test and side-channel analysis of asynchronous circuits

Ricardo Aquino Guazzelli

Université Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Nano Electronique et Nano Technologies (NENT)**

Arrêtée ministériel : 25 mai 2016

Présentée par

Ricardo AQUINO GUAZZELLI

Thèse dirigée par **Laurent FESQUET**

préparée au sein du **Laboratoire Techniques de l'Informatique et de la Microélectronique pour l'Architecture des systèmes intégrés (TIMA)**
dans **l'École Doctorale Electronique, Electrotecnique, Automatique & Traitement du Signal (EEATS)**

Test and Side-channel Analysis of Asynchronous Circuits

Thèse soutenue publiquement le **3 décembre 2020**,
devant le jury composé de :

Laurent FESQUET
Maître de Conférences, Université Grenoble Alpes, Directeur de thèse

Bruno ROUZEYRE
Professeur des Universités, Université de Montpellier, Examinateur

Haralampos STRATIGOPOULOS
Directeur de Recherche, Sorbonne Université, Rapporteur

Alberto BOSIO
Professeur des Universités, École Centrale Lyon, Rapporteur

Giorgio DI NATALE
Directeur de Recherche, Université Grenoble Alpes, Président

"Se eu soubesse antes o que sei agora, erraria tudo exatamente igual."
Humberto Gessinger

Acknowledgements

I would like to thank to Giorgio Di Natale, Alberto Bosio, Haralampos Stratigopoulos and Bruno Rouzeyre for accepting the invitation to compose my thesis jury. I appreciate the remarks from Alberto and Haralampos, who participated as *rapporteurs* (referees) in my thesis and all comments and questions invoked during my defense.

Here, I take a brief moment to acknowledge the commitment that my advisor Laurent Fesquet put on in order to see this thesis finished. I am quite aware that I allocated a reasonable amount of your time in some moments during this journey and this fact highlights the effort you have gave to help me. Without no doubt, your technical expertise gave me a proper direction, allowing me to pursuit a research with meaning, and your support put on track everything I was doing in the last three years. I really cannot express enough thanks for all the technical and administrative help you gave to me during the last years, and I hope can retributive that someday.

Moreover, two other people were extremely important in my professional training and thus must be acknowledged here: Ney Calazans, who introduced to the the research on asynchronous design when I was in engineering school and since then I am trapped; and Matheus Moreira, who was always available to answer my questions and discuss the most wide range of subject. I own you for the microelectronic background I currently have and it was because of that I am here where I am now. Thank you immensely.

Obviously, I have to thank my colleges at TIMA, who had to put up with me: Matheus, my *bro*, for all scientific discussions, ski days, beers, laughs and LAN parties; Thiago, for helping me during the first moments here in France and being such an iconic friend; Leonel, who also was here to help in the beginning, giving me a random nickname; Renato, who I did not in fact interact a lot in the first year of the thesis, but destiny brought us together as friends after catastrophic sequence of events; Yoan, who had to work with me to finish a testchip during the first COVID lockdown and spent countless hours with me discussing about asynchronous design, politics, goat cheese; Gregoire, who was vital for the testchip and had the patience to explain to me how LCS works; Pudu (a.k.a. Rodrigo Iga), for your friendship, mindful discussions and all invitations to go to the mountains and go skiing; Mohammed, one of the most interesting PhD students of the CDSI team (and possible of TIMA), for giving to us another perspective of life and providing such unique (and funny) moments during *pause-café* and happy hours; Medhi, for the advice to calm me down before the defense; Jérémy and its supply of comté; and Assia, Jean, Nils and Liège for all the moments in the lab and good discussions during the *pauses-café*.

I also say thank to my Grenoble friends: Raupp and Vivian, for hosting me at my first day in Grenoble; Katyanne, for being such an incredible friend to me while tolerating my complaints about life; Natália, for the good talks and invitations to eat proper food from Minas, specially pão de queijo; Laura, for all the moments together, being able to talk and laugh even with the curveballs that life keep throwing at us; and Luis (a.k.a. Cocotas), for

all your stories and sense of humor.

I cannot pass this through without saying thank to my friends who accompanied me even thousands of kilometers away: Guilherme (a.k.a Gnomo), for your time, friendship and ability to piss me off; Marcelo (a.k.a Gordo), who started engineering school with me at 2009 and I still can't get rid of him; and the international couples: Ana and Gibiluka, Douglas and Carol, and Bruno and Karine.

Another important person I have to mention here is my lovely Caroline, who helped me go through the final lap of my thesis and has been an incredible partner since the beginning. I thank you for your care and and for being my "guinea pig" when I cook.

The best for the last, I would like to express my gratitude to my family: Mom and dad, for all your love and care with me all these years; All my character and moral concepts are inherited from you and I appreciate everything you taught along the years; Gui, for annoying me all the time but also being an unique brother. Your presence in my life surely shaped me in who I am now; and finally, my incredible grandma Teresinha Xavier Aquino, who I had the honor to live with when I moved to Porto Alegre. You are the reason of me chasing a better education and reach objectives I would never think about before. I could say it is not a coincidence that I am living in France now, isn't it?

Contents

Introduction

For several decades, the semiconductor market has managed to reduce the minimum feature size of transistors and wires, which leads to increasing density and cost-effectiveness to *Integrated Circuit*s (IC), among several other benefits. These technology nodes are still pushing the physical limits of IC design, enabling the fabrication of devices with billions of transistors. Apart from the benefit of reducing the minimum geometry, higher performance and lower power can be obtained due to the lower resistance and capacitance of smaller transistors and wires [1]. However, these advances bring huge challenges to circuit and *Computer Aided Design* (CAD) tool designers. As the transistor minimum feature size approaches fundamental atomic limits, electronic devices gradually behave less and less as ideal switches, and wires behave less and less as ideal electrical connections with negligible delay and impedance. In addition, increased manufacturing parameter variations bring uncertainties to the processes of estimating and/or predicting the timing and power characteristics of circuits.

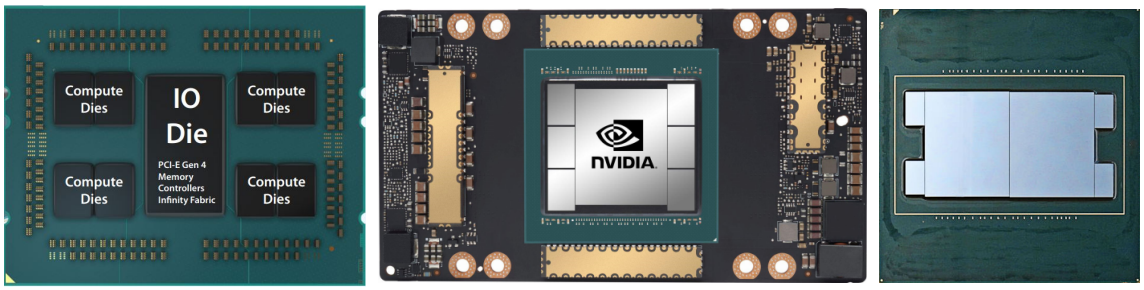

Figure 1: Top view of (a) 2nd generation Epyc, (b) GA100 and (c) Stratix chips.

As this thesis is written, semiconductor foundries already provide *Complementary Metal Oxide Semiconductor* (CMOS) technologies with nodes reaching below 10 nm. Among them, it is possible to cite TSMC's 7-nm [2] and Samsung's 8-nm [3] technologies, which are in mass-production phase. In fact, the commercial use of these technologies is a reality as the market starts to introduce new start-of-the-art products in wide range of applications. In 2019, AMD introduced the 2nd generation of the Epyc processor, containing around 40 billion transistors. With the AMD's Zen micro-architecture, the processor employs nine different dies, where eight dies are x86-based cores and the remaining one is a dedicated for I/O communication – see Figure 1 (a). Taking a look on *Graphic Processor Unit*s (GPU), Nvidia recently introduced the Ampere architecture.

Labeled as A100, the new GPUs target not only graphical applications but also *High-Performance Computing* (HPC) and *Artificial Intelligence* (AI) applications. Nvidia reports that the A100 has been designed with TSMC's 7-nm technology, and its larger design has around eight thousand *Compute Unified Device Architecture* (CUDA) cores and 6 stacks of *High-Bandwith Memory 2* (HBM2) memory, reaching astonishing 40 GB memory space. Figure 1 (b) shows the HBM2 stacks next to the main GPU core, three on each side. All this logic complexity is implemented with around 54 billion transistors. As a final example, in Figure 1 (c), Intel has presented the Stratix 10 GX 10M in 2019, a double-die *Field-Programmable Gate Array* (FPGA) employing 43 billion transistors targeting *Application-Specific Integrated Circuit* (ASIC) prototyping and emulation. Its design has around 10.2 million programmable logic blocks and it is the largest FPGA design yet reported. This small set of start-of-the-art ICs in the market shows clearly how far the density and complexity of current IC designs have reached.

In parallel, as our society continues its path through the information era, new political, economic and ecological trends impact the technology decisions in the semiconductor markets. The AI trend has pushed forward the research and development of machine learning solutions on software and hardware level. In fact, public institutions such as the french government are considering the AI "*savoir-faire*" as a national research strategy. Consequently, semiconductor suppliers are now interested to develop dedicated *Intellectual Property*s (IP) and ICs, providing optimized solutions for autonomous vehicles, automated medical diagnosis, voice input for human-computer interaction, intelligent agents, automated data synthesis, enhanced decision-making and many other applications. Another interesting trend that remains on discussion is the *Internet of Things* (IoT). Different from high-performance applications, IoT applications such as distributed sensor networks and wearable devices define *Ultra-Low-Power* (ULP) consumption and energy efficiency as main design constraints. On top of that, other traditional applications that considered performance as the main design target now have been reviewed to focus on energy efficiency. For instance, data centers consume an estimated 200 TWh each year [4] and companies already reported efforts to make their data centers more power efficient with customized high-performance servers, trying to reduce the power overhead with the higher demand of information processing and storage. These efficiency-driven applications are pushing designers to rethink their design concepts and search for alternatives.

Nowadays, the predominant digital circuit design style adopted by the industry is synchronous. This style takes as fundamental assumption that all components share a common and discrete notion of time, which is guaranteed by the use of a global clock signal distributed throughout the circuit. Figure 2 illustrates a generic synchronous pipeline architecture, controlled by a global clock signal. The clock signal controls every sequential element *Reg* in the design, typically *Flip-Flop*s (FF) and/or latches. The value stored in these elements can only change when the clock switches its logic level in a given direction, or when it remains in a given active state. This is what enables the design of

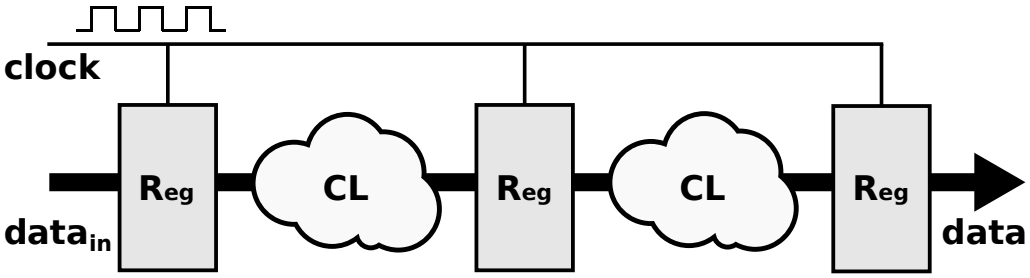

Figure 2: Generic synchronous pipeline architecture. Data flows through pipeline stages according to the clock signal pulses.

sequential blocks to deal with time as a discrete variable, allowing data to flow from one register to the next as the clock signal activates. This characteristic enables designers to ignore wire and logic gate delays, as long as the worst case delay between two registers is never longer than the period of the clock signal controlling them [5]. However, despite the fact that synchronous design has abundant CAD support and is familiar to most designers, it also brings challenges with regard to clock signal distribution, skew and power consumption. On top got that. the current level of precision required on manufacturing operating conditions finally results in substantial variations on the electrical characteristics of fabricated devices, which in turn can lead to significant delay and power consumption variations. To cope with these problems, synchronous designs require margins in the period of the clock signal, which leads to increasing costs in performance, power, area and design time. These margins can indeed become the Achille's heel of synchronous design. For example, 13 years ago Brej [6] stated that industrial circuits could require up to 130% of overhead in the clock period due to the summation of all needed margins, and the situation has only become worse since then. For the sake of illustration, Figure 3 illustrates the delay components contributing to the overall delay in modern digital designs.

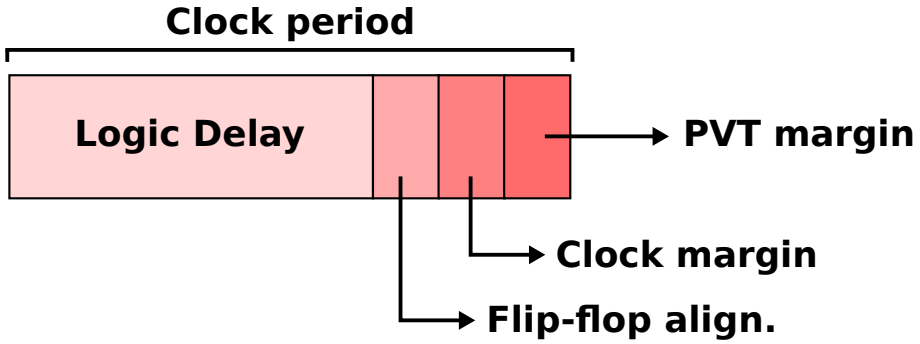

Figure 3: Illustration of the delay components for clock period definition in modern technologies. Designers must not only consider the traditional worst-case logic delay, but also margins regarding FF alignment, the clock tree and Process, Voltage and Temperature (PVT).

Asynchronous circuits are an alternative to overcome issues faced by contemporary

synchronous designers. Unlike the synchronous paradigm main assumption, the design of these circuits do not rely on a discrete notion of time. In this way, the global clock signal is exchanged for local handshaking channels that are added between adjacent memory elements, which establish the synchronization, communication and sequencing of operations [7]. This fundamental assumption of local synchronization avoids clock-related problems and overheads, at the cost of extra hardware for local synchronization. The literature has presented in the last decades the adoption of asynchronous design on a wide variety of applications, going from high-speed to low-power implementations. Table 1 summarizes the semiconductor companies that considered or are still exploring asynchronous circuits as their main market weapon. Founded in 2000, Fulcrum entered in the market providing asynchronous-based Ethernet switch chips, focused on high-speed Ethernet networks. Their design brought such attention that Intel purchased Fulcrum in 2011, trying to grow their presence in the network market. Philips Semiconductors was another company that invested on asynchronous design as well. Their research started in 1986 with the development of asynchronous *Electronic Design Automation* (EDA) tools (Tangram) [8] and, in 2004, Handshake Solutions was founded in order to offer services inside and outside Philips. Hanshake Solutions had reported low-power and low *Electromagnetic* (EM) emission benefits from their technology, targeting smartcard, automative and wireless applications. Still taking in consideration low-power implementations, ETA compute reported an asynchronous dual core ARM Cortex-M3, targeting near-threshold operation with *Dynamic Voltage Scaling* (DVS) scenarios. At the same as the beginning of Handshake Solutions, Achronix introduced the first asynchronous FPGAs [9] in the market, with the argument that they could provide designs operating on higher frequencies and lower power consumption than other FPGA vendors. Achronix still remains but its asynchronous design was being lost through the years as engineering teams were still not prepared to face the "asynchronous" challenge. In France, the asynchronous portfolio expanded with Tiempo [10], which provides robust and secured IPs for mostly security applications. With the recent AI trend, both Intel and IBM implemented neuromorphic chips with asynchronous features. For Intel's case, the Loihi chip [11] considers a fully asynchronous design for spiking neural networks. Regarding IBM's chip [12], the chip implements a fully event driven digital mixed synchronous-asynchronous neuromorphic architecture. Around 2015, a startup called REM [13] also took the asynchronous approach to design neural network architectures, focusing on visual intelligence applications. Finally, ChronosTech has recently proposed the use of asynchronous circuits to enhance *System-on-Chip* (SoC) integration [14]. Their solutions mainly target high-speed and robust *Network-on-Chip* (NoC) interconnections.

Asynchronous circuits are a promising solution for coping with aggressive *Process Variations* (PV) faced in the most advanced silicon technology nodes, as they are able to gracefully accommodate wide ranges in gate and wire delays. A similar phenomenon also takes place in more classical technologies when the performance is not an issue and the

Table 1: Semiconductor enterprises that adopted asynchronous design in their products.

Application	Year	Enterprise
Neuromorphic Design	2016	IBM [12]
	2017	Intel [11]
SoC Interconnections	2015	ChronosTech [14]
Low power	2004	Handshake Solutions [8]
	2014	REM [13]
	2015	ETA Compute [15]
Security	2007	Tiempo [10]
FPGA	2004	Achronix [9]
High-speed Ethernet	2000	Fulcrum [16]

requirements are mainly driven by power consumption considerations. Indeed, in order to drastically reduce power consumption, power management strategies tend to minimize or aggressively shrink the power supply voltage. In such conditions, the PVs are exacerbate because the operating voltage is near-threshold or, worst, sub-threshold. Operating at very low-voltage makes sense, especially with IoT, where the devices may have strong low-power requirements. Most of the ULP circuits operate at very low supply voltages and sometimes in electrical harsh environments, implying less predictable propagation delays and noisy working environments [17]. Delay variations can compromise the circuit functionality, especially if the timing assumptions are strong, which is the case for the synchronous circuits. Indeed, their timing assumption is based on the worst circuit critical path, pushing the designers to over-design and take excessive timing margins. Hence, asynchronous circuits can help designers to avoid such excessive margins by providing an easier timing closure and improving robustness against unexpectedly large delays [18]. The most efficient circuit class to tackle this problem is certainly the *Quasi-Delay Insensitive* (QDI) circuit class because it only requires a very weak assumption on some circuit forks (known as isochronic forks) [7]. Nevertheless, this class suffers from a large circuit area and requires specific skills and dedicated tools, making its adoption more difficult by the industry. In order to overcome these issues, designing *Bundled-Data* (BD) circuits seems more acceptable because they have a similar area compared to the synchronous circuits, while offering a better robustness to process and voltage variations. As they are really good candidate for playing an important role in low power and ULP circuits, it is important to propose effective solutions for testing the imposed timing constraints of such circuits after fabrication or even in their working environment. Moreover, testing

circuits is a mandatory requirements in digital circuits and at-speed becomes especially crucial when the supply voltage is low or changed during operations. As bundle-data circuits are technically close to their synchronous counterparts, they are more comprehensive for the designers and can benefit from the commercial EDA tools usually employed with synchronous design [19]. Based on the existing synchronous *Design-for-Testability* (DfT) approaches, it is possible to develop a similar framework for asynchronous circuits, which could also take advantage of the commercial tools. Thus, testing asynchronous bundle-data circuits requires a limited effort to make them compliant with EDA tools. It is of course needed to adapt the DfT strategy to their specificity.

In this context, this thesis targets dedicated techniques for testing and analyzing asynchronous circuits. For the sake of organization, the following chapters are divided in three main parts. Part I provides the foundation of this thesis: asynchronous design. In a single chapter, the basic concepts of asynchronous circuits are covered, including design considerations about BD and QDI implementations. Part II, called "*At-speed Testing for Bundled-data Circuits*", focuses on the problematic of delay testing on asynchronous BD circuits. It provides basic concepts of digital testing, DfT and an overview of the solutions that the literature have previously presented regarding the matter – see chapter 2. Moreover, in chapter 3, it presents the first contribution of this thesis: the proposed DfT architecture for BD design. The proposed architecture targets to enable path-delay testing on micropipeline-based circuits, whereas taking consideration the compatibility of synthesis and *Automatic Testing Pattern Generation* (ATPG) tools. Part III explores side-channel analysis on asynchronous circuits for *Hardware Trojan* (HT) detection. Initially, chapter 4 presents the state of the art regarding the HT threat and the available detection techniques. Next, chapter 4 contains the proposed HT detection technique, which is the second contribution of this thesis. The proposed technique leverages the intrinsic current characteristics of asynchronous circuits, allowing to detect the presence of HTs through current anomalies.

Part I

Asynchronous Circuits

1 Asynchronous Design

Contents

This chapter covers the concepts for asynchronous design, which are essentials for this thesis. Regarding basic concepts, it presents the concept of asynchronous channels and the types of handshake protocols. This chapter also covers the use of the iconic C-element gate – essential for most of the state-of-the-art asynchronous implementations – and its used in handshake structures. Moreover, it is presented the two major asynchronous design styles: BD and QDI, pinpointing their main characteristics and practical implementations available in the literature.

Due to its simplicity, the synchronous paradigm is heavily applied in the industry for designing digital circuits. This simplification is achieved thanks to the use of a clock signal, which provides temporal reference and synchronization among memory elements present in the circuit. Taking exactly the opposite direction, the asynchronous paradigm assume the absence of a global or regional clock signal in the circuit. Consequently, the clock signal is replaced by asynchronous channels that employ handshake protocols between sequential components to ensure synchronization and communication [7]. This means the synchronization occurs locally between memory elements (or registers) and with dedicated circuitry that explicitly signalizes sending and receiving data. This behavior matches to registers only clocked when needed – if we take a "synchronous" perspective.

1.1 Asynchronous Channels and Handshake Protocols

Asynchronous design usually employs a hierarchical network of blocks, which are interconnected via asynchronous channels [1]. Mainly, an asynchronous channel comprises on three aspects: (i) it contains a bundled of wires, (ii) employs a protocol to synchronize computation and data between blocks and (iii) is uni-directional and typically point-to-point [1]. Figure 1.1 illustrates a basic asynchronous structure between two blocks that are interconnected with an asynchronous channel with two control signals: the request *req* signal and the acknowledgement *ack* signal. Here, the active block sends a 'request' to synchronize with the passive block, which acknowledges when it is ready to communicate.

These interactions between active and passive blocks depends on the selected handshake protocol. Usually, asynchronous channels can employ two main handshake protocols: four-phase or two-phase. Figure 1.2 shows the behavior of the control signals for these protocols during two communication cycles.

Considering the four-phase protocol, the active block initiates the communication by rising *req*. Many authors in the literature refers to this request flag as 'token', as it carries

[1] If there is a bi-directional data communication between circuit blocks, it is necessary to employ two channels in opposite directions.

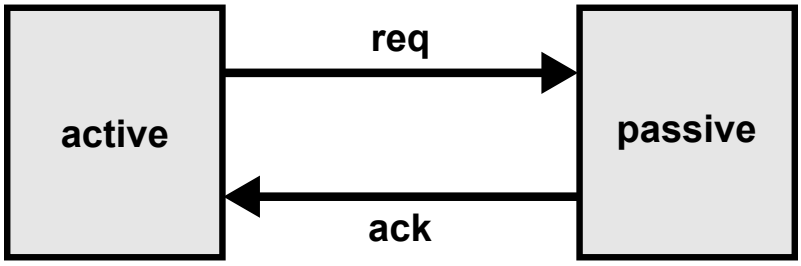

Figure 1.1: Example of a control-only asynchronous channel with two basic control signals (req and ack).

the information that valid data is present. Next, the passive block detects the request and set the *ack* signal. When the active block detects the acknowledgement, the *req* is lowered – informing that the active block has finished the communication. The passive block then acknowledges by lowering the *ack* signal as well, finishing the communication cycle. At this point, the active block can propagate a new token.

For the two-phase protocol, a lower number of transitions are required to perform a complete communication cycle as indicated in Figure 1.2 (b). In this protocol, the active blocks send a token by switching the logic value of the *req* signal. When the passive block detects the request from the active block, it sends an acknowledgement by switching the logic value of the *ack* signal. In contrast to the four-phase protocol, the two-phase protocol has no distinction between rising and falling transitions of the control signals. It considers the transitions them-self of *req* and *ack* as the beginning and, respectively the end of the communication.

Numerous asynchronous design styles have been proposed in the last decades [7, 13, 20–32], covering different protocols and trade-offs such as performance, power, robustness, etc. Among them, it is possible to classify them into two major families: asynchronous hazard-free logic and asynchronous circuits tolerating hazards. The first family is mainly represented by QDI circuits and the second by BD circuits. Sections 1.3 and 1.4 cover the main aspects of BD and QDI design, respectively.

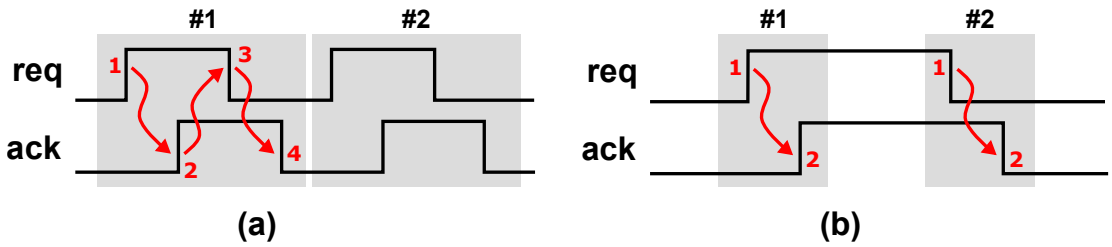

Figure 1.2: Control operation of (a) four-phase and (b) two-phase handshake protocols.

1.2 Handshake Implementation Concepts

This section presents the basic concepts of asynchronous design, giving a brief description of the C-element, its importance and use for handshake structures. Moreover, this section also provides information about non-linear structures and their respective implementation.

1.2.1 The C-element

One of the most iconic and important logic element presents in any asynchronous circuit is the Muller gate or C-element. Table 1.1 indicates the truth table of a 2-input C-element. When the inputs (A and B) have the same value, the C-element asserts its output Z_n to the same value. Otherwise, the C-element retains the previous value Z_{n-1} in the output. Due to this characteristic, any output change to high (low) in the C-element implies that all the inputs are high (low) as well. This plays an important role for event synchronization as it can avoid hazards and races in asynchronous controllers [7].

Table 1.1: Truth table of a two-input C-element gate.

A	**B**	Z_n
0	0	0
0	1	Z_{n-1}
1	0	Z_{n-1}
1	1	1

Figure 1.3 illustrates the (a) gate symbol and (b) a transistor-level implementation of a 2-input C-element. This C-element implementation is often called semi-static or Martin's weak feedback. The *reset* pull-up and *set* pull-down networks are responsible to assert the output Z_n to '0' and '1' respectively. The latch maintains the output value when *reset* and *set* networks are not directly driving the output. Different arrangements of C-elements have been proposed in the literature, including extra functionality such as reset / set logic and asymmetrical versions. Appendix A provides the schematics and descriptions of alternative C-element designs available in the literature.

Studying the concept behind the C-element, it is essential to understand the Muller gate pipeline as well. The intrinsic simplicity of the Muller gate pipeline allows to better understand handshaking functionality and its mechanism is the backbone of almost all asynchronous circuits [7]. Let's take the example of the 3-stage Muller pipeline in Figure 1.4 interconnecting an active and passive block. All of the C-elements have been initialized to '0' – all "local clock signal" clk_i reset – we suppose the active block propagates a token by rising the leftmost request signal req_0. Figure 1.5 illustrates the behavior

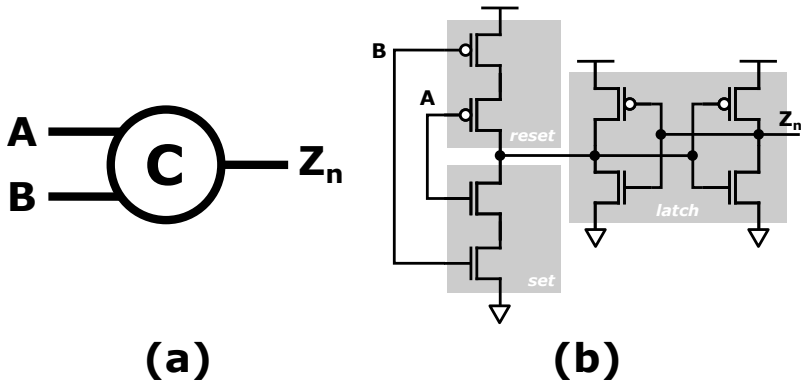

Figure 1.3: The Muller C-element: (a) gate view of a two-input gate and (b) equivalent implementation in transistor level.

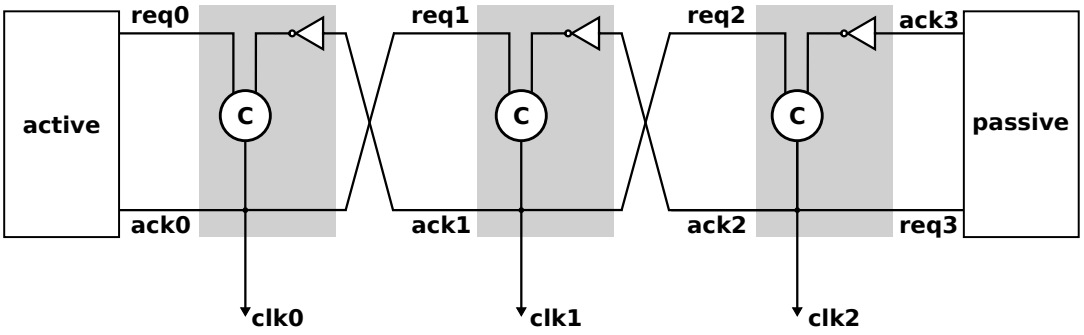

Figure 1.4: Active and passive block interconnected by a 3-stage Muller pipeline.

of all the signals clk_i during token propagation. The C-element C_0 only asserts its output to high if the successor C-element output C_1 is '0' and a token arrives at its input. When C_0's output (clk_0) rises, it also acknowledges back to the active block through ack_0 and propagates the token to C_1, which repeats the same procedure. The token propagates through all the C-element stages until it reaches the passive block. Considering the propagation of the first token in Figure 1.5, it is possible to see a wave created during the token propagation. Thus, the role of a C-element stage in the pipeline is to propagate signal waves in a carefully controlled way maintaining the integrity of each wave. However, if the passive block does not respond, the token stalls in the last stage, awaiting for acknowledgement. This happens in Figure 1.5 when the second token arrives in the last C-element stage and the passive block takes an extra time to rise ack_3. Consequently, the third token in the pipeline also stalls in the first C-element stage as the second stage is unable to process it. After the passive block acknowledges the second token, the pipeline resumes its operation and propagates the third token until the passive block. This highlights how the Muller gate pipeline can easily adapt its operation according to the delays of the external environments. In fact, the Muller pipeline is classified as a DI circuit [7]. On top of that, the pipeline implementations for the two-phase and four-phase protocols are identical. The only difference in the interpretation of the signals clk_i.

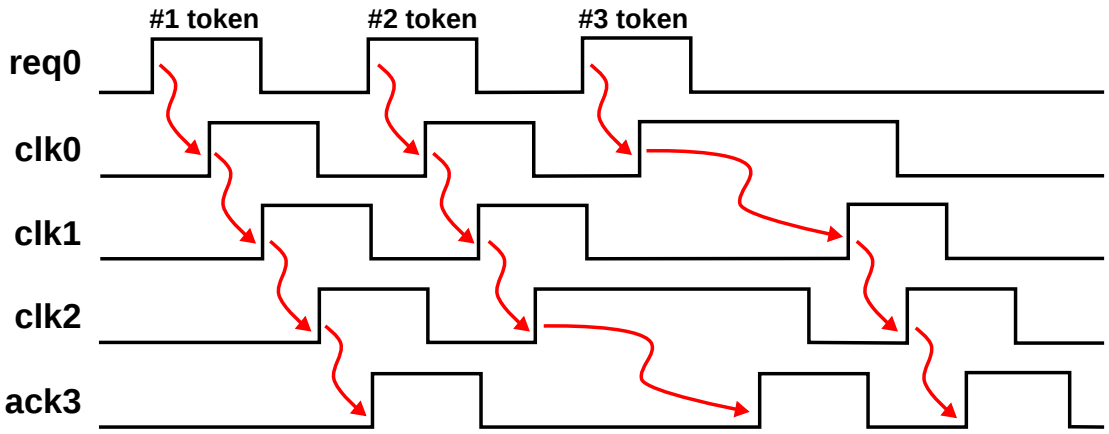

Figure 1.5: Handshake waveform during the propagation of three tokens by the active block. The waveform simulates an acknowledgement delay by the passive block between the first and second token.

