Advanced Structural and Semi-Formal Verification Flow for Clock Domain Crossing (CDC) in Asynchronous Multiclock Systems

Diana Kalel

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Nano Électronique et Nano Technologies**

Arrêtée ministériel : 25 mai 2016

Présentée par

Diana KALEL (doctorante)

Thèse dirigée par **Katell Morin-Allory et Laurent Fesquet**
et coencadrée par **Jean-Christophe Brignone**

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

Advanced Structural and Semi-Formal Verification Flow for Clock Domain Crossing (CDC) in Asynchronous Multi-clock Systems

Thèse soutenue publiquement le **04 juin 2024**,
devant le jury composé de :

Giorgio Di Natale
Directeur de recherche, CNRS délégation Alpes, Président

Ayman Wahba
Professeur, Université Ain Shams, Rapporteur

Sébastien Pillement
Professeur des universités, Ecole polytechnique de Nantes, Rapporteur

Matthieu Moy
Maître de conférences, Université Lyon 1, Examinateur

Michele Portolan
Maître de conférences, Grenoble INP, Examinateur

Jean-Christophe Brignone
Senior Staff ingénieur, STMicroelectronics, Co-Encadrant de thèse

Jerome Avezou
Senior Staff ingénieur, Synopsys, Invité

"Dedication"
To those who persevere in believing in themselves, defying the doubts and criticisms,
to those who maintain faith in the eventual fulfillment of their dreams,
to those who persist in their journey even when the light remains unseen,
and to the kind souls we encounter along our journey, whose presence makes the path brighter and brings warmth and sweetness
this thesis is dedicated

Acknowledgements

I extend my deep gratitude to my thesis jury members, **Giorgio Di Natale**, **Ayman Wahba**, **Sébastien Pillement**, **Matthieu Moy**, **Michele Portolan** for accepting this role and for contributing their valued experience and comments to my thesis.

I want to thank my PhD supervising team at TIMA lab for their time and effort in adding their valued scientific vision to this PhD.

I am also thankful to my industrial advisor, **Jean-Christophe Brignone**, for providing me with the opportunity to dive into the CDC verification domain and teaching me a lot on the technical and the personal level. Special thanks go to my teammates, **Julian Massicot** and **Lionel Picandet**, for their joyful and kind presence. Working alongside you both was truly enjoyable.

I am grateful for the enlightening technical discussions I had with **Irene Serre**, **Sebastien Ferrousat**, **Christophe Chanet-Chene**, **Cyril Chevalier**, **Stephane Farrouch** and **Nicolas Aubailly**, which taught me a lot. A special acknowledgment is owed to our functional verification expert, **Massimo-angelo Calligaro**, for his time and assistance in bridging two activities that were never previously connected. Appreciation is also dedicated to my hierarchical manager, **Guy Durieu**, for the time and support he dedicated to me during this journey.

I feel fortunate to have had the opportunity to work with **Jerome Avezou** and to receive technical support from him. Thank you for bringing such a joyful spirit and for fostering positive and kind vibes.

I am grateful for meeting and being supported by **Jumana Boussey**, **Olivier Rossetto** and **Dominique Schneider**. Their kindness and support were a major turning point in my thesis journey.

On a personal level, I couldn't have achieved any of this without the presence and constant support of my parents, **Joseph** and **Safaa**. Despite the distance, your constant support kept me going.

"if either of them falls down, one can help the other up". Thank you, **Mark**, for this 10 years valued and true friendship that never fails me and always has my back. Your encouragement was what pushed me to the finish line.

The support of my friends was essential in overcoming the difficulties of this journey and persevering until the end. Therefore, I want to extend my gratitude to **Fadi** for our valuable discussions, and to **Nihal** for her support and the joyful moments we shared. Thanks also to my dear friends, **Sara**, **Kiro**, **Roshdy**, and **Awny**, for your constant presence. And to my childhood friends, **Sandra** and **Nada**, I am forever grateful for having you by my side.

Contents

Introduction

Over the past half-century, there has been a remarkable evolution in electronic design, driven by the evolution in the semi-conductors technologies, while the performance requirements are becoming more challenging. Integrated circuits are embedded in most of the devices we are using nowadays, starting from the the mobile phones we use in our everyday life, to the supercomputers performing complex calculations and simulations at unprecedented speed. As the time to market decreases with the extra-high demand, the semiconductor industry now prefers to order pre-made IPs from external providers. Due to various functionalities and power constraints, the different IPs sometimes operate with several clocks. At system level, when different clock domains co-exist and communicate together, data exchange between these different domains require a particular attention. Indeed, Clock Domain Crossing (CDC) is the cause of many problems, which can be tricky to resolve. For instance, in systems with interfaces between processor cores, memory subsystems and peripherals operating at different clock frequencies, the signals crossing these different clock domains can lead to metastability, data loss and other synchronization issues. The digital designers over the years developed multiple solutions for solving these CDC issues, such as CDC synchronization structures and protocols. Because it is crucial to properly synchronize signals crossing the clock domains boundaries, the CDC synchronizers ensure the integrity of the data being transferred. There exist several types of CDC synchronizers that will be discussed later in Chapter 1

The CDC aspects require rigorous verification techniques to be applied to identify potential CDC hazards, such metastability, missing synchronization, data coherency problems and others, at early stage of the design flow. Traditional STA (Static Timing Analysis), while effective on synchronous paths to identify timing violations, cannot be used for the CDC verification. Facing the unpredictable timing relationships and its dynamic characteristics on asynchronous paths, the STA lacks the comprehensive mechanisms to verify the full range of the asynchronous scenarios. Therefore, the EDA (Electronic Design Automation) tool companies offer specialized software for the CDC verification. The CDC verification relies on: (1) the synchronizers detection, (2) the functional validation of the detected synchronizers. The detection of synchronizers involves both semantic and syntactic approaches. The syntactic approach is employed for the synchronizers detection, which usually relies on recognizing structural patterns in the code or the netlist. A semantic approach can also be used for synchronizers detection by analyzing the use of finite state machines or other behavioral characteristics associated with synchronizers. The CDC structural verification is a static verification depending on patterns matching. It is the most, and sometimes the only, used approach by most of the designers to verify the asynchronous paths. The CDC structural verification, its deficiencies and our related contribution are developed in Chapter 2. Once the synchronizers detection is fully done, a functional validation can be performed usually by asserting the properties associated to each detected synchronizer. The properties verification can be done with a formal engine

or in dynamic simulation. Associating the CDC assertions to dynamic simulations allows to perform the CDC semi-formal verification. This is developed in Chapter 3. Finally, the CDC verification can be complemented by a special type of errors injection mimicking the metastability. The metastability injection is the subject of Chapter 4

The industrial CDC verification tools, despite of getting more mature in the last decade, are still not 100% reliable. The verification flow still depends widely on human effort and expertise for setting the tools and the design up. Setting the tools' parameters and the design constraints requires a good understanding of the design and the tools. Any human error in this phase leads to sub-optimal results. In addition, the pre-coded patterns which the tools try to match in order to detect the synchronizers do not work with custom and complex synchronizers. The result is that the tools either oversimplify the structures and report them as valid while they are not, or fail to converge the analysis and report some valid synchronizers as absent. In both cases, the number of false positive and false negative results are unacceptable. Indeed, the majority of the analyzed CDC violations are either false negatives or false positives.

Finally, despite the fact that the technology for verifying the CDC paths exists since 20 years now, we still find post-silicon bugs due to CDC. The existing tools have multiple deficiencies that must be taken into account. In addition, the verification flow existing today is very prone to human errors. As the target of this thesis is to develop and improve the CDC verification flow respecting the working frame constraints of STMicroelectronics, we made our decision to tackle the following subjects respecting the following priority:

- Being the main verification signoff flow, the CDC structural verification flow needs to be enhanced and standardized, overcoming the deficiencies exiting in the native flow and minimizing the number of false results. The target was to make this new flow available as soon as possible for projects. The idea is to have a robust implemented flow, that can be used by the different teams and that does not necessitate a deep expertise in the CDC verification domain.

- The CDC structural verification is a very important task to ensure the presence of the necessary synchronizers, even though, it is not enough to ensure that they function correctly. Therefore, it must be complemented by a formal or a semi-formal analysis. The industrial constraints and the complexity of the designs pushed more toward the development of a semi-formal flow, as a trade-off between exhaustiveness and the time the verification process takes.

- As the metastability is the greatest risk of any asynchronous multi-clock system, its effect should be taken into account in the verification flow. Several metastability injection flows are studied and tested on our test case.

Thesis structure In this thesis, we embark on an exploration of a field that did not grab the attention of a lot of researchers in the past. This explains the narrow literature of the CDC verification we were relying on. The structure of this thesis is designed to provide a progressive understanding of the CDC issues and the existing technologies to verify them. As the main target of this thesis is to develop an optimized flow for the CDC verification, the following chapters navigate through four distinct sections, each contributing to a nuanced analysis of what exists already and a proposal to push the limits of this field. The four sections are the following

- **Chapter 1 Background and state-of-the-art:** presents the CDC aspects and provides a resume about the literature review and the efforts done in the domain.
- **Chapter 2 CDC structural verification:** explains the problems related to the CDC structural verification flow and proposes a new enhanced and rationalized flow. A comparative study between different tools is provided. We also tackle the inclusion of low power logic structures and their effect on the CDC verification. In addition, a new UPF-aware methodology to verify the CDC paths is presented.
- **Chapter 3 CDC semi-formal verification:** in this chapter we present the idea of the semi-formal verification flow. As the verification of the CDC data-paths synchronized by custom synchronizers has always been a challenge, we propose a generic CDC modeling for the data stability verification. The implementation of an universal qualifier detection algorithm to assist the CDC verification is also presented in this chapter.
- **Chapter 4 Metastability injection:** the metastability, being an analog phenomenon, cannot be inspected in a digital simulation. However, the effect of the metastability can be mimicked and injected in a digital simulation. The formalization of the metastability effect and a presentation of the different metastability injection approaches are presented in this chapter. Additionally, we present the first results we obtained injecting metastability on a small CDC test case.

Test Case: The test case we have used for all our studies, evaluations and developments is a commercialized design with a high level of maturity, developed at the CPU division of STMicroelectronics. As shown in Figure 1, it is a CPU subsystem integrating a 64 bits dual-core CPU with two levels of cash memory. The CPU subsystem includes a PLL and a clock divider to generate internal clocks and one reset generator. It is driven by 14 different clock domains, as color coded on Figure 1. The CPU subsystem includes several blocks dedicated to the management of the asynchronism with the external world. Some of them are standard bus interfaces which can be found around the subsystem to dialogue with the logic outside. The unidirectional inputs are passed through synchronizers inside the subsystem, while all unidirectional outputs are defined as asynchronous as it is the responsibility of the receiving clock domain on SoC level to re-synchronize. Some other asynchronous interfaces are for debug and power management purposes. In addition, the design includes DFT (Design For Test) wrapper and a high speed BIST for memories. In terms of power management, the design operates under two voltage domains and three power domains.

Working in the CPU division in STMicroelectronics and given the number of clock domains and of the standard asynchronous interfaces included in this CPU subsystem and its maturity level, we made our choice to use it as test case for all our trials and applications. Whenever a test case is mentioned in the next chapters, we will be referring to this CPU subsystem, unless other test case is explicitly mentioned.

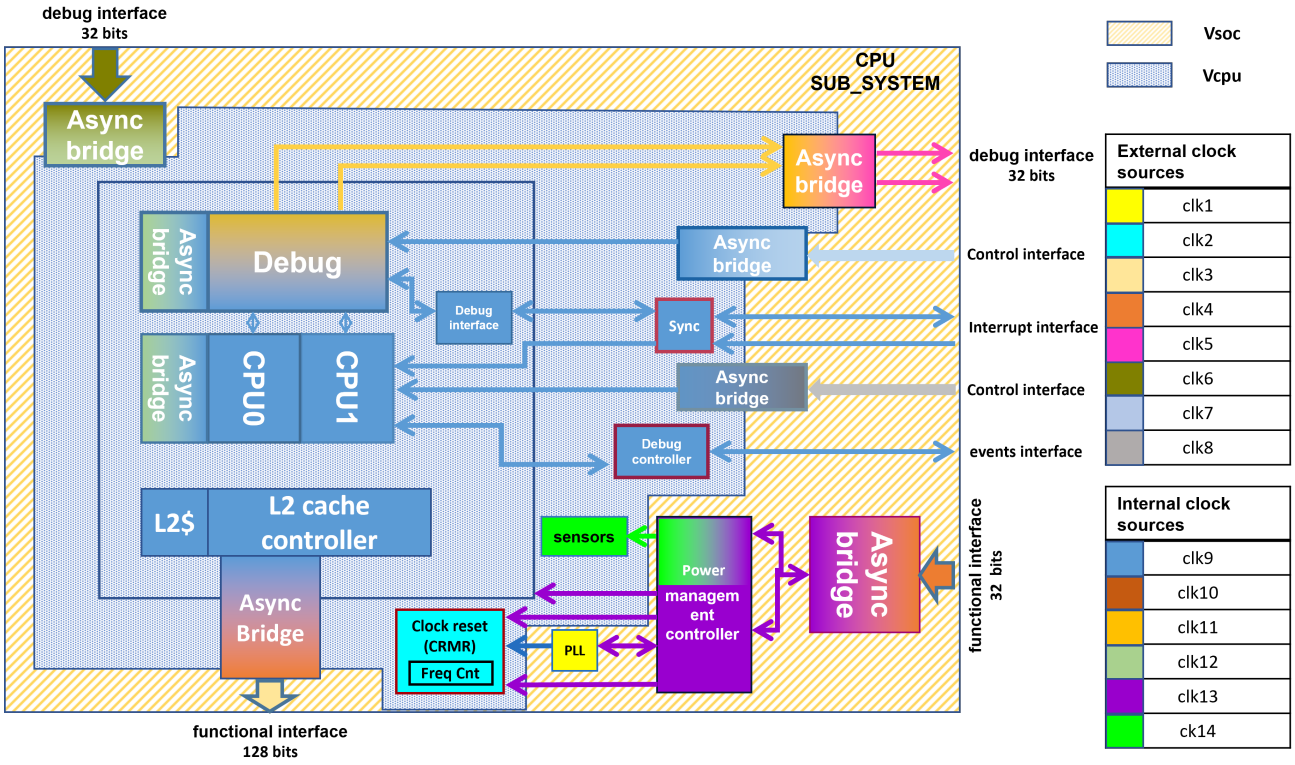


Figure 1: Test Case

1 Background and state-of-the-art

Distributing a global, low-skew clock in extensive System-on-Chip (SoC) infrastructures remains a challenge for traditional fully synchronous systems. To address time-to-market demands and evolving design complexities, the integration of independently clocked and powered IPs, known as Globally Asynchronous Locally Synchronous (GALS) systems, is employed. While this approach avoids large clock-tree designs, it suffers from the absence of global synchronization, particularly in data transmission between different blocks and Clock Domain Crossing (CDC). Despite exhaustive verification efforts for multi-clock systems, focusing on well-designed and functional asynchronous interfaces, the state of the art reveals gaps in the CDC verification flows. There is a need for rationalization and standardization, rectifying deficiencies in the existing native flow. Furthermore, this thesis aims to delve into emerging verification aspects, such as semi-formal verification and metastability injection, to enhance the verification of the asynchronous multi-clock systems.

Contents

1.1 Background

1.1.1 Current circuits state

Synchronous circuits are widely produced nowadays by the industry and their design and verification flows have achieved a high level of maturity. In these circuits, a periodically toggling signal, the clock signal, determines the timing of each operation. This makes the behaviour of such systems predictable, prevents race conditions and makes the design process easier. The timing closures are ensured to be met by the Static Timing Analysis (STA). The design is broken into several timing paths with specific start and end points to ensure that the setup/hold requirements (for data paths) and the recovery/removal requirements (for reset/set paths) are met. Using the gates propagation timings defined in the SDF files and the specified maximum and minimum delays (corresponding to the setup/hold requirements respectively), the timing checks are done and the clock frequency is adjusted. False paths and multi-cycle paths are also specified to be excluded from the analysis [1]. Synchronous circuits dominated the silicon industry since the 1960s and are seen as the safest and easiest approach regarding design reliability, performance, and testing ease. The number of clock cycles needed to perform a task is an easy and reliable metric to evaluate the performance. Although, they present many problems too. Synchronous circuits lead to different types of deficiencies specially in large and fast designs. While data is ready to be used, the system should wait for the next clock tick to sample it and the tick must be long enough to avoid capturing corrupted data, which means a time overhead. The clock signal itself cannot reach all the elements of a circuit at the same time, so, special structures (clock trees) must be used to ensure a perfect alignment between all the clocked elements, which means an area overhead. Finally, a periodically toggling signal consumes power and produces heat, which means a power overhead. That is why the research and the industry are trying to find alternatives.

Asynchronous clockless circuits In the last decade, designers began to revise an old concept which was ignored for years, the asynchronous clockless circuits. They are self-timed and do not use the periodically toggling signal usually used in synchronous circuits to pace the results. Instead, they put them as soon as they are ready. The components use local synchronization represented in specific communication protocols (such as the handshake protocol). The functioning of the asynchronous clockless designs is described to be event based, only the necessary components are active at a specific moment. In general, clockless circuits are so advantageous in terms of power use. The absence of a periodically toggling signal makes remarkable energy savings. In addition, the electromagnetic interference (EMI), that freaks out most of the designers, is extremely low. These factors made the asynchronous clockless circuits recently an important and competitive alternative. However, experts still have some concerns: the lack of development tools and the difficulties they face integrating them with synchronous circuits. But still, the advantages they present to the market are worth the research and the development to normalize their use. More and more resources and tools are developed everyday to reach that target [2] [3] [4].

Asynchronous multi-clock circuits It is the alternative to the globally synchronous and the fully asynchronous clockless circuits. Each block in an asynchronous multi-clock system runs at its nominal minimum speed and then a wrapper takes care of the

communication with the other blocks (by disabling the clock or controlling the exchanged data). The main applications are the large systems with one or more processors that interface with shared peripherals or memories. For example, a CPU subsystem takes advantage of the design on chip bus architecture and is connected with asynchronous interfaces acting as bridges. These systems present a remarkable advantage regarding power consumption and design ease. However, some other problems, due to clock, reset and power domain crossing, start to appear and become the subject of different types of exhaustive verification [5] [6] [7].

The asynchronous multi-clock systems are the major concern on this thesis and will be discussed in more detail in the next section.

1.1.2 Asynchronous multi-clock systems

Building an extensive SoC infrastructure while avoiding the problems of distributing a global, low-skew clock is not possible. The system complexity makes it difficult to maintain a reasonable clock skew over a large area and through millions of gates. This leads to a huge clock-tree, which is power consuming, sensitive to process variations and difficult to balance. The overhead, due to clocking, in terms of power consumption, has become unacceptable [8] [9] [10] as more than one half of the power consumed in a system is due to clocking (see Figure 1.1). To meet time-to-market requirements and to handle the evolution of the design complexity, independently clocked and powered IPs are integrated together. In other words, large synchronous systems are split into a set of small synchronous sub-systems in order to implement more compact clock-trees. This alternative, also known as Globally Asynchronous Locally Synchronous (GALS) systems, prevents from designing large clock-trees but suffers from the loss of a global synchronization.

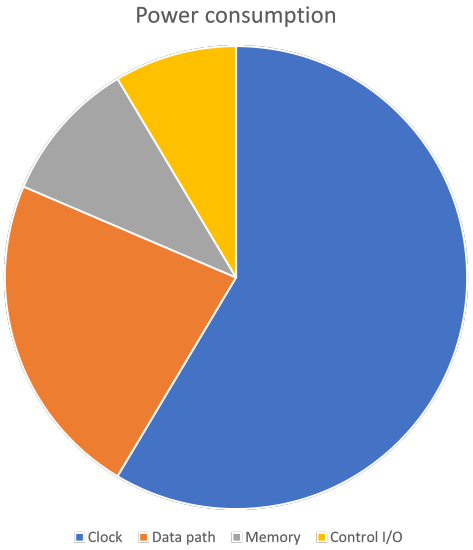


Figure 1.1: Power consumption distribution along design clusters

Definitions

Clock domains Two clock signals belong to the same domain if they are in phase, even if they have different frequencies. Two clock signals can also be considered in the same

domain if they have a constant phase shift and the setup/hold timing are adjusted during the physical implementation. In Figure 1.2, "clk_A.1", "clk_A.2" and "clk_A.3" are in phase. The three clock signals have neither the same frequency nor the same duty cycle, however, they belong to the same clock domain "Clock domain A". While "clk_A.4" has a constant phase shift with respect to "clk_A.3", it can also belong to the the same clock domain "Clock domain A". In Figure 1.3a, "clk1" and "clk2" are issued from the same source while "clk3" is sourced from a different PLL. Figure 1.3b shows that "clk1" and "clk2" have a constant phase shift (phi1=phi1'=phi1''), so they belong to the same clock domain. However, "clk3" has a dynamic phase shift with respect to "clk1" and "clk2" ($phi2 \neq phi2' \neq phi2''$), so, it does not belong to their clock domain [11] .

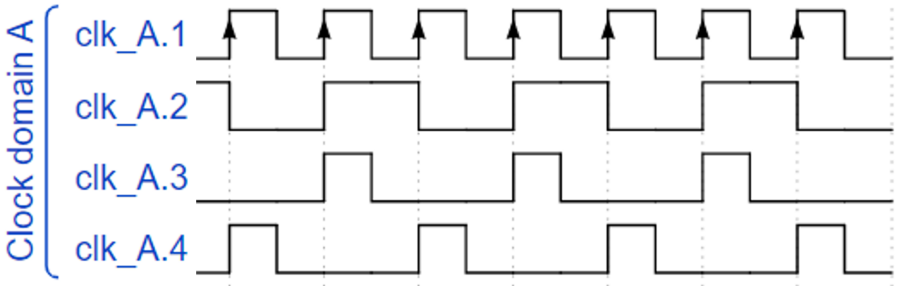
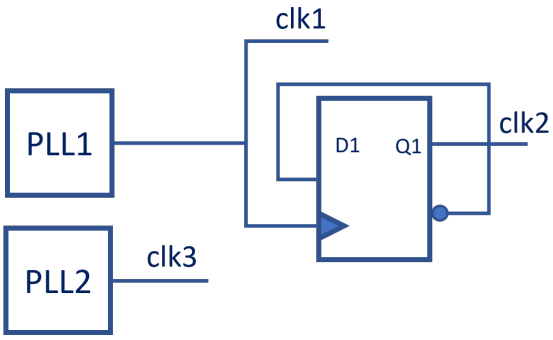
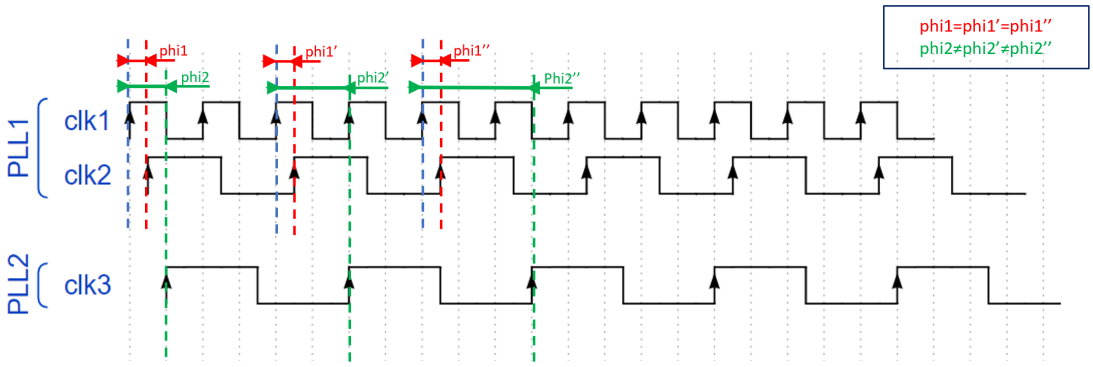


Figure 1.2: Synchronous clock domains



(a) Structural view



(b) Waveform

Figure 1.3: Asynchronous clock domains

Clock domains communication types A communication is called "synchronous" if the two communicating blocks are driven by clock signals belonging to the same clock domain. On the other hand, it is called “asynchronous” if no timing relationship can be defined between the communicating clock domains, in other words, the clock domains are completely considered unrelated with a dynamic phase shift. There exists a third type called “loosely synchronous”. In this case, there is a dependable, well-defined relationship between the communicating clock domains. As shown in Figure 1.4, this dependable relationship may be [6] [7] :

- Mesochronic: the 2 clock signals are of the same frequency with a constant phase difference (due to the propagation delay) and the setup/hold timing are corrected.
- Plesiochronic: both clock signals operate at the same frequency but having some slight mismatches. These mismatches may drift the phase for few parts per million.
- Heterochronic: the sender and the receiver operate completely at different frequencies but are still related with a fixed relationship. A subset of this is the "ratiochronic" relationship, where the sender and the receiver are both multiple of each other and derived from the same source.

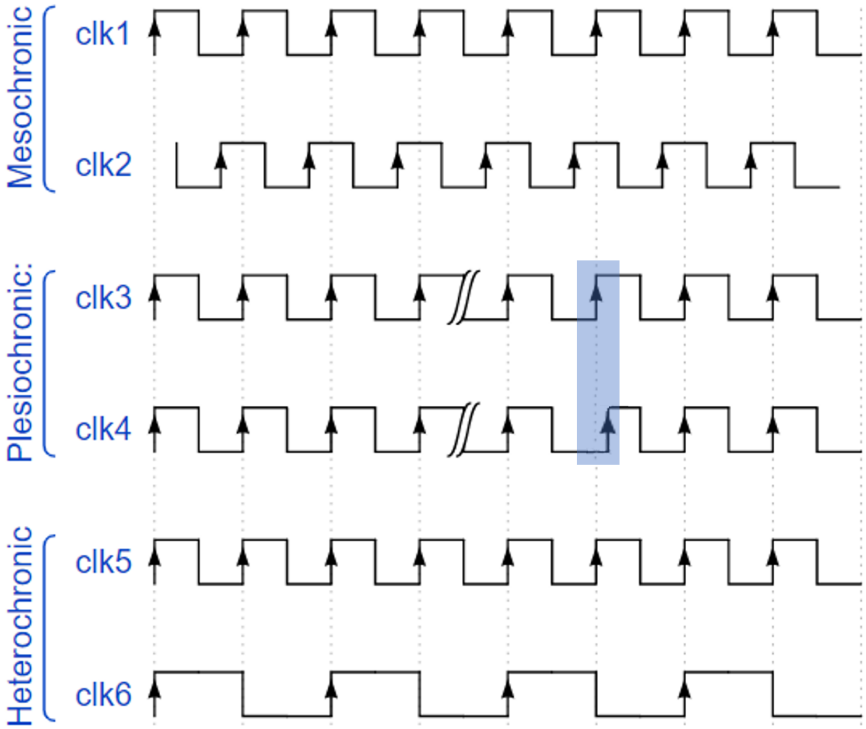


Figure 1.4: Mesochronic, plesiochronic and heterochronic relationships

Locally synchronous globally asynchronous systems (GALS) : In order to meet time-to-market constraint and to optimize the design process, functional blocks, locally synchronous but asynchronous between them, are integrated together on SoCs (see Figure 1.5]. Clocking every block at its minimum speed makes the SoC achieve remarkable power savings [7]. The inter-block exchanged data are meant to be synchronized on the

entry of the receptor block. This approach achieving great power savings, has major problems related to CDC (Clock Domain Crossing) and the risk of metastability propagation.

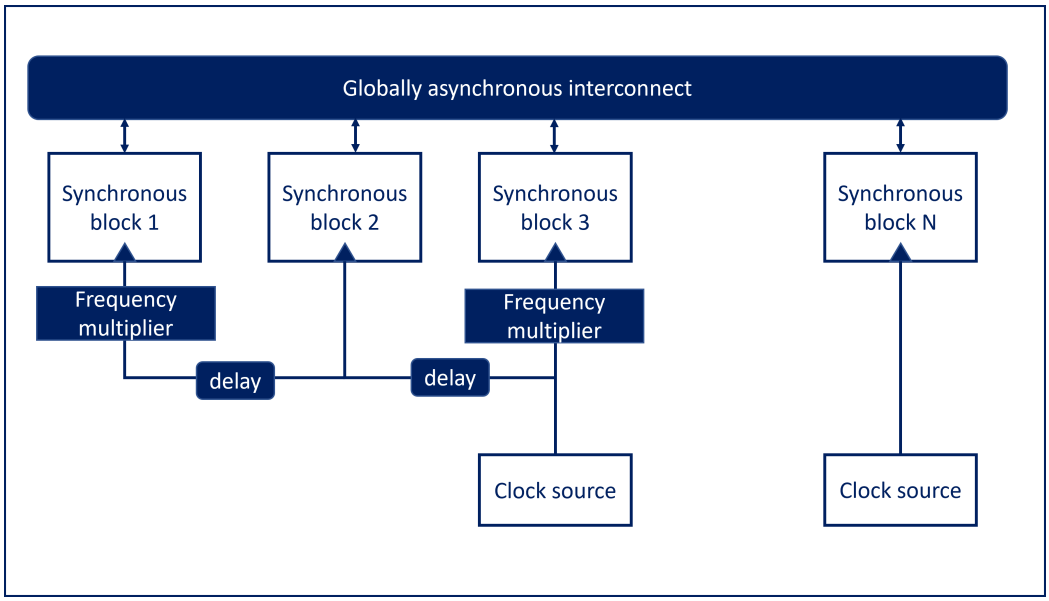
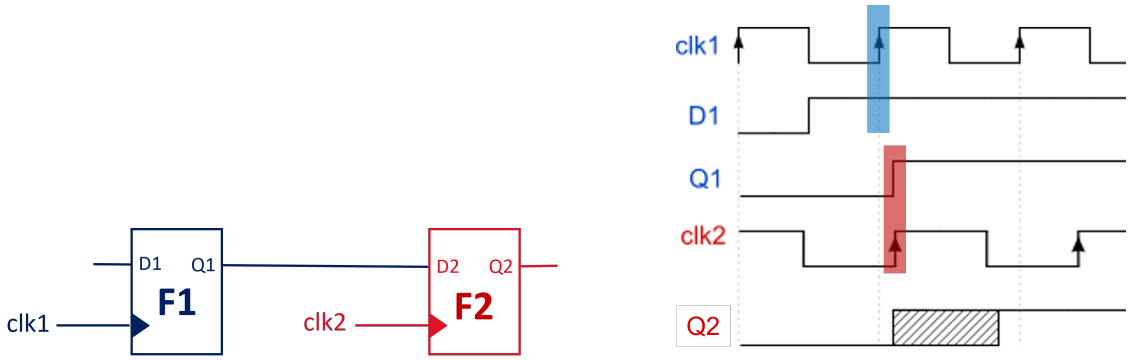


Figure 1.5: GALS block diagram

Clock domain crossing (CDC) In synchronous systems, the data is registered at the source and captured at the destination by the same clock, or with another but synchronous clock, which guarantees a stable and synchronized communication. But if the source and destination are clocked by two different asynchronous clock signals, there is no guarantee that the data will be captured at the right moment (when it is stable) respecting the setup/hold criteria. In Figure 1.6a, "clk1" and "clk2" are asynchronous. As Figure 1.6b illustrates, the rising edge of "clk1" happens too close to the rising edge of "clk2" violating the setup time which risks to propagate a metastability on "Q2" [12].

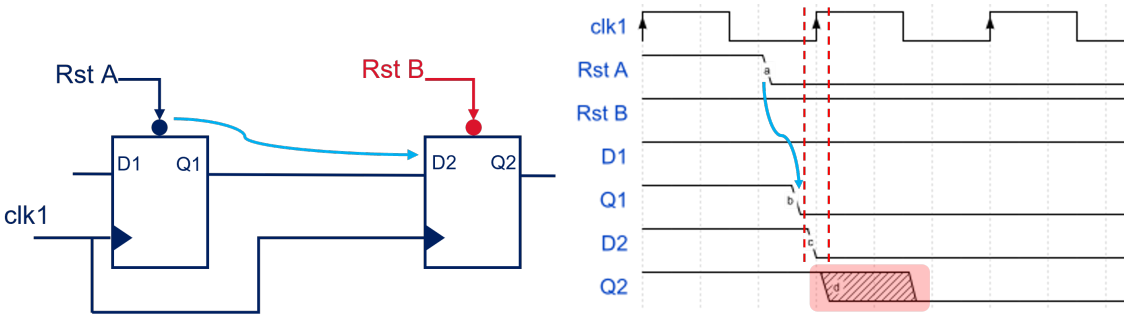


(a) Structural view *(b) Waveform*

Figure 1.6: Clock Domain Crossing (CDC)

Metastability It is a phenomenon that happens when a flip-flop or a latch enters an unstable state and is unable to resolve to a logical value within an acceptable time that guarantees a proper operation. In Figure 1.6b, "Q2" goes metastable due to the setup timing violation happening because of the CDC present between "F1" and "F2".

Reset Domain Crossing (RDC) It is the interface between two sequential elements controlled by two different reset signals (see Figure 1.7a) [13] [14] [15]. In Figure 1.7b, "Rst A" is asserted asynchronously on the falling edge "a" and the value of "Q1" drops very close to the sampling clock edge. That is why a metastability is propagated on "Q2". One of the solutions to avoid the RDC problems is to define a reset ordering. The destination reset should be asserted before the source reset to block the metastability. Other RDC synchronizer can be found in [16].



(a) Structural view *(b) Waveform*

Figure 1.7: Asynchronous clock domains

Power/voltage Domain Crossing The presence of "Power domains" is essential to reduce the power supply voltage to some parts of the circuit. Each block is powered by its minimum required power. Crossing from one power domain to another requires special logic (level shifters, isolation cells, etc.) to be inserted to ensure a safe data communication (see Figure 1.8). These structures are defined in the UPF file (Universal Power Format) [17].

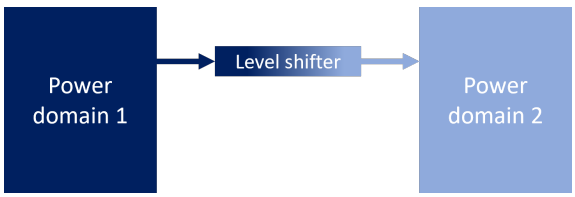


Figure 1.8: Voltage domain Crossing

1.1.3 The global digital design flow

An IC (integrated circuit) has typically two main sections, the analog part which generally interacts with the outside real world using all levels of voltages (eg. RF receiver - sensors) and the digital part which is responsible for data transfer and processing. In this section, we will focus on the digital design flow to locate where exactly the CDC problems appear.

Specification to RTL *(Figure 1.10 section 1)*

The specification is the first step to start a digital project. It is a high level representation of the system including many factors such as performance, functionality, the size of the chip, the design techniques and the fabrication technology (transistor sizing). The design

architecture is then done and implemented following three main steps. The first step is the "Behavioural Design" where the top level function of the design is described. The second step is the "RTL Design", which is the data flow between registers. Finally comes the "Netlist Design" which concerns the direct instantiation of cells from libraries [1]. The different parts of the design are described using a hardware language (VHDL or Verilog). The same is done for each IP and then they are integrated together on a SoC (System on Chip).

RTL integration ***(Figure 1.10 section 2)***

Integrating different IPs -differently powered and clocked- will implicitly create many asynchronous interfaces where the problems related to CDC, RDC and power and volatge domain crossing start to appear. The instrumentation has a very important role to fix these issues. Instrumenting RTL is to make it compliant with the design rules on the different steps of the flow using either standard or customized cells or libraries . This includes the insertion of the synchronizers on clock domains crossings and the insertion of the level shifters on the power domains crossings for example. At this stage, we have a complete description for the whole design needing to be standardized. A specific tool, depending on each provider, gathers all the RTL files written in HDL and transform them to XML files compliant with the IEEE standard IP-XACT. This standard was published in February 2010 by IEEE and its goal was to deliver compatible components from many vendors and to exchange the data between different EDA (Electronic Design Automation) tools. The IP-XACT format packages the IP's :

- ports/interfaces to facilitate their integration to the rest of the design without parsing all the system's verilog files,
- hardware and software's memory map describing the software interface with the IP and creating some header files about the registers addresses and their fields,
- a file manifest identifying all the necessary design's files, what they are and where they do appear.

Synthesis ***(Figure 1.10 section 3)***

Synthesis is the process of translation of the RTL to the implementable gate-level netlist. The overall process goes on several steps and is based on specific constraints [18].

- **Generic synthesis** : The RTL is translated to a technology independent Boolean representation. At this stage, a one-to-one correspondence can be done between the RTL and the generated synthesis (that will not be anymore available after the optimization). The GTECH (generic technology) netlist is the output of this phase which can be reusable for any kind of technology [19]. In general, any design structure can be optimized based on the "Major Design Optimization constraints". This latter is tunable by the user on 3 major axis : Area, Power and performance known as the PPA constraints. These optimizations can break the integrity of the instrumented synchronizers; that is why CDC paths must be exhaustively checked after synthesis.

- **Mapping** : The generic netlist generated from the last step should be mapped to a number of library cells guaranteeing maximum efficiency and re-usability of the design [20]. DesignWare is a set of libraries that contain a number of high-level functional modules such as adders,subtractors, shifters, FIFOs, counters, comparators and decoders. All these modules are parametrizable, synthesizable and technology independent. The GTECH netlist is then mapped to these libraries ensuring that the high-level optimization features, such as resource sharing and arithmetic optimisation, are turned on [21]. The generated netlist can then be mapped to the technology libraries which are a collection of gates associated to certain characteristics and provided by a fabrication house (*e.g.* TSMC).

- **Design for test (DFT) insertion** : it means considering the testing aspects during the design process itself. Testing is done after the fabrication using a mechanical device called tester. Its pins get connected to some specific pins on the chip, they inject a test pattern and read from specific pins dedicated for testing. Sometimes some defects may be found in the fabricated devices (*e.g.*shorted or broken lines) . As a DFT requirement, controllability and observability have to be considered early in the design. But as a chip can not afford a pin at the input and the output of every single gate, the *Scan Chain* is used. The scan chain connects all the flip-flops of the design in series. Figure 1.9a represents a part of a normal design where the data goes in all directions between the combinational and the sequential parts. In Figure 1.9b , all the flip-flops are replaced by a multiplexer and a flip-flop controlled by the "SE" or "scan enable" signal and are connected in series. Any data put on "SI" may be observed on the output of every flop after a specific number of clock cycles. For example, any data on "SI" may be observed 4 clock cycles later on "SO". The "scan insertion" is done in a mechanical way and special clocking is done during the test mode [1]. The functional clock signals are blocked and the testing clock signals are propagated. This poses a new challenge regarding CDC. Propagating new clock signals must be carefully done and checked in order to guarantee the correctness of the new CDC paths.

(a) pre scan chain insertion *(b) post scan chain insertion*

Figure 1.9: Design For Test (DFT) insertion

- **Timing optimization** : Inserting the scan chain, a path that used to meet certain timing constraints may violate them. The extra multiplexer inserted in front of each flop will add additional delay to the data path which will increase the setup time requirement on the destination flop. So, a timing analysis and optimization shall be done after the DFT insertion.

Placement and routing *(Figure 1.10 section 4)*

This is the last step before fabrication. Once the final netlist is ready, the floorplan can be created. The tool calculates the area and the floor plan based on the core utilization factor (area of the design / the core area). A basic floor plan is constituted of 2 main areas : the core area is the central one containing all the cells and the ring area extended on all the perimeter including the I/O and carrying the Vdd and the Gnd which are routed to provide Vdd and Gnd to all the cells in the core area. After placing the standard cells, an evenly distributed clock signals must be ensured. This is done by the clock tree synthesis process whose goal is to ensure minimum clock skew and latency. Finally, based on the logical connectivity, the different signals (power, clock and data) are routed physically and it is ensured that the routes do not violate any timing criteria. Once done, the design is ready to be fabricated.

1.1.4 The global digital verification flow

The verification "*is the activity that determines the correctness of the design that is being created. It ensures that the design does meet the specifications required of the product and operates properly.*" [22] [23] The verification is not a one time thing but it starts form the first day of a project and goes along side-by-side the design process. At each of the previously explained design steps, a verification procedure is done to ensure a clean output to the next step. Check the global verification flow in Figure 1.11. The verification tasks are classified according to different criteria.

Static *vs.* Dynamic verification

A way to classify the different verification processes is the static-dynamic classification which answers to the "How" question.

- **Static verification :***(green part on Figure 1.11)*
  The static verification is a patterns independent approach with no need to stimulate the design in any way. The static checks are done at the early stages of the design and allows many bugs to be found. It's more related to the analytical or the structural techniques such as model checking and patterns matching. But, these approaches are usually complicated and computationally complex [24].

  - Lint checks : It is a static code analysis that checks that the RTL description is conform with thousands of standard/custom rules based on the good coding practice. This step ensures that we come with a clean RTL before proceeding into synthesis and simulation which saves time. The presence of unintentional latches, out of range indexing and combinational loops are some examples of the violations linting aims to find.
  - Low power checks : The UPF (Unified Power Format) file, where power and voltage domains and their logic are described, is syntactically verified. The static low power verification aims to ensure that the low power intent is complete and that the design and the power description are aligned. It also looks for the missing low power logic (isolation cells, level sifters, etc.) [25].
  - Clock/Reset Domain Crossing (CDC-RDC) checks : it ensures that the signals crossing clock and reset domains are received reliably. This is done by pattern

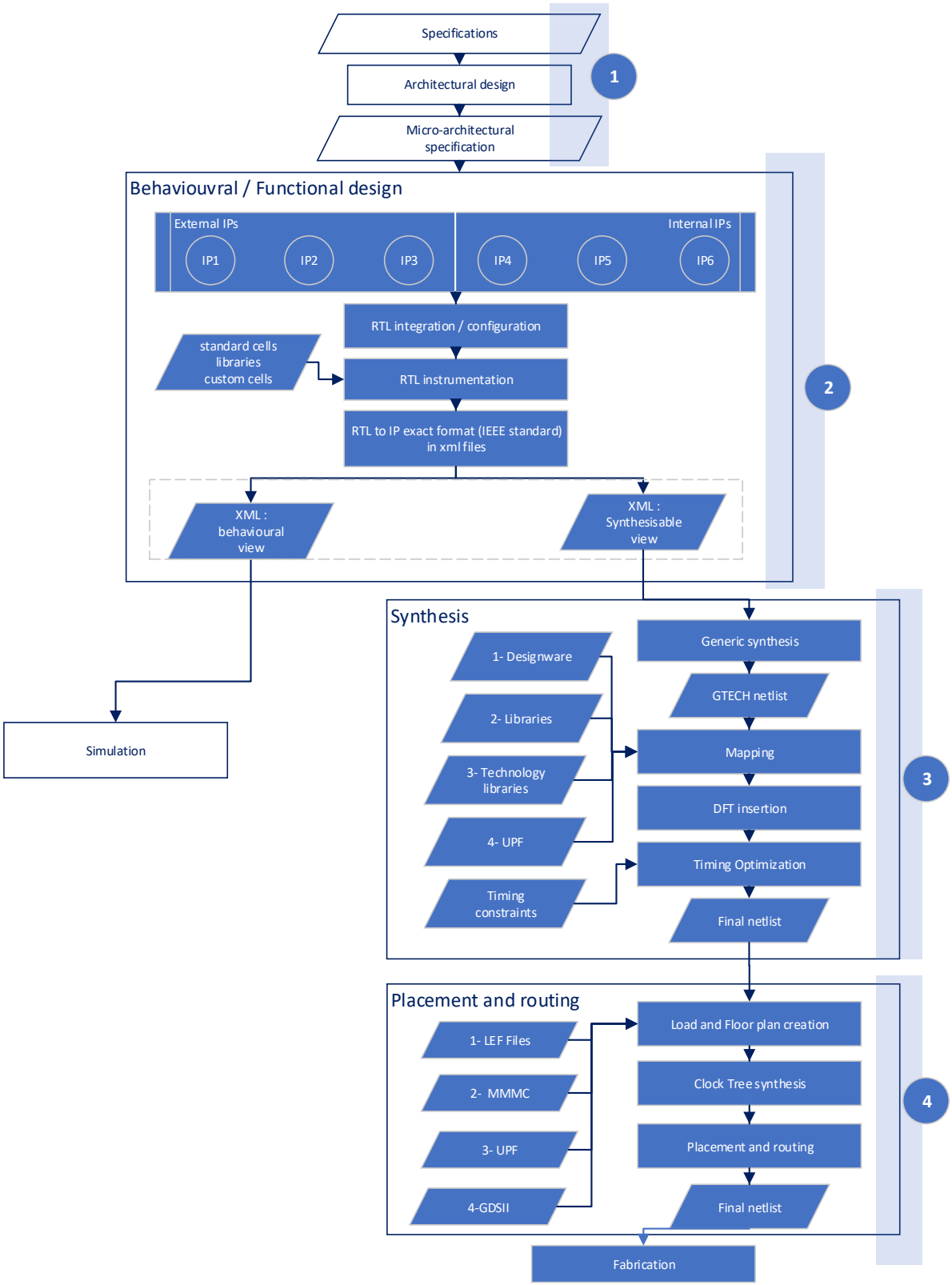


Figure 1.10: Global digital design flow

matching the design to detect the different synchronization schemes and the other problems related to CDC and RDC (re-convergences, glitches, etc.) [26] [27] [28]. The CDC-RDC verification are the focus of this thesis.

- Static Timing Analysis (STA): The STA ensures that the timing closure is met. It ensures that all data transitions occur before the defined setup time value and that none of them arrive before the defined hold time value. Using the gates propagation timings defined in the SDF files and the specified maximum and minimum delays (corresponding to the setup/hold requirements respectively), the timing checks are done and the clock frequency is adjusted [1].

- **Dynamic verification :** *(violet part on Figure 1.11)*
  In the dynamic verification, Which is usually done by simulation, a pattern of stimulus is intended to stimulate the different ports and pins of the design. This is why it is called dynamic, as the different stimulus are propagated through the design's inputs and then the outputs are inspected to ensure whether they respect the expected outputs patterns or not. The target is to explore the complete design functionality using case by case testing. In other words, the verification engineer must examine the design on all the possible execution cases. The dynamic verification is qualified as a non-exhaustive and non-fully-covering approach [29].

  The Universal Verification Methodology (UVM) is a standard methodology built on top of the SystemVerilog language to verify the digital systems in the industry. It aims to create a modular and reusable testbench components to be easily integrated. In a UVM testbench, the UVM test is instantiated along with the Design Under Test (DUT) which is connected through interfaces (usually virtual interfaces). Every UVM test exercises a custom scenario with a custom DUT configuration. Inside each test, a UVM environment is instantiated, customized and configured using the factory methods. The UVM Environment contains one or more UVM agents which interact with and exercise a specific part of the DUT. The UVM Environment also contains a UVM Scoreboard, responsible for evaluating the DUT performance. Inside each UVM Agent there is a UVM monitor which samples the transactions from the DUT through the interfaces. It also sends them to the scoreboard. In addition, a UVM agent instantiates a UVM Sequencer that manages the sequences and sends the sequence items to the Driver. If the agent is active, a UVM Driver drives the sequence items received from the sequencer to the DUT through the interface [30] [31].

Structural *vs.* Functional verification

Another way to classify the different verification processes is the structural-functional classification which answers to the "What" question.

