Design and Verification of Clock Domain Crossing Interfaces

Zaid Al-bayati

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Electrical & Computer Engineering)

at

Concordia University

Montréal, Québec, Canada

April 2012

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Zaid Al-bayati

Entitled: "Design and Verification of Clock Domain Crossing Interfaces"

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science

Complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

____ Chair
Dr. M. Z. Kabir

____ Examiner, External
Dr. J. Bentahar, CIISE To the Program

____ Examiner
Dr. G. Cowan

____ Examiner
Dr. Y. Savaria (Ecole Polytechnique)

____ Supervisor
Dr. O. Ait Mohamed

Approved by: ____
Dr. W. E. Lynch, Chair
Department of Electrical and Computer Engineering

____________20_____ ________________________________
Dr. Robin A. L. Drew
Dean, Faculty of Engineering and
Computer Science

ABSTRACT

Design and Verification of Clock Domain Crossing Interfaces

Zaid Al-bayati

The clock distribution network is an essential component in every synchronous digital system. The design of this network is becoming an increasingly sophisticated and difficult task due to the increasing logic capacity of chips and due to the fact that this network has to reach out to each and every memory element in the chip. Multi-clock domain circuits with Clock Domain Crossing (CDC) interfaces are emerging as an alternative to circuits with a global clock. The design of CDC interfaces is a challenging task due to the difficulty of dealing with two possibly unrelated clock domains and the possibility of propagating metastability into the communicating blocks making CDC interfaces difficult to design and verify. In this work, we present a hybrid FIFO-asynchronous method for constructing robust CDC interfaces. This method avoids the shortcomings of previous interfaces and provides reliable transfer of data and control signals between different clock domains. A complete design is proposed, fully implemented using 90nm TSMC CMOS technology, and simulated using SPICE. Extensive simulations confirmed the robustness of the interface at different temperatures, different workloads, and varying frequency ratios. The reported implementation provides a maximum throughput of 606 Mitems/s. Moreover, we also address the challenging task of the verification of CDC interfaces. Most RTL simulation tools available today are incapable of simulating these interfaces. In this thesis, we present a framework for the formal verification of CDC interfaces. The framework explicitly models metastability by taking advantage of the unique features of probabilistic model checking. The framework is applied to common CDC interfaces by verifying them using the PRISM model checker.

ACKNOWLEDGEMENTS

It has been an amazing experience to accomplish my Master's thesis in the Hardware Verification Group (HVG) at Concordia. It certainly would not have happened without the support and guidance of several people to whom I owe a great deal.

First of all, I would like to thank my supervisor, Dr. Otmane Ait Mohamed. He is the kind of professor and person any student would like to have as his supervisor. He is knowledgable, understanding, supportive, and present in all phases of my work. I am grateful that I had the opportunity to learn from him both in research and in life in general.

Secondly, I sincerely thank Prof. Yvon Savaria, for co-supervising my research work. Prof. Savaria has always given me expert advice, quality feedback and essential directions to continue this work. Also, I would like to thank Dr. Rafay Hasan, this thesis would not have been possible without his guidance, support and encouragements. I am thankful to him for all the time he has given me to introduce me into the topic and guide me throughout this work.

Next, I would like to thank all the members of HVG for being my family here. Especially, I would like to thank two incredible friends who were always there when I needed them, Ghaith Bany Hamad and Naeem Abbasi.

Last but not least, I would like to thank my family, especially my mother, for their constant moral support, encouragement and their prayers. Any words of compliment will never be enough to jusly appreciate their support during this work and my whole life. Their support was invaluable in completing this thesis.

To my mother, father, brother, and sister.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

BDD	Binary Decision Diagrams
CDC	Clock Domain Crossing
CSL	Continuous Stochastic Logic
CTMC	Continuous Time Markov Chain
DTMC	Discrete Time Markov Chain
FIFO	First-Input First-Output
FSM	Finite State Machine
GALS	Globally-Asynchronous Locally-Synchronous
MDP	Markov Decision Processes
MTBDD	Multi-Terminal Binary Decision Diagrams
MTBF	Mean Time Before Failure
PCTL	Probabilistic Computational Tree Logic
PSL	Property Specification Language
RTL	Register Transfer Level
SoC	System on Chip
SPICE	Simulation Program with Integrated Circuit Emphasis
STG	Signal Transition Graph

Chapter 1

Introduction

1.1 Motivation

Most digital systems in use today are built in a synchronous manner. A global clock signal passes through the whole chip and orchestrates its operation. However, as digital designs grow fast in size and complexity, it is becoming more and more difficult to provide a unified and accurate clock to the whole system. The clock distribution network is becoming a big headache for designers as it becomes more and more difficult to deal with large variations in clock signal arrival times at different locations in the chip. This is especially true for modern deep sub-micron technologies in the presence of interconnects of extremely varying lengths. Moreover, the distribution of clock signals at the high frequencies used by today's chips accounts for a considerable share of power consumption and chip area [5].

In order to avoid these problems, the concept of Globally-Asynchronous Locally-Synchronous (GALS) systems was developed. In this design paradigm, the system is composed of several parts (domains) each running on its own clock. Communication between these blocks is usually achieved with asynchronous interfaces known as Clock Domain Crossings (CDC). The CDC can be viewed as a wrapper that encapsulates locally synchronous domains and controls any interaction between different blocks. This design style is being employed more heavily in System-on-Chip (SoC) design. The ITRS2009 road map [4] states that by 2015 about 25% of long interconnects in a SoC will comprise asynchronous handshaking.

The design of CDC interfaces is an inherently challenging task. These interfaces must decouple the timing issues of the communicating blocks and provide a reliable transfer of data and control signals. The design of these interfaces is further complicated by the danger of propagating metastability into the communicating blocks. If the data input of a flip-flop comes from a clock domain that is different from its own, it might violate the setup and hold requirements of the flip-flop. These violations might lead to serious system errors if not handled properly. With proper design of CDC interfaces, the probability of such failures can be made negligible.

The existence of errors in the design is usually detected with verification tools. Verification methods are divided into functional (simulation) methods and formal methods. Simulation at the Register Transfer Level (RTL) is still the most widely used method. However, standard RTL simulation can not model the effect of metastability [1]. This could delay finding some CDC errors to late stages in the design cycle or worse these errors might not be found at all. Therefore, formal verification of CDC interfaces is the alternative needed for finding design bugs early in the design cycle. Appropriate use of formal verification in verifying CDC interfaces can substantially reduce CDC related errors.

This work tries to address CDC issues on both fronts. The design of reliable CDC interfaces is proposed. These interface try to minimize the probability of metastability-related failures in these interfaces. In terms of verification, a methodology for formal verification of CDC interfaces is proposed.

1.2 Metastability

Static storage elements in digital circuits typically have two stable operation points (logic '0' and logic '1'). When the data at the input of a level-triggered storage element such as a flip-flop changes, its output remains the same until the clock signal changes. With the change in clock, the new output is observed after a small propagation delay. The flip-flop operates normally as long as there is adequate timing separation between the change in its data input and its clock input. This period of time in which it is forbidden to change the data input is referred to as the setup time (before the clock edge) and the hold time (after the clock edge).

If the input does change during the forbidden zone the clock-to-output delay, referred to as T_{CQ}, will increase. In fact, as the input change gets closer to the clock edge, the flip-flop takes longer to respond because the energy supplied by the overlap between data and clock inputs gets less and less, so it takes longer and longer to decide [2]. This indecision of whether the output should be 0 or 1 is referred to as metastability [3]. During this decision period, the output of the flip-flop will be at an intermediate level between 0 and 1. This value is roughly mid-way between GND and VDD. However, exact voltage levels depend on transistor sizing as well as on process variations [3].

To better understand metastability in flip-flops, assume the master-slave flip-flop shown in Figure 1.1.

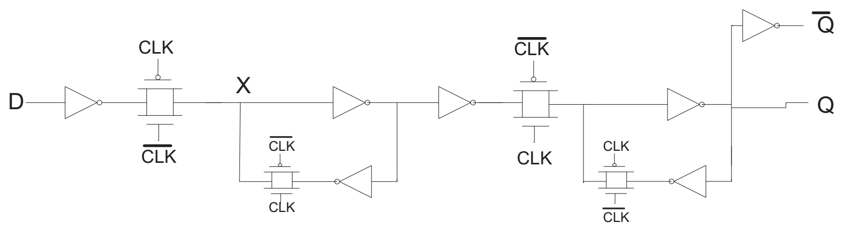

Figure 1.1: Master slave flip-flop

Assume that the D input is rising just before the rising edge of the clock. Initially CLK is low and $\overline{CLK}$ is high. As D starts to rise, the rise is propagated across the transmission gate and the node X starts to rise moving from logic 0 to logic 1. If the clock toggle occurs just as node X is around its metastable point (for the 90nm TSMC technology we used, this point was at about 0.4 VDD), then the cross-coupled inverter latch will be disconnected from the input. In this situation, node X will not have enough strength to force a clear value at the output of the inverter, the latch will therefore hover around its metastable level for an unknown period of time. The metastable value can propagate to the Q output causing a failure in the logic beyond.

Metastability does not only occur because the data and clock inputs change at the same time. Other situations might result in metastability such as when the voltage level of the input is not an appropriate logic 0 or logic 1. Metastability might also occur because of a badly timed clear or reset signal or because of a short clock pulse (due to bad clock gating) [3]. However, we will only focus on metastability caused by unsynchronized data since it is more common and more difficult to deal with than other causes of metastability.

A metastable flip-flop will eventually settle to either high or low logic value, however, the time taken to settle to one of the two values, known as the resolution time t_r, could be long. This time depends on the initial voltage difference between the two terminals of the cross-couples inverter latch and hence on the data arrival time. Figure 1.2 shows the time needed by a CMOS latch to converge to a stable value as a function of the data arrival time [9].

When data arrives at time t_{meta}, the latch requires the maximum time to resolve. Theoretically, the time required to resolve the output would be infinite [4] but practically, noise will force it to converge. Furthermore, the value to which a metastable flip-flop eventually settles is not known in advance. Environmental noise can push the flip-flop to either one of the two stable logic operation points. These

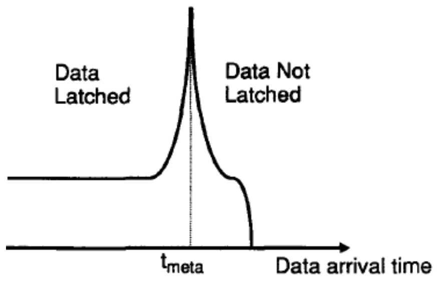

Figure 1.2: Time to generate a stable value vs data arrival time for a latch [9]

issues make the design of interfaces that inherently have the potential to become metastable difficult. This is especially true for clock domain crossings which occur at the intersection of two possibly unrelated clock domains.

1.3 Thesis Contribution

It is desirable to have a clock domain crossing that does not fall into metastability. However, the physical limitations of circuits make it impossible for flip-flops in these interfaces not to fall into metastability. A good design should reduce the probability of falling into metastability as much as possible and reduce the probability of its propagation into the communicating domains. In this work, a robust novel clock domain crossing interface is proposed. Furthermore, as the verification of CDC interfaces is necessry to reduce their errors and since simulation methods are incapable of showing metastability-related errors, a framework for the formal verification of CDC interfaces is proposed.

In terms of reviews of related work, we believe our contribution can be summarized as follows:

- The thesis proposes a novel hybrid CDC interface overcoming limitations in two previous CDC interfaces, namely pausable clocking and FIFO based interfaces.

- A novel circuit named as protocol-pauser is designed at the transistor level. This circuit constitutes the most critical part of the proposed CDC interface.

- All the circuits are implemented at the transistor level using 90nm TSMC CMOS technology. Extensive SPICE simulations under different settings are performed to analyze the performance and demonstrate the robustness of the interface.

- A new formal verification methodology for CDC interfaces is proposed. Unlike previous work on CDC verification, the new methodology explicitly models metastability and captures its probabilistic behavior by using Markov Decision Processes (MDP) which model both stochastic and non-deterministic behavior.

- Two common CDC interfaces; namely FIFO based interfaces and bundled data protocol based interfaces are verified using the new methodology. The proposed CDC interface is also verified using the new approach. MDP models for these interfaces are written and their properties are verified using the PRISM model checker.

1.4 Thesis Outline

The rest of the thesis is organized as follows:

- Chapter 2 provides background information on CDC interfaces and reviews some common CDC designs. The chapter also discusses the concepts of synchronizers and their Mean Time Before Failure (MTBF). Finally, the chapter provides a brief introduction into the verification of CDC interfaces and the existing techniques in this field.

- Chapter 3 discusses our proposed hybrid FIFO asynchronous CDC interface. The chapter elaborates on the overall protocol and then discusses the implementation details block-by-block focusing on the novel protocol-pauser design.

- Chapter 4 presents the results of electrical SPICE simulations for our proposed CDC interface. The proposed interface was simulated under varying workloads, different temperatures, different frequencies, and random phase shifts. The results of these experiments are shown in Chapter 4.

- Chapter 5 presents our verification methodology for CDC interfaces and discusses our new approach of using probabilistic model checking for verifying CDC interfaces. The chapter also discusses applying the methodology to three different CDC interfaces.

- The thesis concludes by summarizing the proposed work and providing some future research directions in Chapter 6.

Chapter 2

Background and Related Work

In this chapter, some background information necessary to understand the thesis are provided. A related work survey both in the design and verification of CDC interfaces is presented. The chapter starts by discussing an essential component in most CDC interfaces; namely the synchronizer, then in Section 2.2, a review of the most important existing CDC interfaces is given. A focus is made on those closely related to our proposed work. In Section 2.3, the verification of CDC interfaces is addressed while in Section 2.4, a review of the related works in the field of CDC verification is given. Section 2.5 summarizes this chapter.