1.2.2 Non-linear Structures

Realistic circuits usually employ more complex structures than linear structures, as the Muller gate pipeline depicted in Figure 1.4, and this also applies to asynchronous circuits as well. This implies that circuits need special flow schemes (or buffers as called by [1]) with multiple input and output channels. Asynchronous circuits may employ unconditional and conditional flow scheme [1, 7]. A flow scheme is called *unconditional* if it awaits for tokens on all the input channels before generating tokens on all the output channels. A *conditional* flow scheme generates token on its output channels according to an additional selection channel. The selection channel indicates which input (or output) channel must be selected for the token propagation. The minimum set of non-linear flow schemes necessary to implement most of the circuits is illustrated in Figure 1.6 with four main types: *fork*, *join*, *split*, *merge*. Their descriptions are below:

- *Forks* are unconditional flow schemes with one input channel and multiple output channels, which propagate any token at its input to all output channels;
- *Joins* are unconditional flow schemes with multiple input channels and a single output channel. In this case, the join only propagates a token to its output channel if it receives tokens in all input channels;
- *Splits* are conditional flow schemes with one input channel, one selection channel and multiple output channels. It synchronizes the input and selection channel, propagating a token to a specific output channel according to the token value in the selection channel. In fact, a split behaves as a DEMUX [2];

[2] Sparsø [7] presents in its book the split and merge flow schemes as DEMUX and MUX, respectively. However, this thesis considers only the definitions presented by Peter Beerel's book [1].

- *Merges* are conditional flow schemes with multiple input channel, one selection channel and a single output channel. The merge synchronizes and propagates a token from a specific input channel to the output channel according to the token value in the selection channel. As the merge implements the opposite of a split, it is possible to associate the merge behavior as a MUX.

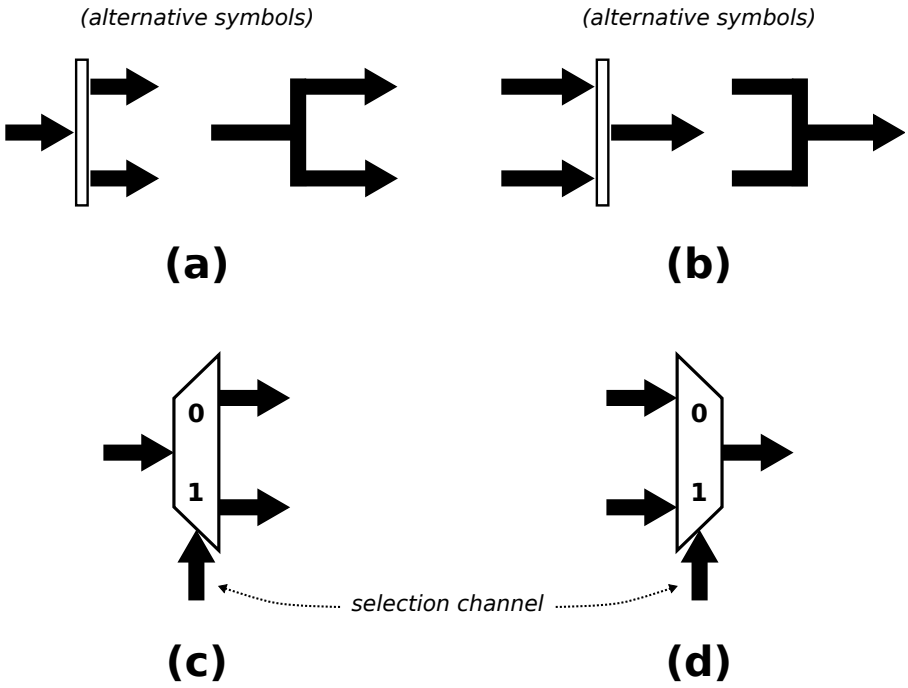

Figure 1.6: Symbols for (a) fork, (b) join, (c) split and (d) merge flow schemes.

1.3 Bundled-Data Channels

BD channels take an implementation approach close to standard synchronous design, employing single-rail data encoding but replacing the global clock signal by local handshake schemes. Figure 1.7 illustrates a generic asynchronous BD push channel, in which data flows through the data path according to the handshaking protocol between active/-passive blocks. BD channels can adopt four-phase or two-phrase handshake protocols, as depicted in Figure 1.8.

Considering four-phase handshaking, three different types of protocols can be adopted: *broad*, *early* and *late*. The broad four-phase handshaking requires that valid data must be stable in the channel during all four handshake signaling. This implies the circuit guarantees that the receiver can capture data at any time during the communication cycle. The early protocol only requires valid data between the two first handshake transitions. Taking the example in Figure 1.8 (a), the capture windows for the receiver is between both rising edges of *req* and *ack*. For the late protocol, however, valid data must be stable

in the channel between the last two handshake transitions. Thus, the capture window is between the falling edges of *req* and *ack* in Figure 1.8 (a). Independently of the type of the four-phase handshaking, both *req* and *ack* signals must be reset to end the communication cycle. Two-phase handshaking considers both rising and falling transitions of *req* to indicate the presence of new valid data, and both transitions of *ack* signify the data capture by the receiver. In that way, the receiver capture windows remains between *req* and *ack* transitions, as depicted in Figure 1.8 (b). Here, there is no necessity to reset the handshake signals as the transition in *ack* already ends the communication cycle.

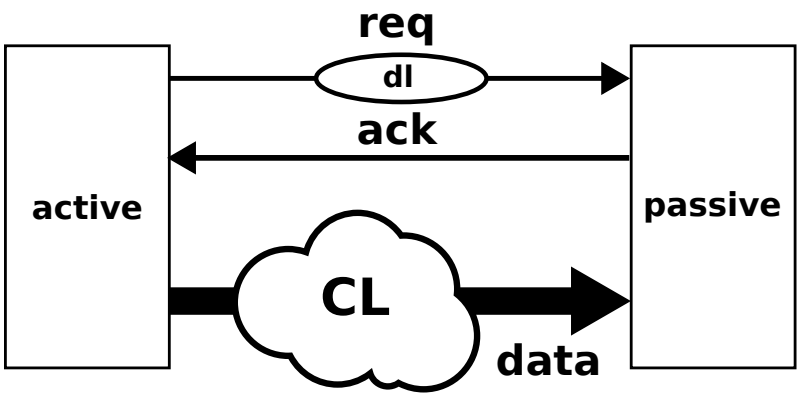

Figure 1.7: Generic scheme of a asynchronous bundled-data push channel.

Due to the handshake schemes, timing constraints between handshake and data signals must be respected in order to guarantee circuit correctness. According to [33], the basic constraints in bundled-data design are: (i) The passive block must not see a request until its input data is stable; (ii) The passive block must not acknowledge the active block until it has no further need for its input data and (iii) The active block must hold its output data steady between sending the *req* and receiving the *ack*. The presence of these timing constraints in any DB design requires the addition of a delay line *dl* on the signal *req* in order to match the worst-case delay of the combinational block *CL*.

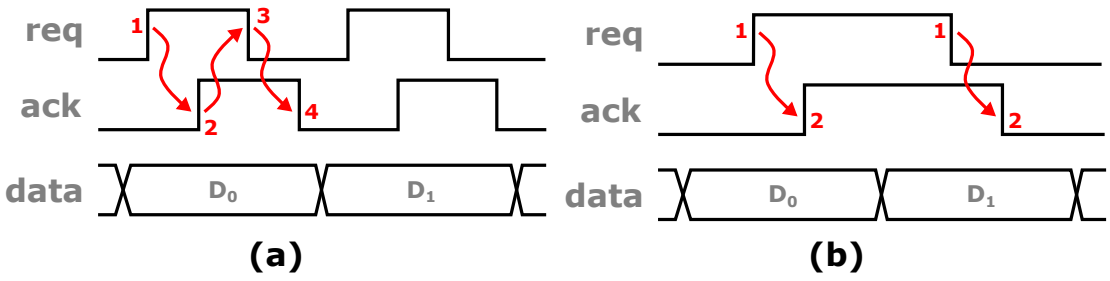

Figure 1.8: Asynchronous bundled-data channel employing (a) four-phase and (b) two-phase handshake protocols.

1.3.1 Bundled-Data Implementations

The definition of BD design gives some guidelines regarding handshake signals and architecture organization. However, these guidelines give space for designers to develop different BD implementations using different handshake protocols and features, allowing to focus on performance, power consumption, area optimization and even robustness. Table 1.2 gives an overview of all discussed BD implementations in this section, highlighting their main characteristics.

Table 1.2: List of available BD implementations in the literature.

Implementation	**Handshake Protocols**	**Registry**	**Design Strength**
Micropipeline [21]	Four- and two-phase	Flip-flop or latch	Performance
GasP [22]	Two-phase	Latch	Performance
Mousetrap [20]	Two-phase	Latch	Performance
Click [23]	Two-phase	Flip-flop	Timing closure and testing
Blade [24]	Two-phase	Latch	Resiliency and efficiency
Sharp [13]	Two-phase	Latch	Resiliency and efficiency
Early Ack [25]	Early four-phase	Flip-flop	Efficiency
Late Capture [26]	Late four-phase	Flip-flop	Area-aware
Maximus [26]	Late four-phase	Flip-flop	Performance and area-aware

One of the most iconic BD implementation in the literature is the *Micropipeline* [21]. Based on the Muller pipeline (further discussed in section 1.2.1), Sutherland presented the Micropipeline as a simple two-phase BD implementation. This implementation was marked as a milestone in the asynchronous community and several BD implementations later proposed use the basic concepts from the Micropipeline. Among these implementations, it is possible to cite *Mousetrap* [20] and *GasP* [22] implementation, both focused on high-performance design. In 2010, Philips presented the Click implementation, which employs flip-flops for both data and control paths. That was an interesting approach as previous implementations was latch-based and flip-flops facilitates the use of EDA tools for timing closure and testing. The literature also presents resilient BD implementations such as Blade [24] and Sharp [13], where the latter is a newer version of the first. Those implementations are able to detect and deal with timing violations during operations, enabling the circuit to operate faster than the worst-case path delays. Moreover, other implementations focus on optimizing the operation of 4-phase BD implementation. For

instance, Mannakkara et Yoneda presents in [25] a early 4-phase BD implementation called *Early Ack* [3]. Other authors present late 4-phase BD implementations such as the *Late Capture* and *Maximus* [26]. With these implementations, the authors takes advantages of the late capture windows of late four-phrase protocols to reduce the use of delay lines in the control path and, consequently, reduce the area overhead.

1.4 Quasi-Delay Insensitive Channels

An alternative to avoid the timing constraints imposed by asynchronous BD channels is to include the control within the data, which is the main strategy characterizing *Delay Insensitive* (DI) and QDI designs. DI design is not useful in practice to create large systems [34], but QDI design is! Moreover, it keeps most of the advantages of DI implementation [35] by adding a constraint on selected wire forks of the circuit, the so-called *isochronic forks* (see Section 1.4.1 for more details). Since isochronic forks can be limited in scope and designed to mostly exist inside the basic components (e.g. logic gates), QDI is a viable family of design techniques. In fact, QDI design is reported by Martin and Nyström and other authors as one of the most practical asynchronous design template, due to its relaxed timing constraints [35] [36]. Its delay insensitivity provides higher robustness against PVT variations, *Single-Event Effect* (SEE) and permits very low *Electromagnetic Interference* (EMI) implementations. On the downside, QDI implementations requires extra hardware, which can lead to significant area overhead.

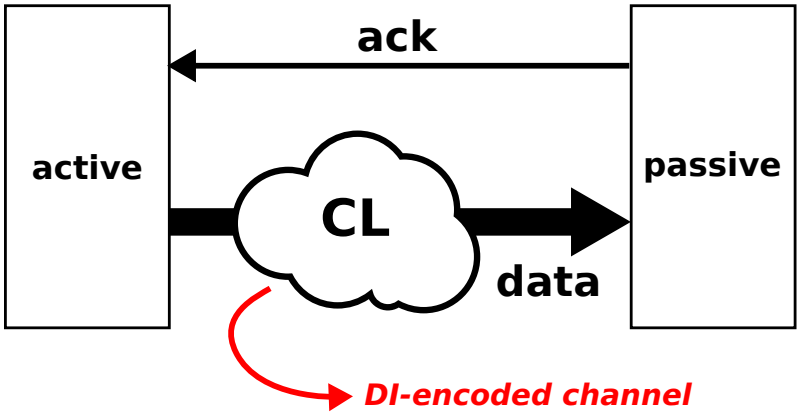

Figure 1.9: Generic scheme of a asynchronous QDI push channel.

Figure 1.9 illustrates a generic QDI push channel. Similarly to BD channels, data flows through the data path according to the handshaking protocol between active/passive blocks. However, QDI channels remove the dedicated request signal as they encode the

[3] The authors refers 'Early Ack" as the handshake protocol used and there is no explicit classification of the implementation itself. Thus, this thesis refers the BD implementation as Early Ack, which employs a early 4-phase protocol. This allows better compatibility with the protocols definitions in section 1.1.

data validity on the data path itself. In order to implement this DI communication, the channel employs DI-encoded data paths, allowing the passive block to detect data validity (or absence).

Any QDI channel requires the choice of a handshaking protocol and of a DI code to represent data and control flow information. One of the most used DI codes is called dual-rail [35]. Refer to Table 1.3 that presents the basic encoding for a 1-bit dual-rail channel. Each bit of data is encoded on two wires called here *d.t* and *d.f*. Usually, the scheme relies on the classic *Return-to-Zero* (RTZ), four-phase handshake protocol [7]. A receiver can obtain the equivalent of a request control signal directly from the codewords made available by the sender. In RTZ schemes, data tokens presence is identified by *d.t.* and *d.f.* being at different logic levels. To represent a high logic level, it is necessary to set *d.t* high (1) and *d.f* low (0). The representation of a low logic level is opposite: *d.t* is set low (0) and *d.f* high (1). Between two consecutive valid data, a spacer must always be inserted. In the case of the RTZ protocol, a spacer is defined as all wires at logic low (0). This work uses the terms *spacer*, *NULL* and *NULL token* as synonyms. Note that the situation where both signals are set to logic high (1) is defined as an invalid and unacceptable value. Beyond RTZ, the designer can adopt another handshaking protocol called *Return-to-One* (RTO) [37]. Similar to RTZ, the RTO protocol also identify data tokens by *d.t.* and *d.f.* being at different logic levels. However, in this case, to represent a high logic level, it is necessary to set *d.t* low (0) and *d.f* high (1). For low logic level representation, *d.t* is set high (1) and *d.f* low (0). Note that the logic level representation is practically the opposite of the RTZ representation – see table 1.3 – including the spacer and invalid encoding. In RTO schemes, both signals in logic low is invalid while the spacer is defined as all wires set to high (1).

Table 1.3: Codification for a 1-bit dual-rail channel using RTZ/RTO protocol.

Signals		**Value**	
d.t	**d.f**	**RTZ**	**RTO**
0	0	spacer	invalid
0	1	0	1
1	0	1	0
1	1	invalid	spacer

Figure 1.10 illustrates two communication cycles of a QDI push channel. As an initial state, all data signals are reset in the beginning of the communication cycle, indicating a spacer. Then, the data channel presents a valid data codification D_i (1). As a consequence, the *ack* signal is asserted, signaling that the data was received (2). Next, the data channel shows a spacer, indicating the absence of valid data (3). At last, the *ack* signal is reset,

ending the communication cycle (4). This behavior applies to both protocols, only using a distinct data encoding.

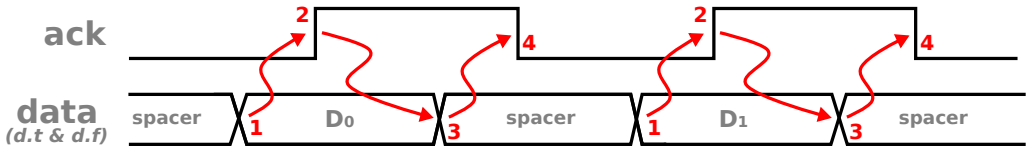

Figure 1.10: Signal behavior during data transmission in a QDI push channel.

1.4.1 The QDI Limitation

In a QDI circuit, gates and wires can display arbitrary delays. However, differently from DI circuits, there is a set of designated wire forks that must respect an isochronic timing constraint. Such isochronic forks have the additional constraint that the delay to different ends of a fork must be the same [34]. According to Sparsø [7], the behavior of an isochronic fork can be explained as follows. Figure 1.11 shows a circuit with three logic blocks (*B0*, *B1* and *B2*) that are interconnected by three wire segments, each with a given delay (*d0*, *d1* and *d2*). In this case, there is a fork F, through which any value produced by the output of the logic block *B0* passes before reaching the respective inputs of blocks *B1* and *B2*. F begins after the wire delay *d0* and has two ends, each one with a wire delay: *d1* and *d2*. Following the definition presented by [34], if the wire delays *d1* and *d2* are identical ($d1 = d2$), the circuit in Figure 1.11 respects the isochronic fork constraint and is thus called an *isochronic fork*. Despite of its elegance, this definition has been later refined to ease the practical implementation of QDI circuits and the verification of the fork isochronicity property.

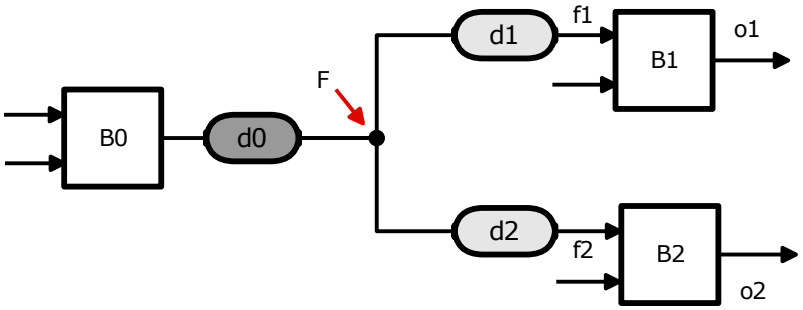

Figure 1.11: Representation of an isochronic fork with logic blocks and delay wires. Adapted from [7].

In 1995-1996, Manohar and Martin presented a new definition of isochronic fork and of the *isochronicity assumption* [38]. Considering the same structure presented in Figure 1.11, according to Manohar and Martin saying that fork F is isochronic means that some transitions on F need not to be acknowledged by a transition in both $o1$ and $o2$,

the outputs of gates $B1$ and $B2$, respectively. For example, when a transition on the input of $B1$ (after delay $d1$) has been acknowledged by a transition on $o1$, then a transition on the input of $B2$ (after delay $d2$) has also completed, even though $o2$ may not have acknowledged it. This is called the *isochronicity assumption*. As an example, consider that a rising transition occurs in F ($F\uparrow$). Thus, this transition will cause $f1\uparrow$ and $f2\uparrow$ respectively after delays $d1$ and $d2$. Next, assume that only $B1$ generates a transition in its output ($o1\uparrow$), while output $o2$ keeps the same logic level. Note that, in this case, it is not possible to visualize through $o2$ whether the transition $f2\uparrow$ was processed by $B2$. If the fork is isochronic though, after seeing an effect in $o2$ due to $F\uparrow$, it is safe to assume that $B2$ already processed transition $F\uparrow$, too. This clearly means that the delay values $d1$ and $d2$ are identical or that their difference is negligible.

Independently from the proposal of Manohar and Martin, van Berkel and others advanced an extended definition of isochronic forks [39]. For instance, the fork in Figure 1.11 respects the *extended isochronic fork* definition if the delay difference between $F \rightarrow o1$ and $F \rightarrow o2$ is less than the delays of the gates driven by the output nodes $o1$ and $o2$. That means that all output nodes must be stable when the following gates are triggered. An important aspect to consider is that this definition does not take into account the wire delays of the fork only, but also the gate delays. This is different from the original definition of the isochronic fork, where gate delays are at all disregarded. Moreover, the extended isochronic fork definition also comprises forks that employ more than one logic block in its branches. According to [39], a fork with only one logic block in each end is a fork of depth 1. For instance, the fork in Figure 1.11 has depth 1. Finally, it is important to notice that such an assumption is practically a weak timing assumption, even if it looks quite strong! Moreover, at design time, it is always possible to label the isochronic forks in order to implement and check that the fork branches are well-balanced.

1.4.2 QDI Implementations

Similar to BD channels, the specification of QDI channels also provides several degree of freedom to the designers by proposing different QDI implementations. In that way, in the recent decades, several QDI implementation alternatives have been proposed, trying to take advantage of its intrinsic delay insensibility. Among them, it is possible to locate QDI implementations focused on several performance attributes, such as power and area for instance. Table 1.4 gives an overview of all further discussed QDI implementations, highlighting their main characteristics.

One of the most known and simplest QDI implementations in the literature is the *Weak-Conditioned Half-Buffer* (WCHB) [28], which was introduced in 1995 by Caltech. Its design presents improved cycle time but also demands high transistor stacking while implementing complex logic. As an alternative, the authors in [28] also present the *Pre-Charged Half-Buffer* (PCHB) implementation. This implementation employs pre-charge

domino logic in order to reduce transistor stacking – in their case, PMOS stacking. Another important point is that both WCHB and PCHB are half-buffer implementations. This means that both implementations cannot hold data tokens at its input and output channels simultaneously. Thus, the *Pre-Charged Full-Buffer* (PCFB) implementation is also presented [28], which employs the same concepts of PCHB but with modified handshake structure to enable the presence of data tokens on all the channels.

Table 1.4: List of available QDI implementations in the literature.

Implementation	**Handshake Protocols**	**Main Feature**	**Design Strength**
WCHB [28]	Four-phase RTZ	Simple and fast cycle-time implementation	Simplicity
DIMS [7]	Four-phase RTZ	Arbitrary DI logic functions	Simplicity
PCHB [1]	Four-phase RTZ	Pre-charged domino logic (half buffer)	Performance
PCFB [1]	Four-phase RTZ	Pre-charged domino logic (half buffer)	Performance
NCL [27]	Four-phase RTZ	m-of-n threshold logic	Logic optimization
NCL+ [29]	Four-phase RTO	m-of-n threshold logic	Power reduction
SDDS-NCL [30]	Four-phase RTZ/RTO	RTZ/RTO protocol mixing	Energy efficiency
SCL [31]	Four-phase RTZ	Sleep logic for fast spacer generation	Low power
ASVHB [32]	Four-phase RTZ	Fine-grained validity mechanism	Low power

The literature also presents the *Delay-Insensitive Minterm Synthesis* (DIMS) implementing arbitrary DI logic functions. According to Sparsϕ [7], DIMS receives this name as its circuits are DI and because the C-elements in the circuits generate all the minterms of the input variables. In this case, DIMS basically utilizes m-of-n (C-elements) and 1-of-n threshold gates (OR gates) with hysteresis, Due to its simplicity, DIMS is also a well-known approach for QDI implementations, albeit its design usually leads to a significant area overhead. Other QDI implementations have been proposed, trying to overcome this area overhead issue. The first one to mention is the *NULL Convention Logic* (NCL) [27] implementation. Developed by Thesus Logic, NCL considers a more sophisticated approach to implement any DI arbitrary logic by employing m-to-n threshold logic gates.

Thus, it is possible to reduce the implementation complexity [7]. As time advances, other NCL variations have been presented in the literature. The most trivial is a RTO-based implementation called *NULL Convention Logic Plus* (NCL+) [29]. NCL+ employs the same concepts from NCL but with logic modifications in the threshold gates to accommodate the RTO protocol. Moreover, Moreira et al. further explore the possibilities of these implementations by mixing them, generating the *Spatially Distributed Dual-Spacer NULL Convention Logic* (SDDS-NCL) [30] implementation. SDDS-NCL relies on a spatial alternation of threshold gates from each of the two base implementations. Experimental analyses with a library of NCL and NCL+ components demonstrate that SDDS-NCL allows better energy efficiency and lower static power than using a single implementation style for a fixed delay. Another interesting NCL-based QDI implementation is the *Sleep Convention Logic* (SCL) [31] implementation. The SCL proposes the logic optimization by integrating a sleep logic in the threshold gates. In this case, the sleep logic reduces the logic that drives the gate output to '0', slightly improving the RTZ protocol. However, SCL utilizes an early-completion mechanism [40], which allows the circuit to acknowledge and store a data token in parallel. Despite of the possible performance advantages of this parallel computation, it also brings extra timing constraints [41].

Finally, one of the most recent QDI implementations in the literature is the *Autonomous Signal-Validity Half-Buffer* (ASVHB) implementation, which claims a focus on ultra-low power operation [32]. This implementation displays three main characteristics: (i) its structure employs integrated autonomous validity signals, which are used to simplify the circuit implementation; (ii) it utilizes a fine-grained gate-level approach, which increases throughput by propagating data through a single-cell data-path pipeline; and (iii) it implements static logic only, which increases node output stability and circuit robustness. Authors pinpoint that the ASVHB implementation can achieve higher energy-efficient design when compared to WCHB and PCHB.

1.5 Conclusions

In this chapter, it is presented the basic concepts of asynchronous design, detailing the concepts of asynchronous channels and handshake protocols. On top of that, it is also presented multiple BD and QDI implementations available in the literature, giving a brief overview of their advantages and limitations. The diversity that asynchronous design brings to the table represents the liberty that researchers and designers had took advantage in the last decades to implement asynchronous circuits with different focuses, such as performance, power, robustness and security. However, taking a testing standpoint, this also implies that asynchronous testing is far from a standard concept, as it must adapt according to the target implementation – if it employs an BD or QDI design, which handshake protocol considered, how data is encoded, if it uses FFs or latches, what kind of controller is used and so on. Consequently, standard synchronous testing procedures and available ATPG tools are not compatible with asynchronous circuits, demanding modifications. On top of that, adapting testing procedures and ATPG tools do not necessarily enables testing of all kinds of asynchronous implementations. This compromises the industrial interest on asynchronous circuit because it requires a significant effort to implement a testing procedure for a single asynchronous implementation. Thus, the first part of this thesis explores the concepts of digital testing, giving emphasis on asynchronous testing, and presents the proposed the at-speed DfT architecture for BD design.

Part II

At-speed Test for Asynchronous Bundled-data Circuits

2

State-of-the-Art on Digital and Asynchronous Testing

Contents

2.1 Stuck-at Faults

The Stuck-at fault model is a simple approach to represent several physical defects in a logic level. In fact, according to [42], the single-stuck fault model is also referred as the *standard* fault model due to its wide adoption and strong study background. As a digital circuit can be modeled as a netlist with gates interconnected by wires, the stuck-at fault model is assumed to only affect the interconnection between gates. This fault is modeled by assigning a fixed (0 or 1) value to a signal line in the circuit, independently of the logical output of the driving gate. If the fault fixes the interconnection at '0', it is called stuck-at-0 (s-a-0). If it fixes at '1', stuck-at-1 (s-a-1). While considering single-stuck faults, three main assumptions are considered:

1. Only one interconnection is faulty;
2. The faulty interconnection is fixed at '0' (s-a-0) or '1' (s-a-1);
3. The fault can be at an input or output of a logic gate.

For instance, Figure 2.1 illustrates a combinational-only circuit with a s-a-1 fault in the OR gate's output. The example assumes all gates work properly except the interconnection with the s-a-1. If the OR gate receives as input "01", "10", "11", the fault produces no effect in the circuit because the correct output is also '1'. However, if the OR gate receives '00', the OR gate must drive its output to '0' – the faulty circuit will have '1'. As the output of the OR gate is not directly observable, it is necessary to force a specific pattern at the inputs of the AND gate in order to propagate the fault to the primary output Z. In this case, applying an input test pattern "1100" – the AND gate's inputs to "11" and the OR gate's inputs to "00" – allows to force the output Z to '0'. If the s-a-1 fault is present, it will be possible to detect it due to the output discrepancy at Z.

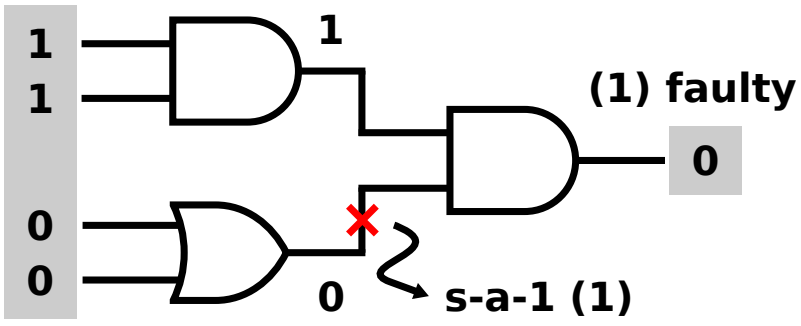

Figure 2.1: Stuck-at fault example. Extracted from [43]

In reality, stuck-at faults can occur in other interconnections as well. In a circuit with n interconnections, $2n$ single-stuck faults can be present. Considering the circuit example in Figure 2.1, which has seven signal lines, 14 stuck-at faults can occur. Depending on the circuit, this value can be reduced by applying fault collapsing [43]. Moreover,

multiple stuck-at faults can occur in the circuit at same time. Despite of being more realistic, the multiple-stuck fault model is not widely adopted as it the number of faults increases exponentially according to the number of interconnections, compromising the fault analysis for even circuits with moderate size. Notice that single-stuck faults are able to achieve a high percentage of multiple-stuck faults [43].

2.2 Path-Delay Faults

The path-delay fault is an important fault model used in delay testing [43]. These faults cause the cumulative propagation delay of a combinational path to increase beyond some specified time duration. The combinational path begins at a primary input or a clocked register, contains a connected chain of logic gates, and ends at a primary output or a clocked register. In synchronous circuits, path-delay faults are directly related to combinational delays that disrespect required timing margins regarding the clock period or the vector period. Propagation delay is defined for the propagation of a signal transition through the path. Thus, for each combinational path there are two path-delay faults, which correspond to the rising and falling transitions.

Path-delay test applies a transition at the input of the path and measures the output value after a specified interval (clock period). As this kind of test applies a pair of test vectors, it is often called *two-pattern* testing. For the test to be an effective measure of the path delay, the "expected or correct" output value must be uniquely controlled by the specified input transition. While testing a target path, any internal signal included in the path is classified *on-path* signals, and signals not present but connected to gates in the target path are called *off-path* signals. A path-delay test consists of a vector pair $(V1, V2)$ with the following two main assumptions:

1. The transition $(V1, V2)$ initiates the appropriate transition at the beginning of the target path;

2. All off-path input signals for the target path assume non-controlling constant values ('0' when the signal feeds into an OR or an NOR gate, and '1' for AND or NAND gates) following the application of $V2$. This conditions known as static sensitization of a path.

By assuming these two assumptions, the path-delay test is classified as *non-robust*. A non-robust path-delay test guarantees detection of a path-delay fault when no other path-delay fault is present. In order to achieve a *robust* path-delay test, which guarantees detection of a path-delay fault independently of the delay distributions in the circuit, it must produce a transition at an output with the following properties:

1. It must be a *real event*. A real event in the output implies the output has a single transition from the initial value to the final value;

2. It must be a *controlling event*. A controlling event permits no other events to appear prior to its own appearance. Thus, the output will remain at the initial value until the controlling event occurs at the input.

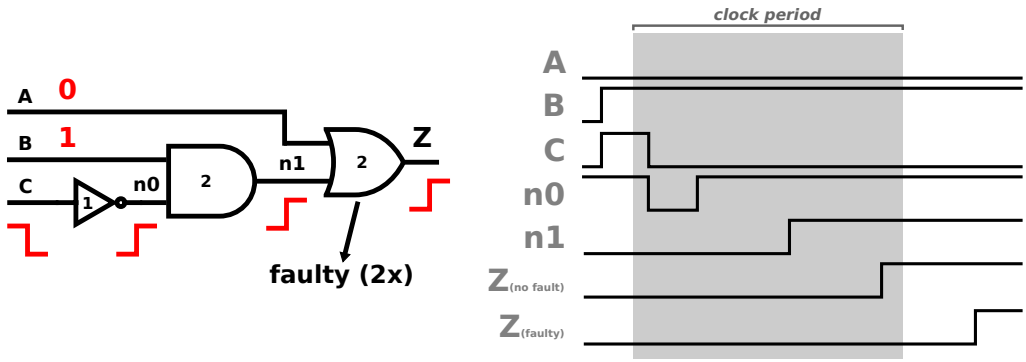

Figure 2.2: Delay fault example. Adapted from [43].

Consider the example in Figure 2.2 (a), where illustrates a combinational-only circuit with three input A, B and C, and a single output Z. We assume the inverter gate has a propagation delay of 1 time unit, both AND/OR gates have 2 units and the circuit must respect a clock period of 6 units. For the sake of simplicity, both falling and rising propagation delays are equal for a given gate. The longest path in the circuit is from C to Z, which has a expected total delay of 5 units. Now, consider we would like to test the delay path $\downarrow C \rightarrow n0 \rightarrow n1 \rightarrow Z$. Initially, the circuit receives an input vector "011" ($A = 0$, $B = 1$ and $C = 1$). This initial condition allows to trigger the target delay path through the input C while maintaining other inputs constant. Then, to trigger that target transition, the circuit receives "010", introducing a vector-pair $(1,0)$ at input C. This transition will trigger the rest of the path until reaching the output Z, where it will generate a rising transition – see Figure 2.2. If no delay fault is present in the target path, the output Z will contain the correct logic value before the next clock edge. However, supposing the presence of a faulty NOR gate that has its propagation delay doubled, Z will contain an incorrect logic value when the next clock edge arrives, Indicating that the target path disrespects the timing constraint.

Regarding the number of path-delay faults in a given circuit with n physical paths, the circuit can contain $2n$ path-delay faults. Generally speaking, any combination of paths can be faulty. However, similar to the "single stuck-at" fault model, it is usually considered delay faults of single paths, despite the fact that multiple paths can be faulty in practice.