- Structural verification :
  The structural verification ensures that the design structure is as expected based on different aspects. It is interested in the presence or the absence of some structures regardless of their functionality. For example, in CDC-RDC structural checks, the tool looks mainly for the presence of proper synchronizers schemes that match the pre-coded patterns in the different tools. The verification is considered passed if

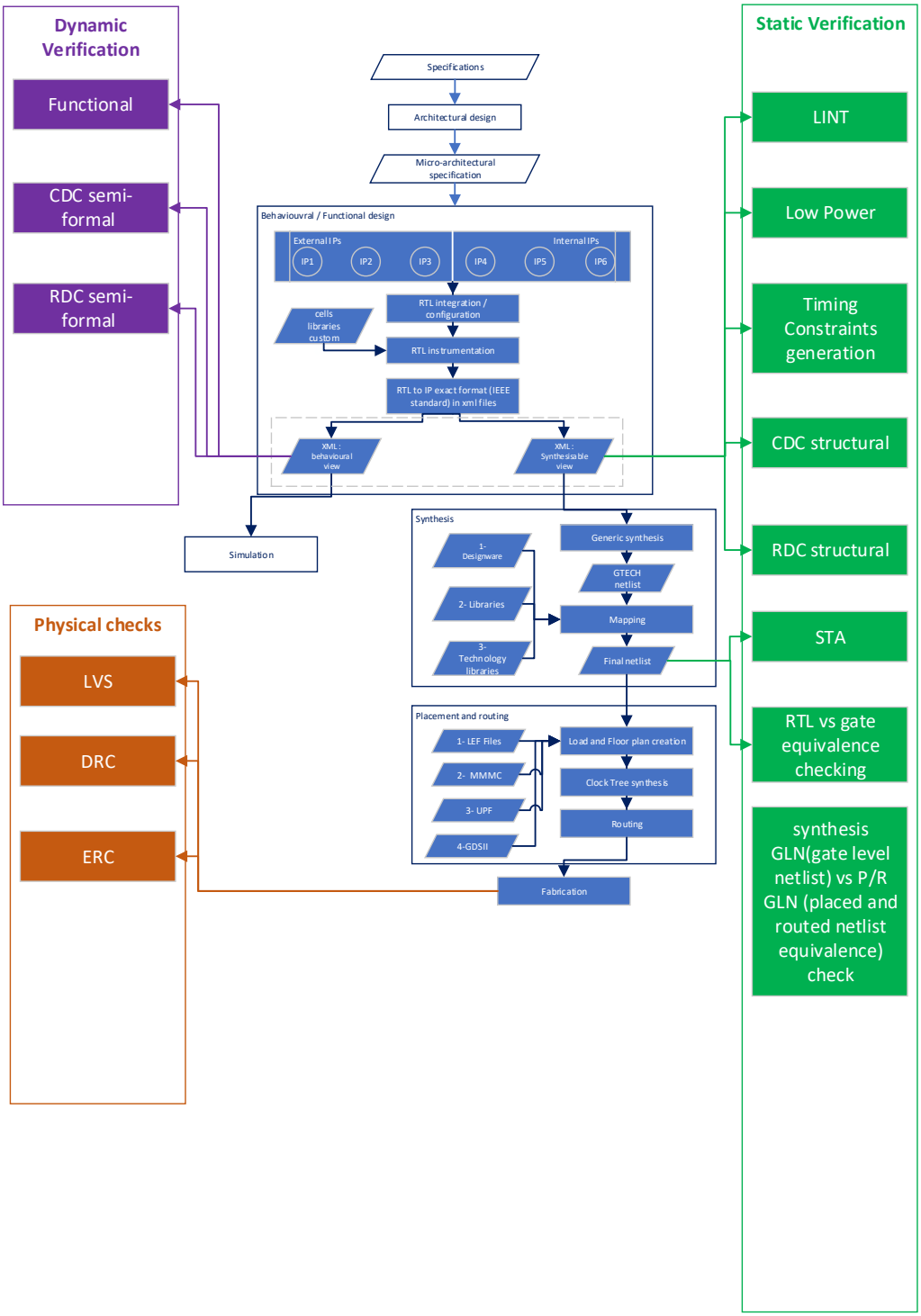


Figure 1.11: Global digital verification flow

the structures in question are found. The structural verification is usually done statically.

- Functional verification :
  Functional verification is used to confirm that the implemented design achieves the desired functionality and to explore all the functional corner cases of the design. It conforms a design to its specification, insures the proper conversion of a specification document to the RTL code without any misinterpretations of the desired specification [32]. It could be done dynamically where the design inputs are stimulated with stimulus that propagate through the design until its outputs then compared with the expected outputs from the specifications (The dynamic functional verification that uses usually the UVM) [33]. It could also be done statically base on mathematical procedures to search all the possible functional paths of the design (the formal verification) [34].

Formal *vs.* Semi-Formal verification

A third way to classify the different verification processes is the Formal-Semi-Formal classification which answers the "value versus cost" question.

- Formal verification :
  Formal verification consists of two main classes: equivalence checking and model checking. Equivalence checking compares two packages (RTL vs RTL – Netlist vs Netlist – RTL vs Netlist) to make sure that the post-processing – scan chain insertion, clock-tree synthesis or manual modification - of one of them did not affect the desired functionality of the design. In Model checking, all the design state spaces are explored statically and verified based on assertions [32].

- Semi-Formal Verification :
  In order to avoid timeout and inconclusive results due to the exhaustive analytical approach in the Formal verification, the semi-formal approach is giving a remarkably compromising approach in terms of the quality of results and the verification required time. That is why we think that this field is worth the exploration and will be explained in details in the next chapters. The semi-formal approach depends mainly on the assertions being injected in a simulation environment, conversely to the static formal approach, the properties are checked in a dynamic environment using a reliable test bench with high functional and code coverage.

1.2 Clock Domain Crossing (CDC)

In a large SoC, where multiple processors share the same memory and peripherals, the process of passing data between multiple different clock domains is not avoidable. The term "CDC" corresponds to the process of passing data between two sequential elements clocked by different and asynchronous clocks [26] [35].

1.2.1 Problems related to CDC

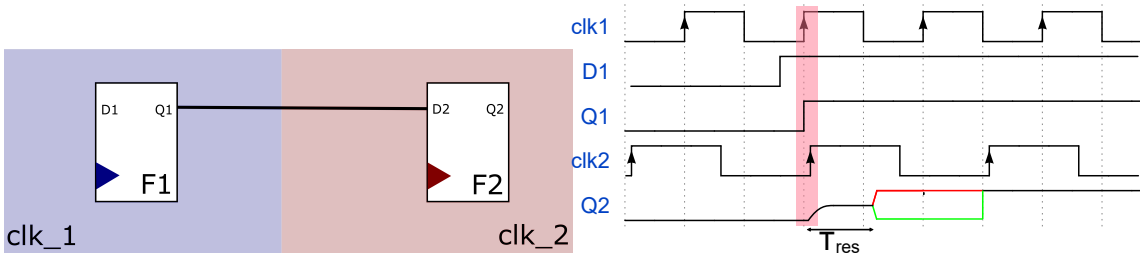
Multiple problems arise due to CDC and pose a real challenge for the VLSI designers. In this section, we will provide an overview of the major challenges related the Multi-clock systems and CDCs [36] [28].

Metastability

Metastability occurs when a sequential element is put in an unstable equilibrium state (not binary 0 nor binary 1) which breaks our standard Boolean abstraction. Violating the setup/hold timings is able to put a flip flop in this unstable state. The CDC, by definition, implies that the data can be captured on the destination side at any time with no guarantee to respect the setup/hold timings [37] [38] [39] [40] [41] [42].

In Figure 1.12a, there is a CDC path between "F1" and "F2". If "Q1" changes too close to the rising edge of "clk2", as illustrated in Figure 1.12b, the setup time is violated and "Q2" goes metastable. The metastability takes a time, called the resolution time "τ_{res}", in order to be resolved to a random boolean stable value.

What is so dangerous about metastability is that it can propagate and infect other logic. If a metastable value is propagated through combinational logic, the system is considered *dead*. Finding the source of the metastability after the fabrication is very difficult as the industrial testers can not understand non-binary values.



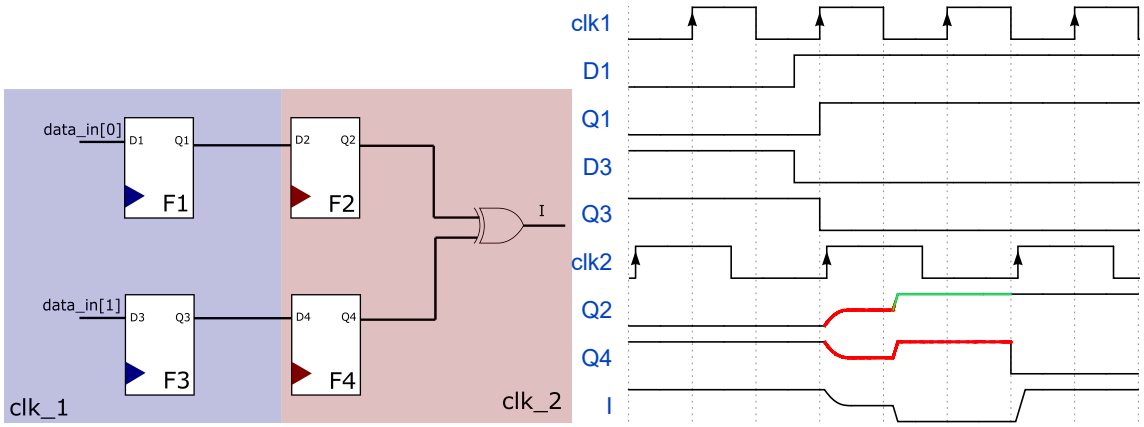
(a) Structure *(b) Waveform*

Figure 1.12: Metastability due to CDC

Incoherency

The nondeterministic resolution time and value of the metastability can lead to incoherency problems. This happens if the crossing signals are the different bits of a data bus or if they re-converge later in the fan-out of the design.

In Figure 1.13a , *data_in[0]* and *data_in[1]* are crossing the clock domain boundary between clock domain *clk1* and *clk2* and then converge on an XOR gate. As illustrated in Figure 1.13b, *"Q1"* and *"Q3"* change too close to the rising edge of clock *"clk2"* violating the setup time which generates a metastability on *"Q2"* and *"Q4"*. After a time "τ_{res}", "Q2" is resolved to its correct expected value "1", but "Q4" is resolved to an incorrect unexpected value "1". While waiting for the next clock cycle to re-sample the correct value, the output of the XOR gate *"I"* will be unexpectedly "0" [43].



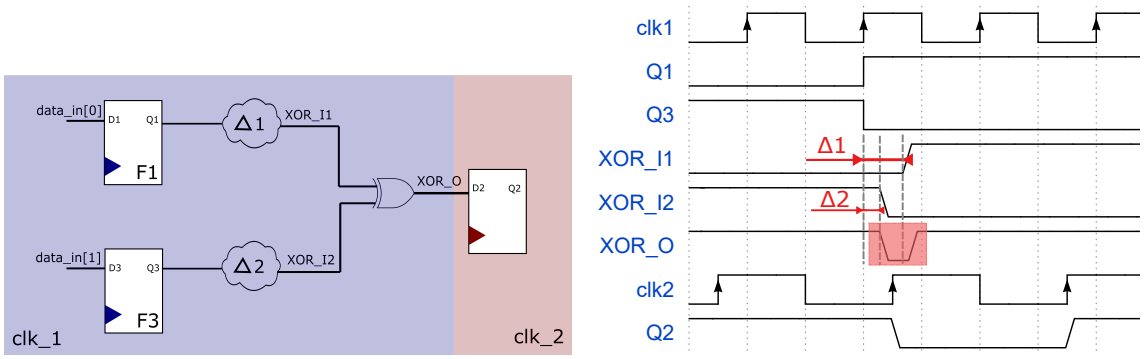
(a) Structure *(b) Waveform*

Figure 1.13: Incoherency due to CDC

Glitch

Glitches are very common in combinational logic. Due to the different propagation delays of their operands, some gates may experience a transient value (glitch) on their outputs. In synchronous paths, the STA ensures that this glitch is resolved within the clock period to prevent sampling it. In a CDC path, as the data can be captured on the destination side at any time, this glitch is more likely to be captured especially in the case of a *Slow-to-Fast* crossing.

In Figure 1.14a, the propagation delays on *"XOR_I1"* and *"XOR_I2"* are not the same, that is why a glitch appears on *"XOR_O"*. The rising edge of *"clk2"* happens in the window where the glitch is not yet resolved the glitchy value appears "Q2" for one complete clock cycle.



(a) Structure *(b) Waveform*

Figure 1.14: Captured glitch due to CDC

Dataloss

Dataloss is a common problem in both synchronous and asycnhronous systems. If the data is crossing from a fast clock domain to a slower one (Fast-to-Slow), it can change multiple times during a clock cycle of the destination clock. This data can be forever lost. In case of a CDC, this problem may also arise even on the properly re-synchronized paths

as most of the synchronization protocols are interested in stopping the propagation of the metastability and few of them gives a solution to the dataloss problem.

1.2.2 CDC synchronization structures

There exist several techniques to synchronize a CDC signal [44] [45] [46] [47]. In order to address the CDC problem and find a proper synchronization structure, the crossing signal type must be identified.

Multi-Flop Synchronizers

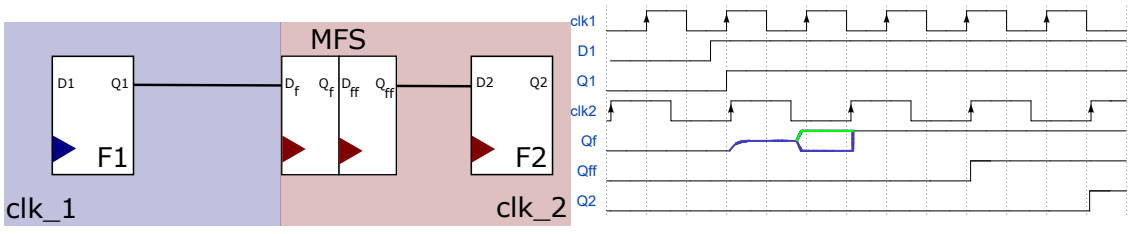
Scalar control signals are usually synchronized using a Multi-Flop Synchronizer (MFS). Multiple flip-flops (usually two or three) are cascaded together to delay a clock domain crossing signal for several clock cycles (see Figure 1.15a). A multi-flop synchronizer cannot prevent the appearance of the metastability; however, it decreases the probability of a metastability to reach the destination flip-flop. It gives more time to the metastability to be resolved before reaching the destination flip-flop [42].

The number of flip-flops in a MFS structure depends on the calculation of the Mean Time Between Failures (MTBF) [11]. In Figure 1.16, the probability of a metastability to enter a design is $p(enter) = \frac{t_w}{t_c} = t_w.F_c$. "$t_w$" being "setup-hold" time window and "t_c" being the period of the sampling clock. In Figure 1.15a, assuming that "Q1" is changing at a rate F_D , the probability of the metastability becomes $t_w.F_c.F_D$. If a latch is metastable at a time $t = 0$, the probability it will remain metastable at time $t > 0$ is $p(exit) = e^{-t/\tau}$. A failure is the fact a flip-flop becomes metastable after the clock's sampling edge, and that it is still metastable for a time s after that [39] [11] [48] [40] [49] [50].

$$p(failure) = \begin{array}{l} p(enter) \times p(exit) \\ (t_w \times F_c \times F_D) \times e^{-s/\tau} \end{array}$$

The inverse of the failure rate is the mean time between failure :

$$MTBF = \frac{e^{s/\tau}}{t_w \times F_c \times F_D}$$



(a) Structure *(b) Waveform*

Figure 1.15: CDC synchronized by MFS

If the concerned signal is a bus control signal, it is still possible to use a MFS on each bit but on a condition. As the metastability can resolve with different rates and to different and unexpected values on each bit, some coherency problems will appear if they re-converge later in the fan-out. Therefore, the Crossing bus must be gray coded or the different bits must be exclusive if each bit is separately synchronized by MFS.

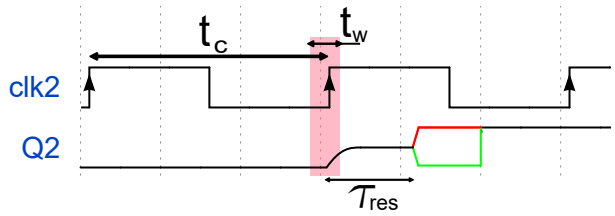
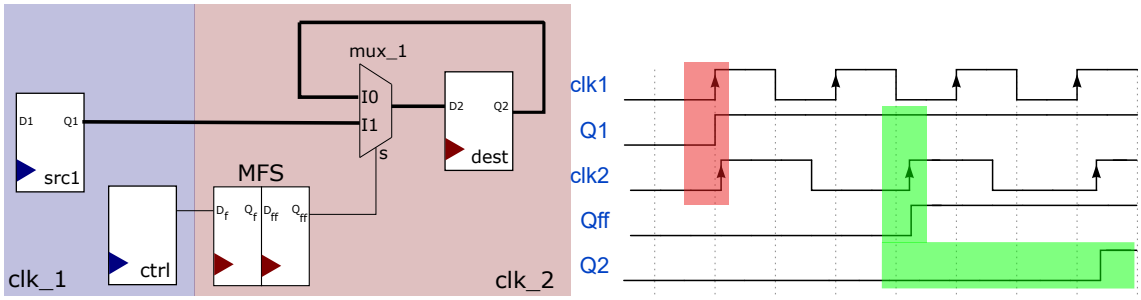


Figure 1.16: Metastability entering and resolution timings

Qualifier based Synchronizers

A qualifier is a control signal initiated in the source clock domain, synchronized by a MFS and blocking the coming data on a blocking logical gate [6] [39] [51]. The most frequent use of a simple qualifier is with a re-circulation mux. In Figure 1.17a, a data coming from “src1” clocked by clk1 is crossing the clock boundary to “dest” clocked by “clk2”. The re-circulation mux "mux_1" enables the new data when it is stable. Otherwise, it re-circulates the old data and blocks the new one. The selection signal “s” is initiated in the same clock domain of the crossing data and synchronized by a MFS driven by the same clock domain as the destination [35]. In Figure 1.17b, following the red rectangle, the rising edge of "clk1" and "clk2" happen almost at the same time while a new data is about to be written on "Q1". The control signal at "Qff", equals to zero, blocks the new data propagation and the old data is propagated. Then, following the green rectangle, once the data is stabilized on "Q1", the control signal at "Qff" enables the propagation of the new data.



(a) Structure *(b) Waveform*

Figure 1.17: CDC synchronized by Qualifier

Handshaking

The handshake is a four-phase protocol used to synchronize data signals. It ensures the data stability using a control signal “request” initiated in the source domain, and another control signal “acknowledge” initiated in the destination domain. In Figure 1.18, the request signal "req" is synchronized by "MFS2" and used with the acknowledge "ack" to calculate the enable signal of the destination to propagate the data. On the other hand, the acknowledge "ack" is synchronized by "MFS1" and used with the request "req" signal to calculate the enable of the source flip-flop to capture the data [52]. The 4-phases sequencing protocol is as follows :

- **Phase 1 :** If the data are transmitted by the source domain, request signal "req" is sent to the destination domain through a MFS.

- **Phase 2 :** Once "req" is received in the destination domain, the data read is enabled. Then, it sends the acknowledge "ack" that must also be synchronized by MFS.
- **Phase 3 :** The acknowledge "ack", being asserted, deactivates the request "req" and then data path is blocked.
- **Phase 4 :** Once "req" deactivated, "ack" becomes also deactivated and the protocol waits for a new request once a new data is available.

The handshake protocol is robust and ensures that no metastability can happen on the data path. Nevertheless, it is not adapted in the rapid applications as it deploys a remarkable delay to read the data.

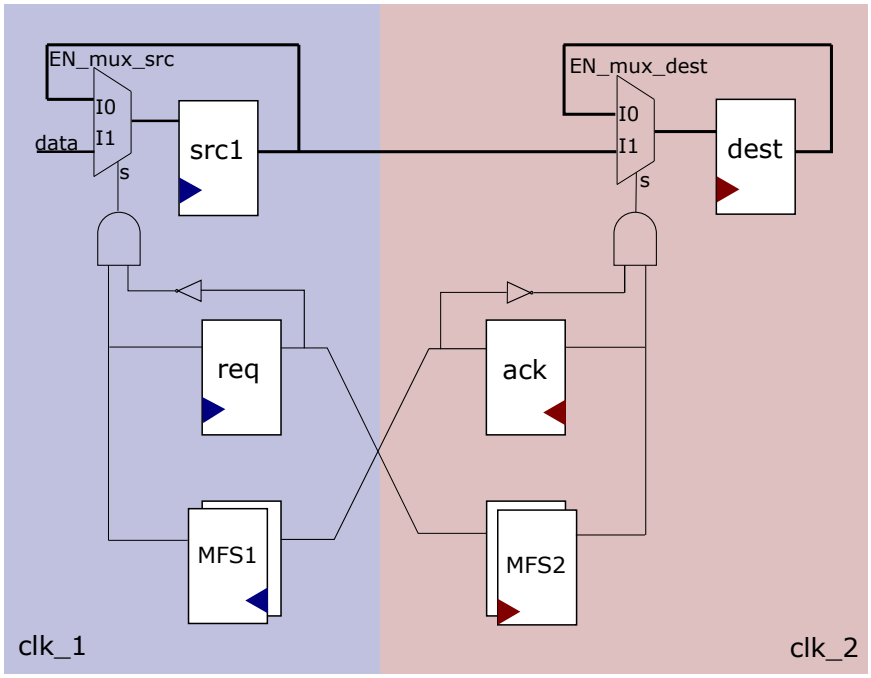


Figure 1.18: CDC synchronized by handshake protocol

Asynchronous FIFO

If the source clock is running faster than the destination clock, a special type of a first in first out memory may be used to synchronize multiple CDC data paths in order to avoid data loss and the probems related to *Fast-to-Slow* crossings. In Figure 1.19, the writing is controlled by the writing address "wrt_add" initiated in the source clock domain, while the reading is controlled by the reading address "rd_add" initiated in the destination clock domain. The read address is gray coded and synchronized by MFS driven by the source clock in order to be used alongside the write address to calculate the "full flag". On the other hand, the write address is gray coded and synchronized by MFS driven by the destination clock in order to be used alongside the read address to calculate the "empty flag". In other words, we can consider an asynchronous FIFO as a synchronization protocol/structure that contains four control signals: the read address and the empty flag to control the data reading, and the write address and the full flag to control the data writing [6] [28].

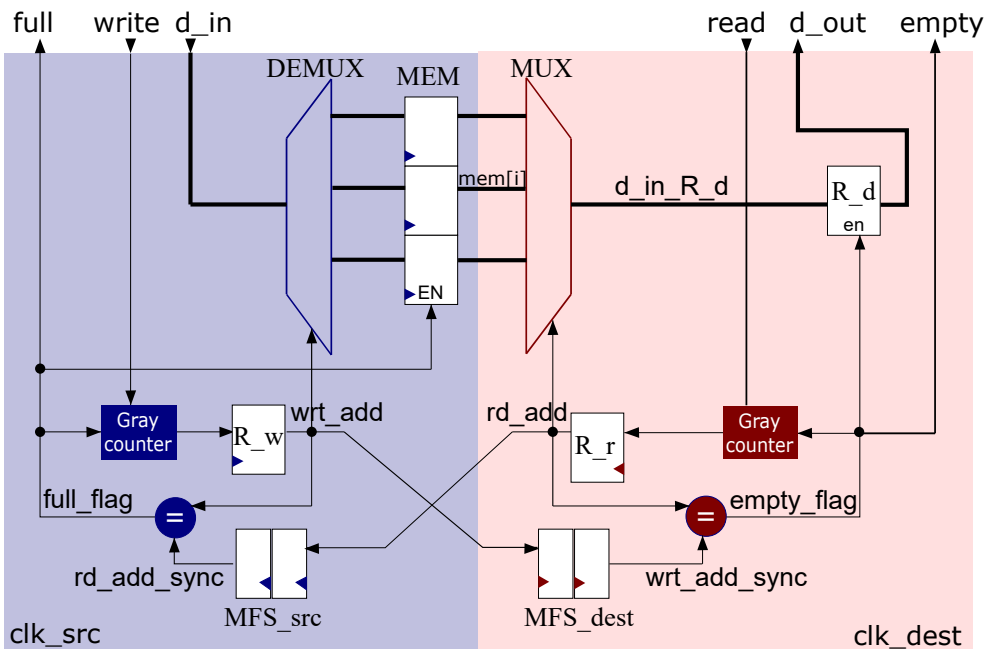


Figure 1.19: Asyncrhonous FIFO

1.3 CDC Verification

The multi-clock systems are the subject of many efforts and studies that aim to guarantee a safe communication between the different asynchronous blocks [53]. The CDC problems are addressed at an early stage of the project's life, starting by an effective design all the way down to the several exhaustive types of verification.

Among the design efforts, we found in [54] Andrei Ivanov proposes a new approach for data encoding to avoid incoherency problems due to the metastability resolution in multi-clock systems. As Gray code is known for allowing to transmit safely only adjacent state transitions, he proposed a coding method that allows to transmit safely any counter state jumps, limited by arbitrary predefined parameter.

Some other efforts were seen to make a CDC testable friendly designs. In [55], Naghmeh Karimi and Krishnendu Chakrabarty propose a methodology to locate CDC faults and to ensure post-silicon recovery. They have used a series of HSpice simulations to quantify the impact of process variations on timing closures. They have found high incidence on the setup hold timings violation even on the paths synchronized by MFS. Then, they proposed an approach to locate the faulty CDC by introducing some faulty patterns and another approach to recover the timing violations by integrating external delay blocks (buffers) to delay the clock signals by the amount of the setup timing violation. Their results show that the post-silicon CDC faults locating and tuning is effective. However, they marked a 15% area overhead which is not negligible. A similar idea was discussed in [56]. C. and Machado, P. and Bexiga, V. and Teixeira, J. P. and Teixeira, I. C. and Silva, J. C. and Lousã, P. and Varela, J proposed a different methodology to test and diagnose CDC post-silicon problems. They proposed embedding a CDC test and diagnosis structure in each locally synchronous domain using the local CDC Interface for accessing the communication channels. The latter are executed in the test scenarios.

The problems related to CDC are the subject of many types of verification. CDC are checked along the project's lifetime, from RTL to post-silicon. In [57], Shubhyant

Chaturvedi proposes a methodology to include the CDC analysis in the STA (being initially set to false path). The methodology is based on detecting all the CDC control paths, set them into a pair of sets having the same launch and capture clocks and to calculate the FIT (Failure In Time) to ensure that the depth of the instrumented MFS is sufficient. This approach can be efficient if the resolution time and the number of synchronization latches (participating in the calculation of the FIT) can be accurately and automatically calculated or detected. This approach is also valid for the CDC control path, however, it did not propose a similar solution for the CDC data paths. Instead, he proposed to place uniquely named buffers, CDC marker cells, at the output of the flop launching such data signals at the RTL stage itself. These CDC marker cells are then identified in downstream STA runs to exclude CDC data paths.

In this thesis, we focus on the CDC verification on RTL as a standalone flow independent of the STA. It aims to detect the CDC problems and structures early in the design flow, even before synthesis [58] [59].

1.3.1 CDC structural verification on RTL

The CDC structural verification is a static verification that aims to detect all the CDC paths, report whether properly synchronized or not, and finds other problems related to CDC such as re-convergences and glitches. The three most famous EDA providers propose some effective solutions to perform this verification in an optimized way. Regardless to some difference in performance and logic, the different EDA tools follow almost the same steps to perform the CDC structural verification on RTL as shown in Figure 1.24. [58] [60] [61] [26] [62].

RTL compilation and elaboration

The design comes in a Hardware Description Language (VHDL or Verilog). The compilation step is the one responsible for reading the code and finding syntax and semantics errors. Then, during elaboration, the design hierarchy is built, the parameters values are computed and the hierarchical names are resolved.

Setup generation

The EDA tools are able to extract the important setup information and to generate a set of constraints that guides the CDC verification later. Usually, the generated setup is not accurate and needs some attention from the verification engineer. The constraints generation and elaboration step is one of the longest steps of the flow and is very prone to human errors. Here are some examples of the primary constraint :

- **Clock signals:** all the clock signals must be properly defined as well as the different clock domains and the relationship between these domains. Each clock signal must be specified by a name and a source and must be associated to clock domain. All the clock signals belonging to the same clock domain are, by definition, synchronous. On the other hand, all the clocks belonging to different clock domains are, by default, asynchronous. The user should be able to customize the relationship between the different clock domains.

- **Reset signals:** all the reset signals must be properly defined and associated to their respective reset domains specifying their active level. A reset ordering can also be specified to avoid RDC problems.
- **Constant configuration signals:** they are the signals that must be set to a constant value in order to put the design in a specific mode (functional or test mode). Their job is to propagate or to block the the clock signals either on the clock multiplexers or the enable signals of the clock gates. In Figure 1.20, if the configuration signal is set to "0", "clk_1" will be propagated to "F2" and the data path will be synchronous. But, if the configuration signal is set to "1", "clk_2" will be propagated to "F2" and CDC path will appear between "F1" and "F2". The Configuration signals are a very important element of the verification setup as they are able to add or to mask a considerable number of CDC paths.

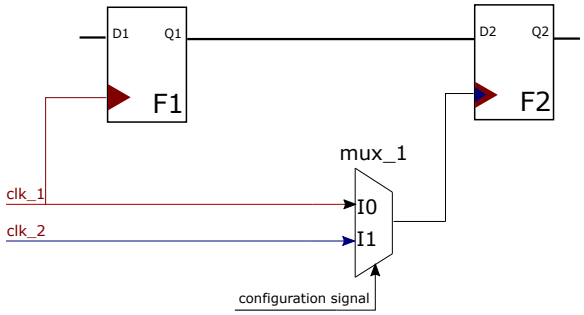


Figure 1.20: Constraining a configuration signal

- **Primary inputs/outputs:** the input/output ports must be defined by their name and clock domain. This step is very important to be able to expect the CDC paths that happen on the boundaries of the different blocks.
- **Black boxes:** they are the cells with an unknown logic functionality (e.g analog block). These cells are black boxed and only their inputs and outputs ports are defined. Each port should be defined in terms of its type, name and associated clock domain (see Figure 1.21).

Figure 1.21: Constraining a black box

The constraints set generated by the EDA tools and modified by the verification engineer are usually written in a scripting language or in SDC (Synopsys Design Constraint) format. They are considered as the hypothesis of the CDC structural verification and they restrict the behavior of the design in a specific way.

Setup verification

The setup verification aims to verify the constraints defined in the last step against some pre-coded rules to ensure their validity. The target is to ensure that the design is :

- **Not under-constrained:** the setup verification flags all the missing clocks and resets definitions in case a sequential element is driven by an undefined clock or reset signal. In addition, the setup verification flags an error in case of a unconstrained clock mux or clock gate. In this case, a clock overlap error is flagged and a composite clock, asynchronous to all the other clocks is propagated.
- **Not over-constrained:** the setup verification flags the contradictory multi-defined signals. For example, if the selection pin of a clock is constrained twice to two different values, an error is flagged and a composite clock, asynchronous to all the other clocks is propagated. In addition, the unpropagated defined clock and reset signals are also flagged.

A well defined setup is the key of a successful CDC verification. But, as the constraints are very prone to human errors, the number of violations usually is unacceptable. The number of violations can go to some hundreds of thousands for a medium size project. The debug of these violations takes usually several months. The setup generation step is revisited iteratively as long as there still are setup errors. Once the setup errors are all fixed, the design and the setup are ready for the structural verification. In [63], Andrew Cunningham and Ireneusz Sobanski propose an inter-tools methodology to reduce the noisy results. They proposed to reuse the data (especially the clock definitions and the configuration signals) coming from the physical implementation and the STA tools for the CDC verification. They have shown that this method remarkably reduced the CDC violations noise. The constraints re-usability may be a good idea to jump quickly to clean results but its usage is always at risk. The STA constraints main focus is to propagate the synchronous clocks in order to analyze a maximum of the synchronous paths, while the target of a CDC verification is to propagate the asynchronous clocks in order to verify a maximum of the asynchronous paths. Thus, the design configuration in both cases is largely different.

Structural verification

The structural verification is based on pre-coded patterns matching to detect the CDC models corresponding to the different synchronization schemes and CDC problems [64]. Once a CDC path is found, the different patterns are tried to be matched to find a proper synchronizer. If no synchronizer is found, the path is reported as unsynchronized and problematic. Here is some examples of what the structural verification is able to report as information:

- **Synchronized control paths :** if a MFS is found on a control signal crossing two defined clock domains, the path is reported as a well re-synchronized CDC path.
- **Synchronized data paths :** finding a synchronized data path is usually a challenge that faces all the EDA tools. Each of them has its own approach to compromise the difficulty of detecting a custom or a complicated sycnhronization protocol. Generally, the search is based on looking for a blocking gate, back-tracing its control input till the first MFS. If a similar structure is found, the data path is reported well or partially re-synchronize.
- **Ignored CDC paths :** Some CDC paths could be ignored due to specific constraining.

The CDC structural verification is also meant to find all the problems related to CDC [65], such as :

- **Unsynchronized control paths :** if no MFS is found on a control signal crossing two defined clock domains, the path is reported as an unsynchronized control CDC path.
- **Unsynchronized data paths :** if no pre-defined synchronization model is found on a data signal crossing two defined clock domains, the path is reported as an unsynchronized data CDC path.
- **Re-convergence :** The re-convergence of multiple signals re-synchronized each with a MFS is reported as problematic. In Figure 1.22, as the metastability resolution time and value on "MFS_1" and "MFS_2" are unknown and may not be similar, some coherency problems may appear as the synchronized signals re-converge later in the fanout.
- **Glitch :** the combinational logic inside a CDC path is prohibited. The data should be registered in the source domain before crossing the clock domain boundary. In synchronous systems, the STA can ensure that a glitch is resolved before the sampling clock edge. On the opposite side, in the case of a CDC path, no timing check can ensure that the glitch will not be captured. In Figure 1.23, if "clk_2" is faster than "clk_1", there will be a high risk to capture a glitch produced by the combinational logic inserted inside the CDC path. A combinational logic inserted inside a CDC path is considered a CDC design error that needs attention.

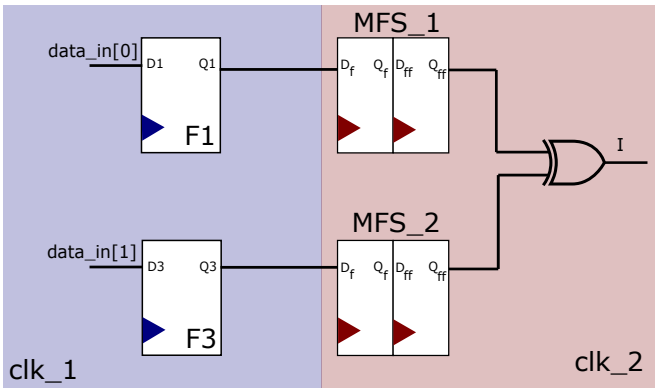


Figure 1.22: Problematic re-convergence

The detection of the different CDC data synchronizers was also discussed in [66]. M. Kebaili, K. Morin-Allory, J.C. Brignone, and D. Borrione explain one of the limitations of the static verification tools recognizing the custom data synchronizers. They propose an approach able to verify a greater variety of synchronizers. They defined a set of properties that only involve the enabling control signals, not the data, which reduces the cone of influence considered for formal verification. This reduced remarkably the execution time and prevents timeout and inconclusive results.

Secondary setup specification

Usually, some extra constraints are added progressively to fix some CDC errors based on the design specification. Here are some examples of the secondary constraints set :

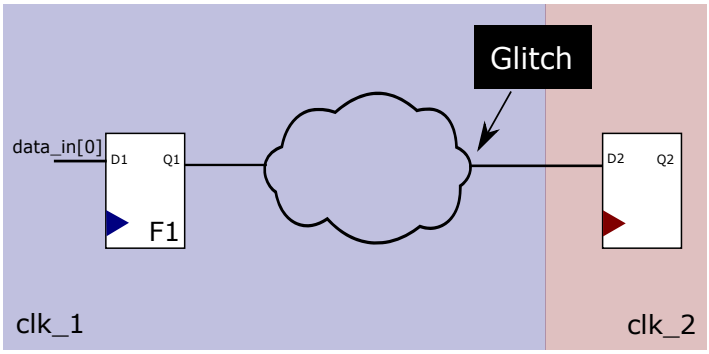


Figure 1.23: Prohibited comb logic in a CDC path

- **pseudo-static signals:** they are the signals that never toggle except if the destination clock is blocked. In this case, even if the signal is coming from an asynchronous clock domain, there is no risk a metastability can be generated. An unsynchronized CDC path can be just ignored if the crossing signal is constrained as pseudo-static.
- **Exclusive or Gray coded signals:** to resolve the re-convergence problems, the converging signals must be exclusive or gray coded. In Figure 1.22, if the bus "data_in" is gray coded, even if the metastability is resolved to an incorrect value, the data coherency will be guaranteed.
- **Custom synchronizer specification:** if the design includes a specific or a custom synchronization structure that does not match any of the CDC models pre-coded in the tools, a synchronizer can be explicitly specified. The EDA tools just skip any check on these structures.
- **False paths:** the EDA tools just skip any check on any path constrained as "False path".

The debug is a closed loop between the structural verification and the secondary constraints. The loop is then broken by either having no remaining CDC violations or by waiving the irrelevant or the tolerated errors. In [67], Makam Manikya Rakshith and Sujatha S Hiremath propose an effective pragmatic methodology for the CDC structural verification using VC-SpyGlass. They focused on the verification three stages (setup, integrity and structural) only on the functional mode. They have shown that their methodology remarkably reduced the number of false negatives in their CDC violations results. However, excluding the test mode from the analysis risks reducing the CDC coverage.

1.3.2 CDC assertions based verification

The detection of all the synchronized and the unsynchronized paths is an essential and relatively a non-costly task. Nowadays, due to time to market requirements, most of the projects' checklists require the structural verification and don't require an exhaustive functional verification for the asynchronous interfaces. However, just detecting the presence of the synchronization schemes doesn't guarantee that they are really functional. The structural verification is crucial and must be done very rigorously, but, still not enough.

The EDA tools propose a formal verification flow for all the synchronization protocols detected by the structural verification. Every synchronizer is associated to a group of

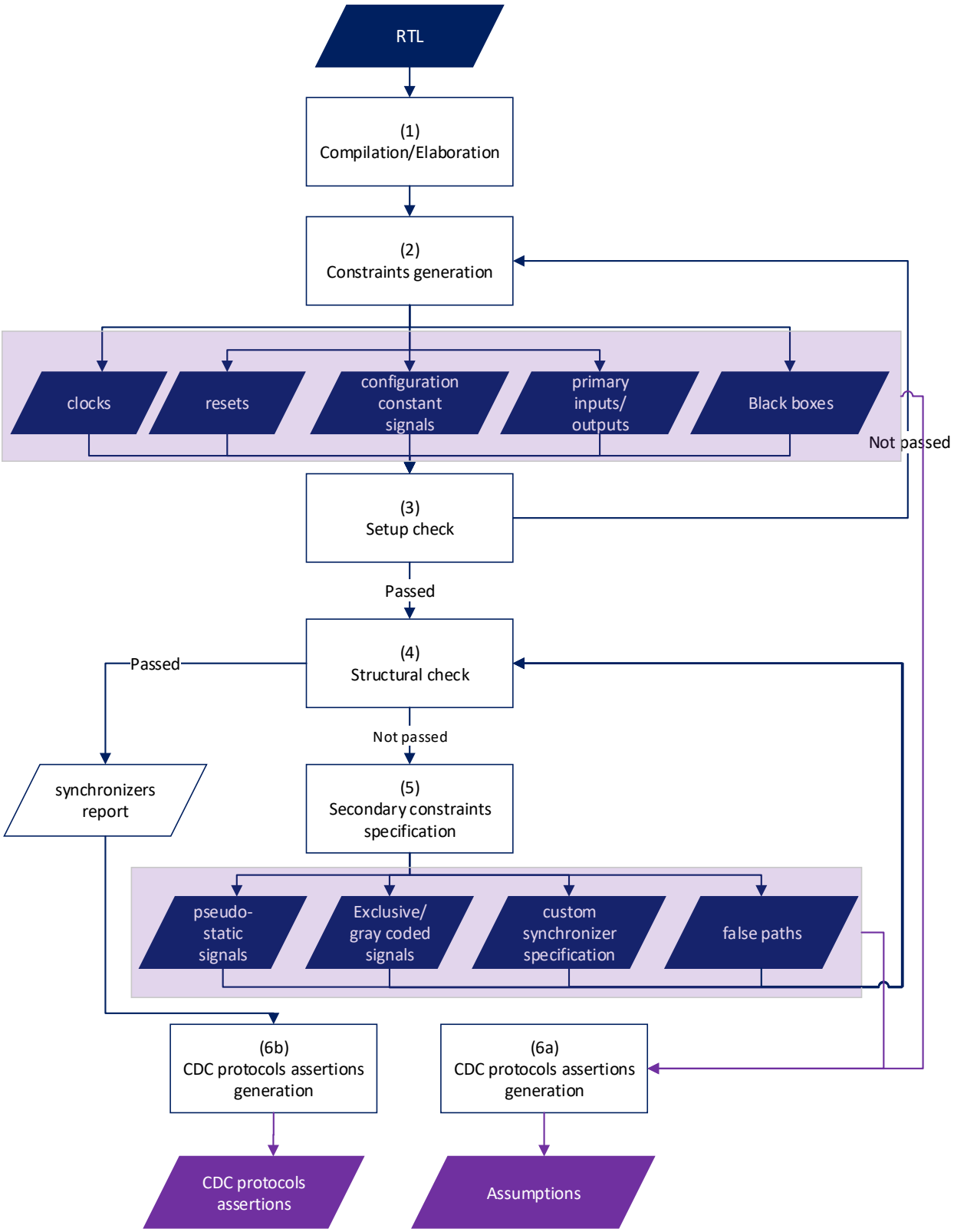


Figure 1.24: CDC Structural verification flow

properties that tell if it is functioning correctly or not. These properties are intended to be verified in a formal way using model checking, constrained by a set of assumptions equivalent to the set of constraints used for the structural verification.

Signal stability

The data stability is the key to ensure that a CDC is safe. All the synchronization protocols aim to guarantee the stability of the data when sampled in the destination domain.

Signals synchronized by a multi-flop synchronizer: if the destination clock is slower than the source clock, the data can be lost. So, the data should be stable long enough for the destination clock to be correctly sampled. In Figure 1.15a, if "clk_1" is faster than "clk_2", "D1" should remain stable for nb_of_cycles

$$nb\_of\_cycles = \frac{T_{clk2}}{T_{clk1}} + 1$$

```
mfs_stable : assert property
(@(posedge clk_2) disable iff (rst)
$changed (D1) |=> $stable(D1)[*nb_of_cycles]);
```

Signals synchronized by a qualifier based synchronizer: the qualifier should enable the data only when stable. This implies the qualifier to be disabled every time the data change value to ensure blocking the propagation of any metastable state. So, in Figure 1.17a, "Q1" should not change if "Qff" is enabling. This is described by the following SVA :

```
dataqualifier_stable : assert property
(@(posedge clk_2) disable iff (rst)
ctrl |=> $stable(Q1));
```

In addition, the qualifier itself should remain stable for a certain number of clock cycles being a signal synchronized by MFS to avoid dataloss. The number of clock cycles nb_of_cycles within which the qualifier should remain stable is calculated in function of the frequencies of the source and the destiantion clocks. This is described by the following SVA :

```
ctrlqualifier : assert property
(@(posedge clk_2) disable iff (rst)
$changed (ctrl) |=> $stable(ctrl)[*nb_of_cycles]);
```

Signals re-converging after synchronization: in Figure 1.22, the different bits of the bus "data_in[0:n]" are synchronized separately by MFS and then re-converged on combinational logic. To ensure the coherency of the re-converging bits, the exclusivity of the

different bits must be ensured. Only one bit or none of the bits should change at once. This is described by the following SVA :

```
Exclusivity : assert property
(@(posedge clk_1) disable iff (rst)
$onehot0($past(data_in)^data_in));
```

Glitch due to combinational logic inside a CDC: In a CDC path, as the data can be captured on the destination side at any time, glitches are more likely to be captured especially in the case of a *Slow-to-Fast* crossing. The number of times the data should toggle within a clock cycle should not exceed "1". This is described by the following SVA :

```
always @(data) begin
toggle_count = toggle_count + 1;
end
always @(posedge clk_2)
toggle_count<=0;
end
glitch : assert (toggle_count<2)
```

In [68], Mohammad Kasim, Vrinda Gupta and Mohandas Jebin propose a structural-formal flow to detect glitches on clock, reset and CDC paths. They generate SVA for the paths that violate the CDC glitch static rule. The SVA is then verified by a formal tool to confirm or to waive the violation. In [69], Ghaith Tarawneh, Andrey Mokhov and Alex Yakovlev propose a new methodology for the CDC formal verification that models metastability propagation. Their approach relies on substituting all the flip-flops that exist on a CDC path by a new flip-flop model taking into account metastable and setup/hold violation special outputs. The combinational logic inside the CDC is then doubled to test whether it is prone to metastability propagation or not. A year later, Ghaith Tarawneh and Andrey Mokhov published a new paper [70] where they explained their model checker tool Xprova which was based on the CDC Formal verification using metastability modeling. They have shown two concrete examples of a metastability propagation and a potential glitch discovered verifying a test case using their new tool. We concluded that the authors in both papers were trying to imitate as close as possible the metastability propagation.

In the state of the art, we also found a lot of effort done in order to enhance the CDC formal verification performance and make it more applicable in industry. In [43], [71] and [72], the authors propose a "Counter-Example Guided Abstraction refinement (CEGAR) where the user influences the algorithm based on information extracted from intermediate abstract counterexamples. It is a semi-automatic verification process where the user aids the verification process by classifying a sequence of automatically inferred constraints. Despite of the promising results shown in the papers, this approach is still not implemented in any verification tool and the time the iterations have taken on their test case is still an open question that was not developed in the papers. In [73], [51] and [74], the authors propose, what they called a "Meta-Model", enabling the extraction of a greater

variety of synchronizers, and their associated set of properties which limit the design area to be model checked and avoid state space explosion. The authors analyzing the root cause of state space explosion, concluded that the properties concerning data signals are a main cause for inconclusive results. As in CDC context the value of the data does not matter and the toggling time is all that matters, the authors proposed a set of properties relying only on control signals and that do not induce any check on data signals. In [75], the authors propose a hybrid flow leveraging the formal verification as a first step for verifying the CDC properties and passing only the remaining assertions, which could not converge, in simulation. This approach, despite of being a good compromise, cannot be always accepted in industry due to the unknown period the formal analysis can take to conclude or not on large designs.

1.4 Conclusion

Synchronous circuits are widely produced nowadays by the industry and their design and verification flows have achieved a high level of maturity. However, building an extensive SoC infrastructure while avoiding the problems of distributing a global, low-skew clock is not possible. To meet time-to-market requirements and to handle the evolution of the design complexity, independently clocked and powered IPs are integrated together. This alternative, also known as Globally Asynchronous Locally Synchronous (GALS) systems, prevents from designing large clock-trees but suffers from the loss of a global synchronization. Therfore, data travelling from a block to another, from a clock domain to another, is the major problem of GALS systems, what we call CDC (Clock Domain Crossing). The multi-clock systems are the subject of several exhaustive types of verification. The target is to ensure that asynchronous interfaces are well designed and function correctly. The state of the art has shown that many efforts were done in the context of the CDC design, verification and testing. However, we concluded that the CDC structural verification flow needs to be rationalized and standardized, avoiding the deficiencies exiting in the native flow. In addition, some new verification aspects, such as the semi-formal verification and the metastability injection, will be the focus of this thesis.

2

CDC structural verification

In this chapter, we present the improvements we developed in order to optimize and to standardize the CDC structural verification flow. In addition, some new fields were visited, studied and included to the reference flow such as the inclusion of the low power logic to the RTL being verified. By the end of this chapter, we also present a comparative analysis in order to evaluate the most important EDA tools based on our newly developed flow and the priorities of the projects.

Contents

2.1 Introduction

For synchronous circuits, the STA (Static Timing Analysis) is an essential step to ensure that the timing closure is respected. However, in multi-clocks circuits, the STA can not ensure the correctness of a clock domain crossing. A signal crossing a clock domain may violate the setup/hold timings of the sampling clock in the destination domain, provoking a metastability. The propagation of metastability, data loss and data corruption or incoherency are the major consequences of an improperly synchronized CDC [76]. In mixed signals systems, where analog and digital components co-exist, clock domains differ between the analog and the digital sections and an inaccurate analog-to-digital conversion may happen due to an improper synchronization between the digital and the analog domains. The multi-cores processors can also be an example of the systems vulnerable to CDC problems. When each core is clocked by its own clock, an inefficient handling of CDC in inter-core communication can lead to data incoherency impacting the parallel processing capabilities. Generally speaking, if a metastability is generated in a system and propagated through its combinational logic, the system is considered *dead* [77].