2.1 The Synchronizer

Before going into the design details of these interfaces, a distinction must be made between three terms that are used extensively throughout this document:

- Globally-Asynchronous Locally-Synchronous (GALS) design: refers to the system-level design paradigm in which a system is composed of several blocks. Each block is synchronous by itself (runs on a single clock) but the blocks are asynchronous to each other (run on different clocks).

- Clock Domain Crossing (CDC): refers to the interfaces that connect the locally-synchronous blocks in a GALS system.
- Synchronizer: is a circuit-level term that refers to a device that samples an asynchronous signal and outputs a version of the signal that has transitions synchronized to a local or sample clock [6].

A GALS system might contain one or more CDCs which might contain one or more synchronizers.

The most common synchronizer employed today is the two-flop synchronizer shown in Figure 2.1.

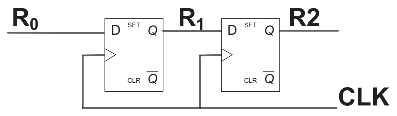

Figure 2.1: The two-flop synchronizer

The job of the synchronizer is to retime the signal to make it synchronous to the new domain and mask the effects of metastability. The first flip-flop can fall into metastability if its input comes from a different domain. The function of the second flip-flop is to make sure that metastable levels are not propagated into the logic beyond. The assumption here is that metastability will resolve within one clock cycle. If this assumption does not hold, then the two-flop synchronizer fails. The input to the first flip-flop (R_0 in the figure above) must come directly from a flip-flop in the sender domain without having any combinational logic in between, otherwise glitches will significantly increase the failure probability of the synchronizer [7].

This failure probability of the synchronizer is usually characterized using the Mean Time Before Failure (MTBF). MTBF is given by [8], [10], [39]:

$$\frac{e^{\frac{t_r}{\tau}}}{f_d f_{clk} T_o} \quad (2.1)$$

where :

t_r: resolution time (including flip-flop clock to output time)

τ: resolution time constant

f_d: the average frequency of data transitions

f_{clk}: clock frequency

T_o: the asymptotic width of the metastability window with no resolution time. τ is a circuit-dependent parameter. It reflects the circuit ability to resolve intermediate voltage levels [9]. T_o is a mathematical parameter that can be obtained by circuit simulations and has no practical meaning. It is used to derive the metastability window T_w (the period of time in which input data transitions can not be resolved within a given resolution time t_r). It is related to T_w by [9]:

$$T_w = T_o e^{-t_r/\tau} \quad (2.2)$$

2.2 CDC Design Styles

Synchronizers are an essential components in most CDC interfaces. A question that might rise here is that why it is not sufficient to use synchronizers only to pass data from one clock domain to another? Why have the CDC interface at all? The answer is that it is fine to use just a synchronizer if we are passing a single control bit from one domain to the other. However, most CDC transfers are usually data transfers or multi-bit control signals. Attempts to synchronize the data usually lead to catastrophic results even if all data lines toggle simultaneously since some bits will take one cycle more than others leading to data loss or inconsistent control states [3].

To overcome these problems, the topic of CDC interface design has received much attention from researchers. Different CDC designs have been proposed with the general aim of having a CDC interface that has high reliability, low latency and high throughput. Based on the hardware architecture, there are three main strategies for implementing interfaces in GALS systems [11]:

- Pausable clocking
- Boundary synchronization based CDC interfaces
- FIFO based CDC interfaces

The last two interfaces apply brute force synchronization using synchronizer circuits and surround the synchronizer with a suitable protocol to achieve correct synchronization. Pausable clocking does not use synchronizers. Instead, it uses a different mechanism which will be elaborated on next. The details of these three interfaces are discussed in the following sections. Special focus is given to pausable clocking interface because our design is closely related to this interface.

2.2.1 Pausable Clocking

Pausable clocking is a GALS interfacing technique that does not try to tackle the problem on the data lines, but instead shifts the local clock to avoid having timing violation [12]. The idea is to avoid synchronization altogether. Instead, this technique stops the clocks of the locally synchronous blocks when there is a data transfer and releases them when the transfer is over. One of the earliest works on this technique was proposed by Chaprio in [13]. However some of his assumptions are invalid for today's designs [5]. One of the first practical circuits for pausable clocking is proposed in [14] and further improved in [5]. An example circuit for pausable clocking is shown in Figure 2.2.

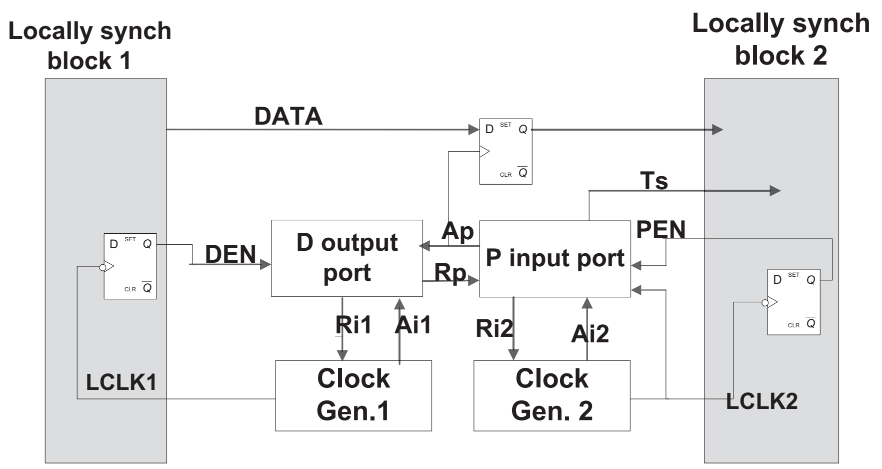

Figure 2.2: Pausable clocking (reproduced from [5])

As shown in the figure, the circuit uses controllable clock generation units usually implemented as ring oscillators with mutual exclusion (ME) elements. When a request is made to the clock generation unit for clock pausing, the next positive edge of the clock gets postponed until the request is de-asserted. The D and P ports are used to perform the asynchronous handshake between the two blocks. They are implemented as asynchronous state machines. The protocol followed by the interface is described in [5] and can be summarized as follows:

1. When the sender puts data on the data line, it enables the D port through a transition on DEN.
2. The D port sends a clock pausing request to the clock generation unit through R_{i1}, consequently $LCLK_1$ in Figure 2.2 pauses. Then, the D port receives the A_{i1} signal acknowledging the pause of $LCLK_1$, consequently the R_p signal

is asserted, which informs the receiver that the sender has requested a data transfer.

3. If the receiver can accept data (a transition on PEN indicates whether the receiver is ready to accept data or not), the P port raises the R_{i2} signal, which stops the receiver clock ($LCLK_2$).

4. Upon receiving A_{i2} assertion, the P port asserts A_p. This positive edge of A_p is used to latch the data in the data path while the clock is still stopped.

5. Once the data is latched, the protocol terminates with the release of the receiver clock followed by the release of the sender clock.

Due to the fact that this interface employs clock stretching, it has several features that makes it an attractive option. These features can be summarized as follows:

1. There is no fear of metastability, thus it avoids latency due to two flip-flop synchronizers. Latency is defined as the time taken from the moment the sender puts a data item on the data bus till it is received by the other side.

2. Clock frequency ratios between communicating modules can be arbitrary.

3. Two communicating modules may also have arbitrary phase discrepancy.

The ISL Laboratory at the Swiss Federal Institute of Technology have implemented several chips that use pausable clocking [18]. Otherwise, this technique did not have much success in commercial circuit design. In summary, the following problems with pausable clocking hampered its adaptation into chip design:

1. Ring oscillators are not practical for use in commercial circuit production. They are very sensitive to process, voltage, and temperature variations [11].

2. When the clock is paused, sometimes it is difficult to stop the registers of the whole system at once. Hence, the requirement of pausing clocks may lead to clock edge discrepancy between the clock source and the registers, which may lead to malfunctioning of the system due to inability of instantaneous pausing of the clock at terminal registers. This phenomenon is known as the clock over-run issue [15].

3. Restarting the clock leads to timing mismatches, known as jitter. Indeed, after restarting a clock that was completely stopped, the dynamics of clock generating circuits make them prone to period by period duration mismatches [16].

4. The pausing of the clock halts the complete system. Hence no activity takes place during the communication phase, and the system loses performance. A pitfall is to believe that energy consumption can be advantageously reduced thanks to clock pausing, however, if the clock is stopped, the system will take longer to finish its job.

5. Stretching the clock often would make the effective clock frequency determined not by the preset clock generator frequency but by communication with other synchronous modules. This is not desirable, especially in systems where the frequencies need to meet performance and power requirements [17].

   In chapter 3, we will show how our proposed design can be used to overcome these shortcomings.

2.2.2 Boundary Synchronization Based CDC Interfaces

Boundary synchronization based CDC interface is the most general and most common interface for applications that do not require a high throughput. This CDC interface consists of two two-flop synchronizers (one at each end). These synchronizers synchronize control signals only. As mentioned earlier, synchronizing data usually leads to catastrophic results. This type of interfaces is discussed in several references such as [6], [19]. The interface will be explained in terms of the four-way handshake protocol but two-way handshaking can also be employed. An example circuit for this interface is shown in Figure 2.3.

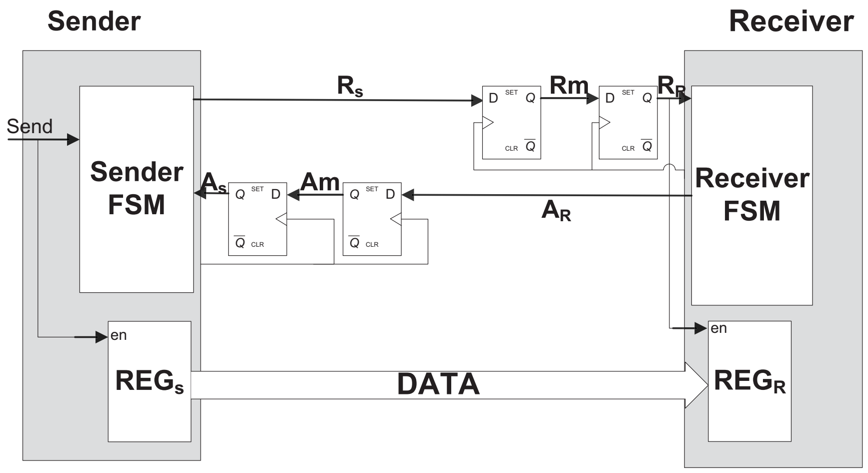

Figure 2.3: Simple boundary CDC interface

The protocol followed by the circuit is described as follows:

1. The sender asserts the R_S signal requesting a data transfer. R_S gets synchronized into the receiving domain using the two-flop synchronizer raising R_R.

2. The rise in R_R enables the receiver register (REG_R in the figure) and the following clock edge latches the data. In parallel, R_R is sampled by the receiver's finite state machine (FSM). The FSM sends an acknowledgement (raises A_R) to the sender.

3. A_R is synchronized by the sender synchronizer causing A_S to rise.

4. The sender's FSM drives R_S down in response to A_S. R_S is synchronized into the receiver de-asserting R_R.

5. The receiver's FSM de-asserts A_R in response to R_R. The negative edge of A_R is synchronized into the sender bringing down A_S. Only when A_S gets de-asserted, is the sender allowed to make a new request.

This technique is simple and low cost. It has a latency of 2-3 receiver clock cycles depending on the phase relationship between the sender and the receiver. However, its main shortcoming is that it has a low throughput. If the receiver processes the request immediately, then the sender can issues a new request after 8-12 clock cycles (4-6 sender cycles + 4-6 receiver cycles) for four-way handshake. For two-way handshake, this number can be reduced to 4-6 clock cycles (2-3 sender + 2-3 receiver) at the expense of increased complexity. All four synchronizations done in this protocol are subject to metastability if the first flip-flop does not settle into 0 or 1 within one cycle. One solution would be to add a third flip-flop to increase reliability but this will further increase the latency and further reduce the already low throughput. To provide a high throughput CDC interface, FIFO based CDC interfaces are used. These interfaces are discussed next.

2.2.3 FIFO Based CDC Interfaces

FIFO-based CDC interfaces [40], [20], [21] are the most widely used CDC interfaces for applications that require high throughput data transfers. One of the prominent FIFO based CDC interfaces is described in [20]. This interface uses a dual-clock circular queue to transfer data across clock domains. The sender places data at the end of the queue and the receiver retrieves them from the front. Hence, the queue has two interfaces; a put interface controlled by the sender clock that takes data from the sender and a get interface controlled by the receiver clock and supplies data to the receiver. This CDC is shown in Figure 2.4.

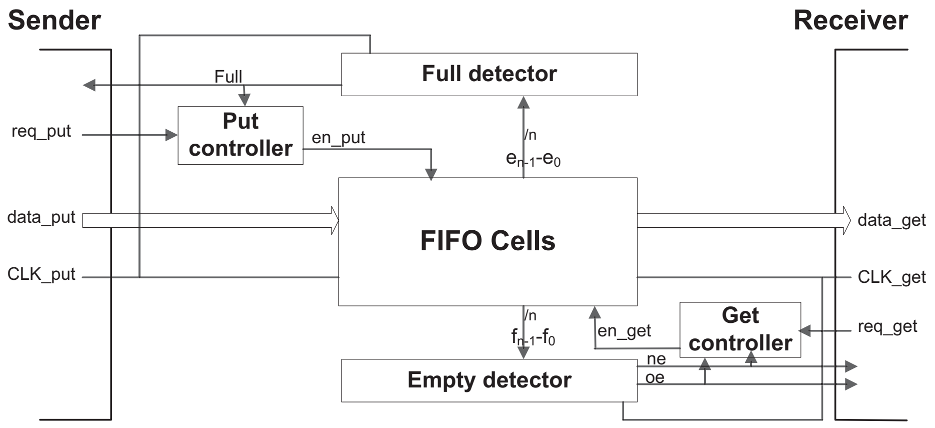

Figure 2.4: FIFO based CDC interface

As shown in the figure, the FIFO consists of the FIFO cells, two port controllers (one for the get interface, and one for the put interface), and two FIFO state detectors (empty detector and full detector).

