple memory element. Thus, it can be replaced by an LSSD. To perform the split in the TD cell, a Spice description and a Liberty file for the transition detector cell were generated. The Liberty file was generated using Liberate by Cadence [SYN14]. Section B details the generated codes for TD.

Even though splitting the TD into two blocks solves the DfT insertion issue, it can present some problems for physical synthesis. If the cells are placed apart from each other, the wire delays can affect the expected cell behavior. Section 4.3.2 presents a solution to this challenge using the Q-Flop as example. The obtained results for DfT insertion in the TD are presented in Section 5.3 that evaluates the case study XTEA criptocore [NW97] using full scan approach.

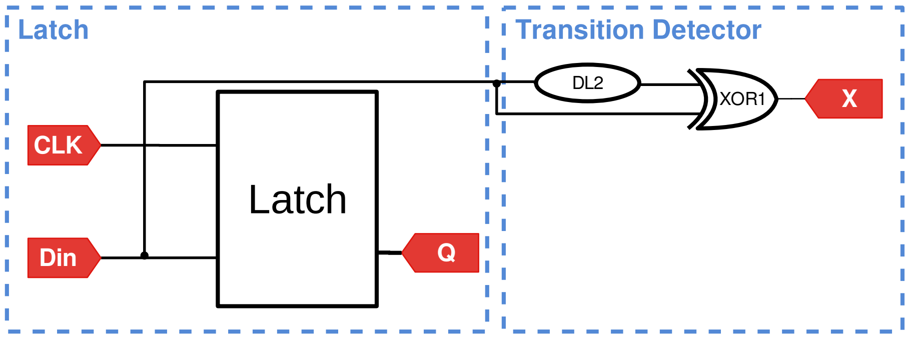


Figure 4.2 – Testable Transition Detector. This diagram is an adaptation of the Blade original paper [HTMH+15].

4.3 Testable Q-Flop

This Section describes part of the contribution number **2** presented in Section 1.3 at page 26. The Q-flop is a complex memory element and it is not natively supported by conventional DfT flows and technology libraries. One solution to enable that is to split the Q-flop cell into two different cells:

- D-Type Flip-Flop;
- Metastability Filter;

Mullins and Moore [MM07] proposed a static-logic implementation of the Q-Element [RMCF88] separating the Q-Flop into a flip-flop and a metastability filter. In this proposal, the clock signal is connected to the reset port to implement the Q-Flop behavior, as shown in Figure 4.3. Thus, the DfT insertion tool recognizes the flip-flop as a standard register cell and can replace it by scannable testable cells. Nevertheless, this kind of connection generates a DfT rule violation: the same signal can not feed a clock and reset inputs of a sequential cell. In other words, the clock and the reset inputs must be independent and with high controllability. Connecting the same signal in these two pins prevents the DfT tool from applying the test protocol to the scan cell, which can lead to a mismatch between test patterns and captured values.

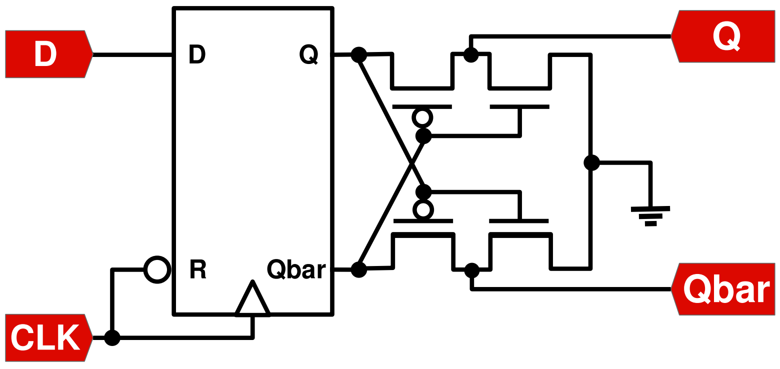


Figure 4.3 – Static-logic Q-Flop implementation [MM07].

To solve that, a new component that deals with the DfT violation and allows automated DfT insertion using a DfT tool was proposed, called testable Q-Flop. Figure 4.4 shows the gate-level diagram of the proposed testable Q-Flop. The D-Type Flip-Flop block is the sequential component of the testable Q-Flop, which can be recognized by commercial DfT tools. The Metastability Filter block is composed of two circuits: the *Dual-Rail Converter* and the *Filter*. The *Dual-Rail Converter* circuit converts the flip-flop output to a dual-rail codification and the *Filter* circuit avoids metastability. This way, it is possible to insert a scannable element without DfT violations and replace the sequential block to a scan cell.

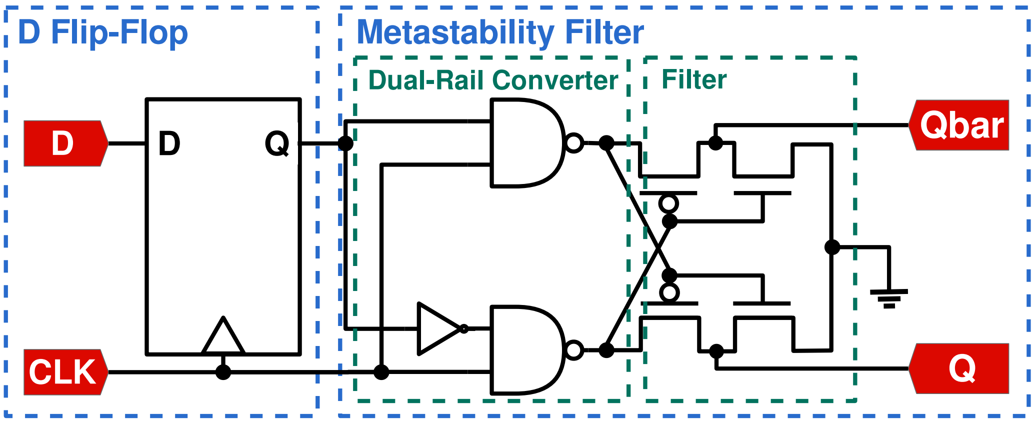


Figure 4.4 – Proposed testable Q-Flop cell.

Moreover, it is not necessary to implement a new sequential cell, once the 28 nm library had standard flip-flop sequential cells. Besides, being able to control the Q-Flop during the test can reduce the test time. This is because its metastability filter can take an unbounded time to resolve its outputs values if it goes metastable, which can happen if input data stability is not guaranteed. With the scan-chain, the filter always receives stable values at the input, increasing the controllability of the filter, and, therefore, the output values. Another advantage is that the electrical characterization is simplified with the testable Q-Flop since its array configuration transistor does not produce unpredictable transitions like the cross-coupled NANDs approach used in the original Q-Flop. The feedback between the NANDs can generate delayed values, which produce this mentioned unpredictable values in the internal nodes and cause miss matches at the characterization simulation. Avoiding the cross-coupled NANDs implementation allows the commercial electrical characterization tools to more easily measure setup and hold times.

A Spice description, a Verilog description, and a Liberty file were generated for the metastability filter cell. The Liberty file was generated using Liberate by Cadence, and the Verilog description was generated using User-Defined Primitives (UDP). Appendix C details the metastability filter codes.

4.3.1 Comparison with the Original Q-Flop

This section compares the proposed solution to the original Q-Flop and evaluates overheads considering propagation delay, leakage power and energy values. Leakage power represents the power of the circuit when the inputs are stable. Energy represents how much the circuit consumes when it is operating and is measured as the average energy per transition in the cell. Propagation delay represents the amount of time required for a change in the input to reach the output. In order to perform a fair comparison between the components, both of them were described in Spice, targeting the 28 nm FD-SOI technology and sized with logical effort assuming a fanout of 4. All reported results are based on Spice simulation, using the Spectre Circuit Simulator.

The experimental setup assumed slew values typical of a simple buffer in the 28 nm target technology, (0.003 ns for best case, 0.065 ns for nominal case and 1 ns to worst case). These values allow a range of possible operating conditions. The columns of Table 4.1 show the obtained values and the respective overheads. The values presented in the table are the average of the three corner cases values, and the overhead column is presented as percentage. The values highlight in red represent a loss of the testable Q-Flop compared with the original Q-Flop.

The testable Q-flop has six transistors more than the original. Part of this overhead is due to the dual-rail converter in the metastability filter. The rise propagation time decreases once the outputs pass to a two-input NANDs instead three-input NANDs, which has a transistor stack smaller. The fall propagation values are similar because in both cases a NAND logic gate drives the output to zero when the CLK is zero. The dynamic energy and leakage power have a small improvement, once the register is implemented with tri-state and standard inverters, which are less complex than crosscoupled NANDs.

Table 4.1 – Comparison of original Q-Flop against testable Q-Flop.

	Q-Flop	Testable Q-Flop	Overhead (%)
Total Transistors	28.00	34.00	21.43
Average Rise Propagation time (ns)	97.56	79.80	-18.20
Average Fall Propagation time (ns)	22.84	22.35	-2.12
Average Dynamic Energy (fJ)	18.16	15.11	-16.79
Average Leakage Power (nW)	161.47	151.03	-6.47

4.3.2 Place and Route Methodology

This Section presents a methodology to keep the register and the metastability filter close in the physical synthesis, which is a requirement to ensure the correct behavior of the Q-Flop. If both cells were placed distant from each other, the filter behavior could be affected by additional wire delays and capacitances, impacting its reliability metrics electrical characteristics. To show the results, a small circuit was implemented containing four Q-Flops, one four-input AND logic gate and one four-input OR logic gate, as shown in Figure 4.5. This example is based on the logic to detect errors of the Blade template. The Encounter tool of Cadence was used to perform the physical synthesis of this case study.

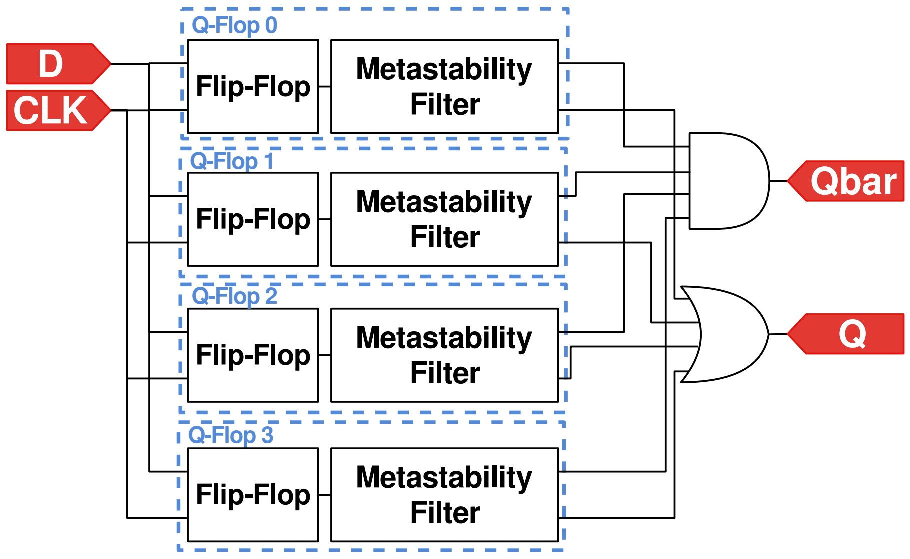
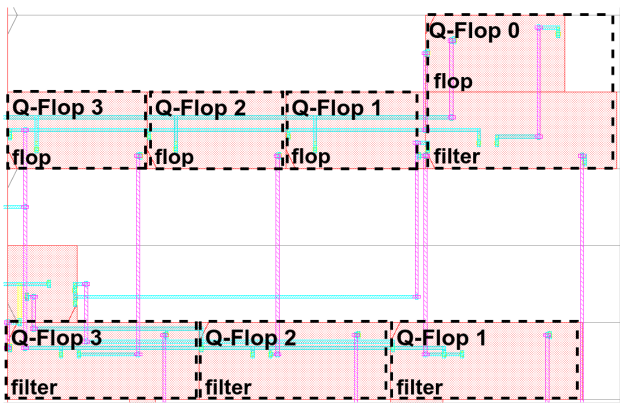


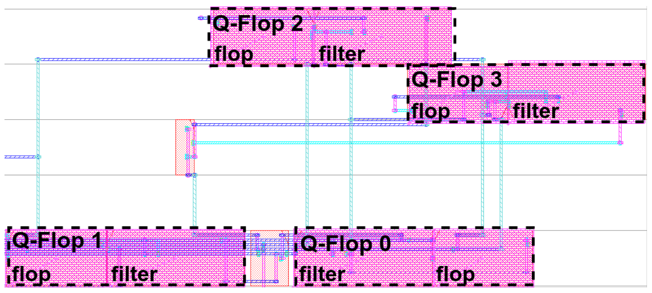
Figure 4.5 – Testable Q-Flop physical synthesis evaluation scenario.

The following methodology was used to ensure that the cells are placed close:

i. Perform the floorplan and powerplan;



(a) Placement results for physical synthesis *without fence and blockage constraints*.



(b) Placement results for physical synthesis *with fence and blockage constraints*.

Figure 4.6 – Testable Q-Flop physical synthesis results.

ii. Put the register and the metastability cells close in the design area, which can be done using a hierarchical approach;

iii. Create fence constraints, which restricts the cell placement to the fenced area;

iv. Create blockage constraints specifying *M2* and *M3* layers;

v. Perform the placement;

vi. Define the cells placement status as *FIXED*, which prevents the tool from moving them in the layout;

vii. Perform the clock tree synthesis;

viii. Perform the routing;

ix. Add the filler cells, metal layers and perform sign-off checks.

This process can be automated by Tool Command Language (TCL) scripts using standard commands to perform an hierarchical synthesis and preserve the fence and blockage constraints. Some specific commands can be used too to put the cells close, such as gravity commands. Besides, the proposed methodology is compatible with any tool that supports hierarchical placement, as the only requirement is that both cells must be placed in a bounded physical region defined by the module that contains them.

Figure 4.6 shows the synthesis result with a conventional flow, and without the fence constraints and layers blockage. Figure 4.6(b) shows that the fence constraint ensures the placement of the flip-flop next to the metastability filter, while Figure 4.6(a) shows that the cells are distant. Thus, it is possible to ensure the correct behavior of the testable Q-Flop split in two different cells using constraints provided by the synthesis tool.

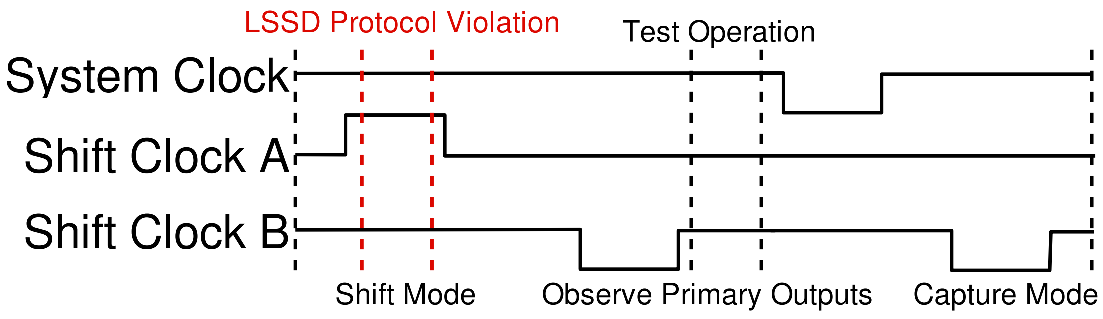


Figure 4.7 – LSSD protocol violation in Yurash's [Yur95] clocked-LSSD cell.

