PUCRS

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

FELIPE AUGUSTO KUENTZER

MORE THAN A TIMING RESILIENT TEMPLATE: A CASE STUDY ON RELIABILITY-ORIENTED IMPROVEMENTS ON BLADE

Porto Alegre

2018

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
FACULTY OF INFORMATICS
COMPUTER SCIENCE GRADUATE PROGRAM

MORE THAN A TIMING RESILIENT TEMPLATE: A CASE STUDY ON RELIABILITY-ORIENTED IMPROVEMENTS ON BLADE

FELIPE AUGUSTO KUENTZER

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

Advisor: Prof. Alexandre de Morais Amory

Porto Alegre
2018

Ficha Catalográfica

K95m Kuentzer, Felipe Augusto

More than a timing resilient template : a case study on reliability-oriented improvements on Blade / Felipe Augusto Kuentzer . – 2018.

100 f.

Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Alexandre de Morais Amory.

1. Timing resilient design. 2. Asynchronous design. 3. Design for testability. 4. Functional testing. 5. Blade. I. Amory, Alexandre de Morais. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS com os dados fornecidos pelo(a) autor(a).

Felipe Augusto Kuentzer

More than a Timing Resilient Template: A Case Study on Reliability-Oriented Improvements on Blade

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 28th, 2018.

COMMITTEE MEMBERS:

Prof. Dr. Peter Anthony Beerel (USC)

Prof. Dr. Marcelo Soares Lubaszewski (PGMicro/UFRGS)

Prof. Dr. Ney Laert Vilar Calazans (PPGCC/PUCRS)

Prof. Dr. Alexandre de Morais Amory (PPGCC/PUCRS - Advisor)

MAIS DO QUE UM CIRCUITO RESILIENTE A VARIAÇÕES DE ATRASO: UM ESTUDO DE CASO SOBRE MELHORIAS ORIENTADAS À CONFIABILIDADE NO BLADE

RESUMO

À medida que o projeto de VLSI avança para tecnologias ultra submicron, as margens de atraso adicionadas para compensar variabilidades de processo de fabricação, temperatura de operação e tensão de alimentação, tornam-se uma parte significativa do período de relógio em circuitos síncronos tradicionais. As arquiteturas resilientes a variações de atraso surgiram como uma solução promissora para aliviar essas margens de tempo projetadas para o pior caso, melhorando o desempenho do sistema e reduzindo o consumo de energia. Essas arquiteturas incorporam circuitos adicionais para detecção e recuperação de violações de atraso que podem surgir ao projetar o circuito com margens de tempo menores. Os sistemas assíncronos apresentam potencial para melhorar a eficiência energética e o desempenho devido à ausência de um sinal de relógio global. Além disso, os circuitos assíncronos são conhecidos por serem robustos a variações de processo, tensão e temperatura. Blade é um modelo que incorpora as vantagens de projeto assíncrono e resilientes a variações de atraso. No entanto, o Blade ainda apresenta desafios em relação à sua testabilidade, o que dificulta sua aplicação comercial ou em larga escala. Embora o projeto visando testabilidade com *Scan* seja amplamente utilizado na indústria, os altos custos de silício associados com o seu uso no Blade podem ser proibitivos. Por outro lado, os circuitos assíncronos podem apresentar vantagens para testes funcionais, enquanto o circuito resiliente fornece *feedback* contínuo durante o funcionamento normal do circuito, uma característica que pode ser aplicada para testes concorrentes. Nesta Tese, a testabilidade do Blade é avaliada sob uma perspectiva diferente, onde o circuito implementado com o Blade apresenta propriedades de confiabilidade que podem ser exploradas para testes. Inicialmente, um método de classificação de falhas que relaciona padrões comportamentais com falhas estruturais dentro da lógica de detecção de erro e uma nova implementação orientada para teste desse módulo de detecção são propostos. A parte de controle é analisada para falhas internas, e um novo projeto é proposto, onde o teste é melhorado e o circuito pode ser otimizado pelo fluxo de projeto. Um método original de medição de tempo das linhas de atraso também é abordado.

Finalmente, o teste de falhas de atrasos em caminhos críticos do caminho de dados é explorado como uma consequência natural de um circuito implementado com Blade, onde o monitoramento contínuo para detecção de violações de atraso fornece a informação necessária para a detecção concorrente de violações que extrapolam a capacidade de recuperação do circuito resiliente. A integração de todas as contribuições fornece uma cobertura de falha satisfatória para um custo de área que, para os circuitos avaliados nesta Tese, pode variar de 4,24% a 6,87%, enquanto que a abordagem *Scan* para os mesmos circuitos apresenta custo que varia de 50,19% a 112,70% em área, respectivamente. As contribuições desta Tese demonstraram que, com algumas melhorias na arquitetura do Blade, é possível expandir sua confiabilidade para além de um sistema de tolerância a violações de atraso no caminho de dados, e também um avanço para teste de falhas (inclusive falhas online) de todo o circuito, bem como melhorar seu rendimento, e lidar com questões de envelhecimento.

Palavras-Chave: projeto resiliente a variações de atraso, projeto assíncrono, projeto visando testabilidade, teste funcional, falhas de stuck-at, falhas de atraso, blade.

MORE THAN A TIMING RESILIENT TEMPLATE: A CASE STUDY ON RELIABILITY-ORIENTED IMPROVEMENTS ON BLADE

ABSTRACT

As the VLSI design moves into ultra-deep-submicron technologies, timing margins added due to variabilities in the manufacturing process, operation temperature and supply voltage become a significant part of the clock period in traditional synchronous circuits. Timing resilient architectures emerged as a promising solution to alleviate these worst-case timing margins, improving system performance and/or reducing energy consumption. These architectures embed additional circuits for detecting and recovering from timing violations that may arise after designing the circuit with reduced time margins. Asynchronous systems, on the other hand, have a potential to improve energy efficiency and performance due to the absence of a global clock. Moreover, asynchronous circuits are known to be robust to process, voltage and temperature variations. Blade is an asynchronous timing resilient template that leverages the advantages of both asynchronous and timing resilient techniques. However, Blade still presents challenges regarding its testability, which hinders its commercial or large-scale application. Although the design for testability with scan chains is widely applied in the industry, the high silicon costs associated with its use in Blade can be prohibitive. Asynchronous circuits can also present advantages for functional testing, and the timing resilient characteristic provides continuous feedback during normal circuit operation, which can be applied for concurrent testing. In this Thesis, Blade's testability is evaluated from a different perspective, where circuits implemented with Blade present reliability properties that can be explored for stuck-at and delay faults testing. Initially, a fault classification method that relates behavioral patterns with structural faults inside the error detection logic and a new test-driven implementation of this detection module are proposed. The control part is analyzed for internal faults, and a new design is proposed, where the test coverage is improved and the circuit can be further optimized by the design flow. An original method for time measuring delay lines is also addressed. Finally, delay fault testing of critical paths in the data path is explored as a natural consequence of a Blade circuit, where the continuous monitoring for detecting timing violations provide the necessary feedback for online detection of these delay faults. The integration of all the contributions provides a satisfactory fault coverage for an area overhead that, for the evaluated circuits in this thesis, can vary from 4.24% to 6.87%, while

the scan approach for the same circuits implies an area overhead varying from 50.19% to 112.70%, respectively. The contributions of this Thesis demonstrated that with a few improvements in the Blade architecture it is possible to expand its reliability beyond a timing resilient system to delay violations in the data path, but also advances for fault testing (including online faults) of the entire circuit, yield, and aging.

Keywords: timing resilient design, asynchronous design, design for testability, functional testing, stuck-at faults, delay faults, blade.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ATPG – Automatic Test Pattern Generation

BIST – Built-in Self-test

DC – Detection Clock

DFT – Design for Testability

DIB – Delayed-Input-Based

DSTB – Double Sampling With Time Borrowing

DVS – Dynamic Voltage Scaling

EDA – Electronic Design Automation

EDL – Error Detection Logic

EDS – Error Detecting Sequential

ERR_NST – Errors in the Next Stage

ERR_ST – Errors in the Faulty Stage

GALS – Globally Asynchronous Locally Synchronous

ITRS – International Technology Roadmap for Semiconductors

LPD – Longer Propagation Delay

LSSD – Level Sensitive Scan Design

MSFF – Master-Slave Flip-Flop

PH – Pipeline Halted

PLA – Programmable Logic Array

PST – Pipeline Output Stuck at a Value

PVT – Process, Voltage and Temperature

RTL – Register-Transfer Level

RTZ – Return-to-Zero

SDF – Standard Delay Format

SLB – Shadow-Latched-Based

SPD – Shorter Propagation Delay

SQF – Scan Q-Flop

SSI – Small Scale Integration

ST0 – Stuck-at-0

ST1 – Stuck-at-1

TD – Transition Detector

TDC – Time to Digital Converter

TDTB – Transition Detector With Time Borrowing

TEDL – Testable Error Detection Logic

TEDL – Testable Error Detection Logic

TRW – Timing Resiliency Window

TV – Timing Violation

UN – Undetectable

VLSI – Very Large Scale Integration

CONTENTS

1. INTRODUCTION

Energy efficiency has become one of the most constant and important demands for contemporary applications, such as for mobile devices, high-performance data centers, and embedded systems. As VLSI design technologies approach more and more in the domain of low power, timing margins become a significant factor when designing traditional synchronous circuits with optimal clock period. These circuits must incorporate timing margins to ensure correct operation under worst-case conditions. The designer must account for timing variability due to the manufacturing process, circuit aging and circuit operation under a wide range of temperatures. Also, voltage supply must also be conservative, even though the chip has the potential to operate at lower voltages, which could help to reduce energy consumption. These timing uncertainties increase the delay margins that must be incorporated to the clock period, which limits the performance gains and increases the power consumption.

According to the 2011 ITRS (International Technology Roadmap for Semiconductors) [ITR11], one of the main challenges in the development of modern integrated circuits is the distribution of a single clock signal for controlling the entire circuit. The asynchronous design style removes the need of a global clock signal, thus eliminating problems related to clock skew and clock distribution. Instead, synchronization occurs using a handshake protocol between circuit elements. Asynchronous circuits are often implemented with one of two designs styles, quasi-delay-insensitive (QDI) and bundled-data (BD). The first one embeds a completion signal in the data representation, which makes the design robust to PVT variations [BEE07], but they are often larger (4x) if compared to traditional synchronous implementations. The second one presents lower area than QDI, but similar to synchronous circuits, they must also be implemented with sufficiently large margins to account for PVT variations. Regarding these timing margins in BD designs, different solution were proposed, such as duplicating the BD delay lines [CHA10] and constraining the design to regular structures such as programmable logic array (PLA) [JAY06].

Timing resilient architectures also emerged as a promising solution to alleviate these process, voltage and temperature (PVT) timing margins in synchronous designs [ERN03] [DAS09] [FOJ13] [KWO14] and in asynchronous BD designs [HAN15] by allowing timing violations. However, these architectures need additional circuitry to detect and recover from these timing violations, leading to high area and recovery penalties. Moreover, some of these architectures are susceptible to metastability problems [BEE14] that can prevent their use.

The Blade template [HAN15] is an alternative that combines the advantages of the asynchronous BD designs and timing resilient techniques to alleviate timing margins and address the metastability issues of previous works. Blade uses two reconfigurable delay lines and error detection logic to detect timing violations coupled to a novel speculative handshake protocol that improves average-case performance. A Blade circuit can achieve as much as 30% power reduction at the same performance, when compared to a similar synchronous circuit, for an area overhead of about 10%. Despite these promising results and increasing evolution of timing resilient architectures,

Blade's practical usage (as other timing resilient circuits) is still hindered by the challenges regarding testability. In general, the Design for Testability (DfT) is a well established research area. Even for asynchronous circuits a lot has been done in the last decades [PAG92] [KHO94] [PET95b] [RON96a] [PET97] [KAN99] [BER02] [BEE03] [SHI05] [GIL06] [RON15].

In classic design style (synchronous non-timing resilient) the most common DfT approach is to apply scan chains to add observability and controllability, and the internal states of the circuit can be initialized by test patterns and the resulting computation of these patterns extracted for analysis. It is further explained later that the fault models (e.g., stuck-at and delay) for synchronous and asynchronous circuits are the same. The difference is how fault effects express themselves. Another difference that affects the testability of asynchronous circuits is the absence of a global clock, which provides a lock-stepped computation that eases the test control on synchronous circuits. Some works address this problem by adding a synchronous test mode to the asynchronous implementation [BEE03], so that a dedicated clock controls the sequential elements only during the test. In this scenario, traditional static timing analysis is performed to identify critical paths in the circuit, and scan-based structural approach with automatic test pattern generator (ATPG) is applied for testing these paths. However, with increasing delay variations in modern technologies, static timing analysis becomes inaccurate [KRS98], even for the classic synchronous design approach. With timing resilient circuits, testing can become even more complicated, since now the timing analysis must not only account for the PVT delay variabilities but it must also consider that a given path can produce timing violations that are only observable and handled during at-speed functional operation.

Delay fault testing of timing resilient circuits is fundamentally different from testing conventional circuits because the former tolerates some timing violations, and not all timing violations become a fault. The work of Yuan [YUA13] is one of the first to address the testing challenges of timing resilient architectures. Yuan proposed a scan-based approach for testing a synchronous timing resilient architecture, where the error detection logic is reused as a fault detection mechanism for path delay faults. To accomplish this task a clock divider and a duty-cycle controller are responsible for creating specific clock configurations that allow delay fault detection through the detection mechanism.

Although the work of Yuan presents a possible solution for testing Blade through a dedicated synchronous mode implementation, the proposed scan design is custom made for the analyzed architecture, and also rely on an additional clock divider and a duty-cycle controller circuitry, which leads to another critical aspect of testing timing resilient architectures. As already mentioned, detection mechanisms in resilient circuits incur area overheads. Thus, the area for test circuitry is constrained. For example, when comparing Blade [HAN15] and Bubble Razor [FOJ13] to a classical synchronous design, the overall area overhead for the Blade implementation is 8.4%, and 21% increase in combinational logic and 280% in the sequential area for the Bubble Razor implementation. The area overhead for DfT on Bubble Razor can be prohibitive, mainly due to the significant increase in sequential area which leads to longer scan chains.

In this sense, the usage of non-standard sequential components in asynchronous circuits hinders the scan-based approach because it incurs in high area penalties. On the other hand, asynchronous designs have some advantages over synchronous regarding its testability. For instance, a stuck-at fault in the asynchronous controller can halt the entire circuit, making such fault detectable with a functional test [PAG92]. Furthermore, timing resilient circuits provide constant time checking, which can be explored for concurrent testing during functional operation.

1.1 Objectives

The Thesis general objective is to demonstrate that the Blade template, with few proposed improvements, is more than an asynchronous timing resilient template. It is a possible DfT approach for asynchronous BD design style, that can cover stuck-at and path delay faults in Blade's specific modules through different testing phases, from online testing to structural testing. The proposed implementation present lower area overheads when compared to classic scan approach, and increased yield and aging properties when compared to synchronous timing resilient and non-timing resilient designs. To reach this goal, the following specific objectives were addressed during this Thesis:

- **Implement a test environment to simulate and analyze the behavior of stuck-at and path delay faults on Blade:** This includes the description of a high-level simulation environment to experiment and validate different approaches for testing Blade.
- **Propose a test method for detecting faults in Blade:** A circuit implemented with Blade is divided into four blocks: (i) controller; (ii) error detection logic; (iii) delay lines; (iv) data path. Each block is individually analyzed for its testability, and a method for detecting stuck-at and/or path-delay faults is presented;
- **Propose alternative implementations to improve fault coverage and area overhead in Blade:** A design implementation aiming testability to improve fault coverage and area overhead for the proposed test method. Thus alternative designs of the controller and error detection logic are considered;
- **Modify the Blade design flow to allow automatic insertion of the proposed test methods:** One goal of this work is to modify Blade's original synthesis flow to support automated insertion of the proposed test methods;
- **Evaluate area overhead and fault coverage of the proposed implementations:** The area overhead and fault coverage of each block is compared to the original implementation.

1.2 Contributions

The main contributions of the Thesis are:

1. **A fault classification method for Blade:** A fault classification method and the fault analysis of Blade's error detection logic was proposed and published in [KUE16] (Section 4.3). The presence of unconventional sequential cells inside the error detection logic prevents the use of standard ATPG. In this case, the fault classification relates behavioral patterns generated by the detection logic to existing faults;

2. **A testable error detection logic for Blade:** Based on the fault analysis of the original error detection logic, a testable architecture of this block was proposed (Section 4.4). This approach removes the necessity of making the data path scannable. Instead of standard ATPG, faults are detected through the proposed fault classification method;

3. **A Click-based controller for Blade:** A controller with a single internal delay line implementation using Click [PEE10] was proposed. (Section 5.2). The Click-based design overcome a limitation of the original Blade flow where the original Burst-Mode controller could not be further optimized by the synthesys tool. Moreover, it presents higher fault coverage with functional testing and lower overall area than the original controller;

4. **A method for testing delay lines with Blade:** An offline method for testing the Blade delay lines according to the design specification and constraints (Section 5.3). Timing resilient circuits account for process variabilities in the data path, but not for process variability in Blade's delay lines. Thus a method for properly measuring the path delay of the delay elements is proposed;

5. **A new method for testing path delay faults in critical paths:** A concurrent test method for detecting path delay faults in the critical path was proposed (Chapter 6). Path delay faults are detected by transforming the error detection logic into a fault detection mechanism. The proposed method presents minimal impact in the area while expanding Blade's usage to yield and aging improvements.

The following papers were derived from this Thesis:

- As first author:
  - On the Reuse of Timing Resilient Architecture for Testing Path Delay Faults in Critical Paths, F. A. Kuentzer, L. R. Juracy, and A. M. Amory: accepted for publication in DATE 2018 (Qualis A1);
  - Fault Classification of the Error Detection Logic in the Blade Resilient Template, F. A. Kuentzer, L. R. Juracy, and A. M. Amory: published on IEEE ASYNC 2016 (Qualis B2)
  - Testable Error Detection Logic Design Applied to an Asynchronous Timing Resilient Template: submitted to IEEE ISCAS 2018 (Qualis A1);
- As co-author:

- Optimized Design of an LSSD Scan Cell, L. R. Juracy, M. T. Moreira, F. A. Kuentzer, and A. M. Amory: published on IEEE Transaction on VLSI 2017 (Qualis A1);
- Testable Q-Flop: A Scannable Metastability-free Memory Element, L. R. Juracy, M. T. Moreira, F. A. Kuentzer, and A. M. Amory: accepted for publication in IEEE Transaction on VLSI 2018 (Qualis A1);
- An LSSD Compliant Scan Cell for Flip-Flops, L. R. Juracy, M. T. Moreira, F. A. Kuentzer, and A. M. Amory: published on IEEE ISCAS 2018 (Qualis A1);

1.3 Document Structure

The rest of the document is organized as follows. Chapter 2 provides relevant background information. Chapter 3 presents state of the art in timing resilient circuits and their testability. This chapter also summarizes the main works in asynchronous BD design testing. Chapter 4 presents the fault analysis and fault classification for Blade's error detection logic along with the proposed testable design. Chapter 5 evaluates the functional testing coverage for the original controller and presents an alternative design that achieves higher coverage. Also, a method for testing the delay lines is proposed. Next, Chapter 6 explores the reuse of Blade's error detection mechanism for detecting path delay faults in the critical paths. Chapter 7 summarizes the different test methods proposed in this Thesis and their applications. All proposed methods were integrated into two case studies to evaluate area overhead and compare these with the classic scan approach. Finally, Chapter 8 presents the final considerations of this work and discusses future works.

2. CONCEPTS

This Chapter presents some basic concepts on DfT, an overview of two common fault models, asynchronous circuits and some relevant BD templates, as well as timing resilient architectures.

2.1 Design for Testability

As designs moved from small scale integration (SSI) to VLSI the complexity to test such circuits became harder. A standard approach to test these VLSI circuits during the 1980s relied mainly on fault simulation of functional patterns to extract a fault coverage. These patterns were developed to exercise the internal states and detect manufacturing defects. Several functional patterns were applied to increase fault coverage. Unfortunately, this approach typically reached a fault coverage around 80%. The design quality dropped, and the costs increased, which led to the development and deployment of Design for Testability (DfT).

The first challenge was to find simpler ways of exercising all internal states of design and increase fault coverage. Initially, ad hoc techniques were proposed to aid the testability and improve the controllability and observability of the circuit. These techniques relied on making local modifications, such as insert test points, avoid combinational feedback loops, avoid asynchronous logic and partition the circuit into small blocks. Even though these methods have substantially improved the testability of a design, it was difficult to reach more than 90% of fault coverage for large designs. Their effects are not systematic, and they have to be custom made for each new project, often with unpredictable results.

An alternative approach for controlling and observing the internal states of sequential circuits is the structured DfT, which allows direct external access to internal storage elements. These storage elements with direct external access are called scan cells. This approach allows full controllability and observability of internal states of the design and reduces the problem of testing the sequential circuit to testing the combinational logic, for which many solutions and tools for automatic test pattern generation (ATPG) already existed.

Scan design is probably the most popular structured DfT approach. Scan designs have the storage elements replaced by scan cells to gain external access. These cells are interconnected to additional scan input ports and shared/additional scan output ports, forming one or more shift registers, called scan chains.

Numerous scan cell designs and scan architectures have been developed. The full-scan architecture consists in replacing all memory elements by scan cells, and combinational ATPG is used for test generation. In partial-scan architecture, only a subset of memory elements is replaced by scan cells and connected into the scan chain, but sequential ATPG is typically used for test generation, increasing the test complexity. The designer's decision will depend on project constraints

regarding area and performance overheads, test complexity and the fault coverage goal. As for the scan technique, the edge-triggered mux-D scan [BUS02] and the level sensitive scan design (LSSD) [EIC77] [JUR17] are best known and most widely practiced for synchronous circuits cells types.

Scan design has also become the basis of more advanced DfT techniques, such as logic built-in self-test (BIST). In such architecture, circuits that generate test patterns and analyze the output responses of the functional circuitry are embedded in the chip or elsewhere on the same board where the chip resides. These techniques emerged as an alternative to reduce the high test expenses of traditional test techniques that use ATPG tools, and to alleviate some test problems that appeared as the semiconductors manufacturing technologies advanced, for instance, the test of assembled boards, system test, periodic maintenance and repair test [WAN06a].

Despite the extensive use of the scan-based approach for testing sequential designs in at speed-tests, functional testing is an alternative way of testing a circuit, that consists in identifying the functions that the circuit is expected to perform, create input data patterns based on these function specifications and determine the expected output pattern for each input. As presented in [KRS03], functional testing may translate into delay fault under-testing, where non-functional delay paths are not covered. On the other hand, structural testing can over-test paths that are not functionally activated, resulting in yield loss. Software-based functional testing can be used to improve the fault coverage of processors [LAI00] [KRA05] [SIN06]. Functional test can also be applied in-field to check the reliability of a device [TEH11].

2.2 Overview on Stuck-at and Delay Fault Models

Fault models are a computational abstraction used to represent defects [MOU00]. In the context of manufacturing defects, the stuck-at fault model is one of the most common to detect defects in integrated circuits. The fault is modeled by assigning a fixed logic value (0 or 1) to a circuit wire. A line consists of an input or output of a logic gate, flip-flop or latch. The most popular is the single stuck-at-1 and single stuck-at-0. For example, a logic OR gate of two inputs with stuck-at-0 at the output will never assume the logical value 1, even in the presence of logic value 1 at both its inputs.

Another type of manufacturing fault is a speed defect, which can be modeled as a delay fault. A circuit has a delay fault when its output fails to reach the correct value within a predefined time constraint. A signal transition is required to observe a delay defect, which involves the application of vector pairs that affect the design delay significantly. One of these vectors is responsible for initializing the circuit under test to a known state. The other is responsible for stimulating the circuit so that, if the delay of a location or path exceeds the timing constraints of the project, invalid values are captured by sequential cells or primary outputs, deviating from the expected values, which characterizes the fault.

Many delay fault models and test techniques have been proposed and used in digital circuits [TEH08] [RED09] [BUS02] [JHA03] [CRO99]. The three basic models for delay faults are transition fault model, gate-delay fault model, and path delay fault model [KRS98]. The transition fault model [LEV86] [CHE93] assumes that the defect affects a single gate in the circuit, and each gate can be associated with a slow-to-rise and a slow-to-fall transition fault. To detect a fault, the transition model assumes that the extra delay prevents a transition from reaching a primary output at the observation time. The advantage of the transition fault model is that the number of faults is linear to the number of gates in the circuit, but expect that the delay fault is large enough to be observed might not be realistic.

The gate delay fault model [CAR87] [PRA97] also assumes that the delay fault affects only one gate, but unlike the transition model, the gate delay fault model implies that an increase in delay might affect the performance. Since it takes into account the circuit delays, it is related to a quantitative model and the transition model to a qualitative one. The limitations of the gate delay model are similar, due to the single gate fault assumption, that may fail to detect delay faults that are a sum of small delay defects. On the other hand, the path delay fault model [SMI85] can detect small distributed delay defects. The delay of a path represents the sum of the gate delays and interconnections on the path. A delay defect on a path can be observed by applying a transition at the beginning of the path and observing its effect during the propagation through this path. A limitation of this fault model is that the number of paths in the circuit can be very large, most often exponential on the number of gates. For this reason, not all paths are tested. Usually, the longest path containing a given signal or paths where the expected delay is greater than a specified threshold are selected.

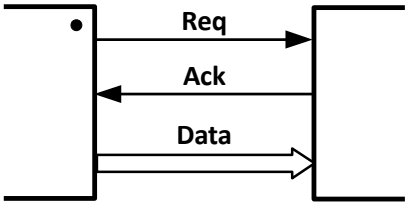
2.3 Asynchronous Circuits

Most currently fabricated digital circuits follow the paradigm of synchronous clocked design. In synchronous circuits, all components share a common discrete notion of time defined by a global clock signal that significantly simplifies their design. On the other hand, asynchronous circuits have no common notion of a discrete time. To perform synchronization, communication, and sequencing of operations, asynchronous circuits use local handshake between communicating components. In synchronous terms, it would be like having a circuit where registers are only clocked where and when needed.

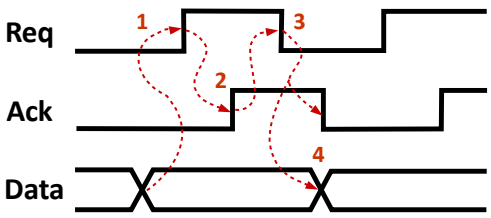
Asynchronous designs have showed advantages in different aspects compared to synchronous designs, like: (i) low power consumption, due to power consumption in standby mode; (ii) high operation speed, where speed is determined by local latencies instead of global worst-case delay (the clock period); (iii) less emission of electromagnetic noise; (iv) robustness towards process, voltage, and temperature variations; (v) no clock distribution and skew [SPA01].

The choice of a handshake protocol and a data encoding scheme is called a template [BEE10]. Currently, there are two main design styles of asynchronous templates, QDI and BD [BEE10]. The first relies on multi-rail data encoding, where the completion signal is embedded into the data representation. For instance, the dual-rail protocol uses two wires to represent one bit of data and the corresponding request signal. Even though they provide relaxed timing constraints, the circuits from this family are usually larger and have higher power consumption than their synchronous counterparts. Since BD is the design style used in the asynchronous timing resilient template that will be further treated here, the discussion focus on BD templates only.