The operation of the interface is described in details in [20] and can be illustrated as follows:

- When the sender wants to send data, it is put into *data_put* bus and the sender makes a request through *req_put*.
- If the queue is not full, the put controller will allow the request to proceed by asserting the *en_put* signal which enables the queue cells for writing operations.
- The put interface has a put token. The use of tokens allows the data to remain in its place when it is enqueued. The token acts as a pointer to the cell at the end of the queue. Once the data item is enqueued, the token moves to the next cell.
- The get interface has a get token indicating the cell at the start of the queue. This interface operates in a similar manner to the get interface. Requests

made by the receiver are processed only if the empty detector indicates that the queue is not empty. In this case data is provided to the receiver through the *data_get* bus.

This mechanism allows this CDC interface to decouple the reading and writing operations. Each operation is controlled by its respective clock. The signals e_n and f_n provide single cell state information on whether each cell is empty/full to the empty and full detectors. The tricky part of the design is the empty and full detectors since the overall state of the queue depends on both get and put operations which are controlled by two different clocks. Therefore, these signals are asynchronous to the sender and receiver. Synchronizers, typically two-flop synchronizers, are used to synchronize empty and full signals before they get outputted by the detectors. However, these synchronizers add latency to the circuit which means that there could be a delay in reading these two signals by the sender and receiver. Three problem might arise from this situation:

- Writing to a full queue: This problem is caused by the delay in generating the full state signal attributed to the use of the two-flop synchronizer. This problem is solved by changing the definition of the full signal to be "the FIFO is considered full when there are either 0 or 1 empty cells left" [20].
- Reading from an empty queue: This problem is caused by the delay in generating the empty state signal attributed to the use of the two-flop synchronizer. This problem is solved by changing the definition of the empty signal to be "the FIFO is considered empty when there are either 0 or 1 cells filled" [20]. The signal 'ne' in figure propagates this information to the port controller.
- Deadlock: Caused by the new empty definition. If the queue contains one data item, it might never be read if the 'ne' signal is used alone. Therefore, the signal 'oe' which is true only if the queue is really empty is used to discover this state and allows the receiver to retrieve this item.

The main advantage of the FIFO interface is that it has a high throughput. It has a maximum throughput equal to one transfer every clock cycle of the slower domain. The FIFO can work at this rate in bulk transfers if it doesn't become empty or full. This interface doesn't pause the clock allowing the sender and receiver to continue doing useful work which allows them to finish other tasks faster.

However, FIFO interfaces have some drawbacks which can be summarized as follows:

- It has a relatively high latency of 3-4 cycles. Latency is the time taken from the moment the sender puts a data item on the data bus until it is received by the other side in an empty queue. This introduced latency might be significant and unacceptable for high-speed applications [11].
- The FIFO CDC is not very flexible in terms of clock frequency ratios. Large differences in clock frequencies cause the FIFO to fail. For example, if the receiver's clock frequency is more than three times the sender's frequency, the receiver might read empty cells [20].
- Area requirement is relatively high and increases with FIFO size.
- Compared to pasusible clocking, the FIFO uses synchronizers which might propagate metastability into the design.
- Throughput is significantly decreased if there is a large mismatch in the communication rate and the FIFO constantly hits the empty and full states incurring latency penalties [20].

2.3 Verification of CDC Interfaces

CDC interfaces are not trivial to design, it is possible to make simple mistakes that will turn a safe CDC interface into a buggy one. Therefore, verifying CDC interfaces is an important step in their design cycle. Verification methods are divided into funtional verification (simulation) and formal verification. RTL simulation, the most common method used by designers to check their designs, typically can not find CDC errors. Digital simulation programs usually generate X's when they recognize setup and hold violations on CDC signals. This can frequently cause gate-level simulations to fail [7]. Moreover, RTL simulation can not model the effects of metastability [1] discussed earlier. Therefore, formal verification based methods are needed to check these interfaces.

Formal verification has recently become an important method to check complex systems. Formal verification uses methods based on mathematics to check the correctness of systems. Its use has been increasing in industrial applications as available tools improve in performance.

Formal verification has recently been applied to verify CDC interfaces. The next section discusses the most important works in this field.

2.4 Existing CDC Verification Methods

Several methodologies have been presented for the formal verification of CDC interfaces. One of the most important works on CDC verification is presented in [23], [42]. In their work, the authors discuss two methods for verifying CDC interfaces. In the first, verification rules are generated as PSL properties from the Signal Transition Graph (STG) representing the protocol specification and are applied to RuleBase model checker. These rules verify that STG events occur in order and that each event happens only in the appropriate states. The second method checks the correctness of data transfers and checks missing or duplicated data. The authors, however explicitly mention that the probability of metastability not resolving in one cycle is ignored in their approach.

Another interesting work is presented in [24]. In this work, the authors use SAT based bounded model checking to verify CDC protocols. The focus is on modeling multiple clocks in the bounded model checker. They assume that setup and hold times are zero. Different clocks are modeled by assigning a state variable for each clock which can be either 0 or 1 at each verification tick.

In [25], the authors use SAL model checker to proof the correctness of a simple CDC interfacing circuit, namely the boundary based CDC interface discussed earlier. The interface is modeled as three interacting processes and a proof is generated to check that the circuit satisfies a basic invariant.

In [26], [41], a methodology for verifying multiple clock designs in SMV model checker is presented. The work focuses on dealing with the zero-delay abstraction performed by formal verification tools. The output of each gate is set to a random value for a single verification step for all gates or components along the critical path between two domains. The authors acknowledge that this method might lead to under- or over-approximations if there is more than one critical path in the interface; however, they assume that these situations do not arise. In reality, hardware paths have significant variations in delay leading to situations that are totally different from the cases predicted by the approach.

2.5 Summary

In this chapter, the concepts of GALS and synchronizers and their relation to CDC interfaces were presented. The operation and characteristics of the two-flop synchronizer were discussed and the concept of MTBF was explained. Related work in CDC design was reviewed especially three common CDC interfaces: pausable clocking, FIFO based, and boundary based CDC interfaces. Finally, the verification of CDC interfaces was briefly addressed and the most important existing approaches for verifying these interfaces were explained.

Chapter 3

The Proposed Hybrid CDC Interface

In this chapter, our new CDC interface; the hybrid FIFO-asynchronous CDC interface is presented. The chapter starts by giving a brief introduction about the proposed design in Section 3.1. Section 3.2 discusses the different components and their functions. Section 3.3 describes the protocol accompanying our CDC interface and highlights its differences from the conventional pausable protocol. In Section 3.4, the implementation details of the important design blocks are discussed. Emphasis is given to the novel protocol-pauser block. Section 3.5 summarized the chapter.

3.1 Design Overview

As discussed in Chapter 2, pausable clocking is one of the most important GALS interfacing techniques, however, it suffers from several problems. To improve on this technique, we need to avoid the performance penalty associated with pausable clocking and avoid ring oscillators. In this chapter, we present a hybrid FIFO-asynchronous method for constructing robust CDC interfaces. This method is built upon the traditional pausable clocking interface and extends it with several new components. A complete design is proposed and fully implemented using 90nm TSMC CMOS technology.

The proposed interfacing methodology uses the bundled data protocol and a customized FIFO. The use of FIFOs avoids performance penalty due to pausing and clock over-run issues as it decouples the locally synchronous modules from clock domain crossing interfaces. Moreover, bundled data protocol is used in a pseudo-deterministic way, similar to pausable clocking interfaces, hence, making the interface highly robust to metastability. Unlike conventional pausable clocking, this interface avoids completely pausing the locally synchronous modules. Unlike FIFO based interfaces, this interface works with any frequency ratio between the two modules. To achieve these features, the proposed solution leverages Signal Transition Graphs (STGs) implemented at the circuit level. SPICE circuit simulation results confirm the operation and robustness of the design at maximum workloads, and arbitrary frequency ratios, over a temperature range of -50 to 50 degrees Celsius.

3.2 The Proposed Design

In this section, the operation of the proposed design and its critical blocks are briefly described. This protocol is implemented in hardware as shown in Figure 3.1.

The middle blocks in Figure 3.1, the D and P ports are asynchronous machines, borrowed from the conventional pausable clocking methodology presented in [5]. In addition to these ports, the interface circuit contains four additional blocks: a synchronous FIFO, two special circuits that are called protocol-pausers, one at each end, and an A_i generator block.

Broadly, transfer requests from the sender block accumulate at the FIFO. This synchronous FIFO blocks the sender from sending more data if it sees the possibility of an overflow. The D and P ports are responsible for performing the asynchronous handshake between the two domains. These two blocks are controlled

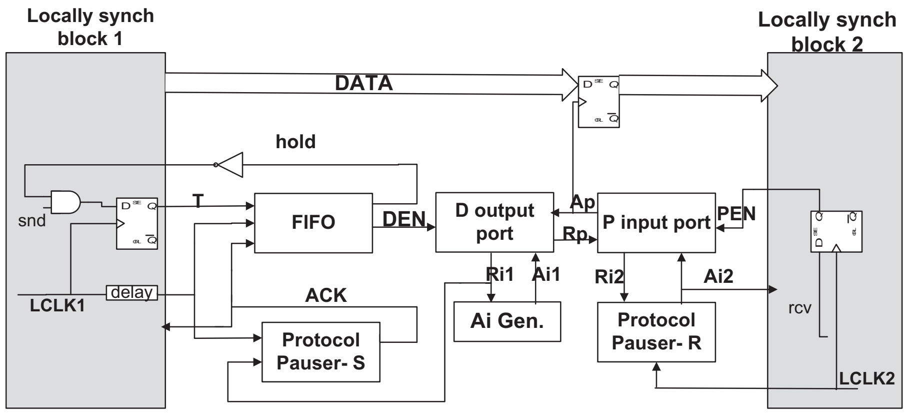

Figure 3.1: Proposed hybrid FIFO-asynchronous CDC interface

by synchronous logic. The two protocol-pausers form the interface between synchronous and asynchronous logic. These two blocks pause the control signals and release them when they cannot cause timing violations at the sender or receiver, which is the most challenging task in this design. We will first discuss the transfer protocol followed by the hardware implementation of the various blocks.

3.3 The Proposed Protocol

Our proposed design is based on the bundled-data asynchronous handshaking protocol. To achieve a level of determinism (and to avoid conventional synchronizers), a few modifications are done in the protocol. These modifications lead to a unique signal sequence described in the following:

1. When the sender requests to send data, the FIFO toggles the DEN signal activating the D port in Figure 3.1.

2. The D port raises the R_{i1} signal, which in turn generates the A_{i1} signal using A_{i1} generator. The D port, upon receiving the A_{i1} signal, sends a request to send data to the receiver using the R_p signal.

3. If the receiver can accept data (a transition on PEN indicates whether the receiver is ready to accept data or not), the P port raises the R_{i2} signal. The protocol-pauser-R holds the processing of the request until the next positive edge of the receiver clock. With the assertion of $LCLK2$, A_{i2} is also raised.

4. Upon receiving A_{i2}, the P port asserts A_p. This positive edge of A_p is used to latch the data into the data flip-flops with a small deterministic delay with respect to the receiver clock. Therefore, data becomes available to the receiver side without violating its timing constraints.

5. Following the RTZ signaling, the handshake signals R_p and A_p are negated. The protocol-pauser-S generates a sender-safe ACK signal once the R_{i1} signal is negated.

An obvious benefit in the proposed protocol as compared to the conventional pausable clocking protocol is that it does not require pausing of the clocks. This has been made possible through the use of protocol pausers, which pause the transfers managed by the protocol rather than pausing the locally synchronous blocks. This allows the blocks to continue their normal operation thus, improving the performance of the system. Comparison of the protocols shows that in the conventional protocol, the clock is paused as soon as R_i is asserted (steps 2, 3 in pausable protocol description in Section 2.2.1). Whereas the proposed protocol halts the operation of the interface (step 3 and 5 above). This is achieved by replacing the controllable clock generation units (Clock Gen. 1 and Clock Gen. 2 in Figure 2.2) with protocol-pausers. In addition, data latching is performed while clocks are stopped in the conventional method. In the proposed design, it is performed while the clocks are running but the protocol-pausers ensure that there is no timing violation using their novel circuit design, which is further elaborated in the next sections. The signal transition graph representing the transfer protocol is shown in Figure 3.2. The superscript ^{T} is used to indicate a signal transition, whether positive or negative.

The bullets (tokens) indicate the initial state of the system. The implementation of the blocks that produce this STG is discussed next.