4.4 Clocked-LSSD Optimization

This Section describes part of the contribution number **1** presented in Section 1.3 at page 26. The previous section described a Q-Flop version that allows the DfT insertion using a standard D-Type Flip-Flop and a metastability filter. However, a typical LSSD test protocol assumes latches as the memory element. In spite of that, there is a particular LSSD-compatible flip-flop scan cell called Clocked-LSSD, described in Section 2.5. However, this original cell has a big area overhead because it is composed of three latches, while a standard flip-flop has two latches.

There is a reduced set of works about Clocked-LSSD about it optimization in contemporary literature. Yurash [Yur95] patented an optimization of Clocked-LSSD that uses two latches instead of three latches in the original design. This cell has 40 transistors against 48 transistors required by the original cell design. However, this patent presents a different test protocol that is not compliant with the standard LSSD protocol, see Figure 2.15(b).

Unlike the conventional Clocked-LSSD, this cell requires that the system clock is kept at a high logic value during the shift operation, as shown in Figure 4.7. In the original test protocol, the system clock is at 0 logic value during this operation. If the clock is at low logic value during the shift operation in the patent cell, the data at input **SI** does not pass through the first latch, interrupting the operation. Moreover, this cell needs an overlap between the clocks **A** and **B** (as shown in Figure 4.7), while in the LSSD test protocol does not allow the test clock overlap to avoid a race condition. Thus, this implementation is not compatible with contemporary standards and commercial synthesis tools, significantly reducing the levels of automation provided for designs using it.

To solve that, a new transistor architecture for a Clocked-LSSD optimization was proposed. Figure 4.8 shows the schematic of the proposed optimized cell. As the cell proposed by Yurash, this optimization uses two latches. This optimization has 2 additional transistors (**ET0** and **ET1**) than the Yurash's cell which ensure that the proposed optimization cell has the same test protocol presented in Figure 2.15(b).

Although it seems at a first look that inverting the system clock signal is sufficient to ensure protocol compatibility, the truth is that a review in this cell is needed to ensure the correct test protocol behavior. The clock inversion just changes the clock edge, and the test protocol violation still happens once the clock must be active during the test operation. In the proposed optimized cell, it is necessary only these extra transistors to implement the logic that ensure the non-influence of data **D** selector in the behavior of the cell when the clock **A** is at logic level 1. Thus, this cell can be integrated with the commercial synthesis tools. The cell sizing was designed by the guideline presented in [JMKA17].

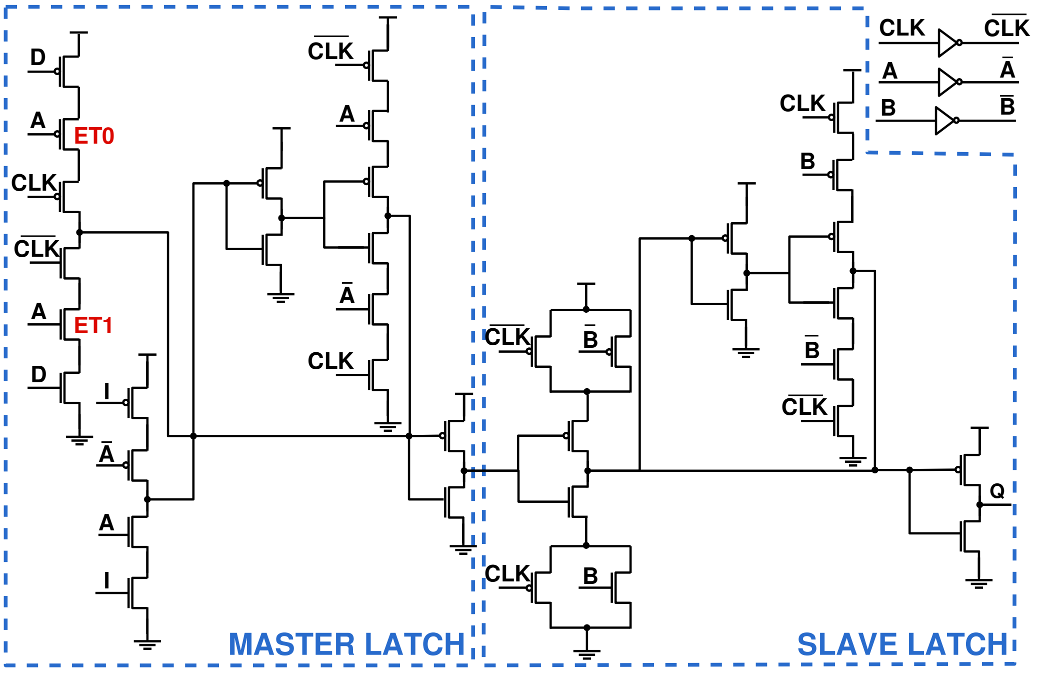


Figure 4.8 – Proposed optimized Clocked-LSSD schematic.

A Spice description, a Verilog description and a Liberty file were generated for the proposed Clocked-LSSD. The Liberty file was generated using Liberate by Cadence, and the Verilog description was generated using User-Defined Primitives (UDP). Section D details the Clocked-LSSD codes.

4.4.1 Results

This Section compares the conventional Clocked-LSSD and the proposed cell using a case study to perform power and area analyses. Note that, because the Yurash's cell is not compatible with the protocol used by the synthesis tool, is not possible to perform a comparison between this cell and the proposed optimized cell using standard EDA tools. In order to perform a fair comparison between the components, both the Clocked-LSSD cells were described in Spice, targeting the 28 nm FD-SOI technology. The steps of the characterization process to generate the liberty file is the same reported in Appendix A and Appendix B show the timing arcs of the Clocked-LSSD.

A *synchronous version* of a pipelined XTEA criptocore was chosen as the case study to perform the synthesis and DfT insertion using both Clocked-LSSD cells. The reason behind the choice for this design is that it enables a controlled environment for inserting test cells and comparing results, as it is a simple pipeline with well defined sequential stages and combinatorial logic. Accordingly, the XTEA design has data inputs of 64-bit width, a 128 bits key input and a 64 bits data outputs. This design has 32 pipeline stages, 23,681 combinational cells, and 6,238 D flip-flop cells. The circuit operates at 200 MHz frequency.

A full flop-based design was chosen to demonstrate these results to ensure that the result will reflect only the Clocked-LSSD characteristics. Synopsys Design Compiler and DFT Compiler were used to perform logic synthesis and automatic *full scan* insertion. Two designs were generated: one using the conventional Clocked-LSSD and one using our proposed cell. To perform the power analyses, a value change dump (VCD) was generated by a simulation with delay provided by the standard delay format (SDF) generated in the logic synthesis. Table 4.2 shows the test overhead reduction of the proposed optimized cell.

Table 4.2 – Area and power evaluation of the synchronous XTEA with full scan and clocked-LSSD scan cells.

	Area (μm^2)	**Leakage Power (nW)**	**Dynamic Power (nW)**
Dft Using the Conventional Clocked-LSSD	106,134.35	1,275,976.24	4,742,273.18
Dft Using the Proposed Clocked-LSSD	88,203.24	985,096.69	1,815,258.32
Proposed Clocked-LSSD reduction (%)	16.89	22.80	61.72

As the table shows, the circuit that used the proposed Clocked-LSSD presents an area reduction of 16.89% when compared with the circuit using the conventional Clocked-LSSD. Also, this reduction was of 22.79% and 61.72%, respectively, considering leakage and dynamic power data. The leakage reduces due to the number of transistor reduction in the proposed optimized cell, once the leakage power has relation with the number of transistors. The dynamic power has a significant reduction due to the extra transistors **EXT0** and **EXT1** (presented in Figure 4.8), once they prevent the master latch data selection from activating simultaneously and not propagating the switching through the circuit. The transistor configuration of the slave latch also contributed to this reduction, once the master latch of the conventional Clocked-LSSD is implemented using a double latch, and the optimized version is implemented with a single latch.

4.4.2 Cell Area Estimation

As the Blade flow only address logic synthesis, the cell described before does not have an implemented layout. Thus, a methodology was adopted to estimate the area value for each test cell. First, was extracted the cell area values of a flip-flop (**DFPQ** cell), Muxed-D (**SDFPQ** cell) and latch (**LDHQ** cell) from 28 nm library documentation [STM12]. After, this value was divided by the number of transistors necessary to implement each cell. Thus, the obtained proportion was used to estimate the values for the LSSD and Clocked-LSSD cells. Table 4.3 shows the obtained values for each cell. By using this method, the cell area for LSSD and Clocked-LSSD was estimated as 6.2 and 6.8 µm^2, respectively.

Table 4.3 – Area estimation for test cells

	Flip-flop	**Muxed-D**	**Latch**	**LSSD**	**Clocked-LSSD**
Number of transistors	18	24	10	36	42
Area (µm^2)	2.61	4.08	1.47	**6.20**	**6.80**
Area (µm^2)/**Number of transistors**	0.15	0.17	0.15	0.17	0.17

5. BLADE DFT SYNTHESIS FLOW

This Section presents the necessary modifications in the Blade synthesis flow to integrate DfT insertion, which represent the contribution number **3** presented in Section 1.3 at page 26. Figure 5.1 presents the proposed Blade synthesis flow with DfT insertion. This Figure is the same presented in Section 1.3 at page 27, and it is shown again in this Section to make the work reading easier. The Blade template has four synthesis phases:

- Synchronous Synthesis;
- FF to Latch Conversion;
- Retiming;
- Blade Conversion.

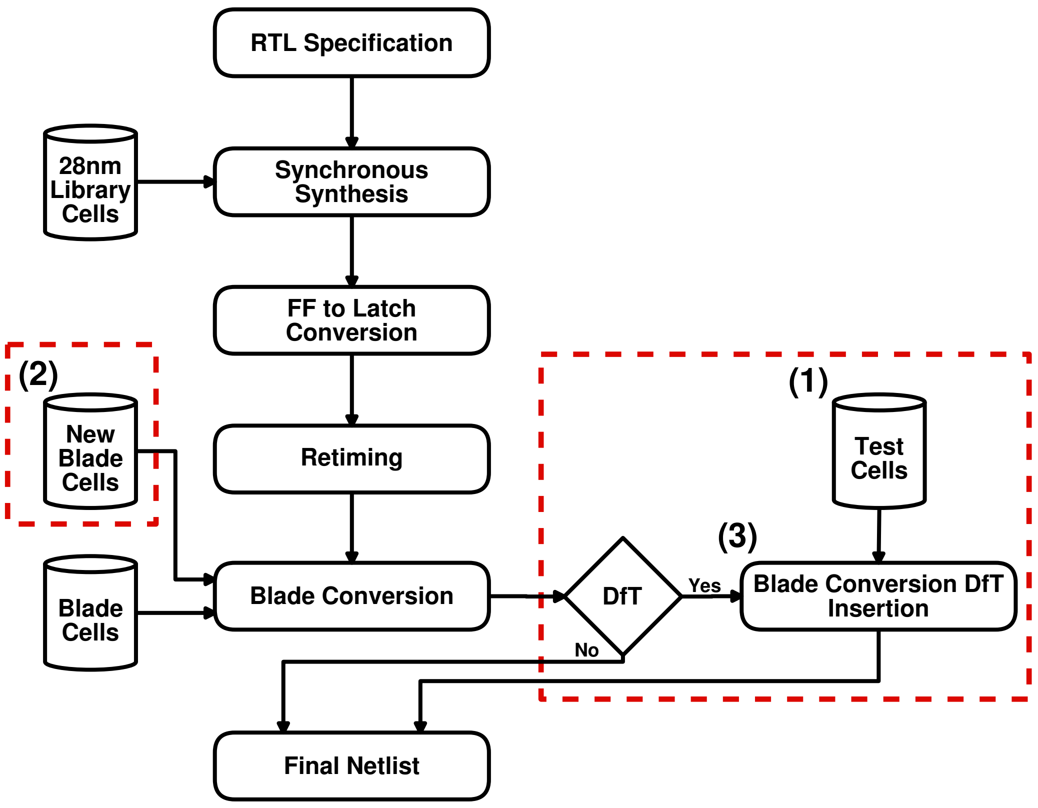


Figure 5.1 – Blade synthesis flow where the red dashed square shows the proposed contribution to improve the Blade testability. The numbers in the Figure represent the three contributions of this work.

The Synchronous Synthesis phase performs a flip-flop based synchronous design synthesis. A synchronous design RTL description is needed as a start point in this phase. This RTL must be described using the pipeline technique. Although it is possible to work around, the flow works best if the RTL has a single clock where all flip-flops work at the same edge and the reset activated at the same level. The FF to Latch Conversion phase replaces all the flip-flops by master-slave latches, keeping the edge-triggered behavior design of the Synchronous Synthesis phase. The Retiming phase applies the register retiming technique [SYN15a] to perform the dual-phase implementation. This phase splits the master-slave latches in single latches, converting the design to a synchronous dual-phase latch based implementation. Thus, the design works at the high and low clock levels, changing the edge-triggered behavior of the Synchronous Synthesis phase. The Blade Conversion phase converts the dual-phase implementation to a Blade implementation. This phase replaces the clocks of the dual-phase design by asynchronous controllers. It also inserts delay lines and error detection logic blocks.

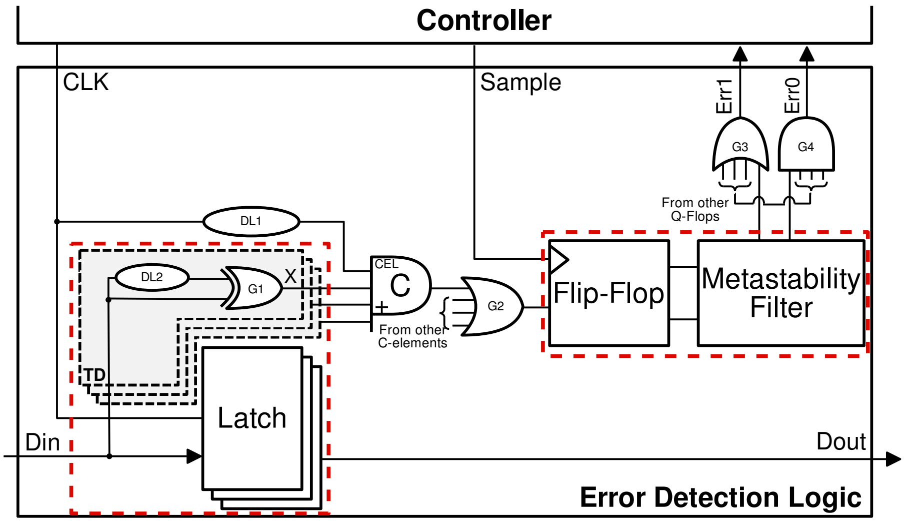


Figure 5.2 – Proposed Error Detection Logic implementation. The TD was split into a latch and a transition detector circuit, and the Q-Flop was split into a flip-flop and a metastability filter circuit. The red dashed squares highlight these modifications.