The CDC structural verification is the first safeguard against the hazards of the asynchronous multi-clock systems. The static approach aims at detecting all the CDC paths existing in a design and finding whether they are properly synchronized or not. The synchronizers detection relies on synchronizers pre-coded patterns. The static tools try to match these pre-coded patterns on each detected CDC path. If a synchronizer pattern is matched, the CDC path is reported as properly synchronized. Otherwise, the CDC path is reported as problematic. The structural verification cannot be done without specifying a number of constraints, considered as the description of the critical signals and structures of the design. In other words, the constraints files are the hypothesis of the structural verification and have a huge impact on its results. In this chapter, we focus on enhancing the CDC structural verification flow, so we can overcome the limitations of the existing tools and flows and on the other hand optimizing the verification results and the time the verification takes.

2.1.1 Design modeling

In order to understand how the design under verification is modeled by the static tools, we can assume that the overall structural design model is defined using a directed graph with labeled vertices $G = (V, E, label)$, where:

- V is the set of vertices representing all the elements of the structural design
- $E \subseteq V \times V$ is the set of edges
- Label: is the labeling function, such that

$$\text{Label} : V \rightarrow L$$
$$\text{with } L = \{L_{in}, L_{out}, L_{zero}, L_{one}, L_{not}, L_{and}, L_{or}, L_{seq}\}$$

  where L_{in} labels a primary input, L_{out} labels a primary output, L_{zero} labels a constant zero, L_{one} labels a constant one, L_{not} labels the output of a NOT gate, L_{and} labels the output of an AND gate, L_{or} labels the output of an OR gate and L_{seq} labels the output of a sequential element.

A structural path $Path(v_1, v_2)$ is a sequence of nodes representing the combinational logic between v_1 to v_2.

$$Path(v_1, v_2) = (v^i)_{i \in 1...n} \mid v^1 = v_1, v^n = v_2, \forall 1 \leq i \leq n, (v^i, v^{i+1}) \in E$$

The CDC paths are defined using a directed sub-graph of G : $G_{cdc} = \langle V_{seq}, E_{cdc}, clock \rangle$, where :

- $V_{seq} \subseteq V$ and $V_{seq} = \{v \in V \mid \text{Label}(v) = L_{seq}\}$ is the set of vertices representing all the sequential elements of the design,
- Clock : is the clock domain labeling function, such that

$$\text{Clock} : V_{seq} \rightarrow C$$
$$\text{with } C = \{clk_0, clk_1, \ldots, clk_n\}$$

- $E_{cdc} \subseteq V_{seq} \times V_{seq}$ is the set of edges representing the existant connection between two different sequential elements existing in two different clock domains (a CDC), such that :

$$\forall v_1 \in V, \forall v_2 \in V, E_{cdc} = \{(v_1, v_2) \mid \text{Clock}(v_1) \neq \text{Clock}(v_2), Path(v_1, v_2) \neq \emptyset\}$$

2.1.2 Verification modeling

The CDC verification *CDCv* is the process of finding all the CDC paths G_{cdc} and the violated CDC properties Π^*_{cdc}, by scanning the design G, and being guided by the design specification Ds, the design rules Π and the CDC properties Π_{cdc}.

$$CDCv : G \times Ds \times \Pi \times \Pi_{cdc} \rightarrow G_{cdc} \times \Pi^*_{cdc}$$

Assuming that $\text{A} = \{\alpha_1, \ldots, \alpha_n\}$ is the set of signals of the design, the design specification Ds is the set pairs where each signal of A is associated to a property p.

$$Ds = \{\alpha \times \{p^i\}_{i \in 1...n} \mid \alpha \in \text{A}, p^i \in \{p_{clk}, p_{rst}, p_{in}, p_{out}, p_{analog}, p_{ps}, p_{ex}, p_{c\_sync}\}\}$$

where p_{clk} is the property describing a clock signal, p_{rst} is the property describing a reset signal, p_{in} is the property describing a primary input, p_{out} is the property describing a primary output, p_{analog} is the property describing an analog signal, p_{ps} is the property describing a pseudo-static signal, p_{ex} is the property describing exclusive signals and p_{c_sync} is the property describing the CDC signals synchronized by custom synchronizers. The set of rules Π is the set of rules that the design specification Ds should respect in order to ensure the design integrity prior to the CDC structural verification. Some of the rules are syntactically checked, such as the rules related the missing signals declarations. Some other rules are semantically checked, such as the check for a potential glitch due to an asynchronous clock enable signal. Some insights on the checked rules are given in Annex A. The set of rules $\Pi_{cdc} = \{\pi^i\}$ is the set of CDC properties that should be respected by the different CDC synchronizers, such as gray coding for data bus synchronized by MFS. Some insights on the set of CDC design rules are given in Annex B.

The CDC structural verification *CDCv* is the composition of three main functions:

1. The Setup Specification *SS*: where the design specification *Ds* of design *G* is translated into a set of constraints κ understandable by the tools.

$$SS : G \times Ds \rightarrow \kappa$$

The set of constraints κ is the union of two subsets of constraints $\kappa = P \cup S$, where:

- Primary constraints *P*: is the set of constraints for the driving signals such as clocks *C*, resets *R*, configuration signals *N*, black boxes *B*, primary inputs *I* and primary outputs *O*.

$$\begin{aligned} & P = C \cup R \cup N \cup B \cup I \cup O \\ \text{with} \quad & C = \{clk_0, clk_1, \ldots, clk_n\} \\ & R = \{rst_0, rst_1, \ldots, rst_m\} \\ & N = \{const_0, const_1, \ldots, const_o\} \\ & B = \{bbox_0, bbox_1, \ldots, bbox_p\} \\ & I = \{in_0, in_1, \ldots, in_q\} \\ & O = \{out_0, out_1, \ldots, out_r\} \end{aligned}$$

- Secondary constraints *S*: is the set of constraints for data signals involved in a CDC path E_{cdc} and that have a specific behaviour (*e.g.* pseudo-static, exclusive, *etc.*).

$$S = Ps \cup Ex \cup Cs$$

where "*Ps*" is the set of pseudo-static signals, "*Ex*" is the set of exclusive signals and "*Cs*"is the set of CDC signals synchronized by custom synchronizers.

2. The Setup Check *SC*: where the set of constraints κ is checked against the design rules $\Pi = \{\pi^i\}$. It returns Π^* the set of violated rules and a new set of modified constraints κ^*.

$$SC : G \times \Pi \times \kappa \rightarrow \kappa^*, \Pi^*$$

3. The Structural verification *SV*: where the design *G* is scanned with the guidance of the modified set of constraints κ^*, inheriting the violated setup rules Π^* if not corrected, and checked against the CDC design rules Π_{cdc}. The target is to find all the CDC paths G_{cdc} and the set of violated CDC rules Π^*_{cdc}. Some insights on the set of CDC design rules are given in Annex B.

$$SV : G \times \Pi_{cdc} \times \kappa^* \times \Pi^* \rightarrow G_{cdc}, \Pi^*_{cdc}$$

To conclude, as shown in Figure 2.1, the CDC structural verification *CDCv* is the composition of three functions as follows:

$$\begin{aligned} CDCv : G \times Ds \times \Pi \times \Pi_{cdc} \rightarrow & \quad G_{cdc}, \Pi^*_{cdc} \\ g, d_s, \pi, \pi_{cdc} \mapsto & \quad SV(g, \pi_{cdc}, SC(g, \pi, SS(g, d_s)) \end{aligned}$$

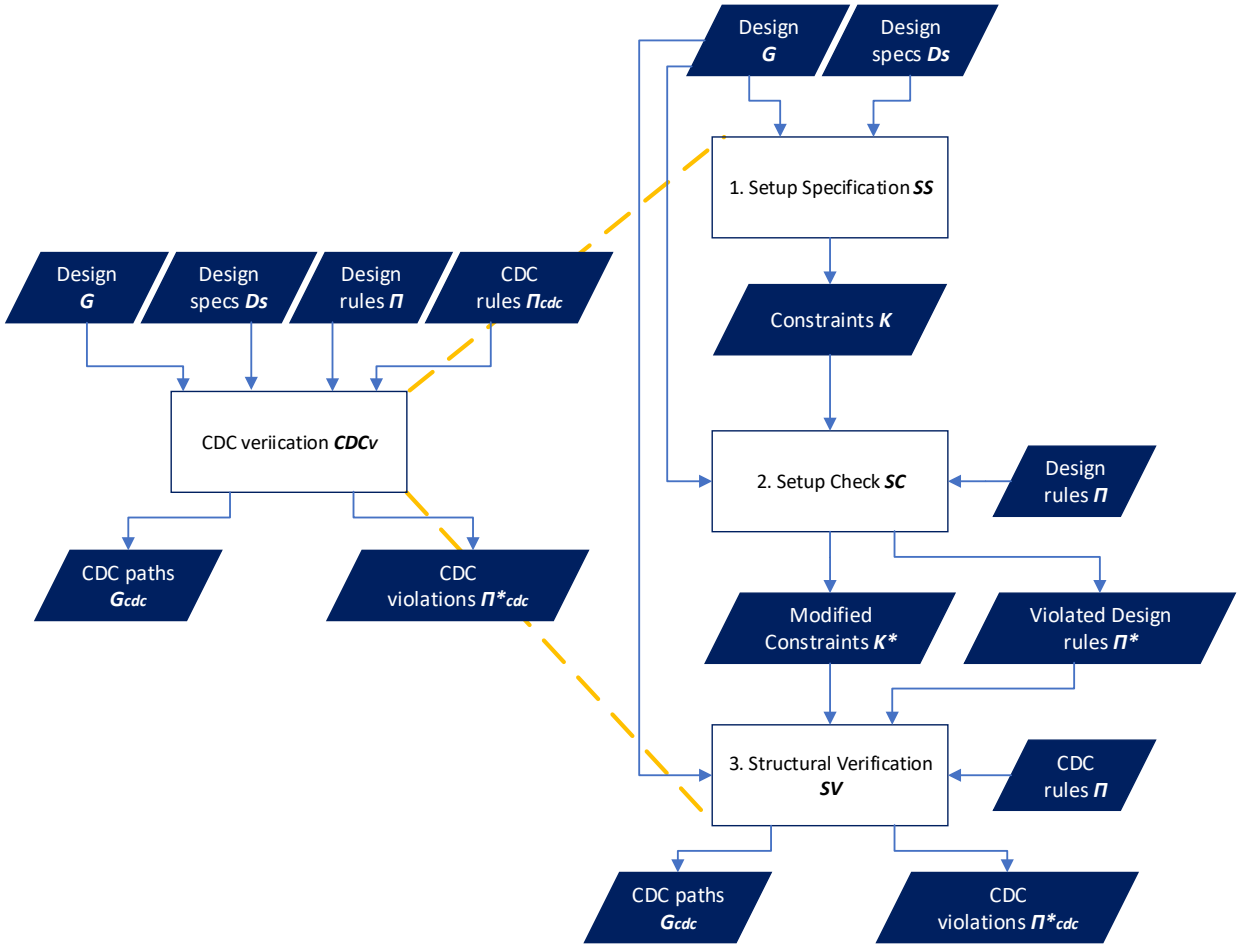


Figure 2.1: CDC structural verification process

2.2 Flow rationalization

The CDC structural verification is the first approach adopted by the industry nowadays. Despite the fact that this technology is widely used and continuously improved, the designers still suffer from the quality of its results. As seen in Chapter 1, the flow depends to a great extent on human efforts, which makes it very prone to human errors. Optimizing and defining a reference CDC structural verification flow was the early priority of this thesis.

2.2.1 Deficiencies of the classic flow

In order to optimize the flow, a study was made to locate the problems of the actual classic CDC verification flow used by the CDC verification team at STMicroelectronics. The major problem that appears with every new project was the required time for the CDC static verification task. It could take weeks or even months to converge the number of violations of the setup check and structural verification steps. Most of the violations are either false negatives due to the inability of the tool to analyze a complex CDC, or redundant violations that have the same root cause that was inherited by multiple rules [78]. The located deficiencies of the classic flow can be summarized in the following points.

Constraints inter-dependencies unrespected

There exists an inter-dependency between the different primary constraints P. As shown in Figure 2.2, some constraints depend and are defined in terms of other constraints. The only constraints that are completely independent are the clock signals C and the resets signals R. The process of defining all the other primary constraints, such as configuration signals N, the black boxes B and the primary inputs I and outputs O, depends on the clocks and resets and sometimes on other constraints. For example, the primary ports should be associated to their correspondent clock domains in order to detect the potential CDC paths by imitating the outside conditions (clock domains). This will force the tool to look for a synchronizer directly connected to the input port. Notice, in the following pseudo-code, how the defined clock is used to define the primary input.

```
# define clock
define_clock -name [name] -source [source] -period [period]
-factor [mult./divide factor]

#define clock domain
define_clock_domain -name [name] -clocks [source]

#define input
define_input -name [input name] -port_clock [port clock
domain] -delay [associated delay]
```

Another example is the black boxes constraints. Each black box input/output is associated to a propagated clock signal in order to detect the need of a synchronizer between a source and the input of a black box, or the output of a black box and a destination. In

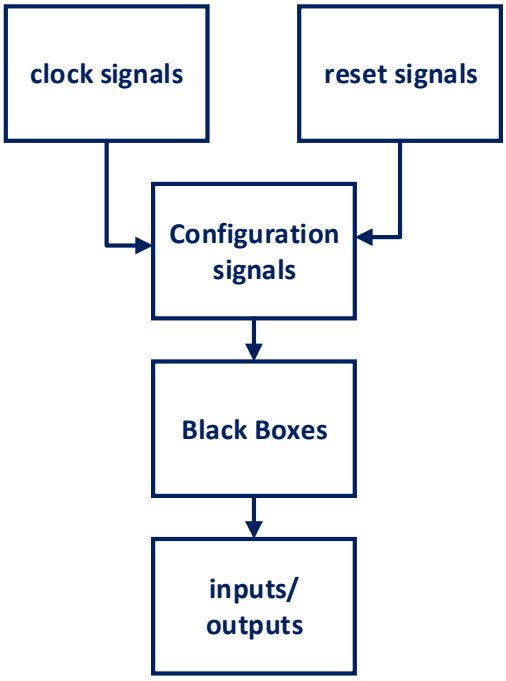


Figure 2.2: Constraints inter-dependencies

Figure 2.3, the set of constraints should be specified as follows to respect the interdependencies between the black boxes constraints and the other constraints :

```
# define clock_1 and its clock domain (1)
define_clock -name clk_1 -source clk_1 -period x
define_clock_domain -name clk_1 -clocks clk_1

# define clock_2 and its clock domain (2)
define_clock -name clk_2 -source clk_2 -period y
define_clock_domain -name clk_2 -clocks clk_2

#define configuration signal to propagate clk_1
define_constant  -signal select -value 0

#define the black box inputs attributes (4)
define_bbox -data_input data_in -data_input_clock clk_1
```

As defining the black box input pin depends on the declaration and the propagation of either clk_1 or clk_2, an error can be easily inherited if the clock signals are not well defined or not correctly propagated (which can be the result of not constraining the select signal of a clock propagation element).

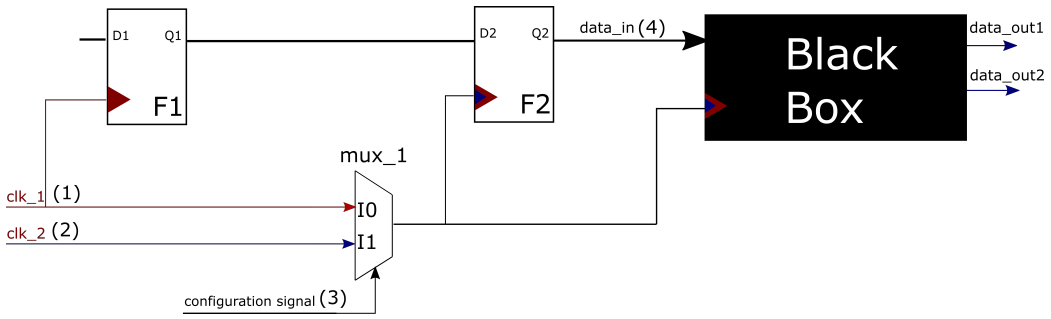


Figure 2.3: Black Box inter-dependent constraints

The native *SS* function of the industrial tools does not respect these interdependencies and generates all the constraints simultaneously. To give a concrete example on that, let us assume that the design specification of the design in Figure 2.4 indicates that "clk_1" and "clk_4" belong to a same clock domain, while "clk_2" and "clk_3" belong to another. If we apply the native SSfunction of the tool, it will generate all the constraints templates one shot, as shown in the following pseudo-code :

```
# define clock signals and their clock domains
define_clock -name clk_1 -source clk_1 -period x1
define_clock -name clk_2 -source clk_2 -period x2
define_clock -name clk_3 -source clk_3 -period x3
define_clock -name clk_4 -source clk_4 -period x4

define_clock_domain -name clk_1 -clocks clk_1
define_clock_domain -name clk_2 -clocks clk_2
define_clock_domain -name clk_3 -clocks clk_3
define_clock_domain -name clk_4 -clocks clk_4

# define configuration
define_constant  -signal sel -value 0/1

#define primary input
define_input -name In_1 -port_clock
{clk_1, clk_2, clk_3, clk_4}

#define black box attributes
define_bbox -data_input bb_in -data_input_clock
{clk_1, clk_2, clk_3,clk_4} -data_output bb_out1 bb_out2
-data_output_clock {clk_1, clk_2,clk_3, clk_4}

#define primary output
define_output -name Out_1 -port_clock
{clk_1, clk_2, clk_3, clk_4}
define_output -name Out_2 -port_clock
{clk_3, clk_4}
```

The previous native constraints template generated by the tool has considered that all four clock signals are asynchronous (as they were declared each in a separate clock

domain). In addition, the select signal "sel" was not properly constrained to a specific value. This will have a huge impact on the violations count and its effect can be seen in different aspects. First, a clock overlap violation will be flagged on the outputs of "mux_1" and "mux_2" and two new composite asynchronous clocks (a mix of their input clocks) will be propagated to the black box and "F2". Consequently, the input of the black box "bb_in" will be automatically associated to these composite clocks, and the same is applied to the two outputs "bb_out1" and "bb_out2". This will be inherited by the design primary output "Out_1" as it will also be automatically associated to the composite clock. Finally, as "bb_out2" is associated to the composite clock, a false CDC will be flagged between "bb_out2" and "F2". We can conclude that one error in the declaration and the propagation of the clock signals was inherited by the black box and all the downstream logic. This single error was able to multiply one simple violation by two or three. All these problems could have been avoided if the constraints were declared respecting the interdependencies and the problems were corrected gradually.

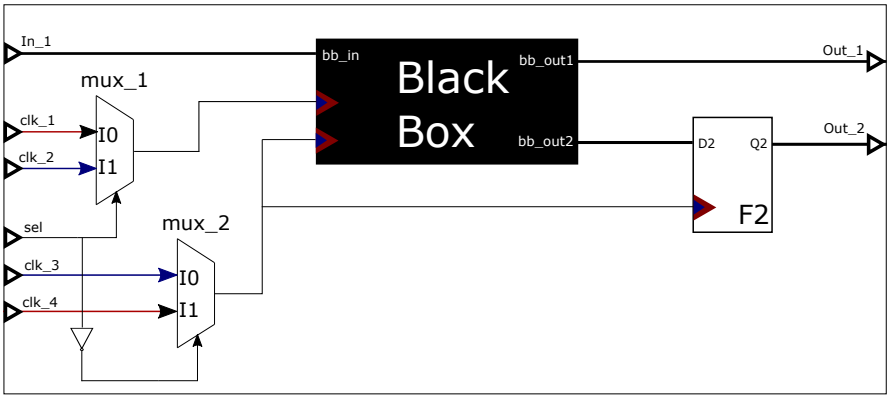


Figure 2.4: Black Box inherited violations

For this reason, the CDC structural verification always suffers from the huge number of violations to be analyzed. The CDC task for an average size design can take months just to analyze hundreds of thousands of violations. From our experience with the different projects, about 80% of the reported violations are always just false negatives. This happens due to the one shot constraint generation.

Global synchronization parameters

The tool analysis is guided by some parameters that the user configures based on his design specification. The synchronization parameter is one of the configuration parameters, responsible for specifying the minimum MFS (Multi-Flop Synchronizer) depth the tool should accept to validate a CDC synchronizer (usually between 2 and 4). The synchronization parameter relies on the MTBF calculation. Sometimes, just one global parameter is not enough to reflect the reality of a design as the MTBF varies from one CDC path to another based on the frequency of the source and destination clocks. Some paths may require a deeper synchronizer than others. Specifying the minimal depth should have been acceptable as a compromised solution except that it provokes a remarkable reporting problem for the deeper synchronizers. The extra flip-flop is just ignored and false violations (such as sequential convergences) may appear.

Exclusive, unrelated or gray coded signals detection

About 50% of the violations at structural checks stage is due to re-convergences. The resolution time and value on each bit being unpredictable, may cause some coherency problems if the signals are converging in their fanout. This type of violations can be easily eliminated by constraining the converging signals as exclusive, unrelated or gray coded. The analysis of thousands of re-converging signals and adding these constraints one by one is a huge work that requires a huge amount of time. The tools today do not auto-generate constraints candidates for exclusive, unrelated or gray coded signals.

The previous deficiencies accumulated degrade the quality of the results of the CDC structural verification despite being a very expensive task in terms of time and effort.

2.2.2 A new reference flow

In the new reference verification flow, the target was to find solutions for the deficiencies mentioned above and to implement an automated robust flow (a set of scripts under a kit) that does not require a lot of knowledge from its user. The ease of use, the portability, noise reduction and the quality of results were the main objectives of this work.

Primary constraints hierarchical prioritization

The first step to reduce the results noise and the huge number of false negatives was the hierarchical prioritization of the specification of the different constraints. As shown in Figure 2.2, some constraints depend or re-use others. The configuration signals usually configure clock and reset multiplexers selection pins, clock gates enable signals or even the primary ports driving a particular functional or test mode. The propagated reset and clock signals are then associated to black boxes' data pins and the design's primary input and output ports. We can set a general rule that states that constraints should be specified starting by the physical fanin to the physical fanout of any design. The well defined constraints on the fanin should not cause a problem if propagated or reused in the fanout. For that we propose the new Primary Setup Specification function to generate only the primary constraints in a pragmatic way that respects the interdependencies. "PSS_{new}" is the composition of a number of sub-functions.

$$PSS_{new} : G \times Ds \ \rightarrow \ P$$

$$ConstraintClkRst : G \times Ds \rightarrow C \times R \times Ds$$
$$ConstraintConfig : G \times Ds \times C \times R \rightarrow Ds \times C \times R \times N$$
$$ConstraintBBox : G \times Ds \times C \times R \times N \rightarrow Ds \times C \times R \times N \times B$$
$$ConstraintInOut : G \times Ds \times C \times R \times N \times B \rightarrow Ds \times C \times R \times N \times B \times I \times O$$
$$P = C \bigcup R \bigcup N \bigcup B \bigcup I \bigcup O$$

The function "*ConstraintClkRst*" is where the clock and the reset signals and domains are extracted from the design specification and translated into a number of definitions (constraints C and R) understandable by the tools.

As the configuration signals "N" are the constant signals responsible for propagating the clock and the reset signals, "$ConstraintConfig$", where the configuration signals are defined, should be executed after "*ConstraintClkRst*".

The "*ConstraintBBox*" function defines the peripherals of black boxed logic (hidden from the CDC analysis), such as the analog parts of a design. These peripherals are associated to the different propagated clocks and resets by the constant configuration signals. That explains the input parameters of "*ConstraintBBox*" to be "Ds, C, R, N".

Finally, as the different primary inputs "*I*" should also be associated to clock domains, and as the different primary outputs "*O*" are affected by all the propagated signals, "*I*" and "*O*" should be defined at the end of the process using all the previously generated constraints as shown in function "*ConstraintInOut*".

Secondary constraints auto-generation

We separated the specification of the secondary constraints in a new function *SSS*. This new function aims at explicitly constraining the synchronization cells in terms of their source and destination flip-flops and to find easily exclusive, gray coded and unrelated signals. The industrial tools nowadays does not have the capability to generate these constraints automatically. However, with some workaround, they could be able to. If the static verification tools are already able to detect CDC paths and re-convergences, they may be able to propose the constraints to eliminate them by being re-fed the results they provide. As the structural verification function *SV* returns the set of violated rules Π^*_{cdc} and the CDC paths G_{cdc}. *SSS* takes these latter as inputs and returns a set of synchronizers *Cs* and the set of exclusive signals *Ex* able to empty Π^*_{cdc}.

$$SSS : G_{cdc} \times \Pi^*_{cdc} \rightarrow Cs \times Ex$$

Including *SSS* in the verification flow, we were able to auto-generate constraints templates to specify the CDC synchronizers and exclusive CDC signals.

- **Automatic generation of the synchronization cell constraints**: the idea was to launch a first structural verification *SV* configuring the global synchronization parameter to a very large number (larger than four). This parameter specifies the minimum synchronizer depth the tool should accept. Doing so and launching *SV*, the tool reports all the CDC paths as unsynchronized (*SV* returns a Π^*_{cdc} full of violated rules). These reports can be used or re-fed to *SSS* so that it proposes the adequate constraints to eliminate these violations and returns. The adequate constraints in this case are the synchronization cell constraints *Cs* specified for each CDC in terms of its source and destination flip-flop and their respective MFS depth as follows :

  ```
  cdc_sync_cell -name [name] -from [source] -to [destination]
  -depth [MFS depth]
  ```

  The latter constraint is then generated automatically and exhaustively for all the CDC paths reported by the tool. The global synchronization parameter is then reset to a normal value (typically between 2 and 4) to continue the verification flow. These constraints can then help the user to detect and solve the sequential re-convergence problems appearing on the deeper synchronizers.

- **Automatic generation of exclusive, unrelated and gray coded signals**: the same concept can be applied to auto-generate the exclusive, unrelated and gray coded signals constraints. The tool detecting all the re-convergence violations, can be re-fed to generate candidates for the exclusive signals constraints Ex that may eliminate them. In this case, the tool generates the following constraint for all the detected re-convergences :

```
exclusive -signals [{converging signals set}]
```

The idea of the new flow is to generate the set of primary constraints P in a more pragmatic way (procedure PSS_{new}) respecting the inter-dependencies between its different constraints. And then, in order to eliminate the inherited problems that increases the number of redundant violations, each set of generated constraints is verified against their corresponding rules before generating the next set (see code below line 1-27). On the other hand, procedure SSS (line 30) can overcome the limitation of the tool not being able to generate automatically the secondary constraints set S. A structural verification is performed and then the tool is re-fed its own reports in order to generate templates for synchronizers and exclusive signals. The flow finally keeps the same shape with the four main steps (line 42-62), performing each of them with more precision.

```
1  #New Primary Setup Specification PSS_new
2   procedure PSS_new(G, Ds, Π)
3     #Step 2.A.1 : clocks "C" and resets "R" generation
4       C,R ⟵ ConstraintClkRst(Ds)
5       #Step 2.A.2 : clocks and resets check and modification
6       C*,R*,Π*_C,R ⟵ SC(G, Π_C,R, C,R)
7       while (Π*_C,R ≠ ϕ)
8         C*,R* ⟵ Modify_ConstraintClkRst(Ds,C*,R*,Π*_C,R)
9         C*,R*,Π*_C,R ⟵ SC(G,Π_C,R,C*,R*)
10    #Step 2.B.1 : configuration signals "N" generation
11      N ⟵ ConstraintConfig(Ds,C*,R*)
12      #Step 2.B.2 : configuration signals check and modification
13      N*,Π*_C,R,N ⟵ SC(G,Π_C,R,N,C*,R*,N)
14      while (Π*_C,R,N ≠ ϕ)
15        N* ⟵ Modify_ConstraintConfig(Ds,C*,R*,N*)
16        N*,Π*_C,R,N ⟵ SC(G,Π_C,R,N,C*,R*,N*)
17    #Step 2.C.1 : Black Boxes "B" generation
18      B ⟵ ConstraintBBox (Ds,C*,R*,N*)
19      B*,Π*_C,R,N,B ⟵ SC(G,Π_C,R,N,B,C*,R*,N*,B)
20      #Step 2.C.2 : Black Boxes check and modification
21      while (Π*_C,R,N,B ≠ ϕ)
22        B* ⟵ Modify_ConstraintBBox(Ds,C*,R*,N*,B*)
23        B*,Π*_C,R,N,B ⟵ SC(G,Π_C,R,N,B,C*,R*,N*,B*)
24    #Step 2.D.1 : Primary Inputs "I" and Outputs "O" generation
25      I, O ⟵ ConstraintInOut(Ds,C*,R*,N*,B*)
26    P = C* ⋃ R* ⋃ N* ⋃ B* ⋃ I ⋃ O
27    return P
28
29  #Secondary Setup Specification "SSS"
30  procedure SSS (G_cdc, Π*_cdc)
31    #check the type of the violated properties in Π*_cdc
```

```
32      foreach p in Π*_cdc
33          #missing synchronizer
34          if (type(p)==p_cdc)
35              Cs ⟵ propose_synchronizer(Gcdc)
36          #reconverging CDC paths
37          else if (type(p)==p_conv)
38              Ex ⟵ propose_exclusive(Gcdc)
39      S = Cs ⋃ Ex
40
41
42  #NEW CDC STRCUTURAL VERIFICATION FLOW "CDCv_new"
43  procedure CDCv_new (G,Ds,Π,Π_cdc)
44      #Step 1 : Design compilation and Elaboration
45        DesignRead (G)
46      #Step 2 : Primary constraints generation "PSS_new"
47         P = PSS_new(G,Ds,Π)
48      #Step 3 : Setup Check "SC"
49          k*,Π* ⟵ SC(G,Π, P)
50          while (Π* ≠ ϕ)
51            k*,Π* ⟵ SC(G,Π, k*)
52      #Step 4 : Secondary Setup Generation "SSG"
53        #synchronizer depth parameter "sync_depth" configured to
    number larger than 4
54            sync_depth = 100
55        #Structural verification : detection of CDC paths and re-
    convergences
56            G_cdc , Π*_cdc ⟵ SV (G,Π_cdc,k*,Π*)
57            S = SSS(G_cdc , Π*_cdc)
58         K* = K* ⋃ S
59      #Step 5 : Global structural check
60        G_cdc , Π*_cdc ⟵ SV(G,Π_cdc,k*,Π*)
61     return G_cdc
```

The chart in Figure 2.5 represents the different steps of the flow. The flow still keeps its four main steps: compilation, constraints generation, constraints check and CDC structural check with more sub-steps for each of them. After step 1, where the design is compiled, the constraints generation is done following four sub-steps :

- **Constraints generation #1 (step 2.A)**: clock and reset constraints are generated (step 2.A.1). Then, a setup check, limited by the check rules that concern only the clock and the reset signals is performed (step 2.A.2).
- **Constraints generation #2 (step 2.B)**: once the clock and the reset signals are validated by the reduced setup check done previously, a new set of constraints, that concern the constant configuration signals is generated (step 2.B.1). Then, same as before, the reduced setup check against the rules that concern the constant signals, the clocks and the resets propagation is performed (step 2.B.2).
- **Constraints generation #3 (step 2.C)**: Once the latter reduced setup check is done and all the violations are eliminated by refining the concerned generated constraints, the black boxes constraints are generated and associated to their corresponding clock domains (step 2.C.1). Then, the reduced setup check is performed to check the validity of the specified black boxes attributes (step 2.C.2).
- **Constraints generation #3 (step 2.D)**: Once all the previously generated constraints are validated, in step (2.D.1), we generate the last set of primary constraints

which is the input and output ports associated to their correspondent clock domains.

Finally, a global setup check across all the setup rules is performed to ensure the harmony of all the specified constraints. Once the global setup check is done, the environment is ready for a CDC structural check. In our new flow, the CDC structural check is done following three sub-steps to assist the tool in the generation of the secondary constraints.

- **Structural check (4.A)**: In this step, we configure the global synchronization parameter to a very large number (larger than 4) then we run a structural check enabling only the rules to detect CDC paths. In this case, the tool reports all the CDC paths regardless of being synchronized or not. The detected CDC paths are then fed to a number of scripts that generate a constraints file that contains a detailed specification of the synchronizers of these CDC paths in terms of their depths, sources and destinations. The generated constraints should be compared to the design specification to keep only the relevant ones.

- **Structural check (4.B)**: The same concept is applied running a structural check, but this time, enabling only the rules to detect re-convergences. Some scripts assist the tool by analyzing these re-convergence violations and generating a constraints files with all the exclusive, unrelated or gray coded constraints to eliminate these violations. The generated constraints are then compared to the design specification and only the relevant ones are kept.

- **Structural check (4.C)**: the final structural check is done across all the CDC rules.

The last structural check can be iterated many times to solve the remaining violations with extra secondary constraints (step 5) or by waiving others. The latter flow was developed in a set of scripts integrated in a "STMicroelectronics Kit" that abstract the normal user commands with a set of easy linux commands. The "Kit" is actually available to all the verification engineers to be used for any of their projects.

2.2.3 Results

We applied the new flow to our CPU subsystem test case using one of the CDC static verification tools. As shown in table 2.1, the number of setup violations we analyzed using the new reference flow are about 14% of the number of setup violations we analyzed using the classic flow. We also observe a remarkable reduction in the number of structural analysis violations. About 75% of the noisy results disappeared using the new reference flow. We can say that with the new flow we analyzed a total of 20% of the violations we have analyzed using the classic flow to reach the same results. The verification time, being directly proportional with the number of violations to be analyzed, was also divided by four. As seen in the last column, both flows detected finally the exact same number of CDC paths. This means that the new flow did not mask any true violation, otherwise, it could not have detected the same number of CDC paths.

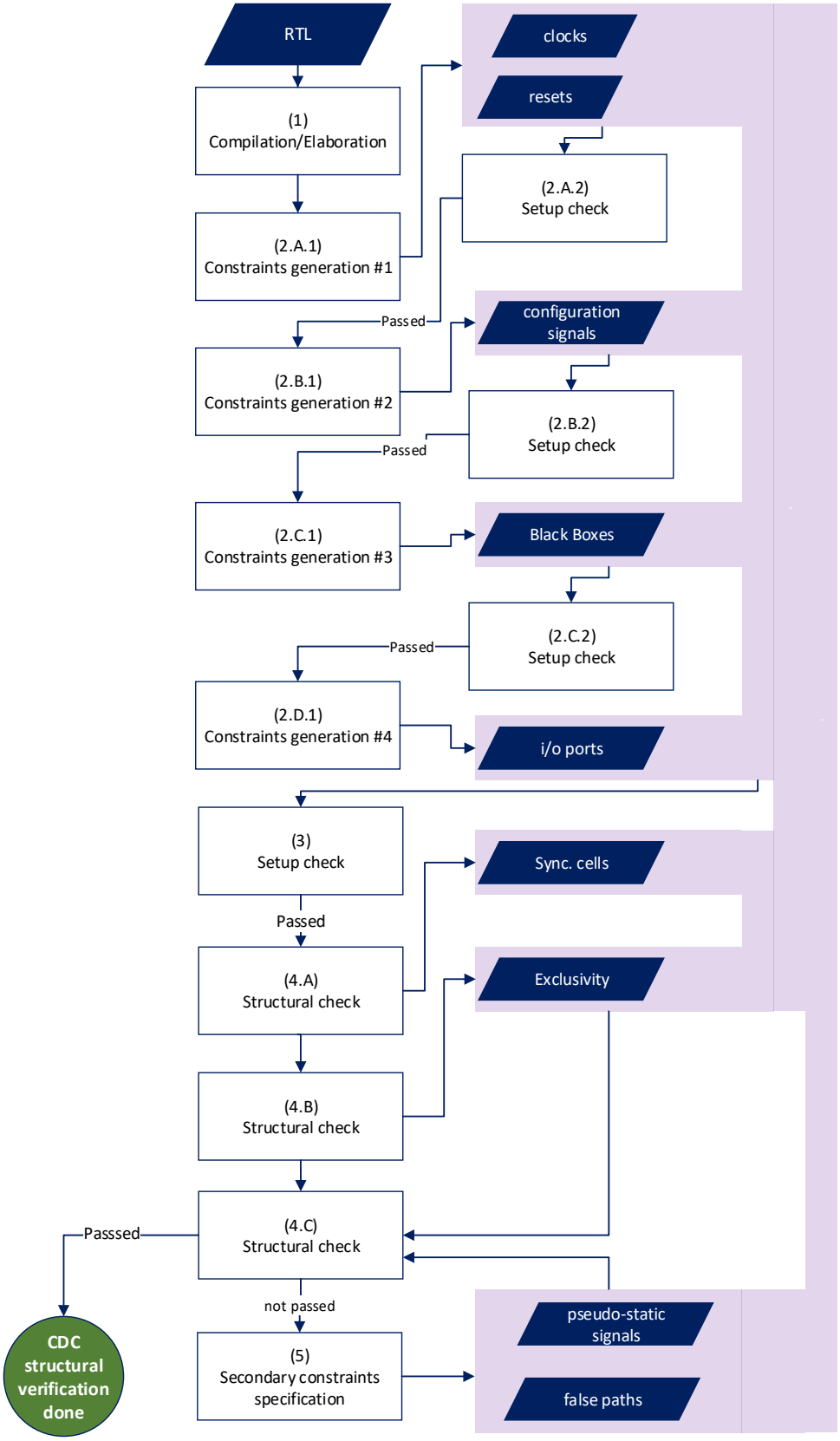


Figure 2.5: New CDC verification flow

Category	#analyzed setup violations	#analyzed structural violations	#detected CDC paths
#Classic flow	1780	6204	8365
#New Flow	250	1600	8365

Tab. 2.1: Classic flow vs. new flow results

2.3 UPF-Aware flow

The second aspect that needed attention to enhance the CDC structural verification was the UPF-aware flow. A design that includes multiple power and voltage domains is the subject of a dedicated static verification called "The Low Power Verification" [79] [80] [81]. The main target of this verification is to check for architectural errors related to low power design and violations related to the UPF specification [82]. The multi-voltage and multi-power domains aspect can affect the CDC verification, and ignoring it risks to hide a lot of CDC problems. In this section, we explain the relationship that exists between the low power logic and the CDC problems and our approach to combine the CDC and the low power verification [83].

2.3.1 Introduction

The UPF (Unified Power Format) is a standard format to describe all the power and voltage domains of a design [84] [85]. The UPF file is used to instrument the netlist with the necessary low power components during the synthesis phase. The following is an extracted example of the UPF file :

```
#Create power domains
create_power_domain pd_top -scope
create_power_domain pd_aon -elements {}

# Isolation strategy
set_isolation isol_clamp_sig
set_isolation_control isol_clamp_sig

# Level Shifter strategy
set_level_shifter LtoH_sig
```

Adding these components may result in functional errors and some of them are related to CDC [86]. An added low power component may corrupt an already synchronized path or create a new CDC path. A UPF aware CDC verification on RTL can detect this kind of problem at an early stage giving a higher quality and coverage [87]. The development of a UPF-aware flow was one the first priorities of the CDC verification team at STMicroelectronics. In this section, the CDC related problems due to the insertion of the power management cells as well as the proposed verification flow to overcome these problems are discussed. The results have shown that a number a CDC problems was

masked applying a verification flow that ignores the power management. Ignoring the power management control logic may introduce CDC bugs only detectable at Gate Level Simulation.

2.3.2 Challenges related to the insertion of the power management cells

The insertion of the power control logic is a challenge from a "CDC verification" point of view. Problems, such as glitches, new CDC paths or corrupted synchronizers, are prone to be masked when verifying CDC without considering the power management components. We started by studying the interdependence between the power control logic and the CDC verification. From what we have seen in the literature [88] [89] [90], we considered that the following three are the major challenges that relate the CDC verification to the power management.

- Level shifter inserted on a clock path :
  A level shifter is used to change the voltage level of a signal crossing two different voltage domains. The inserted level shifter can raise or reduce the voltage level of that signal depending on the voltage level of the destination domain with respect to the voltage level of the source domain [17]. The timing characterization of the level shifter depends on the functional ambient conditions (temperature, voltage, etc.). If inserted on some critical paths, the timing closure may not be satisfied due to the non deterministic clock transitions as the limited characterization precision involves a clock phase uncertainty. As the clock phase cannot be predicted and changes dynamically, crossing a level shifer, the output clock can become asynchronous to the input one. Figure 2.6 shows two flip-flops existing in two different voltage domains and clocked by the same clock. A level shifter must be placed on the data path as well as on the clock path. As the delays introduced by the level shifter on the clock path depend on widely variable parameters, they cannot be fully expected or covered during the static timing analysis (STA). This may lead to an unexpected timing behavior that resembles the timing challenges on a CDC path (setup/hold violation).

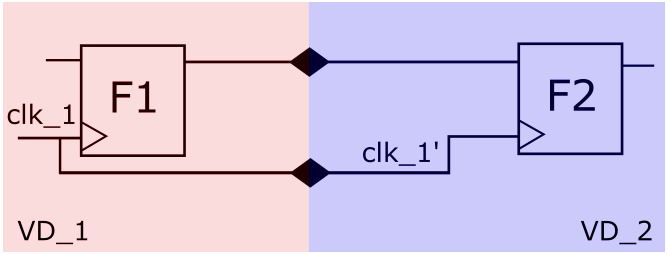


Figure 2.6: Level shifters placed on signals crossing power domains

- Isolation cell asynchronous enable signal :
  Isolation cells are placed at the inputs of the activated power domains to prohibit the logic coming from a shutdown power domain from driving active logic. The insertion of an isolation cell enabled by an asynchronous clock may create a new CDC path between the isolation cell enable signal and the destination flip-flop. In

Figure 2.7, the AND gate is an isolation cell that blocks the data transferred by "F1" when the power domain "PD_1" is off. The logic driving the isolation cell should be placed in the active power domain. As the isolation cell enable signal "EN" is driven by a different clock domain, a new CDC path will appear between the isolation cell enable signal "EN" driven by "clk_2" and "F2" driven by "clk_1". The behaviour of the isolation cells enable signals is often described as pseudo-static because usually the designers ensure that they never toggle if the destination clock domain is active.

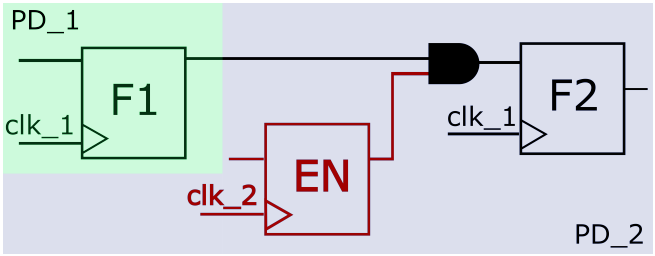


Figure 2.7: Isolation cell driven by an asynchronous enable signal

- Isolation cell inserted inside a CDC path :
  An isolation cell inserted inside a synchronized CDC may be problematic. Additional combinational logic between a source flip-flop and a multi-flop synchronizer is a high risk glitchy structure and can propagate unexpected values [91]. In Figure 2.8, the multiplexer acting as an isolation cell being inserted on a synchronized CDC path should not be allowed.

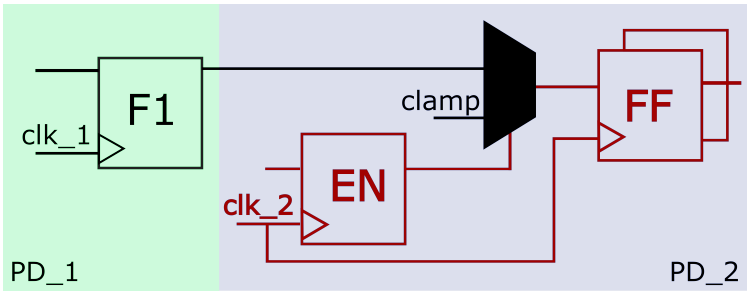


Figure 2.8: Isolation cell inserted inside a CDC path

2.3.3 UPF-aware CDC verification flow on RTL

As the instrumentation process usually happens during the synthesis phase, all the previously discussed CDC problems related to the insertion of the power management logic appears while verifying CDC on GLN (Gate Level Netlist). This requires modifying the RTL, re-synthesizing and re-verifying the CDC aspects at gate level. This process is costly in terms of time (number of iterations) and resources and can be avoided if the UPF is considered in the CDC verification on RTL. The verification engineer should load the UPF file during the design compilation/elaboration phase. The structural verification tool can instrument the design with the necessary power logic cells. This additional logic is then taken into consideration for the CDC structural verification. The problem is that

the tools do not consider, by default, a level shifter placed on a clock path. Therefore, the crossing shown in Figure 2.6 is considered synchronous by default even though the UPF is loaded. To create a UPF aware CDC verification flow on RTL, we identified two essential steps to be added to the original flow:

- The identification of the level shifters placed on clock paths :
  Using an open Tcl tool, we developed a Tcl script to parse the design and list all the level shifters placed on clock paths. Then, a custom tag “SETUP_LS_CLK_UNDEFINED” is developed to warn the user about a potential missing new clock declaration at the output of each of these level shifters. The severity of this custom tag is set to “error”.

- The identification of the enable signals of the isolation cells :
  As illustrated previously, the asynchronous enable signals of the isolation cells may add new CDC paths. Using an open Tcl tool, we developed a Tcl script to parse the design and list all the enable signals of the isolation cells. This report is important to identify the potential missing synchronizers or pseudo-static constraints related to the insertion of the power management cells.

Figure 2.9 shows the new UPF-aware flow on RTL. The orange sections mark the new added steps to the classic flow. In step "1", the UPF is read alongside the RTL in the compilation elaboration phase so that the verification tool can instrument the design with the low power logic based on the power and the voltage domains defined in the UPF. In step "2", a number of Tcl scripts assists the constraints generation phase to detect the different low power logic cells (Level sifters, isolation cells,...) and proposes two new constraints candidates files. The newly generated constraints concern the clock signals declaration at the output of all the level shifters that are instrumented on clock paths (see Figure 2.6), and the enable signals of the isolation cells that may contribute to some additional CDC paths proposing to constrain them as pseudo-static signals (see Figure 2.7). Finally, in step "3", the setup check takes into account the newly generated constraints and flags all the missing clock declarations at the output of the level shifters placed on clock paths with our new custom tag “SETUP_LS_CLK_UNDEFINED”.

2.3.4 Application and results

We applied our proposed flow to our CPU-subsystem test case. Our CPU-subsystem verification environment was built using a commercial CDC static verification tool. Loading the UPF and adding our custom scripts to the verification environment, we found that twelve level shifters were found on clock paths. The new custom error tag “SETUP_LS_CLK_UNDEFINED” is flagged twelve times to warn the user about a potential missing new clock declaration at the output of each of the detected level shifters. The structural verification differential results are shown in table 2.2. The “CDC_UNSYNC_NOSCHEME” tag reports the unsynchronized CDC paths. The UPF-aware verification reports twenty-two more unsynchronized CDC paths, which would have been missed if the UPF had not been included in the verification flow.

