Newcastle University, UK

School of Electrical and Electronic Engineering (EEE)

Solutions and Application Areas of Flip-Flop Metastability

PhD Thesis

Ghaith Tarawneh

August 2013

Abstract

The state space of every continuous multi-stable system is bound to contain one or more metastable regions where the net attraction to the stable states can be infinitely-small. Flip-flops are among these systems and can take an unbounded amount of time to decide which logic state to settle to once they become metastable. This problematic behavior is often prevented by placing the setup and hold time conditions on the flip-flop's input. However, in applications such as clock domain crossing where these constraints cannot be placed flip-flops can become metastable and induce catastrophic failures. These events are fundamentally impossible to prevent but their probability can be significantly reduced by employing synchronizer circuits. The latter grant flip-flops longer decision time at the expense of introducing latency in processing the synchronized input.

This thesis presents a collection of research work involving the phenomenon of flip-flop metastability in digital systems. The main contributions include three novel solutions for the problem of synchronization. Two of these solutions are speculative methods that rely on duplicate state machines to pre-compute data-dependent states ahead of the completion of synchronization. Speculation is a core theme of this thesis and is investigated in terms of its functional correctness, cost efficacy and fitness for being automated by electronic design automation tools. It is shown that speculation can outperform conventional synchronization solutions in practical terms and is a viable option for future technologies. The third solution attempts to address the problem of synchronization in the more-specific context of variable supply voltages. Finally, the thesis also identifies a novel application of metastability as a means of quantifying intra-chip physical parameters. A digital sensor is proposed based on the sensitivity of metastable flip-flops to changes in their environmental parameters and is shown to have better precision while being more compact than conventional digital sensors.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost I would like to thank my supervisor Prof. Alex Yakovlev for having the patience and wisdom to guide me throughout the past four years. His contributions to my academic development are only outmatched by his continuous encouragement and endless personal support. I would like to express my sincere gratitude to him for being a kind mentor, a thoughtful friend and a caring father.

I would also like to thank all my friends and colleagues in the Microelectronics System Design research group whose genius thoughts and mind-provoking discussions have been an integral part of my life over the past four years. My gratitude extends to the academic and non-academic staff in the School of Electrical and Electronic Engineering who have taken part in supporting me as a student and have made my PhD the worthwhile experience that it was. Special thanks to my family and dear friends in both the UK and Jordan for the continuous encouragement over the years.

Finally, I would like to dedicate this work to the late Prof. David Kinniment whom I have known briefly during my first year as a PhD student. Prof. Kinniment suggested that I use a generic simulation tool, which I was developing at the time, to investigate the behavior of metastable flip-flops, setting an unforgettable start to my interest in the phenomenon and culminating in the production of this thesis.

This work was supported by EPSRC grant EP/G066361/1.

Chapter 1

Introduction

1.1 Motivation

For the longest part of the history of integrated circuits, synchronous operation has allowed designers to put together an ever-larger number of components without having to worry much about the complex timing issues of their interoperability. The notion of using a single clock signal to synchronize an entire system is among the basics that can be found on the first pages of many computer design textbooks. Synchronicity, however, is neither a fundamental nor a characteristic feature of computers. To see this it is necessary to first realize that the word "computer" does not solely refer to man-made electronic systems, nor to the larger set including such things as analog, mechanical and biological machines. Instead, all physical processes are computations and consequently *every physical object is a computer*. This is so because every physical object computes its own state based on a set of equations (which us humans try to learn incrementally by practicing science). The computations underlying these states do not rely on any synchronization reference and are performed in a most concurrent fashion. Concurrency, it seems, it a more natural feature of computers.

Electronic man-made computers also have a substantially lower computational density compared to their natural physical counterparts. Computing the atom-level-accurate state of a sand grain, for instance, requires an unimaginable number of

operations that can keep the fastest of our CMOS computers running for millennia. Yet the same computation is performed by a sand grain in an imperceivable time step. Granted that a sand grain is neither a Turing-complete computer nor a system whose state is particularly compute-worthy, the comparison reveals how far behind is our forefront classical computer technology (CMOS) from the stunning density of computations permissible by the laws of physics. Scaling our classical computers down to this ideal (throughout CMOS and beyond) will necessarily require us to free them from our self-imposed synchronous constraints.