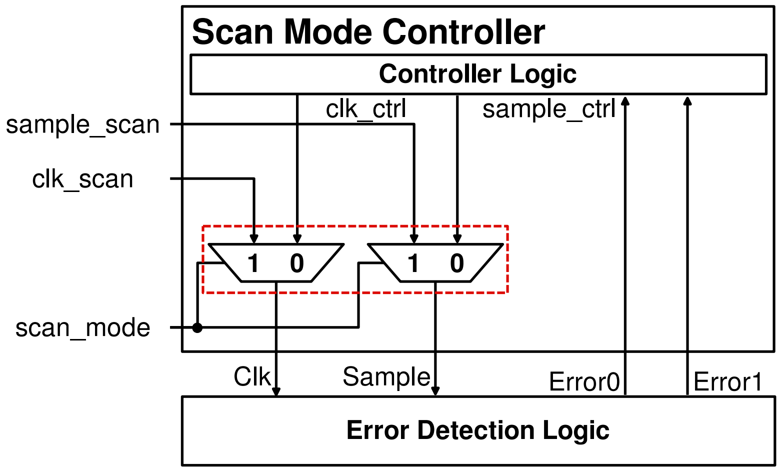


Figure 5.3 – Controller modification to support scan chain insertion. MUX components were added to allow controllability for the clocks. The red dashed squares highlight the MUXs.

The DfT insertion is performed after the Blade conversion synthesis phase, as shown in Figure 1.1. After the end of this phase, the circuit has latches in the datapath and EDLs with transition detector (TD) in the critical paths. Section E details the scripts used to automate the DfT insertion in the Blade synthesis flow. Figure 5.2 shows the new error detection logic block. Each TD is replaced by a standard latch cell and a TD cell, as described in Section 4.2. Also, the Q-Flops are replaced by a standard flip-flop cells and a metastability filter cells, as described in Section 4.3. Thus, it is possible to automatically replace all the latches by LSSDs test cell (as detailed in Section 4.1) and all the flip-flops by Clocked-LSSD cells (as detailed in Section 4.4) using the logic synthesis tool.

Furthermore, the controller must be modified to allow DfT insertion, as shown in Figure 5.3. Tree new inputs are added: the test clocks **clk_scan** and **sample_scan** and the MUX selector **scan_mode**. Besides, two MUX are added in the controller to choose between the signals provided by the controller or the test signals. These modifications allow the clocks to be controllable during the test operation.

5.1 Detailed Blade DfT Flow

This Section details the proposed Blade flow steps executed by the DfT tool, shown in Figure 5.4. TCL scripts automate these steps, and no manual intervention is needed during the flow execution. The first step is the DfT Signals Specification. A netlist from logic synthesis is needed as an input to the DfT tool to perform the first step. The necessary test signals are declared to perform the scan-chain operations, as the test enable and test input. The next step, Test Interface Configuration, defines the behavior of the circuit signals during the test operation, as the level of the test enable signal.

The Test Protocol Definition step specifies the edge-triggered test protocol or the LSSD test protocol. As Blade is a latch-based design, the test operation uses the LSSD test protocol. The Replace Register step replaces all latches of Blade by LSSD cells and all flip-flops of Q-Flops component by Clocked-LSSD cells. This step needs the test cell as an input to the DfT tool. The next step, Connect Scan-Chain, interconnects all LSSD and Clocked-LSSD cells into a scan-chain. The Check DfT Rules step analyzes the design with the scan-chain to detect DfT rules violations.

The Export to ATPG step generates all the files to perform the ATPG step. It exports the final netlist with the scan-chain and the Standard Test Interface Language Procedure File (SPF) used by the ATPG tool. The SPF file contains information about the scan connection, test signals, and signals behavior. The ATPG step reads the final netlist, the SPF file, and the setup script for the fault models. Stuck-At Setup Script is used to generate the stuck-at fault list which contains the faults for the circuit internal nodes. Path-Delay Setup Script extracts the combinational paths from the design. The ATPG step also generates the test patterns used in the Fault Simulation step. Fault Simulation is the last step and simulates the test patterns from the stuck-at or path-delay models to extract reports like fault coverage estimation and total clock cycles used in the test operation.

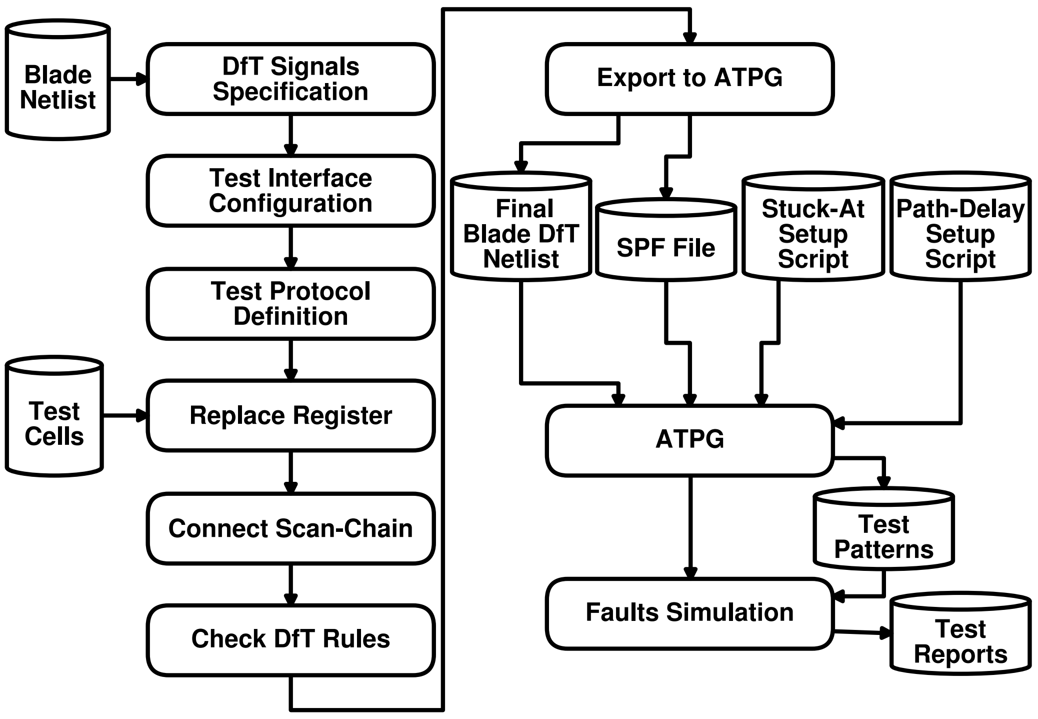


Figure 5.4 – Detailed proposed Blade DfT flow.

5.2 Case Studies

This Section presents the two case studies used to evaluate this work. Section 5.2.1 describes the XTEA crypto core, and Sections 5.2.2 describes the Plasma microprocessor.

5.2.1 XTEA

The Extended Tiny Encryption Algorithm (XTEA) is cryptocore design derived from the Tiny Encryption Algorithm (TEA) [Kap08]. It has a key of 128 bits to encrypt or decrypt data in blocks of 64 bits. The input is split into two blocks of 32 bits (**y** and **z**) and processed by a permutation block (**f**) for *N* rounds, where *N* is an integer number. Figure 5.5 shows the implementation of the XTEA used in this work. This version is based on the Speed XTEA [Kap08], and has 32 rounds. Each round is split into two smaller combinational blocks (**HALFROUND 1** and **HALFROUND 2**), and between this split logic has a temporal barrier (**INNER ROUND**).

XTEA uses a **+/-** block to perform a sum operation (in case of encryption function) and subtraction (in case of decryption function). The cryptocore function is selected by the signal **mode**. The block **Keygen** generates the subkey. The results of **f** and **Keygen** are processed by a XOR gate as a reversible function. For encryption operation, **z** is applied to the left side, **y** to the right side, and all **+/-** blocks are in addition mode. For decryption, the opposite is applied. One round of XTEA computes a new value for **y** and **z**.

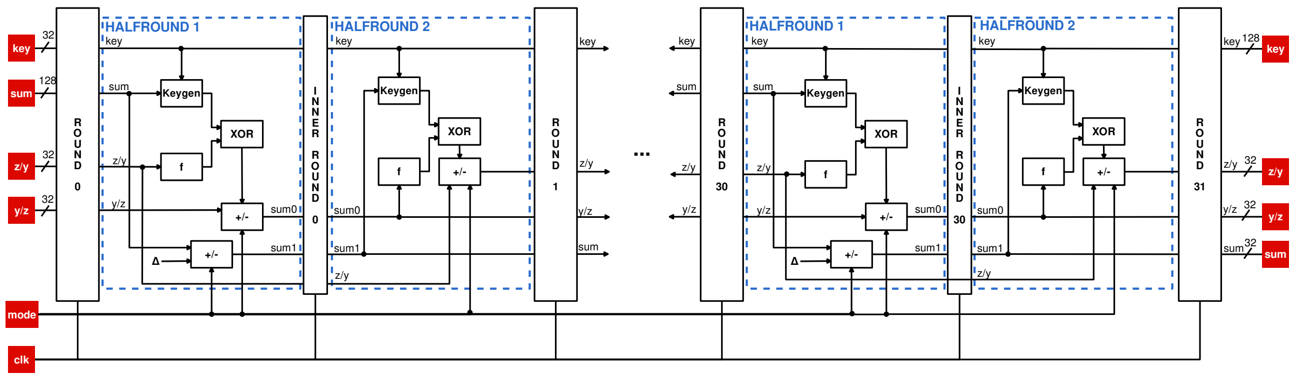


Figure 5.5 – XTEA pipeline design. This design is based in the Speed XTEA version presented in [Kap08].

5.2.2 Plasma

Plasma[1] is a 32-bit RISC microprocessor. It is composed by an interrupt controller, a UART and an SRAM controller. The Plasma CPU executes all MIPS I user mode instructions except unaligned load and store operations. The CPU is implemented with a three stage pipeline with an additional optional stage for memory read and writes, and supports both Big or Little Endian mode. Figure 5.6 shows the block diagram of Plasma microprocessor.

5.3 General Results Setup

This Section presents the obtained results from the Blade DfT insertion. The test flow is performed using a 28 nm library. The DC Compiler [SYN15a] synthesis tool from Synopsys was used in the Blade flow [HTMH+15]. For this reason, the DFT Compiler [SYN15b] was used to apply DfT at Blade flow. The test patterns for the fault models stuck-at and path-delay and the DfT insertion were performed using the ATPG tool TetraMAX [SYN15b] by Synopsys. Figure 5.7 shows a sample of the scan-chain generated by the Synopsys DFT

[1]https://opencores.org/project,plasma

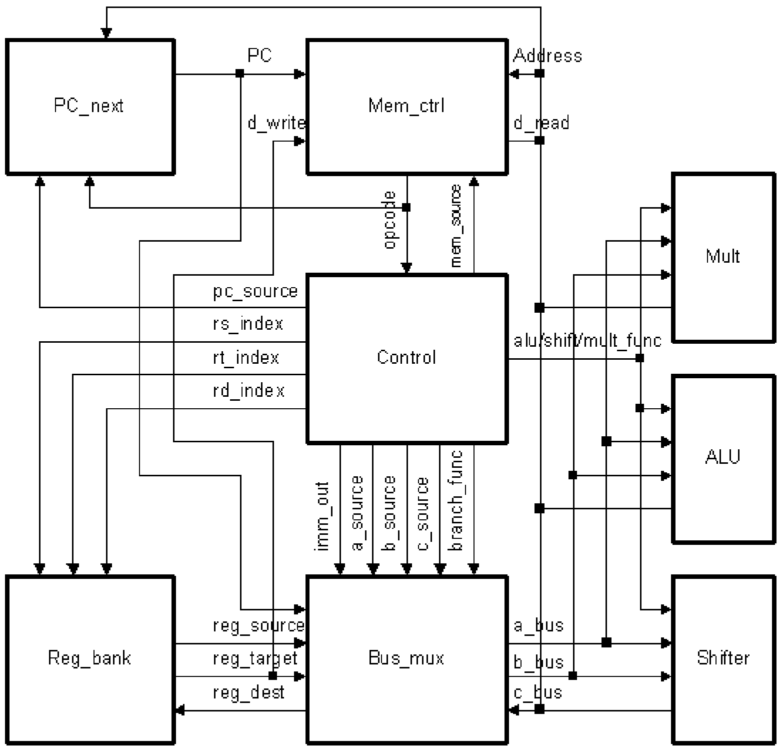


Figure 5.6 – Plasma microprcessor block diagram.

Compiler tool. This sample shows an LSSD cell connected to a Clocked-LSSD cell in the Q-Flop component, forming the Blade scan-chain. The following Sections describe the results for silicon area overhead (Section 5.4.1) and ATPG (Section 5.4.2 and Section 5.4.3). The XTEA and Plasma where synthesized targeting 454MHz.

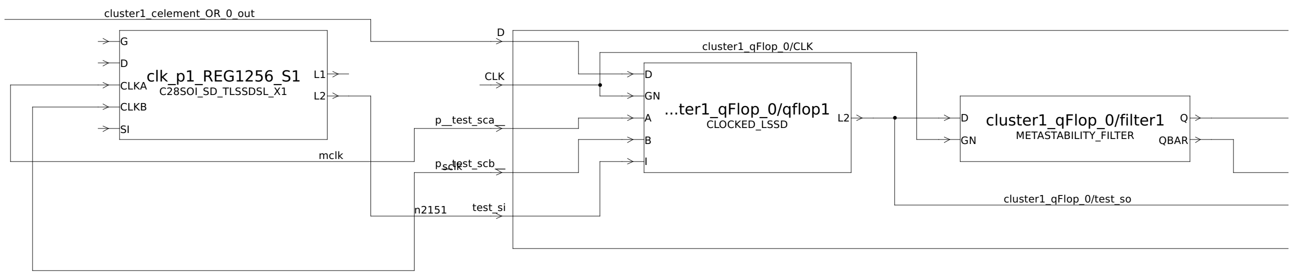


Figure 5.7 – Schematic sample of the scan chain. This Figure was extracted from DC Compiler tool by Synopsys.

5.4 XTEA Evaluation

This Section presents the results analyses for the XTEA case study. Section 5.4.1 shows the area overhead results, and Sections 5.4.2 and 5.4.3 show the ATPG results.

5.4.1 Silicon Area Results

This Section analyses the area overhead results for the XTEA Blade version design, shown in Table 5.1. The rows of the table show the number of cells used in the synthesis and the silicon area of these cells. The columns of the table represent the version of the XTEA design (synchronous, synchronous with DfT, retiming, retiming with DfT, Blade and Blade with DfT) and the overheads (Blade over synchronous, Blade over retiming, synchronous with DfT over synchronous, retiming with DfT over retiming, Blade with Dft over Blade, Blade with Dft over Synchronous with DfT and retiming with DfT over Blade with DfT). Table 5.3 details the area for the extra components required for DfT and Blade conversion and they percentage of the total area of Blade with DfT.

The Blade version of XTEA presents an area overhead of 13.78% (as shown in row **7** and column **6** of Table 5.1) when compared to the synchronous version. This overhead is generated by the addition of the Blade components like the controllers and EDLs. The test overhead of the Blade is 112.16% (as shown in row **11** and column **6** of Table 5.1), while the test overhead for the synchronous version is 0.63% (as shown in row **9** and column **6** of Table 5.1). These results are obtained dividing the total area value of the version with DfT over the total area value of the version without DfT. The low overhead presented by the synchronous version is explained because of the area overhead a Muxed-D when compared to a flip-flop is small, once the MUX component adds a small area to the cell. The high overhead presented by the Blade version is explained because each flip-flop of the synchronous version is replaced by two latches when converted to the Blade version, which means the number of sequential cells is duplicated. Besides, when a latch is replaced by an LSSD, the sequential area is increased about two times, once a single LSSD is composed by two latches, as shown in Figure 2.7, at page 34.