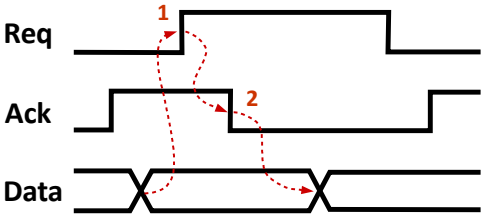
BD templates use standard Boolean encoding to represent information, and separate, request and acknowledge wires, are bundled with data signals to provide synchronization, see Figure 2.1(a). BD designs can employ 2-phase or 4-phase handshake protocols. The 4-phase term refers to the number of communication actions. Figure 2.1(b) illustrates the waveform for a possible 4-phase protocol: (1) the sender issues data and sets *Req* high, (2) the receiver stores the data and sets *Ack* high, (3) the sender responds by taking *Req* low (at this point data is no longer guaranteed to be valid) and (4) the receiver acknowledges by taking *Ack* low. At this moment the sender may initiate a new communication cycle.



(a) Bundled-data channel



(b) 4-phase protocol



(c) 2-phase protocol

Figure 2.1 – (a) Bundled-data block diagram. (b) The 4-phase handshake protocol. (c) The 2-phase handshake protocol.

The 4-phase protocol has the disadvantage of requiring return-to-zero (RTZ) transitions that cost unnecessary time and energy, and potentially reduce the performance of the circuit. The 2-phase bundled-data protocol shown in Figure 2.1(c) avoids the RTZ. The request and acknowledge are encoded as signal transitions. There is no consensus about the protocol type, while the 2-phase improves performance, the 4-phase requires less complex control circuitry.

There are different design styles for implementing BD control blocks. The main available templates in the literature use 2-phase handshake protocols. The concept of Micropipelines, introduced in [SUT89], is implemented with particular capture-pass latches that are event-controlled,

being these capable of sensing transitions at its inputs. Another approach is Mousetrap [SIN07], which implements 2-phase BD circuits using level-sensitive latches and XOR gates.

A more recent template is Click [PEE10]. Instead of using latches and C-elements, that are common in many asynchronous designs, the authors proposed a 2-phase BD template that only uses edge-triggered flip-flops in both data path and control path. This approach facilitates the asynchronous circuit design flow using standard (third-party) EDA tools, especially for physical synthesis and static timing analysis.

Another template is a 2-phase BD that, different from the previous ones, does not avoid RTZ. In the GasP [SUT01] template, instead of separate request and acknowledgment wires, a single wire represents the full or empty states. With this template, in the first phase the sender rises the state wire when valid data is available, and in the second phase, the receiver lowers the same state wire, indicating that the data was consumed.

2.4 Timing Resilient Architectures

Scalability in circuit manufacturing processes allowed performance improvements and increased energy efficiency, but it also led to undesirable side effects, such as PVT variations, and reliability issues due to transistor degradation, which can cause premature failure of the system [TSC09]. Circuit designers add delay margins to compensate for PVT variations and achieve good yield, thus affecting system performance. An alternative for removing some of the increasingly large delay margins is the timing resilient design technique.

Different research areas use the resilient term. In physics, for instance, resiliency is a property of materials that accumulate energy when subjected to stressful situations, such as ruptures. In other words, resiliency is the ability to return something to its natural state, especially after some critical or unusual condition. Thus, a circuit is said to be resilient when it can recover its normal operating status, even after an internal error or caused by external factors.

Different timing resilient architectures have been proposed until now [ERN03] [DAS09] [FOJ13] [KWO14] [HAN15]. These techniques allow the circuit to operate with relaxed timing constraints that eventually cause a timing violation. For such cases, these techniques rely on some error detection logic and on an error recovery mechanism.

For instance, the Razor [ERN03] family was one of the first timing resilient architectures for synchronous circuits to be proposed. This synchronous approach consists in replacing flip-flops (FF) on critical paths of the circuit by special ones called Razor Flip-flops. Figure 2.2 presents the basic diagram of this implementation. The *clk* signal is designed with less pessimistic time margins, and it controls the regular FF (Main Flip-flop) data sampling, while the Shadow Latch data sampling is controlled by a delayed clock (*clk_del*) that meets the worst-case timing constraints that guarantee valid data at the output of the latch. The comparison between the values of the Main Flip-flop and the Shadows Latch flags a timing violation. The error signal causes the circuit to redirect the valid

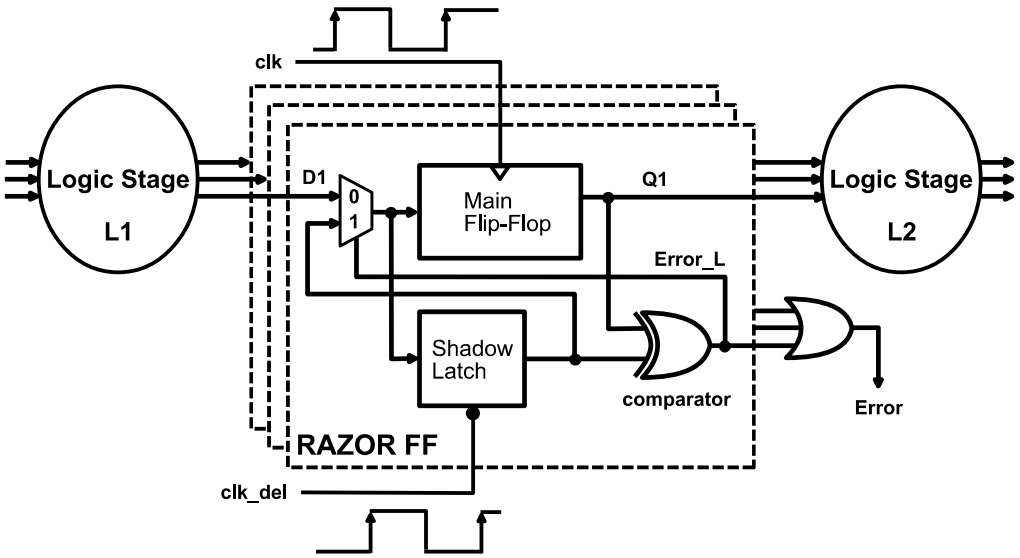


Figure 2.2 – Razor flip-flop [ERN03].

data stored at the latch to be sampled by the FF with a one cycle penalty. The previous stage must stall, and data at the following stages must be flushed.

Timing resilient circuits benefit from average-case performance. They rely on the fact that errors have a low probability of occurrence and the recover penalties have a small effect on performance. At this point, the error rate becomes a vital parameter of this type of design. In this sense, it becomes a challenge to the designer to dominate and solve the trade-off between allowed error rate and the performance, power, and reliability of the circuit.

3. STATE OF THE ART

This Chapter presents an overview of current timing resilient architectures found in the literature and their testability. The works are divided into four categories: Timing resilient architectures, covering the Razor family and its successive developments and the synchronous Blade template and other relevant timing resilient circuits; Testability of synchronous timing resilient circuits; Testability of asynchronous BD circuits since there are no previous works on testing asynchronous timing resilient circuits. The last section presents conclusions about these topics.

3.1 Timing Resilient Templates

3.1.1 Razor

The Razor architecture [ERN03] emerged as an alternative for eliminating worst-case safety margins by using a novel voltage management technique, where the processor operates with dynamic voltage scaling (DVS). As mentioned in Section 2.4, in this technique, Razor FFs replace FFs in critical paths of the circuit. The circuit then can have the supply voltage scaled down to the point of the first failure for a given frequency, eliminating all margins due to global and local PVT variation, thus resulting in energy savings. The supply voltage can be scaled even lower than the first failure point to achieve additional energy savings. In this case, a targeted error rate is deliberately tolerated, although this must be carefully considered, since the correction cost is 18 times more expensive regarding energy than regular operation, as in a 64-bit Kogge-Stone adder.

Based on a set of simulated benchmark experiments, an error rate of 1.5% allows average energy savings of 41% with a maximum performance slowdown of 6%. A 64-bit Alpha processor using the Razor technique was manufactured in a 0.18 µm technology. The circuit operates at 200MHz, and the clock for the shadow latch was delayed by $1/2$ the clock cycle from the system clock. Results also show the error detection and correction circuitry represent a 3.1% total power overhead in an error-free configuration.

3.1.2 Razor II

Razor II [DAS09] is a simplification of the original Razor FF. The Razor II FF removes the error recovery mechanism from the original proposal and error recovery takes place through architectural replay. The authors based their new approach on the fact that the error rate at the point of the first failure is in the order of 1 error in 10 million cycles, which makes recovery energy negligible. Substantially below the point of the first failure, the energy saving was small ($\approx$10%) compared to the energy gain from eliminating the PVT margins ($\approx$ 35% to 45%).

The Razor II FF uses a positive level-sensitive latch instead of a master-slave FF. It also combines a transition detector (TD) controlled by a detection clock (DC) as presented in Figure 3.1. Valid data is captured if data stabilizes before the rising edge of the clock. If data changes during the high-phase of the clock, while the latch is transparent, an error signal is flagged. To prevent valid data transition from being flagged as a timing violation, a short negative pulse on DC is used to disable the transition detector for a small period. It is equivalent to the propagation delay from D to Q after the rising edge of the clock. When a timing violation occurs, the whole pipeline is flushed, and the failing instruction is re-executed. If successive failures occur, the clock frequency is reduced by half during 8 cycles.

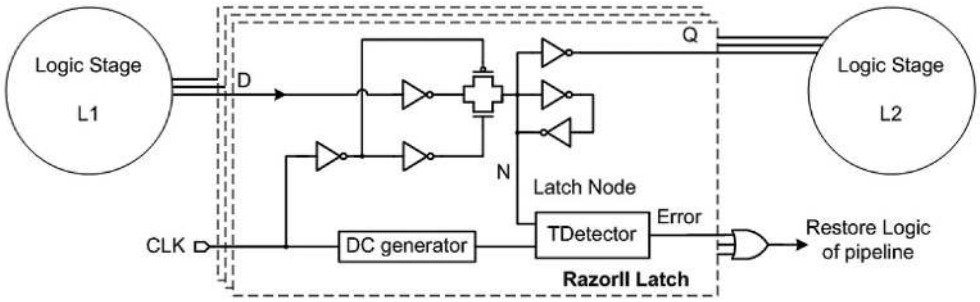


Figure 3.1 – Razor II flip-flop [DAS09].

As a case study, a 64-bit 7-stage Alpha processor featuring Razor II was manufactured in 0.13 µm technology. Results show that with an error rate of 0.04%, energy savings reach up to 37.4%, if compared to the worst-case when the supply voltage is set to ensure correct operation. The total overhead due to the presence of the Razor II circuitry in error-free operation is up to 1.2% of the total power.

3.1.3 Razor Lite

Razor-Lite [KWO14] uses a side-channel detection strategy compatible with standard D FF to reduce additional error detection circuitry. This approach is intended to well-balanced pipeline implementations, which present a large number of critical paths. Thus the number of FFs that must detect timing violations is high.

Figure 3.2 illustrates a conventional FF design and the added logic required by Razor-Lite. The additional circuit connected to the virtual VDD (VVDD) and virtual VSS (VVSS) rails of the FF 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 flagged. A timing violation is flagged if D changes after the rising edge of the clock. If D transition to logic level 0 VVDD is discharged through node DN and VVSS is charged if D transitions to logic level 1. After the falling edge of clock VVDD and VVSS are restored to their original state.

A 64-bit 7-stage Alpha architecture pipeline demonstrates the use of Razor-Lite. The processor was prototyped in 45nm SOI CMOS technology, and all 492 decode and execute registers,

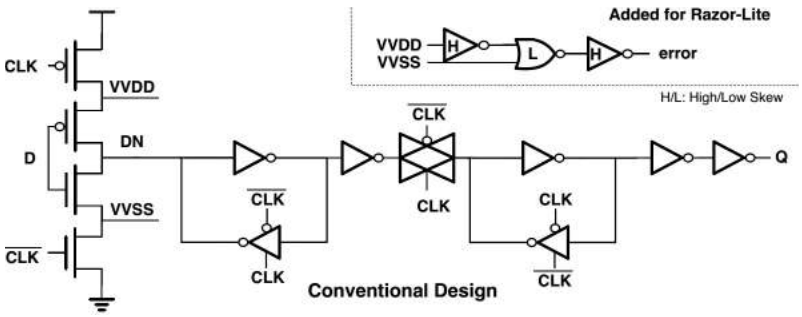


Figure 3.2 – Razor-Lite schematic [KWO14].

which are the processor's critical path, were replaced by Razor-Lite registers. Error outputs are grouped via an OR-tree and recorded at a pipeline register. After the rollback, the clock frequency is reduced by half during 4 cycles to allow the problematic instruction to execute correctly. The total penalty for an error is 11 cycles. Razor-Lite circuitry increased area in 4.42% and power in 0.3%. Peak energy efficiency is improved by 83% compared to the baseline processor.

3.1.4 Bubble Razor

Bubble Razor [FOJ13] is an architecturally independent approach to timing error detection and correction. Bubble Razor uses a two-phase latch-based data path instead of a flip-flop based data path. The error detection circuit presented in Figure 3.3 is similar to the one used in the original Razor. It is based on a shadow latch that captures the data before the main latch. An error is flagged if the value at the main latch changes after it becomes transparent.

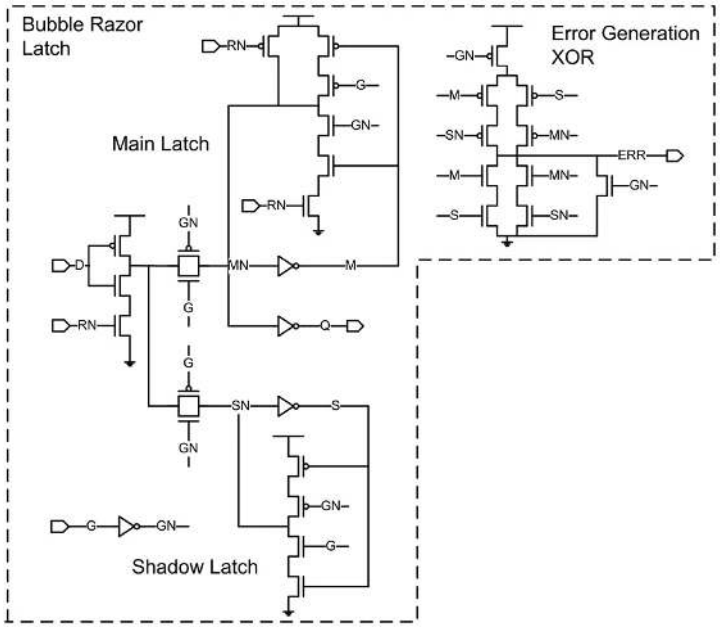


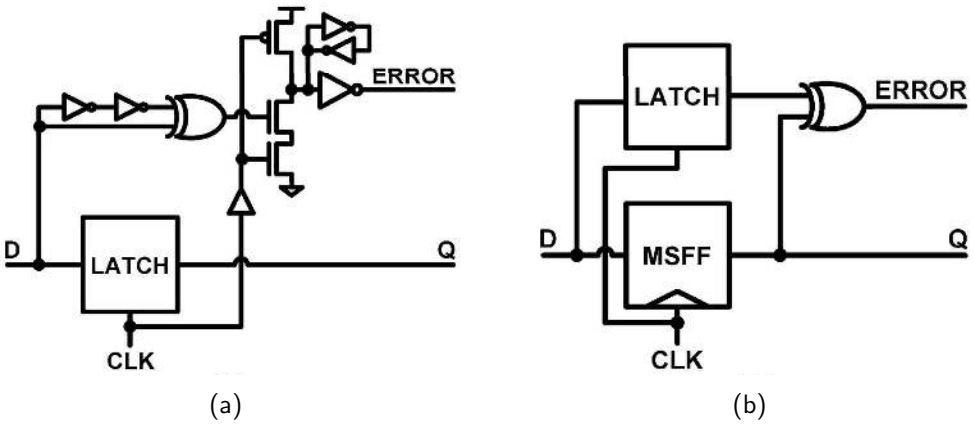
Figure 3.3 – Bubble Razor schematic [FOJ13].

Error correction is accomplished by local stalling. When a timing violation is detected, an error signal (bubble) is propagated to neighboring latches, preventing them from becoming transparent. Propagating bubbles gate off clock pulses throughout the circuit. This is done by a clock gating logic connected at each stage and adds time for the correct data to arrive at the latch that identified the timing violation. Once a neighbor latch receives a bubble, the bubble is propagated to the next stage and so on. For the cases where 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 at a 45nm SOI CMOS technology. A flop-based design was converted to a latch-based implementation. The error detection logic was added to all latches, resulting in 87% area overhead. When considering reduced margins, this implementation enables 100% throughput increase or 60% energy reduction when compared to the microcontroller with worst-case timing margins.

3.1.5 Transition Detector With Time Borrowing and Double Sampling With Time Borrowing

Bowman [BOW09] proposed two error detecting registers, the Transition Detector With Time Borrowing (TDTB) and the Double Sampling With Time Borrowing (DSTB), and also an instruction replay-based error recovery mechanism. The TDTB circuit illustrated in Figure 3.4(a) is a latch-based register that detects input transitions during the high phase of the clock. A data transition generates a pulse at the XOR output. The XOR output and the Clock feed a gate that produces the ERROR signal. This gate is equivalent to an asymmetric C-element, a sequential component used in asynchronous designs. This design removes the risks of metastability being propagated to the data path. However, there is still the risk of the error signal becoming metastable. This may happen if the data transitions occur slightly before the rising edge of CLK. In this case, the metastability is propagated to the control path, compromising circuit operation.



(a) (b)

Figure 3.4 – (a) TDTB circuit [BOW09]; (b) DSTB circuit [BOW09].

The DSTB depicted in Figure 3.4(b) replaces the transition detector of the TDTB by a shadow flip-flop. On the rising edge of CLK, the latch becomes transparent, and the input data

is sampled by the Master-Slave Flip-Flop (MSFF). The latch remains transparent during the CLK high phase. At the end of the CLK's high phase, if the values from the latch and the MSFF differ, an error is detected. Similar to TDTB, this approach eliminates data path metastability but also introduces the risk of metastability on the error signal.

The error recovery mechanism proposed is based on instruction replay. When a timing violation is detected, the erroneous data is invalidated, and the controller determines the appropriate instruction replay. During this process, the clock frequency is halved, and once the replay is complete, the clock is scaled back to its original frequency. A test chip using this technology was manufactured in a 65nm technology. Results show throughput gains up to 32% when operating at nominal supply voltage and up to 37% reduction in power consumption when reducing power supply while maintaining the nominal throughput.

3.1.6 Safe Razor

All known implementations are prone to failures due to the nondeterministic timing behavior producing metastability. SafeRazor [CAN15] is a Razor-based design that combines the Razor principle with globally asynchronous locally synchronous (GALS) design. This approach removes the metastability problems reported in [BEE14], and do not require a pipeline flush once a timing error is detected.

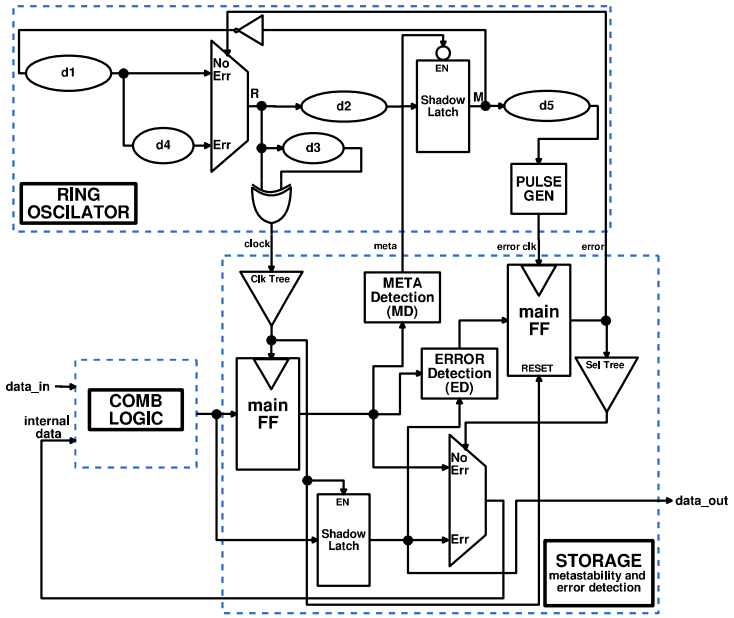


Figure 3.5 – SafeRazor module [CAN15]

Figure 3.5 shows the SafeRazor diagram. In the STORAGE block, a metastability detector and a timing error detector are implemented. If metastability is detected, the MD stalls the ring oscillator until resolving the metastability. If a timing error is detected, the *Err* input of the multiplexer inside the ring oscillator is selected and the cycle period is extended by the delay *d4*.

As a case study, a SafeRazor circuit with 3 GALS islands was implemented. Each island contains a 4-stage pipeline multiplier. Also, a standard flip-flop version and a Razor version were compared to SafeRazor. The designs were synthesized with Synopsys Design Compiler targeting 90nm standard cell library. Compared to the synchronous circuit, the total area overhead for Razor is 27.7%, while for the SafeRazor the overhead is 50.6%. The area increase is mainly due to the ring oscillator. The performance speeds up linearly, and for a higher operation frequency (60%) the performance is 15% better than Razor, while for a doubled frequency it is 7% worse.

3.1.7 Blade Template

Blade [HAN15] is a bundled-data timing resilient asynchronous template that, similar to the SafeRazor [CAN15], was proposed to overcome metastability issues in many of the proposed timing resilient architectures reported in [BEE14], but also to reduce high timing error penalties originated by the recovering mechanism of resilient architectures. Figure 3.6 illustrates the basic Blade architecture. A circuit implemented with Blade can be divided into four main blocks: (i) controller; (ii) delay lines (iii); error detection logic (EDL); (iv) data path. The Blade controller communicates with other stages using typical bundled-data channels. The δ delay controls the moment that data at the output of the combinational logic can be sampled and propagated through the latch. The Δ delay defines the amount of time that the latch is transparent, and is defined as the timing resiliency window (TRW). A timing violation is flagged if data changes during the TRW. The delay values δ and Δ must be designed to be sufficiently large to cover the longest critical path in their corresponding pipeline stage.

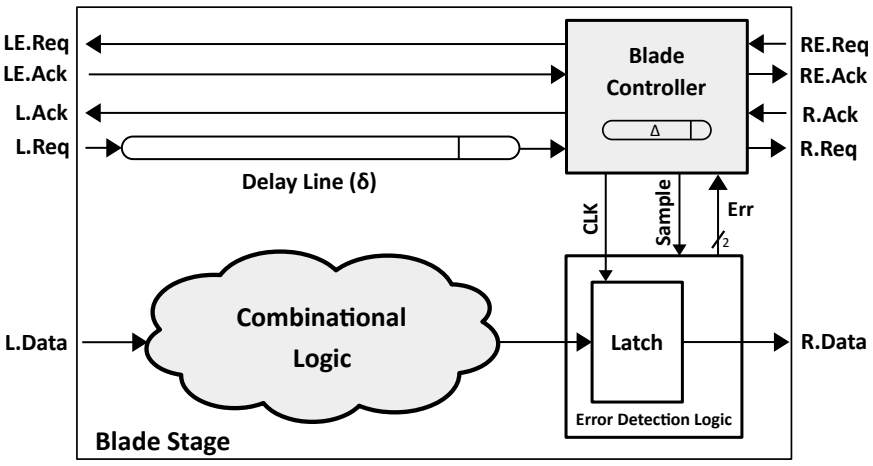


Figure 3.6 – Blade template [HAN15].

The EDL flags a timing violation by asserting its *Err* signal. To recover from the timing violation, the next stage delays by Δ its latch opening, until the correct data is propagated through the combinational logic. The Blade controller then communicates with its neighbors using channel L/R and an additional error channel LE/RE. These two channels are used to implement a new form of asynchronous handshaking protocol, called *speculative handshaking* [HAN15]. The implementation

is divided into three interacting Burst-Mode state machines [FUH95]. A more complete description of these blocks is presented in the following Chapters along with their fault analysis.

Blade's automated flow uses industry standard tools, including DesignCompiler and PrimeTime from Synopsys and NC-Sim from Cadence. A set of TCL and Shell scripts are used to automate the design flow, which converts a single CLK domain synchronous RTL design to an asynchronous Blade design. Figure 3.7 shows the synthesis flow and its five main circuit conversion steps:

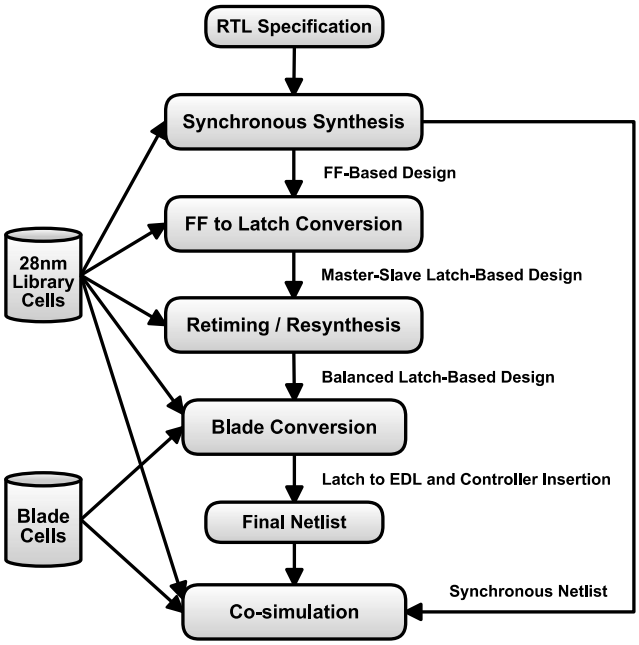


Figure 3.7 – Blade flow, adapted from [HAN15].

1. **Synchronous Synthesis**: The synchronous RTL description is synthesized to a flip-flop based design at a given clock frequency;

2. **FF to Latch Conversion**: Flip-flops are converted to a master-slave latches;

3. **Retiming**: The latch-based netlist is retimed using a target TRW. In this step latches that are near-critical path and thus should be converted to EDLs are identified;

4. **Resynthesis**: Minimize the area and timing overhead by reducing the number of EDLs. This is done by constraining some latches in near-critical path to have a delay no greater than the target frequency, which allows the use of standard latches rather than EDLs;

5. **Blade Conversion**: The resynthesized latch-based netlist is then converted to the Blade template. Clock trees are replaced by Blade controllers, delay lines, and error detection logic is also inserted, resulting in the final Blade netlist.

To validate the implementation a 3-stage version of the microprocessor Plasma [PLA14] targeting a 28nm FD-SOI technology is used as a case study. The circuit was converted from a 666MHz synchronous design to a Blade implementation with a TRW 30% of the clock period. After

retiming, a total of 456 latches were required to be converted to EDLs, and after the resynthesis, the number of EDLs decreased 27%. 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 performance. Performance of the Blade version increased 19%, with an average frequency of 793MHz.