2.3 At-speed Testing

At-speed testing consists of evaluating the circuit operation at the expected clock speed. In this case, the initialization and verification can be performed in lower fre-

quencies, albeit the test run must operate at the rated-clock frequency. This kind of test is essential unless the timing design is too pessimistic and process tolerances are extremely tight. However, both of these attributes are not possible for today's *Very Large Scale Integration* (VLSI) chips that drive extremely high speed systems and are manufactured through leading edge processes. Application of at-speed testing with stuck-at fault test vectors, though used frequently, is not the best strategy. This is because those vectors may not have a high delay fault coverage. Thus, path-delay tests for critical paths should be included in the at-speed testing. Considering the possibility of a very large number of paths, critical path testing is a good approach. According to [43], these tests are very good at uncovering "correlated defects". Variations in the manufacturing process could generate these defects, affecting all components on the chip in a similar way. For instance, the resistivity of all interconnects may increase, or all transistors may operate slower. Thus, the longest delay paths should be the first to fail.

2.4 Digital Design-for-Testability and Scan-based Design

DfT or *Design for Testing* is a well-known approach in the industry to add controlability and observability for a design in order to optimize the testing process. This approach requires the insertion of extra logic into the design to adapt the circuit for testing. The insertion of this DfT logic occurs during the hardware design flow, in which are already automated through commercial synthesis and DfT tools. Moreover, employing DfT also requires the use of ATPG tools to generate testing vectors according to the structure created by the DfT tools. Take as example the DfT/ATPG tools from the three main EDA providers in the market: Modus DFT (Cadence), TestMAX (Synopsys) and Tessent (Mentor Graphics). At the end, applying DfT can increase the design fault coverage, reduce test generation time and achieve lower life-cycle costs [42, 43]. Obviously, this comes with a cost as DfT brings area overhead and higher logic complexity to the design. This trade-off is usually acceptable in the industry as it can provide a "debug-friendly" design that enables better fault detection after fabrication and, mainly, before sending the final product to the customer. In fact, it is common to hear from test engineers the phrase "*pay less now and you shall pay more later without DfT*" when this matter is brought to discussion.

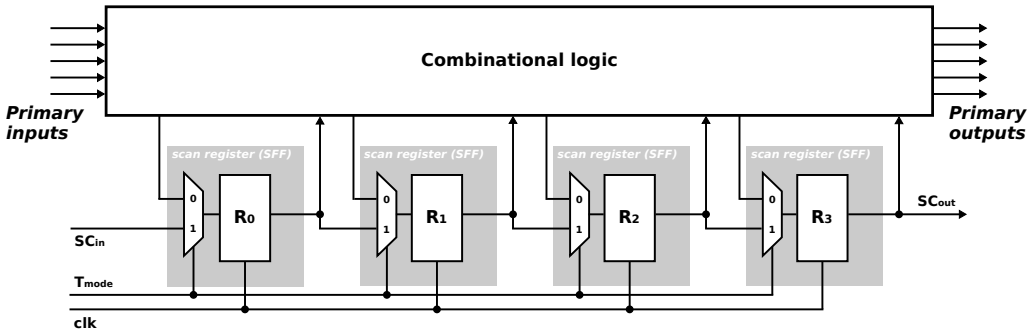

Figure 2.3: Generic architecture of a scan-based design with a single scan chain. In this example, the design has dedicated SC_{in} and SC_{out} signals, albeit these signals are usually multiplexed with primary inputs/outputs. Note that, in practice, a design can have multiple scan chains in order to reduce scan-in/out time.

Among the DfT methods available in the literature, it is possible to highlight two main structured for digital logic: (i)*Built-in Self-Test* (BIST) and (ii) *scan chains*. Applying a BIST architecture enables the circuit to self test during its lifetime. Thus, the circuit is capable to generate testing patterns and compare results with expected values. The biggest advantage of introducing BIST is test cost reduction, as it requires fewer test pins, eliminates the need of high-performance *Automated Testing Equipment* (ATE) and enables test runs at the rated-clock frequency. On the other hand, BIST requires extra circuitry on chip and increases the overall logic complexity.

Scan chains focus on increase the controllability and observability of clocked registers

– usually FFs – while adding few primary testing signals to the design. It adds a test mode that connects all (or selected) FFs in a single shift register chain, allowing to bypass the combinational logic. This facilitates loading and unloading test vectors into the design without taking into account the data path. Beyond the fact that scan chains increases the overall testability of the design with low area overhead and few testing signals, the scan chain flow is already heavily automated by DfT and ATPG tools as mentioned before. Figure 2.3 illustrates a generic scannable architecture with a single scan chain connecting all FFs. In order to connect all scan-chain components, it is necessary to replace the FFs by scannable counterparts, as suggested in Figure 2.3. The scannable FF implementation is often called *Scan Flip-Flop* (SFF) and it comprises of a multiplexed-input master-slave *Type-D Flip-Flop* (D-FF). The multiplexed input adds a testing input TI only used in test mode, connecting the scan chain. Moreover, the scan-based design can present four new testing signals:

- Scan-in (SC_{in}): primary input of the scan chain. This input is used to load test patterns into the scan chain before running a test run;
- Scan-out (SC_{out}): primary output of the scan chain. This output is used to unload test results after running a test run;
- Test mode (T_{mode}): test mode signal. When activated, it indicates that the circuit is in test mode. It is usually used to enable the scan chain, connecting all flip-flops as a shift register. Otherwise, the circuit operates in normal mode.
- $[Optional]$ Scan enable (SC_{en}): dedicated scan enable signal. In case T_{mode} is also used for another testing purpose and the scan chain is not necessarily enabled/disabled according to T_{mode}, it is possible to add SC_{en} to only control the scan chain.

Designs can adopt a full-scan or partial-scan approach according to the design constraints. The most trivial is the full-scan approach, where all flip-flops are translated to SFF. This approach brings the higher controllability and observability in the design at a significant area cost. This higher controllability and observalibity of a full-scan design suits well if testing time is crucial, as it can reduces the number of testing vectors for a given fault coverage. On the other hand, partial-scan allows to select specific flip-flops in the design and translate them to their scannable counterpart, reducing the area impact of the DfT. It is still possible to achieve a high fault coverage while considering a partial-scan approach, albeit it requires more testing vectors than an equivalent full-scan design for a given fault coverage.

Scan-based designs can be employed for stuck-at and path-delay testing. Figure 2.4 shows the signal behavior during a single stuck-at test cycle. Each test cycle has three main steps: (i) scan in, (ii) test run and (iii) scan out. During scan in, the design is set to test mode and the scan chain is loaded through SC_{in} with a test vector IV_i – generated by

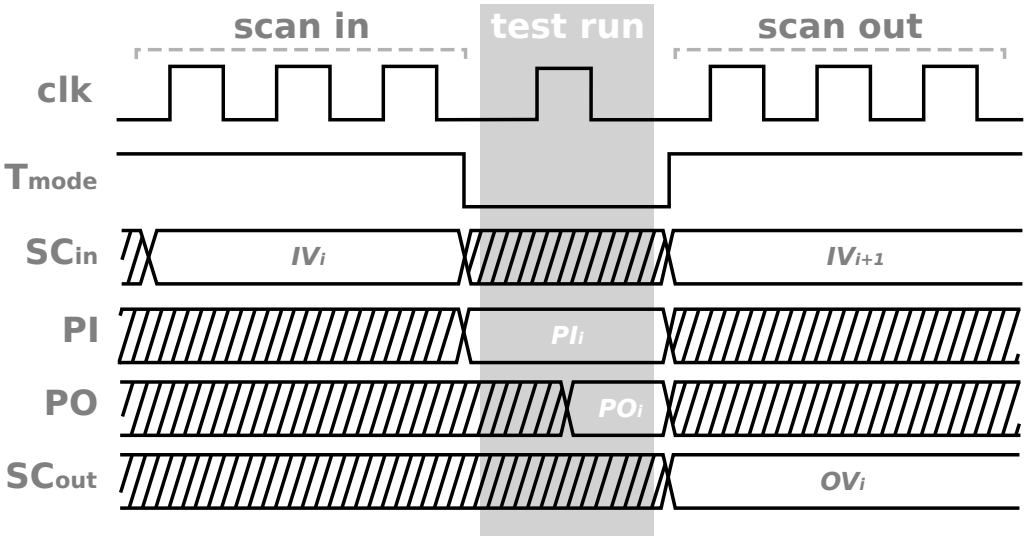

Figure 2.4: Signal behavior for a stuck-at test cycle with a scan-based design.

an ATPG tool. As IV_i is loaded by a tester or any external equipment, the clock frequency is usually lower than the operational frequency, which avoids any timing issues during the test cycle. After loading the last FF in the scan chain, the design is again set to normal mode, propagating the FF outputs to the combinational logic. Note that all primary inputs are set to PI_i, allowing the test any input logic not controlled by FFs. During test run, the clock is pulsed once in order to propagate the combinational logic output to the FFs. At this point, the primary outputs PO_i can be compared with the expected values (e.g. fault-free value or golden reference). Finally, the circuit is set again to test mode and the scan chain is unloaded through SC_{out}. In that way, each bit of the output test vector OV_i is matched with the scan-chain golden reference. If there is any value mismatch in the scan chain or primary outputs, a stuck-at fault was detected in the design. Otherwise, the design is fault free for the considered input test vector. For test optimization purposes, the scan-out step is usually executed with the scan-in step of the next test cycle, allowing to load the next input test vector in parallel. This is illustrated in Figure 2.4 as the SC_{in} receives a new test vector IV_{i+1} while the output test vector OV_i is propagated through SC_{out}.

For path-delay testing, the test cycle has similarities with stuck-at testing. Considering now Figure 2.5, the test cycle has the same three steps: (i) scan in, (ii) test run and (iii) scan out. Identically to stuck-at testing, the scan in step comprises of loading an input test vector IV_i into the design through SC_{in} in test mode. With the scan chain loaded, the test mode is disabled, all primary inputs are set to PI_i and the clock signal is pulsed twice with the target frequency, performing a two-pattern testing. At this point, the primary output values PO_i can be checked with the expected values. Next, the design enters in test mode once again and each bit output test vector OV_i is unloaded and matched with their expected values. If there is any value mismatch in the scan chain or primary outputs, a path-delay fault was detected and the tested data path cannot operate at the target frequency.

Otherwise, the path delay is fault free for the considered input test vector. Similarly to stuck-at testing, the scan-out step can also be executed while performing the scan-in step of the next test cycle, allowing to load the next input test vector IV_{i+1} in parallel.

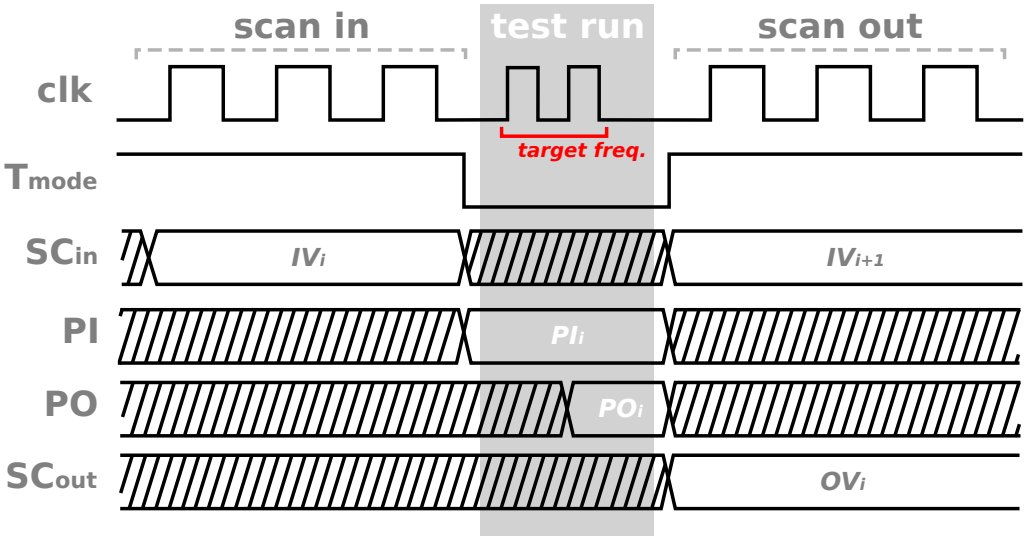

Figure 2.5: Signal behavior for a path-delay test cycle with a scan-based design.

2.5 Asynchronous Testing

The testability of asynchronous circuits is already a topic in discussion in the literature. Several well-known authors in the asynchronous field have presented the issues and solutions while testing BD and QDI circuits. This sections focuses on the test of asynchronous circuits, giving a historical view of how asynchronous testing has been shaped in the last decades.

In 1992, Pagey et al. [44] explores the basics of testing for the micropipeline template. In their work, they indicate that the control part of the circuit is concurrently testable during normal operation and ATPG for the data path can be reduced to that for the combinational circuits. In this case, the authors suggest to set all latches in pass mode and treat the circuit as a single logic block, allowing the test pattern generation. The micropipeline template is also explored by Khoche and Brunvand in 1994 [45]. The authors propose stuck-at and delay testing in the data path by adding scan functionality to latches and C-elements. One year later, the same authors present a partial scan approach for self-timed circuits [46]. Here, C-elements are not included in the scan path and modified in order to test them as combinational logic. For the scan path, the latches employ two non-overlapping clocks input to provide race free operation during testing – basically, a *Level-Sensitive Scan-Design* (LSSD) structure.

In the same year, Petlin et Furber presents a fully-asynchronous scan test technique [47] for a micropipeline-based microprocessor called AMULET. According to the authors,

the techniques allows both stuck-at and delay faults. Moreover, different testing control schemes are presented – called *Scan Test Control Logic* (STCL) – for two and four-phase protocols. Schöber et Kiel also explores a scan path design for micropipeline circuits [48]. They also propose a fully asynchronous scan approach, which in no clock is used during normal and test mode. In order to do that, latches are replaced by an *Asynchronous Scan Latch* (ASL) implementation and an extra two-phase handshaking control permits the scan manipulation (scan-in and scan-out). As the test structure is also asynchronous, this implies in a relatively high area overhead due to the extra handshaking controls. Again, Petlin et Furber covers the subject of micropipeline testing in 1997 [49]. In their work, the authors present a BIST micropipeline design. The circuit employs an asynchronous *Built-in Logic Block Observer* (BILBO) for stuck-at fault detection. The authors also indicates that their BIST approach can check the timing constraint of the micropipeline template.

In 1996, Roncken et Bruls discuss that the autonomous handshake behavior of asynchronous circuits compromises the test quality [50]. Thus, the authors introduce the handshake component HOLD, which is inserted between handshake controls to add more controllability and better applicability of scan and *Quiescent Supply Current* (IDDQ) testing. Roncken et al. [51] also addresses the use of LSSD-based testing design for asynchronous circuits. In their work, the authors propose an algorithm to implement near-optimal scan structures, avoiding the area overhead caused by LSSD. In the end of the 20th century, Roncken takes another step in the subject and presents a defect-oriented testing for asynchronous circuit [52]. This time, Roncken targets IDDQ testing for stuck-at and bridging fault detection. This is done through a HOLD DfT approach, which is responsible to stall the circuit at a desired state and, thus, allowing IDDQ testing in a quiescent state not normally available in an asynchronous design. This kind of testing approach is further addressed in [53], where it introduces the notion of naturalized communication and how to reused the handshake behavior to test two-phase bundled-data circuits, such as micropipeline, GasP, MOUSETRAP and Click templates. The authors present a proper stopper circuit, called MrGO, responsible to controlling actions, freezing or releasing specific handshake parts of the circuit. Details about the MrGO circuitry are presented, 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.

Kishinevsky et al. presents a path-delay fault testing algorithm for asynchronous circuits [54]. The algorithm is responsible to generate test sequences for a set of paths in the circuit that must respect a specific timing constraint. Despite the fact that their work covers delay testing, the authors consider only QDI circuits in their experiments.

In 1999, Kang et al. proposes a scan design for micropipeline circuits, focusing on delay testing [55]. The latches are replaced by their proposed scan latch that, according to the authors, allows full control during two-pattern delay testing. However, the proposed scan latch adds three latches (four in total), implying a relative high area overhead.

At the 21th century begin, Berkel et al. explores the insertion of synchronous and

LSSD modes to C-elements [56], targeting a more general testing approach of any asynchronous implementation. Here, they consider the scan-path technique and level-sensitive operation of LSSD operation to add full controllability and observability in sequential gates. Consequently, asynchronous circuits could take advantage of traditional DfT and ATPG tools. This statement is further explored in [57, 58], where it applies the DfT proposed in [56] and presents fault coverage and area results of five testable bundled-data circuits designed with Tangram, a Philips' design flow for asynchronous circuits. The authors were able to test both data and control paths with 100% and 99% fault coverage, respectively, with area overhead ranging from 90% (full-scan) to 30% (partial-scan). In other to reduce the impact of LSSD, Beest et Peeters also explores a multiplexer-based testing technique to test C-elements [59]. The authors indicates that the proposed multiplexer-based approach achieves the same structural fault coverage of previous LSSD approaches, as well as profits the ATPG from conventional testing tools.

Others authors mainly focus on testing the handshaking controls in asynchronous circuits. For instance, King et Saluja [60] adds testability for the control part of micropipeline circuits. By inserting scan latches in the handshake controllers and restraining combinational loops, the proposed technique synchronously treats the existing asynchronous elements – like a finite state machine. This approach brings extra controllability and observability, allowing stuck-at fault detection in the control part at the cost of a significant area overhead.

Delay testing for the MOUSETRAP [20] template is covered in [61]. The proposed DfT technique consists of controlling the acknowledgement signals between handshake controllers, permitting the user to pause or resume the pipeline at will. To allow this extra handshake control, the proposed technique requires the insertion of multiplexers in the acknowledgement signals and a shift register to configure each multiplexer. Another interesting approach to test the control part of asynchronous circuit is presented in [62]. In this work, a checker logic block is connected in the handshake signals in order to detect any violation in the handshake protocol. For each handshake pair – request and acknowledgement pair signal – the proposed checker logic requires four flip-flops, four *David Cell* (DC) circuits [63] and four delay elements, implying that checking all handshake signal brings a significant area overhead.

In 2006, delay testing for asynchronous circuits is covered in [64], which in presents a non-intrusive delay testing technique for three different templates: MOUSETRAP [20], GasP [22] and *High-Capacity* (HC) pipelines. However, the authors focus on the MOUSETRAP template most of the paper, also presenting how to generate test patterns for non-linear MOUSETRAP implementations – containing forks and joins, for instance.

Regarding the test of QDI templates, testing approaches are recently explored in [31, 65–67]. In [65], fully asynchronous scannable WCHB architecture is proposed. Both dual-rail inputs and acknowledgement signals are multiplexed and connected to a asynchronous scan chain, focusing in stuck-at fault detection. The testability of a well-known

QDI template called NCL is covered in [66, 68]. The work in [66] implements an interleaved scan architecture for NCL circuits, which is also incorporated into an on-line BIST architecture. Similarly to the previous works, it also covers stuck-at faults. In order to avoid area overhead from extra logic, the same authors takes another approach in [68]. Here, they consider a clock-less and DfT-less approach, investigating the effectiveness of IDDQ testing. The literature also presents a testable approach [31] for the SCL, a derived QDI template from NCL. However, the work in [31] only cover stuck-at faults. Huang et al. [67], for instance, replaces dual-rail registers with a pair of *Dual-Rail Asynchronous Circuit Scan* (DAC-scan) latches, which allows synchronous functionality during test. Despite the work covers WCHB implementations, the authors indicate that their work is easily adapted to WCHB implementations. The authors present both stuck-at and delay fault detection with two-pattern and three-pattern testing, respectively. Unfortunately, the insertion of DAC-scan latches implies in a significant area overhead, which is not desired in QDI implementations.

Finally, Kuentzer et al. explore the testing characteristics of the resilient bundled-data template called Blade [69, 70]. First, the authors analyze and classify the faults in the *Error Detection Logic* (EDL) of the Blade template [69], proposing slight modification in the EDL design to increase controllability/observability. As the resilient architecture of the Blade template is capable to detect timing violations during operation, it also implies that the architecture also enables online delay testing of critical paths [70].

2.6 Conclusions

This chapter covers the principles of stuck-at and path-delay faults, two widely fault models used in the industry, and its applicability in scan-based DfT design. Moreover, it presents an state-of-the-art overview on asynchronous testing, discussing the developed techniques on stuck-at, delay and IDDQ testing. It is possible to pinpoint that the literature already explores the asynchronous testing matter for almost three decades and several elegant techniques and workarounds have been proposed. Among them, authors take the approach to develop testing techniques and tools from scratch, whereas others adapt current ones to overcome the industry resistance against asynchronous design. Unfortunately, the industry resistance still remains. This implies the simplicity of synchronous design and its well-established design methodology continues to be a more convenient option than pursuing alternative design paradigms. However, it must take in account that the available literature on asynchronous testing provides a complex set of techniques, which are not straightforward not even for designers familiar with asynchronous implementations. This thesis takes this last point as a motivation in order to explore testing solutions compatible with current techniques and tools, including simple and straightforward steps to implement it.

3

Proposed At-speed DfT Architecture for Bundled-data Design

Contents

This work proposes a DfT architecture for at-speed testing, while taking into consideration the compatibility with traditional stuck-at testing tools. Taking a similiar approach as [53], we desire to freeze and release specific control parts of the circuit and test their respective data path. This is done by adding controllability in the controllers, allowing to control directly the points-of-divergence of the forward timing constraint between control and data paths. Despite the proposed testing architecture imposes modifications in the controllers design to add controllability, it requires only the use of logic gates already available in off-the-shelf standard cell libraries. The proposed architecture is also compatible with the *Local Clock Set* (LCS) flow [19] [71]. The LCS flow enables the implementation of bundled-data circuits with standard EDA tools, allowing the use of scan-chain insertion and *Static Timing Analysis* (STA) features. Moreover, the STA enables better interface with ATPG tools for path-delay testing.

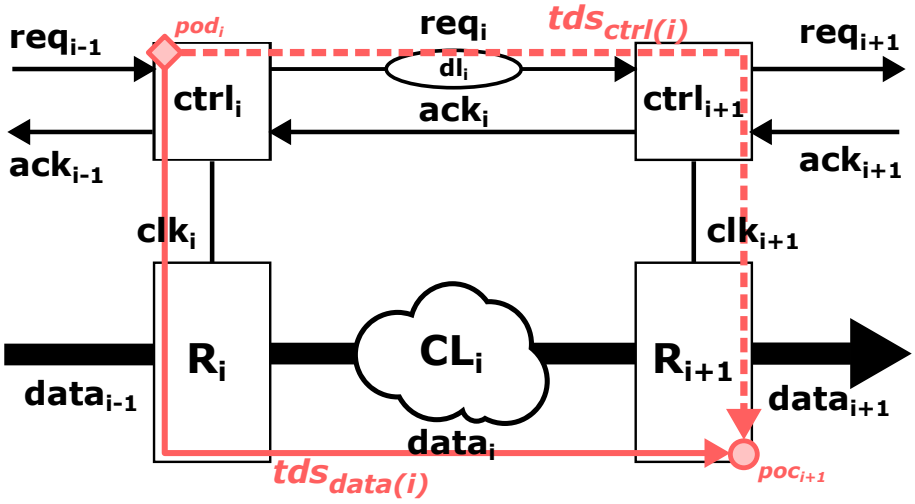

Figure 3.1: Generic scheme of an asynchronous BD push channel, highlighting the timing relationship between handshake and data signals.

3.1 Problem Statement

Due to the handshake schemes, timing constraints between handshake and data signals must be respected in order to guarantee the circuit correctness. According to [33], the basic constraints in BD design are: (i) the receiver must not consider a request until its input data is stable; (ii) the receiver must not acknowledge the sender until it has no further need for its input data and (iii) the sender must hold its output data steady between sending the *req* and receiving the *ack*. Taking Figure 3.1 again, consider that valid data is present at $data_{i-1}$ and we desire to propagate it to $data_{i+1}$. In order to do that, the launch controller $ctrl_i$ receives a token through req_{i-1} (e.g. req_{i-1} rises), indicating valid data is present at the input of its respective registers – in this case, R_i. Next, R_i stores and propagates data through the combinational block CL_i. At the same time, $ctrl_i$ also propagates the

request signal req_i, indicating to the capture controller $ctrl_{i+1}$ in the receiving part that new data are available. If req_i is propagated to $ctrl_{i+1}$, which will pulse clk_{i+1}, before the new data had been completely computed by CL_i, the receiver register R_{i+1} will store invalid data, compromising the circuit operation. This implies that, from the point-of-divergence pod_i in the launch controller, the control path $tds_{ctrl(i)}$ must be tuned in order to match the propagation delay of the launch register and the combinational block $tds_{data(i)}$ at the point-of-convergence poc_{i+1} in the capture controller. Therefore a delay line dl_i is inserted into the request signal between the controllers, delaying the arrival of the request signal at the receiving end. This timing constraint between control and data paths is called forward (setup) constraint and can be represented by Equation 3.1. The main objective to the proposed architecture is to allow verifying whether the timing constraint imposed in Equation 3.1 has been respected after fabrication.

$$tds_{data(i)} < tds_{ctrl(i)} \tag{3.1}$$

3.2 Overview Architecture and Testing Signals

Figure 3.2 illustrates the proposed testing architecture with a push channel structure. The proposed architecture includes signals present in traditional scan-based architecture as well as dedicated signals for the proposed at-speed technique. The required testing signals for the proposed architecture are listed below:

- Test mode (T_{mode}): when enabled, it bypasses the control logic and allows to control registers with an external clock signal T_{clk}. This signal is usually added for stuck-at testing with scan chains;
- Scan enable (*SC_enable*): when enabled, it bypasses the combinational logic presented in the circuit data paths, allowing to load and unload the scannable registers in the circuit. This signal is also presented for stuck-at testing with scan chains;
- *Handshake Breaker* (HSB): this signal allows to disable or enable the left handshake signals of a given controller. When enabled, this signal disables the left handshake signals and permits external control of the left request signal of the controller through *ext_req*. When disabled, the left handshake signals are connected to the previous stage(s) and the controller operates normally;
- External request signal (*ext_req*): this signal is specially added for the at-speed testing methodology in this work. During at-speed testing, this signal is responsible to internally propagate tokens in control paths.

Despite of the additional testing signals, note that the two first signals (T_{mode} and SC_{enable}) are usually already presented into the design whether a scan-chain structure is

employed. Thus, the proposed structure only leverages the scan-chain structure for better controllability and observability of the data path. However, the proposed structure adds testing signals to the control part of the circuit: external request (*ext_req*) and HSB signals. As the name suggest, HSB signals are responsible to 'break' the left handshake communication of a given controller. Consequently, each controller in the circuit requires a reserved HSB signal. When enabled, the HSB sets the controller in **launch** mode during at-speed testing. In this mode, the left handshake signals of the controller are disconnected from the previous controller(s) and bypassed directly to *ext_req*. This isolates the controller from its previous neighbour controller(s) and avoids any interference from another inserted token, which may create timing issues during at-speed testing. However, this also creates a testing limitation as no handshaking communication is possible and, consequently, makes it impossible to verify the timing constraint of the previous control/data path. This requires different HSB configurations in order to cover all existing timing constraints in the design. When the HSB signal is disabled, the controller remains in **capture** mode and it normally communicates with previous controller(s). To reduce the use of test inputs for each HSB signal, a shift register is employed to sequentially load each HSB.

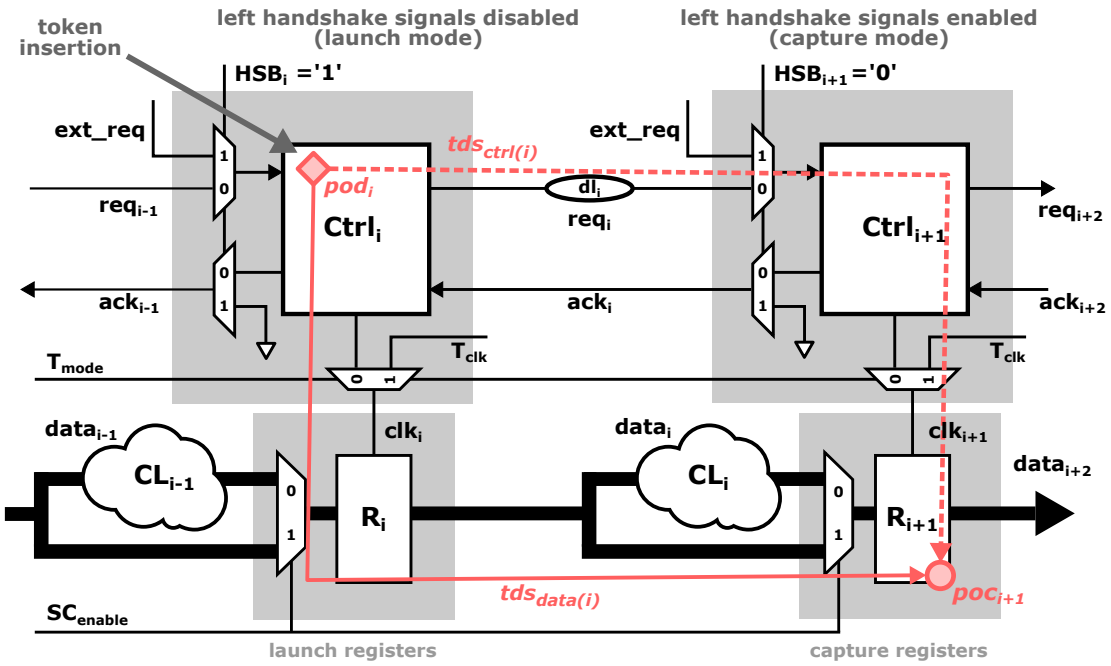

Figure 3.2: Proposed testing structure. Lower MUX logic controlled by T_{mode} and SC_{en} are already presented for stuck-at testing. The proposed testing structure adds two more MUX gates that allow to disable the left handshake signals of the controller.

3.3 Test Cycle

The test cycle is composed by three main steps: (1) scan in, (2) test run and (3) scan out. Figure 3.3 illustrates the behavior of the main test signals. During the first step (1), both T_{mode} and SC_{enable} signals are enabled. This puts the circuit in test mode and allows loading the scan chain and configuring each HSB signal. Once the scan chain and the HSB shift register are loaded, T_{mode} and SC_{enable} signals are disabled, the circuit goes back in normal mode. When ext_req is enabled, it launches all controllers in launch mode (HSB enabled) and propagates tokens to the remaining controllers set in capture mode (HSB disabled) – this is the test run step (2). Finally, the circuit is put in test mode again to scan out (3) the circuit and verify if the capturing registers contain the expected test vectors.

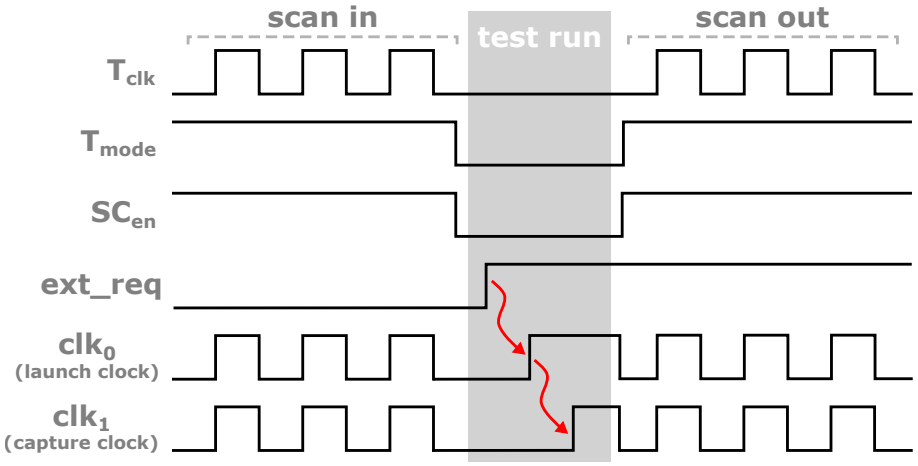

Figure 3.3: Behavior of the testing signals during a test cycle.

3.4 Initialization

For the proposed testing procedure, it is required to initialize both data and control paths. Regarding the data path, test vectors must be loaded into the design for two-pattern testing. Considering a launch register, which will directly be activated by an inserted token, precedent and successor data paths are loaded with test vectors in such a way that, when the circuit is launched, the launch register will stimulate a target path while the token is also propagated through the control path. A straightforward technique to load the data path is a scan chain. To allow that, the controller clock clk_i is usually multiplexed with an external test clock T_{clk} and the registers are replaced by their equivalent scannable implementations. During test mode, the registers are externally controlled by T_{clk} and all data paths are bypassed. During normal mode, registers are controlled internally by the their respective controllers and data paths are not longer bypassed.