Besides, the proportion of combinational cells over sequential cells in the XTEA influences this overhead. Table 5.2 shows the cell proportions of the XTEA synchronous version and XTEA Blade version. The XTEA Blade version has 31,205 combinational cells (as shown in row **5** and column **1** of Table 5.1) and 15,644 sequential cells (as shown in row **5** and column **2** of Table 5.1), which means a proportion of 1.99. The synchronous version has 23,681 combinational cells (as shown in row **1** and column **1** of Table 5.1) and 6,238 sequential cells (as shown in row **1** and column **2** of Table 5.1), which means a proportion of 3.79, bigger than the Blade version. Thus, the bigger is this proportion, the lower is the test overhead, once the test circuitry is mainly applied to the sequential elements. Designs with fewer register elements can present a lower test overhead, as shown in Section 5.6.1.

Another reason that contributes to the high overhead can be the retiming technique applied to the Blade flow. As described in Section 2.3.3, the retiming technique can add new registers at the design to balance the path delays. In the XTEA case, the retiming adds

3168 extra memory elements. Thus, the Blade version can presents more than two times sequential cells when compared with the synchronous version due the retiming process.

Thus, the main contribution for test overhead is the flip-flop replacement by latches plus the retiming, which increases the number of sequential elements, increasing the length of the scan chain. The Blade component has a small contribution to the area overhead. To demonstrate it the results for the synthesis after retiming and after retiming with DfT were extracted. The design after the retiming optimization, without DfT insertion, has an area of 59,832.87µm^2 (as shown in row **3** and column **6** of Table 5.1), while the Blade has an area of 64,695.58µm^2 (as shown in row **5** and column **6** of Table 5.1). Thus, the Blade version without DfT has an overhead of 8.13% (as shown in row **8** and column **6** of Table 5.1) when compared with retiming version. The design after the retiming optimization and with DfT insertion has an area of 131,182.87µm^2 (as shown in row **4** and column **6** of Table 5.1), while the Blade version with DfT was 137,260.11µm^2 (as shown in row **6** and column **6** of Table 5.1). Thus, the Blade version with DfT has an overhead of 4,63% (as shown in row **13** and column **6** of Table 5.1).

Table 5.1 – Synthesis results for the XTEA design regarding the flop-based synchronous version, flop-based synchronous version with DfT, Blade version and Blade version with DfT.

		1	**2**	**3**	**4**	**5**	**6**
		Comb. Cells	**Seq. Cells**	**Total Cells**	**Comb. Area** (µm^2)	**Seq. Area** (µm^2)	**Total Cell Area** (µm^2)
1	**Synchronous**	23,681	6,238	29,919	33,444.09	23,414.96	56,859.04
2	**Synchronous With DfT**	30,305	6,231	36,536	31,825.14	25,392.13	57,217.27
3	**Retiming**	31,086	15,644	46,730	34,829.32	25,003.54	59,832.87
4	**Retiming with DfT**	32,545	15,644	48,189	34,190.07	96,992.80	131,182.87
5	**Blade**	31,205	15,644	48,743	39,692.04	25,003.55	64,695.58
6	**Blade With DfT**	31,205	15,756	48,743	39,499.62	97,760.49	137,260.11
7	**Blade/ Synchronous**	31.77%	150.79%	62.92%	18.68%	6.78%	13.78%
8	**Blade/ Retiming**	0.38%	0.00%	4.31%	13.96%	0.00%	8.13%
9	**Synchronous with DfT/ Synchronous**	27.97%	-0.11%	22.12%	-4.84%	8.44%	0.63%
10	**Retiming with DfT/ Retiming**	4.69%	0.00%	3.12%	-1.84%	287.92%	119.25%
11	**Blade with DfT/ Blade**	0.00%	0.72%	0.00%	-0.48%	290.99%	112.16%
12	**Blade with DfT/ Synchronous with DfT**	2.97%	152.86%	33.41%	24.11%	285.00%	139.89%
13	**Blade with DfT/ Retiming with DfT**	-4.12%	0.72%	1.15%	15.53%	0.79%	4.63%

Table 5.2 – Cell proportion of the XTEA synchronous version and XTEA Blade version.

	Synchronous Version	**Blade Version**
Combinational Cells	23,681	31,205
Sequential Cells	6,238	15,644
Proportion (combinational/sequential)	3.79	1.99

Table 5.3 – Area detailing of extra components used in test insertion and Blade conversion.

	Number of cells	**Total area (μm^2)**	**Percentage of the total area of Blade with DfT**
LSSD	15,644	96,992.80	70.66%
Transition Detector	1,324	2,592.92	1.89%
Q-Flop	112	767.69	0.56%
Metastability Filter	112	164.51	0.12%
Controller	3	68.22	0.05%
C-element	446	1,460.97	1.06%
Total	17,641	102,047.10	74.35%

5.4.2 Stuck-At Fault Model Results

Table 5.4 shows the stuck-at fault model results extracted from the ATPG tool for the XTEA synchronous version and the XTEA Blade version. The Blade version scan chain has more than two times the length of the synchronous version, once each flip-flop is replaced by two latches plus the extra latches included by the retiming, and an LSSD replaces each latch. Besides, the number of collapsed faults in the Blade version is bigger than the synchronous version, which explains the additional test patterns. These facts partially explain the number of patterns and the number of cycles.

The XTEA Blade version has a smaller fault coverage than the synchronous version. It occurs due to some Blade points that are not covered by the scan chain, as the inputs of ORs gates connected to the Q-Flop, C-elements and the Blade controller. Table 5.5 shows the undetected faults in the Blade. Internal undetected faults include internal nodes add the blade, as handshake protocol wires. The C-elements are the main contributor to the undetected faults, being 74.42% of the total undetected faults. It suggests that the C-Element should be included in the scan chain, like the latches and Q-Flops.

5.4.2.1 Stuck-At Fault Model Results Without Q-Flop

Turning the Q-Flop a scannable element enables the test of critical parts of the design, such as the metastability filters and their surrounding logic. However, the testable

Table 5.4 – ATPG stuck-at fault model results for XTEA synchronous version and XTEA Blade version.

	Synchronous With DfT	Blade With DfT
Fault coverage (%)	100	98.17
Number of total faults	299,300	484,482
Number of collapsed faults	47,151	135,137
Number of scan cells	6,231	15,756
Number of patterns	127	204
Number of cycles	797,825	3,230,594

Table 5.5 – Details of the XTEA Blade undetected faults.

Total detected faults	484,482
Total undetected faults	11,013
C-elements undetected faults	8,196
Q-Flop undetected faults	1,344
Controller undetected faults	548
OR tree undetected faults	434
Internal undetected faults	925

Q-Flop does not improve the fault coverage of the design. Table 5.6 shows that the testable Q-Flop has an improvement of 0.15% in the fault coverage. Table 5.7 shows that the addition of the Q-Flop in the scan chain has a small reduction in the number of undetected faults in the controller and the own Q-Flop. It occurs because the number of Q-Flops in the design is small when compared with the others memory elements, as demonstrated in Table 5.1 at page 84. Table 5.7 also shows that, whatever the Q-Flop is at the scan chain or not, the C-element is the main contributor to the total number of undetected faults, which still suggest that is necessary to allow testability to C-elements. Another reason for the Q-Flop low coverage is the lack of an observation point between the controller and the at own Q-Flop. It possibly means that the controller should also be scannable.

Table 5.6 – Comparison of the ATPG results for stuck-at fault model between the Blade with and without scannable Q-Flop (Clocked-LSSD).

	Blade With DfT With Q-Flop	Blade With DfT Without Q-Flop	Change (%)
Fault coverage (%)	98.17	98.02	0.15
Number of total faults	484,482	483,880	0.12
Number of collapsed faults	135,137	135,748	-0.45
Number of scan cells	15,756	15,644	0.72
Number of patterns	204	183	11.48
Number of cycles	3,230,594	2,863,404	12.82

Table 5.7 – Comparison of the undetected faults between the Blade with and without scannable Q-Flop (Clocked-LSSD). The column **%** represents the percentage of the faults detected with the Q-Flop in the scan chain.

	With Q-Flop	**Without Q-Flop**	**%**
Total detected faults	484,482	483,880	-0.12
Total undetected faults	11,013	11,294	2.49
C-elements undetected faults	8,196	8,196	0
Q-Flop undetected faults	1,344	1,568	14.29
Controller undetected faults	548	605	9.42
ORtree Internal undetected faults	434	434	0

5.4.3 Path-Delay Fault Model Results

Table 5.8 shows the path-delay fault model results extracted from the ATPG tool for the XTEA synchronous version and XTEA Blade version. This result covers 1000 combinational paths for both XTEA versions. The Blade version has an improvement in the fault coverage for path-delay faults. Besides, the LSSD reaches a better fault coverage with fewer patterns and fewer clock cycles. Two points can explain this result: i) The retiming process reduces the design combination paths, once they are balanced during this process. The number of combinational paths increases, but these paths are smaller than the synchronous version reducing the complexity of the test; ii) The increase in the number of elements in the scan chain, which is more than twice when compared with the synchronous version. With more scan elements more observability and controllability, increasing the fault coverage.

Table 5.8 – ATPG path-delay fault model results for XTEA synchronous version and XTEA Blade version.

	Sync. XTEA w/ DfT	**Blade XTEA w/ DfT**
Fault coverage (%)	51.02	89.37
Number of total faults	1,000	1,000
Number of collapsed faults	71	12
Number of scan cells	6,231	15,756
Number of patterns	326	61
Number of cycles	2,032,614	977,180

5.5 Optimized XTEA For Normalized Throughput Evaluation

Note that when a synchronous circuit is converted to the Blade template, the application throughput can increase twice due the Blade features, as the Blade 2-phase handshake protocol. Table 5.9 shows the throughput result for the XTEA synchronous version and Blade version. These results were extracted from the final netlists generated by the logic synthesis. The simulation also uses the Standard Delay Format (SDF) to perform the simulation with the logic gates delay. The synchronous version has a clock period of 2.20 ns, and the Blade request signal transits every 0.55 ns, which is one-quarter of the synchronous period.

The Blade version presents a throughput 105.51% greater than the synchronous version. As mentioned before, one reason for this result is the Blade 2-phase handshake protocol, once the circuit processes the combinational logic at the high and low levels of the Blade request signal, while the synchronous version works only in the rising edge of the clock signal. Another reason is the partitioning of combinational logic by the retiming process, once it can reduce the combinational logic paths of the design, as shown in Section 2.3.3 at page 33. Also, the use of latches as sequential elements increase the circuit performance due to its characteristics, as described in Section 2.3.

Table 5.9 – Throughput results for XTEA synchronous version and XTEA Blade version.

	Synchronous	**Blade**	**Blade/Synchronous (%)**
Throughput (MB/s)	3,636.36	7,473.09	105.51

With the presented results, it seems that to perform a fair comparison between a synchronous circuit and a Blade circuit, the synchronous one must have the double of the frequency, to achieve similar throughput of the Blade. Thus, a new version of the XTEA was generated with the double of the frequency presented in Section 5.3, which is 908MHz, with the goal to normalize the results by the throughput. Section 5.5.1 presents the new results for area overhead, and Section 5.5.2 shows the new results for ATPG.

5.5.1 Normalized Silicon Area Results

This Section analyses the area overhead results for the XTEA Blade version design over the normalized synchronous version of XTEA, shown in Table 5.10. The rows of the table show the number of cells used in the synthesis and the silicon area of these cells. The columns of the table represent the version of the XTEA design and the overheads.

The new results show that the normalized synchronous version has a similar area when compared to the Blade version, which means that a synchronous circuit needs a similar area to achieve a similar throughput like Blade. Table 5.10 shows that the Blade has an overhead of 0.31% (shown in row **5** and column **6**). The Blade with DfT over synchronous with DfT reduced from 139.89% (as shown in the row **12** and column **6** of Table 5.1 at page 84) to 102.33%, once the synchronous DfT overhead increase with the new targeted synthesis frequency.

Table 5.10 – Synthesis results for the XTEA normalized design regarding the flop-based synchronous version, flop-based synchronous version with DfT, Blade version and Blade version with DfT.

		1	2	3	4	5	6
		Comb. cells	**Seq. Cells**	**Total Cells**	**Comb. Area**	**Seq. Area**	**Total Cell Area**
1	**Synchronous**	45,398	6,240	51,638	41,058.50	23,436.17	64,494.68
2	**Synchronous With DfT**	48,976	6,239	55,215	42,399.19	25,442.06	67,841.26
3	**Blade**	31,205	15,644	48,743	39,692.04	25,003.55	64,695.58
4	**Blade With DfT**	31,205	15,756	48,743	39,499.62	97,760.49	137,260.11
5	**Blade/ Synchronous**	-31.26%	150.71%	-5.61%	-3.33%	6.69%	0.31%
6	**Synchronous with DfT/ Synchronous**	7.88%	-0.02%	6.93%	3.27%	8.56%	5.19%
7	**Blade with DfT/ Blade**	0.00%	0.72%	0.00%	-0.48%	290.99%	112.16%
8	**Blade with DfT/ Synchronous with DfT**	-36.29%	152.54%	-11.72%	-6.84%	284.25%	102.33%

5.5.2 Normalized ATPG Results

Table 5.11 shows the stuck-at fault model results extracted from the ATPG tool for the normalized XTEA synchronous version and the XTEA Blade version. Even with the increase of area in the synchronous version, the Blade version still has more fault points that the synchronous one, keeping the number of patterns and clock cycles higher than the Synchronous version.

Table 5.11 – ATPG stuck-at fault model results for XTEA normalized synchronous version and Plasma Blade version.

	Synchronous With DfT	**Blade With DfT**
Fault Coverage (%)	99.99	98.17
Number of total faults	393,570	484,482
Number of collapsed faults	47,549	135,137
Number of scan cells	6,239	15,756
Number of patterns	296	204
Number of cycles	1,853,578	3,230,594

Table 5.12 shows the path-delay fault model results extracted from the ATPG tool for the normalized XTEA synchronous version and XTEA Blade version. This result also covers

1000 combinational paths for both XTEA versions. The test coverage for the synchronous XTEA increase from 51.02% (as shown in Table 5.8 at page 87) to 55.43%. This increase can be explained due to the new logical paths that the synthesis with a bigger frequency produces. The new synthesis process can be divided the logic paths or created more parallel logical paths, fragmenting the logic in small parts, reducing the test complexity. Also, the new synchronous version has fewer patterns and spend fewer clock cycles, when compared to the numbers presented in Table 5.8.

Table 5.12 – ATPG path-delay fault model results for XTEA normalized synchronous version and XTEA Blade version.

	Synchronous With DfT	Blade With DfT
Fault Coverage (%)	55.43	89.37
Number of total faults	1,000	1,000
Number of collapsed faults	208	12
Number of scan cells	6,239	15,756
Number of patterns	181	61
Number of cycles	1,136,225	977,180

5.6 Optimized Plasma For Normalized Throughput Evaluation

This Section presents the results analyses for the Plasma case study. For the Plasma case study, only the normalized results were generated, once it ensures a fair comparison between the synchronous and the Blade versions. The normalized synchronous version has a frequency of 908MHz. Section 5.6.1 shows the normalized area overhead results, and Section 5.6.2 shows the ATPG results.

5.6.1 Normalized Silicon Area Results

This Section analyses the area overhead results for the Plasma Blade version design over the normalized synchronous version of Plasma, shown in Table 5.13. The rows of the table show the number of cells used in the synthesis and the silicon area of these cells. The columns of the table represent the version of the XTEA design and the overheads.