3.1.8 Sharp and Error Detecting Latch

Sharp [WAU17] derived from the Blade template [BEE17]. As shown in Figure 3.8, there is one channel on each side of a controller. Both channels include three signals: Open, Close, and Acknowledge. The open signal tells the controller to open the latches and close tells to close the latches and acknowledge indicates that the close signal was received. The open signal speculates that the input data is valid, while the close signal guarantees a valid data.

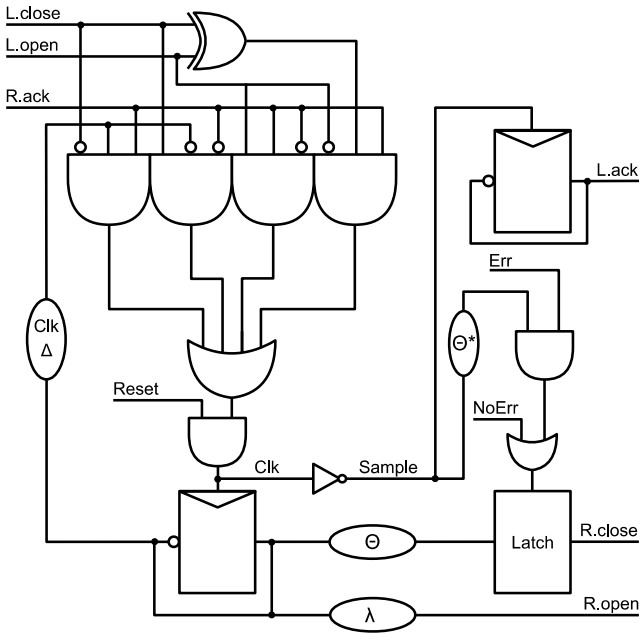


Figure 3.8 – The Sharp controller [WAU17].

Sharp uses a two-phase protocol. Latches are only opened when all neighboring blocks acknowledge that their latches are closed. That means Clk goes high when the XOR output is low and R.ack transitions. Open and close signals control the TRW in a Sharp controller. Sharp uses error detecting latches (EDLs) to signal when data at the latches changes while the latches are transparent [HUA16]. If an error is captured while the clock is high, instead of delaying the opening of later stages, Sharp delays the closing of all the immediate downstream latches, so that valid data propagates through those stages. Simulation results show that Sharp achieves 8% higher throughput when compared to Blade controllers.

An important aspect of timing resilient circuits is the need to detect timing violations. However, the area and power overheads resulting from replacing conventional sequential elements

with their equivalent error detection version can be substantial. In this sense, Sharp uses a new EDL architectures to detect timing violations [HUA16]. The two novel elements are the shadow-latched-based (SLB) EDL, which favors robustness and sensitivity, and the delayed-input-based (DIB) EDL that favors lower energy. Results show that proposed designs can achieve as much as 11.2% less overall power consumption and 7.8% smaller area when compared to state-of-the-art EDLs TDTB [BOW09], Sense-amp [TAD16] and Charge-sharing [KIM15]. Despite these results, neither Sharp or the new EDLs have had their testability addressed.

3.2 Testing Timing Resilient Circuits

A conventional VLSI chip is defective if it cannot pass at-speed delay tests, but timing resilient circuits are inherently tolerant to timing violations [YUA13]. Therefore, the pass/fail criteria for these circuits need to be re-examined. Furthermore, the additional circuitry for timing resiliency, including the error detection logic and the recovery mechanism need to be fully tested.

There are few works in this area. In [ANA15] a Scan Razor flip-flop (SR-FF) is proposed. This work is focused on reducing the power consumption by reusing the existing additional circuitry of the Razor architecture to build a scan cell instead of adding a typical scan design. Another similar approach is the Time Dilation (TimeD) scan architecture proposed in [FLO08]. In this case, a classic mux-D is modified to act as a timing violation detector and recovery mechanism. Besides the support for traditional off-line scan testing, the TimeD architecture is suitable for online (concurrent) timing error detection and recovery. Like the previous work, the authors are more concerned with a design that presents low area and performance overheads but do not show results about the testability of the resilient circuit itself, neither address the supported fault models and fault coverage results. On the other hand, the work of [YUA13], besides the DfT circuit design, it also addresses the new challenges of testing timing resilient designs.

The work of Yuan [YUA13] is probably the first and only to the date that evaluates the problems of testing timing resilient circuits, or as the authors refer to, timing speculative circuits. The proposed test methodology is based on the DSTB design presented in [BOW09], that consists of a latch, a shadow master-slave FF (MSFF) and an XOR gate, as depicted in Figure 3.9(a). The latch operates as a data path memory element for regular computation, while the MSFF captures the input value for timing violation detection. The latch is transparent on the high clock phase while the MSFF samples at the rising edge of the clock. If a timing violation occurs the values stored at the latch and the MSFF will be different, and the XOR gate sets the error signal accordingly. Consequently, with such a timing speculator design, the high clock phase is the detection window for timing violations. The Scan-based version of the DSTB is shown in Figure 3.9(b).

The test flow presented by [YUA13] consists of three parts. First, static fault testing (e.g., stuck-at faults) is performed in the combinational logic and the Error Generation and Propagation (EGP) circuit that is used by the system recovery. Next, the delay test of the combinational logic

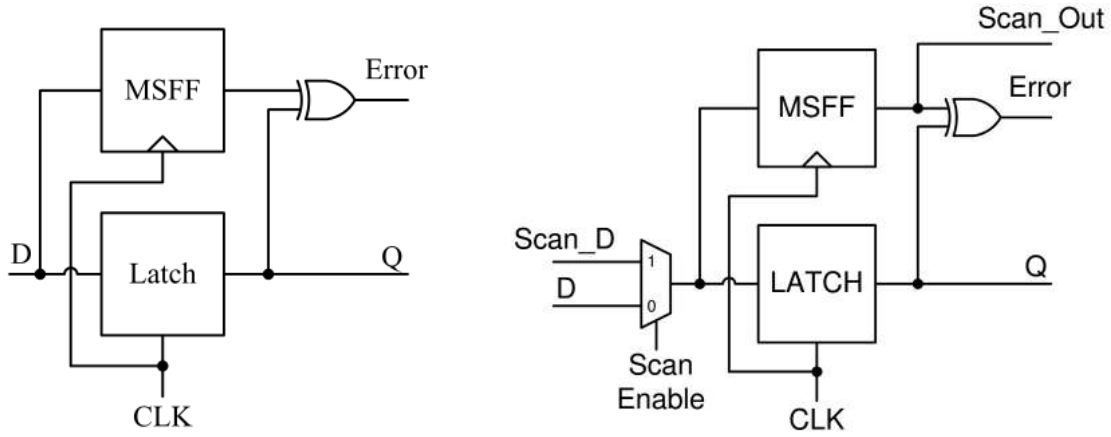


Figure 3.9 – (a) DSTB circuit [BOW09]; (b) Scan-based DSTB circuit [YUA13].

is performed, and at last, the delay testing of the EGP is conducted. As a way to identify delay faults that exceed the detection window, the authors propose that during the test, the clock cycle be increased, so that the high phase (detection window) is equivalent to the worst-case of the combinational logic. If a timing violation is identified at this stage, it means that when in normal operation the designed detection window is not able to detect a timing violation in the tested path. This test approach assumes clock cycle controllability. Results show 100% coverage for the EGP circuits, but despite the author's comments about the low DfT costs, numbers about area overhead are not explicitly shown. Even though the work has proposed an attractive solution for structural testing of path delay and stuck-at faults, the test methodology is designed for synchronous circuits, and the absence of a global clock may hinder the use of such approach with asynchronous timing resilient circuits.

3.3 Testing Asynchronous Circuits

As previously discussed in Section 2.3, asynchronous circuits promise advantages over synchronous circuits, such as lower power consumption, average-case rather than worst-case timing behavior, improved modularity and more tolerance towards PVT variations. Moreover, in [WOJ14], according to 2012 ITRS [ITR12], the asynchronous design style will be widely applied in future nano-electronics. Even though this design style provides advantages, there are two factors that hinder its massive use: (i) lack of mature design tools; (ii) test related approaches.

From the test perspective, the primary difference between asynchronous and synchronous is the self-timed computation versus the lock-stepped computation. From the technology perspective, there is no distinction between manufacturing asynchronous or synchronous CMOS circuits, which means that defects are similar in both designs [RON99]. For this reason, synchronous test techniques have been adapted to address the asynchronous test problem.

Fault models are the same for synchronous counterparts, i.e., stuck-at faults, bridging faults, delay faults, functional faults, etc. However, fault effects may be different when shifting from synchronous to asynchronous circuits. For example, a stuck-at fault in the handshake control signals (e.g., request or acknowledge) of an asynchronous template can result in a system halt, but a stuck-at within the data path of the asynchronous circuit has the same effect as in its synchronous counterpart.

According to Section 2.3, there are two main asynchronous design styles, QDI and BD. As for testability, this division persists, and different test techniques have been proposed for both styles. Remember that this research is focused on BD templates, more specifically, Micropipelines, Mousetrap, GasP, and Click templates, which were recently addressed in works that deal with the test of asynchronous circuits [RON15].

The Micropipeline concept was introduced in the late 80's, and the papers about the test of such circuits are from the early 90's. The test problem is divided into three classes: (i) testing the control part; (ii) testing the combinational logic in the data path; (iii) testing the memory element (latches). This division is still used nowadays. Note that this division does not cover the delay lines required for the correct operation of the Micropipelines. Despite the challenges of testing asynchronous circuits, the authors state that the control part is easily testable, because a stuck-at fault in this part halts the circuit. However, this is not entirely true for other approaches [SHI05], such as Mousetrap. The test strategy proposed by Pagey [PAG92] consists in putting all latches in transparent mode, creating a big and unique combinational circuit, where standard ATPG tools for combinational circuits can be used. This approach can be applied for detecting stuck-at faults and delay faults, except that for delay faults only the overall delay of the combinational logic is evaluated, and not the individual blocks between stages. However, test pattern generation for this technique becomes a complex task as the circuit size scales.

The scan technique is the most relevant DfT test technique in industry. It can be applied to various types of faults including stuck-at, bridging, delay, etc. Full-scan approaches for testing Micropipelines have been proposed by several authors [KHO94] [PET95a] [PET95b] [KAN99]. These publications propose custom scan cells to create the scan path and they can test for stuck-at and delay faults. The full-scan approaches of [BER02] [BEE02] [BEE03] adapt a level-sensitive scan design (LSSD) for testing stuck-at faults in Micropipelines. The same authors later proposed, in [BEE05], a multiplexer based scan design, where instead of latches multiplexers are used to break the feedback loops, and thus they still use combinational test pattern generation without the overhead of scan latches. The problem with these full-scan approaches is mainly the area overhead that in the best scenarios are around 20% to 40% for the evaluated circuits.

The partial-scan technique was also adapted for testing asynchronous circuits to reduce area overheads. In [KHO95], the partial-scan is applied to check for stuck-at faults in the control part of the circuit. In their design method, control circuits are built using a variety of macro modules, such as a *TOGGLE* module, which is modified to a scannable version. It is said to be smaller than the full-can approaches, but few details are presented. The partial-scan method

proposed in [RON96a] [RON96b] is used to test data paths and handshake components, such as multiplexer *MUX* and sequencer (*SEQ*), that are necessary for their asynchronous design approach. Test patterns are generated for stuck-at faults and bridging faults. Furthermore, a new handshake component, *HOLD*, is proposed, and it is used to hold the circuit between the handshake protocol, adding test controllability. A downside of this last approach is that it is specially developed for the Tangram compilation methodology [BER93], where the control circuit is compiled directly from a high-level source code into delay-insensitive modules, rather than being synthesized using a low-level standard-cell library, as it is done in [KIS98]. The results show improvements in the total number of dummy latches, which are the latches not used in normal operation, only for testing. The dummy latches are reduced to less than 1%, but this is compared to a circuit that also has scan, so it is not clear the total area overhead added to the circuit.

Petlin proposes a BIST technique for testing Micropipelines in [PET97]. It consists of adapting the BILBO register [RUS89] to the asynchronous context. The asynchronous BILBO register can be configured as a pseudo-random pattern generator, a signature analyzer, a shift register or to operate in normal mode. This approach allows stuck-at and delay fault testing, but the problem with this approach is the high area overhead required by the several functional configurations. For instance, for delay testing, a double-sized BILBO register must be implemented.

The Mousetrap asynchronous template relies on certain timing constraints to guarantee functional robustness, and unlike Micropipelines, a stuck-at fault may not lead to a pipeline stall [SHI05]. The challenges for testing stuck-at and delay faults are considered by the following works. Shi [SHI05] proposed a test method for the Mousetrap circuit. The data path test is similar to the one presented by Pagey [PAG92], where the circuit is treated as a big and single combinational circuit for stuck-at fault testing. The same research group proposed in [GIL06] a more general test approach for delay fault testing of Mousetrap and GasP, where area and performance overheads are reduced when compared to [SHI05]. However, this is not quantified in the results. The test strategy relies on functional test methods for generating the test patterns that will load and unload the pipeline and are likely to expose the delay faults. For stuck-at faults, the applied technique is the same of [SHI05].

As presented in in Section 2.3, Click is a flop-based asynchronous template that resembles synchronous circuits as much as possible. Besides simplifying the design flow, the use of flip-flops in the control and data path also facilitates the DfT of such circuits, since conventional synchronous scan techniques can be applied. A scan-testable version of the Click controller is presented in [PEE10]. Except for the reference to the multiplexer based scan design method used in the design flow [BEE05], no further details about the test is presented.

Except of [GIL06], all previous works address the testability for a particular design style. An attempt to overcome this problem is presented by Roncken [RON15]. The so-called naturalized communication unifies the existing families of BD circuits. Under this naturalized communication, the difference between Micropipeline, Mousetrap, GasP and Click circuits becomes the communication channels, referred as *links*, while the flow control and data computation, called *joints*, are

identical, thus making them interchangeable. The joints are the meeting points for links to coordinate states and exchange data. Traditional scan test techniques are dedicated to controlling state and a novel proper-start-stop circuit, called MrGO, is dedicated to controlling actions (adds controllability), like freezing joints to put the circuit in a full state. Details about the MrGO circuitry are presented [RON15], but the scan approach used for testing delay and stuck-at faults is not described, the same is true about fault coverage and the area overheads.

3.4 State of The Art Conclusion

Despite the increasing evolution of resilient architectures, little has been done regarding the testability of these designs. A problem when considering the testability of timing resilient architectures is the area overhead of test circuitry since the resilient implementation already incurs in significant area overhead. Some error detection circuits also use custom sequential cells that are not available in most technology libraries. Thus, it is not possible to take advantage of automated DfT flow with commercial EDA and ATPG tools. Moreover, these detection circuits increase the number of sequential cells, which increases the scan cost. Also, most of the existing resilient architecture proposals suffer from metastability problems. Table 3.1 summarizes the timing resilient architectures. The DfT column indicates if a DfT insertion is addressed in the literature. The Metastability column informs if the design suffers from metastability problems and the last one if it is a synchronous, asynchronous or GALS design style.

Table 3.1 – Summary of timing resilient architectures.

	DfT	**Metastability**	**Design Style**
Razor	[ANA15]	Yes	Sync
Razor II	NA	Yes	Sync
Razor Lite	NA	Yes	Sync
Bubble Razor	NA	Yes	Sync
Safe Razor	NA	No	GALS
TDTB/DSTB	[YUA13]	Yes	Sync
Blade	NA	No	Async
Sharp	NA	No	Async

Along with Safe Razor, Blade addresses the metastability issues of previous Razor family and the propagation of metastability for the control path observed with TDTB and DSTB. An advantage of Blade is the lower area overhead when compared to original synchronous non-timing resilient designs. Blade leverages the benefits of asynchronous (e.g., average-case timing behavior) and timing resilient techniques (e.g., tolerance towards PVT variations). However, it inherits the testability issues of both techniques, and adds new challenges, such as delay testing of timing resilient circuits, addressed only in synchronous timing resilient circuits [YUA13], and the testability of specific circuitry, such as the controller and the error detection logic.

4. ERROR DETECTION LOGIC TESTABILITY

As mentioned before, the Blade test problem is divided into four main parts: the EDL, the controller, the delay lines, and the data path. This Chapter presents the challenges of using ATPG tools to test the EDL part, along with contributions 1 and 2 (see Section 1.2). The first contributions consist in Blade's EDL fault analysis, and the fault classification proposed to detect faults in the EDL. The second contribution is the proposal of a testable error detection logic (TEDL).

4.1 Blade EDL

Before any further discussions, it is important to define how the elements are referred to from now on. Different from other works, EDL refers to Error Detecting Logic and not Error Detection Latch, which is the entire logic used to detect and propagate an error to the controller. The set of sequential elements (latches or flip-flops) and the transition detector mechanism is called Error Detecting Sequential (EDS), and the Transition Detector, which is located inside the EDS, is referred as TD.

Blade's EDL [HAN15] (Figure 4.1) consists of latch-based error detecting sequential, that are based on the TDTB EDS [BOW09], asymmetric C-elements and Q-Flops [MOL88]. The TD is based on an XOR function of the data input and a delayed version of this input, which produces a pulse at X, thus indicating a data transition. The C-element acts as a memory cell that stores any violation detected during the high phase of the CLK. Thus, the C-element switches to 0 when CLK is at 0 and to 1 only when both the CLK and an XOR output is at 1. The output of the C-element is sampled by the Q-Flop at the end of the TRW, defined next.

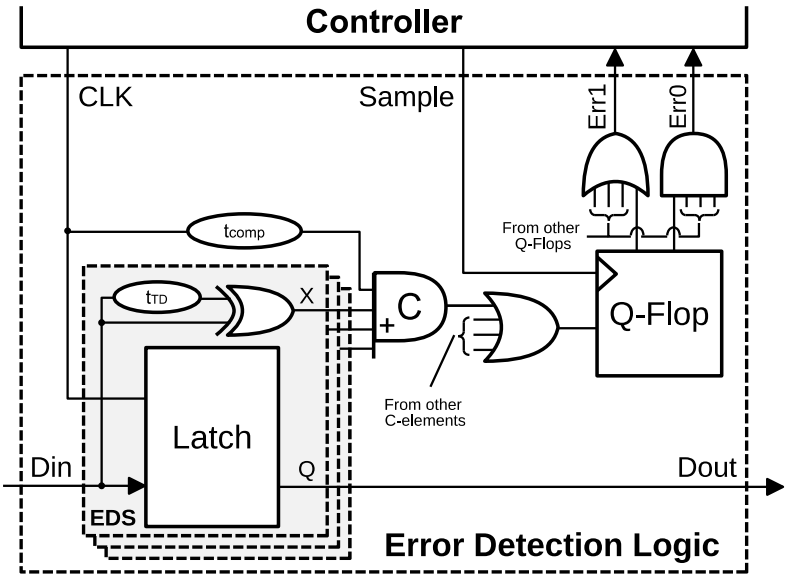


Figure 4.1 – Blade error detection logic diagram, adapted from [HAN15].

The Q-Flop ensures safe operation against metastability in some *X* signal and the C-element by applying a filter that prevents its outputs from becoming metastable. The dual-rail signal *Err*, composed by wires *Err0* and *Err1*, stalls the controller until the outputs are stable and it can safely evaluate if an error occurred. The delay element t_{TD} defines the transition detector pulse width, while t_{comp} is a small compensation delay to ensure that a transition before the rising edge of *CLK* is not flagged as a violation. The other logic elements (ORs and AND gates) are designed to amortize the area overhead of the C-elements and Q-Flops across multiple pipeline stages.

The timing overheads associated with the EDL for a single Blade stage are shown in the timing diagram of Figure 4.2. The delays are divided in five components: (i) the propagation delay from *Din* to *X*, $t_{X,pd}$; (ii) the pulse width of *X* defined by the delay element t_{TD} shown in Figure 4.1, $t_{X,pw}$; (iii) the C-element propagation delay, $t_{CE,pd}$; (iv) the Q-Flop setup time, $t_{QF,setup}$; and (v) the propagation delay of the OR gate before the Q-Flop, $t_{OR,pd}$. The sum of components $t_{X,pd}$ and $t_{X,pw}$ corresponds to the compensation delay t_{comp} described previously. Several timing constraints must be considered for a Blade design [HAN15], such as *Contamination Delay* and *Maximum TRW*, for this work the focus will be the TRW, which is defined in Equation 4.1.

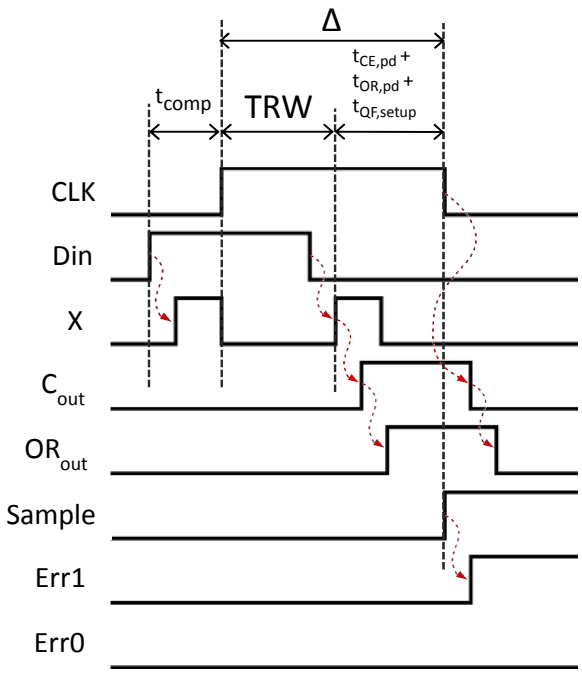


Figure 4.2 – Timing diagram of a timing violation being detected inside the EDL, adapted from [HAN15].

$$TRW = \Delta + t_{X,pw} - (t_{CE,pd} + t_{OR,pd} + t_{QF,setup}) \tag{4.1}$$

4.2 ATPG Fault Coverage Analysis

When considering the testability of timing resilient circuits, previous works look at traditional approaches used in industry, such as the scan technique, which can be applied to various types of structural faults including stuck-at and delay. One of the challenges when testing these circuits is their compatibility with commercial DfT tools, mainly due to the use of custom cells that are not available in most technology libraries, and in the case of Blade, there is the asynchronous factor, that is also not supported by these tools. The analysis in this Section is intended to show the EDL fault coverage using standard DfT techniques and tools.

4.2.1 Test Scenario

A netlist targeting 28nm FDSOI technology is generated with DesignCompiler from Synopsys to evaluate the stuck-at fault coverage and the TetraMAX ATPG tool from Synopsys to reports coverage results. The netlist consists of a two-stage Blade pipeline (Figure 4.3), where the EDSs of the first stage are directly connected to the ones of the second stage. This scenario simplifies the integration with the Synopsys tools since it removes the particularities of an asynchronous resilient template from the analysis. Instead of asynchronous controllers, nonoverlapping clock signals are created, one for each stage, and the circuit becomes essentially a synchronous latch based design, where all primary inputs are controllable, and all primary outputs are observable.

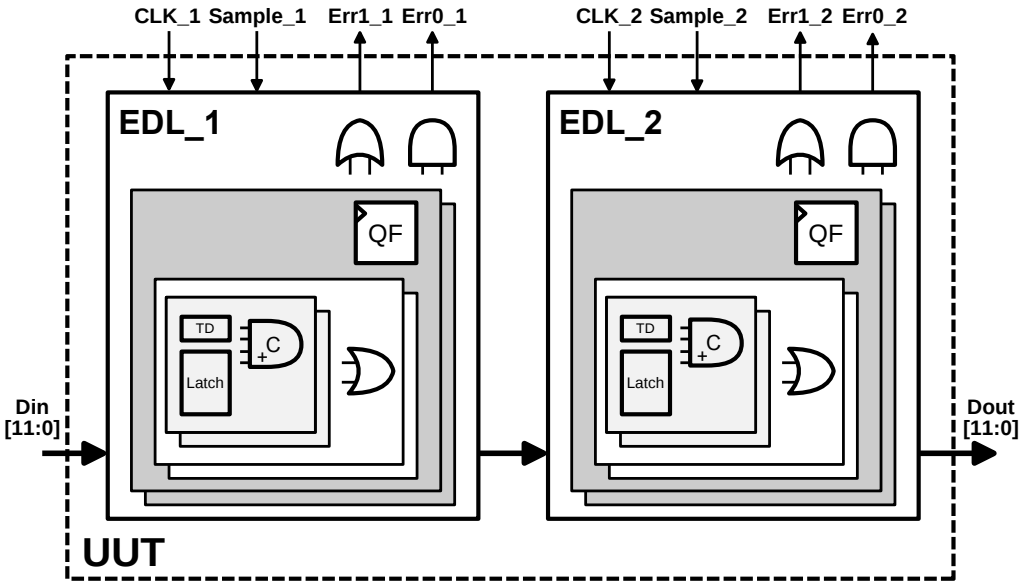


Figure 4.3 – A proposed ATPG test scenario. Transition detectors (TD) are connected to four-input (CLK plus three TDs) C-elements (C) and the C-elemets connected to two-input OR gates. In total there are 6 TDs to each OR gate. The OR gate is connected to a Q-Flop (QF). With 2 QF per stage, each step is 12 bits wide.

In Blade's EDL three custom sequential cells are not available in the foundry standard cell library, the EDS, the C-element and the Q-Flop. Moreover, their correspondent scan cells are either not available in most libraries or not supported by commercial DfT flows. Thus, it would not be possible to use automated tools to obtain the fault coverage reports. Therefore, these unconventional sequential cells are modeled as macroblocks consisting of only cells already supported by the 28nm FDSOI library, including latches and flip-flops, to implement the sequential behavior. An advantage of this approach is that Blade's EDL can be evaluated with or without a scan chain because the sequential cells (latches and flip-flops) are recognized by the DfT tools to perform automatic scan chain generation. The asymmetric C-element behavior is similar to the one in [BEE03] (Figure 4.4), and the Q-Flop is described as a standard flop that is reset when its enable input is low, which is, regarding behavior, equivalent to the Q-Flop.

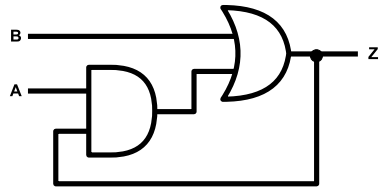


Figure 4.4 – Asymmetric C-element with standard cells [BEE03].

4.2.2 Fault Coverage Results

The fault simulations performed by the ATPG tool considered only single stuck-at faults. The tool is configured to ignore all the first stage, redundant wires, and internal nodes from the macro cells, which gives a total of 12 fault points. These 12 points are the labeled wires of Figure 4.5, named *w1* to *w12*. The fault coverage of the Blade's EDL is evaluated considering two scenarios: using no scan structure and replacing all latches by Level Sensitive Scan Cells described in [JUR17].

Table 4.1 shows the summary reported by TetraMAX. The faults are classified into five classes, and the fault coverage is calculated by the total number of faults divided by the number of Detected plus the Possibly detected faults. The low fault coverage is related to the lack of observability and controllability of internal nodes. Even replacing the latches for LSSD scan cells is not enough to improve the fault coverage. This low coverage is mainly caused by the C-element and the Q-Flop, that, due to their functional behavior, do not allow the tool to stimulate and observe faults in their path.