For the control path, all handshake signals must be set according the initial state of

the employed handshake protocol. In that way, all the controllers must have known handshake signal values and be ready to process the inserted tokens during the test. This can be done by setting the primary control inputs and all the memories inside the control blocks – FFs, latches or C-element – according to the circuit initial state. On top of that, depending of the employed handshake protocol, multiple initial states must be considered to ensure that the circuit is fully tested. For the sake of illustration, Figure 3.4 illustrates block representation (a) of a WCHB controller with its handshake signals and its *Signal Transition Graph* (STG) representation (b). The STG indicates the transition sequences of the left and right handshake signals of the WCHB controller. Note that we consider only a standard four-phrase handshake protocol. In this case, registers are only activated in the first phase of handshake protocol ($R_{req}+$ rising). As our technique focuses on externally generating the rising transition of the left request signal $L_{req}+$, all the remaining handshake signals and internal memories must have the logic values preceding $R_{req}+$. Taking into consideration the STG in Figure 3.4 (b), the left token is generated externally through *ext_req*. However, the right token requires that not only the controller itself would be properly reset, but also all its successor controllers – guaranteeing the desired logic value of R_{ack}.

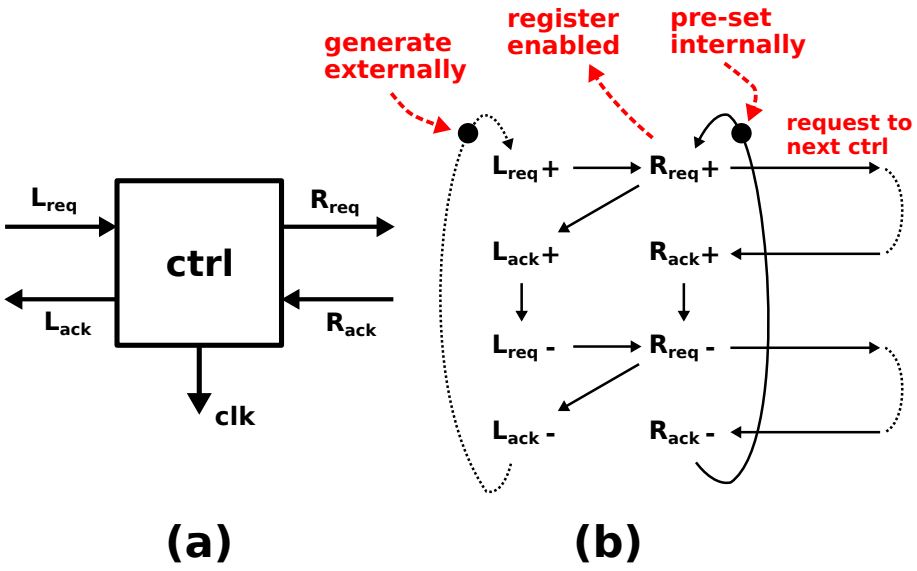

Figure 3.4: Block representation (a) of a WCHB controller with its handshake signals and STG representation (b) of the WCHB controller demonstrating the handshake behavior.

3.5 Checking Circuit Correctness

After launching the circuit, the circuit must be checked to verify circuit correctness. This can only be done after the circuit had computed all tokens and when no token is being propagated through the circuit. Then, the values in the capture registers can be extracted through the scan path and matched with the expected values. If the the capture registers contains the expected values, it indicates that the timing constraints between the control and data paths are respected. Otherwise, the timing constraint between the launch/capture registers have been violated.

3.6 Retrieving Path-Delay Information with Local Clock Sets

As mentioned before, the LCS flow [19] enables the use of standard EDA tools to run synthesis and STA on bundled-data circuits. The basic idea of the LCS flow consists in using root clocks at each controller in order to "break" the combinational loops in the control logic. Based on a given root clock, LCS derives launch and capture generated clocks, which allow to verify hold and setup timing constraints, respectively, beyond neighbour root clocks. For the sake of illustration, we take a linear micropipeline-based structure as shown in Figure 3.5. Here, a root clock clk_i is defined in controller $ctrl_i$. Not only clk_i controls all registers in the stage as it is propagated to the next and previous stages through the *req* and *ack* signal, respectively. The analysis between $ctrl_i$ and the next controller $ctrl_{i+1}$ verifies the setup constraint, whereas the analysis between $ctrl_i$ and the previous controller $ctrl_{i+1}$ verifies the hold constraint. In the next controller $ctrl_{i+1}$, another root clock is also defined, blocking the STA analysis from clk_i to the registers in the next stage. Thus, the LCS flow creates a generated clock *g-setup-*clk_i on clk_{i+1}. The trick here is that $g-setup-clk_i$ considers clk_i as the clock source, allowing to propagate the STA analysis through clk_{i+1} and until its respective registers. Because *g-setup-*clk_i captures data propagated from clk_i, the LCS flow defines this generated clock as a *capture clock* of clk_i. The same idea applies for the previous stage. As clk_{i-1} also blocks the STA analysis, another generated clock is created on clk_{i-1}, now called *g-hold-*clk_i due to the hold analysis. In this case, the LCS flow defines *g-hold-*clk_i as a *launch clock* of clk_i.

Because of this timing support created by the LCS flow, it is possible to check timing violations with standard EDA commands (e.g. *report_timing*). Note that, during STA checking, the tool is able to identify the required transitions to stimulate a given path, such as a critical path or any desired path. This very same information is also the required information that ATPG tools need to know how registers must be initialized for path-delay testing.

It is important to indicate that, in our case, the path-delay extraction is software de-

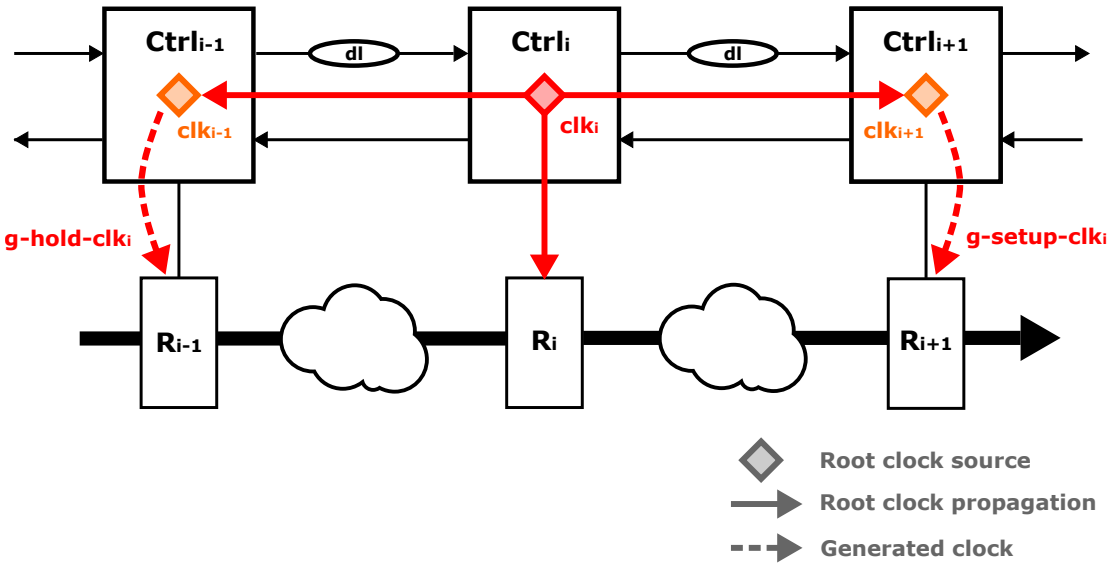

Figure 3.5: Local clock set definition with a linear 3-stage structure. By defining root clocks at each controller, the LCS flow is capable to create generated clocks in the neighbour controllers to allow setup and hold timing analysis.

pendent, specially because it must employ the same tools compatible with the LCS flow. In that way, giving a more practical view of our approach, the path-delay information can be extracted through Synopsys' Primetime tool – currently supported by the LCS flow. On top of that, Primetime interfaces Synopsys' ATPG tool TretaMAX, which reads the given path-delay information and generates test vectors for two-pattern testing. However, the behavior of the local clock signals are not identical to the synchronous two-pattern testing as the root clock and the generated capture clock pulse only once. Consequently, the only valid expected register values generated by TetraMAX is in the capture registers and all values in the launch registers must be ignored (*don't care*).

3.7 Testing Non-linear Structures

Asynchronous circuits employ a complex control scheme far different from the conventional linear scheme as illustrated in Figure 3.1. Thus, they can employ unconditional and conditional flow schemes such as forks (a), joins (b), splits (c) and merges (d), as indicated in Figure 3.6. According to the employed flow scheme, the proposed testing technique must be adapted in order to properly insert tokens in the circuit without stalling the circuit or violating the handshaking protocol.

With a fork scheme, the generated token propagates to all successor branches. This implies that the controller in the sending end must be configured in launch mode and all branch controllers in capture mode. As the fork employs a unconditional control flow, a single token in the receiving controller propagates to all branches, allowing to verify the operation correctness of all branches in parallel. The same idea applies to join schemes. However, in this case, the branches controllers are the sending end. Then, all branch controllers must be configured in launch mode. If the controller in the receiving end does not receive tokens from all branches, no further token is generated and the circuit halts, registering no new data in the receiving end.

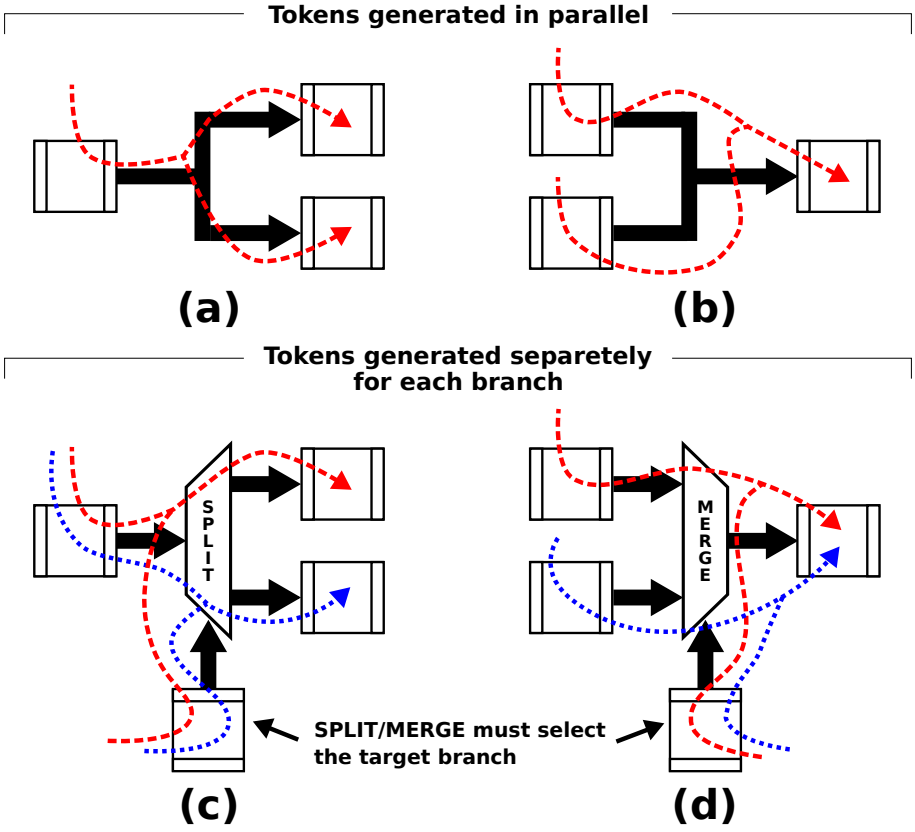

Figure 3.6: Token generation according the control path structure: (a) fork, (b) join, (c) split and (d) merge. The arrows indicate where the token is generated (through ext_req) and where it is propagated.

For conditional control flow, such as splits and merges, additional care is needed. In

split schemes, the split selector is responsible to redirect the flow to a target branch. In order to test all branches, it is required to generate tokens propagating to a target branch, which is selected by the split selector controller. Then, for a target branch, the sending and split selector controller must be set in launch mode and the target branch controller in capture mode. This process is repeated until all branches were selected and tested.

Again, the merge scheme considers a similar logic as the split scheme. For a target branch, the target branch and merge selector controller must be set in launch mode and the receiving controller in capture mode.

3.8 Compatibility with Traditional Stuck-at Testing

Another point that the proposed architecture takes into consideration is to keep the compatibility with traditional testing, including the use of DfT and ATPG tools available in the industry. In this case, we consider scan insertion and traditional stuck-at testing. As the control part utilizes C-elements and combinational loops, the strategy focuses on isolate the control part. Taking as example Figure 3.7, this is done through enabling the T_{mode} signal, which bypasses all the control logic and redirects all the register control to T_{clk}. Consequently, the circuit operates synchronously and the scan test protocol remains the same. T_{mode} must remain enabled during the entire test cycle and SC_{enable} only enabled during scan-in and scan-out steps.

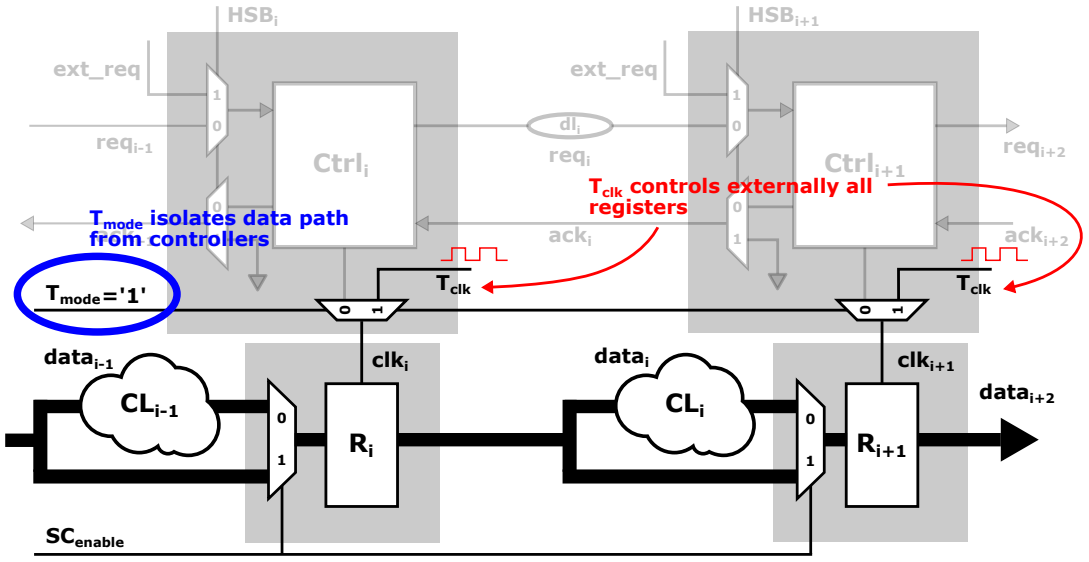

Figure 3.7: Stuck-at testing with proposed approach. Data paths are isolated with T_{mode} signal, avoiding any interaction with controllers. Thus, circuit operates as a conventional synchronous circuit.

3.9 Study-case Circuits

This work applies the proposed DfT architecture in two study-case BD circuit: a simple 2-bit adder and 128-bit *Advanced Encryption Standard* (AES) cryptocore. The target technology in both cases is a 65-nm bulk CMOS technology provided by STMicroelectronics. By considering a simple study case, it is possible to present minor details while implementing the proposed DfT architecture. Thus, the reader will find required specification and *Tool Command Language* (TCL) commands for the LCS flow as well as for ATPG scripting. Next, a more complex study case allows to better understand how the proposed DfT architecture impacts a practical circuit. In this case, the reader will find overall details such as fault coverage, area overhead and particular aspects while testing non-linear structures. Remarks are also presented about the use of the available scripts, standard cell libraries and EDA tools.

3.9.1 A simple circuit: 2-bit adder

Figure 3.8 (a) illustrates the architecture of the 2-bit BD adder. The adder employs a four-phase micropipeline design and it has registered inputs and outputs, controlled by two controllers $ctrl_{in}$ and $ctrl_{out}$, respectively. Moreover, Figure 3.8 (b) shows the internal implementation of the testable controllers used in this case. As the target technology does not provide standard cell libraries with C-element gates, the C-element logic was designed with conventional logic gates – see Figure 3.8 (c), which illustrates the gate-level implementation of a majority-based C-element.

The controllers were instantiated manually before synthesis and they implement the testable functionality required for the proposed DfT architecture. Despite Figure 3.8 indicates the presence of SFFs in the design, the initial Verilog description only implements conventional FFs, leaving the scan insertion to the DfT tool during synthesis. For synthesis, it is considered Synopsys' *Reference Methodology* (RM) flow scripts. To be able to synthesize a BD circuit and perform STA analysis, the RM flow integrates the LCS flow [19]. Listing 3.1 indicates the LCS specification for the 2-bit BD adder. First, it specifies the input channel *adder_in* and the output channel *adder_out* (line 8 and 9). Because those channels interact with the external environment, the LCS requires the explicit specification of the handshake signals and the data signals related to them. For instance, for the input channel *adder_in*, the input handshake signals *req_i* and *ack_i* are explicitly associated to the data signals op_a and op_b. Next, it specifies the root clocks *clk_in* and *clk_out* (line 11 and 12). For each root clock, LCS associates it to its respective controller, initial state (reset or set) and a budget clock period that the synthesis tool uses as timing reference.

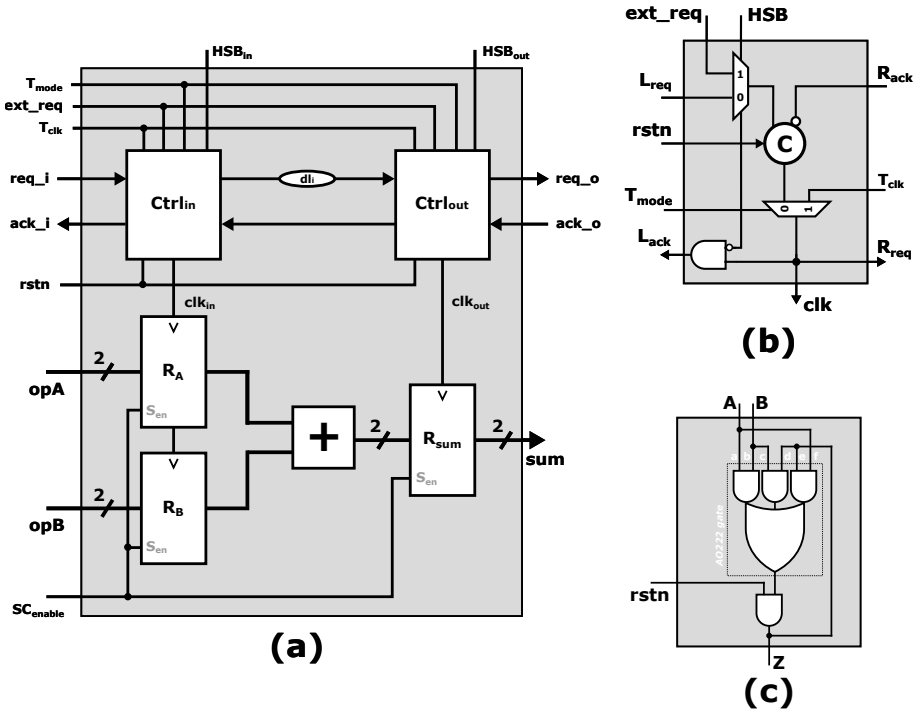

Figure 3.8: Architecture overview of the implementation of the (a) 2-bit BD adder, (b) the testable controller and (c) majority-based C-element gate.

Listing 3.1: Root clock specification for the LCS flow.

```
## Format (input/output channel):
## <channel_name> {<req_sig> <ack_sig> <data_sig> \
##                  <budget_clk_per> <budget_data>}
## Format (root clock):
## <clk_name>   {<inst_ctrl_name> <reset(R)|set(S)> <budget_clk_per>}
array set reg_ctrl "

    adder_in        {req_i ack_i {op_a op_b} 2.5 {0 0}}
    adder_out       {req_o ack_o sum 2.5 {0.5 0.9}}

    adder_stage1    {reg_ctrl_in R 1.0}
    adder_stage2    {reg_ctrl_out R 1.0}
"
```

The LCS specification was also modified to support the majority-based C-element design. The C-element considered in this study-case circuit has not been implemented as a single gate and internal combinational loops are visible by the STA tool. Normally, the STA tool detects these loops and breaks them – disabling the timing arc that creates the loop – in order to perform the STA analysis. To avoid any misjudgement by the STA tool during synthesis, the LCS flow allows the insertion of dummy clocks at the input pins where the loops are created. The STA tool will not perform its analysis beyond the dummy clocks and no timing arc will be disabled. On top of that, the use of these dummy clocks allow to specify timing constraints on the feedback structure, as the majority-based

design contains an isochronic fork that must be respected. Listing 3.2 shows the definition of extra dummy clocks for each C-element present in the circuit. The TCL script directly searches all majority gates [1] present in the circuit and creates dummy clocks *DMC_loop_muller* at the pins connected to the combinational loop. Taking into consideration the majority-based C-element implementation in Figure 3.8 (c), pins *D* and *E* connect the feedback with the output *Z*.

Listing 3.2: Required dummy clocks to avoid combinational loop issues inside the majority-based C-element design.

```
1  set i 0
2  set list_maj [get_object_name [get_cells -hier MAJ]]
3
4  foreach gate_maj $list_maj {
5
6      puts $gate_maj
7      create_clock -name DMC_loop_muller${i}_D \
8          -period ${DEFAULT_PERIOD} [get_pin ${gate_maj}/D]
9      create_clock -name DMC_loop_muller${i}_E \
10         -period ${DEFAULT_PERIOD} [get_pin ${gate_maj}/E]
11
12     incr i
13
14 }
```

The RM script was modified in order to load the LCS constraint specifications as well as to enable scan insertion and avoid any issues with the control part of the circuit. Listing 3.3 shows the DfT-related commands added to the RM script. The commands between lines 3 and 6 defines a basic timing test setup, including the definition of test frequency of 10 MHz. The test setup values are standard and mostly used for tutorials and examples like the study-case circuit. The command at line 8 only reinforces that the selected scan style will utilize multiplexed FFs and line 9 specifies that the DfT tool will implement a single scan chain. Next, it is necessary to specify the required DfT signals for scan insertion. The *set_dft_signal* commands specify all standard scan signals. These signals are the very same signals used for traditional scan-based synchronous implementation and, consequently, allow us to employ the synthesis tool despite of the BD design. Basically, these commands enables the scan insertion into the design. The commands between line 28 and 37 address the control part of the circuit. In this case, all input signals connected to the control part are set to values values, blocking the DfT tool to interact with the control part. Note that it includes DfT signals *req_ext_i* and *HSB* signals. Those signals remain constant during synthesis in order to guarantee the compatibility with the

[1] Note that this study-case circuit only employs majority gates inside the C-elements. In case majority gates are used somewhere else in the design, this script section must adapted to target the majority gates inside C-elements only.

synthesis and LCS flows. On top of that, this avoids issues with theDfT tool due to the combinational loops inside the control part, which may generate errors during synthesis. Having all DfT signal specification, it is possible to create the *STIL Protocol File* (SPF) with the *create_test_protocol* command (line 40). The final four commands (line 41-44) verify, preview and insert the DfT structure into the circuit. More specifically, *dft_drc* verifies the design rules of the DfT specification and structure, *preview_dft* allows to get a preview of the scan architecture before it is actually implemented in the design and *insert_dft* finally implements the scan chain.

Listing 3.3: Additional DfT commands in the RM flow for Synonpsys Design Compiler.

```
1  ## timing setup
2  set test_default_delay 0
3  set test_default_bidir_delay 0
4  set test_default_strobe 40
5  set test_default_period 100
6
7  set test_default_scan_style multiplexed_flip_flop
8  set_scan_configuration -chain_count 1
9
10 set_dft_signal -view existing dft type ScanClock \
11     -port tclk -timing [list 45 55]
12
13 set_dft_signal -view existing_dft -type Reset \
14     -port reset -active_state 0 \
15
16 set_dft_signal -view existing_dft -type TestMode \
17     -port tm
18
19 set_dft_signal -view spec -type ScanDataIn \
20     -port op_a[0]
21
22 set_dft_signal -view spec -type ScanDataOut \
23     -port sum[0]
24
25 set_dft_signal -view spec -type ScanEnable \
26     -port sc_en -active_state 1
27
28 set_dft_signal -view existing_dft -type Constant \
29     -port req_ext_i -active_state 0
30 set_dft_signal -view existing_dft -type Constant \
31     -port hsb[0] -active_state 0
32 set_dft_signal -view existing_dft -type Constant \
33     -port hsb[1] -active_state 0
34 set_dft_signal -view existing_dft -type Constant \
35     -port req_i -active_state 0
36 set_dft_signal -view existing_dft -type Constant \
37     -port ack_o -active_state 0
```

```
38
39 create_test_protocol
40 dft_drc
41 preview_dft
42 insert_dft
43 dft_drc
```

Having a synthesized design with the additional DfT commands in Listing 3.3, it is possible to move forward with the PrimeTime tool, which allows us to perform STA analysis with all timing constraints generated by the LCS flow. Listing 3.4 shows the required commands to load the design on PrimeTime after synthesis and enable STA analysis with LCS. It also demonstrates how to export path-delay information to TetraMAX, which will be later employed for the ATPG.

*Listing 3.4: PrimeTime setup to load the design and enable STA analysis with the timing constraints generated by the LCS flow. It also gives an example of extracting path-delay information from adder_stage*1 *root clock. The script only extracts the critical path in this case.*

```
1  ## load design
2  read_verilog async_2b_adder.mapped.v -hdl_compiler
3  set_design_top async_2b_adder
4  link
5  source async_2b_adder.mapped.sdc
6
7  ## required commands on Primetime to allow STA analysis
8  ## with the root clocks
9  set_multicycle_path 0 -hold -from [get_clocks *] -to [get_clocks *]
10 set_multicycle_path 0 -setup -from [get_clocks *] -to [get_clocks *]
11 set_propagated_clock [all_clocks]
12
13 ## just checking if the report_timing can identify the path
14 ## between the root clock (adder_stage1) and the
15 ## capture clock (adder_stage2_adder_stage1_c)
16 report_timing -from adder_stage1 -to adder_stage2_adder_stage1_c \
17                -path_type full_clock_expanded
18
19 source $TETREMAX_PATH/auxx/syn/tmax/pt2max.tcl
20
21 write_delay_paths -launch adder_stage1 \
22     -capture adder_stage2_adder_stage1_c \
23     -max_paths 1 -nworst 1 -delay_type max \
24     $REPORT_PATH/adder_stage1.rpt
```

The four initial commands (line 3 to 6) are responsible to load the synthesized design and the *Synopsys Design Constraints* (SDC) file containing the timing constraints generated by the LCS flow, including all root clocks and generated capture/launch clocks.

Next, the two following *set_multicycle_path* commands (line 10 and 11) specify that the STA analysis between root clocks has no multi-cycle relationship among them. Thus, the STA tool will verify the timing between root clocks in the same cycle. This applies for both setup and hold. After that, the command *set_propagated_clock* can be applied. Usually, this command propagates delays through the clock network of a synchronous circuit. However, as there is no clock network but the control logic of the BD circuit, the *set_propagated_clock* command propagates delays through the control part of the circuit, considering the latency of the delay line inserted in the request signals. Because this must be done to all root clocks, the command in line 12 uses as argument *all_clocks*. The *report_timing* in line 17 displays the timing checks between the root clocks *adder_stage1* and *adder_stage2*. In this case, PrimeTime verifies if data are correctly propagated from the inputs of the registers R_A and R_B (launched by $adder\_stage1$) until the input of the register R_{sum}, which is captured by $adder\_stage2$. Notice that the *report_timing* command uses $adder\_stage1$ as starting point, with the argument *-from*, and its generated capture clock $adder\_stage2\_adder\_stage1$ as endpoint (argument *-to*). For debugging, the *-path_type full_clock_expanded* is an useful argument in this command as it displays not only the delays of the data path but also the delays of the control path, allowing to see exactly all considered delays by PrimeTime during STA analysis.

If the *report_timing* command indicates that the timing constraints have been respected, it is possible to move forward to ATPG. Fortunately, PrimeTime provides interface with TetraMAX, including the possibility to export path-delay information of the data path using the *write_delay_paths* command. Initially, *write_delay_paths* is not a native command on PrimeTime and, thus, must be loaded with a dedicated script called *pt2max.tcl*, which provides extra commands to better interface PrimeTime and TetraMAX. This script can be found in the directory where TetraMAX has been installed. Line 20 presents a *source* command loading the *pt2max.tcl* script. The *$TETRAMAX_PATH* represents the root directory of TetraMAX and must be set previously. After sourcing it, the *write_delay_paths* command is finally available. Line 22 shows how to export the path-delay information of the critical path between $adder\_stage1$ and $adder\_stage2$ with *write_delay_paths*. Similarly to the *report_timing* command, it is necessary to indicate the launch and capture clocks of the given path. As it extracts the same path checked with *report_timing* in line 17, the clock definition remains the same: launch clock $adder\_stage1$ and capture clock $adder\_stage2\_adder\_stage1$. For the sake of simplicity, this example extracts only the critical path (*-max_paths 1 -nworst 1*) for setup analysis (*-delay_type max*). The result of this command can be seen in Listing 3.5, in which the gates composing the critical path and the required transitions – for stimulating it – are indicated.

Listing 3.5: Output of the write_delay_paths command containing path-delay information of the data path between root clocks.

```
1  $path {
2      // from: reg_a_reg_0
3      // to: reg_sum_reg_1
4      $name "adder_stage1-adder_stage2_adder_stage1_c_1";
5      $cycle 0.0;
6      $slack 0.62;
7      $transition {
8          "U16/A" ^; // (BFX4)
9          "U13/A" ^; // (NAND2X5)
10         "U15/A" v; // (XOR2X9)
11         "reg_sum_reg_1_/D" v; // (SDFPRQX4)
12     }
13 }
```

Having the synthesized design and the path-delay information extraction from PrimeTime, we have the required information to perform ATPG for path-delay testing on TetraMAX. Listing 3.6 shows the basic commands to load the design and the path-delay information. Initially, the design netlist is loaded (line 1) with the *read_netlist* command [2]. From line 4 to 6, TetraMAX builds the in-memory simulation model from the design modules that have been read into TetraMAX, loads the SPF file – generated by Design Compiler during DfT specification – and defines the path-delay model as the target fault model. Next, by using the *add_delay_path* (line 9), it is possible to load the path-delay information recently obtained from PrimeTime and create a list of faults for test generation with the *add_faults* command (line 10). At this point, the *report_delay_paths* can be used to verify whether TetraMAX processed correctly the path-delay information. The output of this command displays the same information as Listing 3.5, but in a specific TetraMAX format. On top of that, TetraMAX provides graphical support to better visualize the given path and how it will be stimulated. This graphical support is useful for debugging, specially in the case where it is not straightforward to check the ATPG results. For example, line 13 presents the usage of the *report_delay_paths* once again, but now explicitly targeting the critical path – labeled as *adder_stage1-adder_stage2_adder_stage1_c_1*. Note that the command uses the *-display* argument to enable the graphical interface. Figure 3.9 shows the output of the *report_delay_paths* commands, highlighting the given critical path and all transitions along the path.

[2] Extra *read_netlist* commands loading cell libraries or any sub-modules were omitted for the sake of simplicity.

Listing 3.6: Loading scan-chain specification (SPF file) and path-delay information into TetraMAX. The loaded path-delay information in this example contains the critical path of the first stage (root clock adder_stage1).

```
1  read_netlist $NETLIST_PATH/async_2b_adder.mapped.v
2
3  ## loading scan-chain info provided from Design Compiler
4  run_build_model async_2b_adder
5  run_drc async_2b_adder.spf
6  set_faults -model path_delay
7
8  ## loading path-delay info provided from Primetime
9  add_delay_path $REPORT_PATH/adder_stage1.rpt
10 add_faults -all
11
12 report_delay_paths -all
13 report_delay_paths adder_stage1-adder_stage2_adder_stage1_c_1 \
14                                                  -display -pindata
15
16 ## setting up and firing the ATPG
17 set_atpg -full_seq_atpg
18 run_atpg -auto
19
20 write_patterns async_2b_adder_stage1.stil -serial -format STIL -replace
```

Finally, the path-delay testing setup is done and the ATPG engine can finally generate the test patterns, through the *run_atpg* command (line 18). In line 20, the *write_patterns* command dumps all testing procedure into a *Standard Test Interface Language* (STIL) file. The STIL file provides the stimulus information for the test cycle, including signal timing, test patterns and the stimuli for scan-in, scan-out and test run. However, it is important to notice that the ATPG tool considers that the *Design Under Test* (DUT) has a synchronous design. Because of that, the stimuli in the generated STIL file employs a standard two-pattern testing and it is not compatible with the proposed DfT architecture. Therefore, the STIL demands modifications before use. In order to adapt the test cycle to the proposed DfT architecture, three main modifications must be done in the STIL file:

- Replace the two clock pulses during the test run by a single pulse in the *ext_req* signal;
- Enable HSB configuration, allowing to set controller to launch or capture mode;
- The T_{mode} signal must be disabled during the test run;
- During scan-out, ignore any value but from the capture registers;

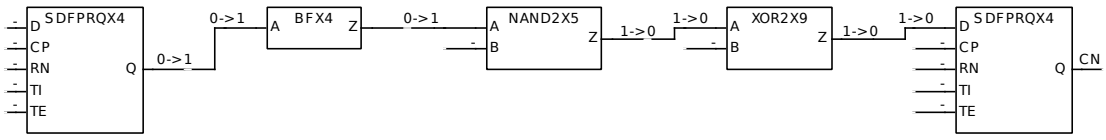

Figure 3.9: Path-delay information displayed on TetraMAX, indicating all transitions along a given path.