As in the XTEA, the normalized Plasma synchronous version has a similar area when compared to Blade version. For this case, the overhead is 5.11% between the Blade template and the synchronous Plasma (row **5** and column **6**), which means that the Blade conversion has a higher area impact in the Plasma whem compared to the XTEA.

For the Plasma, the Blade DfT overhead is smaller when compared to the XTEA case study. The Blade version of Plasma with DfT has an area overhead of 50.57% (row **7** and column **6**). It occurs because the proportion of combinational cells over sequential cells influences the test overhead, as described in Section 5.4.1. The Plasma synchronous version has 4,262 combinational cells and 529 sequential cells, which means a proportion of 8.05. The Plasma Blade version has 3,603 combinational cells and 1,615 sequential cells, which means a proportion of 2.23. This proportion is higher than the XTEA proportion presented is Section 5.4.1. Thus, the Plasma test overhead can be smaller than the XTEA, once Plasma has less sequential cells when compared to the number of combinational cells.

Table 5.13 – Synthesis results for the Plasma normalized design regarding the flop-based synchronous version, flop-based synchronous version with DfT, Blade version and Blade version with DfT.

		1	**2**	**3**	**4**	**5**	**6**
		Comb. Cells	**Seq. Cells**	**Total Cells**	**Comb. Area**	**Seq. Area**	**Total Cell Area**
1	**Synchronous**	4,262	529	4,792	6,941.87	7,258.48	14,200.35
2	**Synchronous With DfT**	4,234	529	4,764	6,882.96	7,605.60	14,488.56
3	**Blade**	3,603	1,615	5,504	7,067.86	7,858.56	14,926.43
4	**Blade With DfT**	3,603	1,615	5,504	7,076.51	15,398.92	22,475.44
5	**Blade/ Synchronous**	-15.46	205.29	14.86	1.81	8.27	5.11
6	**Synchronous with DfT/ Synchronous**	-0.66	0.00	-0.58	-0.85	4.78	2.03
7	**Blade with DfT/ Blade**	0.00	0.00	0.00	0.12	95.95	50.57
8	**Blade with DfT/ Synchronous with DfT**	-14.90	205.29	15.53	2.81	102.47	55.13

5.6.2 Normalized ATPG Results

Table 5.14 shows the stuck-at fault model results extracted from the ATPG tool for the normalized Plasma synchronous version and the Plasma Blade version. The fault coverage of both versions is very similar, which means that the not testable Blade components have a small influence on the test process when compared to the XTEA study case. As in the XTEA results, the Blade version has more logical paths than the synchronous version, which explains the greater number of patterns and clock cycles.

Table 5.15 shows the path-delay fault model results extracted from the ATPG tool for the normalized Plasma synchronous version and Plasma Blade version. As the XTEA, this result covers 1000 combinational paths for both versions. The Blade version has an improvement in the fault coverage for path-delay faults, which also occurs in the XTEA. The

Table 5.14 – ATPG stuck-at fault model results for Plasma normalized synchronous version and Plasma Blade version.

	Synchronous With DfT	Blade With DfT
Fault Coverage (%)	96.49	96.04
Number of total faults	38,078	53,830
Number of collapsed faults	5,967	14,920
Number of scan cells	529	1,619
Number of patterns	677	1,027
Number of cycles	360,019	1,667,414

reasons for this better fault coverage for path-delay fault model in the Blade template was discussed in Section 5.4.3 at page 5.4.3, and also are considered to the Plasma study case.

Table 5.15 – ATPG path-delay fault model results for Plasma normalized synchronous version and Plasma Blade version.

	Synchronous With DfT	Blade With DfT
Fault Coverage (%)	48.00	99.00
Number of total faults	1,000	1,000
Number of collapsed faults	52	0
Number of scan cells	529	1,619
Number of patterns	363	73
Number of cycles	194,389	1,667,414

6. CONCLUSION

The original Blade version [HTMH+15] does not address test issues in its implementation, and its synthesis flow does not support DfT insertion. The lack of testability in the Blade template reduces its fabrication yield, compromising its manufacturability. By enabling manufacturing test, it might improve Blade's manufacturing viability.

Even though the DfT is fully automated, the proposed approach presents an area overhead of 112.16% for the XTEA circuit and 50.57% for Plasma. This overhead is caused by the LSSD cell, once it presents an area 4.21 times bigger than the standard latch cell. Besides, the Blade flow replaces each flip-flop by two latches, plus the extra latches added by the retiming technique. Thus, as each latch in the circuit is replaced by an LSSD cell, the area overhead for sequential elements increases more than twice.

Section 1.1 presented the main hypothesis of this work which is to check if it is viable to perform structural test approach based on the scan on Blade. The main motivation to insert test circuitry is to ease the manufacturing test and increase the yield. However, the obtained DfT area overhead of more than 100% reduces the viability of the method because of the more silicon area for the circuit, less circuits in the wafer, leading to lower yield, which was one of the primary motivations for DfT. Therefore, based on the collected results, it is questionable whether adding DfT to Blade, with its current flow could actually lead to increased yield.

Unfortunately, the source of this overhead cannot be completely assessed only with the work presented in this document, because all our results are a direct function of the case study circuit employed and the Blade design flow. As we discussed in Section 5.4.1, the major contributor for this overhead is the LSSD. Therefore, for a different case study, with deeper logic paths and a smaller sequential and combinational logic ratio, these overheads may be substantially diluted. Plasma has a a smaller sequential and combinational logic ratio than the XTEA and presents a smaller DfT area overhead. Even though, this area overhead is still considered excessive for DfT purposes. Furthermore, the Blade design flow has limited support to timing constraints, limiting the optimization after transformation from synchronous to Blade and during scan insertion.

In summary, these overhead results show that DfT insertion in Blade needs more study, in order to find a generic solution that increases testability with a smaller area overhead. Nevertheless, the work presented in this document is a step forward towards that goal, as it presents an automated solution for DfT. Accordingly, all environments proposed and presented here can be used for further research and development in DfT flows for Blade. The following Sections describe the contributions to the state of the art (Section 6.1), the current limitations and the future works (Section 6.2).

6.1 Contributions

This work presents a test cell library to latch-based or mixed designs which consists of LSSD cells and a Clocked-LSSD cell. The proposed library was implemented using the 28 nm technology [STM12] and has a Liberty file compatible with commercial synthesis tools.

Besides, this work presents a methodology to allow testability at the Q-Flop component. The methodology shows a way to turn testable the Q-Flop, splitting its component into two cells: a standard flip-flop and a metastability filter. The proposed methodology covers the logic synthesis step and the physical synthesis step. A similar approach was implemented with the original TD, splitting it into a latch and transition detector cell.

Another contribution of this work is an optimization for the Clocked-LSSD cell. Instead of the conventional use of three latches to implement its functionality, the proposed implementation uses two latches to implement the same behavior. The proposed optimization presents area and power reductions when compared to the conventional Clocked-LSSD.

All these contributions were used to implement another contribution of this work, the DfT flow to the Blade template. The synthesis flow of the Blade now has a DfT step, where scannable elements replace all the latches and all Q-Flops. The ATPG results show a fault coverage of 98.17% to stuck-at fault model and 89.37% to path-delay fault model for the XTEA, and a fault coverage of 96.04% to stuck-at fault model and 99.00% to path-delay fault model for the Plasma.

6.2 Current Limitations and Future Work

The current Blade version does not allow logic synthesis optimization after the Blade insertion. This is a limitation inherited from the original Blade flow [HTMH+15], and can compromise the timing constraints of the circuit, making it fail. Thus, is not possible to make an optimization after the DfT insertion as in standard logic synthesis flow. May it is possible to bypass this limitation adding more timing constraints in the circuit parts that must be preserved.

Also, the Blade flow does not address physical synthesis. To perform physical synthesis in Blade, it is necessary the layout of the specific Blade cells, as the C-elements and Transition Detectors. In the Q-Flop case, is necessary only a layout of the Metastability Filter, once the flip-flop layouts are present in the 28 nm technology, as the others conventional cells (latches, ANDs, ORS, etc.). In the test cells cases, it is necessary a layout implementation for each LSSD version and the Clocked-LSSD. These limitations imply some error in the area, performance, and power results. However, this error was minimized using the strategy described in Section 4.4.2. Table 6.1 resumes the implementation level of each cell. All the Verilog files are implemented using UDP, except for TD cell, once it needs a buffer to implement its behavior.

Table 6.1 – Implemented levels of the proposed cells.

	Spice	Liberty	Verilog	Layout
LSSD	Yes	Yes	Yes	No
TD	Yes	Yes	Yes	No
Metastability Filter	Yes	Yes	Yes	No
Clocked-LSSD	Yes	Yes	Yes	No

Another limitation is the improvement of the fault coverage. A set of experiments shows that the LSSD is the main contributor to the reached fault coverage for stuck-at fault model presented in Section 5.4.2. The addition of the Q-Flop in the scan chain has a small improvement in the fault coverage. It occurs because the number of Q-Flops in the design is small when compared with the others memory elements, as demonstrated in Table 5.1 at page 84.

Table 5.7 at page 87 shows that the C-element is the main contributor to the total number of undetected faults. 74,42% of the total undetected faults are related to the C-elements. Thus, it is necessary to replace the C-elements to C-elements scan version. The work [tBP05] shows some approaches to allow scan insertion in the C-elements. This work can consider integrating the scannable C-element at the Blade DfT flow. Another possibility to improve the testability of Blade is to study the Blade controller to find some way to improve its testability.

Another issue is the high overhead caused by the LSSD. An alternative to reduce is the use of the L2* cell presented at Section 2.4.3. An estimate shows that the L2* area is similar to the Clocked-LSSD area, using the same method presented in Section 4.4.2 to estimate the test cell area, which is 6.8 μm^2. Unlike the LSSD, L2* cell replaces each pair of Latches in the design. For the XTEA design using as L2* cell in the DfT insertion, the number of LSSDs is reduced from 15,644 to 7,822. Thus, the sequential area will be reduced from 97,760.49μm^2 to about 54,754μm^2, an overhead reduction of 44%. However, the L2* insertion is not supported by the DfT tools. Thus, it is necessary a method to automate the L2* DfT insertion and ATPG.

Another idea to reduce the area overhead of LSSD is to use an approach similar to presented in [ATA15] and use a pulsed clock to make the latch behave like a flip-flop. Thus, its possible to perform the test operation using a single latch, as occurs in the Scan Razor Flip-flop showed at Figure 3.6 at page 55.

Moreover, Blade deals better with specific kinds of circuit, such as pipelined designs, due to the limitations mentioned before. That justifies the use of few study cases to evaluate this work, once their implementations are complex and require time. Besides, the synthesis flow designed for Blade was done targeting the Plasma microprocessor, and requires modifications to support others designs. Hence, others study cases will be implemented and integrated into the Blade DfT flow.

REFERENCES

[ABF94] Abramovici, M.; Breuer, M. A.; Friedman, A. D. “Digital Systems Testing And Testable Design”. Wiley-IEEE Press, 1994, 1 ed., 653p.

[ATA15] Anastasiou, A.; Tsiatouhas, Y.; Arapoyanni, A. “On the reuse of existing error tolerance circuitry for low power scan testing”. In: IEEE International Symposium on Circuits and Systems (ISCAS), 2015, pp. 1578–1581.

[BA02] Bushnell, M. L.; Agrawal, V. D. “Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuit”. Springer Science & Business Media, 2002, 1 ed., 690p.

[BCC+14] Beer, S.; Cannizzaro, M.; Cortadella, J.; Ginosar, R.; Lavagno, L. “Metastability in better-than-worst-case designs”. In: IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2014, pp. 101–102.

[BMS+02] Bailey, B.; Metayer, A.; Svrcek, B.; Tendolkar, N.; Wolf, E.; Fiene, E.; Alexander, M.; Woltenberg, R.; Raina, R. “Test methodology for motorola’s high performance e500 core based on PowerPC instruction set architecture”. In: IEEE International Test Conference (ITC), 2002, pp. 574–583.

[BOF10] Beerel, P.; Ozdag, R.; Ferretti, M. “A Designer’s Guide to Asynchronous VLSI”. Cambridge University Press, 2010, 1 ed., 352p.

[BTK+09] Bowman, K. A.; Tschanz, J. W.; Kim, N. S.; Lee, J. C.; Wilkerson, C. B.; Lu, S.-L. L.; Karnik, T.; De, V. K. “Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance”, *IEEE Journal of Solid-State Circuits*, vol. 44–1, 2009, pp. 49–63.

[Cad11] Cadence. “Known Problems and Solutions in Encounter® RTL Compiler, Version 10.1”. 2011.

[CBC+15] Cannizzaro, M.; Beer, S.; Cortadella, J.; Ginosar, R.; Lavagno, L. “Saferazor: Metastability-robust adaptive clocking in resilient circuits”, *IEEE Transactions on Circuits and Systems*, vol. 62–9, 2015, pp. 2238–2247.

[CKLS06] Cortadella, J.; Kondratyev, A.; Lavagno, L.; Sotiriou, C. P. “Desynchronization: Synthesis of asynchronous circuits from synchronous specifications”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25–10, 2006, pp. 1904–1921.

[CVG07] Chelcea, T.; Venkataramani, G.; Goldstein, S. C. “Area optimizations for dual-rail circuits using relative-timing analysis”. In: IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2007, pp. 117–128.

[DKM05] Drake, A. J.; KleinOsowski, A.; Martin, A. K. “A self-correcting soft error tolerant flop-flop”. In: NASA Symposium on VLSI Design, 2005, pp. 4–5.

[DPH^{+}12] Dillen, S. J.; Priore, D.; Horiuchi, A. K.; Naffziger, S. D.; et al.. “Design and implementation of soft-edge flip-flops for x86-64 AMD microprocessor modules”. In: IEEE Custom Integrated Circuits Conference (CICC), 2012, pp. 1–4.

[DTP^{+}09] Das, S.; Tokunaga, C.; Pant, S.; Ma, W.-H.; Kalaiselvan, S.; Lai, K.; Bull, D. M.; Blaauw, D. T. “RazorII: In situ error detection and correction for PVT and SER tolerance”, *IEEE Journal of Solid-State Circuits*, vol. 44–1, 2009, pp. 32–48.

[EBE05] Efthymiou, A.; Bainbridge, J.; Edwards, D. “Test pattern generation and partial-scan methodology for an asynchronous SoC interconnect”, *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 13–12, 2005, pp. 1384–1393.

[EKD^{+}03] Ernst, D.; Kim, N. S.; Das, S.; Pant, S.; Rao, R.; Pham, T.; Ziesler, C.; Blaauw, D.; Austin, T.; Flautner, K.; et al.. “Razor: A low-power pipeline based on circuit-level timing speculation”. In: IEEE/ACM International Symposium on Microarchitecture (MICRO), 2003, pp. 7–18.

[EPS06] Elakkumanan, P.; Prasad, K.; Sridhar, R. “Time redundancy based scan flip-flop reuse to reduce SER of combinational logic”. In: IEEE International Symposium on Quality Electronic Design (ISQED), 2006, pp. 6–pp.

[EW77] Eichelberger, E. B.; Williams, T. W. “A logic design structure for LSI testability”. In: Design Automation Conference (DAC), 1977, pp. 462–468.

[FFK^{+}13] Fojtik, M.; Fick, D.; Kim, Y.; Pinckney, N.; Harris, D. M.; Blaauw, D.; Sylvester, D. “Bubble razor: Eliminating timing margins in an ARM cortex-M3 processor in 45 nm CMOS using architecturally independent error detection and correction”, *IEEE Journal of Solid-State Circuits*, vol. 48–1, 2013, pp. 66–81.