4.3 Behavioral Fault Coverage Analysis

Different from the previous analysis, instead of relying on classic ATPG approach, a fault classification method based on the functional behavior relates the existence of internal faults to

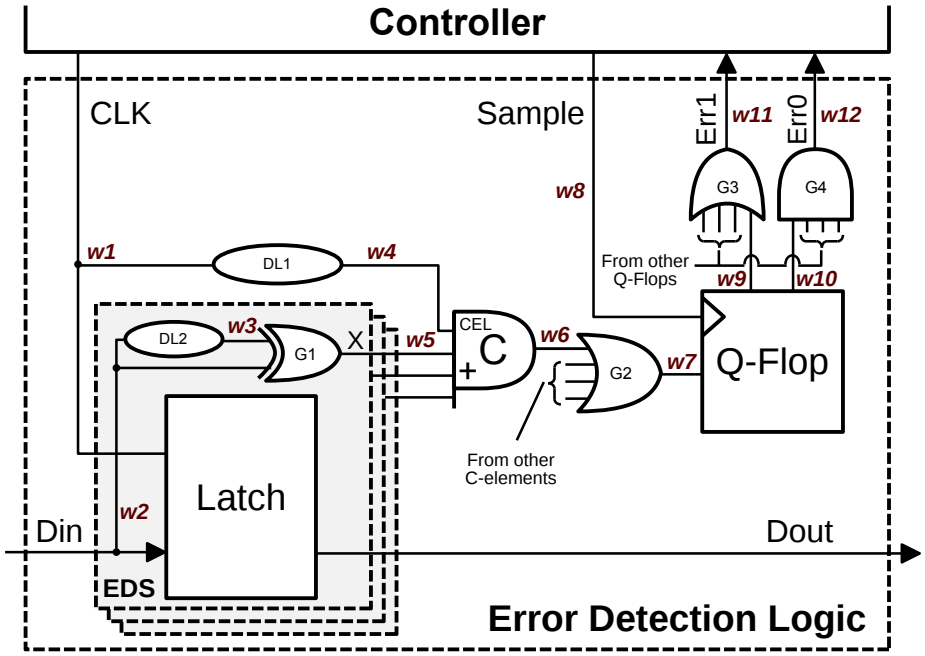


Figure 4.5 – Blade's EDL diagram with labeled wires, adapted from [HAN15].

Table 4.1 – TetraMAX stuck-at fault summary report for an EDL block.

	no-Scan	**Scan**
Detected	6	7
Possibly detected	2	1
Undetectable	1	1
ATPG untestable	0	0
Not detected	15	15
not-controlled	(7)	(7)
not-observed	(8)	(8)
Total Faults	24	24
Fault Coverage	33.3%	33.3%

certain behavioral patterns observed in the Blade design [KUE16]. Faults are simulated and reported through the following fault simulation environment.

4.3.1 Fault Simulation Environment

This Section describes the fault simulation environment that allows controlled fault insertion at specific locations of the circuit, such as forcing wires to fixed values to produce a stuck-at or to modify the propagation delay of logic gates or other elements of the design to simulate a delay fault. This is also possible to simulate the circuit with a timing violation at the EDL's input. It means that the combinational logic takes longer than the δ delay to propagate correct data. As discussed in the following sections, the ability to generate a timing violation is essential to detect some specific faults in the EDL.

The behavioral description of the test scenario (Figure 4.6) consists in a 3-stage Blade pipeline, where all stages are identical regarding the timing constraints. The Verilog descriptions have signal assignments with configurable propagation delays to allow delay fault simulation, such as presented in the code of Figure 4.7. Between each pair of stages, there is a string of inverters acting as the combinational logic. The simulation generates a log file informing the timing violations found in each stage. Another log file describes the injected data pattern at the first stage, and the output data received at the last stage. From these logs, the environment extracts the results presented in the next sections.

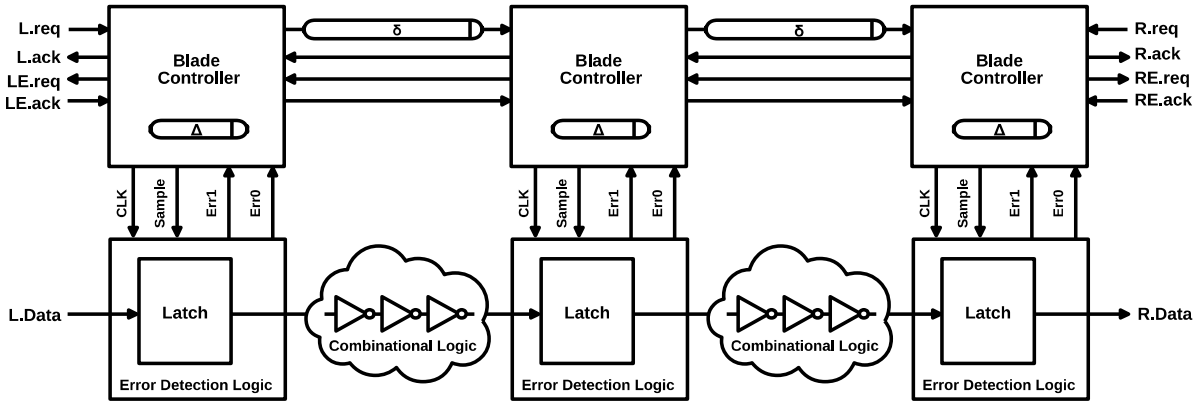


Figure 4.6 – Test scenario used for behavioral fault simulations.

The environment only simulates a single fault at a time, and these faults are always inserted at the middle stage (the second one). The other two stages are fault-free, and they are used to observe how the overall circuit behaves when one of the stages is faulty. The accepted faults are *stuck-at-0* (SA0), *stuck-at-1* (SA1), *shorter propagation delay* (SPD) and *longer propagation delay* (LPD). Stuck-at faults are inserted in all wires inside the error detection logic, and the propagation delay fault is injected in all logic gates. Also, data is not masked by the combinational logic, and a data transition is always propagated from one stage to another. In the following discussion, the use of the term *error* indicates that a timing violation (not the stuck-at or propagation delay fault) was detected by the EDL.

Faults are individually analyzed in the following sections. The elements and labeled wires of Figure 4.5 identify the fault locations considered and state how these are referenced in the text. Throughout the analysis, the observed fault effects are discussed concerning how they can be applied to the fault detection. For the detection of some particular faults, a timing violation (TV) must be generated to stimulate parts of EDL that are only activated in such situations. By default, a TV is not required to detect the fault. However, when the TV is necessary, it is explicitly referred and discussed in the text.

```
module qflop(CLK, D, Q, Qbar);
  parameter  PROP_DELAY = 0.020;
  realtime   PD = PROP_DELAY;

  input CLK, D;
  output Q, Qbar;
  reg Q, Qbar;

  always @(CLK)
  begin
    if (CLK == 1) begin
          Q    <= #(PD) D;
          Qbar <= #(PD) ~D;
        end else begin
          Q    <= #(PD) 0;
          Qbar <= #(PD) 0;
        end
  end
endmodule
```

Figure 4.7 – A Q-Flop Verilog description with configurable propagation delays.

4.3.2 Stuck-at Fault Analysis

Although the controller testability is not the focus of this analysis, understanding its behavior once the EDL has a fault is important for the fault classification. In particular, there are four signals inside the EDL that may affect the controller behaviour: *CLK*, *Sample*, *Err1* and *Err0*. The *CLK* signal controls the rise and fall of the δ delay line inside the controller, while *Sample* controls *Err1* and *Err0* rise and fall. As previously described, the controller stalls waiting for *Err1* or *Err0* signals to either rise or fall before continuing the protocol process, and the same will happen if the *CLK* signal does not change its value. More details about the controller are presented in Chapter 5. As demonstrated next, some particular faults in the EDL can halt the controller.

The first fault to be analyzed is an SA0 at $w1$. In this case, the pipeline halts waiting for the *CLK* to go up. The same will happen with an SA1 in this wire, but instead, the pipeline halts waiting for *CLK* to go down. Both faults are easily *detectable with a functional test* due to a pipeline halt. An SA0 or an SA1 at $w2$ makes the latch to always capture a constant value due to the stuck value in its input. These faults are *detected at the output of the pipeline*.

The next fault point is $w3$. An SA0 or an SA1 at this wire can cause a false error generation that depends on the input data. For instance, if $w3$ is SA0, an error is asserted only if the input data is at logic level 1. This means that the input data must be controllable. The observability of *Err1* or $w7$ would be enough to extract information regarding the presence of timing violations. For the rest of the analyzes, the *Err1* will be the observability point.

A stuck-at fault at $w4$ differs from a fault at $w1$ because the former does not affect the latch. An SA1 at wire $w4$ always produces an error at the faulty stage, while an SA0 causes the stage never to detect a timing violation since C-element clock never goes up. For this case, a TV must be generated at the stage under test, and the *Err1* signal of the next stage observed to detect this SA0 fault. This is possible due to a particular feature of Blade's template, where the timing

violation missed at a stage, in this case because of a stuck-at fault, can still be detected in the following stages assuming that the transition is not masked between stages. Since the faulty stage does not capture the error, the delay is not extended as it would be expected in such a situation, and an invalid data is propagated to the next stage. The following stage eventually detects the TV and delays its latch opening, such that its following stage receives the correct data. This and some other faults are defined as *detectable in the next stage*.

Another fault point is $w5$, but this fault analysis can also be extended to $w6$ and $w7$. An SA0 at these nets cause the EDL not to detect timing violations. As in the previous analysis, a TV must be inserted so that the fault is detectable at the next stage by observing its *Err1* signal. The SA1 is detected at the faulty stage since an error is always flagged by *Err1*. The next signal, $w8$, is the Q-Flop *Sample* signal. The Blade controller requires that the signals *Err0* or *Err1* must go up and later go down before a new protocol cycle is initiated. So, if $w8$ is SA0, both error signals are also at logic level 0, causing the pipeline to halt. The SA1 has the same behavior but in the opposite direction, while the SA0 halts the pipeline because *Err0* or *Err1* never go up, the SA1 halts the pipe because the signals never go down.

As in the previous analysis, the next fault is also detected by a halt in the pipeline. A TV must be generated in the faulty stage to detect an SA0 at $w9$. In this case the *Err1* should go up but, because of the fault, it never goes up. On the other hand, the SA1 at $w9$ can be detected with a functional test when the pipeline halts waiting for *Err1* to go down. The same is true for an SA1 at $w11$. The detection of an SA0 at $w11$ is similar to $w9$, except that a TV is not required in this case. An SA0 at $w10$ can be detected with a functional test once the pipeline halts waiting for *Err0* signaling. On the other hand, a TV must be injected at the data input to detect an SA1 fault. The last fault point is $w12$, where for both SA0 and SA1, the pipeline halts waiting for *Err0* to go up or down, respectively, and they are both testable with functional testing.

4.3.3 Delay Fault Analysis

As mentioned earlier, the propagation delay of gates, latch and delay lines are considered for the delay fault analysis. The faulty gate has either an SPD fault or an LPD fault, compared to the design timing constraints. An SPD fault, except for the delay lines, does not represent a threat to the overall circuit operation. For example, looking at the C-element or the following OR gate, an SPD at anyone of these will increase the TRW presented in Section 3.1.7. With a bigger TRW, the faulty stage catches timing violations that otherwise would not be captured by the EDL. These group of faults that do not affect the circuit functionality can be defined as *don't care* faults. The Q-Flop and the following OR and AND gates, in the presence of an LPD fault, also do not produce any catastrophic failure in the circuit. However, these faults can be detected by looking at a performance degradation of the pipeline, since the slow propagation of signals *Err0* or *Err1* would delay the completeness of the protocol cycle.

Even though this analysis does not consider the data path testability, an LPD fault in the latch can be seen as a delay fault in the data path. In this case, a timing violation is detected at the next stage, and a high error rate at a pipeline stage would suggest that there is a delay fault in the data path. Unlike SPD, an LPD at the C-element and the following OR gate reduces the TRW, and this affects the circuit resiliency directly. In particular, this fault causes the circuit to behave differently depending on the moment where the timing violation occurs. For example, looking at the timing diagram of Figure 4.2, the $t_{CE,pd}$ can be so long that the last *X* pulse is not propagated until the falling edge of the *CLK*, thus missing the TV. In this case, the violation is propagated to the next stage. With the same propagation delay fault in $t_{CE,pd}$, if this last *X* pulse appears right after the rising edge of *CLK*, then the violation would still be captured at the faulty stage. This demonstrates that the TV must be inserted at specific lines and specific moments inside the TRW.

The next fault is an LPD at the XOR gate. Using the timing diagram of Figure 4.2 as the reference, this fault dislocates the *X* pulse inside the TRW by increasing the $t_{X,pd}$, which produces a false error that is dependent on the input data, like the analysis of a stuck-at fault at $w3$. Looking at the DL2 delay line (Figure 4.5), an SPD fault at this element makes the *X* pulse width ($t_{X,pw}$) shorter. This fault harms circuit operation if the pulse gets shorter than the setup time of the C-element. Otherwise, the fault is a *dont't care* fault. In the case where the pulse width is shorter than the setup time of the C-element an error is missed at the faulty stage, but the next stage detects the error. A method for injecting a TV is necessary to detect this last fault. Similarly to the XOR gate LPD fault, an LPD at the t_{TD} delay line generates a false error that depends on the input data, with the difference that instead of shifting the *X* pulse into the TRW, $t_{X,pw}$ increases the pulse width that grows into the TRW.

The last delay fault analysis is at DL1. An SPD at this delay element reduces the compensation delay added to ensure that *X* is not captured before the rising edge of *CLK*. In this case, the C-element captures the *X* pulse before the correct time, which generates a false error. This fault can be dependent on the input data, although it is unlikely that all inputs are always capturing the same logic value. If at least one data input changes every clock cycle, an error is always observed at *Err1*. An LPD at DL1 causes the C-element to late capture the *X* pulses. Comparing with the timing diagram of Figure 4.2, this fault dislocates the *CLK* to the right, which can be seen as a decrease in the TRW, and some timing violations may not be captured by the EDL. Like the LPD fault at the C-element, to detect this fault, a TV must be inserted at the beginning of the TRW. If this fault is present, the error is not captured at the faulty stage, only in the next stage. If the TV is inserted at the end of the TRW, an error is flagged at the faulty stage, which is the expected behavior of a fault-free stage. Thus the fault is not detected.

4.3.4 Discussion about the fault effects on the EDL

As noted in the previous fault analysis, a faulty EDL triggers different effects in the overall circuit that can lead to fault detection. Some of these faults only affect the performance of the

circuit. This behavior can either be caused by a high rate of false errors or due to an LPD at a gate that does not prevent the EDL from detecting timing violations, such as the *AND Gate*. In both cases, the circuit can still be commercialized at a lower cost, since it has some performance degradation, but is fully functional.

There are some faults effects that directly impact the circuit resiliency, by completely disabling the EDL (e.g., ST0 at $w5$, $w6$ and $w7$), increasing the TRW (e.g., SPD at C-element) or decreasing the TRW (e.g., LPD at DL1). Similar to the performance problem, an EDL that has its TRW increased by a fault is still functional and it can later detect timing violations compared to the expected EDL behavior. When the TRW is reduced, the EDL may not capture all the timing violations or, in the worst case, it may miss all the timing violations when the EDL is disabled by a fault, such as an SA0 at the Q-Flop sample signal. The detection of these faults is critical, once the faulty EDL propagates invalid data to be processed by the following combinational logic, which can lead to a significant system failure.

The faults called *detectable in the next stage* can only be detected right in the next stage if the TRW of the next stage is equal or longer than the TRW of the stage under test. Otherwise, when the TRW of the next stage is shorter, the timing violation can be propagated beyond the next stage.

4.3.5 Fault Classification

So far the faults were discussed individually, looking at the side effects observed in the overall operation of the circuit and how each one can be detected. The presented classification generalizes the relationship between cause and effect of the analyzed faults. Table 4.2 shows the fault effects observed during the circuit simulation that are relevant to the fault classification.

Table 4.2 – Relevant effects for the fault classification.

Acronym	Description
UN	undetectable
PST	pipeline output stuck at a value
PH	pipeline halted
ERR_ST	errors in the faulty stage
ERR_NST	errors in the next stage

Based on the individual analysis of the faults, the cause, which is the fault simulated, is correlated to the effects listed in Table 4.2. The result of this is a fault classification that groups faults with similar effects. Tables 4.3 and 4.4 show the classification for the stuck-at faults and the propagation delay faults, respectively. Both tables present the classification assuming that a method to inject timing violations is available (*w/ TV*) and without (*wo/ TV*) this method. It is possible to see that without the TV some faults are undetectable. It demonstrates that the ability to control the injection of timing violations is important for the testability of the EDL.

Table 4.3 – Blade EDL classification for the stuck-at fault model. Timing Violation (TV).

Faulty	SA0		SA1	
Line	wo/ TV	w/ TV	wo/ TV	w/ TV
$w1$	PH	PH	PH	PH
$w2$	PST	PST	PST	PST
$w3$	ERR_ST	ERR_ST	ERR_ST	ERR_ST
$w4$	UN	ERR_NST	ERR_ST	ERR_ST
$w5$	UN	ERR_NST	ERR_ST	ERR_ST
$w6$	UN	ERR_NST	ERR_ST	ERR_ST
$w7$	UN	ERR_NST	ERR_ST	ERR_ST
$w8$	PH	PH	PH	PH
$w9$	UN	PH	PH	PH
$w10$	PH	PH	UN	PH
$w11$	UN	PH	PH	PH
$w12$	PH	PH	PH	PH

Table 4.4 – Blade EDL classification for the propagation delay fault model. Timing Violation (TV).

Faulty	SPD		LPD	
Element	wo/ TV	w/ TV	wo/ TV	w/ TV
Latch	-	-	ERR_NST	ERR_ST / ERR_NST
t_{comp}	ERR_ST	ERR_ST	UN	ERR_ST / ERR_NST
t_{TD}	UN	ERR_NST	ERR_ST	ERR_ST
XOR Gate	-	-	ERR_ST	ERR_ST
C-element	-	-	UN	ERR_NST
OR Gate	-	-	UN	ERR_NST
Q-Flop	-	-	-	-
OR Gate (Err1)	-	-	-	-
AND Gate (Err0)	-	-	-	-

The missing items in Table 4.4 represent the faults described in the analysis as *don't care*. Specifically for this classification, the LPD faults in the OR Gate (*Err1*) and the AND Gate (*Err0*) are classified as *don't care*, since the circuit operates as expected, but with lower performance.

This fault classification guides the next steps towards the design for testability of the EDL. In Table 4.5, the three approaches were evaluated in terms of fault coverage of the 32 possible faults (*don't care* faults are not accounted). The functional test is the first alternative, and the fault coverage is 34%, the smallest coverage among the three. The next approach assumes a scan cell at the *Err* signals of each stage to enhance its observability. Another alternative would be to make the Q-Flops of each stage scannable. Using scan cells at the border of EDL and the controller could also help to improve the controllability of the controller. The observed fault coverage for this second approach is 66%. For a fault coverage of 100%, a timing violation generator must be included to fully exercise the EDL. As previously demonstrated, some faults are only observed when the circuit has a timing violation and, in some particular cases, the timing violation must occur at specific moments, such as at the end of the TRW for detecting a LPD in the C-element.

Table 4.5 – Blade EDL fault coverage obtained per test approach. (*) Fault Detected.

Fault		Functional	Scan Chain	TV Gen.
Type	**Wire/ Element**			
SA0	$w1$	*	*	*
	$w2$	*	*	*
	$w3$		*	*
	$w4$			*
	$w5$			*
	$w6$			*
	$w7$			*
	$w8$	*	*	*
	$w9$			*
	$w10$	*	*	*
	$w11$			*
	$w12$	*	*	*
SA1	$w1$	*	*	*
	$w2$	*	*	*
	$w3$		*	*
	$w4$		*	*
	$w5$		*	*
	$w6$		*	*
	$w7$		*	*
	$w8$	*	*	*
	$w9$	*	*	*
	$w10$			*
	$w11$	*	*	*
	$w12$	*	*	*
SPD	t_{comp}		*	*
	t_{TD}			*
LPD	*Latch*		*	*
	t_{comp}			*
	t_{TD}		*	*
	XOR Gate		*	*
	C-element			*
	OR Gate			*
Coverage		34%	66%	100%

4.4 Testable Error Detection Logic

In the previous sections a complete stuck-at and delay fault analysis of Blade's EDL is presented, where faults are detected through a fault classification method based on the functional behavior of the entire EDL when in the presence of an internal fault. In other words, the EDL is being self-tested by associating its functional response to the occurrence of a fault. By concurrently observing its functional behavior it is possible to extract functional patterns with the proposed fault classification. A similar concurrent test approach has been used for a network-on-chip [CHR16] and a processor [WAN06b]. Chrysanthou et al. [CHR16] use a group of lightweight micro-checker modules in a network-on-chip as concurrent hardware assertions, checking for illegal output patterns while the circuit is in normal mode. This approach provides full fault coverage and diagnostic capability.

Wang and Patel [WAN06b] propose the use of a symptom-based error detection approach to detect atypical events that concurrently hint the occurrence of soft errors in a processor. These events trigger a rollback to a safe checkpoint, restoring the processor state.

The previous study demonstrates that a single stuck-at fault might jeopardize the resilience of part of the circuit. Although one of the goals of this work is to rely mostly on the existing circuitry to detect internal faults, it has been observed that to achieve 100% of fault coverage, a successful test method must be able to inject TV. The necessity of producing errors for testing the detection logic was also addressed by Yuan [YUA13], that proposed a scannable version of the DSTB [BOW09]. The scan cell adds controllability over the error signal of each sequential element. With Blade, the idea is to avoid as much as possible additional test circuitry.

Another observation based on the previous fault analysis is that some faults were only detectable by observing the EDL error signals of the next stage, and not just the signals of the faulty EDL. In these cases, when a fault prevents the EDL from detecting a TV, the TV is propagated to a following stage, and this fault ends up detected, which additionally indicates that the previous stage failed to identify the injected TV. However, the data might be logically masked by the following combinational logic before it reaches the next EDS, leading to an undetected fault. These uncertainties motivate the development of a new testable error detection logic (TEDL) architecture.

4.4.1 Proposed Architecture

The proposed TEDL, depicted in Figure 4.8, is designed to add controllability and observability, for test purposes, and to avoid the assumptions and uncertainties of the previous approach. As previously pointed, part of the EDL is only activated when a TV occurs. So, to properly test this part of the EDL, one must be able to control the EDL's input to produce TVs. The original Blade EDS is modified to generate a TV. First, note that the TD is no longer part of the sequential element. This approach allows better compatibility with commercial EDA tools since the tool no longer sees a custom sequential cell, but a standard D latch. This isolated TD approach can also be applied to the original Blade EDS, opening the possibility to use scan cells with automated commercial DfT tools [JUR18a]. The problem with this approach is the positioning of the standard sequential cell and the TD cell. In the placement phase of the design flow, the designer must guarantee that they are placed side by side, and the timing constraints remain the same as in the original EDS design. This can be done using a hierarchical approach that restricts the cell placement [JUR18a] [JUR18b].

The new TD has one new input called *tv*. This signal can be individually controlled at each stage or it can be a global input signal connected to all TDs. In this work, the global approach is applied. When *tv* is activated, it forces the X output of all TDs to be always high, independently of the input *Din*, thus generating a behavior equivalent to a TV detection in all TDs of the circuit. An alternative solution would be to add a two-input OR gate between each TD output and the corresponding C-element input to force a TV, but there would be less controllability and higher area

overhead. Another possible approach is to make the C-element scannable. Scannable C-elements were proposed in the past, such as in [BEE05] and [IWA10]. However, these solutions are not considered since they also present a higher impact in area.

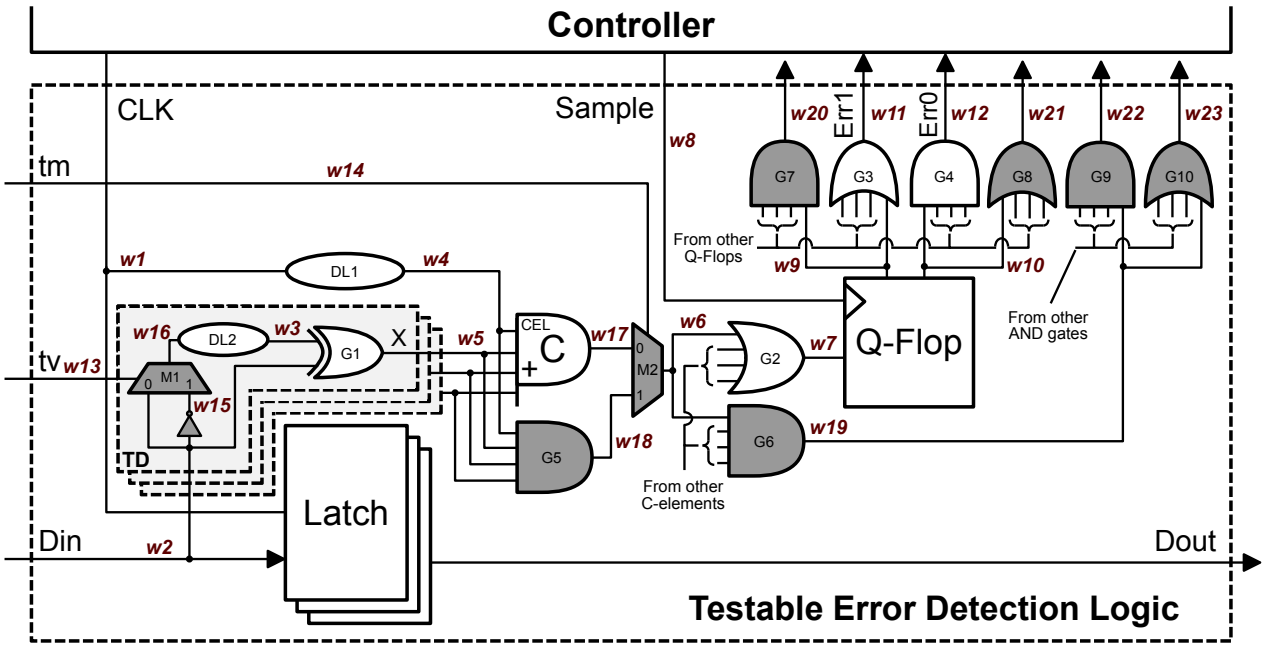


Figure 4.8 – TEDL diagram and labeled wires. The dark gates represent the additional logic included to improve the testability.

The test mode (*tm*) port is another new global input signal. It controls whether the output of the C-element or the output of *G5* is forwarded. This gate and the other gray elements highlighted in Figure 4.8 are concurrent checkers added to detect faults that, otherwise, would not be detected. For instance, an SA0 in *w5* cannot be detected if all *X* signals are at logic level 1, since the C-element output rises if at least one of its inputs plus the *CLK* signal are activated, which means that the fault will be masked and not observed.