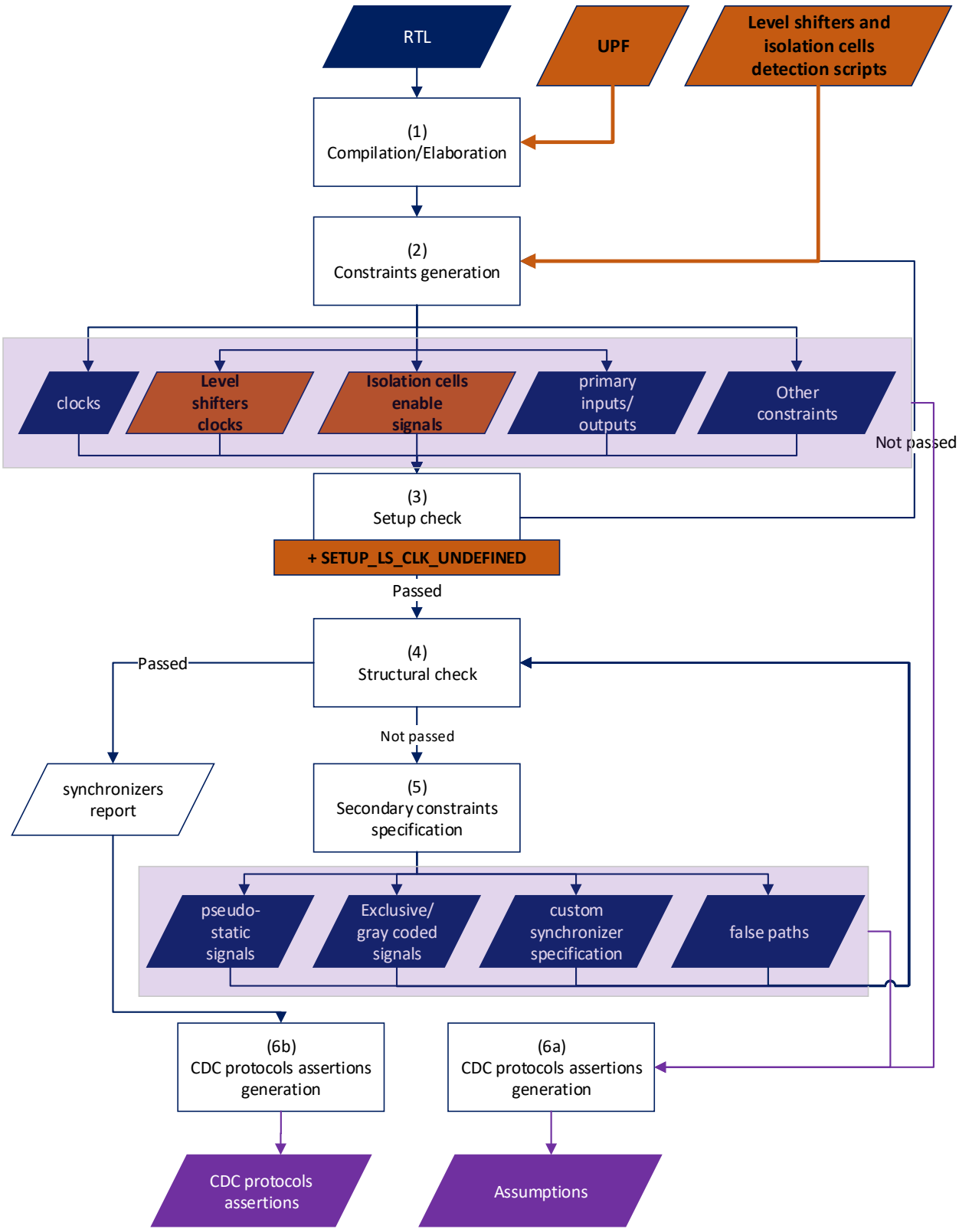


Figure 2.9: UPF-aware CDC verification flow

Detected violation	Classic flow	UPF-aware flow
SETUP_LS_CLK_UNDEFINED	0	12
CDC_UNSYNC_NOSCHEME	x	x+22

Tab. 2.2: UPF-Aware CDC Verification Flow Results

2.3.5 Conclusion

Designers face challenges due to the impact of the insertion of the power management cells on CDC paths. The CDC verification on RTL should be aware of such problems and report them at an early stage in the design cycle. A UPF-aware CDC verification at RTL guarantees higher QoR and coverage. Our test case proves that a lot of CDC issues would have been masked if the power management was not considered for the CDC structural verification on RTL.

2.4 Evaluation of the industrial tools

The EDA tools providers propose different technologies for the CDC verification. This technology appeared twenty years ago, even though, we still see sometimes great deficiencies in the proposed technologies. Therfore, we used our test case and the new optimized flow to rigorously compare the performance of these technologies.

2.4.1 Previous comparative anaylsis

There was an attempt to make a comparative analysis in 2015 by a previous PhD student, Mejid Kebaili, in the same domain [73]. His study was based on 6 criteria :

- **Clock Check (CC)**: the detection of the clock signals and domains.
- **Structural Check (SC)**: the detection of the different types of synchronizers.
- **Formal Properties (FP)**: the formal properties the tools are able to generate.
- **Formal Check (FC)**: the quality of the results performing the functional check using the CDC tool.
- **Tools Parameters (TP)**: the number of parameters to tune in each tool.
- **Environment Setup (ES)**: the number of constraints to use to have acceptable results.

Table 2.3 shows the results obtained by Mejid Kebaili published in his thesis [73]. We can see that five tools participated to the evaluation. "Tool 1" and "Tool 2" are very close in terms of total score, while the other three tools have a significant score gap. The

evaluation of each criteria is given one of three grades: (OK) means that the criteria is fully covered, (POK) means that the criteria is partially covered and (NOK) means that the criteria is not covered. Mejid Kebaili concluded that all the tools still present some deficiencies regarding the different CDC verification aspects, that tools 3,4 and 5 still need a lot of improvement and that only tool 1 and 2 can be considered reliable [73].

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
tools	CC	SC	FP	FC	TP	ES	Total
Tool 1	POK	POK	POK	OK	51	58	109
Tool 2	POK	POK	POK	OK	39	65	104
Tool 3	NOK	NOK	POK	POK	20	63	83
Tool 4	NOK	POK	POK	N.A.	27	N.A.	27
Tool 5	NOK	POK	POK	POK.	17	63	80

Tab. 2.3: Previous tools evaluation results

2.4.2 Evaluation aspects

Between 2015 and 2021, the static verification tools have remarkably evolved and new technologies were developed to assist the CDC verification flow. The inclusion of AI (Artificial Intelligence) to process the results and the possibility to verify the complex designs in a hierarchical way are among the most noticeable improvements. We decided to revisit the static tools evaluation with a new and exhaustive criteria list. The specified criteria were added based on the main needs of the verification team at STMicroelectronics, the new features proposed by the different tools providers and the new needs we encountered developing our new verification flow.

Our new criteria list has a total of 114 criteria. The weight of each criteria depends mainly on its location in the CDC verification flow, the number of its dependencies and its importance to the verification team. The criteria are classified into the following twelve categories:

1. **Constraints Setup Generation (CSG)**: The ability of the tool to detect and generate the different setup constraints able to precisely describe the design. The granularity of the different commands options is taken into account in this section. The CSG category includes **12** criteria describing in detail how the tool should detect and specify clock, reset and data signals.

2. **Constraints Setup Specification (CSS)**: The different options the tool provides the user to constrain his design. In other words, the constraints set the user can use to precisely describe his design. The CSS category includes **39** criteria describing in detail all the constraints that may be needed to describe the different signals and synchronizers.

3. **Constraints Check Verification (CCV)**: This includes the set of rules Π to be checked in order to detect over-constraining and under-constraining, as well as the

wrongly specified configurations such as clock and resets overlapping. The CCV category includes **12** criteria.

4. **Constraints Check Information (CCI)**: The reports the tool is able to provide following a setup check. The CCI category includes **5** criteria including clock and reset matrices.

5. **Structural Checks Verification (SCV)**: This includes the set of rules Π_{cdc} to be checked to detect the unsychronized CDC control, data and reset paths. In addition, the problems related to CDC such as re-convergences and glitches are taken into account. The SCV category includes **7** criteria.

6. **Structural Checks Information (SCI)**: The reports the tool is able to provide following a structural check. The CCI category includes **6** criteria including CDC matrices and CDC paths ignored due to a constrained pseudo-static signal.

7. **Formal Verification Verification (FVV)**: The ability of the tool to generate the properties that correspond to the detected synchronized and unsychronized CDC paths and the assumptions that correspond to the set of constraints targeted for the formal verification. The FVV category includes **8** criteria that summarize the complete set of properties and assumptions the tool should be able to generate.

8. **Formal Verification Information (FVI)**: the **4** criteria in this category evaluate the performance of the static tools formal engine and the quality of reporting it is able to provide in case on an inconclusive result.

9. **Semi-Formal Verification Verification (SFVV)**: The ability of the tool to generate the properties that correspond to the detected synchronized and unsychronized CDC paths and the assumptions that correspond to a set of constraints targeted for the semi-formal verification (dynamic simulation). In addition, the quality of the generated metastability injectors and their coverage is taken into account. The SFVV category includes **6** criteria that summarize the complete set of properties and assumptions the tool should be able to generate.

10. **Semi-Formal Verification Information (SFVI)**: the **4** criteria in this category evaluate the quality of reporting the generated assertions and their coverage.

11. **Hierarchical Model Generation (HMG)**: The ability of the tool to generate a CDC model for each subsystem. The HMG category includes **9** criteria that summarize the set of attributes that a tool should provide to define a CDC model.

12. **Hierarchical Model Verification (HMV)**: the **2** criteria in this category check whether the tool is able to make a bottom-up or a top-down analysis. The bottom-up hierarchical analysis starts by verifying each subsystem, then to generate a CDC model for each of them and to integrate all the CDC models to have a CDC abstracted view of the SoC top. On the other hand, the top-down starts by generating the setup on SoC level and then associating it to the different subsystems.

The number of criteria per section is listed in Table 2.4 below.

Category	CSG	CSS	CCV	CCI	SCV	SCI	FVV	FVI	SFVV	SFVI	HMG	HMV
#criteria	12	39	12	5	7	6	8	4	6	4	9	2

Tab. 2.4: List of criteria of the CDC static verification tools

2.4.3 Evaluation results

We decided to start our evaluation from where Mejid Kebaili has ended his. The decision was taken to re-evaluate only the top three tools in this new evaluation. Unfortunately, we succeeded to obtain only two tools' licenses. This is why our evaluation was limited only to the top 2 tools of the study of Mejid. The evaluation was done using the same test case (our CPU-subsystem) and following the new structural verification flow explained early in this chapter.

The different criteria were weighted based on the priorities and the needs at STMicroelectronics. As the structural verification is the main approach adopted for the CDC verification, the different criteria in this category had a weight of four or three. The criteria in the formal and the semi-formal categories had a weight of two or one because both approaches are still rarely used for the CDC verification. the criteria of the hierarchical verification had a weight of two or three, as the approach strated to be used in many organizations.

The results shown in Figure 2.10 show a very close score between both tools in many categories. The only remarkable differences can be seen in the *CSS*, *CCV* and the *SFVV* categories. Tool 1 outstands tool 2 in the structural categories. The constraints set provided by Tool 1 are more covering with respect to the different design aspects, compared to Tool 2 which lacks some important specifications options. We also noted that Tool 1 had a better synchronizers detection approach than Tool 2 which wrongly clustered a number of the synchronizers in our test case. On the other hand, Tool 2 outstands Tool 1 in the semi-formal category. Tool 2 had the ability to generate more reliable and covering assertions, targeted for simulation, than Tool 1. Another break point is the compatibility of the generated assertions with the third party simulation tool used by STMicroelectronics. Using a third party simulator, we lose some paths coverage (the CDC tools cannot create the hierarchical path of a generate block for a third-party tool). The assertions coverage we lost using Tool 2 was less important than the coverage loss using Tool 1. This aspect was given a heavy weight relative to its importance to the team.

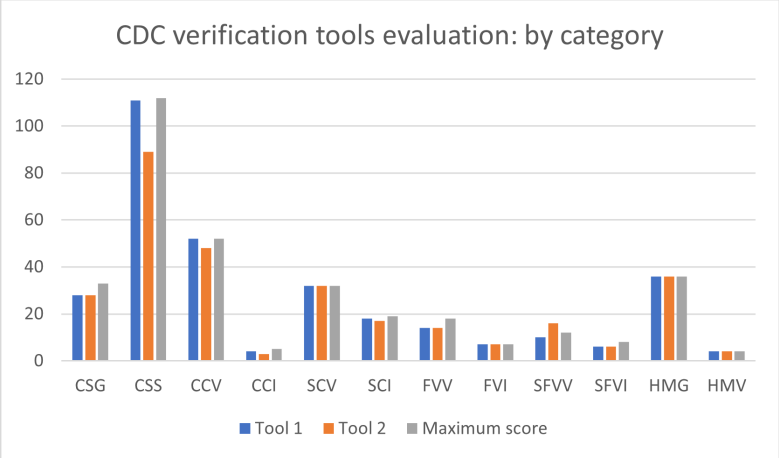


Figure 2.10: Tools evaluation results by category

The overall score shown in Figure 2.11 shows that in general Tool 1 meets more the actual industrial expectations than Tool 2. While Tool 2 has the advance in the semi-formal categories, the structural verification is still the lead approach used for the CDC verification and the most important signoff method. Finally we can conclude that our study and the one done in 2015 by Mejid Kebaili reached the same general conclusion, with a slight precision from our side on the semi-formal aspect.

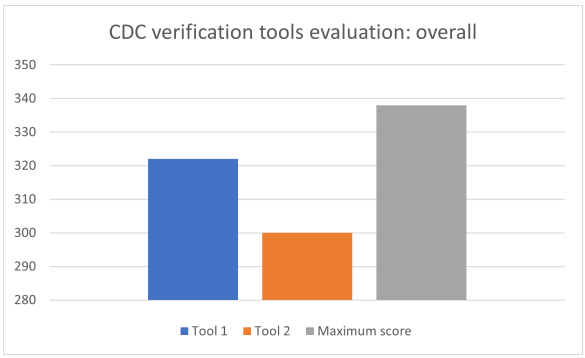


Figure 2.11: Tools evaluation overall results

2.5 Conclusion

For synchronous circuits, the STA (Static Timing Analysis) is an essential step to ensure that the timing closure is respected. However, in multi-clock circuits, the STA cannot ensure the correctness of a clock domain crossing. The CDC structural verification is the first safeguard against the hazards of asynchronous paths. It aims at detecting the CDC paths of the verified design and to ensure the presence of proper synchronizers. Being the most, and maybe the only, adopted approach by the industry for CDC verification, studying and enhancing the structural verification flow was our priority. This chapter has three main axis. The first axis was the development of a new structural verification flow able to overcome the deficiencies existing in the original flow. Our proposed flow was based

on respecting the interdependencies between the different constraints and to re-feed the tool its own reports in order to automate some constraints generation. The new flow presented also a pragmatic approach verifying the different design and CDC rules in order to enhance the tools' violations reporting and to limit the number of false positives and negatives. Applying our proposed flow to verify our test case, about 75% of the noisy results disappeared. The second axis was the inclusion of the low power management logic to the CDC verification flow. A design that includes multiple power and voltage domains is the subject of a dedicated static verification called "The Low Power Verification". The main target of this verification is to check for architectural errors related to low power design and violations related to the UPF specification. Studying the effect of the low power management logic on the CDC aspects, we concluded that it could corrupt an already synchronized CDC or even create new ones, and ignoring it risks hiding a lot of CDC problems. To address this problem, we developed the "UPF-Aware CDC verification flow". The idea of this new flow is to include the UPF file to instrument the verified design with the low power management logic. The focus was to update the tools to consider new clock declarations on any level shifter inserted on a clock signal, and to detect the potential asynchronous control signals controlling isolation cells. Applying this flow, we were able to see more CDC paths than those detected applying the original flow. The last axis was the evaluation of the different CDC static verification tools of the market. The assessment has shown that the tools are still not able to cover all our expectations and the requirements of the CDC verification.

3

CDC Semi-formal verification

This chapter advocates enhancing Clock Domain Crossing (CDC) verification by combining structural verification with assertions-based approaches. The new approach, the CDC semi-formal verification, aims at bridging both static CDC verification and the dynamic functional verification in order to enhance the CDC verification's quality of results. The first section explores CDC semi-formal verification, demonstrating 100% assertion coverage and the discovery of previously undetected bugs. The second section proposes solutions to deficiencies in the semi-formal flow, including a generalized method for verifying CDC data synchronizers using "The Universal Qualifier." The third section introduces the "Hybrid Flow," leveraging semi-formal verification to both complement and assist CDC structural verification during the setup phase. Results reveal the effectiveness of the hybrid approach and emphasize an iterative strategy for coherent clock propagation.

Contents

3.1 Introduction

The nightmare of metastability is a direct implication of multi-clock designs. It was a phenomenon discovered decades ago and if propagated can lead to what so-called *dead system*. Synchronizer design and verification has a first target to ensure that metastability can be properly blocked. The CDC structural verification, which aims to detect the presence of these synchronizers, is an important and fast approach to reach this target. However, the technology still holds a number of limitations that affects remarkably the quality of the obtained results and that leaves a back-door for bugs to remain hidden till the silicon stage.

In this chapter, we discuss the limitations of the CDC structural verification and revisit the CDC assertions-based verification being a complementary analysis able to overcome these limitations. We have chosen the semi-formal approach to verify these assertions relying on high coverage functional verification environment. Our approach aims at constructing a bridge between the CDC verification and the dynamic functional verification, two completely independent activities in the industry nowadays. We will show some examples of CDC bugs found by this new approach that could have not been discovered by the structural verification alone. Assessing the first application of the semi-formal flow, we identified two development areas. The first was the problem of detecting complex synchronizers and generating reliable properties for them. For that, we propose a generic model for the protocols-based synchronizers and the new concept of *The Universal Qualifier*, which is able to verify all synchronizers regardless to their type. The second area was the need to have a robust and a reliable flow to constraint the clock tree and to accelerate the setup phase of the structural verification. For that, we present the *Hybrid Flow* to show how the CDC semi-formal verification can assist the structural verification in what concerns it.

3.1.1 CDC structural verification limitations

The CDC structural verification, despite of being a very important step for detecting CDC bugs, is not enough to ensure the validity of a design. The absence of a synchronizer implies a strong probability that a metastability can be generated and propagated; while the presence of these synchronizers does not necessarily implies the immunity of this design against metastability propagation. Limiting the CDC verification to the structural checks is simply taking a huge risk to miss a fatal bug. The limitations of the CDC structural verification relies in the fact that the static checks are :

1. **Constraints based static checks :** The set of constraints specified at the beginning of the structural verification flow is a very important dependency on which relies the quality of the CDC structural checks. The clock signal propagation is a very simple example that shows how critical a wrongly specified constraint can be. In Figure 3.1, the configuration signal must be constrained to either zero or one in order to propagate either "clk_1" or "clk_2". If the configuration signal is constrained to zero, "F2" will be clocked by "clk_1" and the path "F1" to "F2" will be reported as synchronous. If the configuration signal was intended to be constrained to one and it was constrained to zero, which means that "F2" is intended to be clocked by "clk_2" and not "clk_1", the analysis of the "F1" to "F2" path, which is in this case an asynchronous and unsynchronized path, will be skipped because it will be seen

as a synchronous path. As the constraints generation and specification is a process that mainly depends on the experience and the talent of the engineer writing them, the flow can be very prone to human errors.

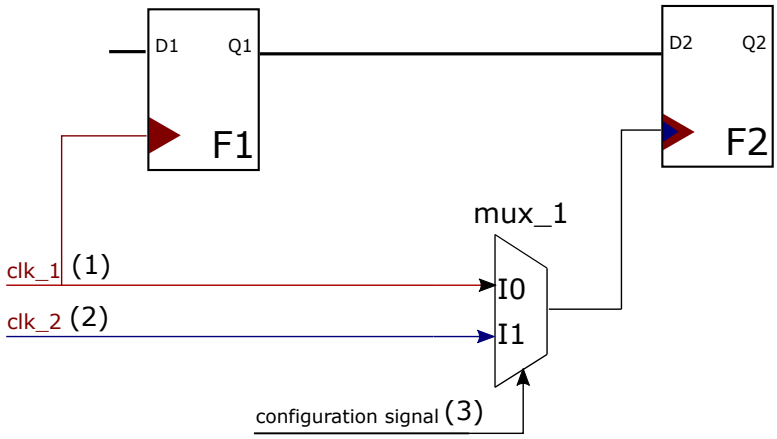


Figure 3.1: Constraining clock propagation control signals

Indeed, the constraints are taken blindly by the the static tools. Nowadays, it exists no technology in order to double check these specified constraints with any pre-defined reference. In order to imagine how critical this can be, look at Figure 3.2. If the "F1/Q1" is constrained as pseudo-static, under the assumption that "clk_2" is cut whenever "F1/Q1" is toggling, the CDC "F1" to "F2" is reported safe with no check. The static tool is not able to verify if the constraints specified by user are really valid, instead it takes them as hypothesis on which the quality of checks completely relies.

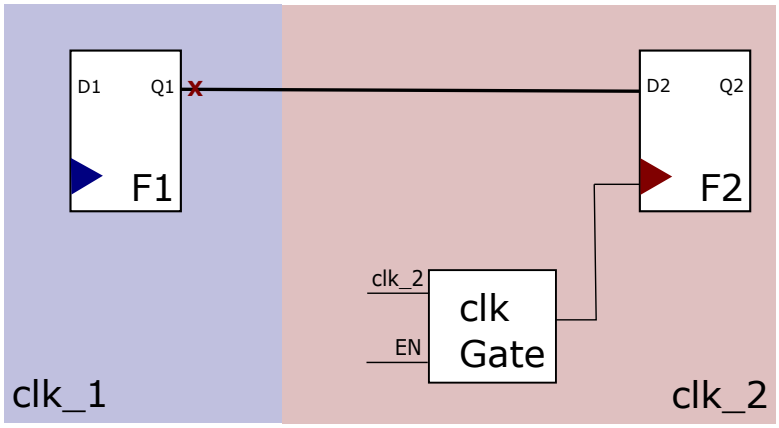


Figure 3.2: Constraining pseudo-static signals

2. **Rules based static checks :** The second limitation relies in the fact that the structural verification is a static approach that depends mainly on pre-coded patterns matching. Regarding the complexity and the customization of the wide variety of CDC synchronizers, specially the data synchronizers, it is impossible to cover all of them by pre-coded patterns. This means that the static tools, analyzing a complex or a custom synchronizer, either oversimplify it and reports it as valid while it is not, or the analysis does not succeed to converge (the tools do not succeed to detect

and categorize the synchronizer) and reports it as problematic while it is not. That explains the tens of thousands of false positives and negatives that we find usually among the results that take weeks and even months to be analyzed and filtered.

In addition, detecting a synchronizer does not imply that the CDC path is safe or immune to metastability. Each synchronizer is associated to a number of functional properties that must be verified and that cannot be verified by a static approach.

The conclusion is that a new complementary approach should be adopted beside the structural verification in order to overcome these limitations.

3.1.2 Assertions-based verification

The assertions based verification is the complement we are looking to overcome the limitations of the structural flow. The current EDA tools generate SystemVerilog assumptions and assertions based on the hypothesis and the results of the structural checks. This set of assumptions and assertions, if correctly generated, is able to overcome the two limitations related to the structural checks we talked about in the previous section.

1. **Overcoming the constraints limitation :** The constraints, being the hypothesis on which relies completely the quality of the structural checks and the formal checks, should be cross checked and qualified. The constraints responsible for clock propagation control the number of clock domains and CDC paths the tool will be able to see. On the other hand, the design constraints, that specify a signal as pseudo-static or a group of signals as exclusive, are also responsible for the number of CDC paths the tool will analyze. In order to validate the results of the structural checks, both types of constraints should be double checked against a pre-defined reference. The CDC verification EDA tools translate the set of constraints specified by the user to a set of assumptions that can be double checked in simulation environment or reused in a formal verification environment. Examples on CDC constraints and their correspondant assertions are given in Annex C

2. **Overcoming the functional limitation :** As discussed before, the CDC synchronizers can be classified into two main classes : control signals synchronizers and data signals synchronizers. Each type of synchronizers is associated with a set of functional properties that must be verified in order to ensure the safety of a CDC path. The current EDA tools are also able to classify the detected synchronizers and to associate a set of properties for each of them.

   Control signals : Multi-Flop Synchronizers (MFS) Multi-flop Synchronizers do not prevent metastability, but instead delay the signal giving the time to the metastability to be resolved. They prevent the propagation of the metastability in the downstream logic. The resolution value of the metastability is unknown. The metastability can be resolved to a value or its opposite one on a completely random basis (metastability will be discussed in details in the next chapter). For that, and in order to avoid data loss, the data should be stable long enough for the destination clock to be correctly sampled. This stability time is usually considered as three edges of the destination clock. This is verified using the following assertion :

```
mfs_stable : assert property
(@(posedge clk_2) disable iff (rst)
$changed (D1) |=> $stable(D1)[*nb_of_cycles]);
```

If the CDC signal, synchronized by MFS is a data bus, the "mutex" property (refer to Annex C) is asserted to ensure the exclusivity of the different data bits and to check the absence of coherency problems for the converging CDC paths as exaplined in section 1.2.1.

Data signals : Enabler-based Synchronizers A qualifier is easily found by the static tools. It is sufficient to detect a re-synchronized signal that blocks the data on a blocking logic. The qualifier should enable the data only when it is stable. This implies the qualifier to be disabled every time the data changes its value to ensure blocking the propagation of any metastable state. This can be ensured by the following assertion :

```
dataqualifier_stable : assert property
(@(posedge clk_2) disable iff (rst)
ctrl |=> $stable(Q2));
```

In addition, the qualifier itself should remain stable for a certain number of clock cycles being a signal synchronized by MFS to avoid dataloss.

```
ctrlqualifier : assert property
(@(posedge clk_2) disable iff (rst)
$changed (ctrl) |=> $stable(ctrl)[*nb_of_cycles]);
```

Data signals : Complex protocols Complex protocols are very challenging to be detected structurally due to the wide variety and the customization of the used designs. However, the static tools providers still put a lot of effort to cover these synchronizers. In case a complex protocol is structurally detected, a set of assertions is directly associated to it. If we take the asynchronous FIFO as an example, ensuring that the write/read addresses are gray encoded and stable with respect to their MFS is sufficient as a protocol check for the static tools. The same concept is applied to the other known protocols exaplined previously in section 1.2.2.

3.1.3 Formal and semi-formal verification

The generated CDC assertions can be verified in either formal or semi-formal approach. In case the structural tools detect all the CDC synchronizers, generate their correspondent functional assertions and the assertions are proven, the CDC paths can be considered as safe.

The formal verification is a functional static verification where mathematical techniques are leveraged to explore the state machine model of the design for any property violation. The advantage of formal verification is that it is an exhaustive approach, that can have the final word in the proof of any functional property. However, this exhaustiveness can also be the downside of this approach. Model checking may not achieve a conclusive result, and sometimes no information is returned because of timeout. This usually happens in large designs, due to state space explosion, where mainly the CDC problems appear. In [72], the authors explained the problem of the inclusiveness in three points :

- Design setup : in designs where multiple clocks can be propagated based on specific configuration, the model checker will try to prove every potential configuration. Sometimes, some configurations are unrealistic. In other words, some clocks can be exclusive or not propagated at the same time. For that, and in order to control the problem of the state space explosion, the design should be constrained and configured only in realistic modes, so that the state space is reduced aand only realistic behaviours can be inferred.
- Abstraction : the inability of the tools to efficiently abstract the related logic is a major cause of state space explosion. In CDC properties, only local control has real influence. That means that if an abstraction is done correctly, the formal engine can achieve reliable results that are not costly in terms of effort and time.
- Secondary constraints : if the user is able to add more constraints as the analysis progresses to assist the engine to converge the proof, that may have remarkable effect of the conclusiveness of the results.

On the other hand, the CDC semi-formal verification comes as an option, never explored by the industry up to our knowledge, that can be very advantageous regarding the quality of results and the time the iterations take. Our decision to explore a full CDC semi-formal verification flow came from an industrial demand and belief that the semi-formal verification can be a quick and exploitable solution, able to overcome the structural verification limitations and compromise the exhausivity, the time and the quality of results of the formal verification. In the next section, we will explain more our approach defining the semi-formal verification flow, the application on our test case and our assessment for the whole approach.

3.2 CDC semi-formal verification flow

The dynamic simulation is no longer a naive verification approach, that depends on hand written stimulus unable to exhaustively and efficiently verify a design. In 2011, Accelera released its UVM "Universal Verification Methodology" standard for building simulation based verification environments. It is a class library based on SystemVerilog that focuses on re-usability for stimulus generation and coverage modeling and checking. The dynamic functional verification is always maintained by a whole team of engineers ensuring high verification coverage. The UVM standard defines two coverage methods [92] :

- Explicit coverage : the user defines the coverage goals and completing these goals is the metric used to determine the completion of the DUT verification. The functional coverage is an example of such a metric. The functional coverage is a measure of the percentage of the pre-defined functionalities and features exercised by the different tests. The problem is that it depends mainly on what the user defines and any missing goal is not taken into account defining the coverage.

- Implicit coverage : the coverage is done with automatic metrics driven by RTL. The code coverage (whether block, expression, toggle or statement code coverage), that ensures that all the code is exercised, and the FSM coverage that ensures that all the states of the state and the state transitions were reached, are examples of the implicit coverage. This coverage is also not complete as a 100% code coverage does not guarantee a the absence of a functional hole.

Both coverage metrics together may achieve completeness. Starting by explicit coverage representing the high-level verification goals and complementing it by implicit coverage is always recommended. If the functional coverage is much higher than the code or FSM coverage, it should be redefined and enhanced.

As the dynamic functional verification is mature enough and taken in charge by a large team who ensures high functional, code and FSM coverage, we thought that we can take advantage of such a mature verification environment to complement the CDC structural verification and, at the same time, avoid the limitations related to the formal verification verifying large designs. The CDC semi-formal verification is a trade-off between the completeness of the formal approach and the rapidity of the dynamic simulation. If all the functional goals are reached (100% functional coverage) and at the same time all the code lines are exercised and all the FSM states are reached (100% code and FSM coverage), it can be a pity to not use such an environment in favor of the CDC verification. We decided, collectively, to explore a CDC semi-formal verification without any idea about the application challenges and the potential areas of development. Working in an industrial context with a large design and with cross verification environments using multiple tools coming from different tools providers were the main challenges we encountered applying this flow. As we progressed, the areas of potential developments started to appear and were prioritized based on the industrial needs and the current projects requirements. In the next section, the applied flow and the encountered challenges will be discussed in more details. In addition, the two research subjects we have chosen to enhance the flow are later developed in the chapter.

3.2.1 Application

As a first step, we applied the CDC semi-formal verification flow on our CPU-subsystem test case, as it is proposed by the different tools, in order to have a first understanding of the state-of-the-art, to make an assessment and to identify the potential development areas.

The verification flow

The CDC semi-formal verification flow, as indicated in Figure 3.3, should start by a static structure analysis in order to detect all the CDC paths and the synchronizers. The static analysis inputs are the RTL design, the constraints files and the UPF. The reports of the

structural verification should be analyzed and filtered from all the false positives and negatives. The real design bugs, such as missing synchronizers or glitches due to combinational logic inside CDC paths, are then reported to the design team to be fixed. This task is iterated several times until no design bugs are detected. The second step is the generation of the different sets of assertions. We can categorize the generated assertions into two categories :

- Constraint assertions : These assertions are generated thanks to the specified constraints files. The target of these assertions is either to be used as assumptions for the formal verification, or as assertions in the functional dynamic simulation in order to be cross checked with the simulation patterns.

- Protocol assertions : These assertions are generated for each detected synchronizer based on its type in order to verify the functional protocol behind the detected synchronizer structure.

The generated assertions, being included in different modules, are then connected to the top design using the SystemVerilog "bind" construct. The functional verification regression tests, which have already successfully passed in standalone, are then launched with both sets of assertions. The key success element of the CDC semi-formal verification is the quality and the coverage of the dynamic functional verification tests.

Test Case : Structural verification

The structural verification on our test case took several months to be accomplished. The constraints specification phase was the longest, due to usual unavailability of a clear design specification till the late stages of any project. Yet, there exists no standard for the designs specification able to shorten the setup phase. Table 3.1 shows the final set of constraints used for our CPU subsystem. We declared 105 clock signals and grouped them in 32 clock groups (clock domains). The design includes 47 reset signals, 400 pseudo-static signals (signals that should not cause any CDC problem because they are changed either when the destination is under reset or when its clock is deactivated), 59 exclusive signals and 169 configuration signals that were set to constant values in order to propagate the different clock signals such that the design is put in a worst case scenario.

Constraints	**Number**
Clocks	105
Clock domains	32
Resets	47
Static signals	400
Exclusive Signals	59
Constant Signals	169

Tab. 3.1: Design constraints

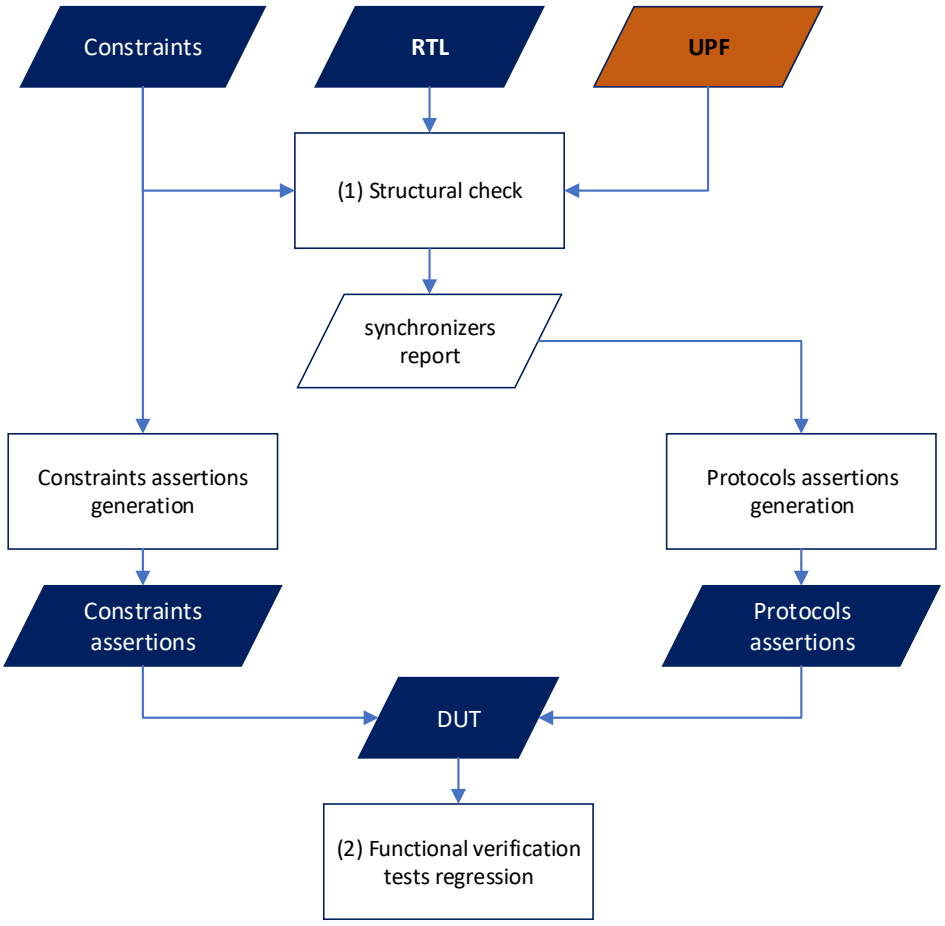


Figure 3.3: CDC semi-formal verification flow

The structural verification was then performed using the set of constraints specified above. Hundreds of thousands of violations were analyzed. The structural violations can be categorized into two categories :

- Unsynchronized CDC paths : A CDC path without a proper synchronizer.
- Problems related to CDC : The other problems that can arise due to, even, the synchronized CDC. For example, a detected combinational logic inside a CDC path is considered as a high risky "glitchy" structure, specially for slow to fast crossings where the glitch is more likely to be captured. Another example can be the reconvergences where multiple signals synchronized by MFS re-converge in the downstream logic. As the time resolution and value of the metastability on these different signals may be different, coherency problems can appear in case of they re-converge if they are not exclusive.

The final report of the structural verification is shown in table 3.2. With our constraints, we managed to resolve all the clock overlaps and the constant clock violations. That means that all the clocks configurations are resolved and all the sequential elements of the design receive a clock signal. All the detected CDC paths were found with a proper synchronizer and can be categorized into :

- Synchronized CDC control path : 302 scalar control signals were found synchronized by MFS.
- Synchronized CDC data path : 115 data signals were found synchronized by qualifier or other complex synchronization protocols.
- CDC paths with static signals : 2781 CDC were skipped because of a pseudo-static constraint on their crossing signal. These paths are usually found without synchronizer and their synchronization depends on the fact that either the destination clock is cut or the destination register is under reset when they toggle. The validity of these paths depends mainly of the specified constraints, and at this stage, no double check is done to ensure their trustworthiness.

Tag	**number**
Clock Overlaps	0
Clock Constants	0
Unsynchronized CDC	0
Synchronized CDC control path	302
Synchronized CDC data path	115
Skipped CDC due to static signal	2781

Tab. 3.2: Structural verification final report

Test Case : Assertions generation

Once all the structural violations are analyzed and corrected, we proceeded to the assertions generation step. As shown in Table 3.3, the generated assertions covered 100% of the constraints specified for the structural verification. On the other hand, we noticed a limitation regarding the generated assertions for the detected synchronizers. 50% for the CDC control paths synchronized by MFS and 20% for the CDC data paths synchronized by qualifier or other protocols were not covered by the generated assertions. The reason of this coverage limitation will be discussed later in the "Assessement" section 3.2.2.

	#constraints/synchronizers		**#Assertions**
Constraints	Static signals	400	400
	Exclusive signals	59	59
	Constant signals	169	169
Synchronizers protocols	Control signals synchronizers	302	150
	Data signals synchronizers	115	90

Tab. 3.3: Generated assumptions and assertions

Test Case : Functional regression

The CPU subsystem functional verification plan was created by our functional verification team at STMicroelectronics. The regression was constituted of 130 tests as indicated in Table 3.4. The tests cover all the functionalities of the different sub-blocks of the design. The functional tests converge as well as the code coverage achieved 100% by the end of the development phase. The dynamic verification environment was mature enough (all the tests had a "passed" status) by the time we started using it to apply the CDC semi-formal verification flow. The functional regression was done using a third party tool, coming from a tools' provider other than the CDC static tool provider with which the structural verification was performed.

Test Case : CDC semi-formal verification

To start the CDC semi-formal verification, and once we put in place the functional verification environment, we connected the generated assertions to the top of our test case using a SystemVerilog "bind" construct. The number of exercised assertions depended on the code coverage of the functional regression. In Figure 3.4 and 3.5, we can see how the number of exercised assertions progressed with the progress of the code coverage. Some assertions were fully exercised by 50% code coverage, such as the assertions of the configuration constant signals (shown in grey in Figure 3.4). Some other constraints were fully exercised with higher code coverage. This shows how the code and the functional coverage of the dynamic verification play a very important role in the completeness and the reliability of the CDC semi-formal verification results. The assertions coverage, expressing the number of exercised assertions with respect to each category of tests can be found in Annex D. It was noticed that some tests exercised more synchronizers assertions,

Test acronym	Scope	#tests	Status
Test 1-21	Basic CPU and SubSystem functionality	21	Passed
Test 22	Memory integration	1	Passed
Test 23-38	Interrupts	15	Passed
Test 39-56	Debug and trace	23	Passed
Test 57-68	System performance and power	11	Passed
Test 69-73	Clock and reset management	4	Passed
Test 74-79	Register programming	5	Passed
Test 80-84	Connectivity	5	Passed
Test 85-92	Memory integration	8	Passed
Test 93-96	Register verification	4	Passed
Test 97-104	Interface verification	8	Passed
Test 105-107	Custom IP verification	4	Passed
Test 108-130	Power aware functionality	23	Passed

Tab. 3.4: Functional verification tests regression

or in other words involved more CDC paths, than other. For example, the the interface verification tests exercised all the synchronizers assertions, which means that this group of tests stimulated all the asynchronous paths detected by the structural verification. Finally we can conclude that the functional verification regression in its totality was able to exercise 100% of the CDC constraints and synchronizers assertions as shown in Figures 3.4 and 3.5.

Test Case : Results

A number of the bound assertions failed while launching the functional regression. In Table 3.5, the 400 pseudo-static assertions, corresponding to the estimated pseudo-static signals, are passed. This can ensure that the implemented protocols really ensure the stability of the CDC signal. This can ensure that the CDC paths skipped during the structural verification because of the static constraints are safe. In other words, the static assumptions or hypothesis we made for the static verification are valid. On the other hand, we can see a number of failed assertions in all the other categories. Thirteen exclusive signals assertions failed, which means that some of estimated gray coded signals are not. Twenty-four constant signals assertions failed. As the constant configuration signals are the one responsible for propagating the clock and the reset signals, the failing assertions can give an indication that the design was not configured the same way for the functional regression and the CDC checks. Finally, some other assertions related to the protocols of the detected control and data synchronizers failed. That can confirm that the fact of detecting statically a synchronizer is not enough to ensure the validity of a CDC path.

In the next section 3.2.2, we analyze in details the detected violations. This will help

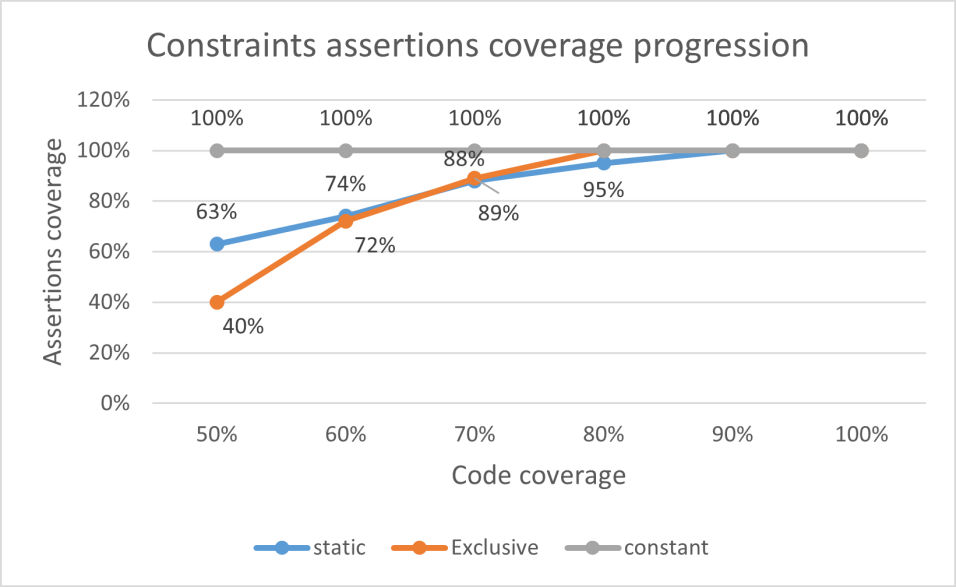


Figure 3.4: Constraints assertions coverage progression wrt to code coverage

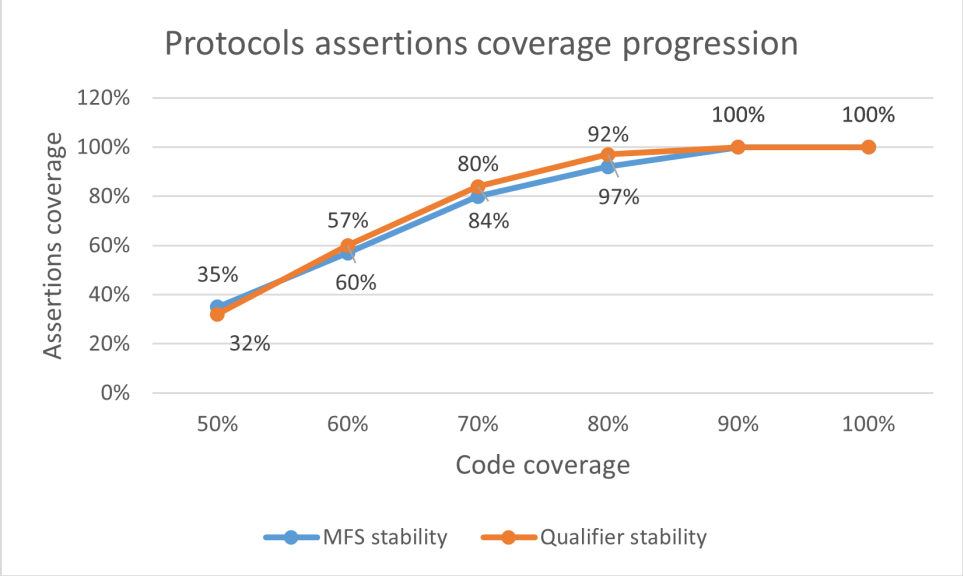


Figure 3.5: Protocols assertions coverage progression wrt to code coverage

	#constraints/synchronizers		#Assertions	#passed	#failed
Constraints	Static signals	400	400	400	0
	Exclusive signals	59	59	46	13
	Constant signals	169	169	145	24
Synchronizers protocols	Control signals synchronizers	302	150	146	4
	Data signals synchronizers	115	90	61	29

Tab. 3.5: Results of the CDC semi-formal flow

to assess the whole methodology and to identify the worthiness of applying this flow. The detected violations were like the compass that oriented the research subject after that, and clarified the potential development areas to push the limits of the semi-formal flow.

3.2.2 Assessment

The CDC semi-formal flow, being applied for the first time in STMicroelectronics, had to be assessed. The assessment aimed at evaluating the worthiness of applying the flow given the quality of its results and the added value versus the time it takes. This was expressed in terms of the CDC bugs we were able to detect using the semi-formal verification flow, that were not detected by the structural verification. The assessment also shows the challenges we encountered applying the flow and clarifies the deficiencies of the actual semi-formal flow. This led to identifying the potential development areas where we can push the boundaries of this technology.

CDC violations analysis

Applying the semi-formal flow, a number of assertions in different categories have failed as shown in Table 3.5. The failing assertions were not only about design bugs. Some assertions failed due a deficiency in the verification flow itself. Some other assertions failed due to inherited setup errors done for the structural verification.

1. **Failing exclusive assertions :** We can see in Table 3.5 that 13 assertions related to the exclusive signals failed.

```
//Exclusive signals
property mutex (data, clk);
    @(posedge clk)
      $onehot0(data ^ $past(data));
 endproperty
```

These failing assertions mean that some of the signals that meant to be exclusive, either due to a constraint specified by the user, or a necessity to avoid coherency problems, are changing at the same time. The failing CDC paths resembled all to Figure 3.6. The data bus "src [N:0]" was categorised as a valid CDC synchronized by MFS. This was justified by the cascade of two flip-flops ("dest1" and "dest2") driven by the destination clock "clk_2" in the fanout of this crossing the structural tool has seen. Consequently, and in order to avoid coherency problems, the "mutex" property was asserted for these paths. Analyzing the waveform of the failing assertions, we noticed that the data bus was not gray encoded. That could be a real design bug if "src1" was really synchronized by MFS. But revisiting the CDC paths, we noticed that the first destination flip-flop "dest1" is controlled by an enable signal "En". "En" is driven by a valid qualifier "ctrl", a control signal synchronized by MFS. The whole control part (indicated by a grey box) was invisible by the tool, which was tricked by the presence of a cascade of flip-flops. Instead of reporting the path as synchronized by a qualifier, the tool reported the path as synchronized by MFS and consequently asserted, wrongly, the "mutex" property for it. This has

shown a huge deficiency regarding the priorities of the algorithm logic responsible for analyzing the CDC structures.

Conclusion: The "mutex" property failed due to the wrong classification of CDC paths as synchronized by MFS instead of a qualifier.