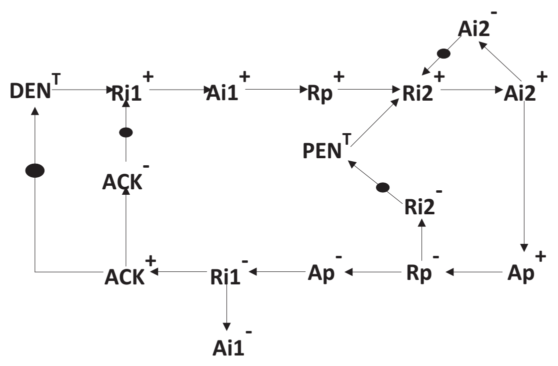

Figure 3.2: STG of the proposed design

3.4 The Implementation

As mentioned earlier, the proposed protocol consists of the following six blocks:

- The synchronous custom FIFO
- The Protocol-Pauser-S
- The Protocol-Pauser-R
- The asynchronous D output port
- The asynchronous P input port
- The A_i generator

The last block (The A_i generator) is just a buffer used to supply the A_i signal to meet the D port requirements. The first three blocks have been custom designed at the transistor level while the two ports were borrowed from pasuable clocking methodology in [5]. These blocks will be explained in the following sections.

3.4.1 The FIFO

The main goal of this FIFO is to decouple the terminating modules from the pseudo pausable interface. The FIFO is also responsible for flow control. The associated controlling state machine of the FIFO keeps checking the overall state of the system. Whenever the data input rate becomes faster than the data consumption rate, this state machine asserts the *hold* signal, which tells the sender module to wait until the FIFO empties some space.

The Mealy state machine for a FIFO queue of size two is shown in Figure 3.3. More stages can be added in the FIFO with little modifications to the state machine. The state machine has two inputs which are the T(transfer) and ACK(acknowledgement) signals. The state machine operates on a delayed version of the sender clock to allow transfer requests from the T input to be latched during the same clock cycle in which they are produced by the sender. Upon receiving a request at its T input, the FIFO issues the request by performing a transition on its DEN output which tells the D port to start a handshake cycle (The D port's DEN input employs transition signalling and any change in DEN is translated into a handshake request). The DEN signal is kept at the same level until the FIFO receives an ACK signal. If a new request arrives at the FIFO's T input during the handshake period, it is accumulated in the queue and the DEN signal is kept stable until an ACK signal arrives. The protocol-pauser-S block, which is discussed next, makes sure that ACK signal does not violate the timing constraints of the FIFO.

3.4.2 The Protocol-Pauser-S

The protocol pausers form the interface between synchronous and asynchronous logic. As mentioned in the previous section, the protocol-pauser-S is responsible for generating the ACK signal so that it is synchronous to the FIFO. The inputs of this block are the R_{i1} signal and the clock. The R_{i1} signal is generated by

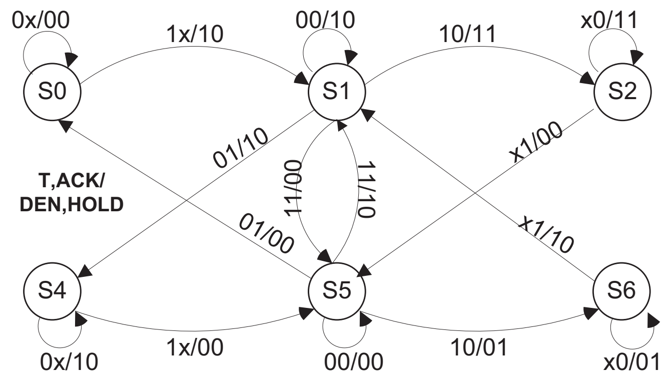

Figure 3.3: FSM of the FIFO

the asynchronous D port. According to the protocol, positive transition in R_{i1} is generated by the D port in response to a transition on DEN (synchronous to the sender). Therefore, it is deterministic with respect to the sender clock, while negative transitions on R_{i1} are generated in response to a negative transition in the A_p signal, which is non-deterministic with respect to the sender clock. Hence, only one of the transitions in R_{i1} requires phase correction, which significantly simplifies the design. The circuit diagram for the Protocol-Pauser-S is shown in Figure 3.4.

The circuit consists of a combination of two Muller C-elements both having R_{i1} as one input. The other inputs to the two C-elements are the delayed version of the sender clock and its complement, as shown in Figure 3.4. The outputs of the C-elements are connected to a NOR gate. The combination of these 3 gates acts as a phase corrector for one transition of the R_{i1} signal.

This correction is illustrated as follows: The 3-gate combination immediately propagates the deterministic positive transition of the R_{i1} signal to node X. For the negative transition, this circuit blocks it till it is safe unless it causes a timing violation in both C-elements concurrently. Timing window of such an occurrence is so small that it is filtered out using the NOR-gate delay. We demonstrated this metastability filtering behavior with our simulations in Section 4.5. As a second level precaution, a worst case delay requirement can be obtained using corner-analysis and

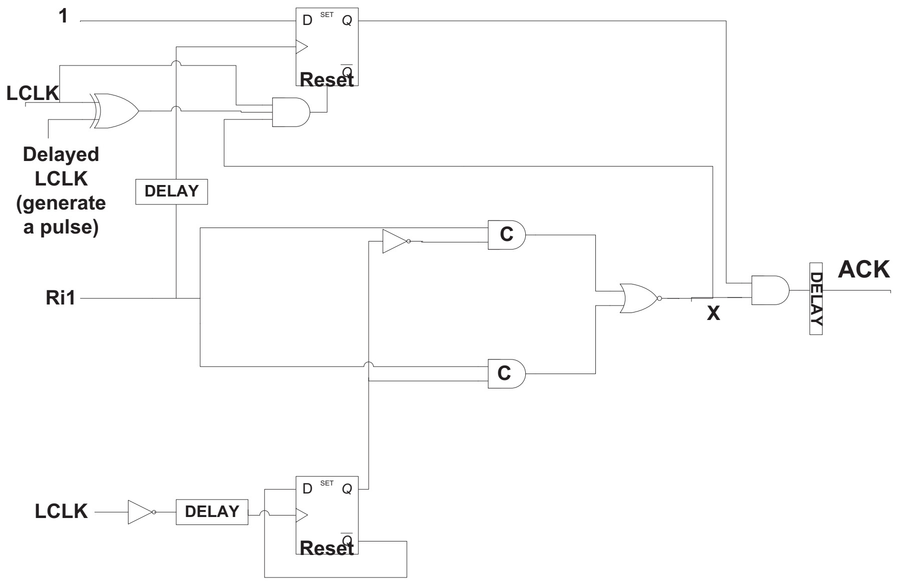

Figure 3.4: The protocol-pauser-S circuit diagram

this can be introduced at the output path as shown in Figure 3.4. Therefore, the resulting ACK signal will not violate the sender's timing constraints.

The timing behavior of the protocol-pauser-S is shown in Figure 3.5. This figure illustrates the approximate timing instances of major events in the protocol with respect to the sender clock. The first event in the figure is the reset in the upper DFF. This occurs at the beginning of the sender clock cycle if R_{i1} is at logic '0'. In this case, the NOR gate output at node X is logic '1'. Therefore, the positive clock edge creates a reset pulse at the top flip-flop in Figure 3.4. This brings the ACK signal to logic '0' (if it is initially high).

When the R_{i1} signal rises, node X becomes logic '0'. R_{i1} also acts as a clock for the upper flip-flop in Figure 3.4, latching a '1' into the upper flip-flop. The delay on the R_{i1} line (connected to the upper flip-flop) makes sure that these two events occur in order. Hence, the output remains stable at logic 0.

As long as R_{i1} remains high, node X will remain low and ACK will remain low.

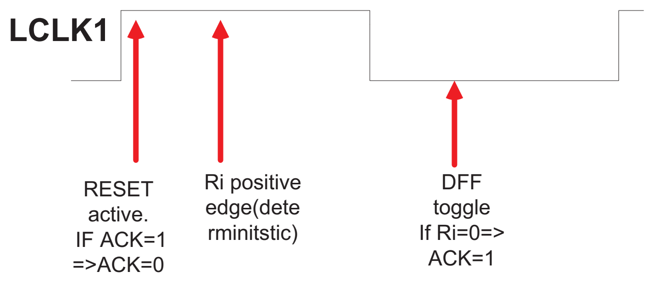
LCLK1
RESET active. IF ACK=1 =>ACK=0
Ri positive edge(dete rminitstic)
DFF toggle If Ri=0=> ACK=1

Figure 3.5: The timing behavior of the protocol-pauser-S

When the negative transition of R_{i1} occurs, node X continues to remain low since at least one of the outputs of C-elements remains high until the lower DFF toggles. The first toggle in the lower DFF that occurs after the negative transition of R_{i1} generates the ACK signal. Hence positive transitions in ACK become deterministic with respect to the sender clock.

The input to the lower DFF's clock is a delayed inverted version of the clock. This delayed version of the clock determines the position of the leftmost arrow in Figure 3.5. It determines the portion of the clock cycle where negative transition of R_{i1} is detected within the same clock cycle. Provided the negative transition of R_{i1} occurs after the positive transition in the inverted delayed version of the sender clock, the transition in R_{i1} is deferred to the next clock cycle. If R_{i1} and the lower DFF's toggle occur approximately at the same time, the simulation results in Chapter 4 demonstrate that the three gate combination (the 2 parallel C-elements followed by NOR) with the delay turn out to be an effective metastability filter. The design works if the timing separations among the three events in Figure 3.5 are considered properly.

3.4.3 The Protocol-Pauser-R

The protocol-pauser-R operates in a similar manner as protocol-pauser-S. It generates the A_{i2} signal such that it does not violate the receiver's timing constraints. For the receiver side, the non-deterministic transition is the positive transition of the A_{i2} signal, which is generated by the asynchronous P port. Whereas negative transitions of R_{i2} follow a deterministic sequence of events with respect to the receiver's clock. Therefore, only the phase of the positive transition of R_{i2} needs to be corrected. The receiver pauser's internal structure is very similar to the sender pauser in Figure 3.4 with few modifications:

1. NOR gate, after the C-element in Figure 3.4, is replaced with an AND gate to correct the phase of positive transitions in the R_{i2} signal instead of negative transitions.

2. The Reset signal for the upper DFF is generated from the A_{i2} signal itself.

3. The final AND gate generating the ACK is expanded to include three inputs, with the third being the receiver's clock directly supplied. This modification makes the rise of A_p receiver-clock dependent hence allowing the protocol to absorb any frequency variations.

3.4.4 The D Port and P Port

The D and P ports are asynchronous finite state machines used to perform the asynchronous handshake between the two ports. As mentioned earlier, they are borrowed from pausable clocking methodology [5]. Port enable signals (DEN and PEN) use transition signalling while the outputs (R_p, A_p, and R_i) use level signalling. The sequence of events followed by the ports are described in the protocol description in Section 3.3 and by the STG in Figure 3.2. The ports are implemented based on the asynchronous FSMs [5] shown in Figure 3.6.

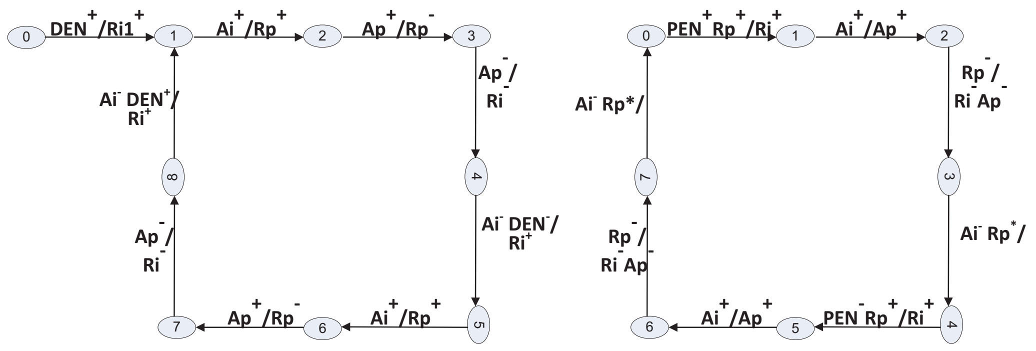

Figure 3.6: Asynchronous FSM of the ports (Left: D output port, Right: P input port)

3.5 Summary

In this chapter, our proposed hybrid FIFO-asynchronous CDC interface that does not pause the clocks of the communicating systems and that requires no external synchronizers was presented. The overall design and the transfer protocol was explained. The implementation of the various blocks was explained. The blocks were implemented in 90nm TSMC CMOS technology. Special attention was given to the protocol-pauser block which is used as an interface between synchronous and asynchronous logic.

Chapter 4

Simulation Results

In this chapter, SPICE circuit simulation results for the design proposed in Chapter 3 are presented. Section 4.1 shows proof of concept simulations for the proposed design. Section 4.2 shows the behavior of the design under temperature and workload variations while Section 4.3 shows its behavior under frequency variations. Section 4.4 elaborates on the maximum throughput the design can achieve. Section 4.5 addresses the issue of the design's robustness to metastability. In Section 4.6, the results are compared with common CDC interfaces discussed in Section 2.2. Finally, Section 4.7 summarizes the chapter.

4.1 General Simulations

The proposed design was fully implemented using the TSMC 90nm CMOS technology. All the sub-blocks and blocks were individually simulated, then the whole design was simulated using SPICE. Figure 4.1 shows a proof of concept simulation of our implementation. In this experiment, we assumed that the sender requests a transfer in each cycle, i.e. the worst case maximum workload. The frequencies for sender and receiver are different and the phase shift is kept arbitrary.