It must be clear that the proposed approach has little impact on Blade's timing constraints. The timing overhead of the *M1* multiplexer and the inverter inside the new TD can be compensated in the *DL2* delay line. Except for the *M2* multiplexer, all the other new elements do not create timing overheads compared to the original architecture. The diagnostics of the internal faults can be obtained by observing the outputs *w20*, *w11*, *w12*, *w21*, *w22* and *w23*, and the pattern associated with internal faults. The observability method used to extract the patterns is still open for study. Although the area of scan cells needed to add observability is presented in the experimental results (section 4.4.4), the scan chain was not implemented in the final netlist. Instead, a behavioral description of these cells is used to generate the captured patterns.

4.4.2 TEDL Operational Modes

During simulation, the TEDL must switch between its four operational modes, defined by the state of input signals *tm* and *tv*. The four modes are: normal mode (NM); normal mode with timing violation (NMTV); test mode (TM); and test mode with timing violation (TMTV). In NM, the multiplexer *M1* selects the path that does not pass through the inverter and *M2* selects the *w17* path. In NMTV, *tv* is enabled and the *w15* path is selected, which forces all *X* signals of the pipeline stage to 1. The TM is used basically to detect stuck-at faults in *w5* by selecting the *w18* path. Finally, the TMTV is applied to force all *X* signals to 1 and to pass them through the *G5* gate.

The test procedure with the TEDL requires that the pipeline is full. With Blade, this is done through the asynchronous handshaking protocol. A full pipeline will not accept a new input data request until an output data is acknowledged, and a lockstep control of the circuit is created. It allows the circuit to alternate between a test pattern capture phase and a test pattern shift phase. The full pipeline also avoids that real TVs are generated since all stages are stable. At this moment, the TEDLs can be configured to the different operation modes to produce the test patterns. After each capture phase, a shift phase is also executed, and the faults are identified through the fault classification method described below.

4.4.3 Fault Classification

The fault classification method principle presented in section 4.3.5 is still applied, but it is modified to take into account the different modes of operation and the new signals used as test patterns. A fault is detected whether the output pattern is different from the expected gold pattern defined for each operational mode (Table 4.6). An important assumption of the classification method is that once *tv* is active, all TDs produce a TV, and when it is deactivated, none of the TDs produce a TV. It can be done by making the circuit to operate in a slower mode, which is possible with the approach presented in Chapter 6.

Table 4.6 – TEDL operation modes and their expected outputs.

Mode	Gold Pattern							
	TM	**tv**	**w20**	**w11**	**w12**	**w21**	**w22**	**w23**
NM	0	0	0	0	1	1	0	0
NMTV	0	1	1	1	0	0	1	1
TM	1	0	0	0	1	1	0	0
TMTV	1	1	1	1	0	0	1	1

The six output ports of Table 4.6 also enable some level of diagnostic. For instance, an SA0 at *w6* is detected through *w22* when *tv* is active. Since all other lines of *G2* are at 1, the pattern of *w20*, *w11*, *w12* and *w21*, do not differ from the gold pattern. On the other hand, an

SA0 at *w7* will affect *w20* and *w21*. So, these two faults have distinct patterns, but this does not mean that these are the only faults detected by these patterns. This is also the case for an SA0 at *w17*, which is detected by the same pattern as the SA0 at *w6*. Also, like the previous method, some faults are not detected by a specific pattern, but they halt the pipeline.

One important thing to notice about the fault patterns is that they cover the additional gates and test signals (*tm* and *tv*). Also, signals *tm* and *tv* are considered in the fault pattern generation. For instance, assume that there is a SA0 *w7*. If a TV occurs, this fault prevents the Q-Flop from registering the TV, and the fault cannot be detected in the NM. Either the NMTV or the TMTV modes put all *G2* inputs at 1 and it is expected that an error is flagged for all internal lines. However, since *w7* is SA0, *w20* remains at 0, indicating the existence of a fault. Another example is *w19* SA1. This wire is expected to be 1 only when *tv* is active, so in NM or TM *w22* and *w23* must be at 0. Since *w19* is SA1, *w23* rises, while *w22* remains at 0 due to other non faulty *G6* gates.

4.4.4 TEDL Case Study: Plasma CPU

The Blade automated flow described in [HAN15] is modified to evaluate fault coverage and area overhead of the proposed testable architecture. This flow uses industry standard tools, including DesignCompiler and PrimeTime from Synopsys and NC-Sim from Cadence. As described in Section 3.1.7, the flow converts a single clocked synchronous RTL design into a Blade netlist. It consists of Tcl and shell scripts, a library of custom cells and a Verilog co-simulation environment. The *Blade Conversion* step was modified to include the proposed TEDL. The same case study of [HAN15], a 3-stage version of MIPS OpenCore CPU called Plasma [PLA14], targeting a 28nm FDSOI technology, is implemented to compare the results.

The co-simulation environment implements a testbench where a stream of inputs is forked to both the synchronous and Blade netlists, and the stream of outputs are compared to validate the implementation. The fault simulation environment described in section 4.3.1 is incorporated into this co-simulation environment. The fault simulation environment has as input parameters: a list of fault points to inject faults, that consists of all the labeled wires presented in Figure 4.8; the gold patterns for each one of the operation modes, shown in Table 4.6; the fault patterns that indicate the existence of an internal fault; and the list of faults that can be related to those fault patterns (diagnostic information).

4.4.5 Fault Coverage

A series of fault simulations were executed to validate the fault coverage of the TEDL, and the modifications made to the flow and co-simulation environment. Each fault point was simulated for a single SA0 and single SA1, alternating between the four test configurations in Table 4.6 to

detect the injected faults. The simulation results showed that 100% of the stuck-at faults inside the TEDL are detectable, while with the original EDL, a little over 30% of the stuck-at faults are detected. There are three fault points in particular inside the TEDL that depend on a transition in *Din* to produce a pattern that can be related to a fault. These are: *w15*; *w16*; and *w3*. Since no control over *Din* is assumed, the source code executing on the Plasma CPU must be able to generate these transitions. Also, since these three nets are internal to the TD cell, they would be collapsed from the fault list, and the ATPG would ignore them.

4.4.6 Area Comparison

The area comparison of both implementations is shown in Table 4.7 and Table 4.8. Table 4.7 presents the EDL and TEDL number of elements and its corresponding area. Both Plasma implementations have the 238 TDs divided into two groups, each one with its controller.

Table 4.7 – EDL vs. TEDL area overhead with the Plasma CPU case study.

Cell	**EDL**		**TEDL**	
	N	Area μm^2	N	Area μm^2
C-element	80	189.055	80	189.055
Q-Flop	20	91.280	20	91.280
Latch	238	427.258	238	427,258
TD	238	466.099	238	893.357
OR G2	20	52.224	20	52.224
OR G3	2	7.507	2	7.507
AND G4	2	7.507	2	7.507
AND G5	-	-	80	112.934
AND G6	-	-	20	55.488
AND G7	-	-	2	7.507
OR G8	-	-	2	7.507
AND G9	-	-	2	7.507
OR G10	-	-	2	7.507
MUX M2	-	-	80	117.504
MUX-D	-	-	14	79.968
Total	362	1240.930	564	1920.606
Area Overhead				54.77%

Table 4.8 – Comparison of Plasma-EDL vs Plasma-TEDL in terms of area (μm^2).

	Plasma-EDL	**Plasma-TEDL**
Combinational area	7095.28324	7829.35683
Buf/Inv area	608.89921	687.23522
Noncombinational area	7860.69133	7860.69133
Macro/Black Box area	228.57500	228.57500
Net Interconnect area	undefined	undefined
Total cell area	15184.54957	15918.62316
Area Overhead		4.61%

Unlike the original EDL architecture, the TD in the TEDL is not incorporated into the sequential element. To compare the original EDL architecture with the TEDL, instead of using a custom cell, the original EDS is also decomposed in standard cells. The resulting area for the original TD and the new TD is 1.958 µm^2 and 3.754 µm^2 respectively. In Table 4.7 the 893.357 µm^2 TD area refers to total number of TDs ($3.754 * 238$). As previously mentioned, the experiment considers that test patterns are captured through scan cells connected to the outputs *w20*, *w11*, *w12*, *w21*, *w22* and *w23* of each stage, but any observability technique can be used. For this work area of a MUX-D cell is used to generate the area overheads.

As the results show, the TEDL imposed an increment of 54.77% in area, when compared to the original EDL in the Plasma design (Table 4.7). Note that this experiment is intended to examine the TEDL using the same scenario of Blade's proposal [HAN15], but Blade allows a trade-off between critical paths covered by EDLs and the area overhead of its implementation. In this particular setup, out of total 529 latches, 238 are modified. When looking at the overall area overhead, with almost half of the latches modified, the Plasma-TEDL shows a 4.61% area overhead when compared to Plasma-EDL, and the TEDL affects mainly the combinational area of the designs (Table 4.8). For future improvements, the possibility of incorporating *M2* multiplexer and *G5* AND gate to the C-element is a possibility, creating a custom test C-element cell that would have the same behaviour, but less area. Note that custom cells usage in this context do not affect the fault analysis, as long as their functional behavior is kept. Finally, the TD is implemented using standard cells from the 28nm FDSOI library. Further area optimization is possible with the development of a custom cell.

5. CONTROLLER TESTABILITY

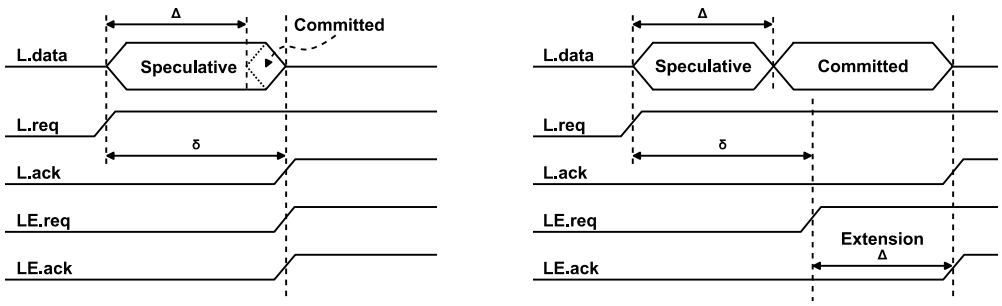
In Section 3.3, one of the first works to evaluate the testability of asynchronous circuits was presented by Pagey [PAG92]. Pagey demonstrates that despite the challenges of testing asynchronous circuits, testing the control part of asynchronous BD designs is an easy task, since the controller tends to halt the entire circuit when it has a stuck-at fault. This statement is true for Micropipeline [SUT89], but for other approaches, such as the Mousetrap [SIN07], this is not true [SHI05]. In Chapter 4, the testability of the EDL is sometimes dependent on the correctness of the controller, since the EDL testing relied on the controller halting to detect some faults. For instance, if the *err1* or *err0* signals remain stuck at some value, the controller will not be able to complete the protocol communication. For these particular signals, it was demonstrated in Section 4.3 that the controller halts. However, more signals must be evaluated to make sure that the Blade controller halts for every stuck-at fault, including the handshake signals and the internal ones.

This Chapter presents contribution 3 (see Section 1.2), where the Blade controller testability is evaluated to check whether the controller halts for every stuck-at fault. Some modifications are suggested for the original controller, and a new controller is proposed to improve even more the testability. Contribution 4 is presented at the end of this Chapter, which consists of a test method for testing the delay line inside the controller (Δ) and the data path delay line (δ).

5.1 Blade Controller

The Blade controller implements a new form of asynchronous handshaking protocol, called *speculative handshaking* [HAN15]. The controller implements two BD channels that operate with a 2-phase handshake protocol. Channel L/R is used as input and output control respectively, thus controlling data propagation from one register to another. The request signals (L.req and R.req) are associated with the δ delay line. The additional channel LE/RE is the error channel used for recovering from a timing violation, so no data transfer is involved. This new protocol reduces the timing violation recovery overhead when compared to synchronous resilience approaches discussed in Chapter 3.

The request signal (L.req) is speculatively asserted assuming that the δ delay is long enough to propagate data through the combinational logic to the next stage without timing violations. The LE channel is used to check if the assumption was correct and a timing violation was not detected by the previous stage. In this case, the previous stage controls how long the current stage needs to wait for valid data input in case of timing violations. Figure 5.1(a) illustrates a situation where no timing violation occurs. Note that the acknowledgment of the additional channel (LE.ack) occurs right after LE.req. On the other hand, Figure 5.1(b) shows when a timing violation is detected by the previous stage. Thus the acknowledgment is extended by Δ, allowing enough time for the correct data to propagate through the data path.



(a) Without extension (b) With extension

Figure 5.1 – Blade speculative handshaking protocol [HAN15].

The implementation is divided into three interacting Burst-Mode state machines [FUH95] and synthesized using the tool 3D [YUN92]. Figure 5.2 presents the source code that describes one of the finite state machines used in Blade. The code has a list of inputs and outputs, and how one of these signals must behave in each state. In this example, all signals are initially at 0. For the state machine to transition from state **0** to state **1** the input *Lreq* must rise ('+' syntax indicates a rise and '−' a fall), and when it occurs, the output *LEreq* must immediately rise too. Note that only the signals specified in each state are allowed to transition, and the 3D tool guarantees this behavior.

```
input Lreq 0
input LEack 0
input goL 0

output LEreq 0
output goR 0
output Lack 0

0 1 Lreq+           |    LEreq+
1 2 LEack+          |    goR+
2 3 goL+            |    Lack+
3 4 Lreq-           |    LEreq-
4 5 LEack-          |    goR-
5 0 goL-            |    Lack-
```

Figure 5.2 – Description of Blade's Burst-Mode controller using 3D syntax.

The diagram of Figure 5.3 shows the three state machines used for pipeline stages with EDLs. The intermediate signals *goL*, *goR*, and *goD* are communication signals between these state machines, and signals *delay*, *edi*, and *edo* are used to add the Δ delay line into the controller. In the original implementation of Blade's controller the Δ delay line is duplicated between *CLK* $\rightarrow$ *delay* and *edo* $\rightarrow$ *edi*, and consolidating these delay lines is left for future work. As will be further described, this merge is mandatory for the delay line testing proposed in Section 5.3.

As described by the authors of Blade [HAN15], the controller design is extended to a token version, which generates an output request right after the reset, and other two, simplified versions, for stages without EDL, creating four distinct Blade controllers. As a reminder, note that the Blade flow does not replace all latches by an EDS, so a simplified controller is needed to control these remaining latches. The 3D tool generates a sum-of-products description for each state machine, which was manually optimized and mapped to an FDSOI 28nm library of gates. The optimization is

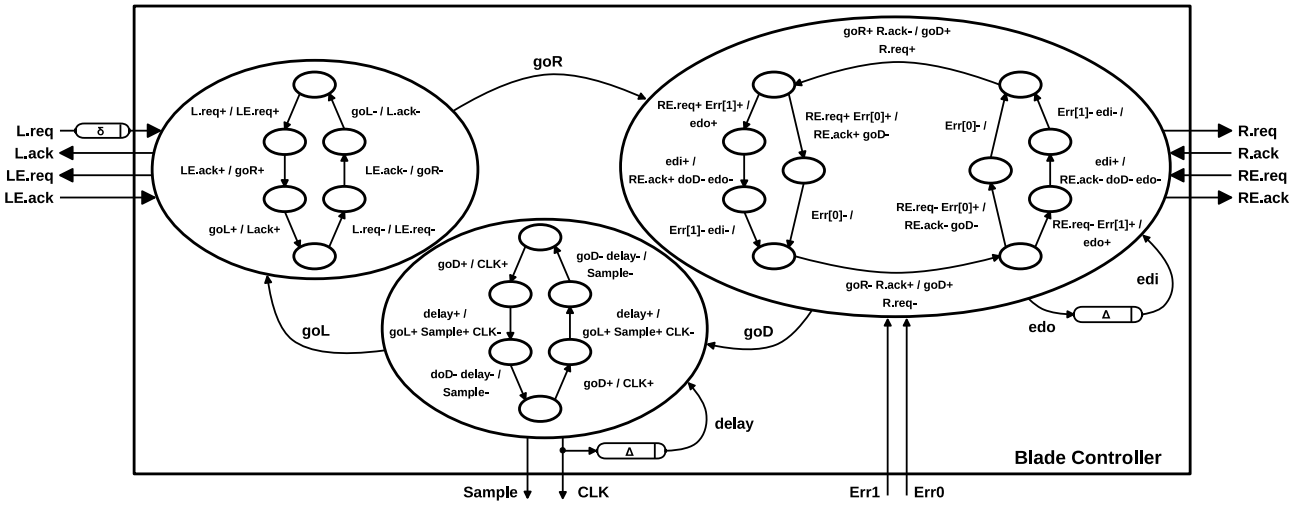


Figure 5.3 – Burst-mode state machines for the Blade controller [HAN15]

possible because the 3D tool does not implement the state machines together, and does not know for instance, that the *goR* behaves just like the *LE.ack*. It is important to notice that this must be done with caution by the designer. Otherwise, the generated circuit will not behave like the 3D input specification.

5.1.1 Stuck-at Fault Analysis

Figure 5.4 illustrates the gate level implementation of the original Blade controller for stages with EDL. For simplicity, the reset circuitry was omitted. The *Lack* circuit will be used as an example to understand how this implementation works. In this circuit a 2-input OR gate groups a 2-input AND gate with the *delay* signal.

Assuming that reset sets all signals initially to 0, in order to have a transition at *Sample*, one of the OR inputs must transition. For this circuit it will only happen when a transition at *delay* occurs, since the *Sample* feedback loop prevents *goD* to act. Now *Sample* remains at 1 as long as *delay* is at 1 or *goD* rises before *delay* fall. For a transition to 0 to take place at *Sample*, *delay* must return to 0 and *goD* must return to 0 if a rise transition had occurred.

Based on the fault analysis of Section 4.3.2, a stuck-at fault in the *Sample* signal prevents the controller from continuing with communication, causing the circuit to halt. For this circuit in particular, an SA0 at *delay* will prevent the *Sample* from ever going up, and an SA1 at *delay* causes an immediate activation of *Sample*, which can translate into a protocol violation, but also into a halt. Since the controller expects *Err1* and *Err0* to return to 0 before continuing with communication, an SA1 prevents this from happening. Looking at *goD*, an SA1 does not affect the first rise of *Sample*, but does not allow it either to ever return to 0, and a halt occurs. However, an SA0 allows the *Sample* to rise and fall following transitions of *delay*, not directly causing a system halt. Finally, an SA1 in the internal signal *and20* produces the same behavior as an SA1 at *delay*, and an SA0 presents the same behavior as an SA0 at *goD*.

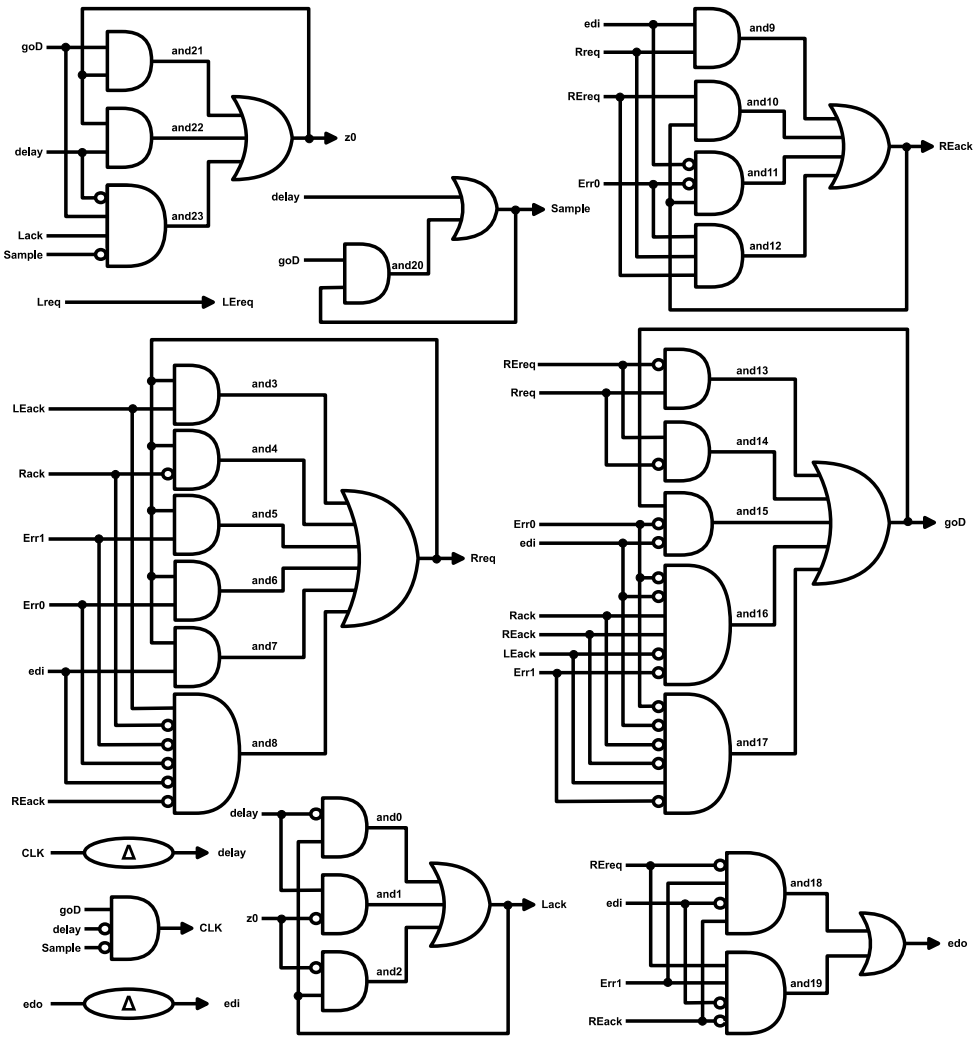


Figure 5.4 – Gate level implementation of the Burst-Mode Blade controller.

5.1.2 Fault Coverage

Although the analysis of this single part of the controller suggests that the Blade controller does not halt for all stuck-at faults, it is necessary to evaluate the entire controller at once, because *goD* is also used in other parts of the circuit, and these other parts may cause a system halt. For this fault analysis, the fault simulation environment presented in Section 4.3.1 is also applied, where single stuck-at faults are injected at all primary inputs and internal signals of the controller. Internal signals include signals resulting from inverters. For example, the *and23* signal can be affected by a stuck-at fault placed before or after the inverters placed at the AND gate inputs. The fault simulations also consider that TV can be generated to stimulate the parts of the controller that handle captured violations.

From the gate level description of Figure 5.4, a total of 51 fault points are extracted for simulation, resulting in 102 possible faults when considering SA0 and SA1 for each point. The fault simulation considered detectable all faults that caused a system halt. The list of undetectable faults is presented in Table 5.1. From the 102 simulated faults, 17 did not produce a system halt,

Table 5.1 – List of undetected faults in the Burst-Mode controller.

Signal	Undetected Fault
and0	SA0
and2	SA0
and3	SA0
and4	SA0
and5	SA0
and6	SA0
and7	SA0
and11	SA0
and15	SA0
and16	SA0
and17	SA0
and21	SA0
and22	SA0
inv_leack	SA0
inv_leack	SA1
inv_err1	SA1
inv_rereq	SA1

an 83.3% fault coverage. Note that most of the undetected faults are SA0, differently from the individual analysis of *goD*. In this last case, the AND gate that generates the *CLK* signal is affected by the *goD* SA0, and the controller halts. The low fault coverage for stuck-at suggests that another designs style should be considered. Thus, delay fault testing of the Burst-Mode controller was not addressed.

5.2 Click-based Controller

The previous fault analysis of Blade's Burst-Mode controller demonstrated that, as suggested in [SIN07], not all asynchronous designs are prone to a system halt when the control part is affected by a stuck-at fault. Therefore, the uncertainty about the existence of faults inside the controller makes it impossible to apply the behavioral fault classification method proposed for testing the EDL (see Section 4.3.5), since it assumes that the controller does not present any faults associated with it.

Burst-mode controllers, in general, are not easily tested. One of the reasons is the presence of feedback loops, which are cannot be treated easily by commercial ATPG tools. Feedback loops are usually broken by the DfT tool to generate test patterns. The main problem of this approach is related to a limitation of the original Blade flow, where the controller must be manually mapped to the desired technology, not allowing any new tool optimization, that otherwise could modify the circuit behaviour generated by the 3D tool.

Another alternative is to modify the state machines descriptions to produce a circuit that presents a halt for every possible stuck-at fault. The problem here is the complexity of the Blade handshaking protocol. Besides, the two BD channels implemented in Blade's controller are 2-phase, which on the one hand are faster than the 4-phase, but on the other hand present a higher complexity

and leads to a larger circuit. For a 2-phase implementation, a control signal at logic values 0 or 1 represents two different communications, while with 4-phase a return to 0 is mandatory for a new communication to start. The 2-phase adds redundancies to the circuit to accept the two possible state transitions. For instance, Figure 5.3 shows that the state machine on the right, must accept *R.ack* rising to rise *goD* and activating the state machine in the middle, but *R.ack* falling must also produce the same transition in *goD*. So, a new Burst-Mode implementation would present an increase in complexity and consequently more area overhead, but more importantly, it would still not guarantee that all faults would halt the controller.

In the just explained context, a new controller is proposed. The new controller is implemented using the Click template [PEE10]. Click is a 2-phase BD design that only uses edge-triggered flip-flops in the control circuit. One of the advantages of this approach is that the design flow with standard (third-party) EDA tools is facilitated, especially for physical synthesis and statical timing analysis. Another advantage is that the authors already demonstrated how the controller could become testable through a scan implementation. This approach for the Blade controller is not entirely new since Sharp [WAU17] already proposed the use of Click-based controllers. However, testability is not addressed in that work, and their implementation is more focused on performance improvements with a new handshake protocol.

The Click-based controller proposed here is compatible with the original Blade speculative handshaking protocol, using two 2-phase BD channels (L/R and LE/RE) to communicate with other stages. In this new implementation, the two Δ delay lines are consolidated into a single one, as suggested in [HAN15]. Sharp, on the other hand, has four different delay lines. The single delay line implementation is believed to be a better approach when the testability of these delay elements are taken into account. This is further discussed in Section 5.3.

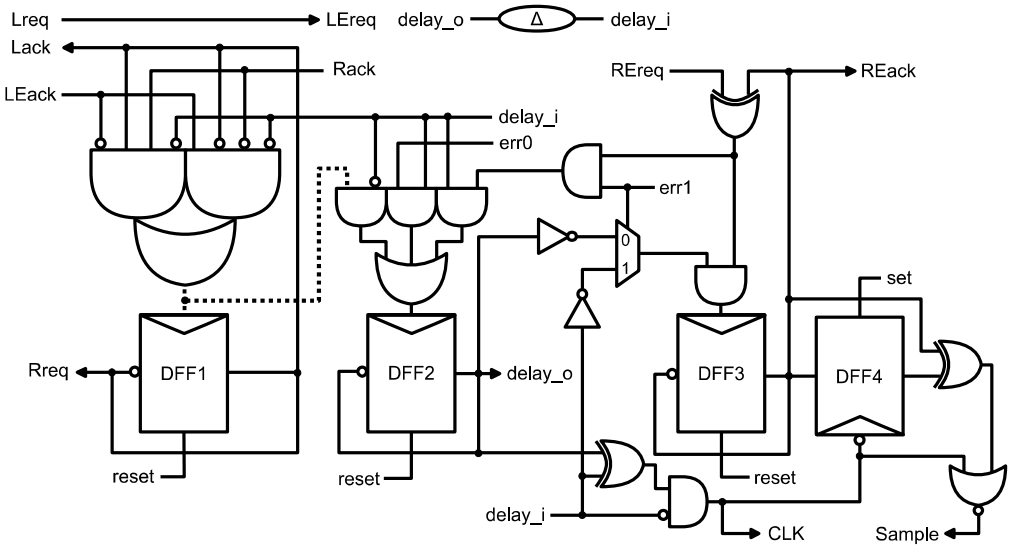


Figure 5.5 – Click-based controller design for the EDL stages.