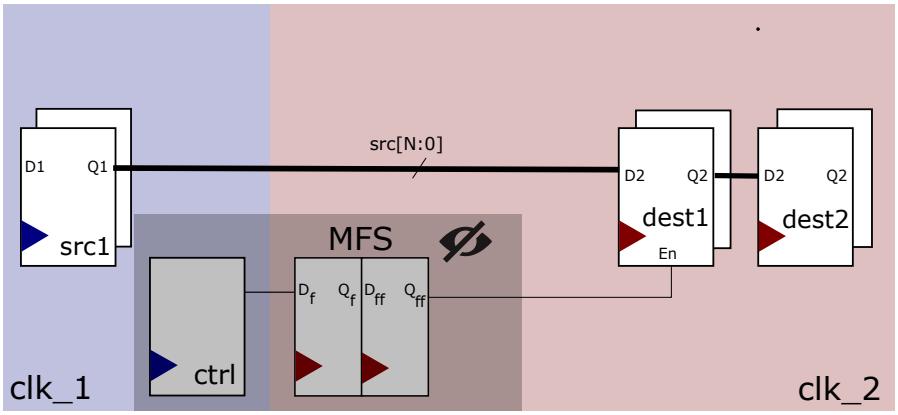


Figure 3.6: Failing mutex assertions due to an undiscovered qualifier

2. **Failing constant assertions :** We can see in Table 3.5 that 24 assertions related to the constant configuration signals failed.

```
//(1)constant assertion
always@*
begin
assert_cdc_constant_prop : assert (select === value)
```

The constant values associated to the configuration signals during the setup stage of the static checks are not the same as the ones propagated for the functional dynamic regression. These configuration signals, being responsible for the clocks and resets propagation, are critical and may be the cause of many undetected CDC paths during the static checks.

Conclusion: The "cdc_constant_prop" property failed due to a misalignment between the configuration of the design for both the CDC structural verification and the functional dynamic verification. This means that the design was not configured in its correct functional mode when it was verified for CDC issues. This can risk to either produce a lot of false negatives or to mask an important number of CDC paths.

3. **Failing stability assertions for CDC control:**We can see in Table 3.5 that four assertions related to the stability of the CDC signals synchronized by MFS failed.

```
mfs_stable : assert property
(@(posedge clk_2) disable iff (rst)
$changed (D1) |=> $stable(D1)[*nb_of_cycles]);
```

The failing paths are normal CDC paths synchronized by MFS as shown in Figure 3.7. The metastability in a CDC synchronized by MFS is not avoidable, however, the role of MFS is to delay the signal propagation so the metastability gets resolved before reaching the downstream logic. In the timing diagram in Figure 3.8, a metastability appears on "Qf" in cycle 2 due to a setup violation on "Q1" and "clk_2". In the same cycle, the metastability is resolved to zero, which does not correspond to the correct value on "Q1". That is called a slow resolution. In the case of a slow resolution, in order to avoid data loss, the signal at the source side should remain stable so that it could be re-sampled on the next clock edge, as seen on "Q1". "Q1" remains stable and is re-sampled in cycle 3 (on the edge "e" of "clk_2"). If we imagine a source clock faster than the destination clock, so "Q1" changes its value before cycle 3, the data risks to be completely lost in case of a slow resolution. That is why the four "mfs_stable" assertions failed.

Conclusion: The four "mfs_stable" assertions being failed have shown a true risk of data loss due to a fast to slow crossing while the data does not remain stable. This is the first true CDC bug we found by the semi-formal verification and that could not have been discovered by the structural verification alone.

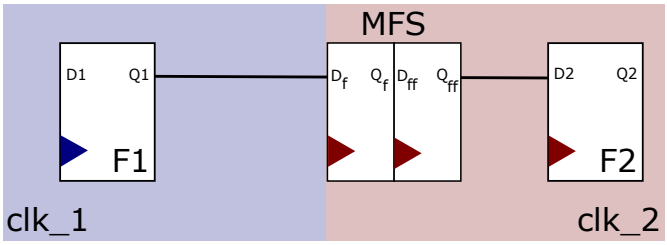


Figure 3.7: Failing mfs stability assertion

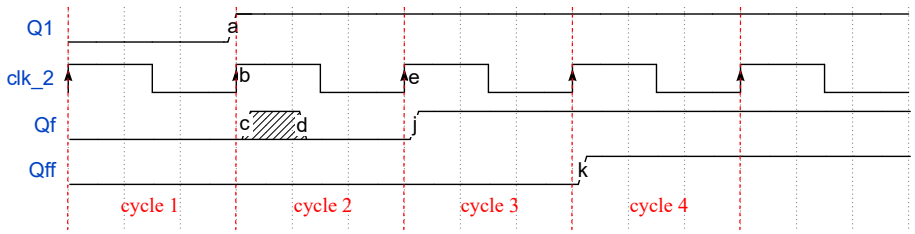


Figure 3.8: Metastability slow resolution

4. **Failing CDC data signals assertions:** We can see in Table 3.5 that 29 assertions related to the stability of the CDC data signals synchronized by qualifier failed.

```
dataqualifier_stable : assert property
(@(posedge clk_2) disable iff (rst)
ctrl |=> $stable(Q1));
```

The assertion above is generated for each CDC data path, under the hypothesis that a CDC data signal should remain stable if its qualifier is enabling. Analyzing the 29 failing assertions, we noticed that the data have effectively toggled while the "ctrl" signal in the assertion above was enabling. This should be a true design bug, except that the case was way more complicated. All the concerned paths were not synchronized by a qualifier. However, they were inside an asynchronous FIFO controlled by a handshake protocol as shown in Figure 3.9. The static tools, trying to match very specific patterns, usually fail to detect the complex synchronization protocols. The compromise the tools usually do is that the algorithm back-traces the destination flip-flop of a complex CDC path till the first MFS synchronizing a control signal and report the latter as a qualifier. Consequently, this detected qualifier replaces the "ctrl" signal in the "dataqualifier_stable" assertion. In our concrete example, as shown in Figure 3.10, the tool failed to detect the synchronization protocol. Oversimplifying the path, and back-tracing the destination "dest", the tool reports the first MFS synchronizing the write address "wrt_add[n-1:0]" as the qualifier. Replacing the "ctrl" in the above assertion by the re-synchronized "wrt_add[n-1:0]", the assertion fails each time the write address points to the highest address (when it is all ones). But in fact this is a false failure, as the "wrt_add[n-1:0]" cannot replace the "ctrl" in the above assertion. In other words, the assertion above is not suitable for this CDC type which the tools failed to well categorize it.

Conclusion: The static tools, trying to match very specific patterns, fail to detect the complex synchronizers. Instead, they oversimplify the CDC path detecting wrong qualifiers. The 29 failures we got are false failures because of the wrong qualifier specification in the "dataqualifier_stable" assertions.

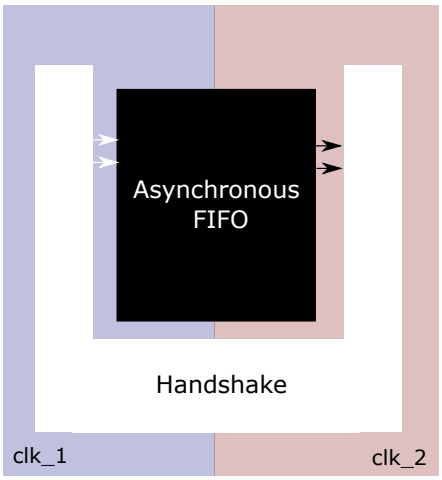


Figure 3.9: Asynchronous FIFO controlled by Handshake

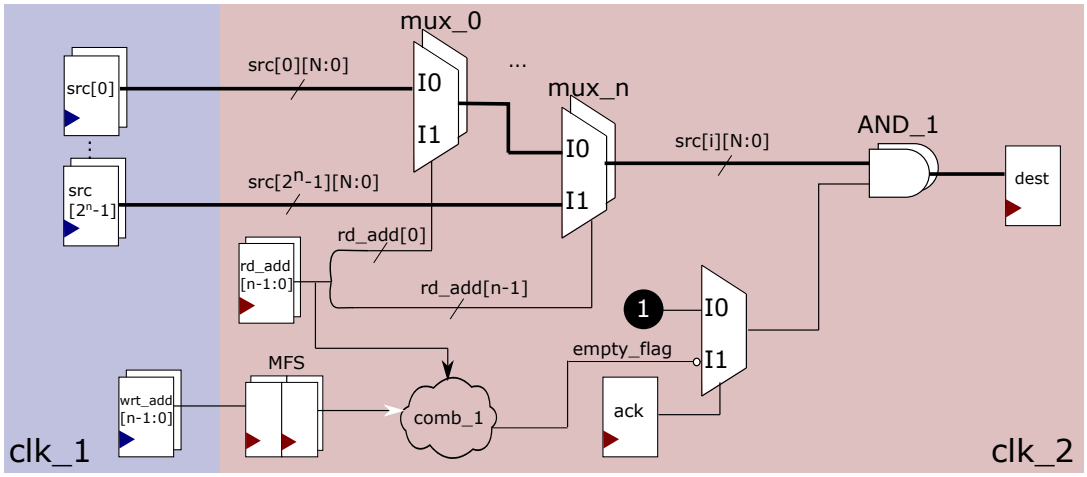


Figure 3.10: CDC path synchronized by asynchronous FIFO controlled by handshake

Overall assessment

The overall assessment of the complementary semi-formal verification for CDC shows the importance of adopting this new verification methodology to complement the CDC structural verification, which is the only approach adopted as sign-off methodology nowadays. Regarding the analysis of the violations detected on our test case, we concluded the following (see Table 3.6):

1. The analysis of the "mutex" failing assertions has shown that the structural tools have a remarkable deficiency in clustering the different CDC paths by synchronizer type. The structural tools were tricked by the presence of the two flip-flops in the destination of the CDC paths and ignored analyzing the presence of a qualifier. The result was the generation of the "mutex" assertions, which were not relevant.

2. The analysis of the "constant" assertions has shown a deficiency in the structural verification flow, specially in the setup generation step. The setup, in terms of configuration signals, not being aligned with the configuration of the design for the functional verification, can be responsible for many false negatives or even hidden CDC.

3. The analysis of the "mfs_stable" assertions has has shown a true CDC bug that cannot be detected by the structural verification. Despite the presence of a valid MFS, the CDC path can still suffer from data loss if the crossing is from a fast to a slow clock domain.

4. The analysis of the "dataqualifier_stable" assertions has shown a deficiency in detecting and analyzing the CDC data complex synchronizers. The tools, trying to match very specific patters, fail to analyze the data syncrhonizers. The result was the generation of irrelevant and wrong assertions that led to multiple false failures.

The work done above was the first step to push the limits of this new methodology. In this context, and with a close collaboration with the EDA providers, the tools' bugs were reported and many enhancement were done on the market's tools thanks to our study.

Challenges: Cross-tools flow The application of this new methodology was not straightforward as it can seem to be. We faced a lot of challenges, specially in what concern the industrial constraints we should respect working at STMircorelectronics. We have to highlight that the CDC verification and the dynamic functional verification are two separated and unrelated tasks in an industrial context. Merging both of them meant merging different tools coming from different providers (as the company has already its own sign-off tools). We had to deal with this and make the tools work together. This explains the assertions coverage limitations shown in Table 3.6. The generated assertions did not cover 100% of the detected synchronizers. As the assertions were generated by the CDC tools, some of the generated assertions were incompatible with the simulation tools due to the hierarchical paths naming. The different EDA providers do not give the hierarchical paths the same names. As Accellera members, we pushed this problem, in the context of the CDC working group that aims to define a new IEEE standard for CDC model, in order to standardize the hierarchical paths naming so they can be reusable across the different tools.

Advantages: True CDC bugs detected Applying the semi-formal flow, we were able to detect true CDC bugs that cannot be detected with the structural verification.

Development: research subjects We identified two areas that need to be developed to push the limits of the semi-formal flow. The first subject is to overcome the tools limitations analyzing a complex synchronizer. In order to succeed to analyze all the types of synchronizers and to generate the relevant properties for them, we should be able to either detect all the customizations of all the CDC synchronizers, or to develop one universal method able to analyze all of them regardless of their type. This will be the subject of section 3.3. The second development areas is the alignment of the design configuration between the CDC structural verification and the dynamic functional verification. To adopt the semi-formal flow, we have to be sure that the design is configured the same way as the functional verification, in terms of operating mode and hence the propagated clocks and resets. This is the hybrid flow developed in section 3.4.

	#constraints/synchronizers		**#Assertions**	**#passed**	**#failed**	**Conclusion**
Constraints	Static signals	400	400	400	0	none
	Exclusive signals	59	59	46	13	tools' bug
	Constant signals	169	169	145	24	flow bug
Synchronizers	Control signals synchronizers	302	150	146	4	CDC bug
	Data signals synchronizers	115	90	61	29	tools' bug

Tab. 3.6: Results and conclusions

3.3 Verification of protocol-based synchronizers

The Multi-flop synchronizers (MFS), responsible mainly for synchronizing scalar control signals, are easily detected by the static tools. However, the detection is more problematic when it comes to data synchronizers. Static verification tools, trying to match very specific patters, can detect standard configurations of the different data synchronization protocols, but struggle with the minimum variation that makes it slightly different to what was pre-coded. Consequently, either the tools fail completely to detect the synchronizer producing a number of false negatives, or oversimplfy and partially detect the synchronizer, which can produce a number of false positives. In both cases, the results of the static tools regarding the data synchronizers are not reliable and we cannot count on them to generate reliable properties. This problem manifested in the failure we had in the "dataqualifier_stable" assertion, which was a false failure.

In this section, we propose the philosophy of the "Universal Qualifier", a new approach able to overcome the limitations of the rigid pre-coded patterns and to effectively verify all the data synchronizers regardless of their types. Thanks to the "Universal Qualifier", we will be able to generate reliable properties for CDC data paths without the necessity of detecting specifically their synchronizers type.

3.3.1 Protocol-based synchronizers

The protocol-based synchronizers are, in general, all the synchronizers that depend on a/multiple control signal to control the data crossing between the source and the destination [93] [76].

Recap on data synchronizers

- **Simple qualifier:** The simplest protocol is based on the use of a control signal called a qualifier. The sampling of the data in the destination domain is enabled thanks to the qualifier. When the data is not stable, the sampling is forbidden. Figure 3.11 illustrates this structure. A data coming from the register "R_s" clocked by "clk_src" crosses the clock boundary to the destination domain clocked by "clk_dest". The qualifier "ctrl" is initiated in the source clock domain, synchronized by a multi-flop synchronizer "MFS" and stops the incoming data on a blocking gate "MUX". The multiplexer "MUX" enables the sampling of a new data when this latter is stable.

  The absence of metastability is ensured by the property *stab_simple*:

  $$\begin{aligned} &\mathit{stab\_simple} : \textbf{assert property } @(\textbf{posedge } \mathit{clk\_dest}) \\ &\mathit{ctrl} \mapsto \textbf{\$stable}(\mathit{d\_in\_mux}) \end{aligned}$$

- **Handshake:** The previous protocol can be extended to a 4-phase protocol like the handshake protocol. In this case, a second control signal is added from the destination to the source domain to acknowledge the correct data reception. In Figure 3.12, the request signal "req" is synchronized by "MFS2" and used with the acknowledge "ack" to calculate the enable signal of the destination to propagate the data. On the other hand, the acknowledge "ack" is synchronized by "MFS1" and

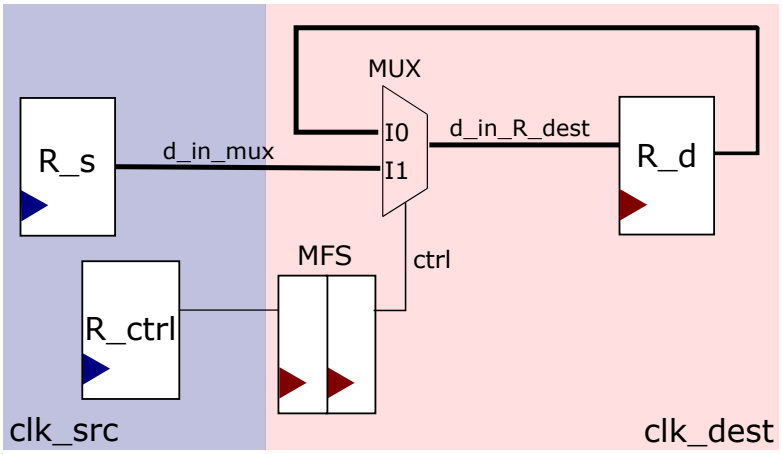


Figure 3.11: CDC synchronized by a re-circulation mux

used with the request "req" signal to calculate the enable of the source flip-flop to capture the data.

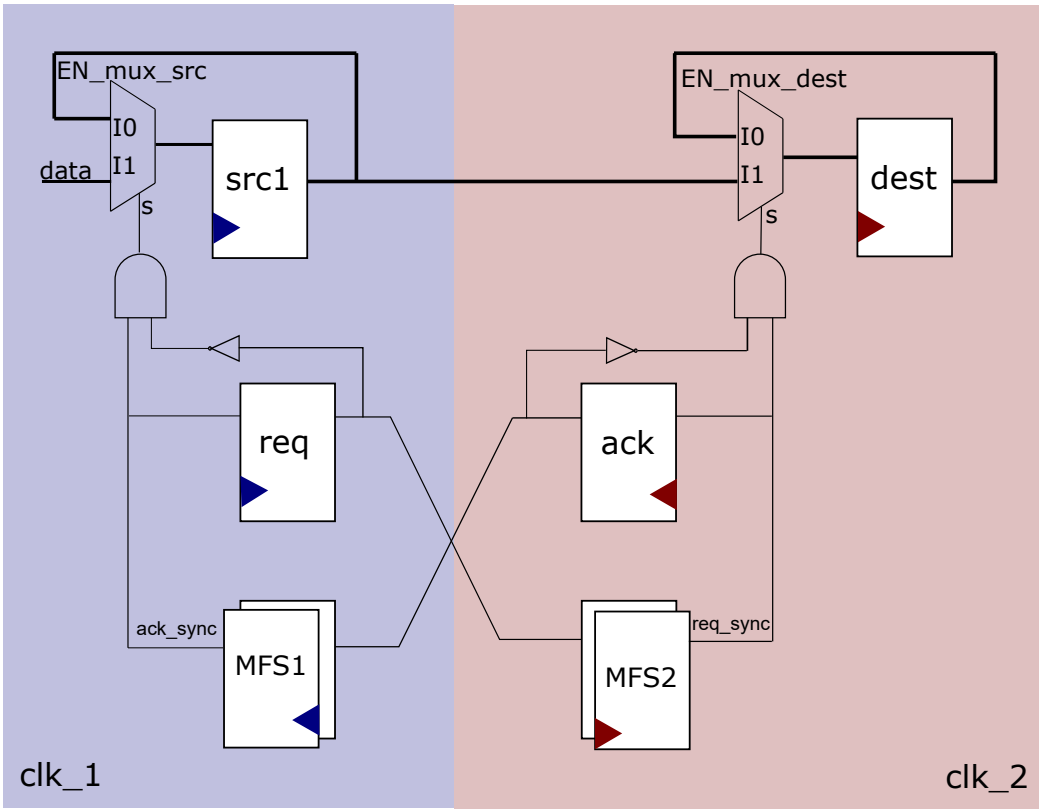


Figure 3.12: CDC synchronized by handshake

- **Asynchronous FIFO:** The previous two protocols are very simple and are not intended for designs where the source clock is running faster than the destination clock. In this case, we prefer to use an asynchronous FIFO. Similar to a synchronous FIFO, it is controlled by four control signals (see Figure 3.13): *write, full, read, empty*. On the source domain side, the input signal "write" indicates that a data "d_in" is ready to be written in the FIFO if signal "full" is not enabled. On

the destination domain side, the input signal "read" indicates that a data "d_out" is ready to be read from the FIFO if the signal "empty" is not enabled. Addresses where the data must be read "rd_addr" or written "wt_addr" in the memory " MEM" are Gray encoded. They are synchronized by multi-flop synchronizers and allow to calculate the different control signals. The addresses are Gray encoded. The read data is sent to the register "R_d" enabled by the control signal "empty_flag".

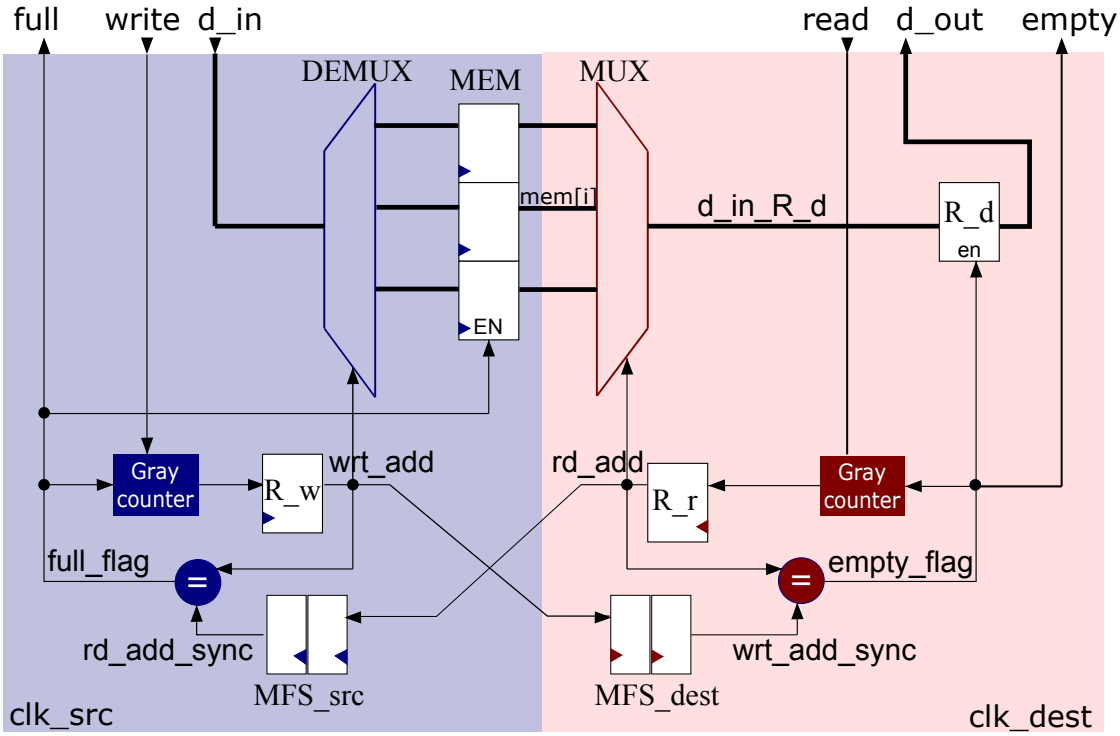


Figure 3.13: CDC synchronized by Asynchronous FIFO

Problem statement

Static verification tools can detect standard configurations of the different synchronization protocols based on the pre-coded patterns and generate a set of assertions for each of them. However, the complex and the custom protocols are harder to detect. To compromise, the tools partially detect the data synchronizers by back-tracing the destination register till the first MFS. It reports its associated signal as a control path that is usually not equivalent to the real Boolean expression that enables the data when stable and blocks it otherwise. Table 3.7 shows the difference between what the static tools are reporting and what we expect as qualifier control for some of the known synchronization protocols. For the simple qualifier shown in Figure 3.11, the tools detect a CDC without MFS. To analyze such a CDC, the tools back-trace the destination flip-flop "R_d" to the first MFS, which is the signal "ctrl" in this case. "ctrl" is then reported as the control signal of the CDC path between "R_s" and "R_d". The same concept is applied to any CDC without MFS that the tools fail to categorize into a known synchronizer category. For the handshake protocol in Figure 3.12, back-tracing the destination "R_d", the first encountered MFS is the re-synchronized request signal "req_sync". The tools report "req_sync" as the control of the CDC path between "R_s" and "R_d". The same concept is applied to the asynchronous FIFO in Figure 3.13 and teh "wrt_add_sync" is reported as the control signal.

Synchronizer	Control reported by the tools	Control able to blockenable the data
Simple qualifier	ctrl	ctrl
Handshake	req_sync	(req_sync).(¬ ack)
Async. FIFO	wrt_add_sync	(rd_add).(¬ emapty_flag)

Tab. 3.7: Tools' detected qualifier vs. the true control expression

To conclude, a classic synchronization protocol with a slight change cannot be detected and recognized as synchronizer by the commercial tools. The tools compromise by back-tracing the destination flip-flop till the first MFS and reporting this latter as a control.

3.3.2 The Universal Qualifier

If we look more openly to any CDC data synchronizer based on protocol, we can notice that all of them are working with the same concept. Synchronization protocols involve a number of control signals that work together in harmony to ensure that the asynchronous data will never cross the source clock domain boundary unless it is stable. The number of control signals and how they are harmonically working together may differ between one protocol and another, but the concept remains the same.

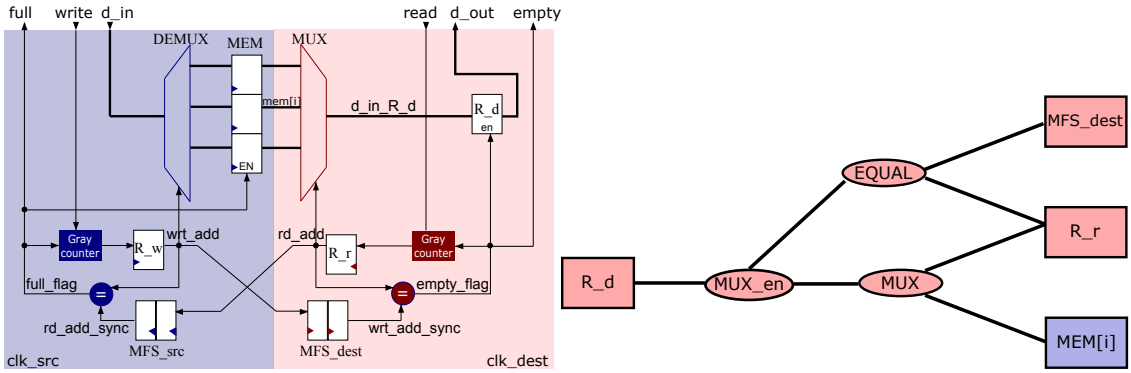
For example, in the asynchronous FIFO, the tools select the signal "write_addr_sync" connected to the output of the multiflop "MFS_dest" as a qualifier, but do not take into account the other control signals occurring on the data path (for example, the signal "rd_add" that selects the data to be read). The properties generated by the different tools are not satisfied in the simulation. In this section, we propose a generic method to automatically detect the correct combinational expressions to be used as qualifiers.

3.3.3 Generic CDC modeling for data stability verification

Our method is based on the detection of all the control signals on the data path and the construction of a combinational expression based on these control signals.

Design Modeling

The set of gates involved in a CDC is the set of gates present in the cone of influence of the destination register cut on each branch at the first sequential elements. For example in the Asynchronous FIFO, for *R_d*, the set of elements are {*R_d, MUX, MEM, Equal, MFS_dest, R_r*}. It can be modeled by a directed graph $T = (V, E)$ where V, the set of vertices, represents the set of gates, and $E \subset V \times V$, the set of edges represents the set of wires connected to the gates. Figure 3.14b illustrates the CDC tree of register *R_d*. The root is *R_d* and the leaves are *R_r, MFS_dest, MEM*. The node *MUX_en* corresponds to the MUX associated with the port enable of the register *R_d*. We call *CDC data path*, a path of T from the root to a leaf representing a register in the source domain. Other paths are called *CDC control path*. For example, in the asynchronous FIFO (*R_d, MUX_en, MUX, MEM*) is a CDC data path, and (*R_d, MUX_en, EQUAL, MFS_dest*) is a CDC



(a) Structure *(b) Tree*

Figure 3.14: CDC synchronized by Asynchronous FIFO

control path. We call *internal node* a node that is neither a root, nor a leaf. Focusing only on CDC data paths (as CDC control paths are synchronized simply by MFS), the correct synchronization of these paths depends on the presence of qualifiers. The only internal nodes that are allowed on the data path are the four Boolean gates: NOT, OR, AND, and MUX. Let $p = (p_n, \ldots, p_0)$ be a CDC data path of depth $n > 1$, where p_n is the root of the tree (a flip-flop in the destination domain), and p_0 is a leaf (a flip-flop in the source domain).

- *1-input Gate:* the input of a gate p_i is the edge (p_i, p_{i-1}), it is denoted $d\_in_{p_i}$.
- *2-input Gate:* One input is the edge (p_i, p_{i-1}). It is denoted $d\_in_{p_i}$. The second one is denoted $ctrl_{p_i}$ since it is an edge of a CDC control path.
- *3-input Gate:* This gate is a MUX. Two of its inputs are edges of a CDC data path. The edge (p_i, p_{i-1}) is denoted $d\_in\_1_{p_i}$ and the other one $d\_in\_2_{p_i}$ (edge of an other CDC data path). The last one is denoted $ctrl_{p_i}$ because it is an edge of a CDC control path. This can be extended to any multiplexer.

Extraction of the qualifier

The qualifier of the path $p = (p_n, \ldots, p_0)$ is a combinational expression of control signals such that $d\_in\_p_1$ (output of the source register) drives the value of $d\_in\_p_n$ (input of the destination register) when it is asserted. Let α_g be the absorbing value of a gate g. The qualifier expression of this path of depth $n+1$ is given by

$$qual_n = \bigwedge_{1<i\leq n-1} (ctrl_{p_i} \neq \alpha_i)$$

Proposition 3.3.1. *If $qual_n$ is enabled, the signal d_in_p_n ingoing the destination register is driven by the signal d_in$_{p_1}$ outgoing the source register. Their values are equal.*

Proof. The proof is done by induction on the depth of the path. Figure 3.15 illustrates the base case with a unique combinational gate. In this case, the qualifier is defined by ctrl_AND. If it is asserted, signal d_in_R_d is equal to d_in_AND. This can be generalized to any gate, using the absorbing value.

Let us now assume that the proposition is true for any path of depth n. Let us prove it for a path of depth $n+1$. The proof is very similar. If $ctrl_{p_{n-1}} \neq \alpha_{n-1}$ is asserted, $d\_in_{p_n}$

is driven by $d\_in_{p_{n-1}}$. Thus, if $ctrl_{p_{n-1}} \neq \alpha_{n-1}$ and $qual_{n-1}$ are asserted, then $d\_in_{p_n}$ is driven by $d\_in_{p_1}$. This concludes the proof. □

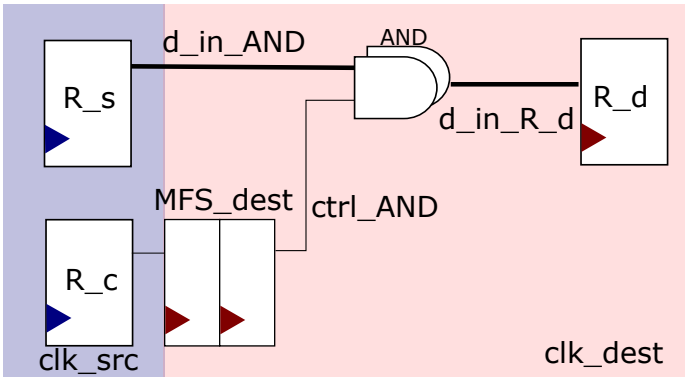


Figure 3.15: Logical AND as a CDC data blocking gate

Absence of metastability

To avoid the metastability in R_d, we must ensure that *d_in_R_d* is never sampled when toggling, or in other words that it toggles outside the setup and hold window of R_d. For this, we define for all CDC data paths *p*, a property that ensures the stability of the data signal outgoing the source register when the qualifier is asserted.

$$stab\_p : \textbf{assert property } @(\textbf{posedge } clk\_dest)$$
$$qual_p \mapsto \textbf{\$stable}(d\_in_{p_1})$$

The absence of metastability in the destination register is given by the following proposition.

Proposition 3.3.2. *Let P be the set of m CDC data paths* $\{p^0, \ldots p^m\}$ *that have a common root denoted* R_d *(i.e.,* $\forall i, p^i_n = R_d$*). If all the properties* $stab\_p^i$ *are satisfied, then register* R_d *can not be metastable.*

Proof. To avoid metastability, signal $d\_in_{p_n}$ ($d\_in\_R_d$ in Figure 3.15) must not be sampled when it toggles. Let us select a path p^i and let us assume that $qual_{p^i}$ is asserted. Signal $d\_in\_R_d$ is driven by *d_in_p_AND* (Proposition 3.3.1) which is stable by property $stab\_p^i$. Signal $d\_in\_R_d$ may toggle only on the rising edge of $qual_{p^i}$ (*ctrl_AND* in Figure 3.15). Since $qual_{p^i}$ is defined in the destination domain, it respects the timing constraints and avoids hold and setup violation. Therefore, the toggling of $d\_in\_R_d$ respects the timing constraints. The same reasoning applies on all paths. We conclude that $d\_in\_R_d$ may always be sampled in the destination domain. □

3.3.4 Implementation

To avoid the metastability, it is sufficient to verify all the properties $stab\_p^i$. We have implemented two algorithms that instantiate these properties. Algorithm COMPUTE_ALL described in Figure 3.16 scans all the CDC paths in a design $\mathscr{C}$, and extracts the CDC data paths and their respective control signals. It returns a dictionary of qualifiers. This latter then replaces $qual_p$ in the *stab_p* for all the CDC data paths in the design $\mathscr{C}$.

The procedure EXTRACT_QUALIFIER computes the qualifier expression of a given path. It first initializes the qualifier, then depending on the gate type, iteratively performs the conjunction with either the control signal or the negation of the control signal.

```
1:  procedure EXTRACT_QUALIFIER(p)
2:      qual = 1                                          ▷ Qualifier initialisation
3:      n =| p |                                          ▷ length of p
4:      for 0 < i < n do
5:          if p_i ∈ internal node then                   ▷ p_i: ith gate in p
6:              if p_i = OR then
7:                  qual = qual ∧ ¬ctrl_{p_i}             ▷ ctrl_{p_i}: control sig. of p_i
8:              else if p_i = AND then
9:                  qual = qual ∧ ctrl_{p_i}
10:             else if p_i = MUX then
11:                 if p_{i-1} = d_in_1_{p_i} then
12:                     qual = qual ∧ ctrl_{p_i}
13:                 else                                  ▷ second data input of MUX
14:                     qual = qual ∧ ¬ctrl_{p_i}
        return qual

15: procedure COMPUTE_ALL(𝒞)                              ▷ 𝒞 is the design
16:     𝒫=extract_CDC_path(𝒞)                             ▷ 𝒫 is the set of CDC paths in 𝒞
17:     𝒫_data=extract_CDC_data_path(𝒫)
18:     for all p ∈ 𝒫_data do
19:         qual[p] =EXTRACT_QUALIFIER(p)
        return qual                                       ▷ qual is an array of qualifiers
```

Figure 3.16: Algorithms to extract the universal qualifier

3.3.5 Results

We have applied our method to our commercial 64-bit dual-core CPU subsystem. This subsystem has fourteen asynchronous clock domains, nine types of standard asynchronous interfaces, and three power domains. Since the asynchronous interfaces are standard, they have been fully formally verified stand alone. Furthermore, we dispose a set of 103 different high quality tests to perform the functional verification by simulation: they cover 100% of the code and 100% of the functionality of the design. The object of our study is to compare the quality of the qualifiers generated by our method and those generated by a commercial CDC verification tool. Both methods generate assertions, that are checked in simulation. Since the design is correct, no assertions should fail. If an assertion fails, it means that the assertion was not correct, or more accurately that the qualifier was not correctly defined.

CDC data path extraction The CDC paths are extracted using the commercial tool. It covers 100% of the CDC data paths present in the design. It reports 5.8k CDC data paths: 4.2k are synchronized by a simple qualifier and 1.6k paths are synchronized by other complex protocols.

Qualifier extraction On each CDC, we extract a qualifier, first with the commercial tool, then with our generic method (extraction time is comparable and negligible in both

methods). We instantiate Assertion *stab* with each set of qualifiers. We get two sets of 5.8k assertions, one from each method, that are bound to the design. Then, we run the 103 tests with all the assertions.

Table 3.8 shows the results we obtain. It compares the results obtained by the commercial tool (tool_qual column) and our generic method (generic_qual column). We note that with the commercial tool, 89 tests fail out of 103 (at least one assertion fails). In contrast, our method never fails. The next question is what is the number of commercial assertions that fail: only 1.6k fail at least once. These assertions all correspond to the qualifiers generated for the complex protocols (handshake and asynchronous FIFO). The assertions generated for simple protocols are satisfied (as expected). This shows that the generation of qualifier proposed by the different commercial tools for complex protocols is not correct.

Tab. 3.8: Results

# CDC path and assertions: 5.8k	**tool_qual**	**generic_qual**
# Failing test	89	0
# Failing assertions	1.6k	0

3.4 Hybrid flow

The hybrid flow is our proposal to solve the problem of misalignment between the CDC verification and the dynamic functional verification developed in section 3.2.2. While we were applying the CDC semi-formal flow for the first time, we noticed 24 assertions, related to the constant configuration signals, have failed in all of the tests of the functional regression. This has shown that the functional verification and the CDC verification are not aligned in what concerns the design operating mode. This can have serious drawbacks on the quality of results of the CDC verification as it is reflected directly on the propagated clocks and the number of analyzed CDC paths. The hybrid flow we propose in this section is about using the semi-formal verification, not only to complement the CDC structural verification, but also to assist in the setup phase. The idea is to include the constraints assertions early in the dynamic functional verification and the CDC verification flow to reach at early stage a full alignment between both of them. In the following section, we will justify the necessity of adopting this flow, explaining what an operating mode is and how the constant configuration signals play an important role to define it.

3.4.1 Clock propagation and operating modes

The idea of the hybrid flow has not only emerged following the failures we have found applying the semi-formal flow for the first time. A number of other problems, related to the clock signals specification and propagation, that we used to face earlier in the flow,

can be solved applying this hybrid flow. The setup violations due to clock propagation are usually counted in thousands and it could take weeks or even months to analyze them.

Clock configuration

The first step in any verification process is to setup the design. This helps the verification tools to analyze the system in a realistic context. For a multi-clock design, in the context of the CDC verification, providing a correct configuration of the clock signals is a crucial setup factor. The primary clock signals in general are propagated from the primary ports to the sequential elements of any design. Between these two, designers develop complex control logic to manage the clocks propagation and the operating modes. The operating mode depends on the required performance and the functionality provided by each mode. The clock frequencies are then adjusted in an optimal way using, what so called, the configuration and the transformation logic [94] [95]. On a clock tree, there exists four common operations that can be done on clock signals [96] [97]:

- **Selection:** choosing between one clock and another ("clk mux." in Figure 3.17)
- **Blending:** merging two clock signals to generate a new one.
- **Gating:** enabling to stop the clock ("clk gt." in Figure 3.17)
- **shaping:** modifying the clock frequency or duty cycle ("clk div." in Figure 3.17)

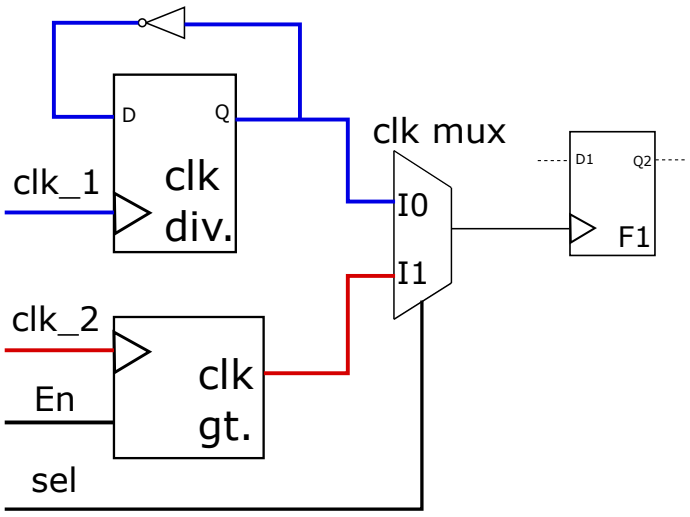


Figure 3.17: Clock network logic

A clock network is constituted of two main agents: clocks ("clk_1" and "clk_2" in Figure 3.17) and configuration signals ("En" and "sel" in Figure 3.17). Clocks are either "primary clocks" when they come from a primary input of the design or the output of an analog module (*e.g.* PLL seen as a black box), or "generated clocks" whenever they undergo one of the four previous operations (selection, blending, gating or shaping). A "generated clock" can be derived from a "primary clock" (*e.g.* the output of "clk div." or "clk gt."), or it can be derived from another "generated clock" (*e.g.* the output of "clk mux"). The configuration signals control the operations and the propagation of the clocks ("En" determines whether "clk_2" will be propagated or not, "sel" chooses between "clk_1" and

"clk_2" the clock that will drive "F1"). Let us assume that "clk_1" is a functional clock and "clk_2" is a test clock. In this case, "sel" determines the operating mode in which the system here will operate, whether a functional mode (in case "sel" is equal zero and propagates "clk_1") or a test mode (in case "sel" is equal one and propagates "clk_2"). Thus, we define an operating mode being the set of configuration signals being set to some constant values. An operating mode is simply a vector of configuration signals that propagate some clocks and block others [98].

When undertaking CDC structural verification, engineers often have the choice between two distinct approaches: the multi-modal approach and the incremental multi-mode approach. The multi-modal approach involves the concurrent propagation of all clocks, while the incremental multi-mode strategy performs multiple CDC verifications for each operating mode. The multi-modal approach concurrently propagates all clocks without constraining configuration signals. This method, however, results in a design state that is overly pessimistic and unrealistic, generating noise and false negatives in the returned results. Despite these drawbacks, the advantage lies in achieving 100% coverage of all CDC paths. On the contrary, the incremental multi-mode approach adopts a more realistic strategy by conducting multiple CDC verifications for each operating mode. Here, each configuration of the configuration signals is treated as a scenario. Nevertheless, a significant challenge arises from the absence of a robust design specification that would enable engineers to precisely define these scenarios.

Lack of design specification

The operating modes and their correspondent clocks and configuration signals are defined in each design specification. The CDC verification engineers should then rely on this specification to setup the design and to verify it in a realistic mode. This is ideally what should happen, but that usually does not. As the verification flow starts in parallel with the design flow, and because of the reuse of modules with incompletely documented legacy components, the design specification are usually not available. To address the absence of a comprehensive specification, CDC engineers often find themselves devising workarounds. CDC static tools, proficient in detecting configuration signals, generate a template for the engineers to utilize in defining the design operating mode. However, the direct implementation of this template is hindered by the inherent lack of a detailed design specification. Consequently, verification engineers opt for a third approach, meticulously analyzing the clock tree on a case-by-case basis. This allows them to configure it in a manner they perceive as the most pessimistic from a CDC perspective. The subsequent section delves into the contemporary flow adopted for configuring the clock tree.

Method A: Manually defined configuration

Instead of modifying the configuration signals template, a setup check is run propagating all the clocks at the same time. This is called a "multi-modal" run. Consequently, the tools report thousands of "clock overlap" violations. This violation tag is reported whenever a configuration signal is not defined. For example, in Figure 3.17, if "sel" is not set to a constant value, the tools will report a "clock overlap" violation on the output of "clk mux" and will propagate a composite clock asynchronous to both clocks on its inputs. The violations are then analyzed one by one to configure the configuration signal related to each of them (refer to Figure 3.18). The configuration signals are set as follows :

1. The configuration signal should prioritize the functional clock in case of a choice between a functional and a test clock.

2. The configuration signal should prioritize the faster clock in case of a choice between two functional clocks

This process puts the design in a pessimistic mixed-mode, where controls propagate the clock signals incoherently and unrealistically. Considering the number of clock gating structures in modern designs, along with complex clock switching, having this non determinism of the propagated clocks can have serious consequences on the verification results. Beside the number of false CDC violations we encounter usually due to this unrealistic mode (results are very noisy), some configurations may be missed and can lead to leaving behind critical CDC paths unanalyzed. In addition, the process of "trial and error", where the violations are debugged and the configuration signals are defined one by one, takes sometimes longer than accepted in the projects road-map. This can also have a drawback if the constraints are re-used as assumptions for the CDC formal verification. In this scope, we can quote from [43] the following:

"*After writing the correct assertion, a model checker takes 13 seconds to guarantee that the FIFO cannot overflow. We then remove the assumption on the clock-gate controls, hence making them fully non-deterministic. Functionally, the clock can be enabled or disabled at any time. The same overflow property now takes 192 seconds to be proven (15 times more). The reason for this runtime increase is an explosion in the design state-space.*"

We can conclude that it is important to define the operating mode in which the design is verified. Also, several configurations can be iteratively verified if the design operates in several modes. Verifying the design in a mixed mode leads to noisy and unrealistic results and can mask some other critical problems.

3.4.2 Semi-formal assisted CDC setup generation

In order to overcome the limitations mentioned above, we propose the hybrid flow where the semi-formal verification can assist the setup generation phase for the structural verification. A closed feedback loop can be developed between the CDC verification and the dynamic functional verification during all the project's lifetime. This can ensure a continuous alignment between both of them.

Noticing results on Table D.1, the constant configuration assertions are 100% covered in every test of the functional regression. In addition, noticing the graph in Figure 3.4, the constant assertions are 100% covered even at the lowest code coverage. This is not a coincidence. If the design has one functional operating mode, every test in its functional regression will be configuring the design in this unique functional mode (the case of our CPU subsystem). On the other hand, if the design has multiple operating modes, each test will give the full definition of at least one operating mode. This means that only one test among the functional regression will be enough to define one operating mode for the CDC verification, and thus defines the configuration signals for this mode. Therefore, we can use the functional verification, even at its early stages, to assist the CDC verification to configure the design and define some of the operation modes.

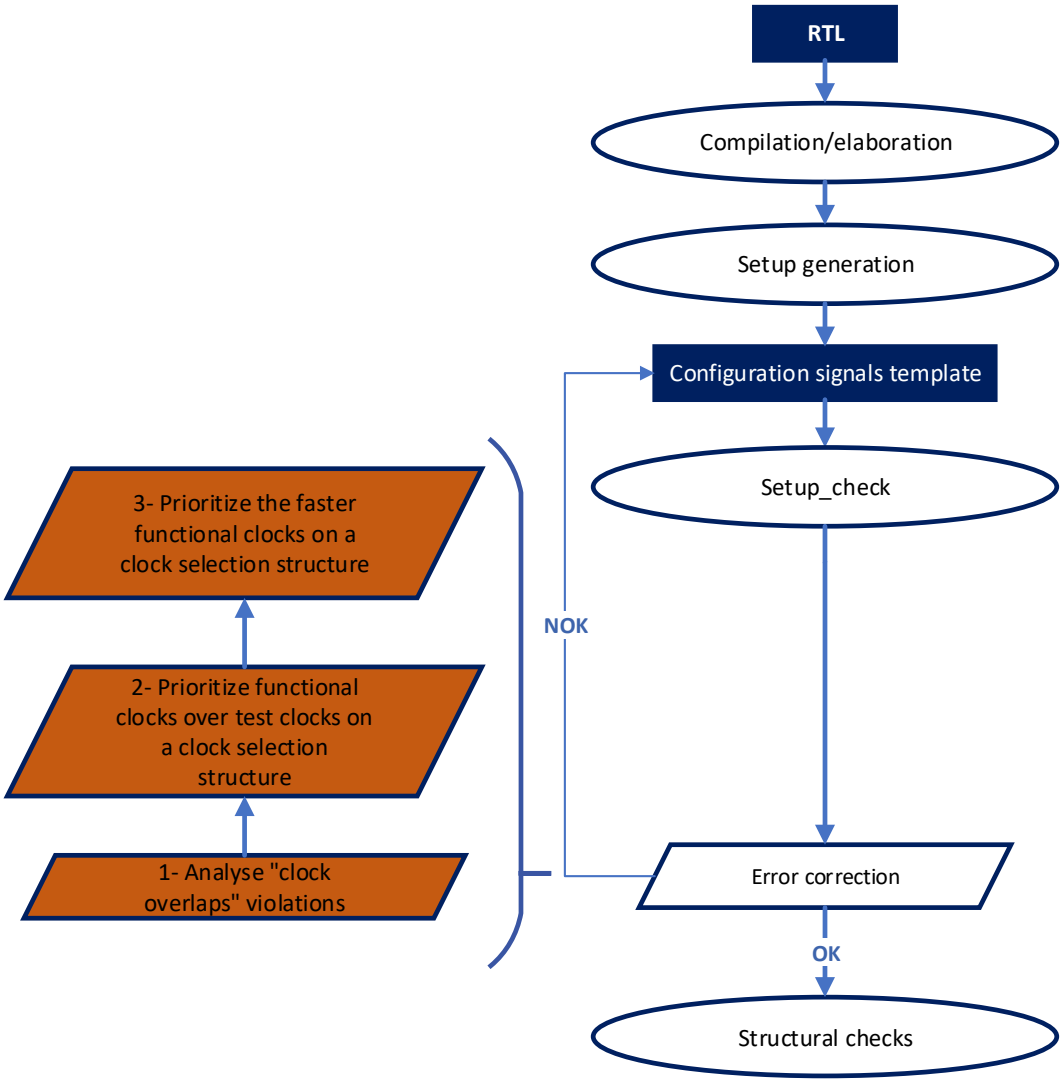


Figure 3.18: Method A: Manually defined setup

Method B: semi-formal assisted configuration

Our proposal is a fully automated flow that relies on generating the configuration signals assertions early in the CDC verification flow. Following the flow in Figure 3.19, after the compilation of the RTL and the generation of the configuration signals template, the configuration signals are all assigned to a same constant temporary value (to one for example). This step is mandatory to generate the constant assertions with an assigned value. The assertions are then bound on the design's top and the functional regression is launched. Launching the functional regression, some of the assertions will certainly fail due to the random assignment done in the previous step. A failing assertion means that the value assigned for the configuration signal in the context of the CDC verification does not match the value assigned for the same signal in the functional verification, which will lead consequently to a different clock propagation and hence to a different operating mode. On the other hand, a passed assertion means that both values match. The simulation log will then be post-processed to identify the correct and the incorrect configuration signals. Our script will use this information to:

1. Eliminate the configuration signals constraints that correspond to the failing assertions in the original "configuration signals template".