[FTK08] Floros, A.; Tsiatouhas, Y.; Kavousianos, X. “The time dilation scan architecture for timing error detection and correction”. In: IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), 2008, pp. 569–574.

[Gin11] Ginosar, R. “Metastability and synchronizers: A tutorial”, *IEEE Design & Test of Computers*, vol. 28–5, 2011, pp. 23–35.

[Har00] Harris, D. “Skew-Tolerant Circuit Design”. Morgan Kaufmann, 2000, 1 ed., 300p.

[HBB+05] Haring, R. A.; Bellofatto, R.; Bright, A. A.; Crumley, P.; Dombrowa, M. B.; Douskey, S. M.; Ellavsky, M. R.; Gopalsamy, B.; Hoenicke, D.; Liebsch, T. A.; et al.. “Blue gene/l compute chip: Control, test, and bring-up infrastructure”, *IBM Journal of Research and Development*, vol. 49–2.3, 2005, pp. 289–301.

[HGJX13] Han, Q.; Guo, J.; Jone, W.-B.; Xu, Q. “Path delay testing in resilient system”. In: IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), 2013, pp. 645–648.

[HTMH+15] Hand, D.; Trevisan Moreira, M.; Huang, H.-H.; Chen, D.; Butzke, F.; Li, Z.; Gibiluka, M.; Breuer, M.; Vilar Calazans, N.; Beerel, P. “Blade - a timing violation resilient asynchronous template”. In: IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2015, pp. 21–28.

[JMKA17] Juracy, L. R.; Moreira, M. T.; Kuentzer, F. A.; Amory, A. M. “Optimized design of an LSSD scan cell”, *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 25–2, 2017, pp. 765–768.

[JOC07] Jagirdar, A.; Oliveira, R.; Chakraborty, T. J. “Efficient flip-flop designs for SET/SEU mitigation with tolerance to crosstalk induced signal delays”. In: IEEE Silicon Errors Logic System Effects (SELSE), 2007, pp. 1–6.

[Kap08] Kaps, J.-P. “Chai-tea, cryptographic hardware implementations of XTEA.” In: International Conference on Cryptology in India (INDOCRYPT), 2008, pp. 363–375.

[KDF+04] Kuppuswamy, R.; DesRosier, P.; Feltham, D.; Sheikh, R.; Thadikaran, P. “Full hold-scan systems in microprocessors: Cost/benefit analysis.”, *Intel Technology Journal*, vol. 8–1, 2004, pp. 69–78.

[KKF+14] Kwon, I.; Kim, S.; Fick, D.; Kim, M.; Chen, Y.-P.; Sylvester, D. “Razor-lite: a light-weight register for error detection by observing virtual supply rails”, *IEEE Journal of Solid-State Circuits*, vol. 49–9, 2014, pp. 2054–2066.

[Kue18] Kuentzer, F. A. “More than a timing resilient template: A case study on reliability-oriented improvements on blade”, Ph.D. Thesis, Pontifical Catholic University of Rio Grande do Sul, 2018, 94p.

[MM07] Mullins, R.; Moore, S. “Demystifying data-driven and pausible clocking schemes”. In: IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2007, pp. 175–185.

[MSZ+05] Mitra, S.; Seifert, N.; Zhang, M.; Shi, Q.; Kim, K. S. "Robust system design with built-in soft-error resilience", *IEEE Computer*, vol. 38–2, 2005, pp. 43–52.

[MZ00] Mourad, S.; Zorian, Y. "Principles of testing electronic systems". John Wiley & Sons, 2000, 1 ed., 420p.

[NS15] Nowick, S. M.; Singh, M. "Asynchronous design - part 1: Overview and recent advances", *IEEE Design Test*, vol. 32–3, June 2015, pp. 5–18.

[NW97] Needham, R. M.; Wheeler, D. J. "TEA extensions", *Report, Cambridge University, Cambridge, UK*, 1997.

[PAG+99] Pyron, C.; Alexander, M.; Golab, J.; Joos, G.; Long, B.; Molyneaux, R.; Raina, R.; Tendolkar, N. "DFT advances in the motorola's MPC7400, a PowerPC TM G4 microprocessor". In: IEEE International Test Conference (ITC), 1999, pp. 137–146.

[PtBdWM10] Peeters, A.; te Beest, F.; de Wit, M.; Mallon, W. "Click elements: An implementation style for data-driven compilation". In: IEEE Symposium on Asynchronous Circuits and Systems (ASYNC), 2010, pp. 3–14.

[RCN03] Rabaey, J. M.; Chandrakasan, A.; Nikolic, B. "Digital Integrated Circuits". Pearson, 2003, 2 ed., 761p.

[RMCF88] Rosenberger, F. U.; Molnar, C. E.; Chaney, T. J.; Fang, T.-P. "Q-modules: Internally clocked delay-insensitive modules", *IEEE Transactions on Computers*, vol. 37–9, 1988, pp. 1005–1018.

[Sav86] Savir, J. "The bidirectional double latch (BDDL)", *IEEE Transactions on Computers*, –1, 1986, pp. 65–66.

[Sav97a] Savir, J. "Reduced latch count shift registers", *Journal of Electronic Testing*, vol. 11–2, 1997, pp. 183–185.

[Sav97b] Savir, J. "Scan latch design for delay test". In: IEEE Test Conference International (ITC), 1997, pp. 446–453.

[SF01] Sparso, J.; Furber, S. "Principles of Asynchronous Circuit Design: A Systems Perspective". Springer Publishing Company, Incorporated, 2001, 1 ed., 337p.

[SN07] Singh, M.; Nowick, S. "MOUSETRAP: High-speed transition-signaling asynchronous pipelines", *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 15–6, 2007, pp. 684–698.

[SP11] Shin, Y.; Paik, S. "Pulsed-latch circuits: A new dimension in ASIC design", *IEEE Design & Test of Computers*, vol. 6–28, 2011, pp. 50–57.

[Spa01] Sparso, J. “Asynchronous circuit design - A tutorial”. Kluwer Academic Publishers, 2001, 1 ed., 152p.

[SS03] Sheth, A. M.; Savir, J. “Single-clock, single-latch, scan design”, *IEEE Transactions on Instrumentation and Measurement*, vol. 52–5, 2003, pp. 1455–1457.

[STM12] STMicroelectronics. “C28SOI_SC_12_CORE_LR Databook: 12 track Standard Cell Library comprising commonly used booleans and sequential cells”, 2012.

[Sut89] Sutherland, I. E. “Micropipelines”, *Communications of the ACM*, vol. 32–6, Jun 1989, pp. 720–738.

[SYN14] SYNOPSYS. “IC Compiler User Guide, Version J-2014.09-SP1,”. 2014.

[SYN15a] SYNOPSYS. “Design Compiler® User Guide, Version K-2015.06”. 2015.

[SYN15b] SYNOPSYS. “DFT Compiler, DFTMAX™, and DFTMAX™ Ultra User Guide, Version K-2015.06”. 2015.

[tBP05] te Beest, F.; Peeters, A. “A multiplexer based test method for self-timed circuits”. In: IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2005, pp. 166–175.

[TBW+09] Tschanz, J.; Bowman, K.; Wilkerson, C.; Lu, S.-L.; Karnik, T. “Resilient circuits: enabling energy-efficient performance and reliability”. In: ACM International Conference on Computer-Aided Design (ICCAD), 2009, pp. 71–73.

[VFT+14] Valadimas, S.; Floros, A.; Tsiatouhas, Y.; Arapoyanni, A.; Kavousianos, X. “The time dilation technique for timing error tolerance”, *IEEE Transactions on Computers*, vol. 63–5, 2014, pp. 1277–1286.

[WKP+02] Warnock, J. D.; Keaty, J. M.; Petrovick, J.; Clabes, J. G.; Kircher, C. J.; Krauter, B. L.; Restle, P. J.; Zoric, B. A.; Anderson, C. J. “The circuit and physical design of the POWER4 microprocessor”, *IBM Journal of Research and Development*, vol. 46–1, 2002, pp. 27–51.

[WWW06] Wang, L.-T.; Wu, C.-W.; Wen, X. “VLSI Test Principles and Architectures”. Morgan Kaufmann, 2006, 1 ed., 808p.

[YK04] Yoshikawa, K.; Kanamaru, K. “Timing optimization by replacing flip-flops to latches”. In: Asia and South Pacific Design Automation Conference (ASP-DAC), 2004, pp. 186–191.

[Yur95] Yurash, S. A. “Dual latch clocked LSSD and method”. US Patent 5,463,338, Oct 31 1995.

[ZM01] Zyuban, V.; Meltzer, D. “Clocking strategies and scannable latches for low power appliacations”. In: ACM International Symposium on Low Power Electronics and Design (ISLPED), 2001, pp. 346–351.

[ZUC01] Zarrineh, K.; Upadhyaya, S. J.; Chickermane, V. “System-on-chip testability using LSSD scan structures”, *IEEE Design & Test of Computers*, vol. 18–3, 2001, pp. 83–97.

APPENDIX A – LSSD DESIGN AND IMPLEMENTATION

This appendix shows the design and the implementation of the LSSD cells used in the Blade DfT flow. As mentioned before, four types of LSSD were implemented:

- An active low LSSD;
- An active high LSSD;
- An active low LSSD with active low reset;
- An active high LSSD with active low reset.

Figure A.1 shows the transistor diagram of an active high LSSD without reset. The values highlighted in blue represent the transistor sizing, and it is implemented using 36 transistors. Figure A.2 shows the transistor diagram of an active high LSSD with reset. The values highlighted in blue represent the transistor sizing, and it is implemented using 40 transistors. The reset function is implemented using a pull-up network. The active low version of the LSSDs uses the same transistor architecture. However, the enable signal **C** is connected to PMOS transistors, while the signal **nC** is connected to NMOS transistors.

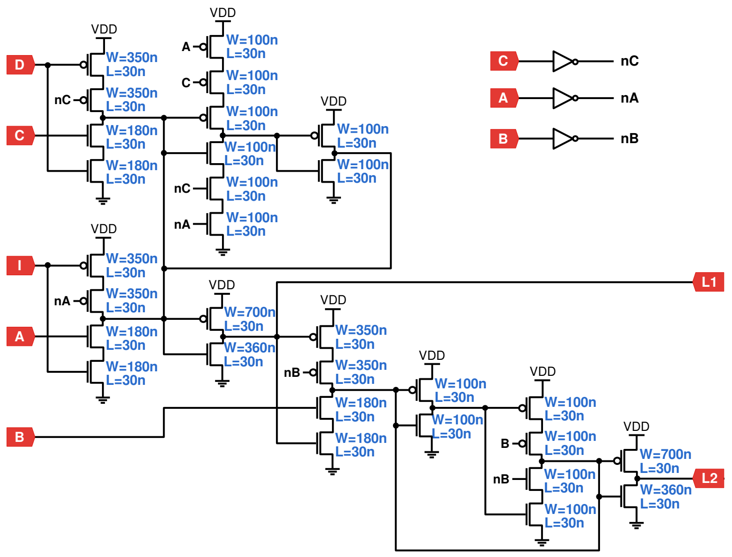


Figure A.1 – LSSD using 28 nm technology node.

Figure A.3 shows the validation of the LSSD behavior in a waveform. This waveform was captured from Spice simulation. In the normal operation, the values at the input **D** are copied to the output **L1** when **C** is high. In test mode, the values at the input **I** are copied to the output **L1** when **A** is high and the values at **L1** are copied to **L2** output when **B** is high.

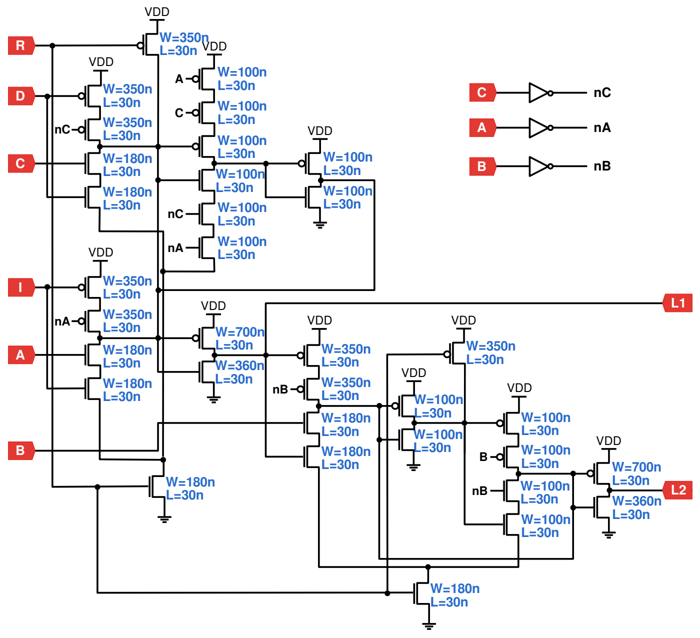


Figure A.2 – LSSD using 28 nm technology node with reset.

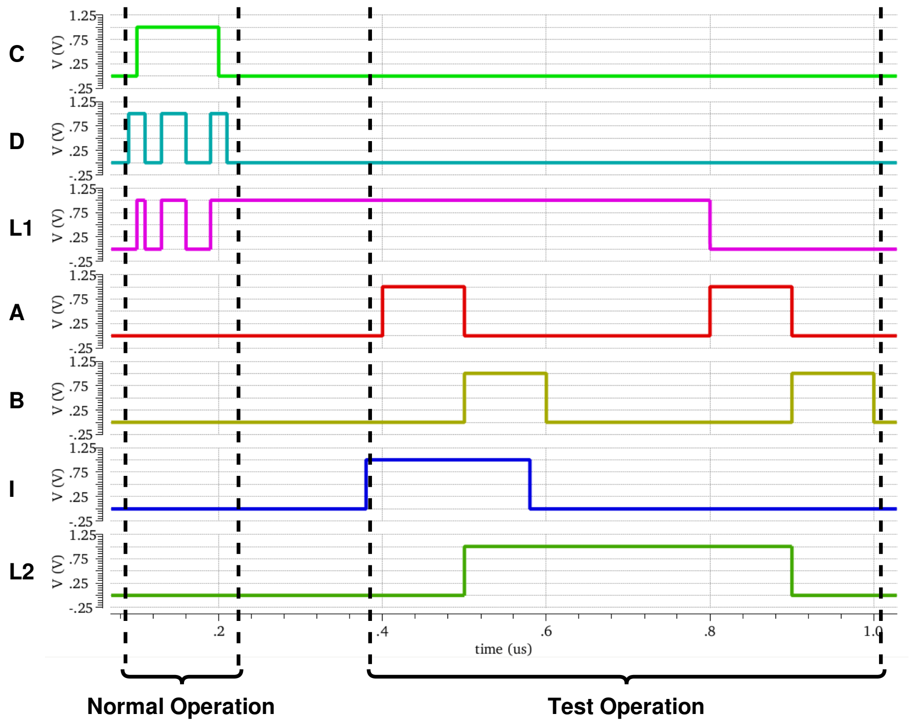


Figure A.3 – LSSD using 28 nm technology waveform.