The figure shows that a request started by DEN triggers the first half of the

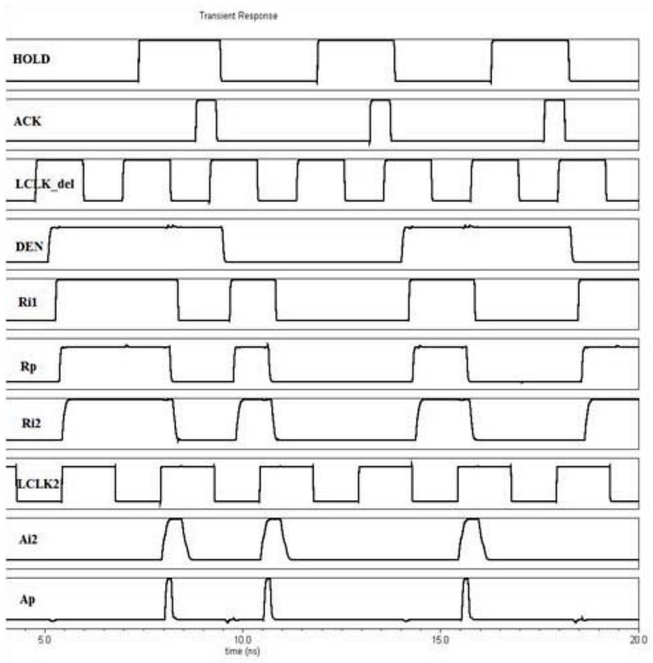

Figure 4.1: SPICE simulation

handshake cycle by enabling the D port. The D port then drives R_{i1} high. After receiving A_{i1} from the A_{i1} generator, the DEN port drives R_p high. The rise in R_p causes eventually R_{i2} to rise. The second half starts when A_{i2} gets asserted triggering an asynchronous sequence eventually bringing down R_{i1} and finally generating the ACK pulse. Most importantly, the figure also shows that if the R_{i2} signal asserts closer to the receiver clock ($LCLK2$) edge (i.e. after the toggle of the lower DFF as mentioned in section 3.4) then the A_{i2} signal asserts to logic '1' after one clock cycle. The second positive transition of R_{i2} in the figure shows that this transition occurs before the toggle of the DFF, consequently asserting A_{i2} in the same cycle. The same applies to the ACK signal which is generated at a suitable time in reference to $LCK_{delayed}$ signal no matter when R_{i1} gets de-asserted. The figure also shows that the *hold* signal asserts after two sender clock cycles to prevent the sender from making more requests. The hold signal negates as soon as an ACK is received.

A second more comprehensive simulation showing all important signals in the design is shown in Figure 4.2. The frequencies for sender and receiver in this simulation are 620 MHz and 1.35 GHz respectively and the initial phase shift is 1.4 ns. The order of signals in the figure correspond to the order of the STG (the first few transitions). This figure also shows how the T signal gets latched in the same clock cycle in which it is generated because the FIFO runs on a skewed version of the clock.

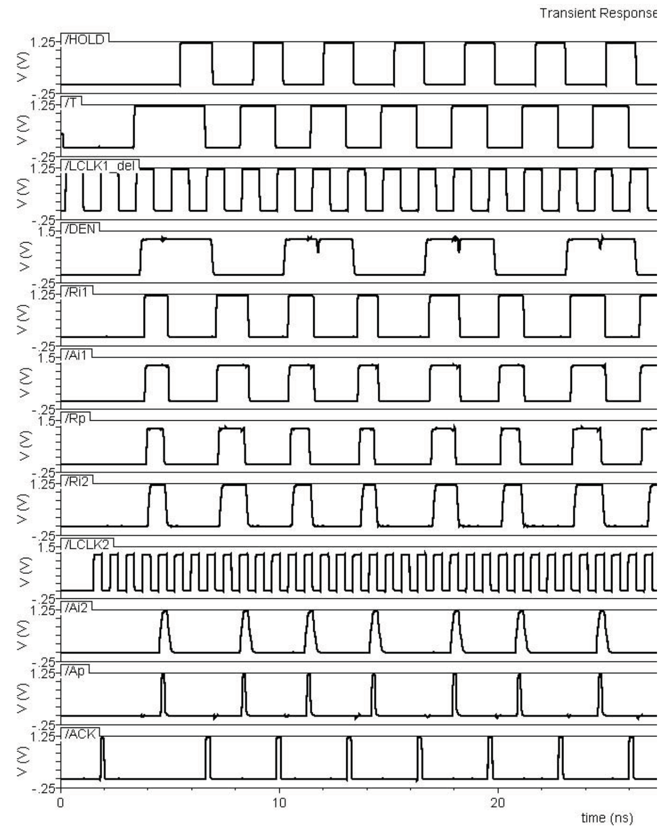

Figure 4.2: SPICE simulation 2

Extensive simulation results show that the design can operate robustly for several extreme variations, such as different workloads, different frequency ratios, and extreme temperature settings. We will elaborate more on these experiments in the following sections.

4.2 Temperature and Workload Variations

The robustness of the design was tested against temperature variations. The temperature was varied from from -50^OC to 50^OC. The design continued to function robustly in all temperatures even at maximum workload. The design's throughput degraded as temperature increased. The maximum throughput was observed at -50^OC and the minimum was at 50^OC. The power consumption of the design increased with temperature increase, however that increase was below 5% for a temperature increase of 100^OC. Table 4.1 shows the results of these simulations. The frequencies of the sender and receiver in this experiment are 1.51 and 2 GHz respectively.

Table 4.1: Temperature and workload variations

	Max. workload			Once in 5 sender cycles	Once in 10 sender cycles
	-50C	25C	50C		
Power (mW)	38.77	39.98	40.51	35.33	32.64
Throughput (M Data items/s)	468	426	404	298	149

The workload was also varied between maximum workload and 1 request every 10 cycles. It was observed that for this set of frequencies (sender=1.5 GHz, receiver=2 GHz) the sender and receiver would work with almost no overhead for workloads of 1 transfer every 4 cycles or lower.

4.3 Frequency Variations

Various frequency ratios between the terminating modules were also used to test the proposed CDC interface. Both integer and non-integer ratios for relative frequencies were used for both slow-to-fast and fast-to-slow cases and with random phase shifts.

Table 4.2 shows some of the frequencies used along with the obtained throughput and power consumption.

Table 4.2: Frequency variations

Sender freq.	Receiver freq.	Throughput (M Data items/s)	Power (mW)
1.2 GHz	1 GHz	400	31.12
1 GHz	1.2 GHz	400	28.39
1.28 GHz	575 MHz	321	31.89
575 MHz	1.28 GHz	301	25.86
500 MHz	700 MHz	318	30.71
700 MHz	500 MHz	289	31.17
333 MHz	1 GHz	331	21.3
1 GHz	333 MHz	325	27.8
1.35 GHz	620 MHz	337	31.46
620 MHz	1.35 GHz	310	25.75

The throughput obtained in these experiments is measured at maximum workload, i.e. when the sender makes a request for communication every clock cycle. The temperature was assumed to be 25 ^oC in these experiments.

4.4 Maximum Throughput

Simulations have shown that the design can achieve a maximum throughput of 606 M items/s. This throughput can be achieved when both the sender and receiver operate at 606 MHz frequency at 25^oC and with a suitable constant phase relationship leading to a successful transfer every clock cycle. In these conditions, the power consumed by the handshake circuitry is 35.6 mW. Figure 4.3 shows the SPICE simulation at maximum throughput.

We achieved operating frequencies of more than 1.5 GHz for the sender and more than 2 GHz for the receiver at 25^OC. Transistor sizing is only optimized to achieve correct functionality. Further optimization may provide even better results.

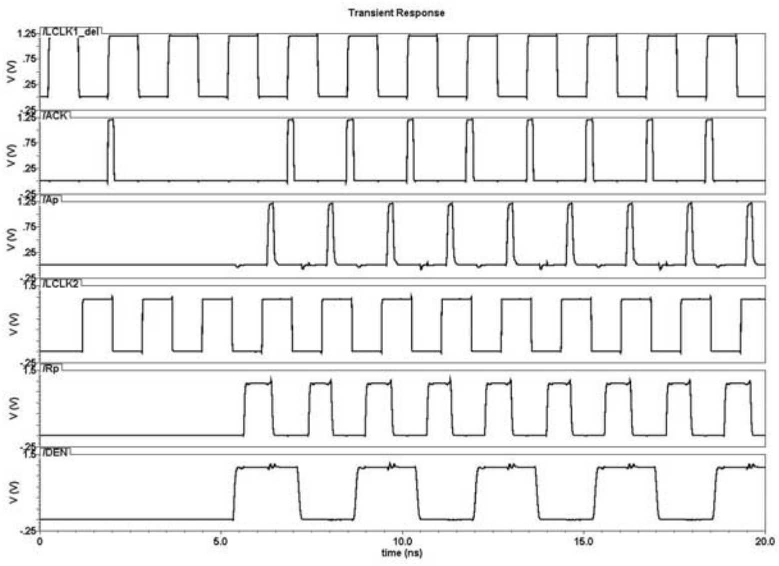

Figure 4.3: Maximum throughput SPICE simulation

4.5 Robustness to Metastability

The proposed design transforms mutually asynchronous signals into pseudo deterministic ones. This enhances the robustness of the design to metastability. This is further illustrated in Figure 4.4. The figure shows parametric analysis of the pauser-S block at 0.01 ps resolution with worst case phase relationship. At one point, the ACK signal is generated at the clock edge immediately after (top waveform). When R_i goes down 0.01 ps later, ACK is paused until the next cycle (middle waveform) without getting into metastability.

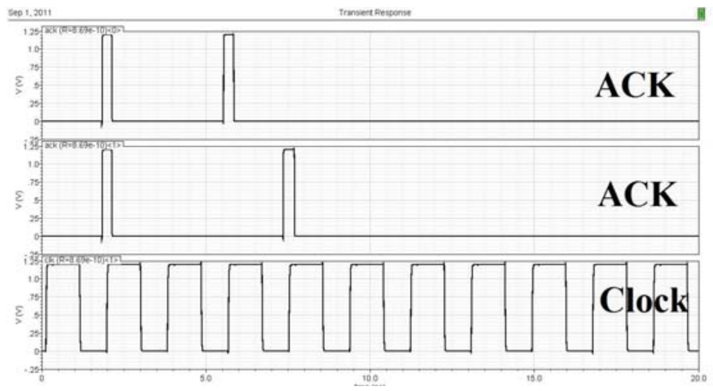

Figure 4.4: Parametric analysis at 0.01ps resolution in SPICE

4.6 Comparison with Other Techniques

Comparing the proposed design with the conventional pausable method [5], the proposed design does not pause the whole clock domain as is the case with pausable clocking technique. The throughput in Tables 4.1 and 4.2 only shows throughput while the two modules are interacting. But the overall throughput of the system considerably increases because the clocks of the sender and receiver continue to run in the proposed design while the sender and receiver are completely stopped in the conventional method. This fact is illustrated in Figure 4.5. The proposed technique also relieves the requirement of using a ring oscillator. This avoids some pausable clocking known problems such as clock over-run and timing mismatches which have been discussed in Chapter 2. This comes at the price of an increase in area and power consumption compared to the original design. However, the interface is only needed when data are transferred between the blocks. If power consumption is very critical for the application, a simple disabling logic can be used to turn off the entire block when data are not being transferred.

The comparison with FIFO based interfaces is summarized in Table 4.3. The FIFO part of the comparison is based on results reported in [20]. The comparison is more complex and depends on the workload and the frequencies of the interacting

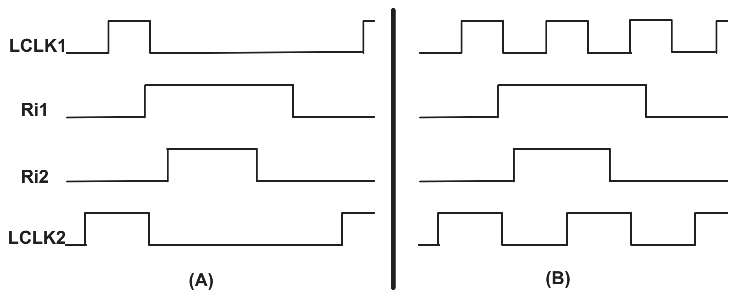

Figure 4.5: Behavior comparison: (A- Clock pausing in conventional method, B- No pausing in proposed method)

domains. For the proposed interface, the protocol pausers pause the protocol twice in each transfer cycle to avoid violating the timing constraints of the synchronous blocks. This pausing makes the design flexible in terms of the frequency discrepancies between the communicating modules. The design functions correctly even if one of the communicating modules runs significantly faster than the other. On the other hand, FIFO based interfaces might fail if the receiver's frequency is more than 3 times the sender's frequency [20]. The cost of avoiding frequency restrictions is a decrease in the throughput of bulk data transfers. The proposed design has a lower throughput than the FIFO design if data is being transferred each clock cycle. However, most of the communicating modules do not send data every clock cycle. A control flow ratio of one transfer every ten clock cycles is considered very pessimistic [22].

Table 4.3: Comparison with FIFO based technique

	Proposed design	FIFO based design
Min. latency (ns)	1.14	$0.5\ T_{snd} + 2.5\ T_{rec}$
Max. latency (ns)	$1.14 + T_{rec}$	$0.5\ T_{snd} + 3\ T_{rec}$
Max. throughput (items/s)	$\frac{1}{1.65ns}$	$\frac{1}{max[T_{rec},T_{snd}]}$
Min. throughput (items/s)	$\frac{1}{1.65ns+T_{rec}+T_{snd}}$	$\frac{1}{max[T_{rec},T_{snd}]}$

A very important performance metric in such designs is latency. Table 4.3

reports latencies as functions of T_{snd} and T_{rec}, where T_{snd} is the clock period of the sender and T_{rec} is the clock period of the receiver. Replacing with values for T_{snd} and T_{rec} shows that for most cases, the proposed design has a lower latency than a FIFO design. For example, when operating at 1 GHz on the send and receive sides ($T_{snd} = T_{rec} = 1$ns), our design decreases the latency of the interface by 39%. Circuit level optimizations can be applied to decrease the latency and increase the throughput further. Our design has a lower latency because it does not require two-flop synchronizers. In summary, the FIFO design can provide higher maximum throughput, while the proposed design provides higher flexibility with different clock speeds, and lower latency.