3.9.2 A more complex circuit: 128-bit AES cryptocore

This section details the implementation of the proposed testing architecture in a more complex circuit: a 128-bit AES cryptocore. The AES was designed and synthesized in-house, using a 65-nm CMOS technology from STMicroelectronics. Moreover, this study-case circuit was tape-out and sent for fabrication - more details about the prototyped testchip can be seen in Appendix B. Similar to the other study-case circuit, this section also includes stuck-at and at-speed testing.

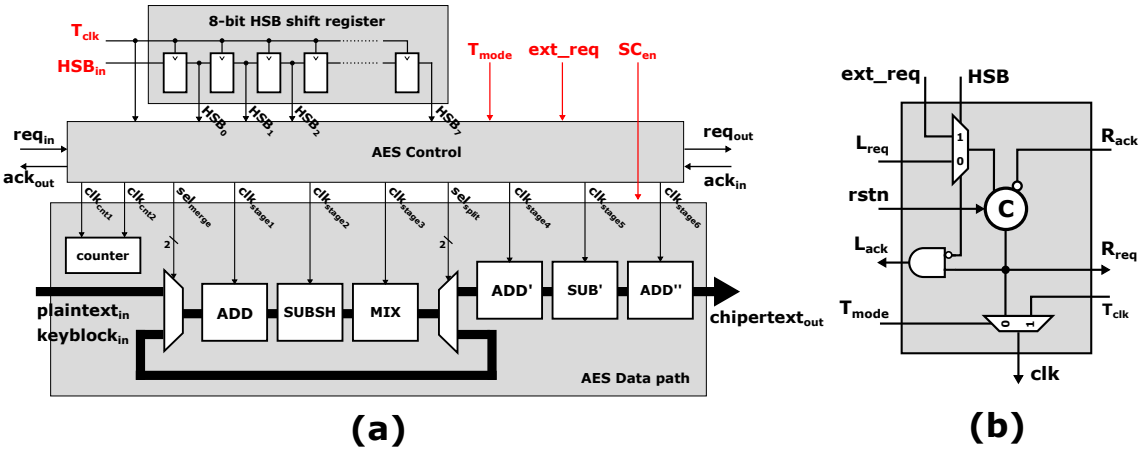

Figure 3.10: Block diagram of the considered micropipeline-based AES core (a) and the testable micropipeline controller (b). Red arrows indicate the additional testing signals.

Figure 3.10 (a) depicts the register stages of the AES core, where each stage is controlled by a separated root clock. The original core comprises two main blocks: control and data path blocks. The control block employs a four-phase handshaking protocol and nteracts with the external handshaking interfaces on one side and control the registers in the data path on the other side. The AES data path block implements a FF-based design and its execution is controlled through 8 root clock signals (clk_{label}) and two one-hot control signals for the merge and split structures (sel_{merge} and sel_{split}), all generated by the control block.

The first main modification in the design is the employment of a testable version of the micropipeline controllers, illustrated in Figure 3.10 (b). The testable version adds the HSB logic (left MUX and AND gate) and a second multiplexer to implement the testing clock bypass. Moreover, a 8-bit shift register has been added into the design to enable the configuration of the HSB logic of each micropipeline controller. Each bit of the HSB shift

Table 3.1: Area results of the original and the proposed testable AES core.

Implementation	Original	Proposed
Combinational (um^2)	45280.80	54752.64
Buffers/Inverters (um^2)	8758.80	10154.64
Non-combinational (um^2)	11146.56	13138.80
Macro/black box (um^2)	86.16	86.16
Total (um^2)	56513.52	67977.60
Area Overhead (%)	-	20.28

register allows to access directly one of the eight root clocks. Note that the T_{clk} controls both AES control block and the HSB shift register. Thus, during scan manipulation, the HSB shift register can be loaded at the same time.

The synthesis step considers the Synopsys' Design Compiler tool with the LCS flow support. All controllers have been manually replaced by their proposed testable counterpart. The LCS flow has been modified to enable the DfT insertion through Synopsys' DFT Compiler. Taking a full-scan approach in this study case, the DFT Compiler replaces all registers for scannable FFs and only considers T_{clk}, SC_{en} and T_{mode} signals to control the scan path. The remaining testing signals (HSB_{in} and *ext_req*) are ignored. As the T_{mode} bypasses the control block when enabled, the DfT tool ignores the AES control block and HSB shift register during scan insertion. Table 3.1 compares the area results of the original and the proposed testable AES core. With the addition of the testable controller, the 8-bit shift register and the scan path, the testable AES presents an area overhead of around 20%. This overhead comes mostly from the scan path, as the shift register and the AES control block only contribute to 0.3% of the area overhead. As the AES control path represents 1.5% of the area consumption, it was expected that the extra circuitry in the control part should not inflict a significant impact.

As the control block is bypassed and the AES data path employs FF registers, it is possible to use a conventional ATPG tool, such as Synopsys' TetraMAX, to generate test patterns for traditional stuck-at testing. Table 3.2 gives a summary of the ATPG for the testable AES core. The ATPG achieves 99.30% of fault coverage considering the full-scan architecture considered in this study case. As the ATPG tool is not able to manage the asynchronous logic of the controllers, the tool is configure to not test the control path, which includes the handshake signals and the signals dedicated to at-speed testing (HSB_{in}

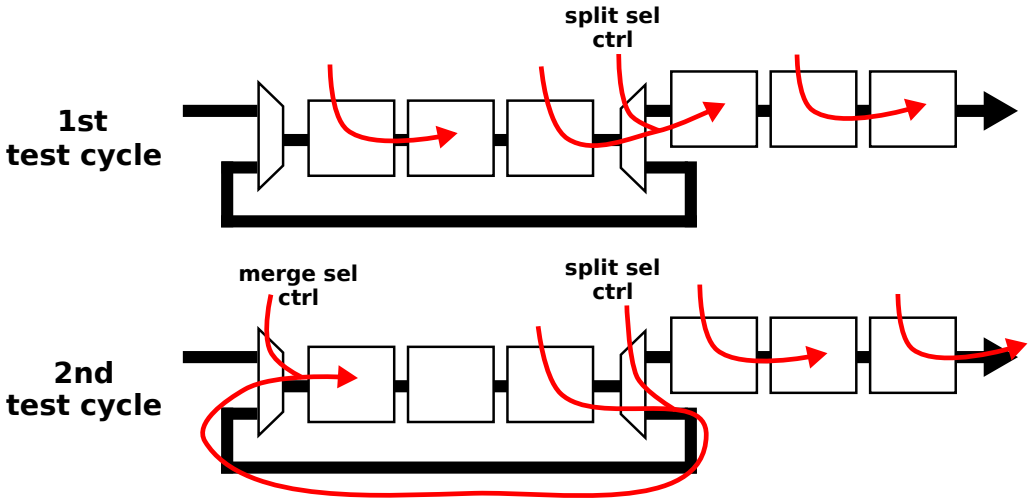

Figure 3.11: HSB configuration for each test cycles during at-speed testing. Arrows represent where each token is inserted and the target path.

and *ext_req*).

Table 3.2: ATPG result summary for stuck-at testing.

Detected	218323
Undetectable	17
ATPG Untestable	1550
Not Detected	0
Total Faults	219890
Test Coverage	99.30%
Scan Patterns	217

During at-speed testing, the proposed architecture verifies whether the delay lines between controllers match the critical paths between controllers – as previously detailed in Equation 3.1 – and also validates the controller operations.

The testing patterns were generated with an ad-hoc approach, stimulating the critical path of each pipeline stage. Here, the LCS flow provides essential information to assist the pattern generation. As the LCS flow creates root clocks and generated launch/capture clocks to enable STA analysis between control and data paths, this same information is used to stimulate the desired critical path. In our case, all critical path information is

obtained with Synopsys' PrimeTime, which is able to read all constraint files created by the LCS flow and indicate the required transitions to stimulate the critical path. For example, Listing 3.7 shows the PrimeTime's output regarding the critical path in the loop between MIX (clk_{stage3}) and ADD (clk_{stage1}) stages. Note that the critical path information provided by PrimeTime contains only the stimuli at the data path – not the control part. Thus, the HSB configuration necessary to properly activate the correct launch/capture clocks was done manually according each case.

Listing 3.7: Example of critical path between two root clocks of the AES circuit. This example considers the critical path in the loop between the third stage (MIX) and the first stage (ADD).

```
$path {
  // from: datapath/round/mix/ \
  //             colmix_reg/outrkey_reg_0__3__3_
  // to: datapath/round/add/ \
  //             addkey_reg/subst_d_reg_3__5_
  $name "aes_stage3-aes_stage1_setup_merge_c" ;
  $cycle 0.0 ;
  $slack 0.679212 ;
  $transition {
    "datapath/U355/D0" ^ ; // (HS65_LH_MUX21X27)
    "datapath/round/add/addkey_comb_in/ \
        keysched1_comb_in/sub0_comb/U286/A" ^ ; //(BFX27)
    "datapath/round/add/addkey_comb_in/ \
        keysched1_comb_in/sub0_comb/U34/A" ^ ;  //(NAND2X43)
    "datapath/round/add/addkey_comb_in/ \
        keysched1_comb_in/sub0_comb/U275/B" v ; //(NOR2X25)
    "datapath/round/add/addkey_comb_in/ \
        keysched1_comb_in/sub0_comb/U127/A" ^ ; //(IVX22)
    "datapath/round/add/addkey_comb_in/ \
        keysched1_comb_in/sub0_comb/U14/A" v ;  //(NOR2X3)
    "datapath/round/add/addkey_comb_in/ \
        keysched1_comb_in/sub0_comb/U47/B" ^ ;  //(NOR2X13)
    "datapath/round/add/addkey_comb_in/ \
        keysched1_comb_in/sub0_comb/U119/C" v ; //(OAI211X3)
    "datapath/round/add/addkey_comb_in/ \
        keysched1_comb_in/sub0_comb/U18/A" ^ ;  //(CBI4I1X3)
    "datapath/round/add/addkey_comb_in/ \
        keysched1_comb_in/sub0_comb/U62/D" v ;  //(CB4I1X18)
    "datapath/round/add/addkey_comb_in/ \
        keysched1_comb_in/sub0_comb/U147/D" v ; //(OAI211X5)
    "datapath/round/add/addkey_reg/ \
        subst_d_reg_3__5_/D" ^ ; //(SDFPRQX4)
    }
}
```

The testing patterns were loaded in the scan chain and the HSB shift register was configured to set which controller would operate in launch or capture mode. Figure 3.11 presents the two testing cycles performed in the study-case circuit, indicating where the test inserts tokens. The end of each arrow represents the last controller where the tokens was propagated and their respective registers contains the resulting pattern to be checked. Moreover, as already discussed in subsection 3.7, the test of split and merge schemes requires that the selector controllers propagate the token to the target branch. The two cycles are required due to the fact that the HSB configuration disables the left handshake communication of the launch controllers. Thus, it is not possible to verify the timing constraints between the launch controllers and any precedent controller.

3.10 Conclusions

This chapter presents an at-speed DfT architecture for bundled-data circuits, applied to micropipeline designs. By modifying the micropipeline controller, it is possible to verify whether the forward timing constraints between the control and data paths have been respected. In addition to that, the architecture still enables traditional stuck-at testing, allowing the use of DfT and ATPG tools already available on the market. The proposed DfT architecture is then implemented in two study-case circuits: a simple two-bit BD adder and a 128-bit AES cryptocore. Considering the two-bit adder, it is presented the synthesis setup to enable the DfT during synthesis and the required LCS configuration. This chapter further details the use of the LCS flow in the ATPG step, showing how the tool can use LCS as reference to collect path-delay information of the circuit. With a 128-bit AES core as study-case circuit, the modifications in the micropipeline controllers and the addition of the HSB shift register only contributes to a total area increase of 0.3%.

The architecture also leverages from the LCS flow to see exactly the stimuli required to activate the critical paths. This allowed us to properly load the scan chain and launch two-pattern testing to verify the timing constraints. However, it is important to highlight that the entire test setup is not yet fully automated and it will be addressed in the future. Another current limitation of the proposed architecture is that, after inserting tokens through the *ext_req* signal, it is not possible to evaluate whether the circuit had finished the test run. In nominal operation, the end of the test could be estimated according an expected worst-case delay, albeit this estimation is not trivial in a voltage scaling scenario, for example. Consequently, the architecture could employ any structure responsible to acknowledge the end of the test run. As a last point, the proposed DfT architecture can be extended to allow at-speed testing for any bundled-data template, covering templates such as Click, MOUSETRAP or GasP.

Part III

Side-channel Analysis of Asynchronous Circuits

4

State-of-the-Art on Hardware Trojan Detection

Contents

4.1 Hardware Trojan Model and Taxonomy

As shown in Fig. 4.1, a HT is decomposed into two parts [72]: (1) a **payload** that is responsible to cause the harmful effects on the functionality/specification of the target IC design; and (2) a **trigger** that is a mechanism to activate the payload – kept inactive until an attacker provokes an activation event. If the HT is an always-on circuit, the activation event happens when the the IC is powered. However, attackers can employ internal or external activation schemes in order to make the payload inactive. Thus, the HT remains stealthy during verification and validation steps, and their harmful effects might be inconspicuous regarding IC manufacturing Process Variations (PVs) as they are such as small parasitic elements in the target IC design. If the HT is always on, it is normally built less perceptible than the payload to reduce impacts on the target IC design and thus prevent its detection. Thereby the HT model is basically a dormant circuit that, once triggered, modifies the system original behavior.

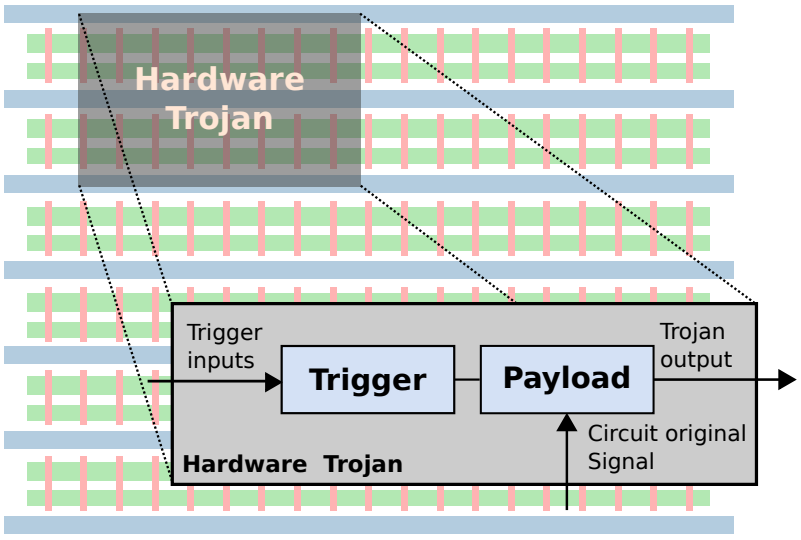

Figure 4.1: Representation of a HT in a standard cell-based IC design. The shadowed part illustrates the area used by the HT. The considered HT model comprises two main logic blocks: a trigger and a payload. The trigger is responsible to activate the payload according to a specific input. Usually, the payload logic remains inactive to avoid detection.

In order to evaluate the risks of HTs, several studies have been reported taxonomies [72–76] abstracting different categories related to the architecture, effects, and insertion of Trojans in ICs. Figure 4.2 summarizes the existing HT taxonomies, highlighting five categories: insertion phase, abstraction level, activation mechanism, effect and location. A discussion about these different categories is presented throughout this section.

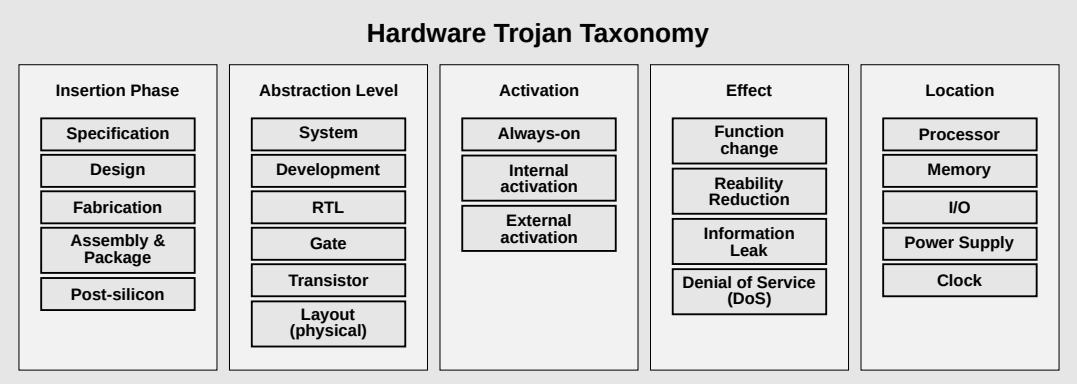

Figure 4.2: Hardware Trojan taxonomy, classified in five main categories: insertion phase, abstraction level, activation mechanism, effect and location.

4.1.1 Insertion

Initially, a HT can be classified according to when it is inserted through the design process. Therefore, the insertion phase of a HT can occur during specification, design, fabrication, assembly & packaging and post-silicon test. Considering the insertion during the specification step, an adversary could intentionally define weak requirements for the system. As a possible result, design reliability may become compromised making the device vulnerable to leak sensitive information. Even if the whole design step could be done entirely in-house and the probability of the insertion of a malicious circuitry is minimal, the simple usage of untrusted tools, libraries, third-party IPs and standard cells may affect the circuit in a harmful way. For instance, untrusted tools may add extra circuitry in the system to introduce backdoors in a genuine design. If any step of the design phase is outsourced, a HT could be directly added to the hardware description files of the genuine circuit. During fabrication, an untrusted foundry, mask shop or their personal are able to retrieve the genuine circuit components and thus predict its behavior and probable applications. Therefore, the design becomes susceptible to modifications. Modifying physical circuits characteristics (sizes and channel doping concentration level) [77] can also increase the circuit vulnerability to fault-based attacks. In the assembly and packaging phase, the IC is encapsulated and the packaged chip is assembled on a *Printed Circuit Board* (PCB) with other hardware components. An adversary may add malicious hardware components surrounding the genuine design to provoke malfunctions or increase leakages. Finally, at the post-silicon testing phase, an adversary is no longer able to modify the genuine IC, however the test set-up, programs or reports may be changed in order to mask the presence of a HT. Moreover, as it is the last step of the IC production flow, it is the last opportunity to detect HT before releasing it to their customers.

4.1.2 Abstraction Level

The abstraction level refers to possible tampers with the design if an adversary has an access to sensitive files at six different abstraction levels: system, development, *Register-Transfer Level* (RTL), gate, transistor and layout. At system level, a HT can simply be alterations in function specifications, protocols, interfaces and constraints of the genuine design. An adversary involved at the system level may add some obscure specifications to give him the control of secret data that is usually not available to the user. As an example, an adversary at the specification phase could change specifications of a *True Random Number Generator* (TRNG) to make it working in a predictable way due to some conditions that only the owner of the HT is aware of. This is able to considerably reduce the reliability of systems based on these architectures and potentially provide confidential data to attackers. During Development, untrusted tools and scripts may present hidden functions, leading designers to generate circuits infected by HT. On top of that, the verification process can be compromised with untrusted simulation tools and testbenches, which could mask the HT effects. Any unreliable third-party vendor is able to insert HTs at this level. At RTL level, a HT can also be a simple modification in genuine RTL codes or constraint files. An attacker can modify circuit functions in order to provoke significant consequences such as failures or out-of-spec behaviors. Attackers (designers) or untrusted code suppliers are possible sources for the HT insertion during the design phase. Next, a gate-level HT comprises the addition or removal of one or more gates in the original netlist. The attacker can also modify *Standard Delay Format* (SDF) files, for instance, changing timing checks, constraints, and delays to hide any effects caused by the netlist modification. Adversaries during the gate design phase are third-party vendors or designers accessing the circuit to implement HT at this level. On transistor-level, it is possible to significantly increase leakages, opening backdoors for attackers to get knowledge about security-oriented circuit internal states. Moreover, transistors may be added to increase critical path delays, leading the circuit to malfunction. Adversaries at the transistor design phase are possibly designers or untrusted tools, libraries and models. Finally, at physical level, original parameters of circuit components are vulnerable even after the layout generation. An attacker can alter original masks, changing transistor geometry or channel doping concentrations. On addition to that, wires can be resized, generating malfunctions or any negative side effects. Adversaries at design and fabrication levels, or third-party mask shops have access to modify the original layout and insert such HT.

4.1.3 Activation Mechanism

The activation mechanism of a HT can be classified as always-on, internal and external activated. If a HT is always activated, its effects on the circuit may upset some device property, making it exposed to verification and validation routines. However, if a HT remains dormant until the deployment phase, its disturbances in the circuit behavior become less

noticeable, creating obstacles for its detection. For this purpose, HT are likely to feature activation mechanisms used to wake-up them under certain conditions accomplished only after the verification and validation phases. Thus, HTs are considered dormant during test routines and hostile after being activated. With an always-on HT, the circuit behavior is always affected by the HT. Consequently, the HT is only composed by its payload and as no trigger. With an internal activation mechanism, the HT is activated when specific internal conditions occur in the circuit. For example, an internal counter may trigger the HT if the counter reaches a certain value. Besides that, internal signal patterns or rare conditions may trigger this type of HT. Considering an external activation mechanism, the HT are activated by an attacker aware of the HT presence in the circuit. In this case, a HT could be triggered when a certain value is set in the logic, such as internal register or even inputs. Thus, attackers knowing the activation mechanism are able to externally wake-up it. Sophisticated trigger mechanisms rely on very rare sequences, conditions or even side-channel attacks, making its detection almost impossible by users, which are not aware of these activation mechanisms.

4.1.4 Effect

A HT may lead the device to different effects depending on the adversary possibilities and intentions. In this category, a HT can be classified in four main effects: function change, reliability reduction, information leak and *Denial of Service* (DoS). Initially, change function is one of the most straightforward effects that a HT can implement. As it suggests, this effect adds or removes original circuit functions. For instance, HTs could lead to improper calculations under specific conditions, compromising the main system operations. Considering now the reduce reliability effect, the HT can downgrade the system performance such as speed reduction, making circuit faulty or downgrading security. This can be used by an attacker to perform side-channel attacks. In other system applications, HT can increase the power consumption, causing a faster battery discharge to interrupt the circuit operation. Another critical effect that HTs could implement is information leak. Mostly in secured applications (e.g cryptography), HTs focus on leaking secured data through primary outputs or side-channel signals. An adversary could add a comparator HT that enables the key leakage whenever a certain input or sequence of outputs is set. Finally, the HT can make the circuit no longer able to work properly, generating a DoS.

4.1.5 Location

Trojans are also classified according to the location they are inserted in the design. In this case, the literature reports five main locations: processor, memory, I/O, power supply and clock. A HT inserted in the processor logic may add and remove instructions of processors, leading it to operate suspicious functions and cause malfunctions. In the memory, an attacker with the control of memory elements may be able to get access to

secured data and clear sensitive data stored in the device. If the I/Os are compromised, pins controlled by the HT may lead the circuit to misbehave, display wrong signals or monitor communications. Next, HTs in the power supply grid may control the device voltages and currents, increasing leakages or causing failures. If the HT affects the circuit clock structure, it may alter circuit frequency or increase clock noise causing glitches or introducing jittery. This threat can cause secure blocks to leak information and create vulnerabilities to side-channel attacks.

4.2 Hardware Trojan Detection

For ensuring the IC trustworthiness, different techniques can be implemented to detect or prevent HT according to the level of trust required for each phase of the design. Several studies have reported comprehensible surveys about most detection methods presented in literature, classifying them in accordance with their approaches. The techniques basically evaluate the deviations caused by HTs on the system behavior or look for possible profiles. To this aim, the designers must be aware of at least a single specific parameter from the genuine device or define a target HT model to be detected. If the deviation produced in the evaluated parameter of a DUTT is greater than an acceptable margin, the DUTT is classified as Trojan infected. In Figure 4.3, a scheme based on prior surveys and works presents the two main categories of testing techniques for HT detection: destructive and non-destructive. Note that this section focuses on non-destructive techniques, as it is the approach selected for the proposed technique in Chapter 5.

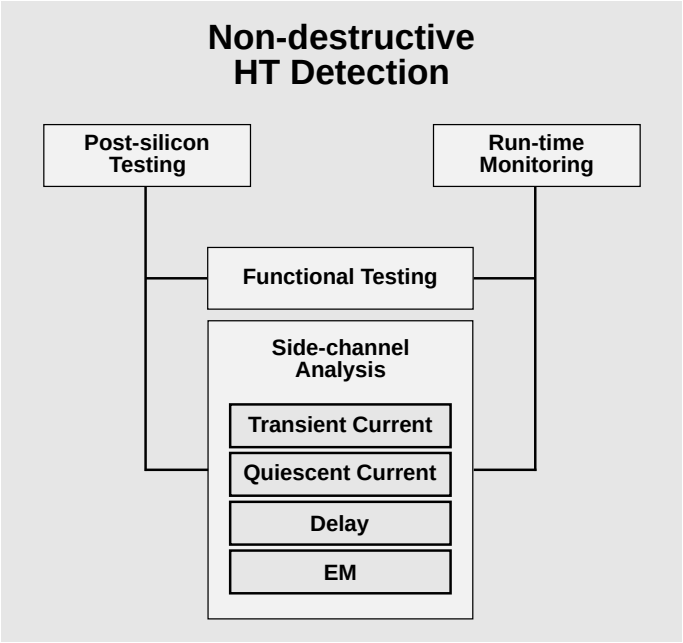

Figure 4.3: Classification of non-destructive HT detection techniques.

The destructive techniques require reverse engineering of the genuine design in order to physically inspect it. A brute-force strategy for HT detection after fabrication is reverse-engineering the manufactured DUTT in order to recover its layout and look for discrepancies in relation to the original trusted one. This approach is possible thanks to high precision optical and *Scanning Electron Microscope* (SEM) after *Chemical Mechanical Polishing* (CMP). Despite presenting reliable results, these techniques feature some drawbacks such as being expensive, time-consuming, destructive, and difficult to be integrated into the regular testing phases. Moreover, to validate the manufactured lot of a particular IC, it would be necessary to sample at a least a few DUTTs to test it. Hence,

even though a DUTT is stated as Trojan-free, it cannot be deployed after the physical inspection.

As indicated in Figure 4.3, the non-destructive techniques are divided into: (1) post-silicon testing techniques that rely on detecting Trojans before the deployment of the device; and (2) run-time monitoring techniques that consist in on-line mechanisms able to detect and indicate – during the normal IC operations – malicious activities or malfunctions caused by Trojans. Functional or logic testing techniques are originated from regular verification and validation phases. Its operation consists in evaluating the behavior of primary outputs and internal nodes of circuit given a set of input vectors. If the DUTT presents suspicious deviations or properties, the design is assumed to be HT infected. Functional tests can therefore be performed to detect Trojans at any step of the IC design. There are different approaches addressing functional testing. A common approach is defining and identifying possible suspicious nets in the netlist. The authors in [78] proposed a method for finding weakly correlated signals or isolated sections in the netlist to find possible HT triggers. In [79], the authors compare HT in the literature in order to define architectural patterns frequently used in HT designs. Then, they implement a score-based classification method to detect Trojans in untrusted netlists. Both methods [78, 79] are able to detect gate-level HT in non-certified netlists based on their assumptions about the HT model. The studies [80–82] propose detecting suspicious activities caused by Trojans in third party IPs. In addition, if a complete trusted specification is available, a high-level golden model can be generated to perform a formal verification method such as *Sequential Equivalence Checking* (SEC) to identify a possible HT. However, without having any trusted specification of the design, the latter is consider as a black box, rendering the HT detection quite challenging. Techniques like in [83, 84] are able to detect Trojans implemented at different levels by applying data vectors at the DUTT primary inputs with the intention of stimulating and activating the HT in order to check possible modifications at the DUTT primary outputs caused by the Trojans. In [85–87], the authors presented test generation strategies to optimize the number of test vectors needed to activate a Trojan. Furthermore, other approaches such as the one presented in [88] are implemented to maximize the probability of triggering Trojans by inputing test patterns based on multiple multiple stimuli of rare logic conditions. This method allows the reduction of the number of required test vectors compared to a weighted random pattern. With the HT activation, the effectiveness of functional tests for its detection is fairly enhanced.

The HT detection techniques based on side-channel analysis focus on the fact that Trojans, even inactivated, cause leakages in terms of power, delays and EM emissions [89]. If a golden model is available, the HT detection is performed by comparing the side-channel traces from certified Trojan-free devices (i.e golden ICs) and DUTTs. In side-channel analysis, security designers have to deal with two main challenges: (1) the PV and environment variations, which possibly masks Trojan effects in the side-channel signals, and (2) the need for a golden model. The effects of PV basically results in an alteration of

circuit parameters such as threshold voltages, channel lengths and oxide thickness. For instance, threshold voltages can approximately fluctuate 20% among its typical value in modern technologies [90]. Thus, ultra-small HT – sized on the order of 100 to 10000 times smaller than the original circuit dimensions – would naturally be masked by PV. This implies that the design and test efforts must be considered in order to reduce or compensate the PV effects. Each method proposes different strategies with this purpose. The need for a golden model is overcome by collecting signatures from golden references obtained from devices certified by physical inspection or certificated fabrication process. Furthermore, the literature also explores alternatives, such as the possibility of generating fingerprints only based on trusted simulation models and measurements from process control monitors, without requiring certified ICs [91]. An illustrative example of the detection procedure is presented in Figure 4.4. A certain input vector is applied at the primary inputs of a set of golden ICs and thus, the side-channel signals are collected to produce a golden signature in a space of parameters. The same test procedure is applied to a set of DUTT devices, producing data to be compared with this golden signature. In Figure 4.4, the golden data are used to generate the "HT-free zone". This zone can be generated through *Minimum Volume Enclosing Ellipsoid* (MVEE) or any classification algorithm able to sort the devices in a HT-free class. Different side-channel signals such as power consumption, delay, EM, temperature, oscillation frequency or even substrate impedance are used to generate signatures from golden devices and DUTTs. Besides that, combination of intrinsically related parameters were also proposed as a solution to compensate PV effects.

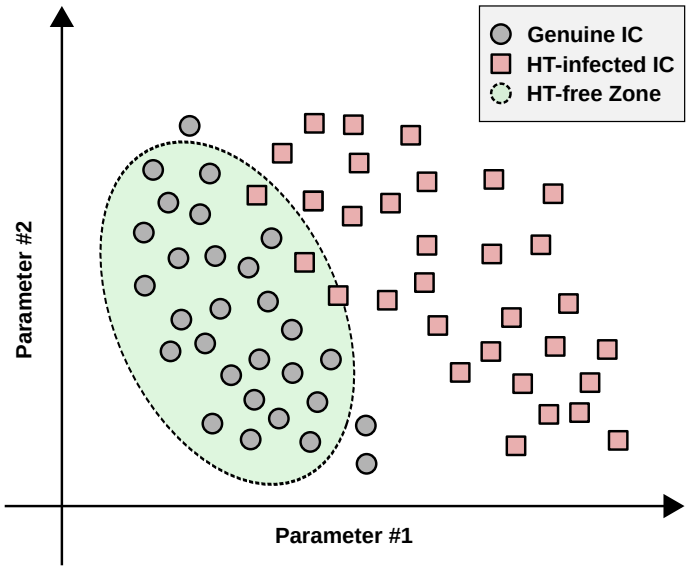

Figure 4.4: Illustrative example of HT detection through side-channel analysis. The example considers two generic parameters extracted from a golden reference to be able to differ HT-free and HT-infected devices.