Luckily, this appears to be already underway. The last few VLSI technology generations have managed to deliver Moore's performance by stepping away from conventional synchronous single-core architectures towards slightly more concurrent many-core systems. More steps in the same direction must now be taken as the number of on-chip cores and the degree of heterogeneity in modern Systems-on-Chip (SoCs) continue to increase. Classical computers are thus slowly (but surely) making their way towards complete concurrency and are most likely to make an inadvertent transition through Globally-Asynchronous-Locally-Synchronous (GALS) systems on their way.

Clock domain interfacing and the problem of flip-flop metastability are among the challenges that must be addressed to facilitate this transition and support the creation of more powerful heterogeneous many-core systems. When components in different clock domains attempt to communicate, the receiver is always at a risk of failure due to the finite probability that sender's request arrives at a bad time. Such occurrences can cause flip-flops on the receiving module to become "metastable" and take a theoretically-unbounded time to decide whether to go logic high or low. If this indecisiveness persists for long, the state machine that hosts the flip-flop can enter an invalid and possibly-unrecoverable state. This phenomenon has long intrigued researchers due its philosophical roots, the illusive nature of the resulting failures and the large numbers of failed attempts that have been made at avoiding it.

On a small scale, the collection of work presented in this thesis aims at contributing to our understanding of the problem of synchronization and the phenomenon of metastability (in flip-flops and other multi-stable physical systems). Particular emphasis is made on the practicality of the presented solutions in terms of their area, power,

latency and reliability costs and perhaps most importantly on their suitability for automation by EDA tools. On a larger scale, it is hoped that the presented work will contribute to advancing our electronic classical computers on the long road towards true concurrency and higher computing densities.

1.2 Main Contributions

The main contributions of this thesis are as follows.

First, two new solutions are proposed for mitigating synchronization latency between different clock domains. The solutions rely on speculation which has been suggested as a work-around technique to hide synchronization latency [2] but has not been explored in depth. The bulk of this thesis presents two novel speculative approaches that provide reliable low-latency communication at the expense of increasing design area. Design heuristics are introduced to reduce the necessary amount of hardware duplication and it is shown that, when using the suggested heuristics, the overall costs are comparable to existing clock domain crossing solutions.

Second, an adaptive synchronizer for Dynamic Voltage Frequency Scaling (DVFS) is presented. The proposed design can adjust metastability resolution time on the fly and hence provide lower average latency compared to conventional synchronizers which are designed for worst case performance. Of particular importance, while resolution time is adjusted in coarse steps (half a clock cycle), the proposed design does not include any arithmetic circuits and hence its area and power costs are negligible. Although similar designs which are able to fine-tine metastability resolution time have been proposed [3], these relied on multipliers and logarithm circuits whose significant costs are typically unaffordable at clock domain interfaces.

Third, the metastable behavior of flip-flops is proposed as a method to sense intra-chip physical parameters such as voltage, temperature and parametric variations. Conventional digital intra-chip sensor designs are very limited because of the difficulty of measuring analog quantities using digital components. The resolution speed of a metastable flip-flop is highly sensitive to its operating conditions and consequently can be exploited to quantify them. Following on this idea, a novel digital sensor for

intra-chip physical parameters is introduced. The proposed design relies on deliberately bringing a flip-flop into metastability and using its failure rate to quantify intra-chip supply voltage and temperature.

1.3 Thesis Organization

Chapter 1: Introduction provides a brief overview of the context of this thesis, outlines the primary motivations that have driven this work and summarizes the presented contributions.

Chapter 2: Background attempts to provide a grounds-up introduction of the problem and describe its fundamental tenets. The chapter also surveys areas of digital design where metastability persists (or is deliberately introduced) and concludes by defining a few terms for particular forms of behavior which are consistently referred to within the thesis.

Chapter 3: Hiding Synchronization Latency by Speculation starts by surveying existing clock domain interfacing solutions and making the case for speculative solutions. Two speculative synchronization solutions are then introduced: *Datapath Unfolding* and *Sequenced Latching*. The chapter concludes by comparing the three speculative methods (the two proposed and the pilot technique in [2]) and making a few notes on designing systems for speculative synchronization.