2. Create a new complementary constraints file "New configuration signals template" that includes the configuration signal constraints that correspond to the failing assertion, but this time with inverted values.

Now the two files together include the configuration signals values that configure the design in a pure functional mode, 100% conform with at least one functional mode as those defined for the functional verification. Repeating the latter with the evolution of the functional verification tests, multiple configurations may be detected translated into multiple possible scenarios for the CDC structural verification. For each detected scenario, the structural flow continues normally as explained in section 2.2.2.

3.4.3 Results

We applied this flow on our CPU subsystem test case. In our results, we focus on the number of CDC paths the tool was able to see applying each flow. The number of CDC paths reflects directly the verification coverage. The test case was already completely verified using "Method A". This means that all the CDC issues were already fixed and all the design secondary constraints were already in place. The results of this verification are shown in Table 3.9. The native number of CDC detected applying "Method A" was in total 3198 CDC paths. As a first application of the hybrid flow, we just eliminated the configuration signals constraints and we regenerated them using the hybrid flow "Method B", keeping all the other secondary constraints that were in place. Being aligned 100%with the functional mode, the tool was able to detect 15901 CDC paths in total, which means five times the number of CDC paths we were able to detect with "Method A". The difference represent the CDC paths we missed during our first analysis and reflects the low verification coverage our first method held.

The second trial was to re-apply both methods after eliminating all the secondary design constraints. As the secondary constraints affect enormously the CDC paths detection, we thought they should be eliminated in order to avoid any bias in the results. In Table

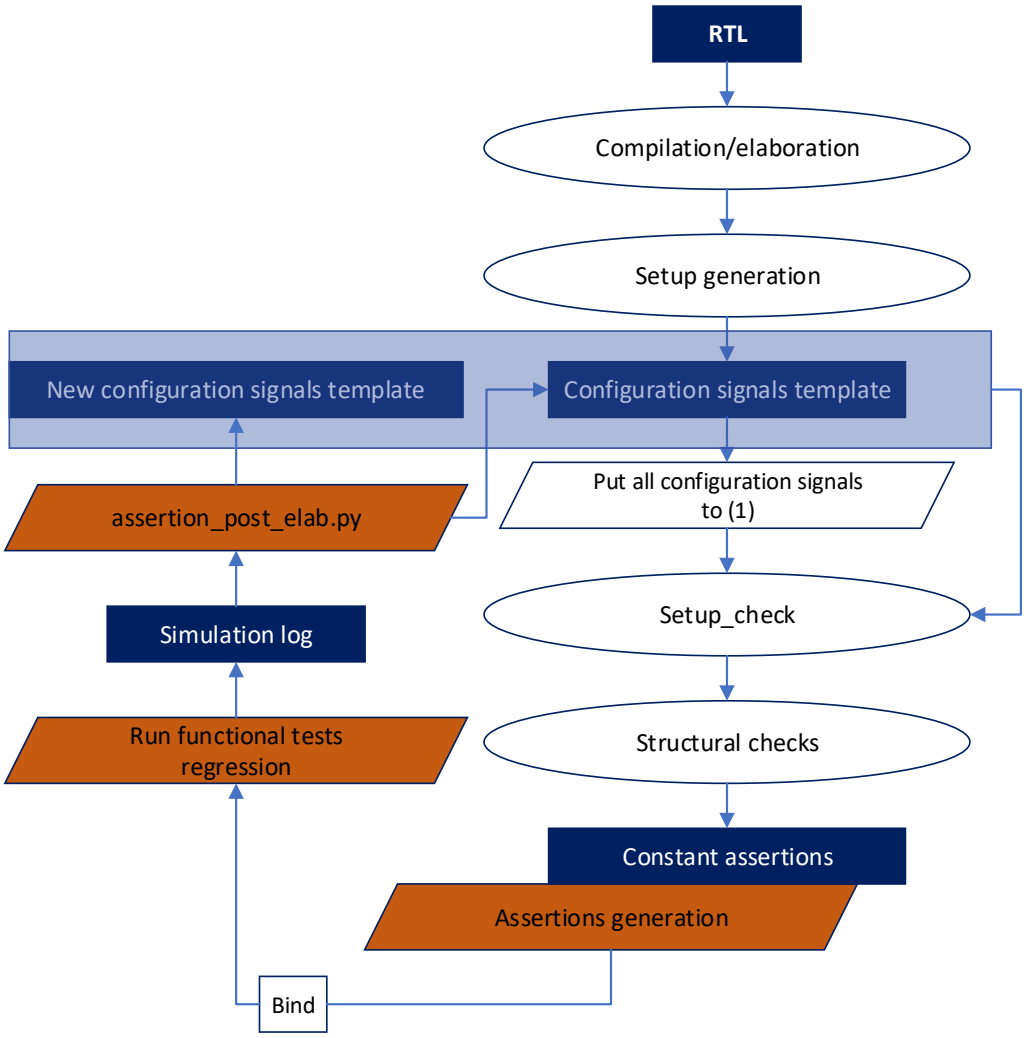


Figure 3.19: Method B: semi-formal assisted configuration

Violations	Method A: Manual configuration	Method B: semi-formal assisted
Clock Overlap	0	0
Clock Constant	0	0
Unsynchornized CDC data	0	47
Unsynchornized CDC control	0	1301
Info	**Method A: Manual configuration**	**Method B: semi-formal assisted**
synchornized CDC data	115	113
synchornized CDC control	302	216
CDC pseudo-static	2781	2513
Total CDC detected	**3198**	**15901**

Tab. 3.9: Method A vs. Method B keeping all the design constraints

3.10, we can notice that the number of CDC paths detected as pseudo-static drops to zero in both method. This is the consequence of eliminating the secondary constraints (including the pseudo-static constraints). Inspecting the number of CDC paths detected by both methods, the conclusion remains the same. Applying "Method A", the tool detected 8536 CDC paths, while applying "Method B" 26759 CDC were detected (three times the number of CDC detected by "Method A").

Violations	Method A: Manual configuration	Method B: semi-formal assisted
Clock Overlap	0	0
Clock Constant	0	0
Unsynchornized CDC data	78	97
Unsynchornized CDC control	7810	25880
Info	**Method A: Manual configuration**	**Method B: semi-formal assisted**
synchornized CDC data	26	95
synchornized CDC control	622	687
CDC pseudo-static	0	0
Total CDC detected	**8536**	**26759**

Tab. 3.10: Method A vs. Method B eliminating the secondary constraints

Finally, as a last trial, we decided to make a "multi-modal" run. In a "multi-modal" run, all the design constraints are eliminated including the configuration signals and all the clocks are propagated concurrently. The results of a "multi-modal" run are the most pessimistic results that we can get. Despite of being full of false negatives, they include

for sure 100% of the CDC paths that exist in a design. The idea was to compare the number of CDC paths detected in Table 3.10 with the number of CDC paths detected by the "multi-modal" run shown in Table 3.11. We can notice that the 28181 CDC paths detected by the multi-modal run is a close number to the 26759 CDC paths detected by method B. Being aligned with the functional mode was more pessimistic in terms of detected CDC paths than the mixed-mode we verified the design in applying "Method A". The number of missed CDC is huge when not aligned to the functional mode.

Violations	**Multi-modal: no constraints**
Clock Overlap	40
Clock Constant	0
Unsynchornized CDC data	97
Unsynchornized CDC control	27676
Info	**Multi-modal: no constraints**
synchornized CDC data	21
synchornized CDC control	987
CDC pseudo-static	0
Total CDC detected	**28181**

Tab. 3.11: Multi-modal run results

Finally, in terms of the time necessary for the specification and the number of used constraints, "Method B" is also more advantageous. Table 3.12 shows that to converge the number of "Clock Overlap" violations, using "Method A", the verification team spent 4 weeks to analyze the verification and to constrain the configuration signals one by one. For that, a total of 169 local constant constraints were used. While using "Method B", it took only 1 hour, thanks to the fully automated flow, to converge the number of "Clock Overlap" violations to zero, using only 46 global constant constraints.

Time/constraints	**Method A: Manual configuration**	**Method B: semi-formal assisted**
Time	4 weeks	1 hour
# constant constraints	169	46

Tab. 3.12: Method A vs. Method B in terms of time and number of constant constraints

3.4.4 Conclusion

For a multi-clock design, in the context of the CDC verification, providing a correct configuration of the clock signals and thus the operating mode is a crucial setup factor. Due to the lack of design specification, this process is usually long and non deterministic and

can result in putting the design in a pessimistic mixed-mode, where controls propagate the clock signals incoherently and unrealistically. Considering the number of clock gating structures in modern designs, along with complex clock switching, having this non determinism of the propagated clocks can have serious consequences on the verification results. In this section we presented our proposal about "The Hybrid Flow". It is about using the semi-formal verification, not only to complement the CDC structural verification, but also to assist in the setup phase to be aligned with the operating modes defined by the functional verification. Applying our fully automated new flow, the results has shown a huge positive gap regarding the CDC coverage applying the new hybrid flow. The new flow detected three times more CDC paths in remarkably less time and using less constraints.

3.5 Conclusion

The CDC structural verification must be complemented by an assertions-based verification to ensure the validity of the setup and of the reported synchronizers. In the first section of this chapter, we explored the CDC semi-formal verification where we took advantage of a mature dynamic functional verification to assert our CDC properties. The semi-formal flow was applied for the first time in order to be assessed and to find the potential development areas. We concluded that by applying our semi-formal, we could, not only achieve 100% coverage of our CDC assertions, but also find new CDC bugs not seen by the structural verification. Given the time the iterations took and the reached coverage, the semi-formal approach for CDC verification can be more advantageous than the formal approach.

In the second section, we presented some proposals to overcome the deficiencies we noticed applying the semi-formal flow. In section 3.3, we propose a generalized method to verify the CDC data synchronizers. Static verification tools can detect standard configurations of the different synchronization protocols based on the pre-coded patterns and generate a set of assertions for each of them. However, the complex and the custom protocols are harder to detect. For that, we presented the concept of "The Universal Qualifier", a generic method able to verify all the CDC synchronizers regardless of their types. The results have shown that applying our method, we were able to generate more reliable properties than the ones generated based on the native results of the industrial tools.

In third section, we presented the "Hybrid Flow" also to overcome one of the deficiencies we noticed in the native flow. The hybrid flow we proposed in this section is about using the semi-formal verification, not only to complement the CDC structural verification, but also to assist it in the setup phase. This gives a solution for the mixed-mode verification approach used due to the usual lack of design specification. The "Hybrid Flow" helped to assume one configuration mode (aligned with at least one of the functional modes verified by the dynamic functional verification) in which the design is verified in CDC. The results has shown a huge gap in the number of the detected CDC paths by the hybrid flow and the mixed-mode flow. This has opened the door to adopt an iterative approach, where several configurations are interatively checked, rather than a "mixed-mode" approach where clocks are propagated incoherently an unrealistically.

4 Metastability injection

This chapter synthesizes our exploration of metastability injection. Initial insights from analog simulations revealed a propagation time prolongation impacting the output Q. Modeling at analog and digital levels led to a conclusive abstraction, interpreting metastability as a delay in the crossing signal. Injecting Metastability in simulations demonstrated its potential to compromise synchronization protocols. Notably, RTL simulations uncovered overlooked re-convergences as well. Despite lacking optimal metrics, the work provides a promising foundation for further exploration and a basis for a comprehensive studies in metastability injection.

Contents

4.1 Introduction

Metastability is a critical issue in digital circuits that can occur when a flip-flop or latch receives a signal that violates its setup or hold time requirements. This can lead to an unpredictable output that can cause data errors and other issues in downstream logic. In multi-clock designs, metastability is one of the most risky aspects, and it is essential to understand how it happens and how to resolve it. In this chapter, we provide a comprehensive presentation of the metastability phenomenon. We begin by exploring metastability on an analog level, conducting analog simulations to gain insights into how it occurs and resolves. We then explain it using a formal model, providing a step-by-step explanation of how metastability is generated and propagated on CDC paths. Finally, we present the concept of metastability injection on RTL and the benefits of applying such a methodology on the quality of results of the CDC verification [99]. By understanding metastability and its impact on digital circuits, designers can improve the reliability and performance of their designs and minimize the risk of data errors and other issues.

4.2 Metastability simulation on analog level

4.2.1 Context

In the realm of the multi-clock systems, understanding and controlling metastability is essential for ensuring the reliability and the stability of the system. Despite of being a pure analog phenomenon that is completely abstracted on digital level, the metastability is a concern that should be resolved at early stages. That explains the different efforts we saw in the state-of-the-art to imitate the metastability effect on digital level. To tackle this subject, our CDC verification team recognized the crucial significance of visualizing metastability on the analog level of the cells used for our CPU subsystem. Witnessing the phenomenon firsthand helped us to have unique insights on how to reproduce it on digital level allowing for more comprehensive understanding of its behavior. Being a digital team at STMicroelectronics, we made a strategic decision to hire a skilled third-year apprentice to tackle the subject of reproducing metastability on analog level using the analog simulations adapted tools and being in contact with the analog experts of the company. This has proven instrumental in merging the analog and the digital aspects of metastability enhancing the team's ability to approach metastability on digital level with a higher understanding.

The primary objective of this work was to force the generation of metastability on the cells of the specific technology employed for our test case. Our test case integrates TSMC library cells in 16nm FinFet. Normal flip-flops have to store data for multiple clock cycles achieving cycle-to-cycle deterministic circuit operation. To fully ensure this, the resulting circuit has relatively large metastability window (setup time plus hold time) where the data are forbidden to change to avoid the propagation of transient state. On the other hand, a synchronizer flip-flop is not the same as a normal data flip-flop. The latter is better to have a narrow metastability window and a short metastability resolution time to limit the probability of generating and propagating a metastability. Additionally, the role of a synchronizer is to accept data arriving at arbitrary timing, but on the side it

is no longer possible to maintain a cycle-to-cycle determinism. For this clear difference between both types of flip-flops, a key focus was placed on comparing the behaviour regrding metastability of the standard cells and the synchronization cells used to build multi-flop synchronizers [100] [101] [102].

Beyond standard operating conditions, we were also interested to have insights of their behaviour under extreme operating conditions, such as extreme temperatures. By undertaking this, our goal was discern the distinctive response of the cells types across a spectrum of operational scenarios.

4.2.2 Building test bench

The test bench was built using Virtuoso, a Cadence tool. We started with the development of a dedicated library to encapsulate the essential components. Within this library, we incorporated a DSPF (Design Specific Parameterized File) to capture the detailed specifications of the different cells (a standard and a sync TSMC in 16nm in our case). Furthermore, we had to create the schematic views for these cells. This was a notable challenge as the provider does not furnish schematic views for the cells. We undertook the task through the utilization of .spx files including the cells essential information and boolean equation. Figure 4.1 shows the created cell instantiated in a test bench with the necessary connections. A first test bench was created to simulate a real case of a CDC path. It had

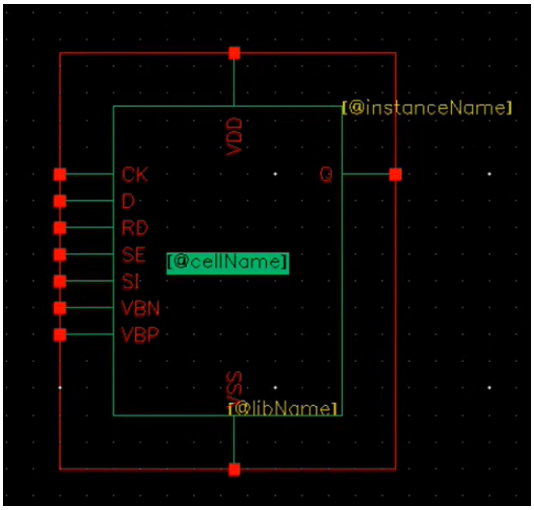


Figure 4.1: Created schematic for TSMC 16nm standard flip-flop

two versions. In the first version we connected two standard cells clocking them with two different clocks. In the second version, we replaced the standard cell in the destination by a sync cell as shown in Figure 4.2.

To comprehensively assesses the behavior of each cell, a second test bench with each cell in standalone was created as shown in Figure 4.3. This dedicated setup aimed at isolating individual cells and inducing metastability by initiating a violation of the setup/hold time by tuning the parameters of the pulse generator controlling the data pin. The parameters can be summarized in the rise and fall time of the data and the pulse width the period. We aimed to inspect how effectively they resolved metastbility and assessed their respective capabilities in preventing the propagation of metastable states. This approach provided insights into the difference between the standard and the sync cell regarding metastability resolution and propagation.

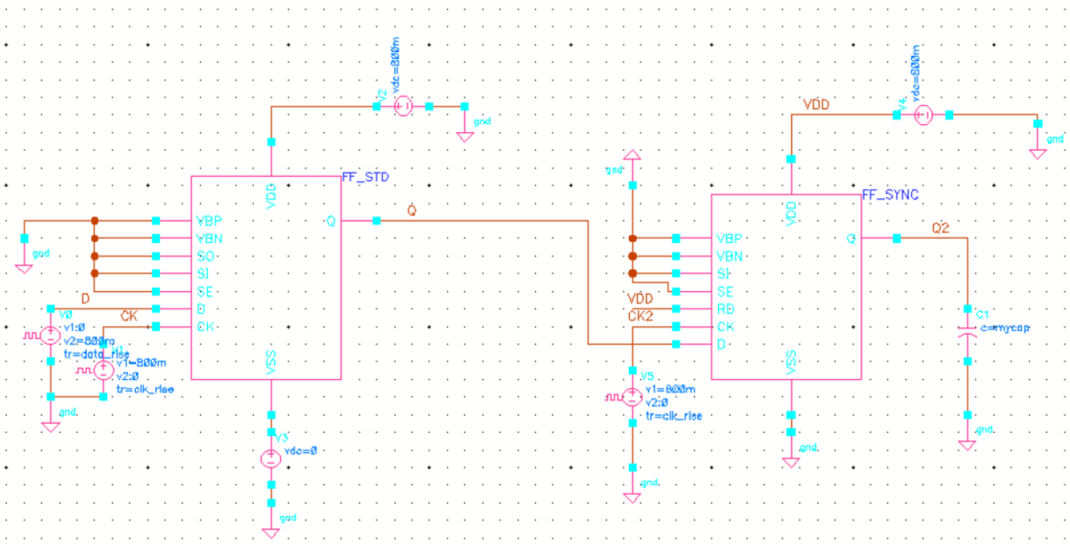


Figure 4.2: Created CDC test bench with a standard cell connected to a sync cell

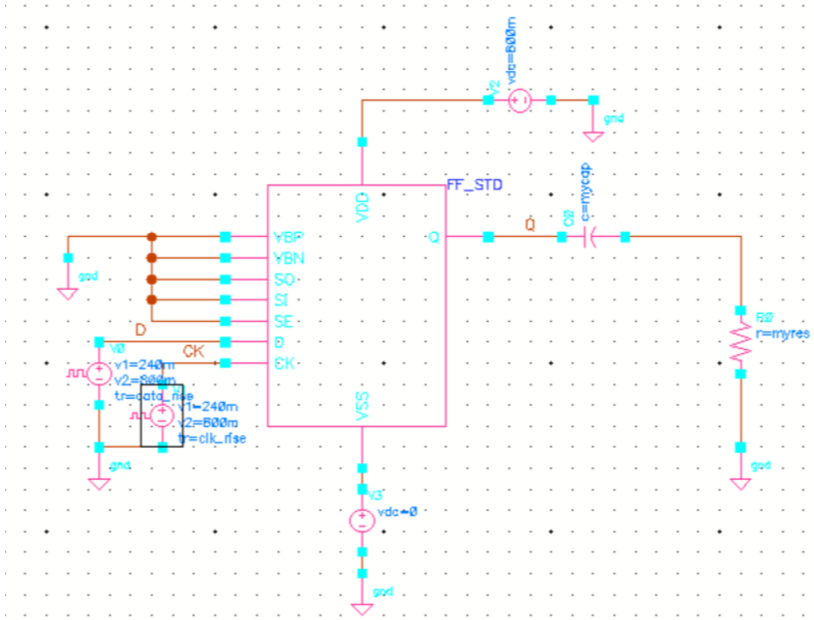


Figure 4.3: Created testbench for a sync cell in standalone

4.2.3 Simulation results

In the first test bench, we wanted to observe the behavior of CDC paths including standard cells or sync cells. The primary objective was to simulate a CDC path with both standard cells connected. As illustrated in the simulation results in Figure 4.4, we observe that when the rising edge of the destination clock $CK2$ is sufficiently distant from the falling edge of $CK1$, data D is safely propagated to Q in cycle 2 and to $Q2$ in cycle 3. This is especially relevant considering the activation timings of $CK1$ on falling edge and $CK2$ on rising edge. This simulation aimed to verify the secure transmission of data on a CDC path with standard cells were the metastability window is avoided.

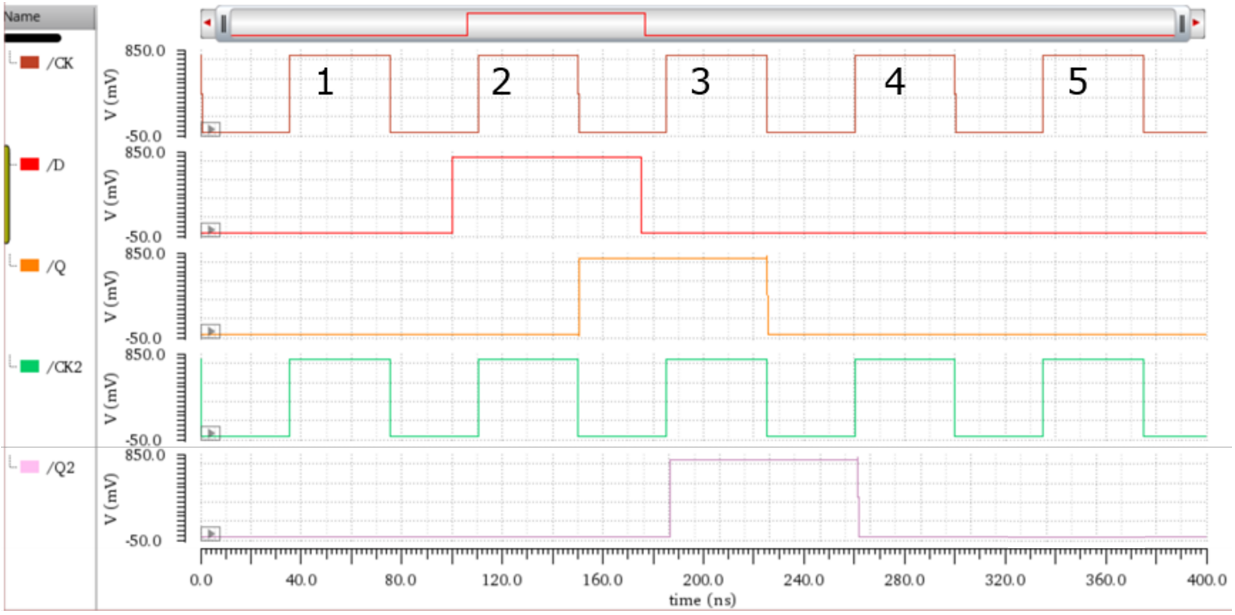


Figure 4.4: Simulation results of a CDC path between two standard TSMC cells

Subsequently, we conducted a follow-up simulation wherein we replaced the standard cells with synchronization cells to assess potential variation in behaviour. The results depicted in Figure 4.5 revealed a consistent conclusion with our previous one. When data changes occur sufficiently far from the metastability window, meaning the sampling edge of the source clock $Clk1$ is significantly distant from the mestactability window of $Clk2$, the data is reliably transmitted to $Q2$ with no observable delays.

In the second test bench, we conducted a thorough simulation of each cell type in standalone mode, systematically varying the delay on the rise time of the incoming data from zero to 500 microseconds with a pitch step of 20 microseconds. Figure 4.6 illustrates the behavior of a standard cell under these conditions. Notably, when observing the pin Q, there is a significant prolongation in the $tc2q$ (time from the sampling clock edge to the pin Q changing). Although this increase in $tc2q$ was calculated to be 130% of the original $tc2q$ —indicative of a metastability according to STMicroelectronics standards— no apparent metastability manifested on the pin Q itself. Subsequently, exploration of an internal node connecting the master and slave latches within the cell, denoted as $STD.I5P$ in Figure 4.6, revealed an unresolved metastability characterized by an intermediate voltage level. This was translated into a prolongation of $tc2q$ on the output Q, attributed to the additional time required for stabilization in the slave latch. Notice that this prolongation of $tc2q$ can conduct to violation of setup or hold timing in the next connected cell (if

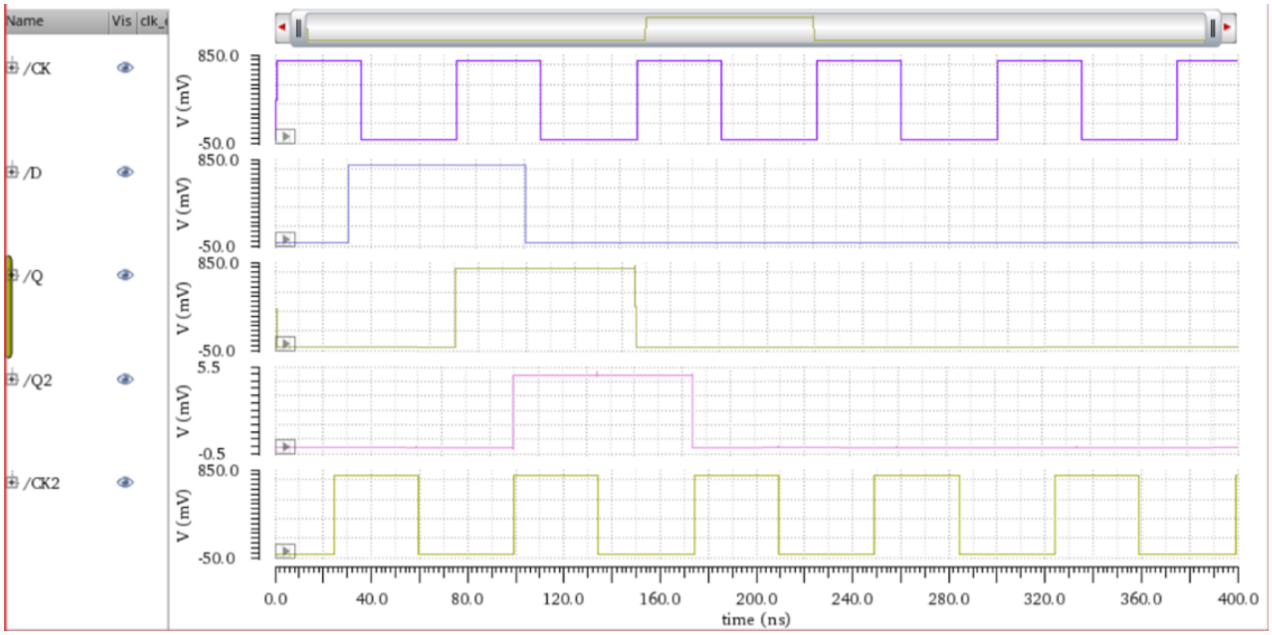


Figure 4.5: Simulation results of a CDC path between two sync TSMC cells

existing) which can generate and propagate a metastability. That explains why, according to STMicroelectronics, a prolongation of 130% of the original $tc2q$ is considered as metastable cell.

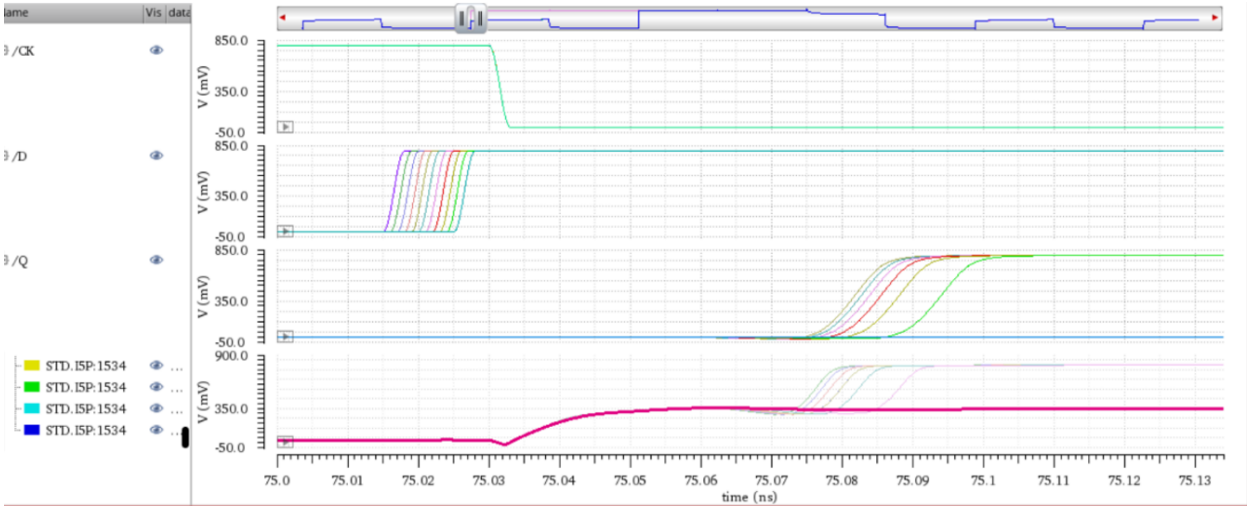


Figure 4.6: Simulation results of producing metastability on a TSMC standard cell

Following this, we applied the same data delay to a standalone synchronization cell to evaluate its response. As depicted in Figure 4.7, the output remained unchanged, with no observable prolongation in $tc2q$. Examining the internal node $sync.I5P$ of the synchronization cell revealed a distinct behavior compared to the standard cell. Here, a slow and gradual rise occurred, passing through an intermediate state. Unlike the standard cell, the internal node of the synchronization cell eventually resolved without stagnating at an intermediate voltage level. Due to the extended resolution time, the output Q did not capture the new data. Importantly, although data in this scenario may experience delays or be lost, the absence of $tc2q$ prolongation indicated the absence of inherent metastability

risk for Q and subsequent cells, providing insights into the robustness of synchronization cells in managing delayed data without introducing metastable states.

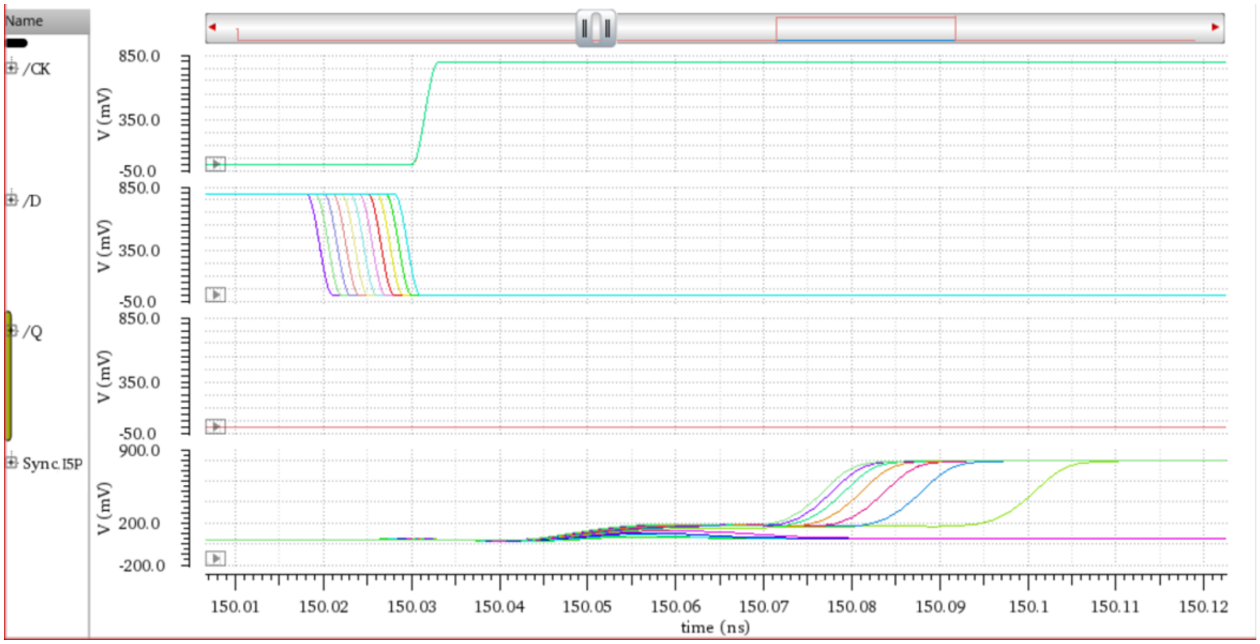


Figure 4.7: Simulation results of producing metastability on a TSMC sync cell

4.2.4 Conclusion and prespectives

In this section, we detailed our exploration of metastability simulation at the analog level, utilizing the same technology library employed in our CPU subsystem test case. The primary aim of these simulations was to gain firsthand experience and insights into the metastability phenomenon, along with understanding associated risks. A comparative analysis between a standard cell and a synchronization cell was conducted, introducing an initiated data delay to violate setup/hold timings of the sampling clock. Upon monitoring the internal node between the master and slave latch of the standard cell, we observed a stagnation at an intermediate voltage level, leading to a 130% prolongation of the original $tc2q$ on the output Q. This presented a potential risk of propagating metastability to subsequent cells. In contrast, the internal node of the synchronization cell demonstrated no stagnation at an intermediate voltage level. Despite taking a relatively longer time to reach a logical level, it eventually resolved, and no prolongation in $tc2q$ was observed on its output Q. However, a risk of data loss emerged in this scenario. The conclusion drawn was that, owing to their distinct physical characteristics, synchronization cells exhibited the ability to eventually resolve generated metastability compared to standard cells, which tended to propagate it.

Looking ahead, our future work involves expanding our comparative study by incorporating the ST FDSOI 28nm technology. This extension aims to provide a more comprehensive understanding of cell behavior in diverse technological. Additionally, we plan to delve into simulations under extreme conditions, specifically focusing on very high temperatures. This strategic approach will offer valuable insights into conducting CDC checks specifically for the specific operating conditions of each product.

4.3 Metastability modeling

4.3.1 Stability and metastability

The stability and the metastability are properties that can be associated to signals, or more precisely to the behaviour of signals. A signal $\mathscr{S}$ is a function in time t. On a digital level, signals are bi-stable: $s \in \mathbb{B}$, where $\mathbb{B} = \{0,1\}$.

$$\begin{aligned} \mathscr{S}: \quad & \mathbb{R} \to \mathbb{B} \\ & t \mapsto s \end{aligned}$$

On silicon, this perfect bi-stability state can be perturbed by many factors, producing volatile states that can involve an internal voltage between 0 and 1. This intermediate undefined state will be called Ω. In this case, $s \in \mathbb{V}$, where $\mathbb{V} = \{0,1\,\Omega\}$.

$$\begin{aligned} \mathscr{S}: \quad & \mathbb{R} \to \mathbb{V} \\ & t \mapsto s \end{aligned}$$

A signal is called stable in a time interval $[t_1,t_2]$ if it satisfies the following function [103]:

$$\begin{array}{rccl} \text{stab} : \mathbb{R}^2 \times \mathscr{S} & \to & \mathbb{B} & \\ t_1,t_2,s & \mapsto & \exists b \in \mathbb{B}, \forall t \in [t_1,t_2], s(t) = b & \end{array}$$

The stability of a storage element depends on its timing characteristics expressed in its own timing properties and the timing properties of its driving clock. Let C be the set of clocks clocking the storage elements of a given design. Each $clk_x \in C$ is associated to a number of attributes:

$$\text{period}: \quad \begin{array}{rccc} \Gamma(clk_x): & C & \to & \mathbb{R} \\ & clk_x & \mapsto & \Gamma_x \end{array}$$

$$\text{date}: \quad \begin{array}{rccc} dt(n,clk_x): & \mathbb{N} \times C & \to & \mathbb{R} \\ & n, clk_x & \mapsto & n.\Gamma(clk_x) \end{array}$$

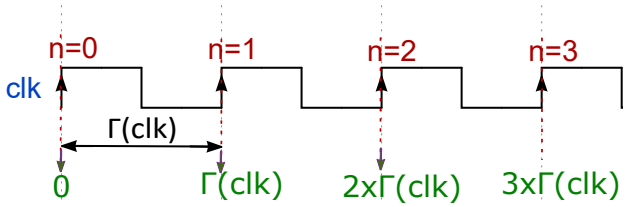


Figure 4.8: clock

Let $\mathscr{F}$ be the set of the registers existing in a design. The following attributes are

associated to each $x \in \mathscr{F}$ (see Figure 4.9)

input :
$$\begin{array}{rccc} D(t,x): & \mathbb{R}\times\mathscr{F} & \to & \mathscr{S} \\ & t,\mathrm{x} & \mapsto & D_x \end{array}$$

output :
$$\begin{array}{rccc} Q(t,x): & \mathbb{R}\times\mathscr{F} & \to & \mathscr{S} \\ & t,\mathrm{x} & \mapsto & Q_x \end{array}$$

control :
$$\begin{array}{rccc} Ce(t,x): & \mathbb{R}\times\mathscr{F} & \to & \mathscr{S} \\ & t,\mathrm{x} & \mapsto & Ce_x \end{array}$$

initial output :
$$\begin{array}{rccc} q^0(x): & \mathscr{F} & \to & \mathbb{V} \\ & \mathrm{x} & \mapsto & q^0_x \end{array}$$

setup time :
$$\begin{array}{rccc} t_{sp}(x): & \mathscr{F} & \to & \mathbb{R} \\ & \mathrm{x} & \mapsto & t_{sp_x} \end{array}$$

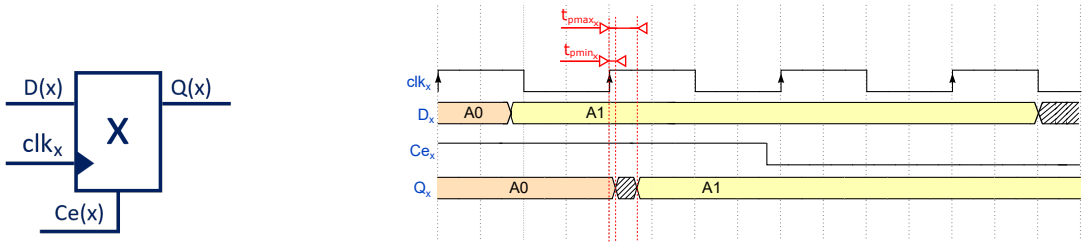
hold time :
$$\begin{array}{rccc} t_{hd}(x): & \mathscr{F} & \to & \mathbb{R} \\ & \mathrm{x} & \mapsto & t_{hd_x} \end{array}$$

time for the output to change :
$$\begin{array}{rccc} t_{pmin}(x): & \mathscr{F} & \to & \mathbb{R} \\ & \mathrm{x} & \mapsto & t_{pmin_x} \end{array}$$

time clock to output :
$$\begin{array}{rccc} t_{pmax}(x): & \mathscr{F} & \to & \mathbb{R} \\ & \mathrm{x} & \mapsto & t_{pmax_x} \end{array}$$

resolution time :
$$\begin{array}{rccc} \tau(f): & \mathscr{F} & \to & \mathbb{R} \\ & \mathrm{f} & \mapsto & \tau_f \end{array}$$

By abuse of notation, all the parameters of the previous functions will be denoted as subscripts in the following models. For example, the data $D(x)$ will be denoted D_x



(a) schematic *(b) Waveform*

Figure 4.9: Flip-flop "x"

Flip-flop stable behaviour

The behaviour of a flip-flop x is described "stable", if its output $Q(x)$ changes respecting a certain profile. Contrarily to the *stab* function, the output of a flip-flop x can hold a value Ω and yet be described as stable. This stays valid if and only if the value Ω enters, remains and exists following the known characteristic timing. The stable behaviour of a flop x is described as follows : if the data of the input D_x changes respecting the setup t_{su_x} and the hold t_{hd_x} timings of the flip-flop, it will be transmitted to its output Q_x at the next clock edge. "Q_x changes respecting three steps :

- It keeps its old value for a certain amount of time denoted t_{pmin_x}.
- It starts to change to a transient state denoted Ω.
- After a time denoted t_{pmax_x}, the new sampled data appears on the output Q_x.

This behaviour is described as follows :

$$R_x : \begin{array}{ll} \mathbb{N}, \mathscr{F}, C & \rightarrow \mathscr{S} \\ n, x, clk_x & \mapsto \end{array}$$

$$\begin{array}{ll} \text{when } n = 0, & \lambda t.q_x^0 \\ \text{when } Ce_x(dt(n)) = 1, & \lambda t. \begin{cases} R_x(n-1,\ldots)(t) & : t \in dt_x(n) + [0 : t_{pmin_x}[\\ \Omega & : t \in dt_x(n) +]t_{pmin_x} : t_{pmax_x}[\\ D_x(dt_x(n)) & : t \in dt_x(n) + [t_{pmax_x} : \Gamma_x] \\ \Omega & : t \notin]dt_x(n) : dt_x(n+1)[\end{cases} \end{array}$$

Flip-flop Metastable behaviour

The behaviour of a flip-flop x is described "Metastable" if an anomaly appears on its output $Q(x)$. An anomaly here is defined to be either the value Ω to remain on $Q(x)$ for an uncertain amount of time andor it gets resolved to an unknown value $x \in 0,1$. If D_x is changed too close to the sampling edge of clk_x and violates the setup or the hold timings, Q_x risks to stay longer in the transient phase Ω. Staying longer in the transient phase means taking longer to propagate the date between the sampling edge of clk_x and Q_x. The

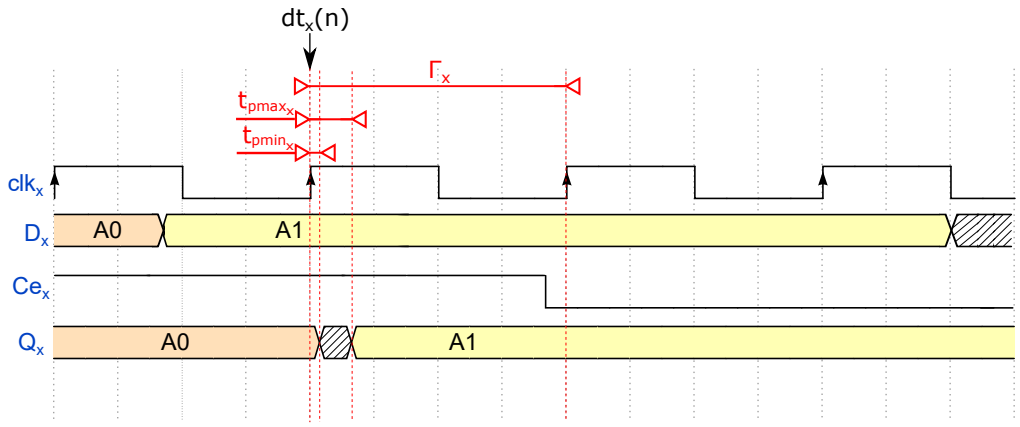


Figure 4.10: 1 flop stable

propagation time between the clock and the output is denoted t_{pcQ}. A longer t_{pcQ} is one of the most important clues that a metastability is generated and being transmitted between the master and the slave latches of the flop. A flip-flop x is described as metastable if Q_x changes as follows:

- It keeps its old value for a certain amount of time denoted "t_{pmin_x}".
- It starts to change to a transient state denoted Ω. This metastable unknown state will remain on Q_x for a time longer than the usual "t_{pmax_x}" until it gets resolved to a known value after a time τ_{res}.
- The metastability will be resolved to a certain logical value after a time "t_{pmax_x}" + τ_{res}". This logical value will remain unknown till a new value is sampled correctly.

This behaviour is described as follows :

$$\tilde{R}_x : \begin{array}{ll} \mathscr{F}, C & \to \mathscr{S} \\ x, clk_x & \mapsto \end{array}$$

$$\begin{array}{ll} \text{when } n = 0, & \lambda t.q_x^0 \\ \text{when } Ce_x(dt_x(n)) = 1, & \lambda t. \begin{cases} \tilde{R}_x(n_x - 1, \ldots)(t) & : t \in dt_x(n) + [0 : t_{pmin_x}[\\ \Omega & : t \in dt_x(n_x) +]t_{pmin_x} : t_{pmax_x} + \tau_{res}[\\ y \in \{0,1\} & : t \in dt_x(n_x) + [t_{pmax_x} + \tau_{res} : \Gamma_x] \\ \Omega & : t \notin]dt_x(n_x) : dt_x(n_x + 1)[\end{cases} \end{array}$$

4.3.2 CDC caused metastability

Synchronous communication

If two communicating flip-flops are clocked by the same clock or by two synchronous clocks, the setup and the hold timings are surely respected and the data is sampled safely.