4.2.1 Power Consumption Monitoring

Regarding power consumption, it is possible to extract dynamic and static information from the power supply. The dynamic and static consumption can be obtained by monitoring the *Transient Supply Current* (IDDT) and IDDQ, respectively. As HT circuitries share the same power supply with the target system, traces obtained from the power supply pins can track possible alterations caused by a Trojan, thanks to the evaluation of the generated current trace. For this purpose, the switching activity in the circuit is used to gain information about the amount and the type of gates consuming dynamic power. The first Trojan detection method using side-channel analysis [92] used indeed the power trace generated by the transient current to gather a set of fingerprints of Trojan-free and Trojan-infected DUTTs. In this study, a *Karhunen-Loève* (KL) expansion is used to eliminate the measurement noise and therefore performs the detection. Alternatively, further studies have also addressed detecting Trojans even in the presence of PV. The approaches rely on measuring multiple power ports or pads individually in order to isolate the Trojan effects to a specific chip location and thus increase its relative impact. In [93, 94], the strategy was to integrate the total current from a specific pad, while in [95, 96], similar methodologies used the IDDT provided by each power port. Another approach present in the literature is to insert built-in sensors into the design able to detect anomalies in the IDDT signature [94]. Moreover, the built-in sensors can be scattered through all the circuit surface, partitioning the design in smaller sections and increasing the sensitivity to the sensors in order to detect small HTs. Silicon demonstrations of Trojan designs and detection in a wireless cryptographic ICs are shown in [97]. In this work, the authors present an always-on Trojan able to leak keys of a 128-bits AES core and detect it by measuring the transmission power obtained on different inputs. Considering now the static power consumption, IDDQ added by a HT is another traceable parameter to identify it, even in scenarios where there is no switching activity in the Trojan. In [98, 99], the authors demonstrated the effectiveness of analyzing the IDDQs simultaneously measured from multiple locations of the chip. A test structure is used in order to emulate the Trojans in different positions in the circuit and perform its detection by measuring multiple power ports. Despite requiring distinct input vectors, test procedures for obtaining IDDQ are very similar to the ones used in IDDT. Most of the proposed methods using IDDQ consider it as an auxiliary signal in multiple parameter analysis.

4.2.2 Delay Monitoring

Another consequent effect of a HT infection is the increased delay encountered on specific nodes of the original circuit. A Trojan inserted between two blocks modifies the authentic data path and thus increases the delay in such paths. Another possible implementation is to directly connect the Trojan on an original circuit node – without necessarily cutting lines – increasing the fan-out and capacitive loads of the previous gates and therefore, the

path delay. Measuring path delays in sequential circuits after fabrication, albeit is not a simple task. If no extra on-chip circuitry is used for this purpose, it is only possible to measure path delays, which originate from primary inputs and terminate at primary outputs. Besides that, as in synchronous circuits the clock controls the data flow stage-to-stage, it is not possible to measure the delay of each stage. For this reason, extra on-chip circuits such as full-scans must be used to enable the measurement of these delays during the post-silicon testing phase. Indeed, Trojan detection techniques based on path delay rely on using mechanisms able to output variables indicating the path delay. In [100], a delay characterization is done by a secondary clock signal controlling shadow registers. Other detection methods propose improving this technique effectiveness by using the comparator outputs as chip authentication [101]. In [102], the authors accentuates the HT impact by generating a vector that sensitizes the shortest path passing via the HT location. In addition, other techniques used embedded test structures for on-chip measurements of path delays [103–105], while in [106] a framework based on self-authentication is proposed. In [107], an effective manner to gather the fingerprint of all path delays is proposed while a more recent approach uses latch-based structures to compare relative delays of different paths in the circuit to identify discrepancies [108].

4.2.3 EM, Thermal and Substrate Monitoring

Beyond delay and power monitoring, the engineers and researchers also consider the extraction of side-channel data such as EM emissions, temperature and substrate impedance in order to detect anomalies caused by HTs. First, switching activity in Trojans nets is a source of unsuspected EM emissions. Non-invasive techniques are therefore used to track DUTT emissions and compare them with a golden reference. Prior studies [109, 110] use similar approaches to detect Trojans inserted in different locations in FPGAs. The analysis consists in scanning the whole circuit with an EM probe able to collect data from different spots of the circuit. The golden and DUTT data are compared in order to generate a map depicting the obtained differences between them. In [111], the authors consider EM-based HT detection with the use of machine learning algorithms. Despite of applying a conventional supervised machine learning algorithm, it is also presented an unsupervised version, requiring no golden reference during classification. The literature also presents another methodology able to detect Trojans using EM emissions without the need of golden ICs nor a netlist [112]. In this case, RTL simulations are used to generate patterns, which will be compared with the ones obtained from the FPGA to perform the detection of activated Trojans. Regarding temperature, authors focus on capturing thermal signatures of the IC surface and evaluate whether there is any non-expected thermal activity in the circuit. This side-channel analysis may require the insertion of thermal sensors [113] into the design or it can take a completely non-intrusive approach, in which the thermal signatures are captured externally [114] and, consequently, avoiding area over-

head. Finally, authors in [115] reports HT detection technique based on the substrate impedance of the circuit. Their work utilizes *Bulk Built-in Current Sensor* (BBICS) scattered through the design in order to detect the presence of abnormal current peak flowing from the bulk (body). Interesting, these sensors are originally focused on online testing applications for detecting radiations or laser-induced transient currents. The proposed technique compares the BBICS behavior of golden and DUTT devices, while applying current pulses at the body of the circuit.

4.2.4 Multi-parameter Monitoring

Another efficient approach consists in combining signatures extracted from different side-channels and thus, increasing the amount of obtained data to enhance the Trojan detection effectiveness. The intrinsic relation between different side-channel signals is a clever strategy to compensate PV effects. Take as an example the relationship between delay and power consumption. If PV acts increasing the power consumption of a specific logic gate, its consequent effect is reducing its path delay. Thus, the value of the power consumption of a given gate allows predicting its path delay in this PV environment. The authors in [116] take advantage of this to propose a detection technique based on transient currents and delays – obtained indirectly by the maximal operation frequency. In [117], the relation between transient and quiescent currents is used while the authors in [118] use delays and electromagnetic measurements to detect HT. An unified framework is proposed in [119] providing detection results for all side-channel signals and thereafter combining them. Another interesting approach is presented in [120], which introduces the clock sweeping technique. In this case, the authors generate delay and power signatures with different clock frequencies. Then, the technique considers one of the signatures according to the path size and applies statistical analysis in order to classify the DUTTs. Logic and side-channel signals are evaluated by run-time monitoring structures embedded in the original design. In this approach, if a Trojan is activated after the deployment phase, the monitoring system is able to generate a flag indicating a Trojan alert. In [121, 122], the techniques treat of the interference in circuit functionalities caused by active HT as faults and thus detecting it.

4.3 Conclusions

In this chapter, the different possible vulnerabilities to HTs have been presented through the ICs design flow. Thereafter, a full HT taxonomy depicting different scenarios in which malicious circuits can be inserted are presented showing that regular steps of IC production are susceptible to HT insertion. Moreover, diverse Trojans, implemented at different abstraction levels, reported in recent bibliography illustrates possible attacks and concerns

that security designers must consider during the design of their ICs. In the second section, the main HT detection techniques have been classified according to their strategies. It is shown that many works in the literature have already address different vulnerable production steps and propose detection solutions. The Trojan concern is, however, far from being overcome. Adversaries aware of the main detection methods may develop more sophisticated Trojans able to be undetectable by the already proposed techniques. Fortunately, as a lot of methods have been proposed, designing an undetectable Trojan becomes challenging. Thus, developing innovative techniques is the key to make attacks difficult and increase the ICs trustworthiness against Trojans.

5

Hardware Trojan Detection Technique for Asynchronous Circuits

Contents

In the recent decades, with the increasing globalization process, microelectronic companies have relied on outsourcing different design steps in order to minimize costs and time-to-market. As a consequence, IC production chains employ multiple companies often based in different continents. Despite the outsourcing benefits, serious security concerns today affect all phases of the IC-design flow. The usage of third-party IP, tools, and manufacturing hampers the design full certification, making it vulnerable to malicious insertions often called HT [72]. HT usually are inserted in systems to change their expected functionality, leak data or even make them able to run malicious functions. With these potential vulnerabilities, security-aware and military applications have pushed the researches towards the implementation of trustworthy ICs and robust techniques able to detect the presence of HT.

Depending on the HT functionality and the attacker's creativity, detecting HTs is challenging as they might be triggered only by a specific input sequence not used during the standard functional testing steps. In order to cope with the HT logic masking, diverse effective test-time methods based on side-channel analysis have been devised to detect HT without destructing the DUTT [92–95, 97, 98, 102, 111, 113, 115, 123–127]. They rely on the extraction of side-channel data – such as EM emission [111], power consumption [93–95, 97, 98, 123], temperature [113], oscillation frequency [124], delay [102, 125–127], or substrate impedance [115] – of selected DUTT samples and infer the presence of a HT or not. If the data obtained from a selected DUTT sample significantly deviates from a reference or golden model, the technique flags a HT. Despite of the multiple available HT detection techniques in the literature, only a few works present strategies to detect HTs in asynchronous circuits [125–127].

Asynchronous circuits have inherent reliable and security design features [128–132] thanks to the use of delay insensitive encoding and local communication protocols instead of a global clock. Because of their security features, researchers have also explored HT detection techniques dedicated for asynchronous circuits. Lodhi et al. [125, 126] evaluate the delay propagation of mixed synchronous/asynchronous systems to detect malicious circuitry. Recently, Guimarães et al. [127] analyze the transient current I_{ddt} peaks and propagation delays of asynchronous QDI circuits and classify the extracted data through a MVEE. Unlike [127], the proposed HT detection technique in this thesis analyzes the entire shape of the IDDT curve, which will be referred as **current signature** in the sequel of this text. As an idle HT can be model as a parasitic component in the circuit, such as a capacitance [120], the HT presence may create distortions in the switching activity and, consequently, in the current signature as well. For instance, take the example of the circuit in Figure 5.1 (a) with three generic logic gates and standby Trojan at the output of the logic gate C. Figure 5.1 (b) represents the current signature of the supply voltage V_{DD} during switching. In this case, we assume that the inputs generate switching activity in all logic gates. This is translated with two current peaks, where the first one represents gate A and B switching in parallel and the second one, gate C. If a Trojan increases the capacitance

in the given node, despite of being disabled for the moment, the logic gate driving the infected node should take longer to drive it – the HT adds parasitic capacitances in the node but the driving strength is still the same. The extra capacitance should deform the peak related to the switching activity of C, as illustrated in Figure 5.1. This is the kind of distortion the proposed HT detection technique desires to highlight.

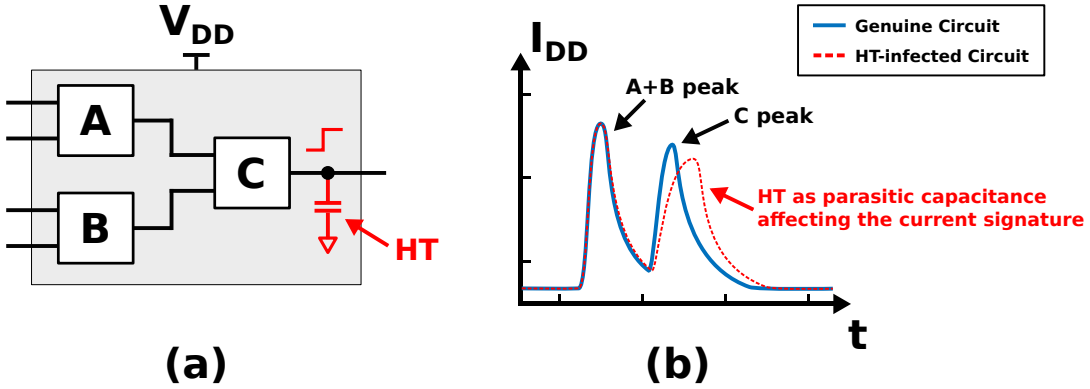

Figure 5.1: Example of current side-effect of an standby HT in a generic circuit. The parasitic capacitance of the Trojan deviates the current signature of the supply voltage V_{dd}.

Then, it deals with them by using a machine-learning algorithm called OC-SVM for classifying DUTT samples into HT-infected or Trojan-free. Liu et al. [97] also use OC-SVM for classifying DUTT samples, however their work differs from our technique into three aspects: (1) they focus on synchronous circuits; (2) stimulate differently the DUTT, and (3) classify DUTT samples by considering as input features the power consumption components of each DUTT operation.

Otherwise, our technique deals with asynchronous circuits that intrinsically allow to individually analyze current signatures from different DUTT parts, and thus better distinguishing HT-induced modifications on the current signature. Then, instead of using as input features the current signature averages of DUTT operations [97], or current signature peaks and delays [127], we take into account each point of individual current signatures, i.e. current points measured from different DUTT parts, as an input feature to train OC-SVM and classify DUTT samples. Considering, for instance, an asynchronous DUTT composed of three pipeline stages (S_0, S_1, and S_2) connected in a linear fashion. By propagating a single input vector – or a data token whether we consider the asynchronous nomenclature in chapter 1 – through the asynchronous DUTT stages, we are able to extract from the global current signature I_{ddt} three individual current signatures I_{ddt0}, I_{ddt1}, and I_{ddt2}, each one within a different time frame, and each one from a different stage. In fact, thanks to the absence of a clock network and the local handshaking communication between the asynchronous DUTT stages, a single input vector is not able to produce switching activity in idle stages, hence each individual current signature (I_{ddt0}, I_{ddt1}, or I_{ddt2}) carries only the switching activity of the active stage in that specific time

frame, making the detection of HT-induced modifications on the global current signature signature I_{ddt} easier. Moreover, we also take advantage of the delay insensitivity of QDI asynchronous circuits through voltage scaling. For each considered supply voltage, current signatures can be extracted for a given data path, highlighting distortions caused by HTs.

5.1 Exploiting the Current Signatures of Asynchronous Circuits

In a synchronous circuit, the global *clock* signal normally controls several pipeline stages – S_0, S_1, and S_2 in Figure 5.2 (a) – switching all of them. If a single vector (herein token) stimulates the primary input $data_0$ of the stage S_0, the global current signature I_{ddt} is influenced – during the first clock period – by the components I_{ddt0}, I_{ddt1}, and I_{ddt2} in Figure 5.2 (c), respectively from: the token activity in stage S_0; and the switching activity of clock tree buffers and input circuitry of registers R_1 and R_2 in idle stages S_1 and S_2, which are both not computing any tokens. Clock-gating, power-gating, techniques for isolating the supply voltage of each stage with multiple supply pins are able to mitigate the current interference of components I_{ddt1} and I_{ddt2} at the expense of additional hardware mechanisms. On the other hand, QDI asynchronous circuits intrinsically copes with these issues by employing local handshake schemes.

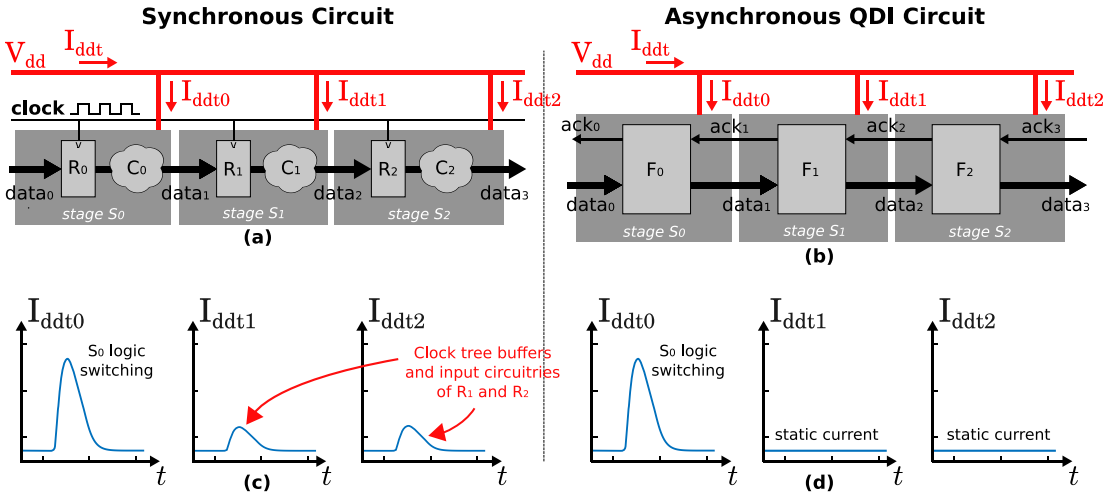

Figure 5.2: Example of a 3-stage linear pipeline: (a) synchronous and (b) QDI asynchronous circuits. The plots in (c) and (d) represent the current signature of each pipeline stage in (a) and (b), respectively, during the propagation of a single input vector through the stage S_0. This example highlights the current influence that occurs in synchronous circuits. Even if only one stage is computing its inputs, the remaining stages still affects the total current signature of $I_{ddt}(t)$.

Taking the same example as in (c), Figure 5.2 (d) illustrates the current signatures of each pipeline stage in (b). In this case, while S_0 computes the input vector, S_1 and S_2 only contribute with static currents. The same applies if the token propagates to further pipeline stages. When the token arrives at S_1, the previous stage S_0 has already computed the token and remains idle as well as S_2. Next, S_2 finally computes the token and S_0 and S_1 are now idle, only contributing with static currents. As QDI circuits avoid the usage of a global clock, the influence from the clock tree is also eliminated. Due to its

local handshaking scheme, QDI circuits also avoid the influence of parallel switching activity of idle stages. These features provide a pipeline-level current signature isolation, making more significant any discrepancies caused by a HT. However, in case the logic path employs fork structures, the propagated token generates parallel switching activity in all branches, implying the current signature comprises the components of each branch.

5.2 Technique Steps

The proposed technique comprises seven main steps depicted in Fig. 5.3: (A) define stimulus procedure; (B) certify golden DUTT samples; (C) extract and partition golden DUTT current signatures; (D) OC-SVM training; (E) select a subset of DUTT samples from a foundry susceptible to HT attacks; (F) extract and partition selected DUTT current signatures and (G) classify randomly selected DUTT samples through OC-SVM classification.

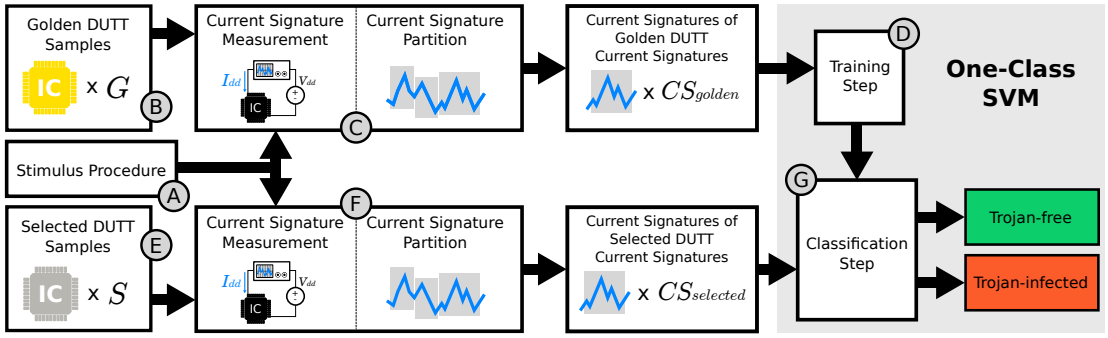

Figure 5.3: Proposed HT detection flow highlighted in seven main steps. Both golden and selected DUTT current signatures are obtained using the same the stimulus procedure, extraction and partition method. While the OC-SVM training step considers golden DUTT current signatures to generate the classifiers, the OC-SVM classification step utilizes the selected DUTT current signatures to classify whether the selected DUTT samples are Trojan-free or Trojan-infected.

5.2.1 Stimulus Procedure

The stimulus procedure is responsible to insert a single vector at primary inputs in order to propagate a single data token. Considering the current signature characteristics in QDI circuits detailed in section 5.1, the single data token stimulates a single pipeline data path of the DUTT, generating a current signature without the dynamic current components of other pipeline data paths. For a complete test, the stimulus procedure generates a set of tokens, not simultaneously, to activate all DUTT nodes and to cover any possible HT-infected node. It only provides a new data token to the DUTT primary inputs whether the

previous token has already been propagated and all logic is idle (with no switching activity). Consequently, the testing time is a function of the delay to propagate a data token through a pipeline data path, multiplied by the number of tokens required to stimulate all pipeline data paths. As QDI circuits employ multi-rail encoding in their data paths, any input vector will generate equivalent number of transitions for a given data path. This indicates that the input vector quality for HT detection depends directly to its ability to avoid parallel activity from other data paths.

5.2.2 Golden DUTT Samples

Initially, the proposed technique requires a reliable reference to differentiate Trojan-free from Trojan-infected DUTT samples. This reference comprises a small set of Trojan-free ICs called golden DUTT samples, which are certified after fabrication in order to guarantee a set of G DUTT samples with no HTs (i.e golden). In this step, we assume the golden DUTT samples can be obtained from a set of DUTT samples fabricated in an untrusted foundry by destructive reverse-engineering [116]. The current signatures of the golden DUTT samples are applied to train a machine learning algorithm, further detailed in subsection 5.2.4, which will be able to distinguish discrepancies in current signatures caused by PV and a HT. However, the minimum number of golden DUTT samples to train the *Support Vector Machine* (SVM) is highly dependent of the PV statistical distribution and the DUTT itself. The available number of golden DUTT samples G are divided in two subsets: training and cross-validation subsets. As the name suggests, the training subset is reserved for the training of the machine learning algorithm, while the cross-validation subset is used to evaluate the classifier generated in subsection 5.2.4.

5.2.3 Golden DUTT Current Signature Extraction and Partition

This step consists in reading the I_{ddt} of the G golden DUTT samples, as indicated in Fig. 5.3. By stimulating each golden DUTT sample with the defined stimulus procedure, the technique extracts C current signatures, each one corresponding to the propagation of a single data token. Then, each current signature is partitioned according to the number of pipeline stages PS in the respective path. For instance, consider the circuit in Fig. 5.2 (b), which contains $PS = 3$ pipeline stages, the current signature is divided in three parts (I_{ddt0}, I_{ddt1}, I_{iddt2}). The time windows for each pipeline stage is defined according to its propagation delay. Moreover, in order to extract more data from each golden DUTT sample, the stimulus procedure can be executed with different V_{dd}. As operations of QDI circuits tolerate the change of V_{dd}, the stimuli rerun only requires the V_{dd} level change itself, without any extra setup. Having extracted and partitioned all current signatures, the proposed technique obtains the CS_{golden} golden DUTT current signatures. Equation 5.1 indicates the relation between CS_{golden} and the number of golden DUTT samples G, the employed supply voltages SV, the path current signatures C and the number of pipeline

stages of each path PS. The use of multiple V_{dd} could be easily compared as clock sweeping HT detection [120] in synchronous circuit, however our “voltage sweep" only focuses on highlighting distortions caused by HTs.

$$CS_{golden} = G \times SV \times C \times \sum_{i=1}^{C} PS_{(i)} \tag{5.1}$$

5.2.4 OC-SVM Training

During the training phase, the OC-SVM algorithm learns how to differentiate Trojan-free and Trojan-infected DUTT samples. In our case, the algorithm only considers the Trojan-free class, i.e. the available golden DUTT samples. That means the algorithm itself is capable to classify Trojan-free and **not** Trojan-free DUTT samples, in which the latter is labeled as Trojan-infected.

The technique employs an approach similar to [111] and considers each current signature extracted point as a training feature. In that way, the OC-SVM generalizes the current signature behavior and establishes the lower and upper margins of each extracted point. For instance, if $CS_{golden} = 50$ and each current signature contains 30 extracted points, the OC-SVM training matrix will have 50×30 dimensions. For each pipeline stage, an OC-SVM receives the respective current signatures and calculates a classifier. Considering another example, if $PS = 3$, three OC-SVMs are trained. Each classifier generalizes the idea of a Trojan-free current signature of a given stage. If a current signature significantly deviates from what was learned, the classifier indicates the DUTT sample as Trojan-infected. Otherwise, the DUTT sample is Trojan-free. To measure accuracy, the technique evaluates the cross-validation subset on the generated classifiers. The accuracy of a classifier n, $Acc(n)$ is defined in Equation 5.2, in which $\#CorrClass\_cross\_val\_samples(n)$ is the number of correctly classified golden DUTT samples and $\#Cross\_val\_samples$ is the total number of golden DUTT samples in the cross-validation subset.

$$Acc(n) = \frac{\#CorrClass\_cross\_val\_samples(n)}{\#Cross\_val\_samples} \tag{5.2}$$

If the classification accuracy of at least one of the classifiers is not satisfactory, the number of golden DUTT samples in the training subset is increased. To have an idea of the accuracy independently of the golden DUTT samples used, it is necessary to estimate the mean and standard deviation of the classification accuracy over multiple training rounds with different golden DUTT samples but maintaining a fixed size of the training subset.

5.2.5 Selected DUTT Samples

The selected DUTT samples represent a subset of S ICs fabricated in a third-party foundry where the designer cannot guarantee a secure production chain, thus susceptible to at-

tacks. Due to this assumption of untrustworthiness, the OC-SVM deals with the current signature data from the selected DUTT samples and classify them, pinpointing whether there is a HT into the DUTT or not. Assuming that a HT affects all ICs fabricated on a untrustworthy foundry, the minimum number of selected DUTT samples can be defined through a statistical sampling methodology. They should be sufficient so that the chance of the majority of the selected DUTT samples being correctly classified respects a chosen confidence level with respect to the statistical results obtained in subsection 5.2.4 on all the generated classifiers.

5.2.6 Selected DUTT Current Signature Extraction and Partition

This step takes the identical approach as section 5.2.3 – same stimulus procedure and current signature manipulation. However, it considers the selected DUTT samples from a vulnerable foundry. If the golden DUTT current signature extraction step consider *SV* supply voltages, the same applies in this step. Consequently, Equation 5.3 represents the number of obtained selected DUTT current signatures $CS_{selected}$ after extraction and partition. $CS_{selected}$ considers the number of selected DUTT samples *S*, employed supply voltages *SV*, path current signatures *C* and number of pipeline stages of each path *PS*.

$$CS_{selected} = S \times SV \times C \times \sum_{i=1}^{C} PS_{(i)} \tag{5.3}$$

5.2.7 OC-SVM Classification

At the last step, the classification step uses the calculated classifiers to predict the class of the selected DUTT samples. A selected DUTT sample is said to be infected if any of the classifiers generated classifies it as Trojan-infected. If the OC-SVMs classify the selected DUTT samples as Trojan-free or infected, the same applies for the whole IC set. At this point, the SVM classifiers are already generated and, then, the classification step can be speed up with a hardware-based implementation – running in an embedded computer or FPGA platform.

5.3 Simulation Experiments

The experiments described in this section analyze the effectiveness of the technique proposed in by inserting single HTs inside a case-study DUTT. This section presents technical information regarding the experiment setup, as well as discussions on the results obtained through the usage of the OC-SVM algorithm.

5.3.1 Experimental Setup

As a case-study DUTT, an 8-bit QDI asynchronous *Arithmetic Logic Unit* (ALU) [127] has been designed in a 28-nm *Fully Depleted Silicon-On-Insulator* (FD-SOI) technology from STMicroelectronics. Figure 5.4 illustrates the experimental setup with the architecture of the case-study circuit. The ALU contains 13974 transistors, distributed in three pipeline stages ($PS = 3$). We have considered as case-study seven different sizes of comparator-based Trojans (190, 142, 102, 62, 52, 44 and 34 transistors) and a pass-transistor-based Trojan [133] (30 transistors). All considered HTs have been disabled, implying they have not been triggered during experiments to reproduce the worst scenario to detect them. A single HT has been tested in each pipeline stage of the DUTT, proving a total of 24 different HT test cases – considering the circuit has three stages. All simulations have been performed at nominal temperature (25°C) and corners (TT). The current signatures has been extracted from the DUTT operating with a supply voltage V_{dd} at 1.0 V and at 0.8 V ($SV = 2$). The stimulus procedure detailed in section 5.2.1 has been repeated for each supply voltage condition.

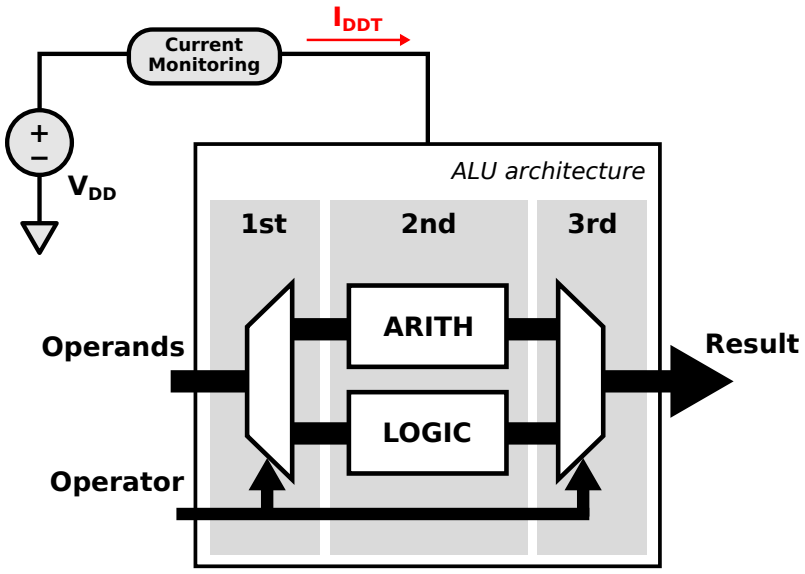

Figure 5.4: Experimental setup overview with the architecture of the study-case circuit. The setup observes the current of the supply voltage V_{DD} in order to see the current behavior while a token is propagated through the case-study circuit.

The current signature data from golden DUTT samples have been represented by $G = 400$ *Monte Carlo* (MC) simulations of a Trojan-free ALU version considering a 3-sigma yield. The experiments also use 50 MC simulations for each HT test case. In total, the data from the Trojan-infected DUTT samples has been represented by current signature resulting from 1200 MC simulations. During simulation, a single data token ($C = 1$) has been propagated through the ALU, stimulating the exact node where the HT has been connected, without triggering it. The data token stimulates half of the Trojan nodes

connected in the original circuit – except for the pass-transistor-based Trojan – showing a more realistic scenario for our detection technique.

5.3.2 OC-SVM Results and Discussion

Fig. 5.5 shows an example of a OC-SVM classifier using a *Radial Basis Function* (RBF) kernel for the third stage of the ALU. The shadowed area represents the classifier generated by the OC-SVM based on the data from the Trojan-free DUTT samples. The red asterisk points represent the data of selected DUTT samples infected by the smallest comparator-based HT (34 transistors), while the solid blue points represents the data of Trojan-free DUTT samples. Note that those selected Trojan-free DUTT samples consist of a cross-validation subset of golden samples. For the sake of illustration, the classifier dimensions have been reduced to two through *Principal Component Analysis* (PCA) [134] – the original data distribution has 42 dimensions and could not be displayed properly. PCA generates a dimension-reduced subspace to represent the data retaining the maximum possible precision represented on a targeted dimensionality. The OC-SVM classifier is able to classify correctly Trojan-free and Trojan-infected DUTT samples with high accuracy. This is an improvement upon the conducted work in [127], as indicated in Table 5.1. Considering the HT with 102 transistors, the former technique provides an accuracy of 77.67%, whereas the proposed technique is able to achieve 100%. The proposed technique maintains the accuracy of 100% even on smaller, i.e. harder to detect, HTs, as the comparator-based HT containing 34 transistors. As the proposed technique considers all current signature sample points and multiple supply voltages – different from the current peak vs. global delay analysis on [127] – the OC-SVMs utilizes more current information, enabling a more sensitive analysis.

Fig. 5.6 presents the classification accuracy according to the number of DUTT samples used for OC-SVM training. Similar to the results in Fig. 5.5, our experiments also consider cross-validation subsets to represent selected Trojan-free DUTT samples. The higher the number of training DUTT samples, the higher the capability of the OC-SVM to correctly classify DUTT samples as Trojan-free – as indicated by the solid blue line in Fig. 5.6. In an under-fitting situation, the OC-SVM is not able to distinguish the current signature distortions caused by PV or a HT, flagging most of the golden DUTT samples as Trojan-infected. By increasing the number of training samples, the OC-SVM further learns how PV affects the current signature. If 100, 150 and 200 DUTT samples are used in the training step, the selected Trojan-free DUTT samples will be correctly classified in 90%, 95% and 96% of the cases, respectively. The Trojan-free classification accuracy reaches 98% with 270 DUTT samples or higher. Independently of the number of DUTT samples used for OC-SVM training, the classification accuracy remains at 100% for all comparator-based HTs. This implies that the proposed technique achieves Trojan-free accuracy classification without losing the capability to detect HTs as large as the considered

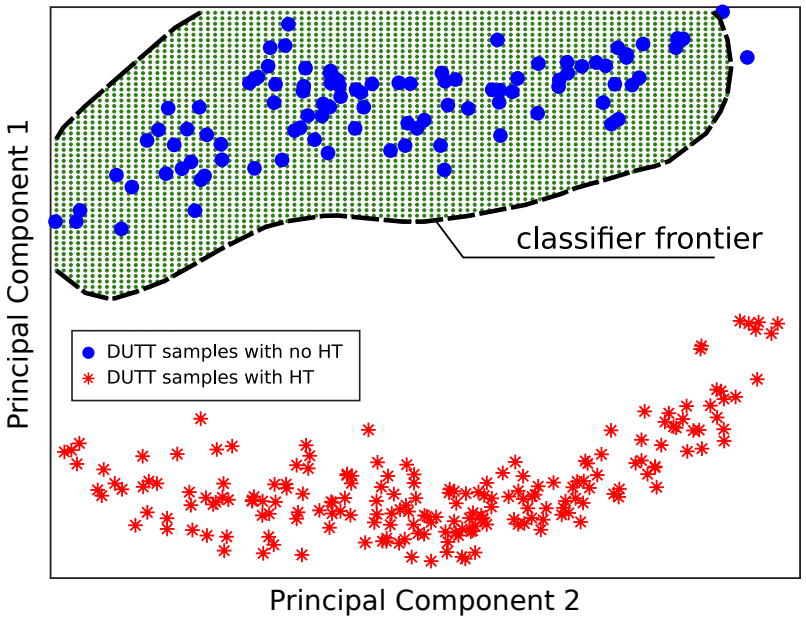

Figure 5.5: Dimension-reduced representation of the OC-SVM classifier for the third stage of the ALU. Shadowed area represents the generated classifier after Principal Component Analysis (PCA).

comparator-based ones. On the other hand, if the selected DUTT samples are infected with pass-transistor-based HTs – traced red lines – the classification accuracy is reduced as such HTs are furtive and small (30 transistors), only slightly affecting the current signature of the DUTT. In this case, this trade-off highlights the importance of selecting an optimal number of training samples that provides high precision for Trojan-free classification as well as high precision to detect small HTs.