4.7 Summary

In this chapter, simulation results for the proposed hybrid FIFO-asynchronous CDC interface were presented. Electrical SPICE simulations were performed for various scenarios including maximum workloads, different temperatures, different frequencies, and worst case phase relationships. The design showed high robustness in all performed experiments. The design achieved operating frequencies of more than 1.5 GHz for the sender and more than 2 GHz for the receiver. The maximum through achievable by the design is 606 Mitems/s.

Chapter 5

Verification of CDC Interfaces

As discussed in Secrion 2.3, CDC interfaces cause most RTL simulations to fail. Thus, formal verification, a technique based on mathematical reasoning usually used to complement RTL simulation, can be used for verifying CDC interfaces. In this chapter, we will present a methodology for verifying CDC interfaces based on probabilistic model checking. First, an overview of the proposed methodology is presented in Section 5.1. In Section 5.2, some details about the technique used, namely probabilistic model checking, are given. Section 5.3 provides an overview of the PRISM model checker which is used for verification in the proposed methodology. In Section 5.4, the proposed framework for verifying CDC interfaces is discussed. Section 5.5 presents the first case study, the verification of the boundary synchronization based CDC interfaces, while Section 5.6 presents the second case study, the verification of FIFO based CDC interfaces. The verification of our proposed design is also presented as a third case study in Section 5.7. Finally, Section 5.8 summarizes the chapter.

5.1 Overview of the Proposed Methodology

In this section, our framework for the verification of CDC interfaces is presented. CDC interfaces are subject to metastability failures which are probabilistic in nature and depend on the protocol used as well as on continuous-time circuit level issues as discussed in Section 2.1. Our framework for verifying CDC interfaces takes into account these issues. Compared to other related works discussed in Section 2.4, our approach is the first approach that takes into account the probabilistic behavior of a synchronizer in a CDC interface. Previous work on this topic has either ignored metastability or modeled it as a random one cycle jitter as initially proposed in [27].

The failures of synchronizers are related to circuit-level issues that can not be modeled at the level of abstraction in which model checkers work. However, in order to take them into account, we use probabilistic model checking and explicitly model the failure probability of the synchronizer within the model. It is important to see these failures in the perspective of the whole system. The general structure of the system including the way synchronizers are used, and the type and number of synchronizers in the design all play an important role in determining its characteristics. All these aspects are modeled in the proposed framework. The type of the synchronizer affects some probabilities inside the model. The structure of the protocol affects the states and transitions of the model constructed. Finally, each synchronizer is explicitly replaced by its model therefore the number of synchronizers is explicitly captured.

Before going into the details of the proposed framework, an overview of probabilistic model checking and the PRISM model checker is presented as it is necessary to understand our work.

5.2 Probabilistic Model Checking

Model checking is one of the most prominent formal verification techniques used today. It was first developed in the early 1980s separately by Clarke and Emerson [30] and by Quielle and Sifakis [43]. In model checking, a system is modeled as a set of states and transitions between them that represents how the system behavior evolves from one state to another over time in response to internal and external stimulus. It is based on the construction of a mathematical model of the system to be analyzed. Properties of this system are then expressed formally in temporal logic and automatically analyzed against the constructed model [28].

Probabilistic model checking [31] is a formal verification method that can be applied to systems that exhibit stochastic behavior. Probabilistic model checking inherits the advantages of model checking such as the exhaustive search through the state space of the model and automatic execution from high level models. Moreover, it adds the ability to reason about quantitative properties. In contrast to discrete-event simulation techniques, which generate approximate results by averaging results from a large number of random samples, probabilistic model checking applies numerical computation to yield exact results [34]. Probabilistic model checking has a wide range of applications in fields such as communication protocols, security, biological process modeling, and reliability analysis.

Probabilistic Model checking requires two inputs [34]:

- A description of the system to be analyzed typically given in some high-level modeling language. The model checker then constructs the corresponding probabilistic model.
- A formal specification of quantitative properties of the system that are to be analyzed, usually expressed in variants of temporal logic.

In probabilistic model checking, probabilistic models such as Continuous Time

Markov Chains (CTMC), Discrete Time Markov Chains (DTMC), and Markov Decision processes (MDP) are usually built. In our framework, CDC interfaces were modeled using Markov Decision Processes (MDP) [44], a commonly used formalism for modeling systems that exhibit a combination of probabilistic and nondeterministic behavior. MDP is formally defined as [29]:

Definition: [Markov Decision Process] A Markov decision process (MDP) is a tuple $M = (\mathrm{S}, \overline{s}, \alpha_{\mathrm{M}}, \delta_{\mathrm{M}}, \mathrm{L})$ where:

- S is a finite set of states,
- $\overline{s}$ is an initial state,
- α_{M} is a finite alphabet,
- $\delta_{\mathrm{M}} : \mathrm{S} \times \alpha_{\mathrm{M}} \rightarrow \ Dist\,(S)$ is a (partial) probabilistic transition function, and $Dist\,(S)$ is a convex distribution over S.
- $\mathrm{L} : \mathrm{S} \rightarrow 2^{AP}$ is a labeling function mapping each state to a set of atomic propositions taken from a set AP.

Figure 5.1 shows an example of an MDP [35].

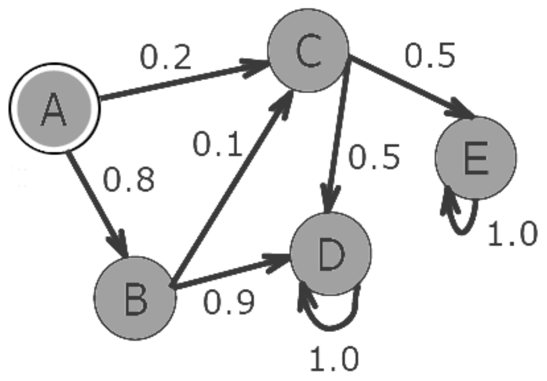

Figure 5.1: Example of an MDP [35]

The properties are specified in quantitative variants of known temporal logics such as PCTL (Probabilistic Computation Tree Logic) , CSL (Continuous Stochastic Logic), and PCTL*. These logics do not just provide a Yes/No answer on whether a property is satisfied by the model, it can also provide quantitative measures on the minimum and maximum probability that a certain property holds. In our framework, the properties were expressed in PCTL [32] (Probabilistic Computational Tree Logic) and PCTL* [33].

The verification of these properties has to be done inside a model checker that supports probabilistic model checking. In this work, the PRISM [36] model checker is used.

5.3 The PRISM Model Checker

PRISM [36] is an open source probabilistic model checker developed at the University of Birmingham and the University of Oxford. PRISM supports many different probabilistic models such as discrete-time and continuous-time Markov chains and Markov Decision Processes (MDP). PRISM has its own high level language to describe these models. This language is described thoroughly in the PRISM manual available in [28]. Models in PRISM are written in the form of state-based modules, each composed by a set of guarded commands. Modules can communicate with each other through global variables. PRISM also supports module synchronization through action labels.

PRISM constructs the global probabilistic model through parallel composition of the component modules following the interleaving semantic of the parallel composition operator. The data structures in PRISM are based on BDDs (Binary Decision Diagrams [45]) and MTBDDs (Multi-Terminal Binary Decision Diagrams [46]). PRISM supports simulations both random and guided and also supports a wide range of probabilistic temporal logics to specify properties to be verified such as PCTL, PCTL* and CSL. We have used the PRISM model checker to verify our MDP models of CDC interfaces. The next section elaborates on the proposed methodology used to verify these interfaces.

5.4 The Proposed Verification Framework

As discussed in Section 5.1, the safety of the CDC interface is affected by both the protocol used as well as the the synchronizer characteristics. Our verification framework is the first framework that captures both these factors in its model. The proposed framework is shown in Figure 5.2.

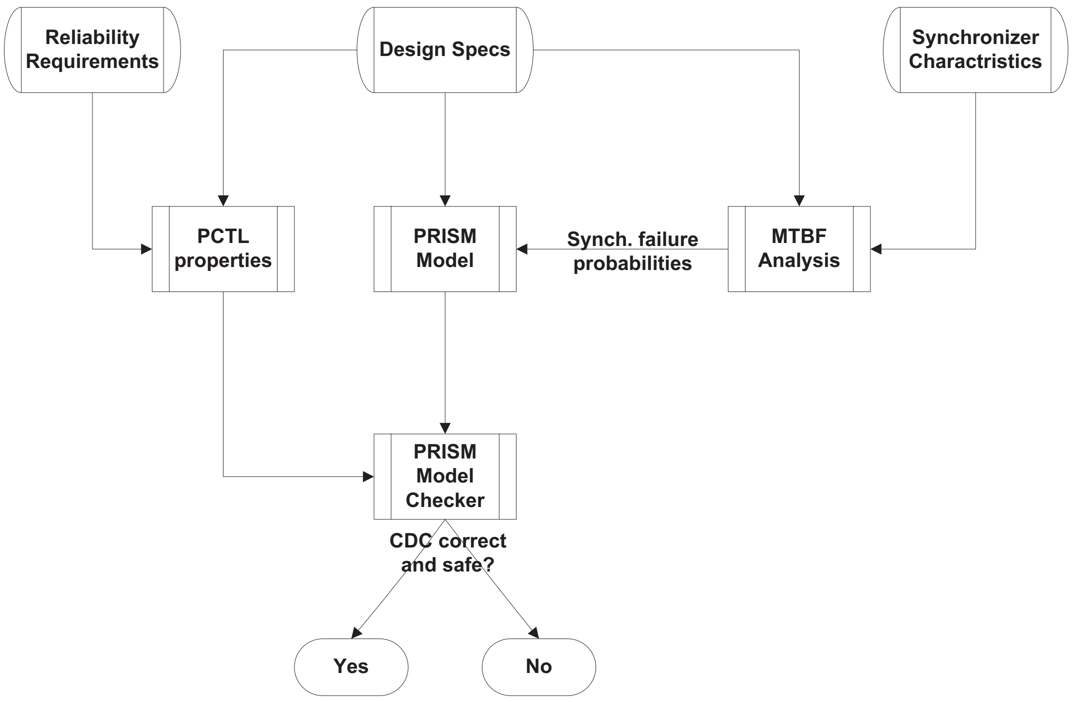

Figure 5.2: The proposed verification framework

The proposed methodology uses both the CDC specification as well as the synchronizer circuit specification as part of its MDP model of the system. These two specs are totally different. Design specs refer to the description of the protocol used in the CDC interface. The structure of the PRISM model for verifying the system is developed based on the protocol description. The model is written in PRISM language as a description for an MDP from which PRISM generates the underlying MDP model. One missing detail in the MDP model which needs to be obtained from synchronizer characteristics is the probability of falling into metastability. This can be obtained using circuit-level MTBF analysis such as in [37], [38].

The properties to be verified depend of the model studied and which aspects of the design the designer wants to verify. Some properties are common to all CDC interfaces and some are specific to some CDC protocols. Reliability requirements for the CDC interface are also used to generate the properties to be verified. The properties are written in PCTL and PCTL*. The PRISM model checker then verifies these properties to check whether the CDC interface is correct and safe.

In our verification model, metastability was modeled explicitly as a state in the system. This modeling can be illustrated with the model for a 0-to-1 transition of a synchronizer as shown in Figure 5.3.

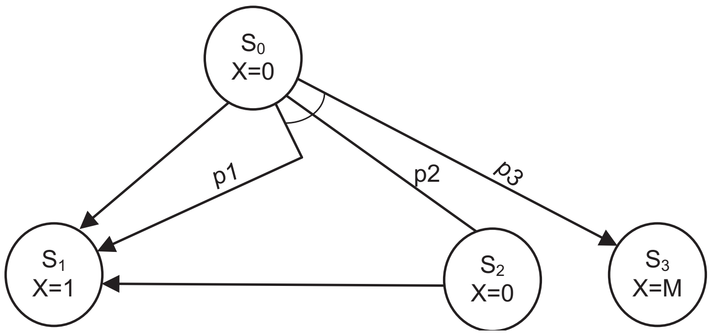

Figure 5.3: Metatsability model for a 0 -> 1 transition of a flip-flop

The transition is modeled as a non-deterministic choice between a normal operation mode in which it does not get into metastability and a metastability mode in which it enters metastability. When a synchronizer enters metastability, its output is set to one of three possible outputs 0, 1, M. Probabilities of entering each state p1, p2, p3 depend on the type and design of the flip-flops or synchronization circuits used. The probabilities can then be plugged in into PRISM for verifying the system as a whole in the presence of metastability. A designer might be especially interested in the overall behavior of the system for a given probability of metastability (p3). The PRISM code representing the metastability model is given below:

```
[] s=0 -> (s'=1)&(X'=1);
[] s=0 -> p1:(s'=1)&(X'=1)+p2:(s'=2)&(X'=0)+p3:(s'=3)&(X'=2)&(metas'=true);
[] s=2 -> (s'=1)&(X'=1);
```

The next few sections show case studies of our verification methodology for three different CDC interfaces.

5.5 Verifying the Boundary Synchronization Based CDC

In the remaining part of this chapter, the modeling of common CDC interfaces in PRISM will be discussed. This section focuses on the verification of the boundary synchronization based CDC interface discussed in Section 2.2.2. Our modeling technique is explained using this interface because it is simple and easy to follow. The boundary synchronization based CDC is shown in Figure 5.4.