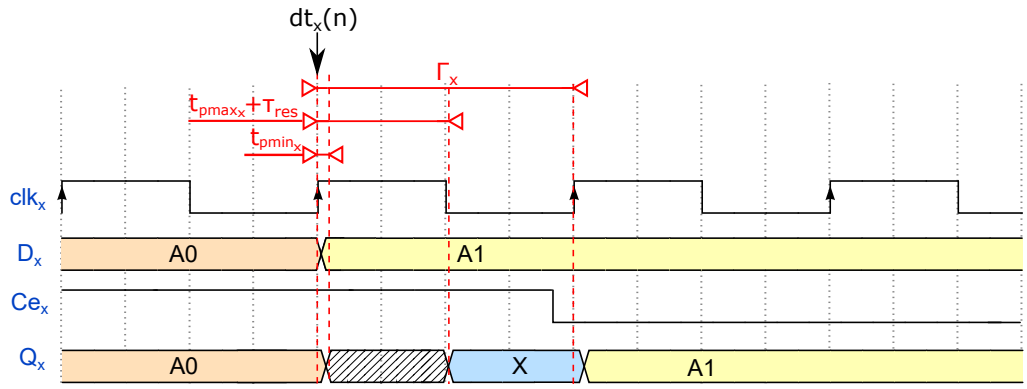


Figure 4.11: 1 flop unstable

The respect of the setup and hold timings ensures that the destination clock clk_d fires its edge far enough from t_{pmin_s} of the source avoiding sampling the old data, and far enough from t_{pmax_s} of the source avoiding sampling the transient value Ω. In this case, the destination flip-flop will have the stable behaviour R_d described previously. Assuming that the control of the source flip-flop s is enabled for one cycle, and the control of the destination flip-flop d is enabled during k cycles, the behaviour of these two communicating flip-flops is described as follows :

$$CDC_x : \begin{array}{ll} \mathscr{F}^2, C^2 & \to \mathscr{S}^2 \\ s, d, clk_s, clk_d & \mapsto \end{array}$$

$$\begin{array}{lll} \text{when} & n_s = 0, & \\ & \lambda t.(q_s^0, q_d^0) & \\ \text{when} & Ce_s(dt_s(n_s)) = 1 \wedge \forall n_s \in]1:k]\ Ce_s(dt_s(n_s)) = 0 & \\ \wedge & \forall n_d \in [1:k]\ Ce_d(dt_d(n_d)) = 1 & \\ & \lambda t. \begin{cases} (R_s(n_s,\ldots)(t), R_d(n_d,\ldots)(t)) & : t \in [dt_s(n_s) : dt_s(n_s+k)[\\ (\Omega, \Omega) & : t \notin [dt_s(n_s) : dt_s(n_s+k)[\end{cases} & \end{array}$$

Asynchronous communication

In case of clock domain crossing, where the two communicating flip-flops are clocked by different asynchronous clocks, the destination clock clk_d can fire its edge very close to t_{pmin_s} of the source sampling the old data or within "t_{pmax_s}" of the source sampling the undefined value Ω. For that, the destination flop is in risk to sample a metastability. That is why, for the clock first cycle, the destination flip-flop d will have the metastable behaviour $\tilde{R}$ described before. Then, if the data is kept stable for k cycles, the destination flip-flop will restore the stable behaviour R. The behaviour of these two communicating flip-flops is described as follows :

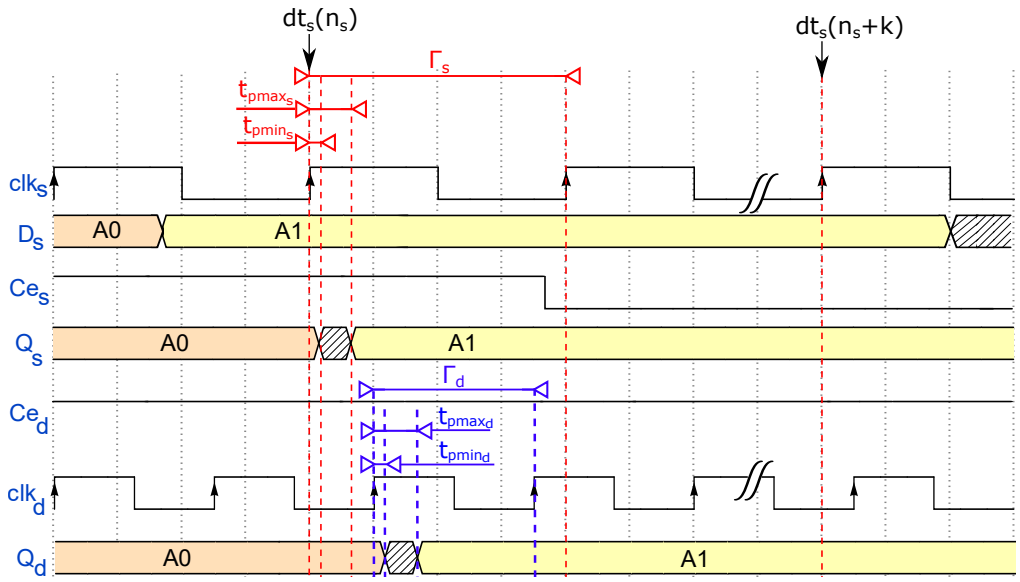


Figure 4.12: CDC 1FF stable

$$\tilde{CDC}_x : \begin{array}{ll} \mathscr{F}^2, C^2 & \to \mathscr{S}^2 \\ s, d, clk_s, clk_d & \mapsto \end{array}$$

$$\begin{array}{lll} \text{when} & n_x = 0, & \\ & \lambda t.(q_s^0, q_d^0) & \\ \text{when} & Ce_s(dt_s(n_s)) = 1 \wedge \forall n_s \in]1:k]\ Ce_s(dt_s(n_s)) = 0 & \\ \wedge & \forall n_d \in [1:k]\ Ce_d(L_d(n_d)) = 1 & \\ & \lambda t. \begin{cases} (R_s(n_s,\ldots)(t), \tilde{R}_d(n_d,\ldots)(t)) & : t \in [dt_s(n_s) : dt_s(n_s+1)[\\ (R_s(n_s,\ldots)(t), R_d(n_d,\ldots)(t)) & : t \in [dt_s(n_s+1) : dt_s(n_s+k)[\\ (\Omega, \Omega) & : t \notin [dt_s(n_s) : dt_s(n_s+k)[\end{cases} & \end{array}$$

General CDC model

We can conclude that $dt_d(n_d) - t_{su_d}$ and $dt_s(n_s) + t_{pmax_s}$ are the important dates on which depends whether the asynchronous communication is safe or not. The communication is described safe or stable if $dt_d(n_d) - t_{su_d}$ is larger than $dt_s(n_s) + t_{pmax_s}$ and the data is kept stable during the study period (described in this model as k number of cycles). This guarantees that the time of the sampling edge minus the setup time never happens before the source clock edge plus t_{pmax_s} avoiding sampling the source data while it is in the transient state Ω. This can be expressed in 4.14 by the purple line. If the sampling edge of clk_d happens after this purple line, no metastability will occur on the destination flip-flop. This general model is described as follows :

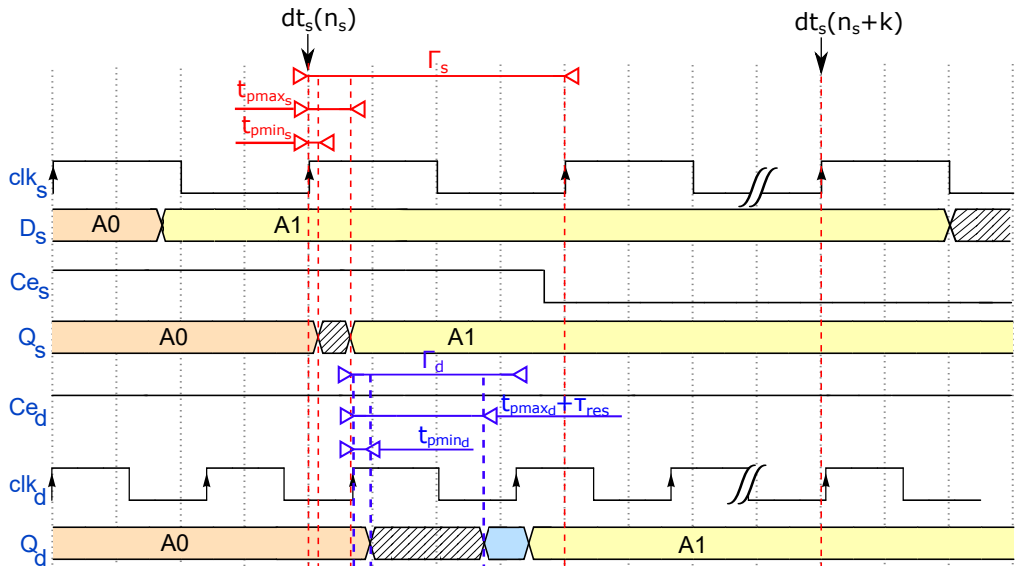


Figure 4.13: CDC 1FF unstable

$$CDC_{1FF}: \begin{array}{ll} \mathscr{F}^2, C^2 & \to \mathscr{S}^2 \\ s, d, clk_s, clk_d & \mapsto \end{array}$$

$$\begin{array}{ll} \text{when } stab(dt_s(n_s) - t_{su_s}), dt_s(n_s + k) + t_{hd_s}, D_s) & \\ \wedge\, dt_d(n_d) - t_{su_d} > dt_s(n_s) + t_{pmax_s} & CDC_{1FF} \\ \text{when } stab(dt_s(n_s) - t_{su_s}), dt_s(n_s + k) + t_{hd_s}, D_s) & \\ \wedge\, dt_d(n_d) - t_{su_d} < dt_s(n_s) + t_{pmax_s} & C\tilde{D}C_{1FF} \end{array}$$

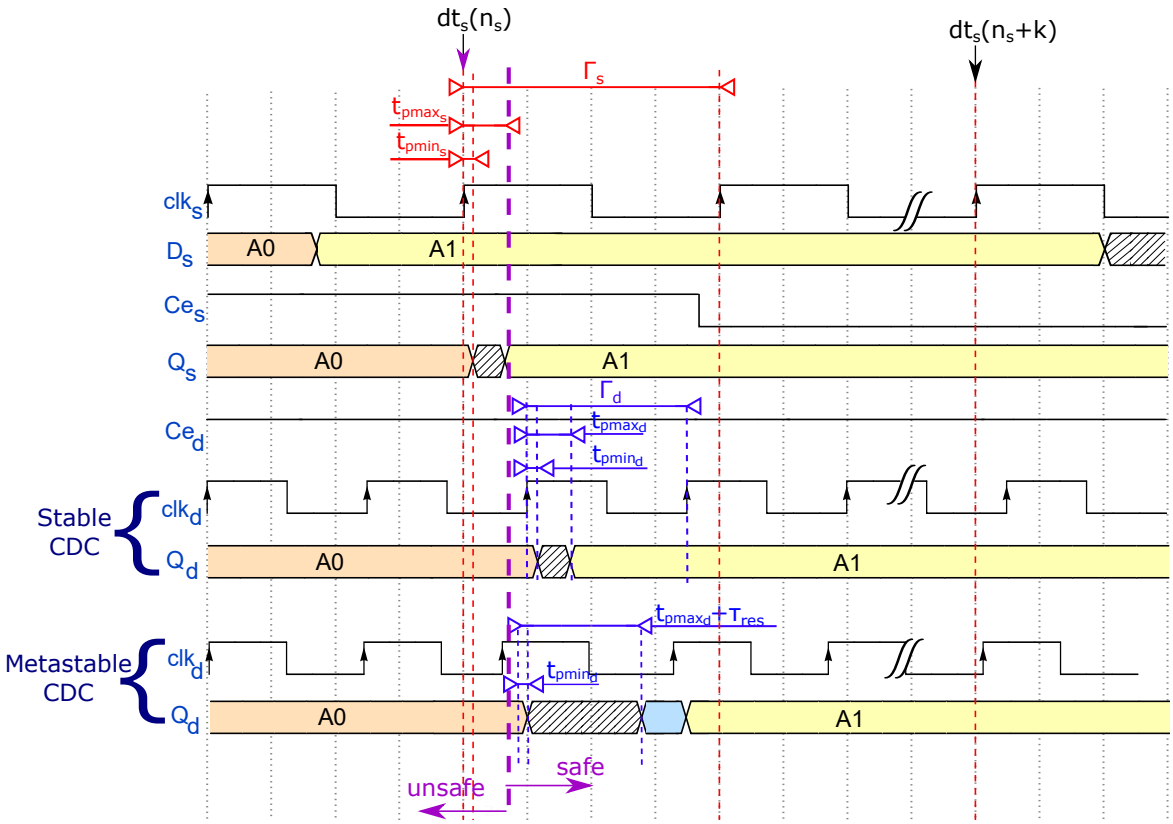


Figure 4.14: CDC 1FF

4.4 Metastability injection

Metastability injection in RTL simulations is an important aspect of digital design verification. Although metastability itself cannot be injected in RTL simulation, it is crucial to discover any potential issues related to metastability at early stages. Metastability injection refers to mimicking the effect of metastability on digital level, which can lead to data loss due to the prolongation of tc2q as seen in the analog simulation results os section 4.2. To understand how this analog phenomenon can be reproduced on digital level, it is necessary to explain metastability from a pure digital point of view. EDA vendors have developed solutions for metastability injection, and it is important to study the effect of metastability on synchronization protocols and the quality of results of the CDC verification. A first trial of metastability injection on a simple asynchronous FIFO provided us valuable insights into this issue. Overall, metastability injection is an important aspect of digital design verification that should not be overlooked.

4.4.1 Metastability effect on digital level

The effect of metastability is dependent on the type of timing violation that led to its occurrence and how it was resolved. Typically, metastability is caused by violations of setup and hold timings. If metastability occurs, it can either be resolved quickly to the expected value of the input data, known as a "fast resolution", or it can resolve to the opposite value of the input data, known as a "slow resolution". The terms "fast" and "slow" refer to the speed at which the metastability is resolved within the same clock cycle. In the case of a slow resolution, it is important for the data to remain stable in

order to be re-sampled in the next clock cycle. Otherwise, it may be completely lost. The effect of metastability can vary greatly depending on the specific circumstances of the timing violation and resolution, making it an important consideration in digital design verification.

Setup violation

Consider the example of a CDC synchronized by a 2-stage MFS, as illustrated in Figure 4.15. If the data on the destination side toggles far enough from the metastability window of the destination clock, then metastability will not occur. This can be observed by following the green arrows sequence in Figure 4.16. When *D_correct* toggles early enough at node *x*, the new data is transmitted to *q*1*_correct* in cycle 2 at node *y* and to *q*2*_correct* in cycle 3 at node z without any metastability.

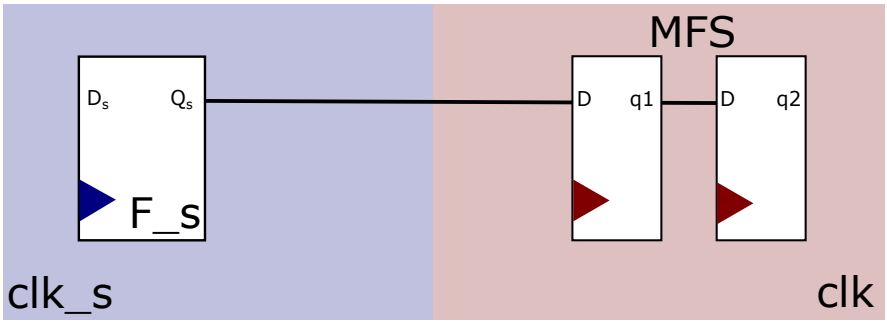


Figure 4.15: CDC path with two stage MFS

In contrast, if the data at *D* is delayed and violates the setup time of the destination clock, a metastability can occur on the first destination flip-flop. Following the blue arrow sequence in Figure 4.16, when the delayed data *D* toggles at node *a*, a metastability appears on *q*1 at node *c* due to the violation of the setup time. The metastability persists in the flip-flop for a certain duration and is eventually resolved at node *d*. If the metastability resolves to the correct data value, as depicted in Figure 4.16, then the correct data is transmitted to *q*2 in cycle 3, meeting the normal operating timing expectations. In such a scenario, the metastability has no impact on the digital level. In this case, the metastability resolution is described as fast resolution. On the other hand, if the metastability

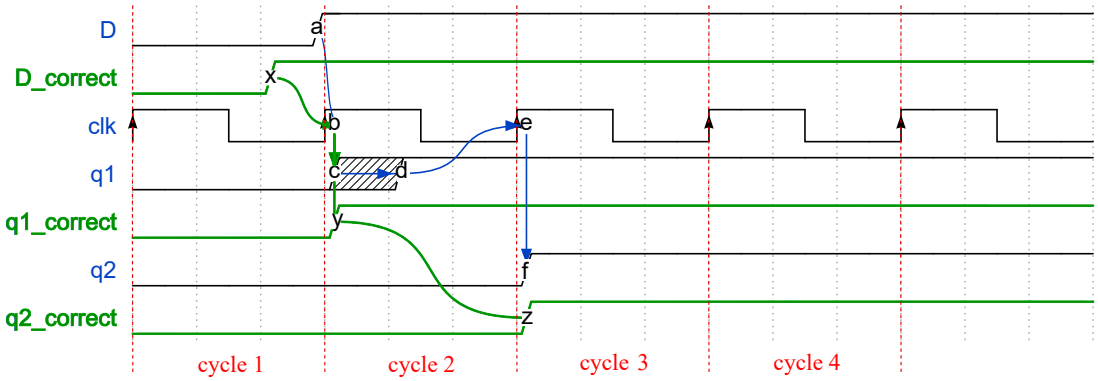


Figure 4.16: Setup violation: Metastability fast resolution

resolves to the incorrect data value at node *d*, as shown in Figure 4.17, the new data must

be re-sampled at the next clock cycle. As a result, it is transmitted to $q2$ in cycle 4 instead of cycle 3. In this scenario, the effect of the metastability manifests as a one-cycle delay on the digital level. The metastability resolution in this case is described as slow due to the additional time it took for the new data to reach the destination.

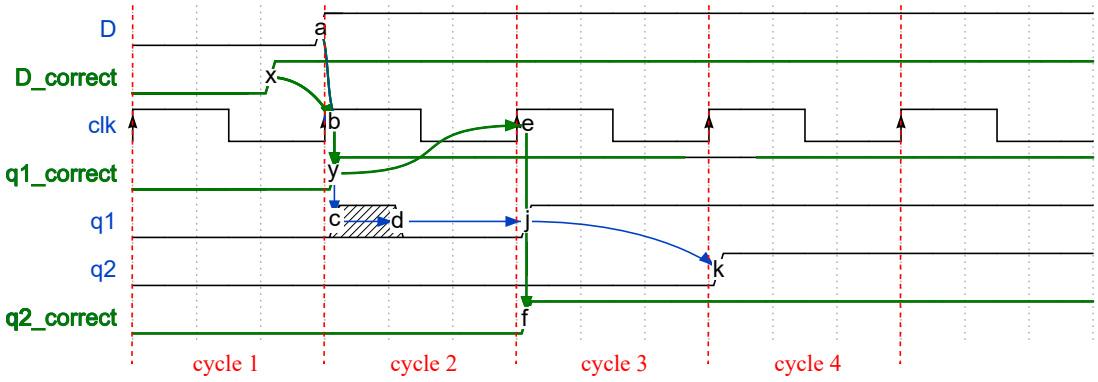


Figure 4.17: Setup violation: Metastability slow resolution

Hold violation

Consider another scenario where the data at D must change after the clock edge to sample the old data. Following the green arrow sequence in Figure 4.18, if *D_correct* changes at node x far from the hold timing of the clock edge b, the new data will be transferred to $q1$*_correct* at cycle 3 and to $q2$*_correct* at cycle 4. However, if the data changes slightly earlier, violating the hold timing as shown at node a, a metastability can occur on $q1$ at node c. In this case, following the blue arrow sequence, if the metastability resolves quickly to the new data at node d, the new data will be transferred to $q2$ in cycle 3 instead of cycle 4. In this scenario, the resolution is described as fast and the effect of the metastability that can be observed on the digital level is the advancement of the data by one cycle.

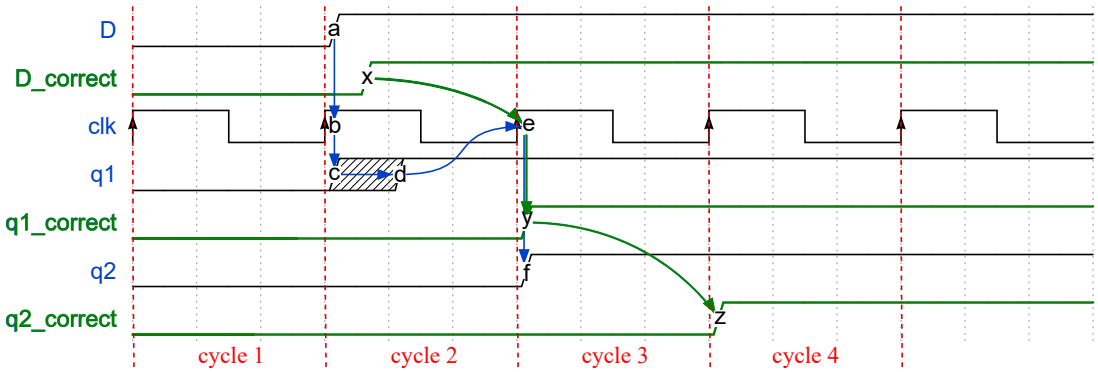


Figure 4.18: Hold violation: Metastability fast resolution

On the other hand, if the metastability resolves to the old data value, as shown at node d in Figure 4.19, the new data is transmitted to $q2$ in cycle 4, meeting the timing

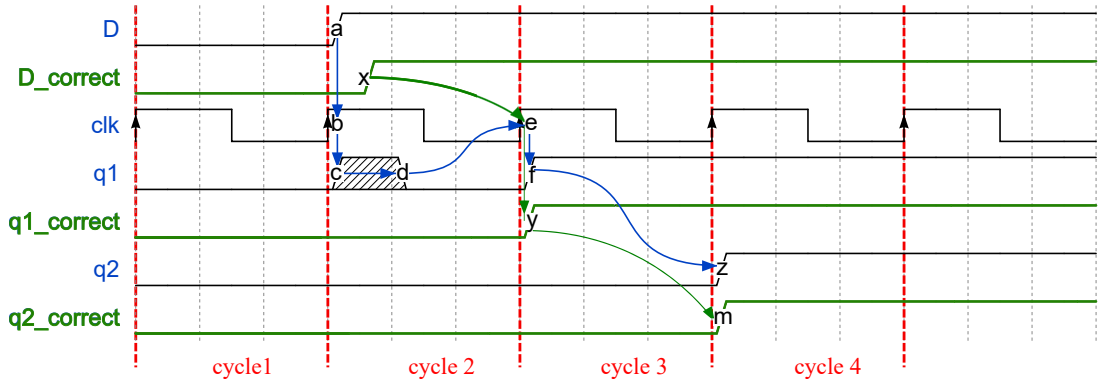


Figure 4.19: Hold violation: Metastability slow resolution

expectations of the original normal behavior. In this scenario, the metastability resolution is described as slow, and no effect appears on the digital level.

Table 4.1 concludes the effect of the metastability on digital level. We can observe 4 corner cases. The first is where the setup timing is violated and the metastability has a fast resolution. In this case the metastability has no effect on digital level. The second is where the setup timing is violated and the metastability has a slow resolution. In this case the data is delayed by one clock cycle. The third case concerns the hold timing violation with fast resolution. The data in this case reaches the destination one clock cycle earlier that expected. And the fourth and last corner case is when the hold timing is violated and the metastability has a slow resolution. No effect is seen on digital level in this case.

Violation	Resolution	q2 changes in cycle #	Effect
None	None	3	None
Setup	Fast	3	None
Setup	Slow	2	One cycle delay
None	None	4	None
Hold	Fast	3	One cycle earlier
Hold	Slow	4	None

Tab. 4.1: Metastability effect on digital level according to the violation and the resolution types

4.4.2 Metastability injection technologies

The metastability effect can be injected as an error in simulation by mimicking the four corner cases concluded in Table 4.1. There exist several techniques to inject this type of errors. After a study we conducted to gather information about what the EDA tools propose in this field, we concluded that there exist two main approaches categories for metastability injection:

- **Intrusive approaches:** An intrusive approach is an approach that changes the verified RTL in order to introduce one or more corner cases out of the metastability corner cases presented in 4.1. The main idea is that the RTL is scanned to detect all the MFS on the different CDC paths and to substitute them with modified MFS code able to introduce a delay or an advancement to the crossing signal. These latter are called "RTL substitution models". An example is shown in Figure 4.20 where the substitution model in illustrated in the dotted frame. This model is inserted between the two stages of the MFS $FF1$ and $FF2$. $FF1'$ delays the crossing signal one clock cycle. *random* is generated everytime a change occurs on $FF1$. If *random* is 1, the data will be delayed for one cycle. *random* is generated by dedicated behavioural RTL using a random seed.

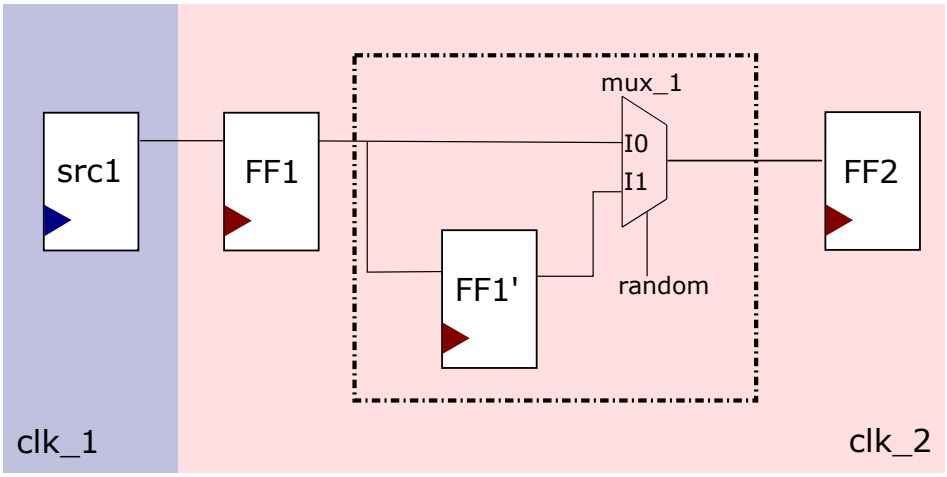


Figure 4.20: Metastability substitution model

- **Non-intrusive approaches:** The non-intrusive methodologies depend on signal forcing during simulation. Dedicated modules, *metastabilitychecker* in Figure 4.21, calculate the possibility of a setup or hold timing violations. The calculation is done based on the source and the destination clocks and a metastability window defined prior to the simulation. The metastability window is usually a percentage of the destination clock period. If the data is changed in this time interval, a timing violation is flagged and the signal at the output if the second MFS stage can be forced to its opposite value. This can delay or advance the crossing signal of one clock cycle mimicking the effect of the metastability. The non-intrusive approaches do not change anything in the native design RTL. Instead they calculate the potential timing violations and force the signals on the spot in simulation.

4.4.3 Metastability injection on asynchronous FIFO

In order to assess the viability of applying a metastability injection flow, previously untested in any project, a strategic decision was made to conduct initial testing on a smaller-scale project. The primary objectives were to understand the flow application, identify potential bugs related to design, and evaluate the overall feasibility and interest in implementing such a methodology. The testing aimed to uncover insights that would guide the application of the flow on a larger design. Despite initial doubts about the

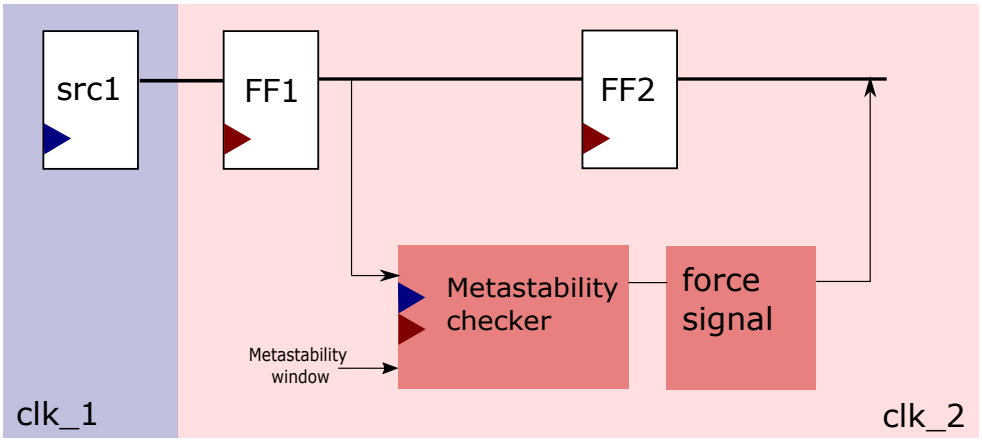


Figure 4.21: Metastability injection logic

feasibility, the preliminary results have proven the importance of this approach providing valuable insights into potential challenges and benefits. This testing phase serves as a foundational step for a more comprehensive application of the metastability injection flow on larger-scale projects.

Test case

We selected an asynchronous FIFO as our test case due to its comprehensive coverage of aspects related to Clock Domain Crossing (CDC). This particular FIFO encapsulates CDC data paths connecting memory and the destination flop. Additionally, it features CDC control paths, where read and write pointers are synchronized bit by bit using Multi-Flop Synchronizers (MFS). The test case also addresses reconvergences observed among various bits of the resynchronized pointers, crucial for calculating empty and full flags.

In light of this, our test comprises five key modules,as shown in Figure 4.22 detailed as follows:

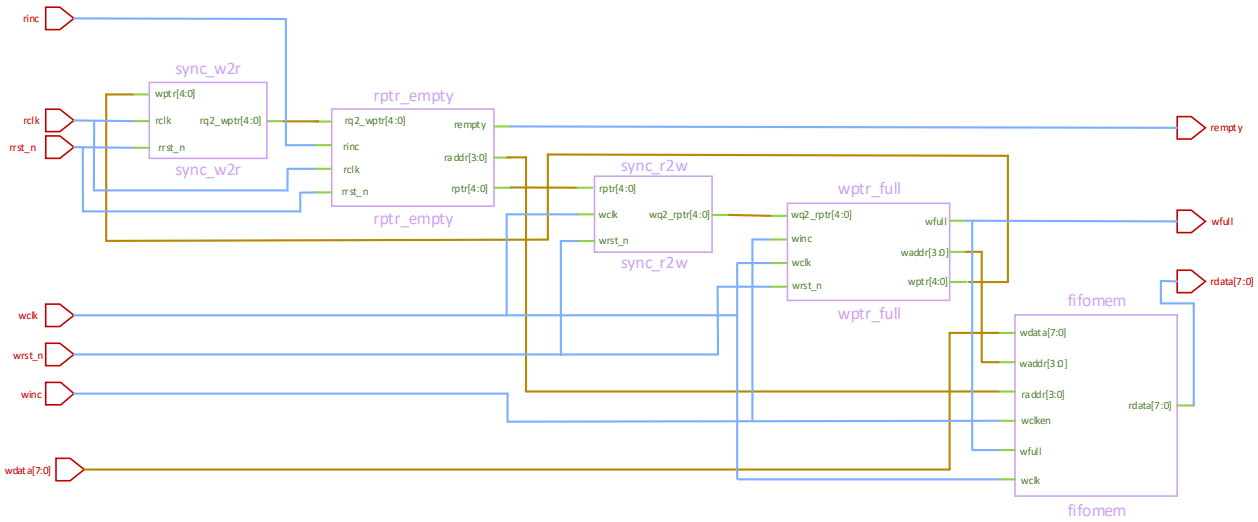


Figure 4.22: Asynchronous FIFO block diagram

- **The FIFO memory** (*fifomem.v*): The memory of the FIFO paramerizd by the data size *DATASIZE* and the addresses size *ADDRSIZE*.

```
module fifomem #(parameter DATASIZE = 8,
// Memory data word width
 parameter ADDRSIZE = 4) // Number of mem address bits
 (output [DATASIZE-1:0] rdata,
 input [DATASIZE-1:0] wdata,
 input [ADDRSIZE-1:0] waddr, raddr,
 input wclken, wfull, wclk);
 `ifdef VENDORRAM
 // instantiation of a vendor's dual-port RAM
 vendor_ram mem (.dout(rdata), .din(wdata),
 .waddr(waddr), .raddr(raddr),
 .wclken(wclken),
 .wclken_n(wfull), .clk(wclk));
 `else
 // RTL Verilog memory model
 localparam DEPTH = 1<<ADDRSIZE;
 reg [DATASIZE-1:0] mem [0:DEPTH-1];
 assign rdata = mem[raddr];
 always @(posedge wclk)
 if (wclken && !wfull) mem[waddr] <= wdata;
 `endif
endmodule
```

- **Write address synchronization** (*sync_w2r.v*): This module is dedicated to synchronize the write pointer with a three stages MFS.

```
module sync_w2r #(parameter ADDRSIZE = 4)
 (output reg [ADDRSIZE:0] rq2_wptr,
 input [ADDRSIZE:0] wptr,
 input rclk, rrst_n);
 reg [ADDRSIZE:0] rq1_wptr;
 reg [ADDRSIZE:0] rq1x_wptr;
 always @(posedge rclk or negedge rrst_n)
 if (!rrst_n) {rq2_wptr,rq1x_wptr,rq1_wptr} <= 0;
 else
 {rq2_wptr,rq1x_wptr,rq1_wptr} <= {rq1x_wptr,rq1_wptr,wptr};
endmodule
```

- **Read address synchronization** (*sync_r2w.v*): This module is dedicated to synchronize the read pointer with a three stages MFS.

```
module sync_r2w #(parameter ADDRSIZE = 4)
 (output reg [ADDRSIZE:0] wq2_rptr,
 input [ADDRSIZE:0] rptr,
 input wclk, wrst_n);
 reg [ADDRSIZE:0] wq1_rptr;
 reg [ADDRSIZE:0] wq1x_rptr;
 always @(posedge wclk or negedge wrst_n)
 if (!wrst_n) {wq2_rptr,wq1x_rptr,wq1_rptr} <= 0;
 else
 {wq2_rptr,wq1x_rptr,wq1_rptr} <= {wq1x_rptr,wq1_rptr,rptr};
endmodule
```

- **Full Flag calculation** (*wptr_full.v*): This module is dedicated to calculate the full flag.

```
module wptr_full #(parameter ADDRSIZE = 4)
 (output reg wfull,
 output [ADDRSIZE-1:0] waddr,
 output reg [ADDRSIZE :0] wptr,
 input [ADDRSIZE :0] wq2_rptr,
 input winc, wclk, wrst_n);

 reg [ADDRSIZE:0] wbin;
 wire [ADDRSIZE:0] wgraynext, wbinnext;

 // GRAYSTYLE2 pointer
 always @(posedge wclk or negedge wrst_n)
 if (!wrst_n) {wbin, wptr} <= 0;
 else {wbin, wptr} <= {wbinnext, wgraynext};
 // Memory write-address pointer
 //(okay to use binary to address memory)
 assign waddr = wbin[ADDRSIZE-1:0];
 assign wbinnext = wbin + (winc & ~wfull);
 `ifdef INJECT_GRAY_ERROR
   assign wgraynext =  wbinnext;
 `else
   assign wgraynext = (wbinnext>>1) ^ wbinnext;
 `endif

 //three necessary full-tests
 assign wfull_val=
 ((wgraynext[ADDRSIZE] !=wq2_rptr[ADDRSIZE])&&
 (wgraynext[ADDRSIZE-1] !=wq2_rptr[ADDRSIZE-1]) &&
 (wgraynext[ADDRSIZE-2:0]==wq2_rptr[ADDRSIZE-2:0]));

 always @(posedge wclk or negedge wrst_n)
 if (!wrst_n) wfull <= 1'b0;
 else wfull <= wfull_val;

endmodule
```

- **Empty Flag calculation** (*rptr_empty.v*): This module is dedicated to calculate the empty flag.

```
module rptr_empty #(parameter ADDRSIZE = 4)

 (output reg rempty,
 output [ADDRSIZE-1:0] raddr,
 output reg [ADDRSIZE :0] rptr,
 input [ADDRSIZE :0] rq2_wptr,
 input rinc, rclk, rrst_n);
 reg [ADDRSIZE:0] rbin;
 wire [ADDRSIZE:0] rgraynext, rbinnext;
 //-------------------
 // GRAYSTYLE2 pointer
 //-------------------
 always @(posedge rclk or negedge rrst_n)
 if (!rrst_n) {rbin, rptr} <= 0;
 else {rbin, rptr} <= {rbinnext, rgraynext};
 // Memory read-address pointer
 //(okay to use binary to address memory)
 assign raddr = rbin[ADDRSIZE-1:0];
 assign rbinnext = rbin + (rinc & ~rempty);
`ifdef INJECT_GRAY_ERROR
   assign rgraynext = rbinnext;
`else
   assign rgraynext = (rbinnext>>1) ^ rbinnext;
`endif

 assign rempty_val = (rgraynext == rq2_wptr);

 always @(posedge rclk or negedge rrst_n)
 if (!rrst_n) rempty <= 1'b1;
 else rempty <= rempty_val;
endmodule
```

A simple test bench, as shown in Figure 4.23 was created to stimulate the asynchronous interfaces. The top *top.v* instantiates a standard counter that generates data on a source clock *clks* to be passed to an asynchronous destination *clkd* through the FIFO. The counter is intended to stop counting when the FIFO is full. The *rempty* flag qualifies the output of the FIFO enabling a register bank, where data are recirculated by a re-circulation mux if the *rempty* is flagged.

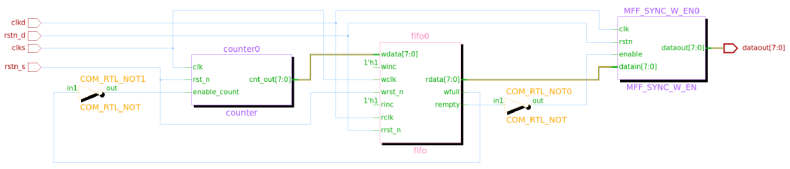


Figure 4.23: Asynchronous FIFO test bench block diagram

Results

Our analysis commenced with a standard structural check using a CDC static tool to guarantee the design's integrity from a CDC perspective. The structural verification successfully passed, revealing no violations. Subsequently, we initiated our test bench simulation to establish a reference simulation database, against which we could compare our results following the injection of metastability.

Following that, we moved on to injecting metastability, opting for a non-intrusive approach. Dedicated EDA tools generated metastability checkers for each signal synchronized by MFS, focusing on both pointers in our test case. We established a predefined metastability window equivalent to 50% of the clock period of the destination clock. This setup meant that if the source-side data coincided too closely with the destination clock edge within this 50% period, based on a random seed, the signal could be manipulated to induce either a delay or an advancement.

The coverage report, illustrated in Figure 4.24, reveals that all relevant signals experienced metastability injection at least once. Signal number 4, for example, exhibited the fewest error injections, with the signal toggling being affected by an error injection in 50% of instances.

```
##############################################################################
#             Dynamic CDC Verification Detailed Coverage
##############################################################################

-------------------------------------------------------------------------------------------------------------
CDC Id  Signal Name                Toggles   Async D                Errors Injected
                                             Toggles   No Jitter        Delayed           Total
1       tb_top.top0.fifo0.sync_r2w.wq1_rptr[0]
                                   131       131       39(29.77%)       92(70.23%)        92(70.23%)
2       tb_top.top0.fifo0.sync_r2w.wq1_rptr[1]
                                   66        66        25(37.88%)       41(62.12%)        41(62.12%)
3       tb_top.top0.fifo0.sync_r2w.wq1_rptr[2]
                                   33        33        14(42.42%)       19(57.58%)        19(57.58%)
4       tb_top.top0.fifo0.sync_r2w.wq1_rptr[3]
                                   16        16        8(50.00%)        8(50.00%)         8(50.00%)
5       tb_top.top0.fifo0.sync_r2w.wq1_rptr[4]
                                   16        16        8(50.00%)        8(50.00%)         8(50.00%)
6       tb_top.top0.fifo0.sync_w2r.rq1_wptr[0]
                                   138       138       55(39.86%)       83(60.14%)        83(60.14%)
7       tb_top.top0.fifo0.sync_w2r.rq1_wptr[1]
                                   69        69        23(33.33%)       46(66.67%)        46(66.67%)
8       tb_top.top0.fifo0.sync_w2r.rq1_wptr[2]
                                   35        35        9(25.71%)        26(74.29%)        26(74.29%)
9       tb_top.top0.fifo0.sync_w2r.rq1_wptr[3]
                                   17        17        6(35.29%)        11(64.71%)        11(64.71%)
10      tb_top.top0.fifo0.sync_w2r.rq1_wptr[4]
                                   17        17        7(41.18%)        10(58.82%)        10(58.82%)
-------------------------------------------------------------------------------------------------------------
```

Figure 4.24: Metastability injection coverage report

During the results inspection, we compared the waveforms between standard simulation and metastability injection. The metastability injection points are marked with orange dots in the chronogram in Figure 4.25 on different bits of the write pointer. As we examined other key control signals, an initial observation was a false pulse on the empty flag. This implies that the empty flag was erroneously flagged between $t = 1000$ and $t = 1500$, violating the condition $(rq2\_wptr == rgraynext)$. While an incorrectly flagged flag may be inconvenient, the converse situation – a flag that should be flagged but isn't – can be potentially critical.

Upon examining the impact of metastability injection on another control signal, specifically the full flag post-error injection on the read pointer, critical instances were identi-

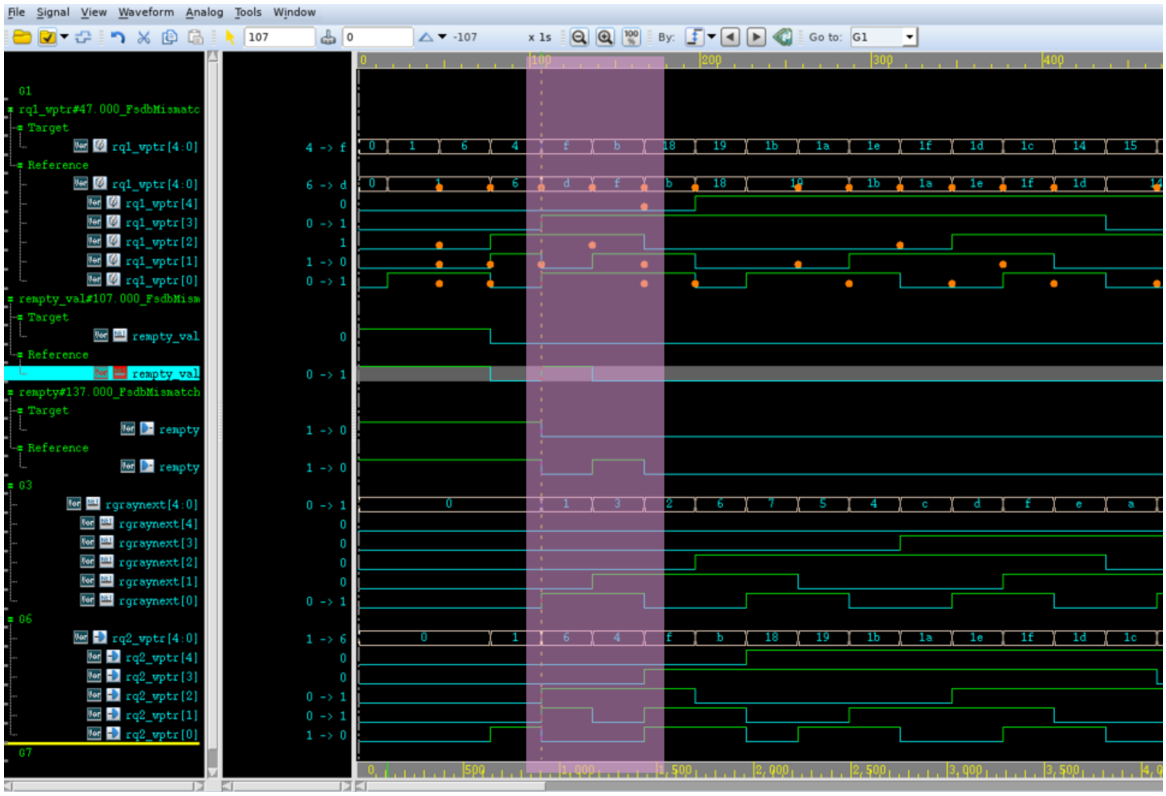


Figure 4.25: Metastability injection effect on Empty flag

fied. The chronogram in Figure 4.26 illustrates situations where the full flag should have been flagged, yet it wasn't (*e.g.*, between t=100 and t=300). This resulted in an unintended data write, where *wdata* changed when it should have remained stable, contradicting the intended prohibition.

Tracing back the root cause of the unexpected value propagated, as shown in Figure 4.27, on the full flag, we identified it as a consequence of reconvergence of different bits of the read pointer where metastability was injected. This raised the question: why were not these reconvergences detected during structural checks? It turns out that the tool typically halts reconvergence analysis when the signals pass through a defined number of sequential elements before the reconvergence. In our case, with three stages of MFS synchronizing the pointers, and the tool's minimum depth being 2, the third flop was considered an additional sequential depth, leading to the skipping of reconvergence analysis. The metastability injection proved importance in uncovering these skipped reconvergences, highlighting its effectiveness in detecting critical reconvergences that might lead to functionality failure, often masked by the sequential depth parameter in structural analysis.

4.5 Perspectives and future work

Metastability injection serves as a vital complement to CDC verification by addressing potential design vulnerabilities. While initial test cases may not have yielded significant

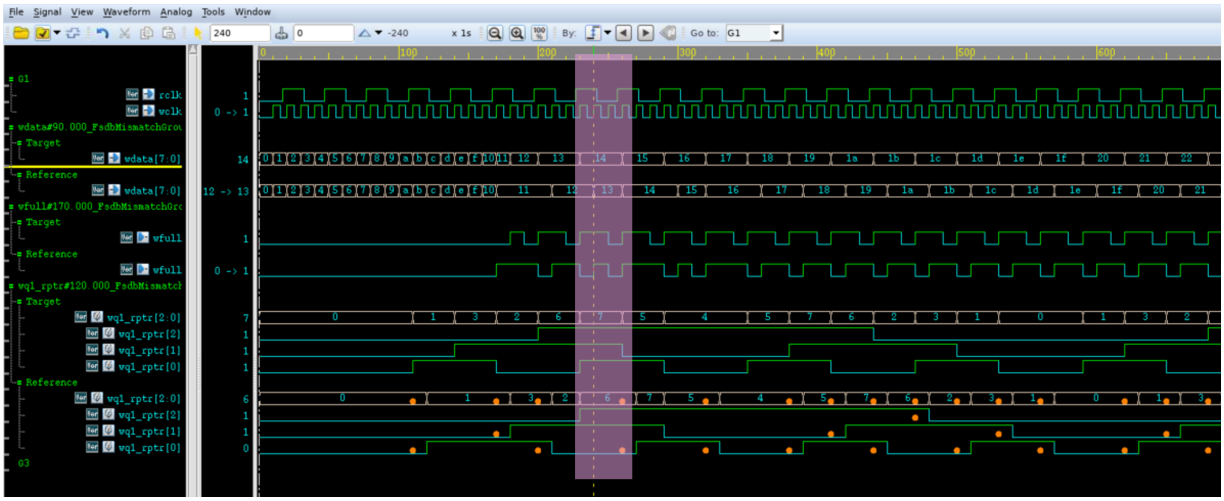


Figure 4.26: Metastability injection effect on Full flag

Figure 4.27: Root cause tracing for full flag failure

results, it's imperative to delve deeper into the domain. Additional studies are recommended to standardize a flow and identify potential bugs that may not have surfaced during the initial trials.

We suppose that the lack of conclusive results could be attributed to the existing approaches. That is why we highlight the need for innovative methodologies. Investing in further research can uncover new approaches for metastability injection that prove more effective in ensuring design immunity against metastability issues.

Moreover, there is an intriguing prospect in the field of formal verification. Developing a metastability model tailored for formal verification becomes essential. This model would abstract the notion of fictive time, a concept prevalent in current simulation-centric approaches. This exploration could pave the way for more comprehensive and accurate verification methods in dealing with metastability.

4.6 Conclusion

The chapter effectively synthesizes our studies and efforts in the metastability injection domain. Initially, we gained firsthand insights into metastability by reproducing the phenomenon in analog simulations, offering a nuanced understanding of related definitions. We observed a prolongation in the propagation time, causing an impact on the output Q of the slave latch when violating the setup time of the receiving flip flop.In the second section, we delved into modeling the effects of metastability at both analog and digital levels. The provided generic modeling for CDC paths helped distinguish where metastability could and could not occur, leading to a conclusive abstraction of the phenomenon on the digital level. Here, we interpreted metastability as a mere delay or advancement in the crossing signal. Through fault injection in simulations, we demonstrated how metastability can compromise coded synchronization protocols, as shown in the asynchronous FIFO case. Notably, metastability injection in RTL simulations uncovered re-convergences overlooked by CDC static tools due to substantial sequential depth. Despite lacking insights on optimal application and evaluation metrics, our work serves as a promising starting point. It lays the groundwork for potential further exploration, offering a solid foundation for a comprehensive PhD study in this intriguing subject.

General conclusion

During my three-year PhD program at STMicroelectronics and TIMA Laboratory, my primary responsibility was to investigate and enhance the verification of asynchronous interfaces on multi-clock systems. As a member of the CPU team at STMicroelectronics, our initial focus centered on validating high-performance CPU subsystems with multiple cores and shared peripherals. The critical challenge of Clock Domain Crossing (CDC) in multi-clock systems posed a potential threat to silicon integrity, necessitating early consideration in the design flow. Our mission aimed to optimize CDC verification, introducing novel aspects to elevate the quality of verification. The comprehensive results yielded positive outcomes, successfully addressing the targeted aspects.