Table 5.1: Comparison table of HT detection accuracy of the proposed technique and the conducted work in [127] using the same case-study DUTT. The results of the proposed technique employ 100 golden DUTT samples as training samples for the OC-SVM algorithm. Trojan size column represents the number of transistors used.

Trojan Size	Trojan nodes	Stimulated nodes	Guimarães et al. [127]				Proposed technique			
			HT detection accuracy – according HT location (%)							
			1st	2nd	3rd	AVG	1st	2nd	3rd	AVG
190	8	4	100	100	100	100	100%			
142	8	4	100	94	90	94.67				
102	8	4	100	87	46	77.67				
62	7	3	-							
52	6	3								
44	5	2								
34	4	2								
30	2	2					72.6	84.5	91.8	82.9

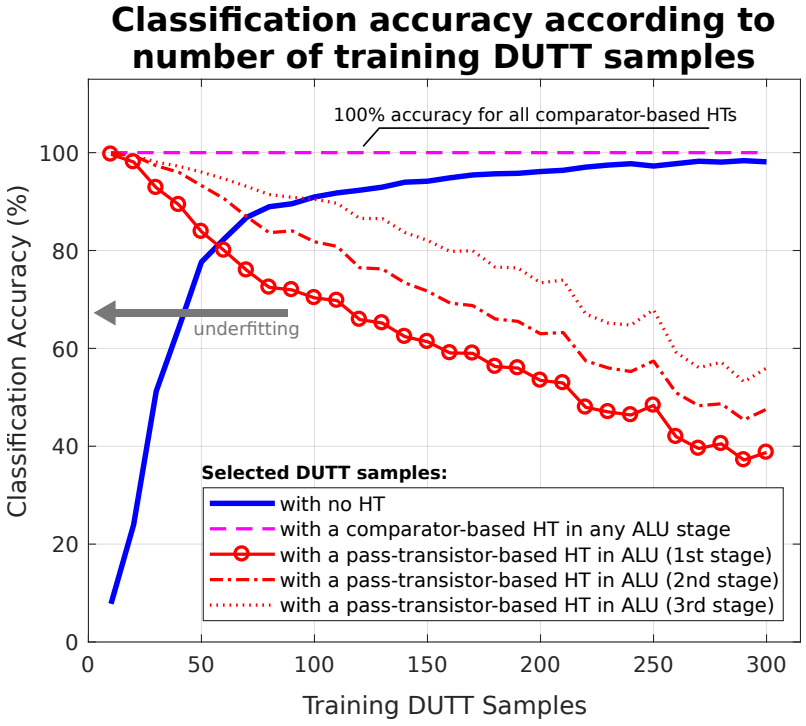

Figure 5.6: Classification accuracy according number of training samples. The OC-SVM detects all comparator-based HTs with 100% accuracy, whereas the pass-transistor detection decreases as the number of DUTT samples for OC-SVM training increases.

5.4 Conclusions

This Chapter presents a side channel HT detection technique dedicated to QDI asynchronous circuits, revealing the advantages of their current signatures for the detection of HTs. By using OC-SVM to classify a set of selected DUTT samples, our technique allows the detection of HTs with a few dozens of transistors. Similarly to our previous work in [127], our proposed technique also provides high detection accuracy without requiring any extra-circuitry. The evenly distributed current peaks, intrinsic of asynchronous circuits, make them more sensitive to side-channel deviations than synchronous circuits, thus enhancing HT detection potential. Moreover, as the technique only requires the measurement of I_{ddt}, the same testbench setup applied on the regular post-silicon testing phase can be reused for the purpose of detecting HTs. If the post-silicon testing phase considers a similar stimulus procedure described in this work, the technique can measure I_{ddt} with the same input vectors usually applied for structural testing. The the same input vectors usually applied on structural testing can also be adopted for our technique. This integration would consequently reduce test time overhead depending on how compatible the structural testing is with our technique.

6

Conclusions

Along the journey of this thesis, it is easy to notice the remarkable the effort that researchers have allocated to develop design methodologies and tools to support asynchronous design. Yet, the adoption of this kind of circuits is still limited due to its design complexity and difficult mind-set change in the industry. However, this does not imply that the industry considers a mistaken approach. As said in this work, synchronous circuits employ a simple design and its design flow has been used for decades now. The engineers are trained to understand the synchronous paradigm and use the available EDA tools. Consequently, changing this basic concept of digital VLSI design requires a drastic effort from the industrial and academic sectors, preparing new training, tools and components. Economically speaking, designers are willing to face the challenges imposed by synchronous design than changing an entire work flow, risking to lose a time-to-market window. Obviously, the research on asynchronous design was and it is still valuable as asynchronous solutions – like *Globally Asynchronous, Locally Synchronous* (GALS) design – are already a reality in the industry. Moreover, further research will provide interesting solutions for the IC technologies. Indeed, these latter will face important evolution in the upcoming years due to the challenges introduced by new technological nodes, 3D integration and very complex designs. In that way, this thesis makes a step forward for better asynchronous design support and acceptance. This chapter summarizes the contributions of this thesis, putting them into context with related works as well as giving their limitations. Final remarks and perspectives are addressed in order to close the discussions of this thesis.

6.1 Contributions on Testability of Asynchronous Circuits

Part II of this thesis covers the at-speed testing problematic of asynchronous circuits, focused mainly on BD circuits. The proposed DfT architecture provides at-speed testing support for micropipeline-based circuits, allowing controllability to fire test runs without requiring at-speed frequencies generated externally. This work makes the effort to enable compatibility of the DfT architecture with traditional synthesis and ATPG tools. It is done by considering FF-based circuits and the use of scan chains. In that way, the DfT architecture can take advantage for the automatic scan chain insertion available in synthesis tools, as well as the ATPG for stuck-at testing. For stuck-at testing, the proposed DfT architecture employs a testing clock – an approach already explored in the literature. In test mode, it bypasses the control logic of the circuit, allowing direct data transfer in the registers with an external testing clock. This indicates that a clock tree must be implemented to guarantee correct operation, albeit the clock frequencies during testing are usually slower and only requires relaxed timing constraints. This work accepts this clock tree overhead in order to have a better ATPG compatibility. To deal with the control logic of BD circuits, this work employs the LCS flow [19] in order to enable synthesis and STA analysis. The compatibility with the LCS flow was an essential part for this work, as it allows to integrate the automatic DfT insertion and the synthesis flow without major issues.

Moreover, LCS provides useful information for ATPG due to its root clock constraints. The clocks can be used as launch and capture clocks in ATPG tools, giving reference on how registers must be initialized to stimulated a given path for path-delay testing. Thus, testing vectors can be automatically generated. Finally, the proposed DfT architecture was implemented with two study-case circuits. The first one, a simple 2-bit adder, is a suitable circuit to easily evaluate the architecture. This study-case circuit was initially implemented as a straightforward example to make a fast evaluation of the proposed DfT architecture, but it can also be useful in the future in order to evaluate further modifications in the architecture. Considered as second study-case circuit, the AES cryptocore allows to see the impacts of the proposed DfT architecture in a more realistic design. At the end of this thesis, the AES was also sent to tape-out and the testchip will probably be available in January 2021. The testchip enables further use of the architecture, allowing the possibility to employ it in an industrial testing environment with a ATE.

It is important to notice that several points must be addressed in the future regarding the proposed DfT architecture. Firstly, the use of the architecture is currently software dependent with a specific tool flow. This work employs the Synonpsys' synthesis (Design Compiler/PrimeTime) and ATPG (TetraMAX) tools and script must be adapted in case other tools are desired – synthesis and ATPG tools from Cadence or MentorGraphics, for instance. Despite the fact that the proposed DfT architecture is compatible with EDA tools, the testing flow is not fully automated. It is desirable to automate the definition of all launch/capture clock pairs in the circuit. Currently, the clock pairs are specified manually on PrimeTime in order to generate the path-delay input for TetraMAX. Another limitation of the proposed DfT architecture is the absence of a mechanism signalizing the end of the test. After firing a test run, it is not possible to check if the circuit has finished its computation.this can be complicated if the circuit is operating with a wide range of supply voltages, which is a standard capability of asynchronous circuits. Indeed, this normally requires to own the characterization libraries for all the possible supply voltages. A better alternative to overcome this is issue will be to integrate an handshake mechanism in the test interface, allowing the circuit to acknowledge the end of computation.

6.2 Contributions on Side-channel Analysis for Asynchronous Circuits

Regarding the second part of this thesis, the proposed HT detection technique enhances the advantages of asynchronous QDI circuits from the security point of view, adding another benefit than robustness to *differential Power Analysis* (DPA) attacks, transient faults and reduced EM emissions: the natural ability to detect HTs. The proposed technique takes a non-intrusive approach, avoiding the need of extra circuitry such as sensors or additional power ports. Due to the delay insensitivity of QDI design, it is possible to retrieve current signatures of the circuit operating in different supply voltage levels requiring no extra setup – the circuit is capable to accommodate the delay changes by itself. Consequently, the proposed technique can evaluate the behavior of the current signatures along a range of voltage supply levels, highlighting any anomalies caused by the presence of a HT. On top of that, this approach not only detects HTs but also the infected region of the IC. Indeed, thanks to the stimuli, the HT location can be retrieved. With the proposed technique, secure-aware applications can target QDI circuits to strengthen their systems not only during field operation but also during production phase. Finally, future works will explore the automated generation of the stimulus procedure, as well as applying the proposed technique in a realistic scenario with fabricated Trojan-free and Trojan-infected samples of the study-case circuit used in this work.

First of all, it is essential for a better acceptance of the technique to replace the manual generation of the testing vectors, which is time-consuming, by an automated procedure. Another interesting aspect that could be reconsidered is the use of a golden reference. The need of a golden reference implies higher costs and an increased complexity through the design flow. Thus, exploring the concepts used in [91, 112], which employ golden chip-free techniques, can provide interest insights for modifications of the proposed HT technique. Moreover, the proposed technique was just evaluated by simulation. In order to have a more realistic scenario, it is necessary to deploy it on silicon. This scenario will consider issues regarding current measurements, jitter and PVT, impacting the technique performance. Finally, the proposed technique can also be extended to any asynchronous design, including BD circuits. However, it is necessary to be careful how the voltage scaling will be applied. Indeed, BD circuits have not the same robustness as QDI circuits and aggressive voltage scaling can compromise the circuit operation.

6.3 Perspectives: uniting strengths

In the two last sections, some perspectives of each part of this thesis were highlighted and discussed individually for further improvements. However, the works carried out in both testing and side-channel analysis for asynchronous circuits can also be correlated and employed somehow together. From the first part of this thesis, the proposed DfT architecture provides controllability during testing and allows to stimulate a given path in the circuit. From the second part, the proposed HT detection technique needs specific stimulus and requires that only a given path must be stimulated, isolating its current signature. As the HT detection technique in part III can easily be adapted for BD circuits, the proposed DfT architecture is a possible candidate to be integrated with. The scan chain structure would facilitate the initialization and verification of the circuit and the ATPG compatibility of the DfT architecture allows to automatically generate the testing vectors dedicated for HT detection.Such an approach could ease the HT detection by combining this technique with voltage scaling and time analysis. Of course, some minor modifications would be necessary in the ATPG setup. Currently, the ATPG setup focuses on verifying the worst-case paths between launch/capture clock pairs, whereas the proposed HT detection technique would require to stimulate nodes that may not be related to the critical path.

Taking a final perspective, the integration of both works could also consider the use of an embedded *Time-to-Digital Converter*s (TDC), adding intra-chip delay monitoring. In fact, with the TDC architecture proposed in [135] for example, it is possible to achieve time monitoring with picosecond precision and low area overhead due to its digital implementation. From a testing point of view, the TDC would be useful to check the delay lines used in handshake controllers for BD circuits. The behavior of delay lines could be finely characterized and further evaluated through the circuit lifetime. In addition to that, according to the measured delay, a configurable delay line could offer tuning capability making possible post fabrication corrections. Finally, the delay-monitoring feature could also highlight delay distortions due to faults (e.g. short, open or bridge) or even HT.

Bibliography

[1] P. A. Beerel, R. O. Ozdag, and M. Ferretti. *A Designer's Guide to Asynchronous VLSI*. Cambridge University Press, 2010.

[2] Illumi Huang EE Times. "Apple and Huawei use TSMC, but their 7nm SoCs are different", 2020. [online article available in https://www.eetimes.com/apple-huawei-use-tsmc-but-their-7nm-socs-are-different/#].

[3] Samsung Newsroom. "Samsung Completes Qualification of 8nm LPP Process", 2017. [online article available in https://news.samsung.com/global/samsung-completes-qualification-of-8nm-lpp-process].

[4] Nicola Jones. How to stop data centres from gobbling up the world's electricity. *Nature*, 561:163–166, 2018.

[5] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, 2nd edition, 2003.

[6] Charlie Brej. High Performance Asynchronous Circuit Design Method and Application. In *UK Async Forum*, page 5p., 2007.

[7] J. Sparsø. *Introduction to Asynchronous Circuit Design*. Independently published, 2020.

[8] J. Kessels and A. Peeters. The tangram framework: asynchronous circuits for low power. In *Proceedings of the ASP-DAC 2001. Asia and South Pacific Design Automation Conference 2001*, pages 255–260, 2001.

[9] J. Teifel and R. Manohar. An asynchronous dataflow fpga architecture. *IEEE Transactions on Computers*, 53(11):1376–1392, 2004.

[10] M. Renaudin and A. Fonkoua. Tiempo asynchronous circuits system verilog modeling language. In *IEEE 18th International Symposium on Asynchronous Circuits and Systems*, pages 105–112, 2012.

[11] A. Lines, P. Joshi, R. Liu, S. McCoy, J. Tse, Y. Weng, and M. Davies. Loihi asynchronous neuromorphic research chip. In *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 32–33, 2018.

[12] N. Imam, F. Akopyan, J. Arthur, P. Merolla, R. Manohar, and D. S. Modha. A digital neurosynaptic core using event-driven qdi circuits. In *2012 IEEE 18th International Symposium on Asynchronous Circuits and Systems*, pages 25–32, 2012.

[13] M. Waugaman and W. Koven. Sharp - a resilient asynchronous template. In *2017 23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 83–84, 2017.

[14] M. Moreira and Stefano Giaconi. Chronos Link: A QDI Interconnect for Modern SoCs. *26th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 67–68, 2020.

[15] Gopal Raghavan. Asynchronous processor that adjusts a respective operating voltage for causing a task to consume substantially all of a respective allocated time interval for the task, 2019. U.S. Patent 9,423,866.

[16] P. A. Beerel, G. D. Dimou, and A. M. Lines. Proteus: An asic flow for ghz asynchronous designs. *IEEE Design & Test of Computers*, 28(5):36–51, 2011.

[17] J. Kwong and A. P. Chandrakasan. Variation-driven device sizing for minimum energy sub-threshold circuits. In *ISLPED'06 Proceedings of the International Symposium on Low Power Electronics and Design*, pages 8–13, Oct 2006.

[18] R. D. Jorgenson, L. Sorensen, D. Leet, M. S. Hagedorn, D. R. Lamb, T. H. Friddell, and W. P. Snapp. Ultralow-power operation in subthreshold regimes applying clockless logic. *Proceedings of the IEEE*, 98(2):299–314, Feb 2010.

[19] G. Gimenez, A. Cherkaoui, G. Cogniard, and L. Fesquet. Static timing analysis of asynchronous bundled-data circuits. In *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 110–118, May 2018.

[20] M. Singh and S. M. Nowick. MOUSETRAP: High-speed transition-signaling asynchronous pipelines. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(6):684–698, June 2007.

[21] I. E. Sutherland. Micropipelines. *Commun. ACM*, 32(6):720–738, June 1989.

[22] I. Sutherland and S. Fairbanks. Gasp: a minimal fifo control. In *Proceedings Seventh International Symposium on Asynchronous Circuits and Systems. ASYNC 2001*, pages 46–53, 2001.

[23] A. Peeters, F. t. Beest, M. d. Wit, and W. Mallon. Click elements: An implementation style for data-driven compilation. In *2010 IEEE Symposium on Asynchronous Circuits and Systems*, pages 3–14, 2010.

[24] D. Hand, M. T. Moreira, H. Huang, D. Chen, F. Butzke, Z. Li, M. Gibiluka, M. Breuer, N. L. V. Calazans, and P. A. Beerel. Blade – a timing violation resilient asynchronous template. In *2015 21st IEEE International Symposium on Asynchronous Circuits and Systems*, pages 21–28, 2015.

[25] C. Mannakkara and T. Yoneda. Asynchronous pipeline controller based on early acknowledgement protocol. In *2008 8th International Conference on Application of Concurrency to System Design*, pages 118–127, 2008.

[26] J. Simatic, A. Cherkaoui, R. P. Bastos, and L. Fesquet. New asynchronous protocols for enhancing area and throughput in bundled-data pipelines. In *2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6, 2016.

[27] K. M. Fant and S. A. Brandt. Null convention logic: a complete and consistent logic for asynchronous digital circuit synthesis. In *Proceedings of International Conference on Application Specific Systems, Architectures and Processors: ASAP '96*, pages 261–273, 1996.

[28] Andrew Matthew Lines. Pipelined Asynchronous Circuits. Technical report, California Institute of Technology - CalTech, 1995. TR no. CS-TR-95-21, available at https://authors.library.caltech.edu/26834.

[29] M. T. Moreira, C. H. M. Oliveira, R. C. Porto, and N. L. V. Calazans. Ncl+: Return-to-one null convention logic. In *2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 836–839, 2013.

[30] M. T. Moreira, P. A. Beerel, M. L. L. Sartori, and N. L. V. Calazans. Ncl synthesis with conventional eda tools: Technology mapping and optimization. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(6):1981–1993, 2018.

[31] F. A. Parsan, S. C. Smith, and W. K. Al-Assadi. Design for testability of Sleep Convention Logic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(2):743–753, Feb 2016.

[32] W. Ho, K. Chong, B. Gwee, and J. S. Chang. Low power sub-threshold asynchronous quasi-delay-insensitive 32-bit arithmetic and logic unit based on autonomous signal-validity half-buffer. *IET Circuits, Devices Systems*, 9(4):309–318, 2015.

[33] R. M. Davies and J. V. Woods. Timing verification for asynchronous design. In *Proceedings of European Design Automation Conference (EURO-DAC)*, pages 78–83, 1996.

[34] Alain J. Martin. The Limitations to Delay-insensitivity in Asynchronous Circuits. In *Sixth MIT Conference on Advanced Research in VLSI*, AUSCRYPT '90, pages 263–278, 1990.

[35] A. J. Martin and M. Nyström. Asynchronous Techniques for System-on-Chip Design. *Proceedings of the IEEE*, 94(6):1089–1120, June 2006.

[36] C. Myers. *Asynchronous Circuit Design*. John Wiley & Sons, Inc., 2001.

[37] M. T. Moreira, C. H. M. Oliveira, R. C. Porto, and N. L. V. Calazans. NCL+: Return-to-one Null Convention Logic. In *56th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 836–839, August 2013.

[38] Rajit Manohar and Alain J. Martin. Quasi-delay-insensitive Circuits are Turing-Complete. Technical report, California Institute of Technology - CalTech, 1995. TR no. CS-TR-95-11, available at http://vlsi.cornell.edu/∼rajit/ps/qdi.pdf.

[39] K. van Berkel, F. Huberts, and A. Peeters. Stretching Quasi Delay Insensitivity by means of Extended Isochronic Forks. In *IEEE Conference on Asynchronous Design Methodologies (ASYNC)*, pages 99–106, May 1995.

[40] S. C. Smith. Speedup of self-timed digital systems using early completion. In *Proceedings IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI Systems Design. ISVLSI 2002*, pages 107–113, 2002.

[41] R. A. Guazzelli, W. L. Neto, M. T. Moreira, and N. L. V. Calazans. Sleep convention logic isochronic fork: An analysis. In *2017 30th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 103–109, 2017.

[42] Arthur D. Friedman Miron Abramovici, Melvin A. Breuer. *Digital Systems Testing and Testable Design*. IEEE Press, 1990.

[43] M. Bushnell and Vishwani Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer Publishing Company, Incorporated, 2013.

[44] S. Pagey, S. D. Sherlekar, and G. Venkatesh. Issues in fault modelling and testing of micropipelines. In *Proceedings First Asian Test Symposium (ATS '92)*, pages 107–111, Nov 1992.

[45] A. Khoche and E. Brunvand. Testing micropipelines. In *Proceedings of 1994 IEEE Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 239–246, Nov 1994.

[46] A. Khoche and E. Brunvand. A partial scan methodology for testing self-timed circuits. In *Proceedings 13th IEEE VLSI Test Symposium*, pages 283–289, April 1995.

[47] O. A. Petlin and S. B. Furber. Scan testing of asynchronous sequential circuits. In *Proceedings. Fifth Great Lakes Symposium on VLSI*, pages 224–229, March 1995.

[48] V. Schober and T. Kiel. An asynchronous scan path concept for micropipelines using the bundled data convention. In *Proceedings International Test Conference 1996. Test and Design Validity*, pages 225–231, Oct 1996.

[49] O. A. Petlin and S. B. Furber. Built-in self-testing of micropipelines. In *Proceedings Third International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 22–29, April 1997.

[50] M. Roncken and E. Bruls. Test quality of asynchronous circuits: a defect-oriented evaluation. In *Proceedings International Test Conference 1996. Test and Design Validity*, pages 205–214, Oct 1996.

[51] M. Roncken, E. Aarts, and W. Verhaegh. Optimal scan for pipelined testing: an asynchronous foundation. In *Proceedings International Test Conference 1996. Test and Design Validity*, pages 215–224, Oct 1996.

[52] M. Roncken. Defect-oriented testability for asynchronous ICs. *Proceedings of the IEEE*, 87(2):363–375, Feb 1999.

[53] M. Roncken, S. M. Gilla, H. Park, N. Jamadagni, C. Cowan, and I. Sutherland. Naturalized communication and testing. In *2015 21st IEEE International Symposium on Asynchronous Circuits and Systems*, pages 77–84, May 2015.

[54] M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Saldanha, and A. Taubin. Partial-scan delay fault testing of asynchronous circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(11):1184–1199, Nov 1998.

[55] Yong-Seok Kang, Kyung-Hoi Huh, and Sungho Kang. New scan design of asynchronous sequential circuits. In *AP-ASIC'99. First IEEE Asia Pacific Conference on ASICs (Cat. No.99EX360)*, pages 355–358, Aug 1999.

[56] K. van Berkel, A. Peeters, and F. te Beest. Adding synchronous and LSSD modes to asynchronous circuits. In *Proceedings Eighth International Symposium on Asynchronous Circuits and Systems*, pages 161–170, April 2002.

[57] F. T. Beest, A. Peeters, M. Verra, K. van Berkel, and H. Kerkhoff. Automatic scan insertion and test generation for asynchronous circuits. In *Proceedings. International Test Conference*, pages 804–813, Oct 2002.

[58] Frank Beest, Ad Peeters, Kees Berkel, and Hans Kerkhoff. Synchronous full-scan for asynchronous handshake circuits. *J. Electronic Testing*, 19:397–406, 08 2003.

[59] F. Beest and A. Peeters. A multiplexer based test method for self-timed circuits. In *11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 166–175, March 2005.

[60] M. L. King and K. K. Saluja. Testing micropipelined asynchronous circuits. In *2004 International Conference on Test*, pages 329–338, Oct 2004.

[61] Feng Shi, Yiorgos Makris, S. M. Nowick, and M. Singh. Test generation for ultra-high-speed asynchronous pipelines. In *IEEE International Conference on Test, 2005.*, pages 10 pp.–1018, Nov 2005.

[62] D. Shang, A. Yakovlev, F. Burns, F. Xia, and A. Bystrov. Low-cost online testing of asynchronous handshakes. In *Eleventh IEEE European Test Symposium (ETS'06)*, pages 225–232, May 2006.

[63] V. Varshavsky, Michael Kishinevsky, Vyacheslav Marakhovsky, V. Peschansky, L. Rosenblum, A. Taubin, and B. Tzirlin. *Self-timed control of concurrent processes*. 1990.

[64] G. Gill, A. Agiwal, M. Singh, Feng Shi, and Y. Makris. Low-overhead testing of delay faults in high-speed asynchronous pipelines. In *12th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'06)*, pages 11 pp.–56, March 2006.

[65] Chi-Hsuan "Cheng and James Chien-Mo" Li. "an asynchronous design for testability and implementation in thin-film transistor technology". *"Journal of Electronic Testing"*, "27"("2"):"193–201", "Apr" "2011".

[66] N. Nemati, M. C. Reed, K. Fant, and P. Beckett. Asynchronous interleaved scan architecture for on-line built-in self-test of Null Convention Logic. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 746–749, May 2016.

[67] K. Huang, T. Shen, and C. Li. Test methodology for dual-rail asynchronous circuits. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2017.

[68] N. Nemati, P. Beckett, M. C. Reed, and K. Fant. Clock-less DfT-less test strategy for Null Convention Logic. *IEEE Transactions on Emerging Topics in Computing*, 6(4):460–473, Oct 2018.

[69] F. A. Kuentzer and A. M. Amory. Fault classification of the error detection logic in the Blade resilient template. In *2016 22nd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 37–42, May 2016.

[70] F. A. Kuentzer, L. R. Juracy, and A. M. Amory. On the reuse of timing resilient architecture for testing path delay faults in critical paths. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 379–384, March 2018.

[71] G. Gimenez, J. Simatic, and L. Fesquet. From signal transition graphs to timing closure: Application to bundled-data circuits. In *25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 86–95, May 2019.

[72] M. Tehranipoor and F. Koushanfar. A Survey of Hardware Trojan Taxonomy and Detection. volume 27, pages 10–25, Jan. 2010.

[73] R. S. Chakraborty, S. Narasimhan, and S. Bhunia. Hardware trojan: Threats and emerging solutions. In *IEEE International High Level Design Validation and Test Workshop*, pages 166–171, 2009.

[74] N. Jacob, D. Merli, J. Heyszl, and G. Sigl. Hardware trojans: current challenges and approaches. *IET Computers & Digital Techniques*, 8(6):264–273, 2014.

[75] H. Li, Q. Liu, J. Zhang, and Y. Lyu. A survey of hardware trojan detection, diagnosis and prevention. In *14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)*, pages 173–180, 2015.

[76] H. Salmani, M. Tehranipoor, and R. Karri. On design vulnerability analysis and trust benchmarks development. In *IEEE 31st International Conference on Computer Design (ICCD)*, pages 471–474, 2013.

[77] R. Kumar, P. Jovanovic, W. Burleson, and I. Polian. Parametric trojans for fault-injection attacks on cryptographic hardware. In *Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 18–28, 2014.

[78] B. Cakır and S. Malik. Hardware trojan detection for gate-level ics using signal correlation based clustering. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 471–476, 2015.

[79] M. Oya, Y. Shi, M. Yanagisawa, and N. Togawa. A score-based classification method for identifying hardware-trojans at gate-level netlists. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 465–470, 2015.

[80] M. Banga and M. S. Hsiao. Trusted rtl: Trojan detection methodology in pre-silicon designs. In *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 56–59, 2010.

[81] X. Chen, Q. Liu, S. Yao, J. Wang, Q. Xu, Y. Wang, Y. Liu, and H. Yang. Hardware trojan detection in third-party digital intellectual property cores by multilevel feature analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(7):1370–1383, 2018.

[82] T. Reece and W. H. Robinson. Detection of hardware trojans in third-party intellectual property using untrusted modules. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(3):357–366, 2016.

[83] S. Jha and S. K. Jha. Randomization based probabilistic approach to detect trojan circuits. In *2008 11th IEEE High Assurance Systems Engineering Symposium*, pages 117–124, 2008.

[84] R. S. Chakraborty, S. Paul, and S. Bhunia. On-demand transparency for improving hardware trojan detectability. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 48–50, 2008.

[85] M. Flottes G. Dinatale, S. Dupuis and B. Rouzeyre. Identification of hardware trojans triggering signals. In *Workshop on Trustworthy Manufacturing and Utilization of Secure Devices*, May 2013.

[86] N. Lesperance, S. Kulkarni, and Kwang-Ting Cheng. Hardware trojan detection using exhaustive testing of k-bit subspaces. In *The 20th Asia and South Pacific Design Automation Conference*, pages 755–760, 2015.

[87] H. Salmani, M. Tehranipoor, and J. Plusquellic. A novel technique for improving hardware trojan detection and reducing trojan activation time. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(1):112–125, 2012.

[88] S. Paul C. Papachristou R. S. Chakraborty, F. Wolff and S. Bhunia. Mero: A statistical approach for hardware trojan detection. In *Cryptographic Hardware and Embedded Systems (CHES)*, pages 396–410. Springer Berlin Heidelberg, 2009.

[89] Mohammad Tehranipoor and Cliff Wang. *Introduction to Hardware Security and Trust*. Springer Publishing Company, Incorporated, 2011.

[90] C. Chiang and J. Kawa. Design for manufacturability and yield for nano-scale cmos. In *Series on Integrated Circuits and Systems*, 2007.

[91] Y. Liu, K. Huang, and Y. Makris. Hardware trojan detection through golden chip-free statistical side-channel fingerprinting. In *51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2014.

[92] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan Detection using IC Fingerprinting. In *Proc. IEEE Symposium on Security and Privacy (SP'07)*, pages 296–310, May 2007.

[93] X. Wang, H. Salmani, M. Tehranipoor, and J. Plusquellic. Hardware trojan detection and isolation using current integration and localized current analysis. In *2008 IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, pages 87–95, Oct 2008.

[94] B. Shanyour and S. Tragoudas. Detection of low power trojans in standard cell designs using built-in current sensors. In *Proc. IEEE International Test Conference (ITC)*, pages 1–10, Oct 2018.

[95] R. Rad, J. Plusquellic, and M. Tehranipoor. Sensitivity analysis to hardware Trojans using power supply transient signals. In *Proc. IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*, pages 3–7, June 2008.

[96] R. M. Rad, X. Wang, M. Tehranipoor, and J. Plusquellic. Power supply signal calibration techniques for improving detection resolution to hardware trojans. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 632–639, 2008.

[97] Y. Liu, Y. Jin, A. Nosratinia, and Y. Makris. Silicon Demonstration of Hardware Trojan Design and Detection in Wireless Cryptographic ICs. volume 25, pages 1506–1519, April 2017.

[98] J. Aarestad, D. Acharyya, R. Rad, and J. Plusquellic. Detecting Trojans Through Leakage Current Analysis Using Multiple Supply Pad I_{DDQ}s. volume 5, pages 893–904, Dec 2010.

[99] I. Wilcox, F. Saqib, and J. Plusquellic. GDS-II Trojan detection using multiple supply pad V_{DD} and G_{ND} *IDDQs* in ASIC functional units. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 144–150, 2015.

[100] J. Li and J. Lach. Negative-skewed shadow registers for at-speed delay variation characterization. In *25th International Conference on Computer Design*, pages 354–359, 2007.

[101] J. Li and J. Lach. At-speed delay characterization for ic authentication and trojan horse detection. In *IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 8–14, 2008.

[102] B. Cha and S. K. Gupta. Trojan Detection via Delay Measurements: a new approach to select paths and vectors to maximize effectiveness and minimize cost. In

Proc. Design, Automation Test in Europe Conference (DATE), pages 1265–1270, March 2013.

[103] D. Ismari, J. Plusquellic, C. Lamech, S. Bhunia, and F. Saqib. On detecting delay anomalies introduced by hardware trojans. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7, 2016.