Like the original Blade proposal, there are also four versions of the proposed Click-based design. The controller for EDL stages, illustrated in Figure 5.5, a token version to generate an

output request after reset, and a simplified version for stages without EDL and its equivalent token version. These last three implementations are in Appendix A. The speculative handshake protocol remains the same as the original Burst-Mode implementation. The difference between the two is how internal signals are activated to control the communication between controllers. Figure 5.6 shows the Click controller timing diagram for two data transactions, the first without extension and the second one with extension. Except for the propagation time of δ and Δ delay lines, no other propagating delays are represented.

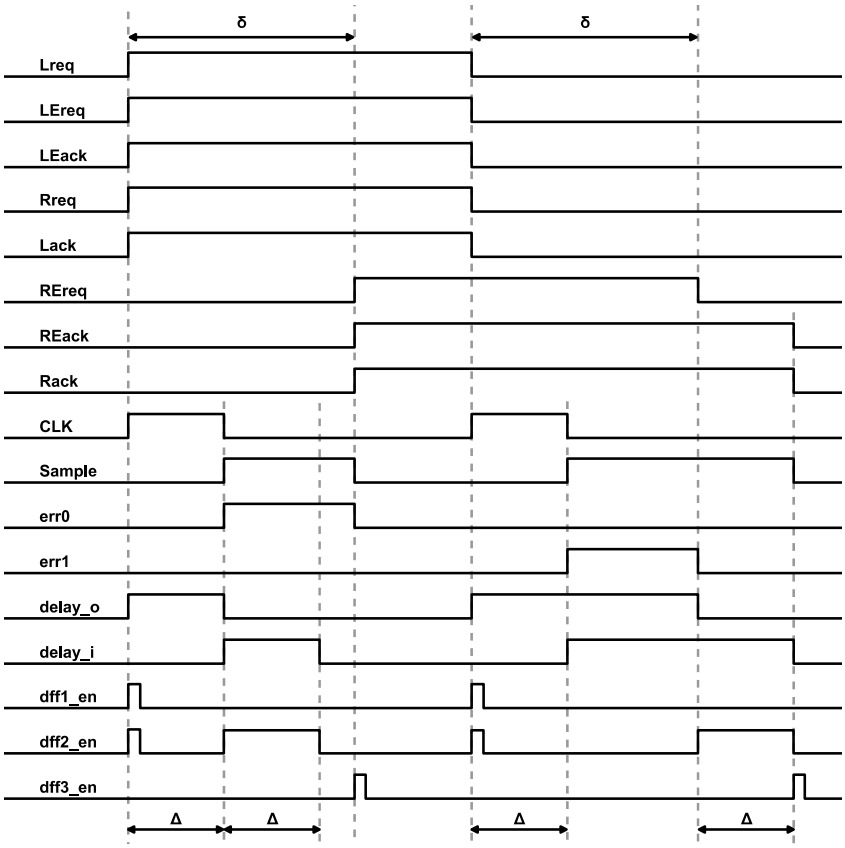


Figure 5.6 – Speculative handshake protocol timing diagram with the Click-based controller.

After reset, all control signals are at logic level 0. The controller connected to the left channel sends a request (*Lreq*) and *LEreq* is flagged back. The controller in the left channel acknowledges that a valid data was propagated. This enables the DFF1 flop (*dff1_en*), which rises the request for the controller connected to the right channel (*Rreq*). The *Rreq* signal is fed back as *Lack*, which ends the communication with the left controller. *Rreq* feedback also resets the *dff1_en*. However, the *dff1_en* is also used to generate the *dff2_en*. This creates a timing constraint that must be satisfied during synthesis, in which the propagation time from *dff1_en* rising (Figure 5.5 dotted line) to *dff2_en* rising must be higher then the propagation time from *dff1_en* rising to *dff1_en* falling.

The DFF2 flop activates the Δ delay line (*delay_o*) and consequently the *CLK* signal through the XOR gate with *delay_i* and *delay_o* as inputs. The *CLK* stays high for Δ time, which in other words can be translated into the TRW. Once *delay_o* is propagated through the delay line, *delay_i* rises and deactivates *CLK*. The falling edge of *CLK* enables the DFF4 flop, that

activates *Sample*. At the same time *delay_i* deactivates the *dff2_en* and blocks the *dff1_en* from being activated again before ending the communication with the right channel. At this moment the controller stalls waiting for one of the error signals to rise. In the first communication of the timing diagram no timing violation is detected by the EDL, so *err0* rises and *dff2_en* rises again, which deactivates *delay_o*. Now the controller waits for the *REreq* from the controller at the right channel. Once the *REreq* rises the DFF3 flop is enabled (*dff3_en*), and the *REack* is sent to the right channel along with *Sample* deactivation. When the acknowledgment (*Rack*) from the right channel rises, the controller is able to accept a new incoming data.

The timing diagram of Figure 5.6 also presents how the controller signals behave when a timing violation is detected by the EDL. Initially the communication starts the same way as in the previous description. The difference starts when the controller rises *Sample* and the *err1* signal rises too. Now *dff2_en* is blocked until the controller connected to the right channel changes the *REreq* state. Only after *REreq* changes *dff2_en* rises, deactivating *delay_o*. The *err1* also selects the MUX input connected to *delay_i*. Since *delay_i* at this moment is still high, and *dff3_en* is blocked until *delay_i* falls. Note that the single delay line serves two purposes: working as 2-phase protocol, where a rising transition propagated through the delay line controls the high phase of *CLK*; and the fall transition controls the delay extension needed for the valid data to arrive at the next stage. Finally, *dff3_en* is activated, and as previously, *REack* rises and *Sample* is deactivated. After *delay_i* falls and *Rack* is sent by the controller to the channel, a new communication can be started.

The use of a single delay line does not affect the timing constraints from the original Burst-Mode controller. As illustrated in Figure 5.3, the state machines connected to the delay lines expect that at some point both *delay* and *edi* signals return to zero before starting a new communication. This behavior is achieved in the Click-based design by the *delay_i* signal, that blocks different parts of the circuit in its high phase.

5.2.1 Fault Coverage

The same simulation environment presented in Section 4.3.1 is used to extract stuck-at fault coverage results for the new Click-based controller. The Burst-Mode controller replacement is transparent for the rest of the circuit since it has the same interface and communication protocol. Only single stuck-at faults are injected at all primary inputs and internal signals of the controller, including the inverted signals. Also, TV is simulated to stimulate the parts that handle the error recovery. The analysis of delay faults is left as future work. However, the delay lines test method further described in Section 5.3 may be extended for this purpose, where controller's internal delays are incorporated into the delay lines paths.

In the Click-based controller, there are 35 fault points and a total of 70 possible faults when considering SA0 and SA1 for each point. The experiments reported that the achieved fault coverage achieved 100%. However, it is important to explain that not all faults caused a system halt

but instead affected the *CLK* generation directly. For instance, an SA0 at *delay_o* does not block the communication protocol, because DFF3 can still be enabled once *REreq* arrives. However, if *CLK* never rises, data is not propagated through the data path, and the fault can still be detected. From the total possible faults, 11 fall into this situation.

5.2.2 Area Comparison

In order to compare the Click design with the Burst-Mode design, a netlist targeting 28nm FDSOI technology is generated for the different versions of each design. The Burst-Mode controllers are manually mapped to the target technology, while the Click controllers are automatically mapped from an RTL design using DesignCompiler from Synopsys. This automatic mapping is an important advantage of this approach since the synthesis tool can freely optimize the circuit following the design constraints. Moreover, DfT tools can be used for testing the circuit, where existing feedback loops are already broken by the flip-flops. The flip-flops can also be automatically replaced by scan cells as demonstrated by the proponents of Click [PEE10] to improve controllability and observability, but this is left for future work.

Table 5.2 shows the area comparison for the different controllers. The Click Controller version presents an area overhead of 21.10% when compared to the Burst-Mode equivalent, which is the worst case among all comparison. On the other hand, the Click Token Controller version has the same area as the Burst-Mode, and in the case of the Click Token Controller EDL, a reduction in area is observed, with 8.38% less area than the Burst-Mode. Note that these results do not account for the difference in the number of Δ delay lines.

Table 5.2 – Comparison of Burst-Mode controller vs Click-based controllers in terms of area.

	Burst-Mode	**Click**	**Overhead**
Controller	14.0352	17.7888	21.10%
Token Controller	18.4416	18.4416	0.00%
Controller EDL	24.6432	26.4384	6.79%
Token Controller EDL	29.5392	27.2544	-8,38%

The size of the delay lines varies depending on the RTL design and the configuration parameters of the Blade flow. The Plasma CPU case study presented in Section 4.4.4 is used to extract these results. Disregarding the other elements of the final netlist, that are the same for both designs, the total delay lines area for the Burst-Mode and the Click implementation corresponds to 44.0640 μm^2 and 22.0320 μm^2 respectively, while Burst-Mode controllers area is 68.2176 μm^2 and Click controllers area is 71.4816 μm^2. The total area for the controllers differs from Table 5.2 because the final netlist does not implement any Token Controller, only the other three are used. Therefore, when considering the delay lines area, the Click design is 20.07% lower than the Burst-Mode. One can argue that the same approach of removing the additional delay line could be applied to the Burst-Mode, but circuit complexity would increase, and area comparison would be even more favorable to the Click-based approach.

5.3 Delay Lines Testability

Asynchronous bundled-data designs, in general, must incorporate delay lines in the control signal to compensate for data path propagation through computational logic. The correctness of delay lines is crucial for the system to work correctly. One problem with bundle-data designs is that, just as in the traditional synchronous designs, they suffer from PVT variations. Thus timing margins must also be incorporated. Although timing resilient architectures alleviate the added timing margins, variability can affect the entire circuit, including the delay lines in Blade.

In bundled-data designs, the circuit is faulty if the data path became slower or if the delay lines became faster than the timing specifications. In the case where the data path is slower, there are some alternatives before discarding the chip. One of them is presented in the next Chapter, but this approach relies on delay lines properly tested, with their delays following the design constraints. Thus testing the delay lines is necessary.

In the literature, there are not many works that address the testability of delay lines in bundled-data designs. One of the few works that get near to solve the problem of testing delay lines in bundled-data designs is presented by Sato [SAT15]. However, the proposed method does not take delay measurements of the delay lines. It measures the delay of the combinational block. Then the configurable delay lines are set to a value that is long enough to compensate for the data path delay. The two-pattern test is used to activate data paths for delay measuring, where the first pattern is set by the scan mode and the second pattern is the response of the first pattern from the combinational logic. Timing information is captured by a time to digital Converter (TDC) and shifted out through a scan chain implemented inside the TDC.

The test method proposed by Sato could be adapted for measuring the delay of the combinational blocks in a circuit implemented with Blade since both δ and Δ delay lines are proposed initially as reconfigurable delay lines. However, making the data path scannable to apply the two-pattern test would incur in a considerable area overhead. Moreover, process variability can affect the delay lines, and the delay time may not be as the designed timing constraints. If variability affects the data path, the EDL should be able to detect the timing violations, but if variability affects the delay lines, the circuit cannot rely on the timing violations flagged by the EDL, and the circuit must be discarded. Therefore, the proposed method for testing Blade's delay lines consists of an offline test method for measuring the propagation delay of the delay lines of the manufactured circuit. This test must be executed before delay testing of other structures, such as the combinational logic.

5.3.1 Proposed Method

The proposed method aims to use only primary inputs and outputs to take delay measurements, which include the handshake protocol pins (L/R channel and LE/RE channel) and an additional pin that serves as observation point for the EDL *err1* signals. Besides that, a scan chain

to add controllability over the Q-Flops state is assumed. As previously discussed, a scannable data path is undesired for testing Blade due to high area overheads, but instead, the proposed method assumes the replacement of Q-Flops for Scan Q-Flops (SQF), which represent a smaller number of scan elements than a full scan. For instance, in the case study presented in section 4.4.4, there are 238 latches and only 20 Q-Flops. The SQF is mandatory for the proposed test method since the Q-Flop acts as a metastability filter, and there is an unbounded time for the metastability to be resolved inside the Q-Flop filter and settle the correct output error signal. With the SQF, this time uncertainty no longer exists, and correct timing assumptions can be made for the proposed test method. The SQF cell development was addressed by Juracy et al. [JUR18b].

5.3.2 Test Architecture

Figure 5.7 shows a simplified 3-stage pipeline example of the architecture. In this scenario, the left channels of C0 are connected to primary inputs and outputs, and the right channels of C2 are connected the primary inputs and outputs. There is a single SQF representation for each stage, but in a real circuit implementation, this number increases, and the error signals are grouped before the controller, see Section 4.1. Through the proposed scan chain *err1* and *err0* can be forced to a controlled state immediately after *Sample* rises. Internal *err1* signals are grouped with an OR gate, and its output is mapped to a primary output pin called *Error1*. This design allows delay measurement of the *err1* rising moment for the different stages. In this case, it is assumed that only *ScanQflop1* (Figure 5.7) is configured to raise its *err1* output signal, and all the others to rise only their *err0* signal. For this example, the rising edge observed at the primary output corresponds to the *err1* connected to controller C1, assuming no fault at the OR gate.

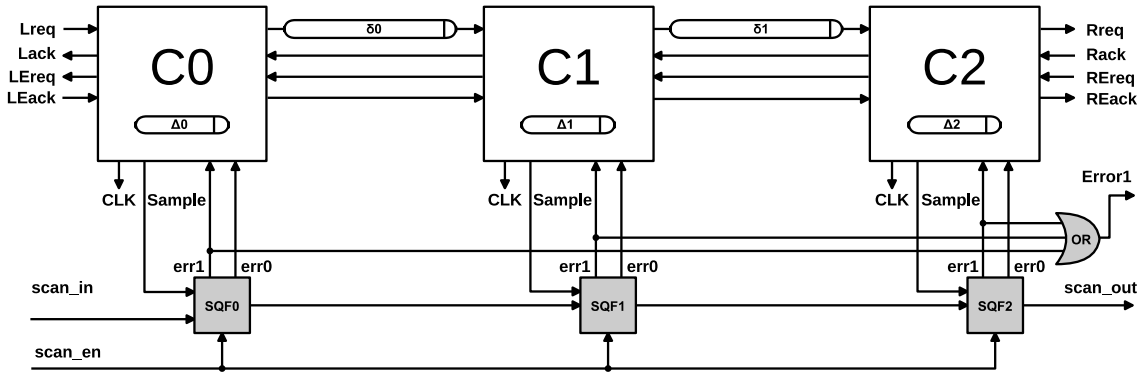


Figure 5.7 – Delay lines test scenario with Blade design.

A requirement for testing Blade's delay lines with the proposed method is that the Δ delay line inside the controller is consolidated into a single delay element. Thus, the Click-based controller presented in Section 5.2 is used. Nevertheless, the proposed test method is compatible with the Burst-Mode controller, as long as it is also modified to implement only one Δ delay line. For Sharp [WAU17], the higher number of internal delay lines may suggest that the test principle cannot be used. However, further analyzes are required before completely discarding the proposed

test method for testing Sharp, since the handshake protocol is different, and the time measurements could be derived from other pins and internal observation points. Accordingly, this is left for future work, along with an extension to consider forks and joints.

5.3.3 Test Procedure

To understand the test procedure, some important aspects about the operation of the controller and the implemented communication protocol must be clear, and the timing diagram of Figure 5.6 can be used as the reference for the following explanations. The controller's internal delays are assumed to be compensated into the delay lines design, and wire delays are assumed to be so small compared to the delay elements that they can be neglected. So, a request arriving at the primary input *Lreq* is immediately forwarded from the left channel to the right channel connected to C0. This means that if no timing violations are flagged (only *err0* rises), and the pipeline is empty, the time from *Lreq* to *Rreq* is equal to the sum of all δ delay lines propagation ($\delta 0$, $\delta 1$ and $\delta 2$). Another thing to notice is that at the same time that *Lreq* is flagged, internally the *CLK* signal rises and the Δ delay line is activated.

After the high phase of *CLK* the *Sample* signal rises, and *err0* or *err1* are immediately flagged, which is possible with the use of the SQF previously configured through the scan chain. In this case it is possible to assume that the time from *Lreq* to *err1* or *err0* is equal to the Δ delay line propagation. For this test method, *err1* is used as the observation point. The reason for this choice is that during the different steps of the test, one stage at a time has its error signal observed through the OR gate. If *err0* were chosen, only the SQF in the observable stage would be configured to propagate the *err0*, and all the other would activate the *err1*, which would activate the protocol extension phase, thus modifying the delay measurements. Choosing *err1* means that only one stage will extend the delay, and the delay measurements will detect the additional time that corresponds to the Δ delay extension.

One last thing to mention is that the *REack* is the protocol signal that acknowledges that all delay lines ended their signal propagation, regardless of which error signal was flagged, as shown in Figure 5.6. Now it is possible to describe the test steps and the equations used to calculate the propagation delay of all delay lines in a circuit implemented with Blade. Between each step, the circuit goes through a complete reset.

- **Step 1:** The first delay measurement corresponds to the sum of all δ delays. All SQFs are configured to raise their *err0* output signal and the communication is started at the left channel. Equation 5.1 defines that for a given pipeline with N stages, the sum of all δ delays is the difference time between the first *Lreq* transition (T_{Lreq}) and the first *Rreq* transition (T_{Rreq}) that arrives at the end of the pipeline.

$$\sum_{i=0}^{n=N-2} \delta_i = T_{Rreq} - T_{Lreq} \tag{5.1}$$

- **Step 2:** This step is repeated for each controller to take the delay measurement of the internal Δ delay line. For this test only the SQF connected to the target controller is configured to raise its *err1* output signal. By forcing the delay extension, the pipeline propagation time is increased by the $Delta$ delay. Equation 5.2 is used to calculate the Δ delay of each controller. The difference time between the first *Lreq* transition (T_{Lreq}) and the first *REack* transition (T_{REack}) that arrives at the end of the pipeline, minus the sum of all δ delays calculated in the previous step, is equal to the Δ delay of the target controller i.

$$\Delta_i = T_{REack} - T_{Lreq} - (\sum_{i=0}^{n=N-2} \delta_i) \text{ , for } err1_i = 1 \tag{5.2}$$

- **Step 3:** The last step is used for measuring the δ delay lines between controllers. This step is repeated for each δ delay line. For this test the SQF connected to the controller that follows the target delay line is configured to rise the *err1* signal, and the transition observed in *Error1* corresponds to this SQF. Equation 5.3 is used to calculate the δ delay of each controller. The target δ delay i is given by the difference time between the first *Lreq* transition (T_{Lreq}) and the first *Error1* transition (T_{Error1}), minus the sum of δ delays previously calculated in this current step, minus the Δ delay of the controller that follows the target delay line ($i+1$). This last subtraction accounts for the fact that the $i+1$ *err1* only rises after the high phase of *CLK*, which is controlled by the Δ_{i+1}.

$$\delta_i = T_{Error1} - T_{Lreq} - (\sum_{k=0}^{n=i-1} \delta_k) - \Delta_{i+1} \text{ , for } err1_{i+1} = 1 \tag{5.3}$$

5.3.4 Test Simulation Example

The same simulation environment presented in Section 4.3.1 is used. For each step, a new simulation is started, and the testbench generates a log file with all primary inputs and outputs transition times observed during the simulation. A post-processing script reads the log file to extract the required time transitions needed for each step calculation. SQFs configurations are applied through force commands during simulation, thus emulating the scan chain.

For the test scenario of Figure 5.7, the delays are: $\Delta_0 = 0.66ns$; $\Delta_1 = 0.54ns$; $\Delta_2 = 0.86ns$; $\delta_0 = 1.54ns$; $\delta_1 = 1.80ns$. The SQFs configuration is represented by a 3-bit vector $<000>$, one for each SQF, where 1 means that the *err1* must rise and 0 the *err0* must rise. In the first step, SQF vector is configured to $<000>$. After reset, the first *Lreq* transitions occurs at $50.00ns$ and *Rreq* at $53.34ns$. The result of Equation 5.1 is calculated and the result is $3.34ns$.

Now the simulation environment moves to the second step, where three simulations are executed. For all these simulations T_{Lreq} is $50.00ns$. In the first simulation, the SQF vector is configured as $<100>$, and the first *REack* transition was captured at $54.00ns$. For the second simulation, the SQF vector is configured as $<010>$, and the first *REack* transition was captured at $53.88ns$. In the last simulation of this step, the SQF vector is configured as $<001>$, and the first *REack* transition was captured at $54.20ns$. Applying these values into the Equation 5.2, leads to $\Delta_0 = 54.00 - 50.00 - 3.34 = 0.66ns$, $\Delta_1 = 53.88 - 50.00 - 3.34 = 0.54ns$ and $\Delta_2 = 54.20 - 50.00 - 3.34 = 0.86ns$.

After finishing the second step, the third step can be executed. Like the in the previous step, for all the simulations executed in this step the T_{Lreq} is $50.00ns$. For measuring the δ_0 delay, the SQF vector is configured as $<010>$. Note that SQF1 is configured to rise its *err1*, which generates a trasition in *Error1*. In this simulation, the first *Error1* transition was captured at $52.08ns$. For measuring δ_1, the SQF2 is the one that configured to raise its *err1*, and the vector is $<001>$. In this last experiment, the first *Error1* transition was captured at $54.20ns$. Putting these times into the Equation 5.3, results in $\delta_0 = 52.08 - 50.00 - 0.00 - 0.54 = 1.54ns$ and $\delta_1 = 54.20 - 50.00 - 1.54 - 0.86 = 1.80ns$. Note that for the δ_0, the sum of previous δ measurements is 0.00, while for δ_1 this sum is equal to δ_0. At the end, the simulation environment presents a log file with all the calculated delays. As future works, this part of the simulation environment will be incorporated into the Blade design flow to experiment with a netlist mapped to a standard cell library.

6. DELAY FAULT TESTING OF CRITICAL PATHS

This Chapter presents contribution 5 (see Section 1.2). Different from the previous fault analysis of the Blade specific blocks, this is focused on delay fault testing of the combinational logic in the data pah, that is the same regardless of whether it is a classic synchronous or a Blade implementation. This work is the object of recently published DATE conference paper [KUE18].

As previously discussed, from the technology perspective, there is no distinction between manufacturing asynchronous or synchronous CMOS circuits, which means that expected defects are similar in both designs [RON99]. This can also be extended to synchronous or asynchronous timing resilient circuits. However, from the test perspective, timing resilient circuits are fundamentally different when compared to conventional circuits, since the former tolerate some timing violations, and not all timing violations are faults. So, a test method for delay fault testing of the combinational logic must be capable of distinguishing between a recoverable timing violation and an actual delay fault.

Despite this difference, a natural path when considering the testability of timing resilient circuits is to look at classical approaches used in industry, such as the scan technique, which can be applied to various types of structural faults, including stuck-at and delay. Some of these attempts where presented in Section 3.2, such as the Scan Razor [ANA15], the TimeD scan architecture [FLO08] and the test methodology presented in [YUA13] [HAN13]. All these works proposed to transform EDS elements into scannable elements to test synchronous timing resilient circuits. As already discussed, one of the problems with the scan approach for testing timing resilient circuits is the area overhead of the test structures.

Different from the two first works ([ANA15] and [FLO08]), the test method proposed in [YUA13] [HAN13] address the problem of distinguishing a recoverable timing violation from an actual delay fault. Their method reuses the error detection logic to detect delay faults by applying two-pattern delay test through the data path scan chain. A clock divider and a duty-cycle control circuit to select different test speeds and duty-cycle configurations are used to differentiate timing violations from delay faults in the combinational logic. The idea of reusing the error detection logic for test purposes is also developed in [YI14], but it is more focused on developing a scan-based aging monitoring scheme. The circuit is monitored during normal operation, where no faster-than speed testing is applied, and it gives an alarm if aging is detected so that actions can be taken before a major failure, such as reducing the operating frequency of the circuit. Experimental results show that the proposed solution has less switching activity, thus consuming less power. This method also implies in relatively less overhead in large designs when compared to the previously reviewed methods, including Razor [ERN03], because no data/clock delay circuit and additional clock tree are needed. Unfortunately, the authors do not present synthesis results for area comparison. It is interesting to see in these works that the reuse of the error detection logic for test purposes is often applied. It reflects the area overhead problems frequently observed in timing resilient architectures,

where the designer is looking for ways to amortize the test circuitry by reusing the resiliency circuit for test purposes.

6.1 Proposed Test Method

The constant timing monitoring of resilient circuits provides feedback about the current state of the circuit during functional operation, which suggests the reuse of the EDL for detecting timing variations in the combinational logic that can be associated to a path delay fault. Therefore, the test method proposed here explores the concurrent test capabilities of circuits implemented with Blade by reusing the EDL as a fault detection mechanism. It also relies on functional testing to produce test patterns for the critical paths, reducing the area overhead required by a dedicated scan chain.

As presented in [KRS03], functional testing may translate into delay fault under-testing, where non-functional delay paths are not covered. On the other hand, structural testing can over-test paths that are not functionally activated, resulting in yield loss, because the chip is discarded due to faults in non-functional paths. An important assumption the proposed test method is that if the defect does not affect the behavior of the circuit in functional mode, a missed defect should not cause the system failure. In this case, the chip should not be rejected based on a failure in a non-functional path. Specifically for testing delay faults in critical paths with Blade circuitry, functional testing can achieve a satisfactory fault coverage with a low test circuitry overhead.

The proposed test approach focuses on online testing of path delay faults of critical paths as a natural consequence of Blade's implementation, where critical paths have an EDL constantly monitoring possible timing violations. The Blade controller is modified in this approach to detect faults through the EDL by shifting the TRW, which essentially means to make pipeline stages to operate slower. Note that this slower mode is only applied for testing, and the modifications do not affect the functional mode of operation. With the shifted TRW, the EDL now detects timing violations later than the violations identified during functional mode. These later timing violations are in fact delay faults being detected, which would not be captured by the EDL.