The characterization was performed using the Liberate by Cadence. In the characterization process, some delay arcs must be defined to ensure the correct liberty generation. The first step is to define the arcs to characterize setup times. Figure A.4 shows the setup time definition between the arcs $D \rightarrow C$, $I \rightarrow A$, and $I \rightarrow B$.

```
# D => C
define_arc −vector {RRXFFRX} −pin D −related_pin C −type setup −probe L1 SD_TLSSDSL_X1
define_arc −vector {FRXFFFX} −pin D −related_pin C −type setup −probe L1 SD_TLSSDSL_X1

# I => A
define_arc −vector {XFRRFRX} −pin I −related_pin A −type setup −probe L1 SD_TLSSDSL_X1
define_arc −vector {XFFRFFX} −pin I −related_pin A −type setup −probe L1 SD_TLSSDSL_X1

# I => B
define_arc −vector {XXRRRXR} −pin I −related_pin B −probe L1 −type setup SD_TLSSDSL_X1
define_arc −vector {XXFRRXF} −pin I −related_pin B −probe L1 −type setup SD_TLSSDSL_X1
```

Figure A.4 – LSSD using 28 nm technology setup characterization time arcs definition.

The second step defines the time arcs between the inputs and the outputs to define the arcs delays. Figure A.5 shows the arc definitions to $C \rightarrow L1$, $A \rightarrow L1$, $B \rightarrow L2$, $D \rightarrow L1$, $I \rightarrow L1$, $D \rightarrow L2$ and $I \rightarrow L2$.

```
# C => L1
define_arc −vector {RRXFFRX} −related_pin C −pin L1 SD_TLSSDSL_X1
define_arc −vector {FRXFFFX} −related_pin C −pin L1 SD_TLSSDSL_X1

# D => L1
define_arc −vector {RRXFFRX} −related_pin D −pin L1 SD_TLSSDSL_X1
define_arc −vector {FRXFFFX} −related_pin D −pin L1 SD_TLSSDSL_X1

# A => L1
define_arc −vector {XFRRFRX} −related_pin A −pin L1 SD_TLSSDSL_X1
define_arc −vector {XFFRFFX} −related_pin A −pin L1 SD_TLSSDSL_X1

# I => L1
define_arc −vector {XFRRFRX} −related_pin I −pin L1 SD_TLSSDSL_X1
define_arc −vector {XFFRFFX} −related_pin I −pin L1 SD_TLSSDSL_X1

# B => L2
define_arc −vector {XXXXRXR} −related_pin B −pin L2 SD_TLSSDSL_X1
define_arc −vector {XXXXRXF} −related_pin B −pin L2 SD_TLSSDSL_X1

# D => L2
define_arc −vector {RRXXXXR} −related_pin D −pin L2 SD_TLSSDSL_X1
define_arc −vector {FRXXXXF} −related_pin D −pin L2 SD_TLSSDSL_X1

# I => L2
define_arc −vector {XXRXXXR} −related_pin I −pin L2 SD_TLSSDSL_X1
define_arc −vector {XXFXXXF} −related_pin I −pin L2 SD_TLSSDSL_X1
```

Figure A.5 – LSSD using 28 nm technology delay characterization time arcs definition.

Both setup arcs definitions and delay arc definitions use vectors to define the inputs and outputs values needed to stimulate the required arc. The "R" vectors mean a transition of the zero value to one, the "F" means a transition of one to zero and the "X" means a don't care value. Thus, its possible to generate a correct file to the LSSD cells.

However, the Liberate does not identify the LSSD as a test cell, and can not recognize the cell function. Some manual modifications are needed to allow the synthesis tool recognize the LSSD as a test cell. First, it is required to add the function *test_cell* in the file to notify the synthesis tool that the cell is a test cell. Second, is required a *statetable* function to allow the tool recognize the LSSD function, once the LSSD has a not common function as latches and flip-flops. Finally, it is necessary to add in the output pins the internal nodes correspondents to the state table. Thus, the synthesis tool can recognize the LSSD liberty file and use it to insert DfT. Figure A.6 shows this modification in a sample of a liberty code.

```
library (LSSD) {
...
statetable("C D   A B I ",        " L1 L2 ") {
  table :  "L -   L - -   : -   - : N   -,\
            H L/H L - -   : -   - : L/H -,\
            L -   H - L/H : -   - : L/H -,\
            H -   H - -   : -   - : X   -,\
            - -   - L -   : -   - : -   N,\
            - -   - H -   : L/H - : -   L/H";
}
test_cell() {
   pin(C)  {
     direction : input;
   }
   pin(D)  {
     direction : input;
   }
   pin(L1)  {
     direction : output;
     function : "IQ";
   }
   pin(A)  {
     direction : input;
     signal_type : "test_scan_clock_a";
   }
   pin(I)  {
     direction : input;
     signal_type : "test_scan_in";
   }
   pin(B)  {
     direction : input;
     signal_type : "test_scan_clock_b";
   }
   pin(L2)  {
     direction : output;
     signal_type : "test_scan_out";
   }
   latch (IQ,IQN) {
     data_in : "D";
     enable : "C";
   }
}
pin (L1) {
   direction : output;
   internal_node : "L1";
...
pin (L2) {
   direction : output;
   internal_node : "L2";
...
}
```

Figure A.6 – LSSD liberty code sample presenting the manual modification necessary to the synthesis tool recognize the LSSD function. The *statetable* and the *test_cell* functions are added manually after the characterization.

APPENDIX B – TRANSITION DETECTOR CODES

This Sections details the Spice code, the Verilog code, and shows a sample of the liberty file. Figure B.1 shows the Verilog code to the TD cell. As the TD cell has a delay element (**DI2**), the Verilog code was described using the cells of the 28 nm technology library, an XOR gate (**C12T28SOI_LR_XOR2X8_P0**) and a buffer (**C12T28SOI_LR_BFX8_P0**). For ATPG process, the internal nodes of the Verilog are removed in the fault analyses, once this component must be considered a single cell. Figure B.2 presents a sample of the liberty file. This liberty file was characterized as detailed in Section A. Figure B.3 shows the Spice description for the TD cell. The buffer **DI2** was designed with successive simulations to find the correct length in terms of inverters.

```
module BLADE_TD ( D, X );
  input D;
  output X;
  wire   INTERNAL1;
  C12T28SOI_LR_BFX8_P0 buf1 ( .A(D), .Z(INTERNAL1) );
  C12T28SOI_LR_XOR2X8_P0 xor1 ( .A(INTERNAL1), .B(D), .Z(X) );
endmodule
```

Figure B.1 – Verilog code to Transition Detector cell.

```
...
  cell (TD) {
    area : 1.1312;
    cell_leakage_power : 262.149;
    pg_pin (gnd) {
      pg_type : primary_ground;
      voltage_name : "gnd";
    }
    pg_pin (vdd) {
      pg_type : primary_power;
      voltage_name : "vdd";
    }
    leakage_power () {
      value : 0;
      when : "(D * X)";
      related_pg_pin : gnd;
    }
    leakage_power () {
      value : 308.767;
      when : "(D * X)";
      related_pg_pin : vdd;
    }
..
    pin (X) {
        direction : output;
..
    pin (D) {
        direction : input;
```

Figure B.2 – Transition Detector liberty code sample.

```
.SUBCKT TD D X gnd vdd
*******************
********DI2********
*******************
*Buffer
M27 o5 D vdd vdd pfet w=1.2u l=0.03u
M28 o5 D gnd gnd nfet w=0.5u  l=0.03u

M41 o6 o5 vdd vdd pfet w=1.2u l=0.03u
M42 o6 o5 gnd gnd nfet w=0.5u  l=0.03u

M43 o6 o5 vdd vdd pfet w=1.2u l=0.03u
M44 o6 o5 gnd gnd nfet w=0.5u  l=0.03u

M45 o7 o6 vdd vdd pfet w=1.2u l=0.03u
M46 o7 o6 gnd gnd nfet w=0.5u  l=0.03u

M47 o8 o7 vdd vdd pfet w=1.2u l=0.03u
M48 o8 o7 gnd gnd nfet w=0.5u  l=0.03u

M49 o9 o8 vdd vdd pfet w=1.2u l=0.03u
M50 o9 o8 gnd gnd nfet w=0.5u  l=0.03u

M51 o10 o9 vdd vdd pfet w=1.2u l=0.03u
M52 o10 o9 gnd gnd nfet w=0.5u  l=0.03u

M53 o11 o10 vdd vdd pfet w=1.2u l=0.03u
M54 o11 o10 gnd gnd nfet w=0.5u  l=0.03u

M55 o12 o11 vdd vdd pfet w=1.2u l=0.03u
M56 o12 o11 gnd gnd nfet w=0.5u  l=0.03u

M57 o13 o12 vdd vdd pfet w=1.2u l=0.03u
M58 o13 o12 gnd gnd nfet w=0.5u  l=0.03u

M59 o14 o13 vdd vdd pfet w=1.2u l=0.03u
M60 o14 o13 gnd gnd nfet w=0.5u  l=0.03u

M61 o15 o14 vdd vdd pfet w=1.2u l=0.03u
M62 o15 o14 gnd gnd nfet w=0.5u  l=0.03u

M63 o16 o15 vdd vdd pfet w=1.2u l=0.03u
M64 o16 o15 gnd gnd nfet w=0.5u  l=0.03u

M65 o17 o16 vdd vdd pfet w=1.2u l=0.03u
M66 o17 o16 gnd gnd nfet w=0.5u  l=0.03u

M29 oN o17 vdd vdd pfet  w=1.2u l=0.03u
M30 oN o17 gnd gnd nfet  w=0.5u l=0.03u

*******************
*********X*********
*******************
*nD
M19 nD D vdd vdd pfet w=21.6u l=0.03u
M20 nD D gnd gnd nfet w=9.6u  l=0.03u

*noN
M31 noN oN vdd vdd pfet w=21.6u l=0.03u
M32 noN oN gnd gnd nfet w=9.6u  l=0.03u

*XOR1
M33 xor1 noN  vdd  vdd pfet w=21.6u l=0.03u
M34 xor2  oN  vdd  vdd pfet w=21.6u l=0.03u
M35    X   D  xor1 vdd pfet w=21.6u l=0.03u
M36    X  nD  xor2 vdd pfet w=21.6u l=0.03u
M37    X  oN  xor3 gnd nfet w=9.6u l=0.03u
M38    X noN  xor4 gnd nfet w=9.6u l=0.03u
M39 xor3   D   gnd gnd nfet w=9.6u l=0.03u
M40 xor4  nD   gnd gnd nfet w=9.6u l=0.03u

.ENDS
```

Figure B.3 – Spice code to Transition Detector cell.

APPENDIX C – METASTABILITY FILTER CODES

This Sections details the Spice code, the Verilog code, and shows a sample of the liberty file. Figure C.1 shows the Verilog UDP code to the Metastability cell. Figure C.2 presents a sample of the liberty file. This liberty file was characterized as detailed in Section A. Figure C.3 shows the Spice description for the Metastability Filter cell.

```
primitive q_filter (G0, R0, R1);
output G0;
input R0, R1;
table
//A     B       :Q+1
0       0       : 0 ;
0       1       : 0 ;
1       0       : 0 ;
1       1       : 1 ;
endtable
endprimitive

primitive qbar_filter (G1, R0, R1);
output G1;
input R0, R1;
table
//A     B       :Q+1
0       0       : 0 ;
0       1       : 0 ;
1       0       : 1 ;
1       1       : 0 ;
endtable
endprimitive

module METASTABILITY_FILTER (Q, QBAR, GN, D);
input GN, D;
output Q, QBAR;
q_filter    i1 (Q, GN, D);
qbar_filter i2 (QBAR, GN, D);
specify
(GN => Q)     = (0.01,0.01);
(GN => QBAR)  = (0.01,0.01);
(D  => Q)     = (0.01,0.01);
(D  => QBAR)  = (0.01,0.01);
endspecify
endmodule
```

Figure C.1 – Verilog User Defined Primitives code to Metastability Filter cell.

```
...
cell (METASTABILITY_FILTER) {
    area : 0;
    cell_leakage_power : 7.94166;
    pg_pin (gnd) {
      pg_type : primary_ground;
      voltage_name : "gnd";
    }
    pg_pin (vdd) {
      pg_type : primary_power;
      voltage_name : "vdd";
    }
    leakage_power () {
      value : 0;
      when : "(GN * D * Q * Qbar)";
      related_pg_pin : gnd;
    }
    leakage_power () {
      value : 10.44;
      when : "(GN * D * Q * Qbar)";
      related_pg_pin : vdd;
    }
...
    pin (Q) {
      direction : output;
...
    pin (Qbar) {
      direction : output;
...
   pin (D) {
      direction : input;
...
    pin (GN) {
      direction : input;
```

Figure C.2 – Metastability Filter liberty code sample.

```
.SUBCKT METASTABILITY_FILTER GN D Q Qbar gnd vdd
*******************
********INV********
*******************
M07 Dn D vdd vdd pfet w=0.70u l=0.03u
M08 Dn D gnd gnd nfet w=0.36u l=0.03u
*******************
********NAND0******
*******************
M23 vdd D   n1  vdd pfet w=0.70u l=0.03u
M24 vdd GN  n1  vdd pfet w=0.70u l=0.03u
M25 n1  D       n2  gnd nfet w=0.36u l=0.03u
M26 n2  GN  gnd gnd nfet w=0.36u l=0.03u
*******************
********NAND1******
*******************
M29 vdd Dn n3  vdd pfet w=0.70u l=0.03u
M30 vdd GN n3  vdd pfet w=0.70u l=0.03u
M31 n3  Dn n4  gnd nfet w=0.36u l=0.03u
M32 n4  GN gnd gnd nfet w=0.36u l=0.03u
*******************
******FILTER*******
*******************
*Q1
M35 Qbar n3 n1 vdd pfet w=0.70u l=0.03u
M36 Qbar n3 gnd gnd nfet w=0.36u l=0.03u
*Q0
M37 Q n1 n3 vdd pfet w=0.70u l=0.03u
M38 Q n1 gnd gnd nfet w=0.36u l=0.03u
.ENDS
```

Figure C.3 – Spice code for Metastability Filter cell.