Our model of the this interface is shown in Figures 5.5 and 5.6. The sender module representing the sender side of the CDC interface is shown in Figure 5.5 and the receiver module representing the receiver side of the CDC is shown in Figure 5.6.

The interface is modeled as MDP. Metastability is modeled as shown in Figure 5.3. The sender and receiver interact through global variables. As shown in Figure 5.5, The transitions correspond directly to the boundary synchronization based CDC protocol in Section 2.2.2. When both the modules are in their initial state, the send signal which is an input to the sender part of the CDC starts the handshake cycle.

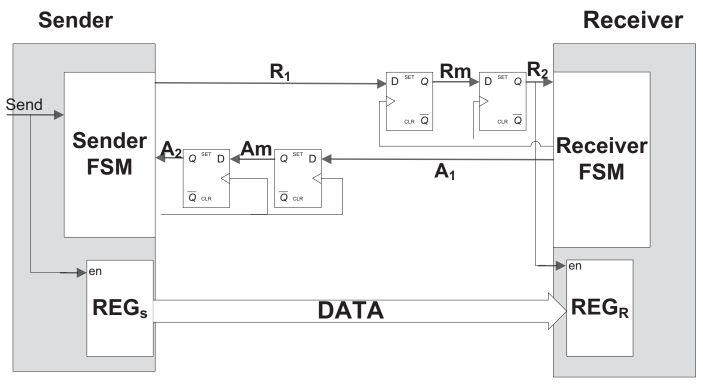

Figure 5.4: The boundary based CDC interface verified

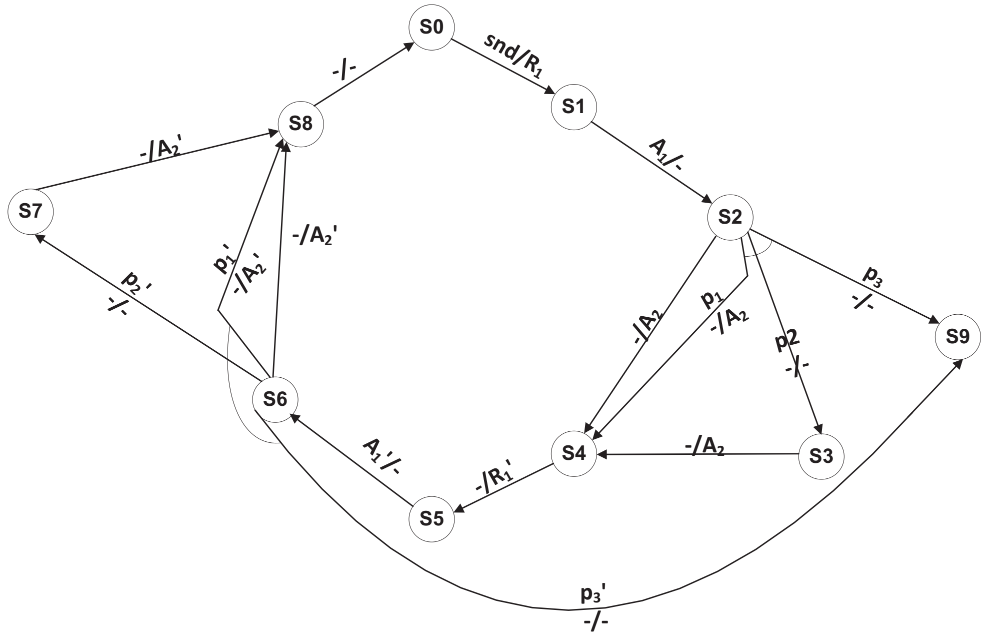

Figure 5.5: The sender side of the CDC

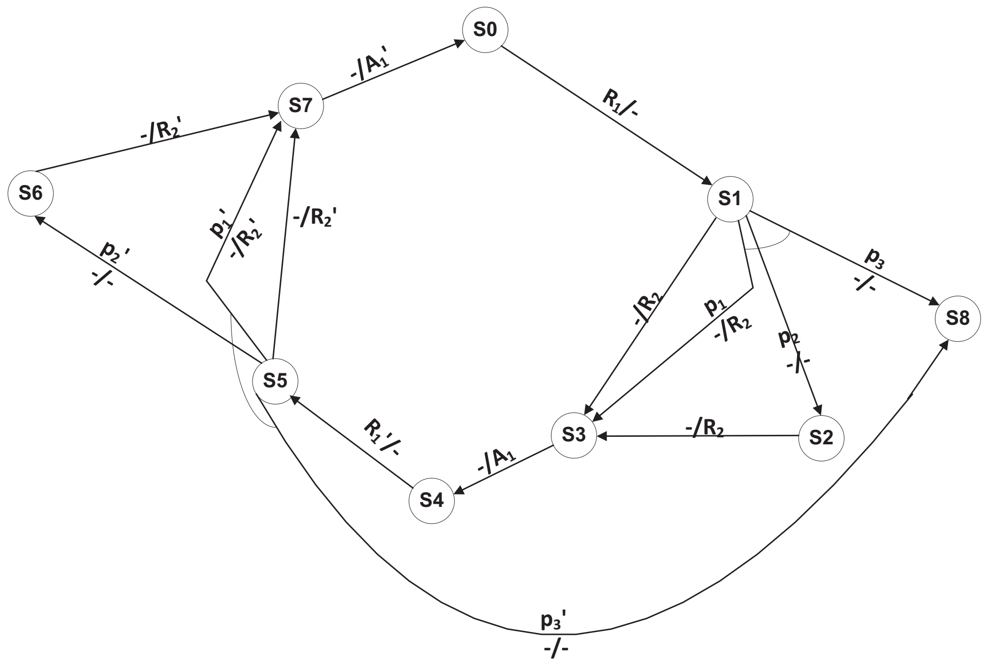

Figure 5.6: The receiver side of the CDC

The sender module drives R_1 to logic 1 in response. The R_1 signal gets synchronized into the receiver and the receiver module moves from the initial state to state S1. In this state R_m is asserted to logic 1. Afterwards, the receiver's MDP model chooses non-deterministically between the left transition from state S1 in Figure 5.6 to state S3 (modeling the case in which metastability does not occur) and the right transition from state S1 (modeling the case in which metastability does occur). This second transition is further subdivided into three probabilistic choices:

- The synchronizer converges to logic 1 before the next clock edge (represented by the transition to state S3 with probability p1).
- The synchronizer converges to logic 0 before the next clock edge (represented by the transition to state S2 with probability p2).
- The synchronizer does not converge to a stable logic value by the next clock

edge (represented by the transition to state S8 with probability p3).

The two modules then continue following the request/ack mechanism as defined in the protocol description in Section 2.2.2. For the present time, assumptions are made for the values of synchronizer failures and convergence probabilities [p1, p2, p3] to prove the concept of our verification framework. These values are affected by several factors such as the type of synchronization circuit, the type of flip-flops, and technology used. The probability of the sender flip-flop entering metastability and staying for more than one cycle was assumed to be $5 * 10^{-4}$, and for the receiver flip-flop, it was assumed to be $2.5 * 10^{-4}$. Several references such as [37] discuss the probability of failures for different types of flip-flops.

Three properties for this model were checked:

- P1: Pmin=? [(snd=true) => (F(A_2=1))] Returns the minimum probability that a request to send data is acknowledged.

  F is a the temporal operator commonly referred to as "evenually". *F p*, where *p* is a property, is true in a path if *p* becomes true at some point in that path.

- P2: Pmax=? [(request=false) and (R_2=1)] Returns the maximum probability that data is latched to the receiver without being sent by the sender. Where request is a variable that keeps track of ongoing requests that have not been acknowledged.

- P3: Pmax=? [F<K (metas=true)] Returns the maximum probability that metastability occurs within K cycles.

The PRISM model constructed consists of 90 states and 339 transitions. Memory requirements for the constructed model were 56 KB. The table below shows the returned result and the time required for model checking for each of the properties specified. The verification has been conducted on a machine with an Intel Core i5 CPU running at 2.27 GHz and with a 4 GB memory. It is natural that the probability that the system enters a metastable state increases with time as more requests are synchronized. Figure 5.7 shows the probability of entering a metastable state as a function of the number of cycles (K). The probability increases quickly at the start and as the number of cycles exceeds 25000, it slowly converges to 1.

Table 5.1: Verification results for boundary synchronization CDC

Property	Model Checking Time (Seconds)	Result (P=?)
P1	0.037	0.99925
P2	0.031	0
P3 (K=1000)	0.027	0.095

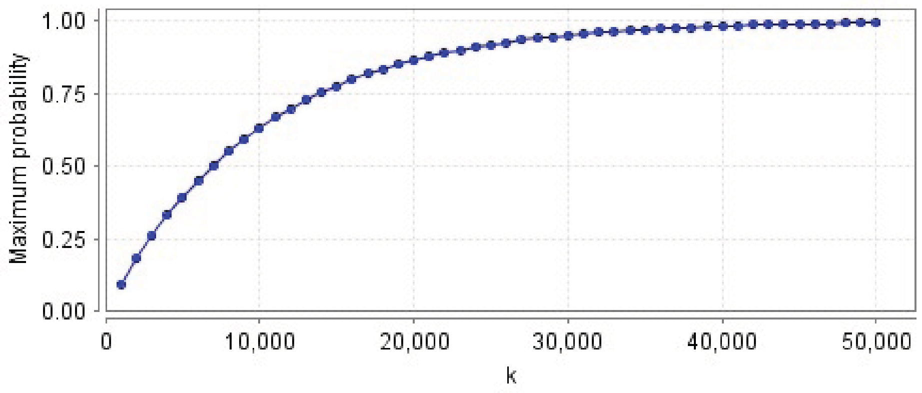

Figure 5.7: Maximum probability as a function of number of cycles for boundary synchronization CDC

5.6 Verifying FIFO Based CDC

The FIFO interface as discussed in Secion 2.2.3 was modeled in PRISM in a similar way to the boundary synchronization protocol interface. An MDP module for a put interface and a get interface was constructed. A FIFO size of 4 cells was modeled. The same metastability probabilities of $5 * 10^{-4}$ and $2.5 * 10^{-4}$ were used. The PRISM model constructed consists of 33128 states and 157680 transitions. Memory requirements for the constructed model were 4.2 MB. Some properties verified are shown below:

- P1: Pmin=?[((reqput)and(!full)and(ptoken=n))=> (F((reqget)and(!empty)and(gtoken=n)))] Returns the minimum probability that a data item written to the FIFO is eventually read.
- P2: Pmax=? [F $<$ K (metas=true)] Returns the maximum probability that metastability occurs within K cycles.
- P3: Pmax=?[(enput)and(ptoken=n)and(cfull$_n$=true)] Where enput is an internal signal in the FIFO that enables writing, and cfull$_n$ indicates whether cell n of the FIFO is full. The property returns the maximum probability of writing to a full cell.
- P4: Pmax=?[((enget)and(gtoken=n)and(cfull$_n$=false)] Where enget is an internal signal in the FIFO that enables reading. The property returns the maximum probability of reading from an empty cell.

The verification results are shown in the table below. Figure 5.8 shows the probability of entering a metastable state as a function of the number of cycles. The probability is generally higher than boundary synchronization based interface since the interface is more complex and data can be transmitted at a higher rate leading to a higher frequency of input changes at the inputs of the flip-flops.

Table 5.2: Verification results for FIFO

Property	Model checking time (seconds)	Result (P=?)
P1	6.393	1
P2 (K=1000)	7.248	0.28
P3 (n=0)	0.139	0
P4 (n=0)	0.802	0

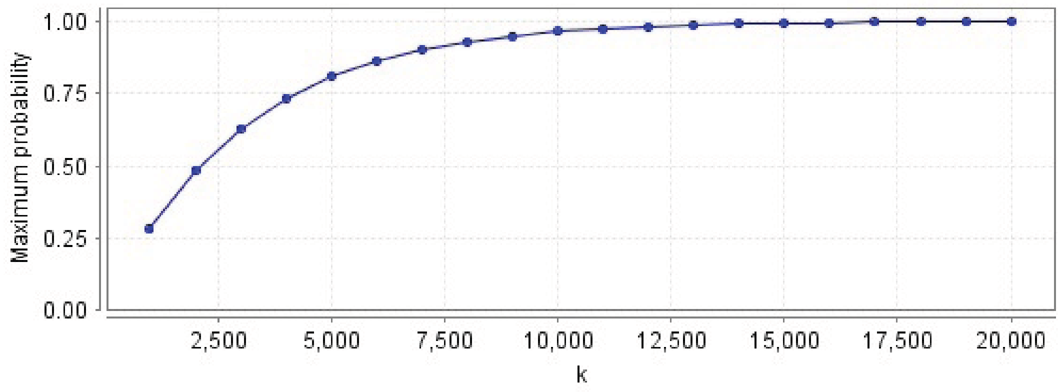

Figure 5.8: Maximum probability as a function of number of cycles for FIFO

5.7 Verifying The Proposed CDC

The proposed design which was discussed in Chapter 3 was also modeled and verified using the new verification approach. A PRISM model consisting of 203 states and 757 transitions was constructed. The same metastability probabilities of $5 * 10^{-4}$ and $2.5 * 10^{-4}$ were used. Memory requirements were 69KB. Some properties verified are shown below:

- P1: Pmin=? [(T=true)=> (F (ACK=1))] Returns the minimum probability that a request to send data is acknowledged.
- P2: Pmax=? [(fifoEmpty=true)and(A_p=true)] Returns the maximum probability that data is latched to the receiver without being sent by the sender.
- P3: Pmax=? [F < K (metas=true)] Returns the maximum probability that metastability occurs within K cycles.