6.1.1 Test Architecture

Despite the difference in the Burst-Mode and Click designs, the proposed test circuitry is the same. The following examples use the Burst-Mode implementation as the reference, but all functional results can be related to a circuit implemented with the Click based controller. Figure 6.1(a) shows the original delay *CLK* control, and Figure 6.1(b) illustrates the associated timing diagram. Figure 6.1(c) shows the modified circuit that creates the shifted TRW for the Burst-Mode controller. A Click-based example is illustrated in Appendix A (Figure A.4). The AND gate before the multiplexer ensures the *CLK* deactivation immediately after the controller deactivates

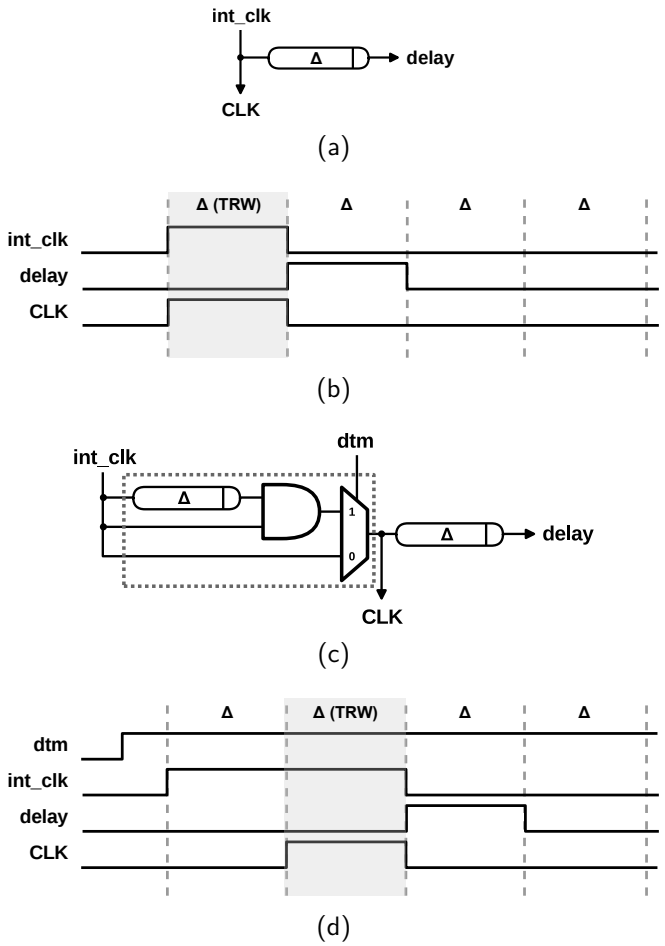


Figure 6.1 – (a) Original Blade burst-mode controller CLK output circuit. (b) Timing diagram of the original controller circuit. (c) Modified CLK output circuit. (d) Timing diagram of the modified circuit.

the *int_clk*. Otherwise, the TRW would also be extended by the additional Δ delay. The *dtm* (delay test mode) signal selects whether the TRW must be shifted or not. The *dtm* signal is directly connected to all controllers. Thus if activated, all controllers will shift TRW. The timing diagram of Figure 6.1(d) illustrates this behavior.

The rest of the controller circuit remains unchanged, and the modifications are transparent to the rest of the circuit, since the controller still waits for the *delay* signal transition to continue with the handshake protocol. Timing constraints and the performance in normal mode are not affected either, and the delays of the additional gates can be compensated in the delay line. Note that, the method is limited to detect faults that do not exceed the extra Δ time, and faults that trespass the shifted TRW are not captured.

The same way as the delay lines test method presented in Section 5.3, an OR gate groups the *err1* signals from all the EDLs on the circuit and its output are mapped to the Error1 pin as an observation point for flagging the faults detected by an EDL. Once the *dtm* is enabled, the *Error1* is constantly monitored while the circuit executes a functional test. If at any moment a transition at *Error1* occurs, a delay fault is considered detected.

The proposed modifications assume that the TRW shift is implemented with an additional delay line. This delay line is the one inside the dotted region of Figure 6.1(c). Although the test method presented in Section 5.3 requires the removal of the extra delay line (*edo* $\rightarrow$ *edi*) from the Burst-Mode controller (Figure 5.3), which led to the development of the Click based controller, the additional delay line of Figure 6.1(c) does not affect its testability. Since they are interconnected, the Step 2 (see Section 5.3) is split into two separate test. The first one with *dtm* enabled, resulting in a time measurement that is equivalent to 2Δ, and the second test with *dtm* disabled, measuring only the time of original delay line. The time difference between these two tests is used to extract the propagation delay of each delay line.

Although the proposed method is limited to one TRW shift, it is also possible to have an alternative design where, for instance, multiples activations of a single delay line are controlled, thus extending the shift to detect longer delay faults. The necessity of detecting delay faults that exceed the shifted TRW depends on more analysis that will be addressed in the future. However, there are some modifications in the proposed architecture that are discussed in the following section.

6.1.2 Yield Improvement and Aging Monitoring

In the architecture presented in the previous section, a single primary input is connected to all internal *dtm* signals, and it affects all controllers. In this case, all controllers will shift the TRW at the same time. Alternatively, the *dtm* signal of each controller can be controlled individually by using a scan chain as presented in Figure 6.2. The dashed *global_dtm_i* is replaced by an auxiliary scan chain connected to *dtm_i*.

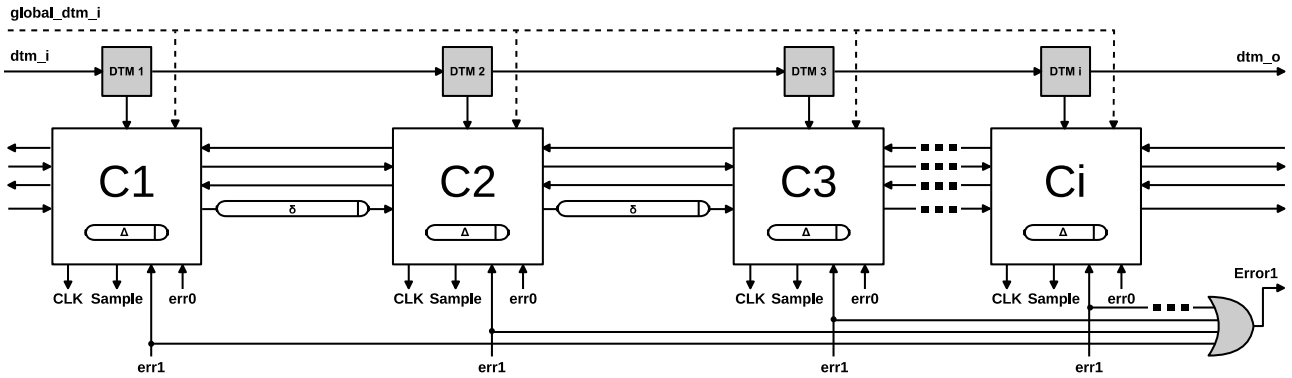


Figure 6.2 – An alternative design with individual control over the *dtm* signal.

This alternative design opens some new possibilities regarding fault diagnose, yield improvement and aging monitoring. For instance, assume that there is a delay fault in a critical path between controllers C1 and C2. If the *global_dtm_i* approach is applied, there is no way to determine, just by observing *Error1*, which stage contains the fault. If the *dtm* of each controller can be set individually, a diagnosis procedure can be executed to discover which stage captured the fault. For this example, only the DTM2 scan register holds a *dtm* enable for shifting C2 TRW. In this

scenario, the delay fault is captured, and it is known for sure that the *Error1* transition came from C2, and the combinational logic between C1 and C2 is faulty.

Individual control for each *dtm* can also be used to increase yield if the fault is detected during manufacturing tests. For instance, if a delay fault is detected at the shifted TRW, and the fault location can be diagnosed, the *dtm* signals of stages where faults were identified can be enabled permanently, even during normal operation. The consequence is a stage that is Δ time slower than its original design. Moreover, with the advantage of being an asynchronous design, only the faulty stages can be slowed down, and the overall performance impact will not be like in a synchronous circuit, where all stages would be affected by a clock frequency reduction that accounts only for the faulty stages. In this case, a circuit that would be discarded can still be commercialized as a lower performance version.

Still, on the new possibilities of this implementation, an aging monitoring mechanism can be applied to detect a performance degradation during the circuit lifetime by monitoring the *Error1* signal. The constant monitoring of this signal can be translated into a reliability metric, the error-rate. An increase in the error-rate can be related to circuit aging, and actions can be taken to avoid the circuit failure, increasing the circuit lifetime. Moreover, if aging is detected, a diagnose test, similar to the one described earlier, can be performed to detect which stages are being affected. Also, like the solution to increase yield, the affected stages could be purposely set to operate in a slower mode by enabling their *dtm* during normal operation. Further evaluations on yield and aging improvements using the proposed method are left as future works.

6.2 Experiments and Results

The proposed test method was included in Blade's automated flow, automatically adding *dtm* port, instantiating the OR gate to group the *err1* signals and making their proper interconnections. The evaluated results consist of fault coverage and area overhead. Although the proposed method can also improve yield and aging, these evaluations are left for future works. The flow targets a 28nm FDSOI technology and the method is evaluated with a 32-bit XTEA crypto core, based on the Speed XTEA described in [KAP08], and the same case study presented in [HAN15] was implemented, a 3-stage 32-bit MIPS OpenCore CPU called Plasma [PLA14], including the original Burst-Mode controller. For both case studies, the synchronous base netlist was synthesized for a $2.20ns$ clock period.

The fault simulation environment (see Section 4.3.1), previously incorporated into Blade's co-simulation environment for testing the TEDL (Section 4.4.4), now has some new input parameters. The parameters are: a list of critical paths extracted from the Blade flow, where the EDL is placed, and the delay faults are injected; the Δ delay; and the original SDF (Standard Delay Format) file. For each critical path, a mutant SDF is generated with a timing violation for that specific path. The additional time is equivalent to the Δ delay plus the slack of the path. It adds a propagation

delay that shifts the path transitions to the shifted TRW, thus if any transition occurs in this path, it will be flagged by the EDL as a delay fault. After all critical paths with EDL are simulated, the fault simulation environment presents a report with the total simulated paths and the fault coverage.

Table 6.1 presents the fault coverage for both case studies. For the XTEA fault simulation, a testbench executes 2000 encryption and decryption operations with randomly generated data, totaling 2000 handshake cycles, while 18601 instructions are executed by the Plasma, one for each handshake cycle. The simulation environment shows that 100% of the delay faults in the critical paths were detected with the XTEA.

Table 6.1 – Fault coverage for critical paths of Blade implementations of the Plasma CPU and XTEA crypto core.

	Plasma	**XTEA**
Total Critical Paths	625	6271
EDL Critical Paths	238	937
Detected Faults	172	937
Fault Coverage	72.27%	100%

The experiments with Plasma used the testbench provided with the OpenCore package to produce the input data. Reports for Plasma fault simulation show a 72.27% fault coverage. This lower fault coverage observed with Plasma is explained by the use of functional code that was not developed aiming functional testing. Most of the uncovered paths are related to high-order bits from registers that had no transition during the simulation, such as the *program counter* and *memory address* registers. As already pointed, a transition must occur in the path, so the EDL captures the delay fault. Although this is out of the scope of this work, software-based testing can be used to improve fault coverage of processors [LAI00] [SIN06]. For example, the work in [KRA05] used Plasma as a case study and reported 95% of fault coverage.

Table 6.2 – Comparison between Blade implementations with and without the proposed DTM circuitry. Blade Plasma vs Blade Plasma-DTM (PDTM) and Blade XTEA vs Blade XTEA-DTM (XDTM) in terms of area.

	Plasma	**PDTM**	**XTEA**	**XDTM**
Combinational	7095.28	7124.33	35561.44	35599.80
Buf/Inv	608.90	630.93	2959.80	2991.13
Noncombinational	7860.69	7860.69	21828.00	21828.00
Macro/Black Box	228.58	228.58	1280.01	1280.01
Net Interconnect	undefined	undefined	undefined	undefined
Total Area	15793.45	15844.53	61629.25	61698.94
Area Overhead	-	**0.32%**	-	**0.11%**

Table 6.2 shows that for both case studies have area overhead lower than 1%. The additional area comes from the increment in the Combinational logic and the Buf/Inv, which is the extra delay line. Moreover, the area overhead of the proposed test circuitry does not scales up with the number of critical paths covered. If more paths are selected in the Blade flow to receive an EDL, the area overhead comes only from the additional EDL circuitry. This way, the designer can

tradeoff between additional EDL area and the number of paths covered by timing resilience and the proposed test method. For future works, the area overhead can be further reduced by reusing the existing delay line instead of creating a new one.

7. CONTRIBUTIONS OVERVIEW

This Chapter presents an overview of this Thesis contributions, relating the different proposed test approaches with the test application phase, the supported fault models for each approach and an area overhead comparison between classic scan test approach and all the proposed test techniques integrated into a final Blade netlist.

7.1 Scan Comparison

One of the biggest concerns that motivated the research of different test approaches for Blade was the silicon area cost related to testing circuitry. Timing resilient architectures already incur in significant area overheads for the detection and recovery circuitry, such as the 21% increase in combinational logic and 280% increase in the sequential area for the Bubble Razor [FOJ13]. Even so, some works proposed the classic scan approach for testing synchronous timing resilient architectures with TimeD [FLO08] and the Scan Razor flip-flop [ANA15]. Despite the authors comments about the lower area overheads of their approaches, they do not present quantitative results for a real case study.

The original Blade [HAN15] has an overall area overhead of 8.4%, while the Bubble Razor [FOJ13] the overall overhead is 87%. Like the Bubble Razor architecture, Blade is a latch-based design, where LSSD [JUR17] is the scan design for this type of sequential cell, but LSSD presents a significant area overhead once each latch is replaced by two latches. Moreover, Blade is also an asynchronous design, thus scan DfT can present some challenges due absence of a global clock and the usage of non-standard sequential components (C-element and Q-Flop). Therefore, it seems that the use of scan technique for Blade can present a high area overhead, but the question is how much higher. In this context, the work of Juracy [JUR18a] was developed in parallel to this Thesis, exploring the DfT scan approach in Blade, and allowing a quantitative area comparison between the proposed test methods and the scan design.

In his work, Juracy proposed an optimization for the LSSD [JUR17] and Clocked-LSSD test cells. This last scan cell implements the LSSD protocol for flip-flops. Thus a design containing both latches an flip-flops can be interconnected in the same scan chain. Juracy also presents a testable implementation for the Q-Flop cell. His implementation allows automatic scan replacement of the Q-FLop without affecting the internal metastability filter [JUR18b]. Automatic scan replacement and connection is also described in [JUR18a], where the Blade flow described in Section 3.1.7 is modified to implement a synchronous mode in the Blade design. Test clock trees are added and the Blade controllers modified with an additional multiplexer to select between their internal clock and the test clock externally generated during the synchronous test mode.

The area results for the scan approach presented by Juracy [JUR18a] and the proposed test methods proposed in this Thesis considered two test cases, the 32-bit XTEA crypto core, based

Speed XTEA described in [KAP08], and Plasma [PLA14], a 3-stage 32-bit MIPS OpenCore CPU. Both synthesis target the 28nm FDSOI technology and a $2.20ns$ clock period. The synthesis results are presented in Table 7.1.

Regarding the area results for this Thesis work, the results include the area of elements that are not still incorporated automatically in the final netlist, which are the scan elements used to extract the patterns for the TEDL proposed in Section 4.4 and the scan Q-Flop used for testing the delay lines. These area estimations are calculated based on the cell area values extracted from the 28 nm library documentation. The area of the 14 mux-D scan elements needed for both Plasma and XTEA designs is 79.968 µm^2. For the 20 Q-Flops of Plasma the additional area is 62.04 µm^2, and for the 112 Q-Flops of XTEA the additional area is 347.42 µm^2

Table 7.1 – Synthesis results for the Blade XTEA (B-XTEA) and Blade Plasma (B-Plasma) designs with scan DfT approach [JUR18a] and the implementations of this work.

	Comb. Cells	Seq. Cells	Total Area (µm^2)	Overhead
B-Plasma	3603	1615	14964.74	-
B-Plasma Scan	3603	1615	22475.44	50.19%
B-Plasma This Work	3742	1631	15599.13	4.24%
B-XTEA	31205	15644	64532.75	-
B-XTEA Scan	31205	15756	137260.11	112.70%
B-XTEA This Work	32199	15660	68966.70	6.87%

Integrating all the proposed modifications of this work resulted in an area overhead of 4.24% with the Plasma and 6.87% with the XTEA, while the area overhead with the scan approach is 50.19% with the Plasma and 112.70% with the XTEA. Regarding only the scan approach, Juracy [JUR18a] explains that the significant difference in area overheads is related to sequential cells proportion present in each case study. Comparing his results with this work demonstrate that silicon cost for the proposed test methods is significantly lower than the classic scan approach. However, these two test approaches are not covering the same set of faults. Thus a fault coverage comparison would not reflect a qualitative measure adequately for each one. Even though, these results suggest that the scan approach may be too expensive and not ideal for testing Blade.

7.2 Proposed Test Methods Application

As presented in Section 3.1.7, a Blade circuit can be divided into four main blocks: (i) controllers; (ii) delay lines (iii); error detection logic (EDL); (iv) data path. Following this division, each one of these blocks had their testability analyzed in the previous chapters, and different test methods were proposed for detecting delay and stuck-at faults in these blocks. Table 7.2 summarizes the fault models supported by the test method proposed for each block. The number of check marks gives a notion of how efficient the proposed methods is for detecting the faults.

The fault classification and TEDL architecture proposed for testing Blade's EDL presented high stuck-at fault coverage, but delay faults were partially evaluated since only the original EDL

Table 7.2 – Supported fault models by the proposed test methods for each Blade block.

Block	Stuck-at	Delay
EDL	✓✓	✓
Controller	✓✓	-
Delay lines	✓✓	✓
Datapath	-	✓

architecture was analyzed. However, extending the fault classification to consider delay faults does not require any modification in the proposed TEDL architecture and can be analyzed the same way as the original EDL analysis. This analysis is left for future works.

Regarding the controller block, the analysis of the Burst-Mode and the proposed Click-based only address the stuck-at faults, where the Click-based implementation achieved better results when considering the detection of stuck-at faults through the observation of system halt. The delay lines test method was only validated with the equations and behavioral simulations. Thus further experiments are required to prove its efficiency. Stuck-at faults in the delay lines are equivalent to stuck-at faults in the interconnections between the controllers, which cause the system halt. Delay faults in critical paths of the data path are covered by the test method presented in Chapter 6, but it relies strongly on functional test patterns to produce transitions in these critical paths, and stuck-at faults cannot be detected through this method. A built-in self-test (BIST) method could be used for this purpose, where the expected response is compressed with multiple-input signature registers (MISR).

Table 7.3 shows a how the proposed test methods can be classified regarding the test application phases. In this classification, there are three possible phases, *manufacturing* test, that considers structural fault testing before selling the circuit, *online* test, where the circuit is tested concurrently to its normal operation, and *offline* test, where the normal operation mode must be suspended for the test.

Table 7.3 – Test phase classification of the proposed test methods.

Block	Manufacturing	Online	Offline
EDL	✓✓	-	✓
Controller	✓✓	✓✓	-
Delay lines	✓✓	-	-
Datapath	✓✓	✓✓	✓✓

As shown in Table 7.3, the fault testing method proposed for testing the EDL is used for detecting stuck-at faults in the manufacturing phase, although it can be applied for periodic offline testing. Online testing is not possible since the normal circuit operation must be suspended for extracting the behavioral patterns for fault classification. The Click-based controller can be tested during the manufacturing phase, where the circuit will be tested for a system halt detection. This behavior can also be considered for online testing. If the controller circuit develops a stuck-at fault over time, which can occur due to circuit aging, the fault can also be observed by the system halt. The delay lines test method is only applied during circuit manufacturing to measure post-fabrication

propagation delay. The test method proposed for testing the critical paths of the data path is the one that presets the most possibilities since it can be used for detecting path delay faults in the manufacturing phase, but also serves as an online monitoring mechanism, and even for offline diagnosis (fault location and identification).

Finally, Table 7.4 presents a high-level comparison between classic scan approach applied to Blade in [JUR18a] and the work of Yuan [YUA13], which also applies the scan approach. Different from the work of Juracy [JUR18a], Yuan proposed a custom scan call for the DSTB, which is the EDS used in their resilient architecture, while Juracy applies standard LSSD scan cells, that are compatible with commercial DfT tools.

Table 7.4 – High level test phase comparison with three different approaches for testing timing resilient circuits.

Test Application	This Work	Blade Scan [JUR18a]	DSTB Scan [YUA13]
Manufacturing	✓✓	✓✓✓	✓✓
Online	✓	-	-
Offline	✓✓	-	-
Yield	✓	-	-
Aging	✓	-	-

Since the works of Juracy [JUR18a] and Yuan [YUA13] are full scan approaches, they are ideal for structural testing in the manufacturing phase and can be applied for full testing of the data path. Yuan also addresses the reuse of EDS for testing, but still rely on the structural test patterns, which are not applied for testing after manufacturing phase. However, the work of Yuan may not be supported by commercial DfT and ATPG tools. Thus, the work of Juracy is more suited for manufacturing test. Although the proposed test methods of this Thesis do not achieve the level of fault testing and fault coverage of the other two works, they extend the testability of the Blade template to different test applications phases that covers manufacturing, online and offline testing. The yield and aging improvements discussed in Section 6.1.2 demonstrate that Blade can be a reliability-oriented implementation for different test application phases, from manufacturing to infield concurrent testing.

One last thing to discuss is how generic are the proposed test methods, and if they can be applied to other similar timing resilient templates. The proposed fault classification can be extended to other EDLs, but the behavioral patterns must be generated for each design, which is similar to developing a fault model for different logic gates, necessary modifications (concurrent checkers), such as the proposed TEDL must be custom made. Regarding the controller, relying on system halt for detecting faults in the control part is restricted to asynchronous controllers. However, it was demonstrated that this behavior is not present in all asynchronous controllers. Although the asynchronous Sharp [WAU17] is also a Click implementation, Sharp implements a distinct handshake protocol, which results in a different circuit. Thus, claiming that Sharp is also self-tested for stuck-at faults like the Click-based design proposed for Blade, depends on further experiments, similar to the ones executed in Chapter 5. The delay lines test method could be adapted, but it may not be ideal for Sharp since the number o delay elements increase to four. Deriving the test time equations for

the different delay lines would require additional observability and controllability. Finally, the delay fault testing of critical paths seems to be applied to other asynchronous timing resilient with minimal effort. In fact, the same circuit proposed in this Thesis could be used to shift the TRW for detecting path delay faults. Assuming that the same circuit is used, the yield and aging improvements are also applicable.

8. CONCLUSION

Timing resilient architectures became a promising alternative for reducing the timing margins due to PVT variations, while still improving performance and energy efficiency. On the other hand, some proposed designs suffer from metastability issues. They also present high recovery penalties and high area overheads, which leads to restrictions for test circuitry. Still, the scan approach is the most accepted, even for these resilient circuits. However, testing timing resilient circuits is fundamentally different from testing standard synchronous designs since they tolerate timing violations, and the existence of a timing violation does not necessarily determine the chip should be discarded. Thus the pass or fail criteria is different. All these factors hinder the usage of timing resilient templates by the industry.

Different from the majority of timing resilient proposals, Blade is an asynchronous template that presents lower area and recovery penalties while removing the metastability problems, but testability can become even more challenging. The absence of a global clock in asynchronous circuits and the usage of non-standard sequential cells difficult the implementation of scan-chains. As a latch-based design, the use of scan-chains in Blade is limited to LSSD scan approach, which is known for presenting high area overheads. The proposed test methods demonstrated that Blade can be tested in different phases (manufacturing, online and offline) for stuck-at and delay faults. Fault analysis and synthesis results presented throughout this document show that these reliability improvements present a satisfactory fault coverage for an area overhead of 4.24% and 6.87% for the Plasma CPU and the Speed XTEA crypto core respectively. The full scan approach was not considered, and the Blade testability was evaluated from a different perspective, where Blade is not only a timing resilient template, but a design with self-testing capabilities and increased reliability properties, including high potentials for yield improvements and aging monitoring, which where not explored in previous papers. Thus, the Blade's features could be expanded from a timing resilient template to a template with multiple reliability-oriented properties. The rest of this Chapter is dedicated to present contributions and future works.

8.1 Contributions

This work presents a fault classification method for stuck-at and delay faults inside Blade's EDL. These faults were classified based on the functional behavior of the circuit in the presence of a fault. Initially, the test method used with the fault classification relied solely on the existing EDL circuitry to produce behavioral patterns for detecting faults. However, controllability and observability were necessary to improve fault coverage additional, which leads to another contribution, the testable EDL. For the TEDL, the fault classification was expanded to incorporate additional observation points and to account for the possibility of forcing timing violations, improving the stuck-at fault coverage.

Another contribution is stuck-at fault analysis of Blade's controller and the subsequent development of the Click-based design for the speculative handshake protocol. The controller fault analysis demonstrated that the testing of the original Burst-Mode could not rely on circuit halt. However, this behavior was observed in the Click-based design, making it a better alternative for Blade. Moreover, the Click-based approach presented an overall area reduction for the set of four different versions required for the Blade flow. This reduction was mainly caused by the removal of an additional delay line that was present in the original Burst-Mode implementation. Another improvement regarding the Click-controller is that it can be automatically mapped to the desired technology optimized by the synthesis tool, while for the Burst-Mode the technology mapping is done manually, and synthesis optimization cannot be allowed since the synthesis tool may modify redundancies implemented in the Burst-Mode circuits.

Testing the delay lines in the Blade template is another contribution. The proposed test method assumes the Q-Flop controllability through a scan Q-Flop implementation and the observability of the internal error signals. The Click-based controller is required for this tests assuming that a single internal delay is implemented. The test is offline, assuming controllability of primary inputs and observability of primary outputs, and precise methods to measure time (e.g., test equipment). Different test steps and measurements are required to solve the equations that define the propagation delay of each delay line.

The last contribution provides a path delay fault detection for the critical path in a circuit implemented with Blade. The Blade controller is modified to shift the timing resiliency window. While the shifted window is enabled, the timing violation reported by the EDL are in fact path delay faults that otherwise would be missed by the EDL. Functional testing is used to stimulate the circuit paths during the test. The proposed circuit modifications present area overheads that vary from 0.32% to 0.11% depending on the case study, but most importantly, this proposal expands the Blade usage for yield and aging improvements.

8.2 Future Works

Some research topics were not considered or not completely evaluated. Thus they can be addressed in future works. The different topics are discussed following the contributions order presented in the previous section. The first analysis presented for the Blade's EDL considered stuck-at and propagation delay faults, and fault simulations were executed with a high-level test environment. However, for the TEDL, only stuck-at fault simulations were integrated into the co-simulation environments of Blade's flow. Thus, before extracting the delay fault coverage for the TEDL, the Blade flow must be able to increase the propagation delay of the internal elements of the TEDL. The next step would be to expand the fault classification to consider behavioral patterns for the delay faults.

Still, regarding the TEDL integration to the Blade flow, all the internal modification proposed to increase the TEDL are automatically inserted by the logic synthesis, except for the scan chain proposed for extracting the fault patterns. Once these scan elements are added to the final netlist, the test simulation environment must be expanded to operate the scan chain and obtain the patterns for the fault classification. Lastly, further area optimization can be achieved by developing custom cells for the transition detector, which at this point is described with standard cells.

Like the TEDL, the Click-based controller was not evaluated for delay faults and requires the integration of delay faults simulations into the Blade's flow. An alternative approach is to detect the delay faults in the controller through the delay lines test method. Another approach is to assess first the Click-controllers testability separately from a Blade design, where commercial DfT and ATPG tools are used. This approach is possible because the Click design only uses standard cells. The flip-flops can also be automatically replaced by scan cells as demonstrated by the author of Click to improve controllability and observability. In this scenario, all the fault models supported by the commercial tools can be applied, such as bridging and stuck-open fault models, and the post-fabrication tests of the controller would be isolated from the rest of the circuit.