[104] C. Lamech, J. Aarestad, J. Plusquellic, R. Rad, and K. Agarwal. REBEL and TDC: Two embedded test structures for on-chip measurements of within-die path delay variations. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 170–177, 2011.

[105] C. Lamech and J. Plusquellic. Trojan detection based on delay variations measured using a high-precision, low-overhead embedded test structure. In *IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 75–82, 2012.

[106] A. Davoodi, M. Li, and M. Tehranipoor. A sensor-assisted self-authentication framework for hardware trojan detection. *IEEE Design & Test*, 30(5):74–82, 2013.

[107] Yier J. and Y. Makris. Hardware trojan detection using path delay fingerprint. In *IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 51–57, 2008.

[108] G. Zarrinchian and M. S. Zamani. Latch-based structure: A high resolution and self-reference technique for hardware trojan detection. *IEEE Transactions on Computers*, 66(1):100–113, 2017.

[109] J. Balasch, B. Gierlichs, and I. Verbauwhede. Electromagnetic circuit fingerprints for hardware trojan detection. In *IEEE International Symposium on Electromagnetic Compatibility (EMC)*, pages 246–251, 2015.

[110] O. Söll, T. Korak, M. Muehlberghuber, and M. Hutter. Em-based detection of hardware trojans on fpgas. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 84–87, 2014.

[111] D. Jap, Wei He, and S. Bhasin. Supervised and Unsupervised Machine Learning for Side-Channel Based Trojan Detection. In *Proc. International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, pages 17–24, July 2016.

[112] J. He, Y. Zhao, X. Guo, and Y. Jin. Hardware trojan detection through chip-free electromagnetic side-channel statistical analysis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2939–2948, 2017.

[113] C. Bao, D. Forte, and A. Srivastava. Temperature Tracking: Toward Robust Run-Time Detection of Hardware Trojans. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1577–1585, Oct 2015.

[114] K. Hu, A. N. Nowroz, S. Reda, and F. Koushanfar. High-sensitivity hardware trojan detection using multimodal characterization. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1271–1276, 2013.

[115] L. A. Guimarães, R. P. Bastos, and L. Fesquet. Detection of Layout-Level Trojans by Monitoring Substrate with Preexisting Built-in Sensors. In *Proc. IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 290–295, July 2017.

[116] Narasimhan, Seetharam and Du, Dongdong and Chakraborty, Rajat and Paul, Somnath and Wolff, Francis and Papachristou, Chris and Roy, Kaushik and Bhunia, Swarup. Hardware Trojan Detection by Multiple-Parameter Side-Channel Analysis. volume 62, pages 2183–2195, Nov 2013.

[117] B. Hou, C. He, L. Wang, Y. En, and S. Xie. Hardware trojan detection via current measurement: A method immune to process variation effects. In *2014 10th International Conference on Reliability, Maintainability and Safety (ICRMS)*, pages 1039–1042, 2014.

[118] X. Ngo, I. Exurville, S. Bhasin, J. Danger, S. Guilley, Z. Najm, J. Rigaud, and B. Robisson. Hardware trojan detection by delay and electromagnetic measurements. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 782–787, 2015.

[119] F. Koushanfar and A. Mirhoseini. A unified framework for multimodal submodular integrated circuits trojan detection. *IEEE Transactions on Information Forensics and Security*, 6(1):162–174, 2011.

[120] K. Xiao, X. Zhang, and M. Tehranipoor. A clock sweeping technique for detecting hardware trojans impacting circuits delay. *IEEE Design Test*, 30(2):26–34, April 2013.

[121] X. Cui, K. Ma, L. Shi, and K. Wu. High-level synthesis for run-time hardware trojan detection and recovery. In *51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2014.

[122] E. Dubrova, M. Näslund, G. Carlsson, and B. Smeets. Keyed logic bist for trojan detection in soc. In *International Symposium on System-on-Chip (SoC)*, pages 1–4, 2014.

[123] Y. Zheng, S. Yang, and S. Bhunia. SeMIA: Self-Similarity-Based IC Integrity Analysis. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(1):37–48, Jan 2016.

[124] T. Hoque, M. Mustapa, F. Amsaad, and M. Niamat. Assessment of NAND based ring oscillator for hardware Trojan detection. In *Proc. IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4, Aug 2015.

[125] F. K. Lodhi and S. R. Hasan and O. Hasan and F. Awwad. Hardware Trojan Detection in Soft Error Tolerant Macro Synchronous Micro Asynchronous (MSMA) Pipeline. In *Proc. 57th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 659–662, Aug 2014.

[126] F. K. Lodhi and S. R. Hasan and O. Hasan and F. Awwad. Formal Analysis of Macro Synchronous Micro Asychronous Pipeline for Hardware Trojan Detection. In *Proc. Nordic Circuits and Systems Conference (NORCAS): NORCHIP International Symposium on System-on-Chip (SoC)*, pages 1–4, Oct 2015.

[127] L. A. Guimarães, T. F. de Paiva Leite, R. P. Bastos, and L. Fesquet. Non-intrusive Testing Technique for Detection of Trojans in Asynchronous Circuits. In *Proc. Design, Automation Test in Europe Conference (DATE)*, pages 1516–1519, March 2018.

[128] G. F. Bouesse, M. Renaudin, S. Dumont, and F. Germain. DPA on Quasi-Delay Insensitive Asynchronous Circuits: Formalization and Improvement. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 424–429, March 2005.

[129] J. Lim, W. G. Ho, K. S. Chong, and B. H. Gwee. DPA-Resistant QDI Dual-rail AES S-Box based on Power-balanced Weak-conditioned Half-buffer. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2017.

[130] G. F. Bouesse, M. Renaudin, A. Witon, and F. Germain. A Clock-less Low-voltage AES Crypto-processor. In *Proc. European Solid-State Circuits Conference (ESSCIRC)*, pages 403–406, Sept 2005.

[131] Y. Monnet, M. Renaudin, and R. Leveugle. Hardening techniques against transient faults for asynchronous circuits. In *Proc. 11th IEEE International On-Line Testing Symposium*, pages 129–134, July 2005.

[132] Ivan E. Sutherland and Jo Ebergen. *Computer Without Clocks*. Scientific American, 2002.

[133] L. A. Guimarães, R. P. Bastos, T. F. de Paiva Leite, and L. Fesquet. Simple Tri-state Logic Trojans Able to Upset Properties of Ring Oscillators. In *Proc. International*

Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS), pages 1–6, April 2016.

[134] Abdulhamit Subasi and M. Ismail Gursoy. EEG Signal Classification using PCA, ICA, LDA and Support Vector Machines. volume 37, pages 8659–8666, 2010.

[135] A. El Hadbi. *Time-to-Digital Conversion based on a Self-Timed Ring Oscillator*. PhD thesis, École Doctorale Électronique, Électrotechnique, Automatique & Traitement du Signal (EEATS), 2019.

[136] M. T. Moreira, M. E. Arendt, R. A. Guazzelli, and N. L. V. Calazans. A new cmos topology for low-voltage null convention logic gates design. In *20th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 93–100, 2014.

List of Figures

List of Tables

7

List of Publications and Presentations

7.1 Publications

1. GUAZZELLI, R. A.; and FESQUET, L.; "*At-speed DfT architecture of Bundled-data Circuits*". IEEE International Test Conference (ITC), 2020;

2. GUAZZELLI, R. A.; TRINDADE, M. G.; GUIMARÃES, L. A.; LEITE, T. F. P.; FESQUET, L.; POSSAMAI BASTOS, R.; "*Trojan Detection Test for Clockless Circuits*". Springer Journal of Electronic Testing: Theory and Applications, JETTA, 2020.

3. GUAZZELLI, R. A.; TRINDADE, M. G.; FESQUET, L.; POSSAMAI BASTOS, R. "*Learning-Based Reliability Assessment Method for Detection of Permanent Faults in Clockless Circuits*". Elsevier Microelectronics Reliability Journal, 2019.

7.2 Presentations

1. Paper presentation "*Learning-Based Reliability Assessment Method for Detection of Permanent Faults in Clockless Circuits*" at 30st European Symposium on Reliability of Electron Devices, Failure Physics and Analysis (ESREF), Oct. 2019;

2. National presentation "*Exploring a Non-conventional Testing Technique for Asynchronous Circuits*" at 21ème édition des Journées Nationales du Réseau Doctoral en Micro-nanoélectronique (JNRDM), June 2019;

8 Acronym List

ACSR *Architecture Configuration Shift-Register* 136

AES *Advanced Encryption Standard* 50, 62, 132, 136

AHSL *Asynchronous High-Level Synthesis* 132

AI *Artificial Intelligence* 2, 4

ALU *Arithmetic Logic Unit* 88

ASIC *Application-Specific Integrated Circuit* 2

ASL *Asynchronous Scan Latch* 36

ASVHB *Autonomous Signal-Validity Half-Buffer* 23

ATE *Automated Testing Equipment* 32, 96

ATPG *Automatic Testing Pattern Generation* 6, 24, 32–35, 37, 41, 46, 47, 49, 50, 54–56, 62, 95, 96, 98

BBICS *Bulk Built-in Current Sensor* 76

BD *Bundled-Data* 5, 6, 10, 11, 15, 17, 18, 21, 24, 35, 41, 50, 52, 55, 62, 95, 97, 98, 115, 118

BILBO *Built-in Logic Block Observer* 36

BIST *Built-in Self-Test* 32, 36, 38

CAD *Computer Aided Design* 1, 3

CMOS *Complementary Metal Oxide Semiconductor* 1, 50

CMP *Chemical Mechanical Polishing* 71

CUDA *Compute Unified Device Architecture* 2

DAC-scan *Dual-Rail Asynchronous Circuit Scan* 38

DC *David Cell* 37

D-FF *Type-D Flip-Flop* 33

DfT *Design-for-Testability* 6, 24, 32, 33, 36–38, 41, 49, 50, 52–54, 56, 57, 62, 95, 96, 98, 132

DI *Delay Insensitive* 18, 19, 22

DIMS *Delay-Insensitive Minterm Synthesis* 22

DoS *Denial of Service* 69

DPA *differential Power Analysis* 97

DUT *Design Under Test* 57

DUTT *Device Under Trojan Test* v, 71–76, 79–81, 84–90, 92, 116

DVS *Dynamic Voltage Scaling* 4

EDA *Electronic Design Automation* 4, 6, 17, 32, 41, 46, 50, 95, 96

EDL *Error Detection Logic* 38

EM *Electromagnetic* 4, 73, 75, 80

EMI *Electromagnetic Interference* 18

FD-SOI *Fully Depleted Silicon-On-Insulator* 88

FF *Flip-Flop* 2, 3, 24, 33, 34, 45, 50, 52, 95, 114

FPGA *Field-Programmable Gate Array* 2, 4, 75, 87

GALS *Globally Asynchronous, Locally Synchronous* 95

GPU *Graphic Processor Unit* 1, 2

HBM2 *High-Bandwith Memory 2* 2

HC *High-Capacity* 37

HPC *High-Performance Computing* 2

HSB *Handshake Breaker* 42–44, 57, 62, 136

HT *Hardware Trojan* 6, 66–76, 80–82, 84–90, 92, 97, 98, 115, 116

IC *Integrated Circuit* 1, 2, 66, 67, 71–77, 80, 85, 87, 95, 97, 115

IDDQ *Quiescent Supply Current* 36, 38, 74

IDDT *Transient Supply Current* 74, 80

INSR *Input Shift-Register* 118, 132, 133

IoT *Internet of Things* 2, 5

IP *Intellectual Property* 2, 4, 72, 80

KL *Karhunen-Loève* 74

LCS *Local Clock Set* 41, 46, 47, 50, 51, 53, 54, 62, 95, 96

LSSD *Level-Sensitive Scan-Design* 35–37

MC *Monte Carlo* 88

MVEE *Minimum Volume Enclosing Ellipsoid* 73, 80

NCL *NULL Convention Logic* 22, 23, 38, 130

NCL+ *NULL Convention Logic Plus* 23

NoC *Network-on-Chip* 4

OC-SVM *One-Class Support Vector Machine* v, 79, 81, 84, 86, 87, 89, 92, 116

OUTSR *Output Shift-Register* 132

PCA *Principal Component Analysis* 89

PCB *Printed Circuit Board* 67

PCFB *Pre-Charged Full-Buffer* 22

PCHB *Pre-Charged Half-Buffer* 21–23

PUF *Physical Unclonable Function* 132

PV *Process Variations* 4, 5, 72–74, 76, 85, 89

PVT *Process, Voltage and Temperature* 3, 18, 97, 114

QDI *Quasi-Delay Insensitive* 5, 6, 10, 11, 18–24, 35–38, 80, 82–85, 88, 92, 97, 114, 116, 118

RBF *Radial Basis Function* 89

RM *Reference Methodology* 50, 52, 53

RTL *Register-Transfer Level* 68, 75

RTO *Return-to-One* 19, 23, 118

RTZ *Return-to-Zero* 19, 23, 118

SCL *Sleep Convention Logic* 23, 38

SDC *Synopsys Design Constraints* 54

SDDS-NCL *Spatially Distributed Dual-Spacer NULL Convention Logic* 23

SDF *Standard Delay Format* 68

SEC *Sequential Equivalence Checking* 72

SEE *Single-Event Effect* 18

SEM *Scanning Electron Microscope* 71

SFF *Scan Flip-Flop* 33, 50

SoC *System-on-Chip* 4

SPF *STIL Protocol File* 53, 56

STA *Static Timing Analysis* 41, 46, 50, 51, 54, 55, 95

STCL *Scan Test Control Logic* 36

STG *Signal Transition Graph* 45, 115

STIL *Standard Test Interface Language* 57

STR *Self-Timed Ring* 132

SVM *Support Vector Machine* 85, 87

TCL *Tool Command Language* 50, 52

TDC *Time-to-Digital Converter* 98

TRNG *True Random Number Generator* 68

ULP *Ultra-Low-Power* 2, 5

VLSI *Very Large Scale Integration* 31, 95

WCHB *Weak-Conditioned Half-Buffer* 21–23, 37, 38, 45, 115

A

Asynchronous Components

A.1 C-Element Alternative Topologies

Figure A.1 illustrates the different 2-input C-element topologies available in the literature. Note that we are considering symmetrical implementations only. First, the Martin's weak feedback (a) C-element is the most classic implementation. Its implementation employs of a pull-down (*set*) and a pull-up (*reset*) network that allows the inputs *A* and *B* to drive the output Z_n directly. When all inputs are at '0', *set* is activated. When all inputs are at '1', *reset* is activated. In case the inputs have different logic values, the *latch* is the only active logic block and it maintains the logic value at the output.

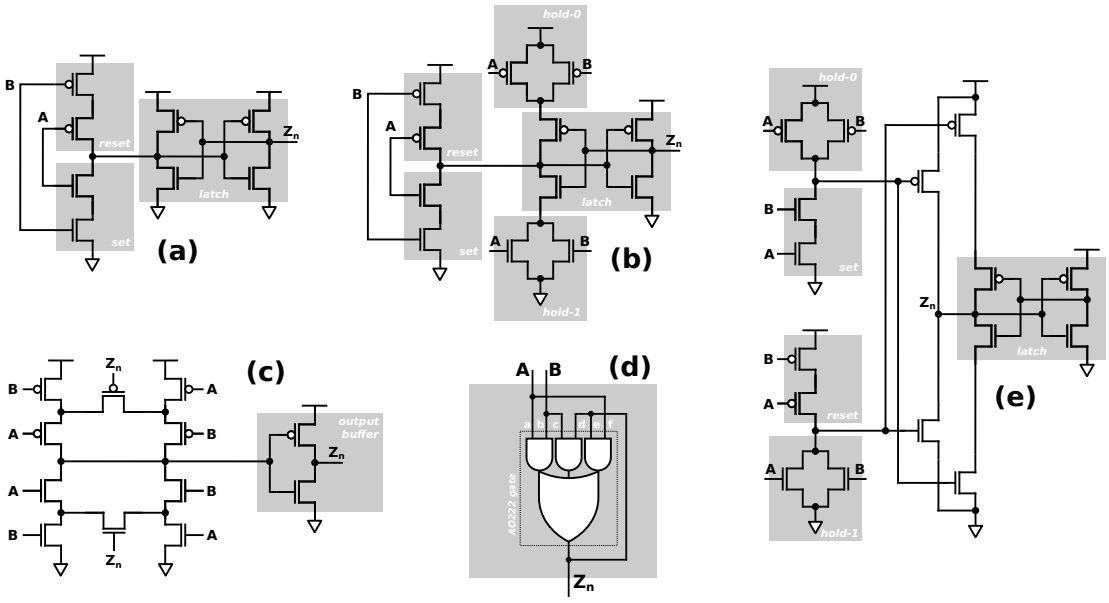

Figure A.1: The Muller C-element alternative topologies: (a) Martin's (weak feedback); (b) Sutherland's (static); (c) Van Berkel's, (d) majority-based and (e) Moreira's.

Sutherland's C-element topology (b) was proposed in 1989 and employed in his micropipeline design. The topology uses a similar structure as Martin's topology, with identical *reset*, *set* and *latch* blocks. However, Sutherland's C-element adds two logic blocks called *hold-0* and *hold-1*, connecting the sources of the feedback inverter to the supply voltage and ground. These additional logic blocks are responsible to establish static connectivity to the voltage references and guarantee better output integrity when *reset* and *set* blocks are not driving the output. As the name suggests, *hold-0* helps to "hold" a low logic level at the output, while *hold-1* "holds" a high logic level.

Figure A.1 (c) illustrates the schematic of Van Berkel's topology. Different from previous topologies, this C-element des not implement a traditional latch at its output. Instead, the feedback logic is integrated to the *set* and *reset* logic through the P_{fb} and N_{fb} transistors. So it is possible to say that *set* integrates the *hold-0* block into its logic and *reset* integrates *hold-1*. In that way, the inputs are driving the output all the time, independently of the output value.

In case no dedicated C-element implementation is available, designer can consider an implementation with standard logic gates, such as the one illustrated in Figure A.1 (d). This C-element is a majority-based implementation and can be implemented with AND / OR gates available in standard cell libraries and avoids the necessity of designing a C-element gate from scratch. In order to implement the memory logic, C-element has a feedback connection between the output and two internal inputs. However, it is important to highlight that a timing constraint is present between the output and the feedback connections. In this case, it is necessary to guarantee that the feedback connection are updated before the arrival of new input transitions, otherwise the circuit can malfunction. Consequently, it is advised to specify constraints in synthesis tools or implement the feedback connection manually.

Finally, in 2014, Moreira et al. [136] presented a different approach to implement NCL gates for low-voltage applications. including C-elements as the one represented in Figure A.1 (e). The topology employs the basic *set*, *reset*, *hold-0* and *hold-1* blocks, an output inverter and a *latch* controlled by two additional transistors *PC* and *NC*. According to the authors, this implementation employs a higher number of transistor but better performance and ebergy trade-offs.

B

TIMA Asynchronous Testchip

B.1 Overview architecture

Figure B.1 presents the layout of the testchip sent to the foundry. The testchip contains two asynchronous 128-bit AES cryptocores and a *Self-Timed Ring* (STR) *Physical Unclonable Function* (PUF). Regarding the two AES, one of them employs the DfT architecture proposed in chapter 3 and the other was implemented through an *Asynchronous High-Level Synthesis* (AHSL) flow. Note that the STR PUF not the AHSL AES are not related to this thesis. In fact, the STR PUF are logically isolated from the AES cryptocores.

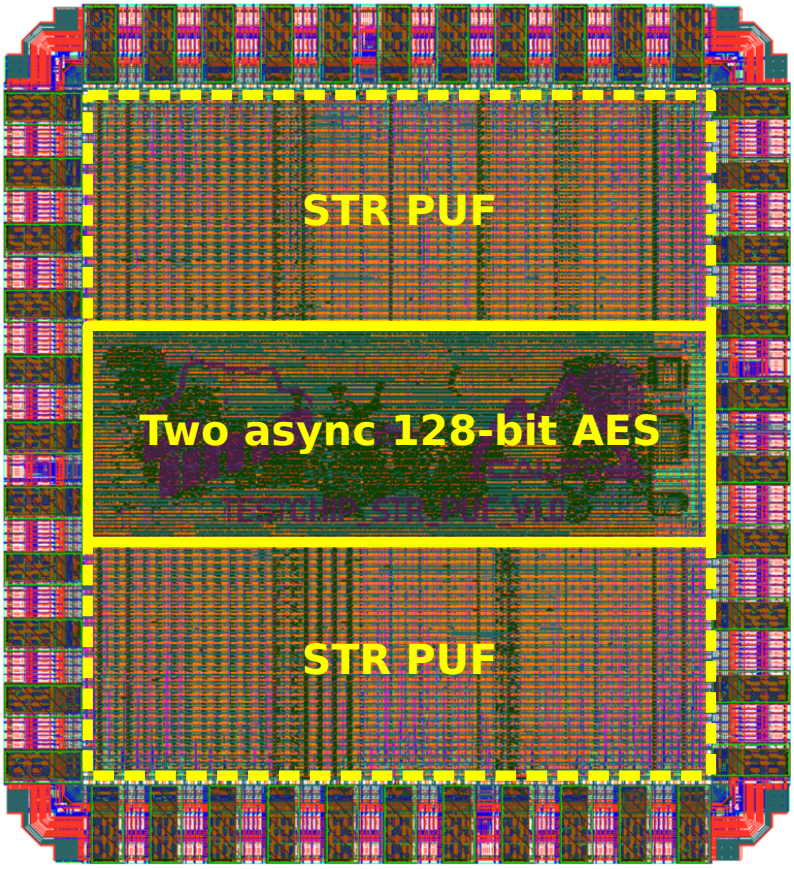

Figure B.1: Layout of TIMA's testchip.

Figure B.2 illustrates the block-level architecture integrating the two asynchronous AES cryptocores. The integration logic consists of an input, an output and a configuration shift registers, handshake loop blocks and three multiplexers. The INSR contains 256 bits and allows the external charge through *serial_AES_i*. Table B.1 indicates the connections between INSR and AES inputs (plaintext and keyblock). The *Output Shift-Register* (OUTSR) contains 128 bits and it is responsible to store and propagate the AES result to the external environment. It employs an extra control signal called *output_mode* that allows to set OUTSR in normal or shift mode. In normal mode, OUTSR can store the AES output. In shift mode, OUTSR is disconnected to the AES output and allows to shift its value through *serial_AES_o*.

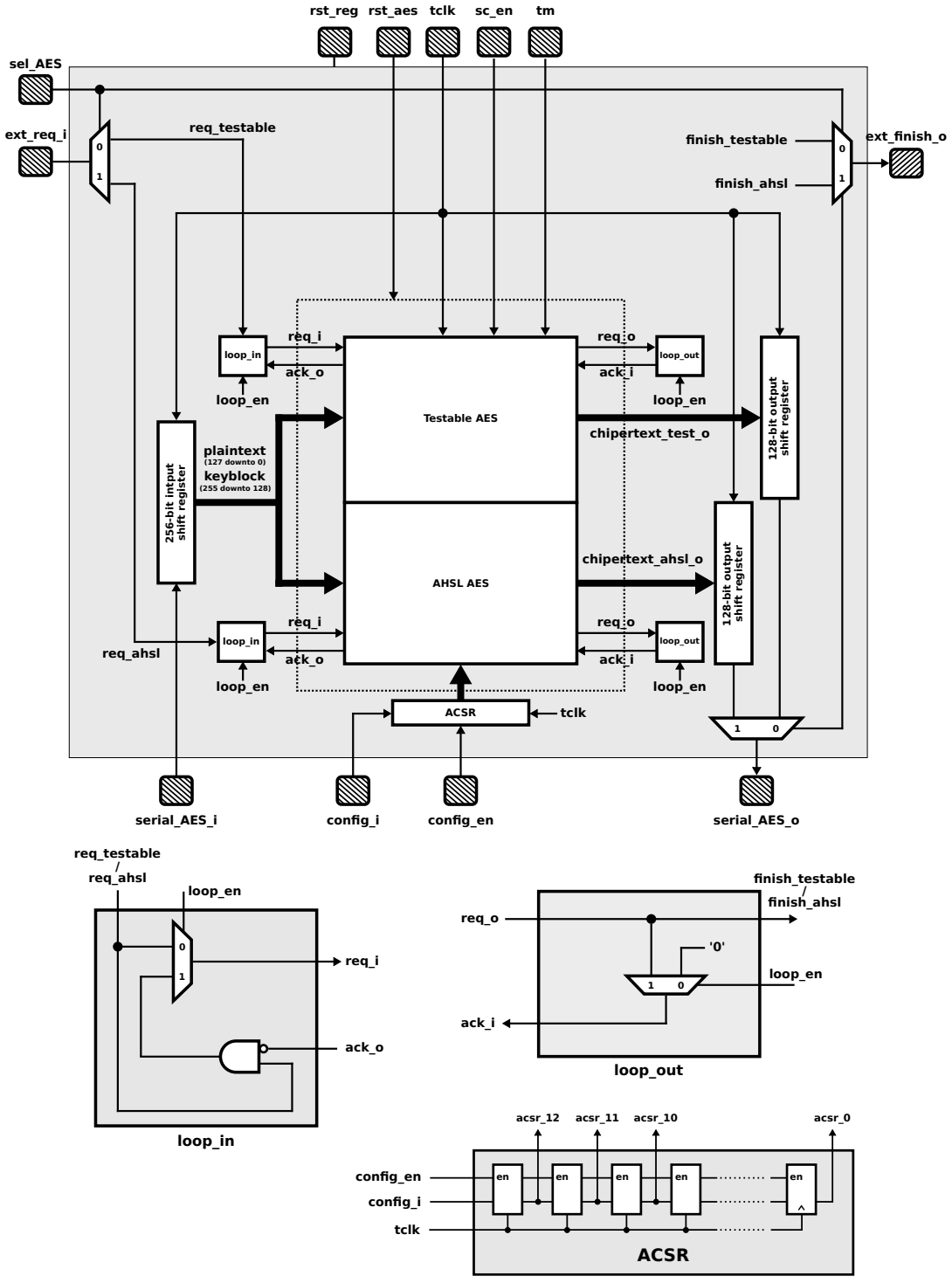

Figure B.2: Integration architecture of both asynchronous AES cryptocores.

Table B.1: Relation between INSR signals and AES inputs.

INSR Signal	AES Signal
127 downto 0	plaintext
255 downto 0	keyblock

B.2 Testchip PINOUT (AES part only)

Table B.2: Input and output pin information of the testchip. This table only presents the pins related to the AES blocks.

Signal	Direction	Pull Config.	Hyst.	Drive	Description
tclk	in	no	yes	2	Testing clock. When tm = '0', this clock signal is used to charge (discharge) the input (output) shift registers. Otherwise, the signal controls the scan and HSB logic of the testable AES.
rst_reg	in	no	yes	2	High-active reset signal for INSR/ OUTSR/ACSR. This signal does not reset the AES blocks.
rst_aes	in	no	yes	2	High-active reset signal for AES blocks.
serial_AES_i	in	no	yes	2	Serial input of both AES blocks.
serial_AES_o	out	no	yes	8	Serial output of both AES blocks.
config_i	in	no	no	2	ACSR input.
config_en	in	no	no	2	ACSR enable signal (active high).

ext_req_i	in	down	yes	2	External input request. If the AHLS AES is enabled, this signal is mainly used as input request to initiate the AES operation. For the testable AES, this signal is used for the at-speed testing and normal operation. This signal can be used as a normal input request signal if ONLY the less significant bit of the HSB configuration register is set – all remaining bits reset.
ext_finish_o	out	no	no	8	External output request. This signal only indicates if the selected AES block has finished its operation.
tm	in	down	no	2	Test mode (only used by the testable AES).
sc_en	in	no	no	2	Scan chain enable (only used by the testable AES).

B.3 Architecture Configuration Shift-Register

This 13-bit shift register is responsible to enable the other shift registers, the loop logic and configure the HSB signals. Table B.3 indicates the functionality of the *Architecture Configuration Shift-Register* (ACSR) signals for each AES crytocore.

Table B.3: Architecture Configuration Shift-Register (ACSR) bit usage.

<table>
<tr><th>Signal</th><th>AHLS AES</th><th>Testable AES</th><th>Description</th></tr>
<tr><td>ACSR[0]</td><td colspan="2">sel_AES</td><td>Selects the target AES. When sel_AES = 0, the multiplexers drive input and outputs to the testable AES.
When sel_AES = 1, multiplexers drive to the AHLS AES.</td></tr>
<tr><td>ACSR[1]</td><td colspan="2">input_en</td><td>High-active enable for INSR. Allows to load INSR through sel_AES_i.</td></tr>
<tr><td>ACSR[2]</td><td colspan="2">loop_en</td><td>High-active enable for the loop logic.</td></tr>
<tr><td>ACSR[3]</td><td colspan="2">output_en</td><td>High-active enable for OUTSR.</td></tr>
<tr><td>ACSR[4]</td><td colspan="2">output_mode</td><td>Selects OUTSR mode. On low, enters in normal mode. It enters in shift mode on high.</td></tr>
<tr><td>ACSR[5]</td><td rowspan="8">Not used</td><td>hsb[0]</td><td rowspan="8">HSB configuration for the testable AES.
On high, the HSB signal connects the input request signal of a given controller to ext_req_i. Otherwise, the controller operates normally.</td></tr>
<tr><td>ACSR[6]</td><td>hsb[1]</td></tr>
<tr><td>ACSR[7]</td><td>hsb[2]</td></tr>
<tr><td>ACSR[8]</td><td>hsb[3]</td></tr>
<tr><td>ACSR[9]</td><td>hsb[4]</td></tr>
<tr><td>ACSR[10]</td><td>hsb[5]</td></tr>
<tr><td>ACSR[11]</td><td>hsb[6]</td></tr>
<tr><td>ACSR[12]</td><td>hsb[7]</td></tr>
</table>

"Test and Side-channel Analysis of Asynchronous Circuits"

Résumé

Les circuits asynchrones sont étudiés depuis plusieurs décennies comme une alternative pour surmonter les difficultés liées à la conception synchrone, en particulier avec les nœuds technologiques récents qui flirtent avec les limites physiques. Ainsi, les variations de processus, de tension ou de température (PVT) peuvent avoir un impact significatif sur le comportement des circuits. Cette situation a ouvert la voie à l'usage de circuits asynchrones dans un large éventail d'applications. En raison de leur mode de conception non conventionnel, il n'est pas toujours évident de les concevoir. C'est pourquoi des méthodologies et des outils ont été mis en place pour faciliter leur adoption. Cependant, le flot de conception synchrone est bien établi et il est difficile de changer les habitudes, rendant difficile l'adoption et le développement d'un flot de conception asynchrone automatisé et optimisé. Cela se traduit également sur le développement de techniques de test et de diagnostic spécifiques à ce type de circuits. Dans ce contexte, cette thèse porte sur les techniques dédiées au test et à l'analyse des circuits asynchrones. La première partie présente une architecture pour la testabilité (DfT) des circuits asynchrones *Bundled-data* (BD) permettant d'exécuter des tests à pleine vitesse, tout en maintenant la compatibilité avec les outils de synthèse et de test d'une part, et en limitant l'impact en surface en surface du dispositif de test d'autre part. La seconde partie explore l'analyse des circuits asynchrones par canaux cachés dans un but de détection des chevaux de Troie matériels en tirant parti de leur signature en courant et de leur comportement intrinsèque. Grâce à de nombreuses simulations, il a été démontré que les circuits asynchrones sont à même de fournir des signatures locales de courant propres à une sous-partie du circuit, facilitant ainsi l'identification et la présence de chevaux de Troie.

Mots-clés : circuits asynchrones; Design-for-Testability (DFT); analyse par canaux cachés; bundled-data; Quasi-Delay Insensitive (QDI); tests à la vitesse

Abstract

Asynchronous circuits have been explored in the last decades as an alternative to overcome the issues brought by synchronous design, especially as recent technology nodes reach physical limits and process, voltage and temperature (PVT) variations significantly impact circuit behavior. This pushes forward the use of asynchronous circuits on wide range of applications. Due to their non-conventional design style, it is not so trivial to design them. Therefore, methodologies and tools have been introduced to help its adoption. However, the well-established synchronous design flow impedes alternatives and even creates resistance to further develop an automated and optimized design flow. As a side effect, this also impacts the development of testing and diagnosis techniques for this kind of circuits. In this context, this thesis targets dedicated techniques for testing and analyzing asynchronous circuits. A first part presents a Design-for-Testability (DfT) architecture enabling at-speed testing on asynchronous Bundled-data (BD) circuits, while maintaining low area overhead and compatibility with synthesis and testing tools. The second part explores side-channel analysis on asynchronous circuits for Hardware Trojan (HT) detection, taking advantage of their current signature and intrinsic behavior. Through simulation experiments, it is shown the ability of asynchronous circuits in providing local current signatures for identifying the presence of HTs.

Keywords : asynchronous circuits; Design-for-Testability (DfT); side-channel analysis; bundled-data; Quasi-Delay Insensitive (QDI); at-speed testing; Hardware Trojan; current signature; delay testing