The verification results are shown in the table below. Figure 5.9 shows the probability of entering a metastable state as a function of the number of cycles. The probability is generally lower than the other two protocols indicating a more robust interface at the protocol level.

Table 5.3: Verification results for the proposed CDC

Property	Model Checking Time (Seconds)	Result (P=?)
P1	0.054	1
P2	0.040	0
P3 (K=1000)	0.05	0.055

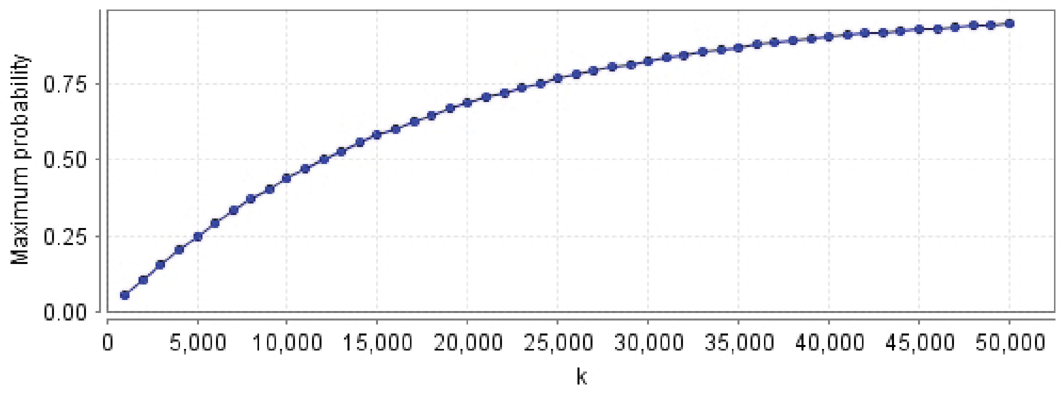

Figure 5.9: Maximum probability as a function of number of cycles for the proposed CDC

5.8 Summary

In this chapter, a framework for verifying CDC interfaces using probabilistic model checking was presented. The framework allows for modeling metastability. CDC interfaces are modeled as Markov Decision Processes integrating both probabilistic and non-deterministic behavior. Three different CDC interfaces; boudary synchronization based CDC, FIFO based CDC, and our proposed CDC were modeled and verified using this approach. Properties were written in PCTL and PCTL* and verified using the PRISM model checker.

Chapter 6

Conclusion and Future Work

In this thesis, we have presented a novel interface design to be used as a clock domain crossing in chips that employ many clock domains. The proposed design is especially useful in the booming SoC design field where components of varying characteristics are being integrated on a single chip. The proposed design allows the data and control signals to cross clock domain boundaries safely. The design relieves the communicating modules of the communication overhead and allows them to continue their normal operation while the interface takes care of the transmission.

The proposed technique is a hybrid technique that uses the benefits and avoids the shortcomings of other techniques. Unlike previously proposed CDC interfaces, the new interface does not require the communicating modules to stop their operation during transfers or to have a particular frequency ratio range. The interface resolves these issues by using a mixed synchronous/asynchronous communication protocol utilizing the asynchronous part for the handshake and the synchronous part for communication with the synchronous sender and receiver. The interface uses two special circuits (protocol-pausers) custom designed at the transistor level to separate the synchronous world from the asynchronous world by pausing the protocol whenever there is a fear of timing violations.

The proposed technique was implemented block-by-block at the transistor level using TSMC 90nm technology and extensively simulated using SPICE. The proposed design proved robust and continued to function correctly even in extreme conditions such as low and high temperatures, varying frequencies, and different workloads. Even with worst case phase relationships tested at very fine resolutions, the design produced distinct output signals. The design has a maximum throughput of 606 M data items/s and a low latency. The design operates regardless of the frequency ratio of the communication modules.

A methodology for the formal verification of CDC interfaces was also proposed. To our knowledge this is the first methodology for formal verification of CDC interfaces that takes into account the failure probability of the synchronizers. The framework uses probabilistic model checking and models the system as Markov Decision Processes (MDP) which allows the modeling of probabilistic and non-deterministic behavior.

Common CDC interfaces, namely boundary synchronization based CDC interface and FIFO based CDC interface were verified using the proposed framework. The proposed CDC interface was also verified using the new approach. The PRISM probabilistic model checker was used for verification. The proposed framework is important for checking the correctness of error-prone CDC interfaces as it allows for verifying design properties in a more realistic environment in which a system contains possibly many failure-prone synchronization circuits. The framework provides the minimum and/or maximum probabilities that a certain probability is satisfied by the system in the presence of metastability.

The proposed CDC design can be optimized in the future by applying proper transistor sizing to achieve better performance. The promising results of the pauser circuit means that this circuit could form the basis of new designs for other similar problems. One interesting field for improvement will be extending the interface to allow more than two modules to communicate at the same time. This also embeds the challenge of arbitration if more than one request arrive exactly at the same time.

Our future work also includes exploring MTBF analysis for different synchronization circuits to generate accurate failure probabilities to be included within our verification framework.

Bibliography

[1] Feng Y, Zhou Z, Tong D, Cheng X. Clock domain crossing fault model and coverage metric for validation of SoC design. In Proc. Conference on Design, Automation and Test in Europe (DATE'07), Nice, France, April 16-20, 2007, pp. 1385-1390.

[2] Kinniment, D. J., Synchronization and Arbitration in Digital Systems, 2007, John Wiley & Sons, Ltd.

[3] Ginosar, R., Metastability and Synchronizers: A Tutorial, IEEE Design and Test of Computers, Vol. 28, Issue 5, pp. 23-35, 2011.

[4] ITRS 2009, available online at http://www.itrs.net/Links/2009ITRS/Home2009.htm.

[5] Muttersbach, J., Villiger, T., and Fichtner, W. Practical Design of Globally-Asynchronous Locally-Synchronous Systems. In Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC'00), Eilat, Israel, 2000, pp. 52-59.

[6] R. William J. Dally and John W. Poulton, Digital Systems Engineering, Cambridge University Press, 1998.

[7] Cummings C., Clock Domain Crossing (cdc) Design and Verification Techniques Using System Verilog, SNUG-2008, Boston, MA, 2008.

[8] H. Veendrick, The behavior of flip-flops used as synchronizers and prediction of their failure rate. IEEE Journal of Solid-state Circuits, vol. SC-15, no. 2, pp. 169-176, 1980.

[9] C. L. Portmann et al., Metastability in CMOS library elements in reduced supply and technology scaled applications, IEEE Journal of Solid-State Circuits, vol. 30, pp. 39-46, 1995.

[10] U. Ko and P. Balsara. High-Performance Energy-Efficient D-Flip-Flop Circuits. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 8, 2000, pp. 94-98.

[11] M. Krstic, E. Grass, F. K. Gurkaynak, and P. Vivet. Globally asynchronous, locally synchronous circuits: Overview and outlook. IEEE Design & Test of Computers, vol. 24, no. 5, pp. 430-441, 2007.

[12] Jens Muttersbach. Globally-Asynchronous Locally-Synchronous Architectures for VLSI Systems. Ph.D. thesis, ETH, Zurich, 2001.

[13] Daniel M. Chapiro. Globally-Asynchronous Locally- Synchronous Systems. Ph.D. thesis, StanfordUni versity, October 1984.

[14] Z. K. Y. Yun and R. P. Donohue. Pausible clocking: A first step toward heterogeneous systems. In Proc. International Conf. Computer Design (ICCD'96), Austin, TX, 1996, pp. 118-123.

[15] Dobkin, R., Ginosar, R., and Sotiriou, C.P. Data Synchronization Issues in GALS SoCs. in 10th IEEE International Symposium on Asynchronous Circuits and Systems, April 2004, pp. 170-179.

[16] Teehan, P., Greenstreet, M., and Lemieux, G., A Survey and Taxonomy of GALS Design Styles. In IEEE Design & Test of Computers, Vol. 24, Issue 5, pp. 418-428, 2007.

[17] A. Iyer and D. Marculescu. Power and performance evaluation of globally asynchronous locally synchronous processors. In Proceedings of the 29th Annual International Symposium on Computer Architecture, , 2002, pp. 158-168.

[18] F.K. Gurkaynak et al., GALS at ETH Zurich: Success or Failure?. Proc. 12th IEEE Intl Symp. Asynchronous Circuits and Systems (ASYNC'06), 2006, IEEE CS Press, pp. 150-159.

[19] R. Ginosar, Fourteen ways to fool your synchronizer, in Proc. IEEE Int. Symp. Asynchronous Circuits and Systems (ASYNC'03), 2003, pp. 8996.

[20] T. Chelcea and S. M. Nowick, Robust interfaces for mixed-timing systems, IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 12, no. 8, pp. 857-873, 2004.

[21] A. Chakraborty and M. Greenstreet, Efficient Self-Timed Interfaces for Crossing Clock Domains. Proc. 9th Intl Symp. Asynchronous Circuits and Systems (ASYNC'03), 2003, IEEE CS Press, pp. 78-88.

[22] Mekie, J., Chakraborty, S., Venkataramani, G., Thiagarajan, P.S., and Sharma, D.K. Interface design for rationally clocked GALS systems. In Proc. 12th IEEE Intl. Symp. on Asynchronous Circuits and Systems (ASYNC'06), 2006, pp. 160-171.

[23] T. Kapschitz and R. Ginosar, Formal verification of synchronizers. CHARME 05, vol. 3725 of LNCS, pp. 359-362. 2005.

[24] E. M. Clarke, D. Kroening, and K. Yorav, Specifying and Verifying Systems with Multiple Clocks. In Proc. 21st International Conference on Computer Design (ICCD'03), 2003, pp. 48-55.

[25] Brown, G., Verification of a Data Synchronization Circuit For All Time. In Proc. 6th International Conference on Application of Concurrency to System Design (ACSD'06), 2006, pp. 217-228.

[26] Safranek, D., Smrcka, A., Vojnar, T., Rehak, V., Rehak, Z., and Matousek, P, Verifying VHDL Design with Multiple Clocks in SMV, FMICS, vol. 4346 of LNCS, Springer, 2006.

[27] K. Yorav, S. Katz, and R. Kiper. Reproducing synchronization bugs with model checking. In CHARME'01, LNCS Volume 2144/2001, pp. 98-103, Springer, 2001.

[28] The PRISM manual. available online at http://www.prsimmodelchecker.org.

[29] V. Forejt, M. Kwiatkowska, G. Norman and D. Parker, Automated Verification Techniques for Probabilistic Systems. Formal Methods for Eternal Networked Software Systems (SFM'11), vol. 6659 of LNCS, pp. 53-113, Springer, June 2011.

[30] E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In Logic of Programs: Workshop, Yorktown Heights, NY, May 1981, volume 131 of Lecture Notes in Computer Science. Springer-Verlag, 1981.

[31] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07), volume 4486 of LNCS, pp. 220-270. Springer, 2007.

[32] A logic for reasoning about time and reliability. Formal Aspects of Computing, Vol 6,pp. 102-111, 1994.

[33] C. Baier., On algorithmic verification methods for probabilistic systems, Habilitation thesis, Faculty of Mathemaitcs and Informatics, University of Mannheim. 1998.

[34] M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic model checking for performance and reliability analysis. ACM SIGMETRICS Performance Evaluation Review, Vol. 36, No. 4,pp. 40-45, 2009.

[35] V. Shmatikov. Probabilistic Model Checking for Security Protocols [online]. Accessed on March 10th, 2012.
http://www.stanford.edu/class/cs259/WWW04/lectures/07-Probabilistic%20Model%20Checking.pdf.

[36] M. Kwiatkowska, G. Norman, and D. Parker, PRISM 4.0: Verification of Probabilistic Real-time Systems, In Proc. 23rd International Conference on Computer Aided Verification (CAV11), 2011, vol. 6806 of LNCS, pp. 585-591, Springer.

[37] D. Li, P. Chuang, and M. Sachdev, Comparative Analysis and Study of Metastability on High Performance Flip-flops," in 11th International Symposium on Quality Electronic Design, March 2010, pp. 853-860.

[38] Foley, C., Characterizing metastability, Second International Symposium on Advanced Research in Asynchronous Circuits and Systems, 18-21 Mar 1996, pp. 175-184.

[39] Kinniment, D.J., Bystrov, A., Yakovlev, A.V., Synchronization circuit performance, IEEE Journal of Solid-State Circuits, vol.37, no.2, pp. 202-209, Feb 2002.

[40] Chelcea, T., Nowick, S.M., Low-latency asynchronous FIFO's using token rings, Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC'00), 2000, pp. 210-220.

[41] Smrcka,A.: Verifcation of Asynchronous and Parametrized Hardware Designs, Information Sciences and Technologies Bulletin of the ACM Slovakia, Vol. 2, No. 2, pp. 60-69, 2010.

[42] R. Dobkin, T. Kapshitz, S. Flur and R.Ginosar, Assertion Based Verification of Multiple-Clock GALS Systems,. Proc. IFIP/IEEE Int. Conference on Very Large Scale Integration (VLSI-SoC), 2008.

[43] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In Proceedings of the 5th Colloquium on International Symposium on Programming, pages 337-351, London, UK, 1982. Springer- Verlag.

[44] Puterman M., Markov Decision Processes: Discrete Stochastic Dynamic Programming (1st ed.), John Wiley Sons , New York, NY, USA., 1994.

[45] R. E. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, Vol. 35, No 8, pp. 677-691, 1986.

[46] M. Fujita, P.C. McGeer and J.C.-Y. Yang, Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. Formal Methods in System Design, Vol. 10, Numbers 2-3, pp. 149-169, 1997.