The delay line testing was only validated with behavioral test environment using high-level RTL descriptions for the different circuit components. Despite the successful simulations, the test scenario does not describe a real circuit and the test method does not account for forks and joint. Thus, it must be evaluated with more realistic circuits to prove its effectiveness. It is expected that timing simulations of a real case study implemented with Blade will require an updated in the proposed equations to account propagation delays inside the controller. Although Blade allows reconfigurable delay lines, these were not accounted. However, the same test method could be repeated for all delay line configurations for completeness or a subset of delay line configurations when test speed is more important.

Another subject for the future works is related to the proposal for testing delay faults in the critical paths of Blade's data path. This proposed test method relies on functional testing. Thus, software-based testing approaches can be further analyzed to improve fault coverage of Plasma (or other processors) along with the development of other case studies. Moreover, the suggested yield and aging improvements can also be explored with complete implementations and experiments. Although the area overhead for this proposal is quite low, further area optimization can be applied. The additional delay line required for shifting the resiliency window may be replaced by a circuit to control multiple activations of a single delay line. This multiple activation delay line would also serve to extend the shift to detect longer delay faults, but the actual necessity of detecting such longer faults must be evaluated.

One last topic to be addressed in the future is regarding the compatibility of the proposed test methods with other timing resilient templates, more specifically the Sharp template since it is based on Blade. Finally, a more general study would evaluate performance and power overheads. Different from power, timing overheads of the proposed implementation were briefly discussed

throughout this work suggesting that the impact is minimal to none. However, it is interesting to demonstrate this analysis quantitatively.

REFERENCES

[ANA15] Anastasiou, A.; Tsiatouhas Y.; Arapoyanni, A. "On the Reuse of Existing Error Tolerance Circuitry for Low Power Scan Testing". In: *International Symposium on Circuits and Systems*, 2015, pp. 1578-1581.

[BEE02] Beest, F. te; Peeters, A.; Verra, M.; Berkel, K. v; Kerkhoff, H. "Automatic Scan Insertion and Test Generation for Asynchronous Circuits". In: *International Test Conference*, 2002, pp. 804-813.

[BEE03] Beest, F. te; Peeters, A.; Berkel, K. van; Kerkhoff, H. "Synchronous Full-Scan for Asynchronous Handshake Circuits". *Journal of Electronic Testing*, vol. 19-4, Aug 2003, pp. 397-406.

[BEE05] Beest F. te; Peeters, A. "A Multiplexer Based Test Method for Self-Timed Circuits". In: *International Symposium on Asynchronous Circuits and Systems*, 2005, pp. 166-175.

[BEE07] Beerel, P. A.; Roncken, M. E. "Low Power and Energy Efficient Asynchronous Design". *Journal of Low Power Electronics*, vol. 3-3, Dec 2007, pp. 234-253.

[BEE10] Beerel, P. A.; Ozdag, R.; Ferretti, M. "A Designer's Guide to Asynchronous VLSI". Cambridge, UK: Cambridge University Press, 2010, 352p.

[BEE14] Beer, S.; Cannizzaro, M.; Cortadella, J.; Ginosar, R.; Lavagno L. "Metastability in Better-Than-Worst-Case Designs". In: *International Symposium on Asynchronous Circuits and Systems*, 2014, pp. 101-102.

[BEE17] Beerel, P. A.; Breuer, M.; Cheng, B.; Hand, D. "A Timing Violation Resilient Asynchronous Template", USA patent 9558309, Jan 2017.

[BER93] Berkel, K. van "Handshake Circuits: An Asynchronous Architecture for VLSI Programming". Cambridge, UK: Cambridge University Press, 1993, 239p.

[BER02] Berkel, K. van; Peeters, A.; te Beest, F. "Adding Synchronous and LSSD Modes to Asynchronous Circuits". In: *International Symposium on Asynchronous Circuits and Systems*, 2002, pp. 161-170.

[BOW09] Bowman, K.; Tschanz, J.; Kim, N. S.; Lee, J.; Wilkerson, C.; Lu, S.; Karnik, T.; De, V. K. "Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance," *Journal of Solid-State Circuits*, vol. 44-1, Jan 2009, pp. 49-63.

[BUS02] Bushnell M. L.; Agrawal, V. D. "Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits". New York, USA: Springer, 2002, 690p.

[CAR87] Carter, J. L.; Iyengar, V. S.; Rosen, B. K. "Efficient Test Coverage Determination for Delay Faults". In: *International Test Conference*, 1987, pp. 418-427.

[CAN15] 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, Aug 2015, pp. 2238–2247.

[CHA10] Chang, I. J.; Park, S. P. Roy, K. "Exploring Asynchronous Design Techniques for Process-Tolerant and Energy-Efficient Subthreshold Operation". *Journal of Solid-State Circuits*, vol. 45-2, Feb 2010, pp. 401-410.

[CHE93] Cheng. K.-T. "Transition Fault Testing for Sequential Circuits". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12-12, Dec 1993, pp 1971-1983.

[CHR16] Chrysanthou, K.; Englezakis, P.; Prodromou, A.; Panteli, A.; Nicopoulos, C.; Sazeides, Y.; Dimitrakopoulos, G. "An Online and Real-Time Fault Detection and Localization Mechanism for Network-on-Chip Architectures". *ACM Transactions on Architecture Code Optimization*, vol. 13-2, Jun 2016, pp. 22:1–22:26.

[CHU03] Chung, K. Y.; Gupta, S. K. "Structural Delay Testing of Latch-Based High-Speed Pipelines with Time Borrowing". In: *International Test Conference*, 2003, pp. 1089-1097.

[CHU06] Chung, K. Y.; Gupta, S. K. "Low-cost Scan-based Delay Testing of Latch-based Circuits with Time Borrowing". In: *VLSI Test Symposium*, 2006, pp. 8-15.

[CRO99] Crouch, L. A. "Design-for-Test for Digital IC's and Embedded Core Systems". New Jersey, USA: Prentice Hall, 1999, 347p.

[DAS09] Das, S.; Tokunaga, C.; Pant, S.; Ma, W.-H.; Kalaiselvan, S.; Lai, K.; Bull, D.; Blaauw, D. "RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance". *Journal of Solid-State Circuits*, vol. 44-1, Jan 2009, pp. 32-48.

[EIC77] Eichelberger E. B.; Williams, T. W. "A Logic Design Structure for LSI Testability". In: *Design Automation Conference*, 1977, pp. 462-468.

[ERN03] Ernst, D.; Kim, N.; Das S.; Pant, S. "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation". In: *International Symposium on Microarchitecture*, 2003, pp. 7-18.

[FLO08] Floros, A.; Tsiatouhas, Y.; Kavousianos, X. "The Time Dilation Scan Architecture for Timing Error Detection and Correction". In: *International Conference on Very Large Scale Integration*, 2008, pp. 569-574.

[FOJ13] Fojtik, M.; Fick, D.; Kim, Y.; Pinckney, N.; Harris, D.; 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". *Journal of Solid-State Circuits*, vol. 48-1, Jan 2013, pp. 66-81.

[FUH95] Fuhrer, R.; Lin, B.; Nowick, S. "Symbolic Hazard-Free Minimization and Encoding of Asynchronous Finite State Machines". In: *International Conference on Computer-Aided Design*, 1995, pp. 604-611.

[GIL06] Gill, G.; Agiwal, A.; Singh, M.; Shi, F. Makris, Y. "Low-Overhead Testing of Delay Faults in High-Speed Asynchronous Pipelines". In: *International Symposium on Asynchronous Circuits and Systems*, 2006, pp. 45-56.

[HAN13] Han, Q.; Guo, J.; Jone W. B.; Xu, Q. "Path Delay Testing in Resilient System". In: *International Midwest Symposium on Circuits and Systems*, 2013, pp. 645-648.

[HAN15] Hand, D.; Moreira, M. T.; Huang, H.-H.; Chen, D.; Butzke, F.; Li, Z.; Gibiluka, M.; Breuer, M.; Calazans, N. L. V.; Beerel, P. A. "Blade – A Timing Violation Resilient Asynchronous Template". In: *International Symposium on Asynchronous Circuits and Systems*, 2015, pp. 21-28.

[HAR01] Harris, D. "Skew-Tolerant Circuit Design". San Francisco, USA: Morgan Kaufmann, 2000, 300p.

[HUA16] Hua, W.; Tadros R. N.; Beerel, P. A. "Low Area, Low Power, Robust, Highly Sensitive Error Detecting Latch for Resilient Architectures". In: *International Symposium on Low Power Electronics and Design*, 2016, pp. 16-21.

[ITR11] ITRS 2011, "International Technology Roadmap for Semiconductors". Available at: http://www.itrs2.net/2011-itrs.html, Mar 2018.

[ITR12] ITRS 2012, "International Technology Roadmap for Semiconductors". Available at: http://www.itrs2.net/2012-itrs.html, Mar 2018.

[IWA10] Iwata, H.; Ohtake, S.; Inoue, M.; Fujiwara, H. "Bipartite Full Scan Design: A DFT Method for Asynchronous Circuits". *Asian Test Symposium*, 2010, pp. 206-211.

[JAY06] Jayakuma, N.; Garg, R.; Gamache, B.; Khatri, S. "A PLA Based Asynchronous Micropipelining Approach for Subthreshold Circuit Design". In: *Design Automation Conference*, 2006, pp. 419-424.

[JHA03] Jha, N.; Gupta, S. "Testing of Digital Systems". Cambridge, UK: Cambridge University Press, 2003, 1016p.

[JUR17] 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 (VLSI) Systems*, vol. 25-2, Feb 2017, pp. 765-768.

[JUR18a] Juracy, L. R. "A Design for Testability Methodology for Blade Resilient Asynchronous Template: A Structural Approach", Master Dissertation, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2018, 101p.

[JUR18b] Juracy, L. R.; Moreira, M. T.; Kuentzer F. A.; Amory, A. M. "Q-Flop: A Scannable Metastability-free Memory Element". *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, accepted for publication, 2018.

[KAN99] Kang, Y.-S.; Huh, K.-H.; Kang, S. "New Scan Design of Asynchronous Sequential Circuits". In: *Asia Pacific Conference on ASICs*, 1999, pp. 355-358.

[KAP08] Kaps, J.-P. "Chai-Tea, Cryptographic Hardware Implementations of xTEA". Berlin, Germany: Springer Berlin Heidelberg, 2008, pp. 363–375.

[KHO94] Khoche, A.; Brunvand, E. "Testing Micropipelines". In: *International Symposium on Asynchronous Circuits and Systems*, 1994, pp. 239-246.

[KHO95] Khoche, A.; Brunvand, E. "A Partial Scan Methodology for Testing Self-Timed Circuits". In: *VLSI Test Symposium*, 1995, pp. 283-289.

[KIM15] Kim, S.; Seok, M. "Variation-Tolerant, Ultra-Low-Voltage Microprocessor With a Low-Overhead, Within-a-Cycle In-Situ Timing-Error Detection and Correction Technique". *Journal of Solid-State Circuits*, vol. 50-6, Jun 2015, pp. 1478-1490.

[KIN04] King, M.; Saluja, K. "Testing Micropipelined Asynchronous Circuits". In: *International Test Conference*, 2004, pp. 329-338.

[KIS98] Kishinevsky, M.; Kondratyev, A.; Lavagno, L.; Taubin, A. "Partial-Scan Delay Fault Testing of Asynchronous Circuits". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17-11, Nov 1998, pp. 1184-1199.

[KRA05] Kranitis, N.; Paschalis, A.; Gizopoulos, D.; Xenoulis, G. "Software-Based Self-Testing of Embedded Processors". *IEEE Transactions on Computers*, vol. 54-4, Apr 2005, pp. 461–475.

[KRS03] Krstić, A.; Liou, J.-J.; Cheng, K.-T.; Wang, L. C. "On Structural vs. Functional Testing for Delay Faults". In: *International Symposium on Quality Electronic Design*, 2003, pp. 438-441.

[KRS98] Krstić, A.; Cheng, K.-T. "Delay Fault Testing for VLSI Circuits". New York, USA: Springer, 1998, 191p.

[KUE16] Kuentzer, F. A.; Amory, A. M. "Fault Classification of the Error Detection Logic in the Blade Resilient Template". In: *International Symposium on Asynchronous Circuits and Systems*, 2016, pp. 37-42.

[KUE18] Kuentzer, F. A.; Juracy, L. R.; Amory, A. M. "On the Reuse of Timing Resilient Architecture for Testing Path Delay Faults in Critical Paths, In: *Design, Automation and Test in Europe*, 2018, pp. 379-384.

[KWO14] 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". *Journal of Solid-State Circuits*, vol. 49-9, Sep 2014, pp. 2054-2066.

[LAI00] Lai, W.-C.; Krstić, A.; Cheng, K.-T. "Test Program Synthesis for Path Selay Faults in Microprocessor Cores". In: *International Test Conference*, 2000, pp. 1080–1089.

[LEE10] Lee, H.; Paik, S.; Shin, Y. "Pulse Width Allocation and Clock Skew Scheduling: Optimizing Sequential Circuits Based on Pulsed Latches". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29-3, Mar 2010, pp. 355-366.

[LEV86] Levendel, Y.; Menon, P. R. "Transition Faults in Combinational Circuits: Input Transition Test Generation and Fault Simulation". In: *International Fault Tolerant Computing Symposium*, 1986, pp. 278-283.

[MOL88] Rosenberger, F.; Molnar, C.; Chaney, T.; Fang, T.-P. "Q-modules: Internally Clocked Delay-Insensitive Modules". *IEEE Transactions on Computers*, vol. 37-9, Sep 1988, pp. 1005-1018.

[MOU00] Mourad, S.; Zorian, Y. "Principles of Testing Electronic Systems". New Jersey, USA: John Wiley & Sons, 2000, 440p.

[PAG92] Pagey, S.; Venkatesh, G.; Sherlekar, S. "Issues in Fault Modeling and Testing of Micropipelines". In: *Asian Test Symposium*, 1992, pp. 107-111.

[PEE10] Peeters, A.; te Beest, F.; de Wit, M.; Mallon, W. "Click Elements: An Implementation Style for Data-Driven Compilation". In: *International Symposium on Asynchronous Circuits and Systems*, 2010, pp. 3-14.

[PET95a] Petlin, O. A.; Furber, S. B. "Scan Testing of Asynchronous Sequential Circuits". In: *Great Lakes Symposium on VLSI*, 1995, pp. 224-229.

[PET95b] Petlin, O. A.; Furber, S. B. "Scan Testing of Micropipelines". In: *VLSI Test Symposium*, 1995, pp. 296-301.

[PET97] Petlin, O. A.; Furber, S. B. "Built-in Self-Testing of Micropipelines". In: *International Symposium on Asynchronous Circuits and Systems*, 1997, pp. 22-29.

[PLA14] OpenCores, "Plasma CPU". Available at: http://opencores.org/project/plasma, Mar 2018.

[PRA97] Pramanick, A. K.; Reddy, S. M. "On the Fault Coverage of Gate Delay Fault Detecting Tests". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16-1, Jan 1997, pp. 78-94.

[RED09] Wunderlich, H. J. "Models in Hardware Testing". Netherlands: Springer, 2009, 257p.

[RON96a] Roncken, M.; Bruls, E. "Test Quality of Asynchronous Circuits: A Defect-Oriented Evaluation". In: *International Test Conference*, 1996, pp. 205-214.

[RON96b] Roncken, M.; Aarts, E.; Verhaegh, W. "Optimal Scan for Pipelined Testing: An Asynchronous Foundation". In: *International Test Conference*, 1996, pp. 215-224.

[RON99] Roncken, M. "Defect-Oriented Testability for Asynchronous ICs". *Proceedings of the IEEE*, vol. 87-2, Feb 1999, pp. 363-375.

[RON15] Roncken, M.; Gilla, S. M.; Park, H.; Jamadagni, N.; Cowan, C.; Sutherland, I. "Naturalized Communication and Testing". In: *International Symposium on Asynchronous Circuits and Systems*, 2015, pp. 77-84.

[RUS89] Russell, G.; Sayers, I. L. "Advanced Simulation and Test Methodologies for VLSI Design". Netherlands: Springer, 1989, 378p.

[SAT15] Sato, S.; Ohtake, S. "A Delay Measurement Mechanism for Asynchronous Circuits of Bundled-Data Model". In: *International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2015, pp. 243-248.

[SHI05] Shi, F.; Makris, Y.; Nowick, S. M.; Singh, M. "Test Generation for Ultra High-Speed Asynchronous Pipelines". In: *International Test Conference*, 2005, pp. 1009-1018.

[SIN06] Singh, V.; Inoue, M.; Saluja, K. K.; Fujiwara, H. "Instruction-Based Self-Testing of Delay Faults in Pipelined Processors". *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14-11, Nov 2006, pp. 1203–1215.

[SIN07] Singh, M.; Nowick, S. "Mousetrap: High-speed Transition Signaling Asynchronous Pipelines". *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15-6, Jun 2007, pp. 684-698.

[SMI85] Smith, G. L. "Model for Delay Faults Based upon Paths". In: *International Test Conference*, 1985, pp 342-349.

[SPA01] Sparsø, J.; Furber, S. "Principles of Asynchronous Circuit Design - A Systems Perspective". New York, USA: Springer, 2001, 337p.

[SUT89] Sutherland, I. E. "Micropipelines". *Communications of the ACM*, vol. 32-6, Jun 1989, pp. 720-738.

[SUT01] Sutherland, I.; Fairbanks, S. "GasP: A Minimal FIFO Control". In: *International Symposium on Circuits and Systems*, 2001, pp. 46-53.

[TAD16] Tadros, R.; Hua, W.; Moreira, M.; Calazans, N. L. V; Beerel, P. A. "A Low Power, Low Area Error Detecting Latch for Resilient Architectures 28nm FDSOI". *IEEE Transactions on Circuits and Systems II*, vol. 63-9, Sep 2016, pp. 858-862.

[TEH08] Tehranipoor, M.; Ahmed, N. "Nanometer Technology Designs: High-Quality Delay Tests". New York, USA, Springer, 2008, 281p.

[TEH11] Tehranipoor, M.; Peng, K.; Chakrabarty, K. "Test and Diagnosis for Small-Delay Defects". New York, USA: Springer, 2011, 2012p.

[TSC09] Tschanz, J.; Bowman, K. "Resilient Circuits - Enabling Energy-Efficient Performance and Reliability". In: *International Conference on Computer-Aided Design*, 2009, pp. 71-73.

[WAN06a] Wang, L.-T.; Wu, C.-W.; Wen, X. "VLSI Test Principles and Architectures". San Francisco, USA: Morgan Kaufmann, 2006, 808p.

[WAN06b] Wang, N. J.; Patel, S. J. "Restore: Symptom-Based Soft Error Detection in Microprocessors". *IEEE Transactions on Dependable and Secure Computing*, vol. 3-3, Jul 2006, pp. 188–201.

[WAU17] Waugaman, M.; Koven, W. "Sharp - A Resilient Asynchronous Template". In: *International Symposium on Asynchronous Circuits and Systems*, 2017, pp. 83-84.

[WOJ14] Wojcicki, T. "VLSI: Circuits for Emerging Applications". Boca Raton, USA: CRC Press, 2014, 486p.

[YI14] Yi, H.; Yoneda, T.; Inoue, M. "A Scan-Based On-Line Aging Monitoring Scheme". *Journal of Semiconductor Technology and Science*, vol. 14-1, Feb 2014, pp. 124–130.

[YUA13] Yuan, F.; Liu, Y.; Jone, W.-B; Xu, Q. "On Testing Timing-Speculative Circuits". In: *Design Automation Conference*, 2013, pp. 1-6.

[YUN92] Yun, K.; Dill, D.; Nowick, S. "Synthesis of 3D Asynchronous State Machines". In: *International Conference on Computer Design*, 1992, pp. 346-350.

APPENDIX A – CLICK CONTROLLERS

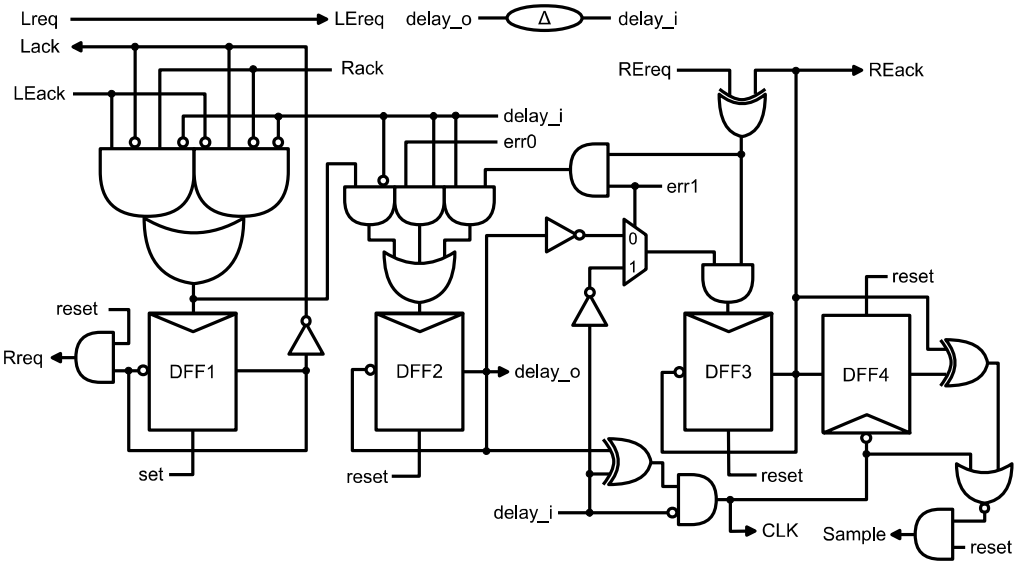


Figure A.1 – Click-based token controller for EDL stages.

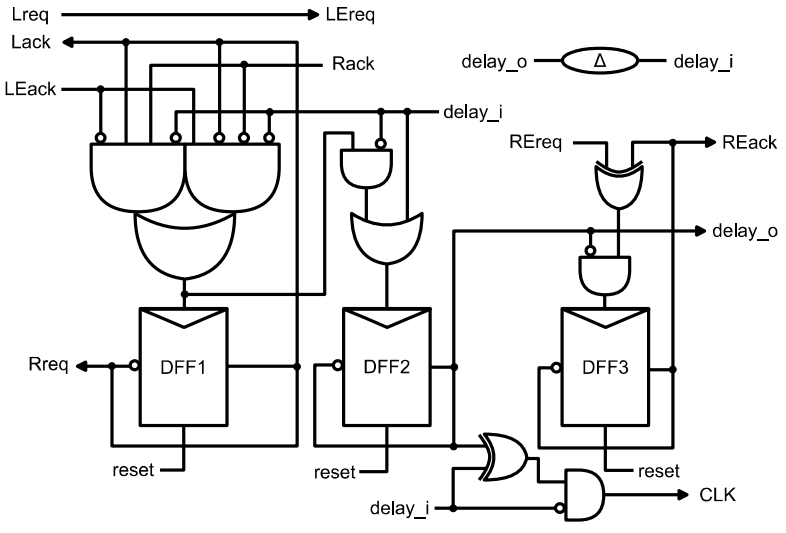


Figure A.2 – Click-based controller circuit.

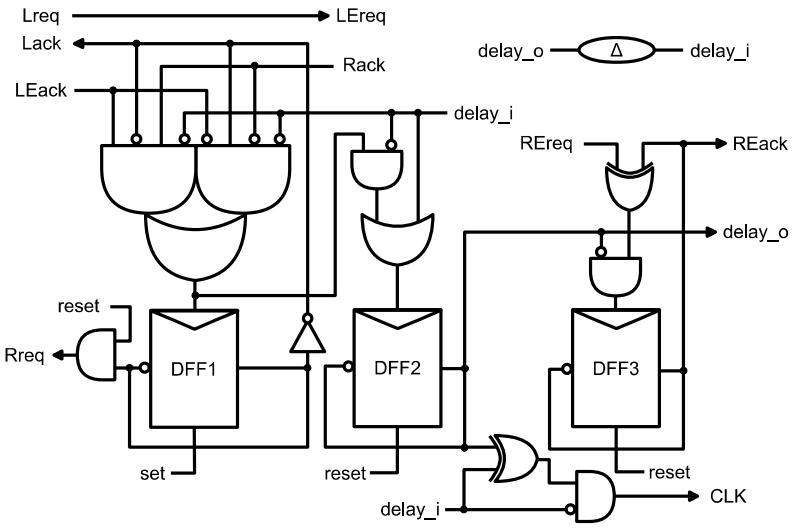


Figure A.3 – Click-based token controller circuit.

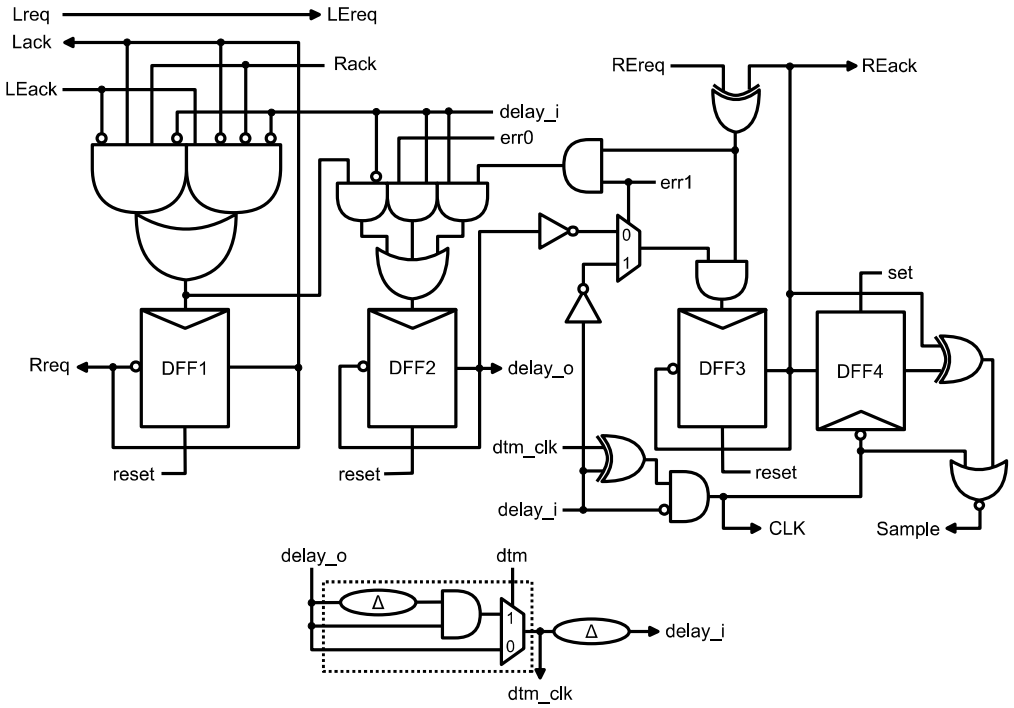


Figure A.4 – Click-based controller for EDL stages modified for the delay test mode (*dtm*).

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br