APPENDIX D – PROPOSED CLOCKED-LSSD CODES AND SCRIPTS

This Sections details the Spice code, the Verilog code, and shows a sample of the liberty file. Figure D.1 shows the Verilog UDP code to the proposed Clocked-LSSD. Figure D.2 presents a sample of the liberty file. This liberty file was characterized as detailed in Section A. Also, liberty needs the same manual modification detailed in Section A. Figure D.3 shows the Spice description for the proposed Clocked-LSSD cell. The characterization process was performed using the Liberate by Cadence. The characterization steps are described in Section A. Figure D.4 shows the timing arcs of the Clocked-LSSD.

```
primitive udp_double_latch(L2, D1, C1, D2, C2);
        output L2;
        input D1, C1, D2, C2;
        reg L2;
        table
        // D1 C1 D2 C2 : L2 : +L2 ;
                ? 0 ? 0 : ?  : -   ;
                0 1 ? 0 : ?  : 0   ;
                1 1 ? 0 : ?  : 1   ;
                ? 0 0 1 : ?  : 0   ;
                ? 0 1 1 : ?  : 1   ;
        endtable
endprimitive

primitive udp_latch(Q, D, C) ;
        output Q;
        input D, C;
        reg Q;
        table
        //  D C  : L2 : +L2 ;
                ?  0  : ?  : -   ;
                0  1  : ?  : 0   ;
                1  1  : ?  : 1   ;
        endtable
endprimitive

module S_CLOCKED_LSSD (L2, D, GN, A, B, I);
        input D, GN, A, B, I;
        output L2;
        udp_latch(D_internal, D, GN);
    udp_latch(I_internal, I, A);
        udp_double_latch(L2, D_internal, GN, I_internal, B);
endmodule
```

Figure D.1 – Verilog User Defined Primitives code to Clocked-LSSD cell.

```
...
  cell (CLOCKED_LSSD) {
    area : 7;
    cell_leakage_power : 6.57219;
    pg_pin (gnd) {
      pg_type : primary_ground;
      voltage_name : "gnd";
    }
    pg_pin (vdd) {
      pg_type : primary_power;
      voltage_name : "vdd";
    }
...
    test_cell() {
pin(D,GN)  {
  direction : input;
}
pin(A)  {
  direction : input;
  signal_type : "test_scan_clock_a";
}
        pin(B)  {
  direction : input;
  signal_type : "test_scan_clock_b";
}
pin(I)  {
  direction : input;
  signal_type : "test_scan_in";
}
pin(L2)  {
  direction : output;
  function : "IQ";
  signal_type : "test_scan_out";
}
ff ("IQ","IQN") {
 next_state : "D";
 clocked_on : "GN";
 }
    }
   statetable ("GN D   A B  I",          "MQ  L2") {
      table : " R  H/L L L  -   : -   - : H/L H/L,\
               ~R   -   - H   -   : H/L - :  -  H/L,\
               ~R   -   - L   -   : -   - :  -  N,\
                R   -   H -   -   : -   - :  X  X,\
                R   -   - H   -   : -   - :  X  X,\
               ~R   -   H -  H/L : -   - : H/L -,\
               ~R   -   L -   -   : -   - :  N  -";
   }
    pin (MQ) {
      direction : internal;
      internal_node : "MQ";
    }

    pin (L2) {
      direction : output;
..
    pin (A) {
      clock : true;
      direction : input;
...
    pin (B) {
      clock : true;
      direction : input;
...
    pin (D) {
      direction : input;
...
    pin (GN) {
      clock : true;
      direction : input;
...
    pin (I) {
      direction : input;
...
```

Figure D.2 – Proposed Clocked-LSSD liberty code sample.

```
.SUBCKT CLOCKED_LSSD GN A B D I L2 gnd vdd
**************
*DOUBLE LATCH*
**************
*nGN
M01 nGN GN vdd vdd pfet  w=0.35u l=0.03u
M02 nGN GN gnd gnd nfet  w=0.18u l=0.03u

*nA
M03 nA A vdd vdd pfet  w=0.35u l=0.03u
M04 nA A gnd gnd nfet  w=0.18u l=0.03u

*X0
M05 i0 D   vdd vdd pfet w=0.35u l=0.03u
M41 x0 A   i0  vdd pfet w=0.35u l=0.03u
M06 i1 GN  x0  vdd pfet w=0.35u l=0.03u
M07 i1 nGN x1  gnd nfet w=0.18u l=0.03u
M42 x1 nA  i2  vdd nfet w=0.35u l=0.03u
M08 i2 D   gnd gnd nfet w=0.18u l=0.03u

*X1
M09 i3 I  vdd vdd  pfet w=0.35u l=0.03u
M10 i1 nA i3 vdd  pfet w=0.35u l=0.03u
M11 i1 A  i4 gnd  nfet w=0.18u l=0.03u
M12 i4 I  gnd gnd  nfet w=0.18u l=0.03u

*X3
M13 i5 i1 vdd vdd pfet w=0.1u l=0.03u
M14 i5 i1 gnd gnd nfet w=0.1u l=0.03u

*X2
M15 i6 nGN vdd vdd pfet w=0.1u l=0.03u
M16 i7 A   i6  vdd pfet w=0.1u l=0.03u
M17 i1 i5  i7  vdd pfet w=0.1u l=0.03u
M18 i1 i5  i8  gnd nfet w=0.1u l=0.03u
M19 i8 GN  i9  gnd nfet w=0.1u l=0.03u
M20 i9 nA  gnd gnd nfet w=0.1u l=0.03u

*X4
M21 DL i1 vdd vdd pfet w=0.70u l=0.03u
M22 DL i1 gnd gnd nfet w=0.36u l=0.03u

***************
*SPECIAL LATCH*
***************
*nB
M23 nB B vdd vdd pfet   w=0.35u l=0.03u
M24 nB B gnd gnd nfet   w=0.18u l=0.03u

*X5
M25 i10 nGN vdd vdd pfet w=0.35u l=0.03u
M26 i10 nB  vdd vdd pfet w=0.35u l=0.03u
M27 i12 DL  i10 vdd pfet w=0.35u l=0.03u
M28 i12 DL  i13 gnd nfet w=0.18u l=0.03u
M29 i13 GN  gnd gnd nfet w=0.18u l=0.03u
M30 i13 B   gnd gnd nfet w=0.18u l=0.03u

*X6
M31 i15 i12 vdd vdd pfet w=0.1u l=0.03u
M32 i15 i12 gnd gnd nfet w=0.1u l=0.03u

*X7
M33 i16 GN  vdd vdd pfet w=0.1u l=0.03u
M34 i17 B   i16 vdd pfet w=0.1u l=0.03u
M35 i12 i15 i17 vdd pfet w=0.1u l=0.03u
M36 i12 i15 i18 gnd nfet w=0.1u l=0.03u
M37 i18 nGN i19 gnd nfet w=0.1u l=0.03u
M38 i19 nB  gnd gnd nfet w=0.1u l=0.03u

*X8
M39 L2 i12 vdd vdd pfet w=0.70u l=0.03u
M40 L2 i12 gnd gnd nfet w=0.36u l=0.03u
.ENDS
```

Figure D.3 – Spice code for the proposed Clocked-LSSD cell.

```
# Define delay arcs

# D => GN
define_arc −vector {RXXRXF} −pin D −related_pin GN −type setup −probe L2 CLOCKED_LSSD
define_arc −vector {FXXRXF} −pin D −related_pin GN −type setup −probe L2 CLOCKED_LSSD

# GN => L2
define_arc −vector {RFFFXF} −related_pin GN −pin L2 CLOCKED_LSSD
define_arc −vector {RXXRXR} −related_pin GN −pin L2 CLOCKED_LSSD

# D => L2
define_arc −vector {RFFFXF} −related_pin D −pin L2 CLOCKED_LSSD

# A => L2
define_arc −vector {FRRXRR} −related_pin A −pin L2 CLOCKED_LSSD
define_arc −vector {FRRXFF} −related_pin A −pin L2 CLOCKED_LSSD

# B => L2
define_arc −vector {FRRXRR} −related_pin B −pin L2 CLOCKED_LSSD
define_arc −vector {FRRXFF} −related_pin B −pin L2 CLOCKED_LSSD

# I => L2
define_arc −vector {FRFXRR} −related_pin I −pin L2 CLOCKED_LSSD
```

Figure D.4 – Proposed Clocked-LSSD liberty code sample.

APPENDIX E – DFT AUTOMATION SCRIPTS

This Section describes the scripts used to automate the DfT insertion in the Blade synthesis flow. Tool Command Language (TCL) scripts were generated to automate the DfT insertion and the ATPG tool. The DfT scripts configure the test protocol, define the test pins and insert the scan-chain. Design Compiler, DFT Compiler and TetraMAX ATPG tools by Synopsys were used to perform DfT insertion and validation.

First, are created the test signals necessary to perform the test using the LSSD cells. Figure E.1 shows the declaration of the system clocks (**clk**,**clkbar** and **clk3**), test clocks (**mclk** and **sclk**), input and output signals (**scan_in** and **scan_out**) and test enable signal (**scan_mode**). Also, the created signals are connected to the respective nets and pins.

```
set clk_1 "clk"
set clk_2 "clkbar"
set clk_3 "clk3"
set clk_a_name "mclk"
set clk_b_name "sclk"
set scan_in_name "scan_in"
set scan_out_name "scan_out"
set scan_mode_name "scan_mode"

create_port $clk_1 -dir in
create_port $clk_2 -dir in
create_port $clk_3 -dir in
create_port $clk_a_name -dir in
create_port $clk_b_name -dir in
create_port $scan_in_name -dir in
create_port $scan_out_name -dir out
create_port $scan_mode_name -dir in
create_net $clk_1
create_net $clk_2
create_net $clk_3
create_net $scan_mode_name

connect_net $clk_1 $clk_1
connect_net $clk_2 $clk_2
connect_net $clk_3 $clk_3
connect_net $scan_mode_name $scan_mode_name
connect_net $clk_2 cluster1_ctrl/clk_scan
connect_net $clk_1 cluster2_ctrl/clk_scan
connect_net $clk_3 cluster3_ctrl/clk_scan
```

Figure E.1 – DfT insertion signals declaration.

The next step is to declare the test protocol. Figure E.2 shows the configuration of the test protocol style (in this case, LSSD) and definition of the test interface. Each test signal has a function and characteristics as value and wave phase. The same process occurs to Q-Flop component and its respective signals, as shown in Figure E.3. To finish the test insertion process, the final netlist and the Standard Test Interface Language Procedure File (SPF) are generated. These files are necessary to perform the ATPG analyses. The netlist contains the gate level circuit description and the SPF contains the pins and their configuration.

```
# Set test protocol style
set_scan_configuration -style lssd
set_dft_configuration -fix_clock enable -fix_reset enable -fix_set enable

# Declare the Test interface signals
set rst_name "rst"

set_dft_signal -view spec -type MasterClock -port $clk_1
set_dft_signal -view spec -type MasterClock -port $clk_2
set_dft_signal -view spec -type MasterClock      -port $clk_3
set_dft_signal -view spec -type ScanMasterClock -port $clk_a_name
set_dft_signal -view spec -type ScanSlaveClock  -port $clk_b_name
set_dft_signal -view spec -type ScanDataIn -port $scan_in_name
set_dft_signal -view spec -type ScanDataOut -port $scan_out_name
set_dft_signal -view existing_dft -type MasterClock -port $clk_1 -timing {40 30}
set_dft_signal -view existing_dft -type SlaveClock -port $clk_2 -timing {70 60}
set_dft_signal -view existing_dft -type SlaveClock -port $clk_3 -timing {70 60}
set_dft_signal -view existing_dft -type ScanMasterClock -port $clk_a_name -timing {30 40}
set_dft_signal -view existing_dft -type ScanSlaveClock -port $clk_b_name -timing {60 70}
set_dft_signal -view existing_dft -type Reset -port $rst_name -active_state 0
set_dft_signal -view existing_dft -type Constant -port $scan_mode_name -active_state 1

# Create test protocol
create_test_protocol
```

Figure E.2 – DfT insertion test protocol declaration.

```
set clk_s_name "clk_samp"

create_port $clk_s_name -dir in
create_net $clk_s_name

connect_net $clk_s_name $clk_s_name
connect_net $clk_s_name cluster1_ctrl/sample_scan
connect_net $clk_s_name cluster2_ctrl/sample_scan
connect_net $clk_s_name cluster3_ctrl/sample_scan

set_dft_signal -view spec -type MasterClock -port $clk_s_name
            set_dft_signal -view existing_dft -type MasterClock
    -port $clk_s_name -timing {45 55}
```

Figure E.3 – DfT insertion using Q-Flop test setup insertion.

The stuck-at fault model ATPG script analyses the netlist generated in the synthesis to add faults, check for DfT violations, determine the test input vectors, perform the design simulation to validate the test patterns, and report the obtained fault coverage. The delay fault model ATPG script works similarly, but it reads an input file containing the paths that must be tested. Figure E.4 shows the netlist, the behavior description of the cells and the SPF loading. For stuck-at simulation, it is necessary to set the ATPG tool with the stuck-at fault model option and initialize the fault list, as shown in Figure E.5. As mentioned in Section B, the internal fault of the TD Verilog description must be removed of the fault list. Figure E.6 shows the commands used to remove these internal faults from fault list.

For delay path simulation, first, it is necessary to read a file with the paths of the circuit. This file was generated using the Synopsys PrimeTime static timing analysis tool. After the file generation, the process is the same that the stuck-at simulation, as shown in Figure E.7. After setting the ATPG parameters, the simulation is started and its possible to estimate the fault coverage for both fault models.

```
# Read design
read_netlist ../../../blade_${DESING}/bd_conversion/dft_outputs/${DESING}.v

# Read libs
read_netlist /soft64/design-kits/stm/28nm-cmos28fdsoi_24/C28SOI_SC_12_CORE_LR@2.0@20130411.0/
    behaviour/verilog/C28SOI_SC_12_CORE_LR.v -library
read_netlist /soft64/design-kits/stm/28nm-cmos28fdsoi_24/C28SOI_SC_12_CLK_LR@2.1@20130621.0/
    behaviour/verilog/C28SOI_SC_12_CLK_LR.v -library
read_netlist ../../../common/blade_lib.v -library
read_netlist ../../../common/lssd.v -library
read_netlist ../../../common/CLOCKED_LSSD.v -library
read_netlist ../../../common/METASTABILITY_FILTER.v -library

# Build atpg design
run_build_model ${DESING}

# Specify the stil file - define scan chains
set_drc ../../../blade_${DESING}/bd_conversion/dft_outputs/${DESING}.spf
run_drc
```

Figure E.4 – ATPG tool initialization commands.

```
#initialize fault list - add all possible faults int ATPG design
    model
set_faults -model stuck
add_faults -all
```

Figure E.5 – Specifying the stuck-at fault model simulation for the ATPG process.

```
remove_faults
    xtea_pipeline_i_dec_kernel_i_0_inner_round_stage_inst_data_flop_data_reg_48__m_Q_reg_TD/xor1/Z
remove_faults
    xtea_pipeline_i_dec_kernel_i_0_inner_round_stage_inst_data_flop_data_reg_48__m_Q_reg_TD/xor1/Z
remove_faults
    xtea_pipeline_i_dec_kernel_i_0_inner_round_stage_inst_data_flop_data_reg_49__m_Q_reg_TD/xor1/Z
remove_faults
    xtea_pipeline_i_dec_kernel_i_0_inner_round_stage_inst_data_flop_data_reg_50__m_Q_reg_TD/xor1/Z
remove_faults
    xtea_pipeline_i_dec_kernel_i_0_inner_round_stage_inst_data_flop_data_reg_48__m_Q_reg_TD/buf1/Z
remove_faults
    xtea_pipeline_i_dec_kernel_i_0_inner_round_stage_inst_data_flop_data_reg_48__m_Q_reg_TD/buf1/A
remove_faults
    xtea_pipeline_i_dec_kernel_i_0_inner_round_stage_inst_data_flop_data_reg_48__m_Q_reg_TD/xor1/A
remove_faults
    xtea_pipeline_i_dec_kernel_i_0_inner_round_stage_inst_data_flop_data_reg_48__m_Q_reg_TD/buf1/Z
remove_faults
    xtea_pipeline_i_dec_kernel_i_0_inner_round_stage_inst_data_flop_data_reg_48__m_Q_reg_TD/buf1/A
remove_faults
    xtea_pipeline_i_dec_kernel_i_0_inner_round_stage_inst_data_flop_data_reg_48__m_Q_reg_TD/xor1/A
remove_faults
    xtea_pipeline_i_dec_kernel_i_0_inner_round_stage_inst_data_flop_data_reg_48__m_Q_reg_TD/xor1/B
...
```

Figure E.6 – ATPG commands to remove the internal faults of the Transition Detector cell.

```
# Read paths
add_delay_paths "./timing200withoutmob.rpt"

# Initialize fault list - add all possible faults int ATPG design
    model
set_faults -model path_delay
add_faults -all
```

Figure E.7 – Specifying the path-delay fault model simulation for the ATPG process.