Pre-PhD Context

CDC verification, traditionally a standalone process in design and verification plans, is typically isolated in the industry, relying on specialized tools. However, when we commenced our mission, the status of this verification activity was far from mature.

CDC Structural Verification The current approach involved CDC structural verification, conducted either at the RTL or gate level. It sought to statically identify all CDC paths and ensure the presence of suitable synchronizers on each path. This involved matching pre-coded synchronizer patterns with the design RTL and categorizing each based on the matched pattern. Challenges emerged in this approach, notably with manually written and error-prone design constraints. These constraints, forming the basis of structural verification, lacked cross-verification due to the absence of reliable references and specifications. The clock tree constraining task, for instance, consumed months before initiating structural verification, resulting in noisy and error-ridden outcomes. Additionally, the native flow neglected important aspects such as constraints interdependencies, power management logic, and establishing clear CDC coverage. Furthermore, the pre-coded patterns struggled with custom or complex synchronizers, relying on partial detection that could lead to undetected errors with catastrophic consequences on silicon.

CDC Functional Verification Complementing structural verification, CDC functional verification addresses the limitation of solely relying on structural analysis. While structural verification identified CDC paths and synchronizers, it fell short in validating their functionality. Efforts in CDC formal verification were observed in the state of the art, yet the large-scale, multi-clock domain nature of CDC often rendered formal analysis inconclusive due to timeouts. Consequently, due to the unachievability of satisfying results and the impossibility to do such an analysis within the projects' timeframes, no CDC functional check was ever done on projects.

PhD Contributions

In the course of this doctoral study, the primary objective was to identify and address existing limitations in the field of Clock Domain Crossing (CDC) verification. Early in this endeavor, we questioned the singular and immature approach often employed in CDC verification executed by designers themselves. The rarity of dedicated verification engineers for CDC verification contributes to its overall immaturity. Unlike functional verification, which has a wealth of literature, clear coverage definitions, and multiple IEEE standards, CDC verification's literature is limited, lacking any IEEE standard. To address these shortcomings, our focus shifted towards enhancing existing structural verification methods and exploring connections with functional verification. By tapping into the mature methodologies of functional verification, we aimed to elevate the maturity of CDC verification. The contributions of this PhD can be briefly summarized into three main achievements.

Optimization of CDC Structural Verification Flow The initial goal of this phase was to enhance and standardize the CDC structural verification flow, serving as the primary defense against the risks posed by asynchronous paths (refer to Chapter2). This objective materialized through two key proposals. Firstly, the introduction of a **New CDC Structural Verification Flow** involved the development of an innovative approach to address deficiencies in the original flow. This new flow systematically verified various design and CDC rules, aiming to improve violation reporting, expedite task completion, and minimize false positives and negatives. When comparing the old and proposed flows for our test case, approximately 75% of the erroneous results vanished. Secondly, the **UPF-Aware CDC Structural Verification** initiative investigated the impact of low-power logic on CDC aspects. Recognizing that low-power logic could compromise existing synchronized CDCs or introduce new ones, we introduced the "UPF-Aware CDC Verification Flow." This flow involved incorporating the UPF file to integrate low-power logic into the verified design. The focus was on updating tools to recognize new clock declarations and identify potential asynchronous control signals controlling isolation cells. The application of this flow successfully uncovered CDC paths that remained undetected by the original flow.

Integrating Dynamic Functional Verification with CDC Verification In contrast to the isolated nature of CDC verification, a highly skilled team of engineers routinely develops mature environments for dynamic functional verification, achieving notable functional and code coverage. This prompted the exploration of a connection between dynamic functional verification and CDC verification, leveraging the efforts invested in the former (refer to Chapter 3). Three key proposals emerged within this context. Firstly, the introduction of the **CDC Semi-Formal Verification Flow** involved assertions-based verification within a high-coverage dynamic functional verification environment. This approach ensured the validity of the setup and reported synchronizers. Notably, our semi-formal flow achieved 100% coverage of CDC assertions and unearthed new CDC bugs not identified by structural verification. Given the time efficiency and coverage achieved, the semi-formal approach for CDC verification demonstrated advantages over the formal approach results present in the state-of-the-art. Secondly, the **Universal Qualifier** presented a unified method capable of verifying all CDC data synchronizers, irrespective of their types. This marked a significant advancement in verifying custom data synchronizers that

industrial tools often struggle to detect. Results indicated that our method generated more reliable properties than those based on the native outputs of industrial tools. Finally, the **Hybrid Flow** utilized semi-formal verification not only to complement CDC structural verification but also to assist in the setup phase, particularly concerning clock tree configuration. This provided a solution to the mixed-mode verification approach (where clocks are propagated randomly without referring to any realistic operating mode) commonly used due to the absence of design specification. The outcomes revealed a substantial gap in the number of detected CDC paths by the hybrid flow compared to the mixed-modes approach. The hybrid flow identified five times more CDC paths, paving the way for an iterative approach where various configurations are iteratively checked, as opposed to a mixed-modes approach with incoherent and unrealistic clock propagation.

Pioneering the First IEEE Standard for CDC Models CDC models serve as concise representations of blocks and sub-blocks, encapsulating essential CDC information crucial for verifying CDC aspects at the periphery of a block integrated into a larger design context. In hierarchical CDC verification, these models are integrated from different blocks to verify the top level, offering a more efficient approach for extensive designs. However, the lack of portability between various CDC static verification tools hinders widespread adoption, primarily due to the absence of a standard for these CDC models. In an important initiative, a dedicated working group was established by Accellera to define the inaugural **IEEE CDC standard**. Our involvement in this CDC working group resulted in the development and release of two versions of this Language Reference Manual (LRM), both subjected to public review. This pioneering effort marks a significant step towards establishing standardized CDC models.

Future Directions

In the forthcoming work, our aim is to integrate all proposed solutions into a coherent and unified flow. The Hybrid Flow, in synergy with the New CDC Verification Flow, will contribute to configuring clock trees reliably and supporting the adoption of an incremental multi-mode CDC verification approach. This integration will pave the way for defining a straightforward CDC coverage metric, measuring the percentage of detected CDC in each mode, as opposed to the previous mixed-mode approach lacking a clear definition for CDC coverage.

Additionally, we plan to push for the implementation of the Universal Qualifier concept in verification tools, enabling the verification and analysis of any CDC data synchronizer, regardless of its type. This implementation will significantly enhance the quality and reliability of automatically generated protocol assertions.

Addressing the aspect of metastability injection (refer to Chapter 4), our initial study has provided a unique synthesis of the metastability problem on both analog and digital levels. We have explored various metastability injection approaches proposed by tools providers and researchers. A preliminary application of metastability injection has demonstrated its potential in detecting CDC re-convergences that were previously undetectable statically by structural verification. Despite time constraints, we recognize the untapped potential in the domain of metastability injection, particularly in dynamic simulation and formal verification. For that, we started developing a metastability formal model (refer to **??**) that can be the first milestone in this subject. Future exploration in these areas holds promise for further discoveries and advancements.

Publications and Conferences

Throughout my PhD, I had the opportunity to contribute significantly to various conferences, showcasing our findings through publications in scientific and industrial papers. Additionally, I actively participated in diverse seminars and webinars across different contexts. Here is a synthesis of our substantial contributions over the three years of my thesis.

Published Scientific conference papers

[1] Kalel, D., Brignone, J. C., Serre, I., Massicot, J., Avezou, J. (2023, June). UPF-Aware CDC Structural Verification on RTL. In 2023 21st IEEE Interregional NEWCAS Conference (NEWCAS) (pp. 1-2). IEEE.

[2] Kalel, D., Brignone, J. C., Fesquet, L., Morin-Allory, K. (2023, December). A Generic CDC Modeling for Data Stability Verification. In 2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS) (pp. 1-4). IEEE.

Published Industrial conefrence papers

[3] Kalel D, Brignone JC, Massicot J, Avezou J. Generating and Elaborating CDC-RDC UPF Aware Constraints with VC-Spyglass. Synopsys SNUG Europe 2022 Oct 17-18 (TECHNICAL COMMITTEE AWARD, **HONORABLE MENTION**)

[4] Kalel D, Brignone JC, Avezou J, Implementation of A Universal Qualifier Detection Algorithm to Assist the CDC Verification. Synopsys SNUG Silicon Valley 2024 Mar. 20-21 (TECHNICAL COMMITTEE AWARD, **THIRD PLACE BEST PAPER**)

[5] Massicot J, Perret F, Kalel D, Brignone, Avezou J, Multi-scenario CDC SAM Generation. Synopsys SNUG Europe 2024

Published CDC IEEE standard

[6] Clock Domain Crossing Standard Draft Version 0.1, https://accellera.org/images/downloads/drafts-review/Clock_Domain_Crossing_Standard_Version_0.1.pdf

Delivered Tutorial sessions and Seminars

[7] Accellera CDC WorkingGroup, Hierarchical CDC Closure with Standard Abstract models, Tutorial Session, DVCON US 2024

[8] Accellera CDC training SWG, CDC-RDC closure with Abstracts from Different tools, Tutorial Session, DVCONEurope 2023

[9] UPF-Aware CDC Constraints generation Flow, Synopsys Symposium Grenoble 2022

[10] Complementary Semi-Formal Analysis for the CDC Structural Verification, Siemens U2U, Munich 2023

[11] Complementary Semi-Formal Analysis to Assist the CDC Structural Verification Constraining the Clock Tree, Synopsys Symposium Grenoble 2023

[12] CDC Semi-Formal Verification, Siemens Forum Grenoble 2023

[13] CDC Static Verification Tools Evaluation, ST forums and BCD days, Grenoble, Castelleto, Catania 2023

[14] Constraints-Driven CDC and RDC Verification Including UPF Aware Analysis, Synopsys-STM Webinar, https://www.synopsys.com/verification/resources/webinars/constraints-driven-cdc-and-rdc-verification.html

Bibliography

[1] S. Churiwala and S. Garg, *Principles of VLSI RTL design: a practical guide*. Springer Science and Business Media, 2011.

[2] H. Jacobson, *Asynchronous circuit design: a case study of a framework called ACK*, 1996.

[3] S. Smith and J. Di, *Designing asynchronous circuits using NULL convention logic (NCL)*. Springer Nature, 2022.

[4] D. Geer, "Is it time for clockless chips?[asynchronous processor chips]," *Computer*, vol. 38, no. 3, pp. 18–21, 2005.

[5] S. F. Smith, *A multiple-clock-domain bus architecture using asynchronous FIFOs as elastic elements*. University of Idaho, 2003.

[6] W. J. Dally, W. J. Dally, and J. W. Poulton, *Digital systems engineering*. Cambridge university press, 1998.

[7] P. Teehan, M. Greenstreet, and G. Lemieux, "A survey and taxonomy of gals design styles," *IEEE Design and Test of Computers*, vol. 24, no. 5, pp. 418–428, 2007.

[8] J. Shinde and S. Salankar, "Clock gating—a power optimizing technique for vlsi circuits," in *2011 annual IEEE India conference*. IEEE, 2011, pp. 1–4.

[9] P. Zhao, Z. Wang, and G. Hang, "Power optimization for vlsi circuits and systems," in *2010 10th IEEE International Conference on Solid-State and Integrated Circuit Technology*. IEEE, 2010, pp. 639–642.

[10] V. Konstantakos, K. Kosmatopoulos, S. Nikolaidis, and T. Laopoulos, "Measurement of power consumption in digital systems," *IEEE Transactions on Instrumentation and measurement*, vol. 55, no. 5, pp. 1662–1670, 2006.

[11] S. Beer, R. Ginosar, R. Dobkin, and Y. Weizman, "Mtbf estimation in coherent clock domains," in *2013 IEEE 19th International Symposium on Asynchronous Circuits and Systems*. IEEE, 2013, pp. 166–173.

[12] R. Melchiorre, "A study of metastability in cmos latches," Ph.D. dissertation, Lehigh University, 1992.

[13] Y. Mirsky, "Comprehensive and automated static tool based strategies for the detection and resolution of reset domain crossings," *DVCon US*, 2016.

[14] M. Fawzy, A. Elgohary, and H. Ibrahim, "Noise reduction in reset domain crossings verification using formal verification," in *2020 IEEE East-West Design & Test Symposium (EWDTS)*. IEEE, 2020, pp. 1–5.

[15] C. Kwok, P. Viswanathan, and P. Yeung, "Addressing the challenges of reset verification in soc designs," in *Design and Verification Conference and Exhibition United States*, 2015.

[16] J.Faucher and JC.Brignone, "An optimization of rdc structural checks flo," in *SNUG Synopsys User Group Munich Germany*. Synopsys, 2019.

[17] R. Chadha and J. Bhasker, *An ASIC low power primer: analysis, techniques and specification*. Springer Science and Business Media, 2012.

[18] D. C. U. G. Synopsys and O. U. Guide, "Ver," *D-2010.03*, 2010.

[19] S. Kotha and D. Orne, "Generic manufacturing strategies: a conceptual synthesis," *Strategic management journal*, vol. 10, no. 3, pp. 211–231, 1989.

[20] S. Gayathri and T. Taranath, "Rtl synthesis of case study using design compiler," in *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*. IEEE, 2017, pp. 1–7.

[21] P. Kurup and T. Abbasi, *Logic synthesis using Synopsys®*. Springer Science and Business Media, 2012.

[22] S. Vasudevan, *Effective functional verification: principles and processes*. Springer Science and Business Media, 2006.

[23] L. Pierre and T. Kropf, *Correct Hardware Design and Verification Methods: 10th IFIP WG10. 5 Advanced Research Working Conference, CHARME'99, Bad Herrenalb, Germany, September 27-29, 1999, Proceedings*. Springer, 2003.

[24] J. Yuan, C. Pixley, and A. Aziz, *Constraint-based verification*. Springer Science and Business Media, 2006.

[25] K. Haripriya, A. SOMKUWAR, and L. KUMRE, "Low power checks in multi voltage designs," *Wseas Transactions on Electronics*, vol. 11, pp. 105–111, 2020.

[26] S. Verma and A. S. Dabare, "Understanding clock domain crossing issues," *EE Times*, 2007.

[27] S. Chaturvedi, "Static analysis of asynchronous clock domain crossings," in *2012 Design, Automation and Test in Europe Conference Exhibition (DATE)*. IEEE, 2012, pp. 1122–1125.

[28] S. Sarwary and S. Verma, "Critical clock-domain-crossing bugs," *EDN*, vol. 53, no. 7, pp. 55–64, 2008.

[29] K. Salah, "A uvm-based smart functional verification platform: Concepts, pros, cons, and opportunities," in *2014 9th International Design and Test symposium (IDT)*. IEEE, 2014, pp. 94–99.

[30] H. Height, *A practical guide to adopting the universal verification methodology (UVM)*. Lulu. com, 2012.

[31] "Ieee standard for universal verification methodology language reference manual," *IEEE Std 1800.2-2020 (Revision of IEEE Std 1800.2-2017)*, pp. 1–458, 2020.

[32] J. Bergeron, *Writing testbenches: functional verification of HDL models*. Springer Science and Business Media, 2012.

[33] S. K. Roy and S. Ramesh, "Functional verification of system on chips-practices, issues and challenges," in *Proceedings of ASP-DAC/VLSI Design 2002. 7th Asia and South Pacific Design Automation Conference and 15h International Conference on VLSI Design*. IEEE, 2002, pp. 11–13.

[34] E. Seligman, T. Schubert, and M. A. K. Kumar, *Formal verification: an essential toolkit for modern VLSI design*. Elsevier, 2023.

[35] C. E. Cummings, "Clock domain crossing (cdc) design & verification techniques using systemverilog," *SNUG-2008, Boston*, 2008.

[36] A. Danese, T. Ghasempouri, and G. Pravadelli, "Automatic extraction of assertions from execution traces of behavioural models," in *2015 Design, Automation and Test in Europe Conference Exhibition (DATE)*. IEEE, 2015, pp. 67–72.

[37] S. Beer and R. Ginosar, "An extended metastability simulation method; extended node short simulation (enss)," in *2012 IEEE 27th Convention of Electrical and Electronics Engineers in Israel*. IEEE, 2012, pp. 1–4.

[38] D. Chen, D. Singh, J. Chromczak, D. Lewis, R. Fung, D. Neto, and V. Betz, "A comprehensive approach to modeling, characterizing and optimizing for metastability in fpgas," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, 2010, pp. 167–176.

[39] R. Ginosar, "Metastability and synchronizers: A tutorial," *IEEE Design and Test of Computers*, vol. 28, no. 5, pp. 23–35, 2011.

[40] S. Golson, "Synchronization and metastability," *SNUG Silicon Valley*, 2014.

[41] T. J. Chaney and C. E. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," *IEEE Transactions on computers*, vol. 100, no. 4, pp. 421–422, 1973.

[42] M. Thakur, B. B. Soni, P. Gaur, and P. Yadav, "Analysis of metastability performance in digital circuits on flip-flop," in *2014 International Conference on Communication and Network Technologies*. IEEE, 2014, pp. 265–269.

[43] G. Plassan, "Conclusive formal verification of clock domain crossing properties," Ph.D. dissertation, Université Grenoble Alpes, 2018.

[44] S. Hatture and S. Dhage, "Multi-clock domain synchronizers," in *2015 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC)*. IEEE, 2015, pp. 0403–0408.

[45] R. Kol and R. Ginosar, "Adaptive synchronization for multi-synchronous systems," in *ICCD*, vol. 98. Citeseer, 1998, pp. 188–198.

[46] N. Karimi and K. Chakrabarty, "Detection, diagnosis, and recovery from clock-domain crossing failures in multiclock socs," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 32, no. 9, pp. 1395–1408, 2013.

[47] M. Herlev, C. K. Poulsen, and J. Sparsø, "Open core protocol (ocp) clock domain crossing interfaces," in *2014 NORCHIP*. IEEE, 2014, pp. 1–6.

[48] J. Cox, T. Chaney, and D. Zar, "Simulating the behavior of synchronizers," *white paper, www. blendics. com*, 2011.

[49] T. Mak, "Truncation error analysis of mtbf computation for multi-latch synchronizers," *Microelectronics Journal*, vol. 43, no. 2, pp. 160–163, 2012.

[50] H. J. Veendrick, "The behaviour of flip-flops used as synchronizers and prediction of their failure rate," *IEEE Journal of Solid-State Circuits*, vol. 15, no. 2, pp. 169–176, 1980.

[51] M. Kebaili, K. Morin-Allory, J.-C. Brignone, and D. Borrione, "Enabler-based synchronizer model for clock domain crossing static verification," *Languages, Design Methods, and Tools for Electronic System Design: Selected Contributions from FDL 2015*, pp. 103–124, 2016.

[52] R. Ginosar, "Fourteen ways to fool your synchronizer," in *Ninth International Symposium on Asynchronous Circuits and Systems, 2003. Proceedings.* IEEE, 2003, pp. 89–96.

[53] S. Zaychenko, P. Leshtaev, B. Gureev, and M. Shliakhtun, "Structural cdc analysis methods," in *2016 IEEE East-West Design & Test Symposium (EWDTS)*. IEEE, 2016, pp. 1–4.

[54] A. Ivanov, "Method of counter states encoding for passing the state jumps between asynchronous clock domains," in *2022 Austrochip Workshop on Microelectronics (Austrochip)*. IEEE, 2022, pp. 33–36.

[55] N. Karimi and K. Chakrabarty, "Detection, diagnosis, and recovery from clock-domain crossing failures in multiclock socs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 9, pp. 1395–1408, 2013.

[56] C. Leong, P. Machado, V. Bexiga, J. P. Teixeira, I. C. Teixeira, J. Silva, P. Lousã, and J. Varela, "Built-in clock domain crossing (cdc) test and diagnosis in gals systems," in *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*. IEEE, 2010, pp. 72–77.

[57] S. Chaturvedi, "Static analysis of asynchronous clock domain crossings," in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2012, pp. 1122–1125.

[58] S. Churiwala, "Tackling multiple clocks in socs," *EE Times March*, vol. 15, 2004.

[59] A. B. Chong, "Clock domain crossing verification challenges," in *2021 2nd International Conference on Electronics, Communications and Information Technology (CECIT)*, 2021, pp. 383–387.

[60] G. Vrinda, K. Mohammad, and J. Mohandas, "Towards improving clock domain crossing verification for socs," *Journal of electrical and electronics engineering*, vol. 14, no. 2, pp. 19–24, 2021.

[61] W. M. Trochim, "Outcome pattern matching and program theory," *Evaluation and program planning*, vol. 12, no. 4, pp. 355–366, 1989.

[62] A. B. Mehta and A. B. Mehta, "Clock domain crossing (cdc) verification," *ASIC/-SoC Functional Design Verification: A Comprehensive Guide to Technologies and Methodologies*, pp. 149–166, 2018.

[63] A. Cunningham and I. Sobanski, "Implementation of a closed loop cdc verification methodology," 2023.

[64] P. Sharma and S. K. Patel, "An automation methodology for amelioration of spyglasscdc abstract view generation process," in *2021 6th International Conference for Convergence in Technology (I2CT)*. IEEE, 2021, pp. 1–5.

[65] A. B. Chong, "Clock domain crossing verification challenges," in *2021 2nd International Conference on Electronics, Communications and Information Technology (CECIT)*. IEEE, 2021, pp. 383–387.

[66] M. Kebaili, K. Morin-Allory, J. Brignone, D. B. Mejid Kebaili, J.-C. Brignone, K. Morin-Allory, and D. Borrione, "Enabler-based synchronizer model for clock domain crossing static verification," in *2015 Forum on Specification and Design Languages (FDL)*, 2015, pp. 1–7.

[67] M. M. Rakshith and S. S. Hiremath, "Design and analysis of clock domain crossing using vc spyglass tool," 2023.

[68] M. Kasim, V. Gupta, and M. Jebin, "Methodology for detecting glitch on clock, reset and cdc path," in *2020 5th International Conference on Communication and Electronics Systems (ICCES)*. IEEE, 2020, pp. 300–304.

[69] G. Tarawneh, A. Mokhov, and A. Yakovlev, "Formal verification of clock domain crossing using gate-level models of metastable flip-flops," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*. IEEE, 2016, pp. 1060–1065.

[70] G. Tarawneh and A. Mokhov, "Xprova: Formal verification tool with built-in metastability modeling," in *2017 17th International Conference on Application of Concurrency to System Design (ACSD)*. IEEE, 2017, pp. 74–79.

[71] G. Plassan, H.-J. Peter, K. Morin-Allory, F. Rahim, S. Sarwary, and D. Borrione, "Conclusively verifying clock-domain crossings in very large hardware designs," in *2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2016, pp. 1–6.

[72] M. Kebaïli, G. Plassan, J. Brignone, and J. Binois, "Conclusive formal verification of clock domain crossings using spyglass-cdc," *SNUG France*, 2016.

[73] M. Kebaili, "Réflexions autour de la méthodologie de vérification des circuits multi-horloges: analyse qualitative et automatisation," Ph.D. dissertation, Université Grenoble Alpes, 2017.

[74] M. Kebaili, J.-c. Brignone, and K. Morin-Allory, "Clock domain crossing formal verification: a meta-model," in *2016 IEEE International High Level Design Validation and Test Workshop (HLDVT)*. IEEE, 2016, pp. 136–141.

[75] S. Chalana, S. Mitra, S. Bharath, R. Bhimireddy, S. K. Manickam, and S. Kumar, "Enhancing coverage of clock domain crossing assertion verification leveraging formal," in *2023 IEEE Women in Technology Conference (WINTECHCON)*. IEEE, 2023, pp. 1–6.

[76] A. Hudec, R. Ravasz, R. Ondica, and V. Stopjakova, "Selected issues causing metastability in digital circuits and approaches to avoiding them," in *2023 21st International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE, 2023, pp. 206–211.

[77] T. Chaney and C. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," *IEEE Transactions on Computers*, vol. C-22, no. 4, pp. 421–422, 1973.

[78] Y. Lee, N. Kim, J. B. Kim, and B. Min, "Millions to thousands issues through knowledge based soc cdc verification," in *2012 International SoC Design Conference (ISOCC)*. IEEE, 2012, pp. 391–394.

[79] J. M. Rabaey and M. Pedram, *Low power design methodologies*. Springer Science & Business Media, 2012, vol. 336.

[80] F. Bembaron, S. Kakkar, R. Mukherjee, and A. Srivastava, "Low power verification methodology using upf," *Proc. DVCon*, pp. 228–233, 2009.

[81] A. Srivastava, R. Mukherjee, E. Marschner, C. Seeley, and S. Dobre, "Low power soc verification: Ip reuse and hierarchical composition using upf," *DVCon proceedings*, 2012.

[82] J. Liu, M.-S. Hong, B. H. Lee, J. Choi, H. Won, K.-M. Choi, H. Vardhan, and A. Kher, "Low power verification with upf: Principle and practice," in *Conference on Electronics Systems Design and Verification Solutions, DVCON*, 2010.

[83] D. Kalel, J.-C. Brignone, I. Serre, J. Massicot, and J. Avezou, "Upf-aware cdc structural verification on rtl," in *2023 21st IEEE Interregional NEWCAS Conference (NEWCAS)*. IEEE, 2023, pp. 1–2.

[84] V. Gourisetty, H. Mahmoodi, V. Melikyan, E. Babayan, R. Goldman, K. Holcomb, and T. Wood, "Low power design flow based on unified power format and synopsys tool chain," in *2013 3rd Interdisciplinary Engineering Design Education Conference*. IEEE, 2013, pp. 28–31.

[85] E. Garat, D. Coriat, E. Beigné, and L. Stefanazzi, "Unified power format (upf) methodology in a vendor independent flow," in *2015 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. IEEE, 2015, pp. 82–88.

[86] P. Yeung, E. Marschner, and K. Liu, "Multi-domain verification: When clock, power and reset domains collide," *DVCon, March*, 2015.

[87] K. Takara, C. Kwok, N. Jain, and A. Hari, "Next-generation power aware cdc verification–what have we learned," in *Design and Verification Conference and Exhibition United States*, 2015.

[88] J. S. Nagendra and N. Ramavenkateswaran, "An overview of low power design implementation of a subsystem using upf," in *2020 International Conference on Inventive Computation Technologies (ICICT)*. IEEE, 2020, pp. 1042–1047.

[89] R. Sharafinejad, B. Alizadeh, and M. Fujita, "Upf-based formal verification of low power techniques in modern processors," in *2015 IEEE 33rd VLSI Test Symposium (VTS)*. IEEE, 2015, pp. 1–6.

[90] P. Khondkar, *Low-Power Design and Power-Aware Verification*. Springer, 2018.

[91] R. K. Sinha, A. Hari, and S. K. Khare, "Advancetechnique to accompolish power aware cdc verification," *DVCon Europe*, 2018.

[92] A. Initiative, "Universal verification methodology (uvm) 1.2 user's guide," *Accellera Systems Initiative: Elk Grove, CA, USA*, 2015.

[93] E. Hekkala, "Cdc logic verification in rtl design," B.S. thesis, E. Hekkala, 2023.

[94] L. Xiu, *From Frequency to Time-Average-Frequency: A Paradigm Shift in the Design of Electronic Systems*. John Wiley & Sons, 2015.

[95] L. Benini and G. DeMicheli, *Dynamic power management: design techniques and CAD tools*. Springer Science & Business Media, 1997.

[96] E. G. Friedman, "Clock distribution networks in synchronous digital integrated circuits," *Proceedings of the IEEE*, vol. 89, no. 5, pp. 665–692, 2001.

[97] M. C. Chi and S.-H. Huang, "A reliable clock tree design methodology for asic designs," in *Proceedings IEEE 2000 First International Symposium on Quality Electronic Design (Cat. No. PR00525)*. IEEE, 2000, pp. 269–274.

[98] D.-J. Lee and I. L. Markov, "Multilevel tree fusion for robust clock networks," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2011, pp. 632–639.

[99] J.-F. Vizier, D. Ramaekers, and Z. H. Zhou, "Verifying clock-domain crossing at rtl ip level using coverage-driven methodology."

[100] J. Cox, G. Engel, D. Zar, and I. W. Jones, "Synchronizers and data flip-flops are different," in *2015 21st IEEE International Symposium on Asynchronous Circuits and Systems*. IEEE, 2015, pp. 19–20.

[101] L.-S. Kim and R. W. Dutton, "Metastability of cmos latch/flip-flop," *IEEE Journal of solid-state circuits*, vol. 25, no. 4, pp. 942–951, 1990.

[102] S. Friedrichs, M. Függer, and C. Lenzen, "Metastability-containing circuits," *IEEE Transactions on Computers*, vol. 67, no. 8, pp. 1167–1183, 2018.

[103] J. Schmaltz, “A formal model of clock domain crossing and automated verification of time-triggered hardware,” in *Formal Methods in Computer Aided Design (FMCAD’07)*. IEEE, 2007, pp. 223–230.

List of Figures

List of Tables

A

Annex 1 : Design Rules

This annex is a non exhaustive list of the setup violations a user can need to solve before starting a CDC structural verification.

The setup rules, referenced in Chapter 2 as the design rules Π, is the set of rules that the design specification should respect in order to ensure the design integrity prior to the CDC structural verification. This is a non-exhaustive list of these design rules:

- **ERROR_CLKPROP_NO_CLK :** No clock definitions found
- **ERROR_CLOCK_UNDECL :** Clock net does not receive any defined clock
- **ERROR_CLOCK_CONSTANT :** Clock net is set to a constant
- **ERROR_ASYNC_CLOCK_OVERLAP:** Two or more clocks from different domains overlap
- **ERROR_CLOCK_GLITCH :** Asynchronous source converges with different domain clock(s)
- **ERROR_CLOCK_MODULEGLITCH:** specified module driven by clock(s) contains glitchy logic
- **ERROR_PORT_UNCONSTRAINED:** Port has no constraints
- **ERROR_PORT_PARTIALLY_CONSTRAINED:** Port is partially constrained
- **ERROR_BBOXPIN_UNCONSTRAINED:** Reports black-box pins that are unconstrained or partially-constrained
- **ERROR_LIBCELLPIN_UNMODELLED:** Reports un-modeled library cell pin(s)
- **ERROR_LIBCELL_COMBO_DRIVEN_ASYNCPIN:** Reports library cell async pins driven by combinational logic
- **ERROR_CGLIBCELL_DEF_INCOMPLETE:** Clock gating cells missing complete definition
- **ERROR_INPUT_MULTICLOCK_LOAD:** Primary input signal is sampled by multiple clock-domains
- **ERROR_OUTPUT_MULTICLOCK_DRIVER:** Primary output signal is in the fan-out cone of multiple clock-domains
- **ERROR_CLOCK_GATING_UNSAFE:** An unsafe clock gating with enable and source clock detected
- **ERROR_MULT_ASYNC_CLK_MUX:** Multiple asynchronous clocks reaching MUX with dynamic select
- **ERROR_MULT_ASYNC_CLKGATE:** Multiple asynchronous clocks reaching logic gate
- **ERROR_COMBO_DRIVING_REDUNDANT_LOGIC:** Multiple clocks reaching logic gate with redundant logic in its fanout
- **ERROR_MUX_SELPIN_CONSTRAINED:** Multiple clocks reaching MUX with constant select

- **ERROR_MUX_OUTPUT_CLOCK:** Multiple clocks reaching MUX which has another clock defined on its output
- **ERROR_COMBO_OUTPUT_CLOCK:** Multiple clocks reaching logic gate which has another clock defined on its output
- **ERROR_CONST_DATA:** Sequential element with no set/reset pin that has data pin tied to a constant
- **ERROR_CONST_DATA_SET:** Sequential element with set pin that has data pin tied to high
- **ERROR_CONST_DATA_RESET:** Sequential element with reset pin that has data pin tied to low
- **ERROR_IGNORE_COMMAND:** Reports commands that are ignored partially/-fully due to error in specified options
- **ERROR_LIBCELL_CONNECTIVITY_MISMATCH:** Mismatch in path between functional and timing model found for cell module
- **ERROR_TIMING_ARC_PATH:** Path exists in timing model only
- **ERROR_FUNCTIONAL_PATH:** Path exists in functional model only
- **ERROR_LIBCELL_DEF_INCOMPLETE:** Instance of library cell module has pin(s) with incomplete definition
- **ERROR_OVERRIDE_COMMAND:** Reports if a command value is overridden
- **ERROR_RESET_ASSERT_MISSING:** Reports sequential elements that are not asynchronously asserted by active reset/set
- **ERROR_RESET_CONSTANT_ACTIVE:** Reset/set pin is tied to active constant value
- **ERROR_RESET_UNDECL:** Asynchronous reset/set input pin does not receive any asynchronous defined reset signal, defined by create_reset -async
- **ERROR_SYNCRESET_UNDECL:** Net does not receive any synchronous defined reset signal
- **ERROR_SDC_CLK_NOTFOUND:** SDC clock specified in SDC command does not exist
- **ERROR_SDC_CLKGRP_INVALID:** Same clock object used multiple times in set_clock_groups or clock group type is missing
- **ERROR_SDC_CLKNAME_MISSING:** Source object or -name missing in create_clock or create_generated_clock commands
- **ERROR_SDC_EMPTY_CLKGRP:** Empty clock groups detected
- **ERROR_SDC_OBJECT_NONEXIST:** Object specified in the SDC command does not exist

- **ERROR_SDC_OBJECT_NOTFOUND:** Objects specified with SDC command not found in design
- **ERROR_SDC_DUTYCYCLE_INVALID:** Duty cycle option is invalid in the create_generated_clock SDC command
- **ERROR_SDC_OPTIONS_INCOMPLETE:** Inter dependent options of SDC commands missing
- **ERROR_MUX_DRIVING_REDUNDANT_LOGIC:** Multiple clocks reaching MUX with redundant logic in its fanout
- **ERROR_RESET_SENSE_INVALID:** Command is ignored

B

Annex 2 : CDC Rules

This annex is a non exhaustive list of the CDC violations and the related CDC problems that can be detected by the CDC structural verification.

Contents

The CDC violations, referenced as the set of CDC rules $\Pi_{cdc} = \{\pi^i\}$ in Chapter 2, is the set of CDC properties that should be followed by the different CDC paths and synchronizers, such as gray coding for data bus synchronized by MFS. This is a non exhaustive list of the CDC rules:

CDC Synchronization checks

- **ERROR_CDC_UNSYNC_NOSCHEME:** Unsynchronized CDC Paths found
- **ERROR_CDC_UNSYNC_CTRL:** Partially matched control synchronization scheme found on the CDC path
- **ERROR_CDC_UNSYNC_DATA:** Partially matched data synchronization found on the CDC path

CDC unsynchronized checks

- **ERROR_CDC_UNSYNC_ASYNCRESET:** Partially matched synchronization scheme found for a CDC path crossing.
- **ERROR_CDC_ASYNCRESET_CTRLSYNC_INVALID_USE:** defined reset synchronizers drive an invalid (non set/reset) pin.
- **WARNING_CDC_COHERENCY_ASYNCRESET:** Reset signal is getting synchronized multiple times

CDC Convergence Checks

- **ERROR_CDC_COHERENCY_SINGLESRC_RECONV_COMB:** Combinational convergence of single source divergence found at convergence point
- **ERROR_CDC_COHERENCY_SINGLESRC_RECONV_SEQ:** Sequential convergence of single source divergence found at convergence point
- **ERROR_CDC_COHERENCY_RECONV_COMB:** Combinational convergence found at ConvergencePoint
- **ERROR_CDC_COHERENCY_RECONV_SEQ:** sequential convergence of multiple synchronized signals on which the check for gray encoding of source bits is skipped.
- **ERROR_CDC_COHERENCY_MULTI_SYNC:** Signals control synchronized multiple times in same domain without convergence.
- **WARNING_CDC_COHERENCY_MULTI_SYNC_SRCBUS:** Separate synchronization of different bits of same source bus at destination bus
- **WARNING_CDC_COHERENCY_BUS_NOCONV:** Control synchronized bits of same source vector signal that are neither converging nor checked for presence of gray encoder
- **WARNING_CDC_COHERENCY_VECTOR_DIFF_SYNC:** Mismatch in synchronization among different bits of the same vector source.

- **ERROR_CDC_COHERENCY_ASYNCSRCS_RECONV_COMB:** Combinational convergence of multiple synchronized signals from mutually asynchronous sources
- **ERROR_CDC_COHERENCY_ASYNCSRCS_RECONV_SEQ:** Sequential convergence of multiple synchronized signals from mutually asynchronous sources
- **ERROR_CDC_META_DESIGN_HIER:** Default instance name is used for CDC jitter model as meta_design_hier command is not specified or specified value for the name of meta_design_hier command does not match with the current design name.
- **ERROR_CDC_COHERENCY_IGNORED:** Ignored convergence due to -ignore_among_singals argument of the configure_cdc_convergence command.
- **ERROR_CDC_SEQCONV_ASYNC_SRCS:** Sequential convergence found.

CDC Glitch Checks

- **ERROR_CDC_GLITCH_CTRL:** Glitch found on control crossing paths
- **ERROR_CDC_GLITCH_UNSYNC:** Glitch found on unsynchronized crossing paths.
- **WARNING_INTEGRITY_RESET_GLITCH:** PinType pin of sequential element DestReset is driven by combinational logic.

C

Annex 3 : Constraints Assertions Examples

This annex is a non exhaustive list of the constraints that can be translated to properties used to either constraint the design for formal verification, or to be cross checked by the functional dynamic simulation.

The CDC verification EDA tools translate the set of constraints specified by the user to a set of assumptions that can be cross checked in simulation environment or reused in a formal verification environment. Here as some examples of the constraints and their correspondent assertions.

- Constant signals constraining the clock tree

```
#(1)constant constraint
define_constant -value [0\1] -signal [signal\_name]
```

```
//(1)constant assertion
always@*
begin
assert_cdc_constant_prop : assert (select === value)
```

- Pseudo-static signals where CDC static analysis is skipped

```
#(2) pseudo-static signals constraint
define_pseudostatic -name [signal name]
-stopped_clock [yes/no] -under_reset [yes/no]
```

```
//(2) pseudo-static signals assertion
property pseudo_static (EN);
    @(posedge clk) disable iff(reset)
      nexttime $stable(EN);
endproperty
```

- Exclusive CDC signals that can converge in the design fanout

```
#(3) Exclusive signals constraints
define_exclusive -signals [set of signals names]
```

```
//(3) Exclusive signals
property mutex (data, clk);
    @(posedge clk)
      $onehot0(data ^ $past(data));
 endproperty
```

D

Annex 4 : Simulation Results

This annex is a synthesis of the simulation results we obtained verifying our CPU sub-system test case.

The following tables give insights on the simulation results and the assertions exercised in each of our dynamic functional regression tests.

Test acronym	Scope	static /400	Exclusive /59	Constant /169
Test 1-21	Basic CPU and SubSystem functionality	21	28	169
Test 22	Memory integration	140	30	169
Test 23-38	Interrupts	304	49	169
Test 39-56	Debug and trace	279	47	169
Test 57-68	System performance and power	361	59	169
Test 69-73	Clock and reset management	361	59	169
Test 74-79	Register programming	361	59	169
Test 80-84	Connectivity	361	46	169
Test 85-92	Memory integration	316	32	169
Test 93-96	Register verification	347	51	169
Test 97-104	Interface verification	361	59	169
Test 105-107	Custom IP verification	184	36	169
Test 108-130	Power aware functionality	400	48	169

Tab. D.1: Constraints assertions coverage wrt functional regression tests

Test acronym	Scope	MFS /150	Qualifier /90
Test 1-21	Basic CPU and SubSystem functionality	118	71
Test 22	Memory integration	105	40
Test 23-38	Interrupts	128	70
Test 39-56	Deebug and trace	137	78
Test 57-68	System performance and power	147	82
Test 69-73	Clock and reset management	150	86
Test 74-79	Register programming	142	78
Test 80-84	Connectivity	125	63
Test 85-92	Memory integration	138	45
Test 93-96	Register verification	123	84
Test 97-104	Interface verification	150	90
Test 105-107	Custom IP verification	67	47
Test 108-130	Power aware functionality	150	68

Tab. D.2: Protocols assertions coverage wrt functional regression tests

E

Annex 6: Asynchronous FIFO

This annex gives the verilog code of our asynchronous FIFO test case used for the metastability injection flow. We selected an asynchronous FIFO as our test case due to its comprehensive coverage of aspects related to Clock Domain Crossing (CDC).

FIFO memory

```
module fifomem #(parameter DATASIZE = 8,
// Memory data word width
 parameter ADDRSIZE = 4) // Number of mem address bits
 (output [DATASIZE-1:0] rdata,
 input [DATASIZE-1:0] wdata,
 input [ADDRSIZE-1:0] waddr, raddr,
 input wclken, wfull, wclk);
 `ifdef VENDORRAM
 // instantiation of a vendor's dual-port RAM
 vendor_ram mem (.dout(rdata), .din(wdata),
 .waddr(waddr), .raddr(raddr),
 .wclken(wclken),
 .wclken_n(wfull), .clk(wclk));
 `else
 // RTL Verilog memory model
 localparam DEPTH = 1<<ADDRSIZE;
 reg [DATASIZE-1:0] mem [0:DEPTH-1];
 assign rdata = mem[raddr];
 always @(posedge wclk)
 if (wclken && !wfull) mem[waddr] <= wdata;
 `endif
endmodule
```

Write address synchronization

```
module sync_w2r #(parameter ADDRSIZE = 4)
 (output reg [ADDRSIZE:0] rq2_wptr,
 input [ADDRSIZE:0] wptr,
 input rclk, rrst_n);
 reg [ADDRSIZE:0] rq1_wptr;
 reg [ADDRSIZE:0] rq1x_wptr;
 always @(posedge rclk or negedge rrst_n)
 if (!rrst_n) {rq2_wptr,rq1x_wptr,rq1_wptr} <= 0;
 else
 {rq2_wptr,rq1x_wptr,rq1_wptr} <= {rq1x_wptr,rq1_wptr,wptr};
endmodule
```

Read address synchronization

```
module sync_r2w #(parameter ADDRSIZE = 4)
 (output reg [ADDRSIZE:0] wq2_rptr,
 input [ADDRSIZE:0] rptr,
 input wclk, wrst_n);
 reg [ADDRSIZE:0] wq1_rptr;
 reg [ADDRSIZE:0] wq1x_rptr;
 always @(posedge wclk or negedge wrst_n)
 if (!wrst_n) {wq2_rptr,wq1x_rptr,wq1_rptr} <= 0;
 else
 {wq2_rptr,wq1x_rptr,wq1_rptr} <= {wq1x_rptr,wq1_rptr,rptr};
endmodule
```

Full flag calculation

```
module wptr_full #(parameter ADDRSIZE = 4)
 (output reg wfull,
 output [ADDRSIZE-1:0] waddr,
 output reg [ADDRSIZE :0] wptr,
 input [ADDRSIZE :0] wq2_rptr,
 input winc, wclk, wrst_n);

 reg [ADDRSIZE:0] wbin;
 wire [ADDRSIZE:0] wgraynext, wbinnext;

 // GRAYSTYLE2 pointer
 always @(posedge wclk or negedge wrst_n)
 if (!wrst_n) {wbin, wptr} <= 0;
 else {wbin, wptr} <= {wbinnext, wgraynext};
 // Memory write-address pointer
 //(okay to use binary to address memory)
 assign waddr = wbin[ADDRSIZE-1:0];
 assign wbinnext = wbin + (winc & ~wfull);
 `ifdef INJECT_GRAY_ERROR
   assign wgraynext =  wbinnext;
 `else
   assign wgraynext = (wbinnext>>1) ^ wbinnext;
 `endif

 //three necessary full-tests
 assign wfull_val=
 ((wgraynext[ADDRSIZE] !=wq2_rptr[ADDRSIZE])&&
 (wgraynext[ADDRSIZE-1] !=wq2_rptr[ADDRSIZE-1]) &&
 (wgraynext[ADDRSIZE-2:0]==wq2_rptr[ADDRSIZE-2:0]));

 always @(posedge wclk or negedge wrst_n)
 if (!wrst_n) wfull <= 1'b0;
 else wfull <= wfull_val;

endmodule
```

Empty flag calculation

```
module rptr_empty #(parameter ADDRSIZE = 4)

 (output reg rempty,
 output [ADDRSIZE-1:0] raddr,
 output reg [ADDRSIZE :0] rptr,
 input [ADDRSIZE :0] rq2_wptr,
 input rinc, rclk, rrst_n);
 reg [ADDRSIZE:0] rbin;
 wire [ADDRSIZE:0] rgraynext, rbinnext;
 //-------------------
 // GRAYSTYLE2 pointer
 //-------------------
 always @(posedge rclk or negedge rrst_n)
 if (!rrst_n) {rbin, rptr} <= 0;
 else {rbin, rptr} <= {rbinnext, rgraynext};
 // Memory read-address pointer
 //(okay to use binary to address memory)
 assign raddr = rbin[ADDRSIZE-1:0];
 assign rbinnext = rbin + (rinc & ~rempty);
`ifdef INJECT_GRAY_ERROR
   assign rgraynext = rbinnext;
`else
   assign rgraynext = (rbinnext>>1) ^ rbinnext;
`endif

 assign rempty_val = (rgraynext == rq2_wptr);

 always @(posedge rclk or negedge rrst_n)
 if (!rrst_n) rempty <= 1'b1;
 else rempty <= rempty_val;
endmodule
```

Advanced Structural and Semi-Formal Verification Flow for Clock Domain Crossing (CDC) in Asynchronous Multi-clock Systems

Résumé

Pendant mon doctorat, nous nous sommes concentrés sur l'amélioration de la vérification des interfaces asynchrones sur les systèmes multi-horloges. Notre mission était d'optimiser la vérification de la traversée de domaine d'horloge (CDC) sur RTL pour éviter les bugs potentiels qui pourraient affecter l'intégrité du silicium. Nous avons introduit de nouveaux aspects pour élever la qualité de la vérification et avons atteint avec succès leurs objectifs. Nous avons commencé par améliorer les méthodes de vérification structurelle existantes en introduisant le flot de vérification CDC UPF-Aware. Nous avons également cherché à intégrer la vérification fonctionnelle dynamique avec la vérification statique CDC à travers le flot de vérification semi-formelle CDC. Cela a conduit à la modélisation du "Qualificateur Universel", une méthode générique pour vérifier tous les types de synchroniseurs de données CDC, et du "Flot Hybride", une façon d'utiliser des assertions pour accélérer la contrainte de l'arbre d'horloge pour la vérification statique CDC. De plus, en collaboration avec Accellera, notre équipe a été pionnière dans la création de la première norme IEEE pour les modèles CDC, qui servent de représentations abstraites concises de blocs et de sous-blocs pour la vérification hiérarchique CDC. Dans les travaux futurs, nous visons à intégrer toutes les solutions proposées dans un flot cohérent et unifié. Nous prévoyons également de traiter l'injection de métastabilité et d'explorer son potentiel dans la simulation dynamique et la vérification formelle.

Mots-clés : CDC, domaine d'horloge, vérification, formelle, dynamique, statique, qualificateur, synchroniseur

Abstract

During my PhD, we focused on improving the verification of asynchronous interfaces on multi-clock systems. Our mission was to optimize Clock Domain Crossing (CDC) verification on RTL to avoid potential bugs that may affect silicon integrity. We introduced novel aspects to elevate the quality of verification and successfully met their goals. We started by enhancing existing structural verification methods introducing the UPF-Aware CDC Verification Flow. We were also interested to integrate the dynamic functional verification with CDC static verification through the CDC Semi-Formal Verification Flow. This led to the modeling of the "Universal Qualifier", a generic method to verify all types of CDC data synchronizers, and the "Hybrid Flow", a way to use assertions to accelerate the clock tree constraining for the CDC static verification. In addition, in collaboration with Accellera, our team pioneered the first IEEE standard for CDC models, which serve as concise CDC abstracted representations of blocks and sub-blocks for CDC hierarchical verification. In future work, we aim to integrate all proposed solutions into a coherent and unified flow. We also plan to address metastability injection and explore its potential in dynamic simulation and formal verification.

Keywords : CDC, Clock Domain Crossing, verification, formal, dynamic, static, qualifier, synchronizer