Chapter 4: Adaptive Synchronization for DVFS discusses the compounded difficulties of adjusting synchronization time in DVFS systems and presents a novel adaptive synchronizer that outperforms conventional synchronizers in these cases.

Chapter 5: Physical Parameter Sensor for FPGAs introduces a novel digital sensors that exploits flip-flop metastability to quantify intra-chip physical parameters such as voltage and temperature.

Chapter 6: Conclusions summarizes the contributions of the thesis and recommends interesting areas for further investigation.

1.4 Publications

1. Tarawneh, G.; Yakovlev, A.; Mak, T., "**Eliminating Synchronization Latency Using Sequenced Latching**," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.PP, no.99, pp.1,1, 0 doi: 10.1109/TVLSI.2013.2243177

2. Tarawneh, G.; Yakovlev, A., "**An RTL method for hiding clock domain crossing latency**," Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on , vol., no., pp.540,543, 9-12 Dec. 2012 doi: 10.1109/ICECS.2012.6463557

3. Tarawneh, G.; Mak, T.; Yakovlev, A., "**Intra-chip physical parameter sensor for FPGAS using flip-flop metastability**," Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on , vol., no., pp.373,379, 29-31 Aug. 2012 doi: 10.1109/FPL.2012.6339207

4. Tarawneh, Ghaith, and Alex Yakovlev. "**Adaptive Synchronization for DVFS Applications**." Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation. Springer Berlin Heidelberg, 2013. 93-102.

Chapter 2

Background

2.1 Synchronous Logic Fundamentals

This thesis presents ideas and circuits that involve flip-flops whose setup and hold times have not been constrained. Before discussing these unconventional designs, it is necessary to cover the fundamentals of flip-flop use and behavior in conventional synchronous logic. This background will serve as a reference for the discussions in the following chapters.

2.1.1 Latches and Flip-Flops

Latches are the elementary blocks of computer memory. Every latch contains a positive feedback loop that has two stable electrical states corresponding to logic high and low. The loop is created by cross-coupling two inverting gates and typically contains an additional switch that opens or closes the loop. The latch also contains input buffers which function as gatekeepers and are used to pull the loop to either state. When these buffers are enabled, the output copies or "follows" the input and the latch is said to have become *transparent*. When the buffers are disabled, the latch retains the last copied value and becomes *opaque*. The latch outputs are isolated from cross-coupled inverting gates by one or more output buffers which reduce the loading of the feedback loop and allow the latch to drive larger loads. Figure 2.1 shows a typical library latch.

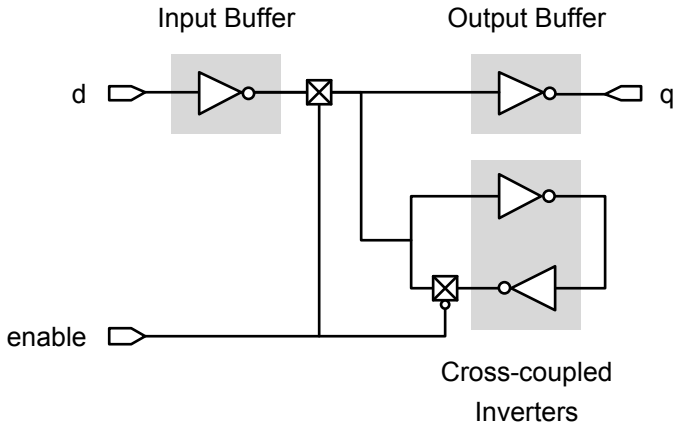


Figure 2.1: Latch circuit

Latches are referred to as level-sensitive devices because they are triggered by the level of the enable signal. An alternative, an edge-triggered device, can be obtained by cascading two latches, a master and a slave, which are enabled in alternating phases. The resulting circuit is referred to as a flip-flop. The input of a flip-flop is copied to its output at the instance the enable signal (now called the clock) is asserted. In other words, flip-flops retain input values at the active edge of the enable signal.

Cell libraries contain flip-flops of different drive strengths and with different combinations of set-reset, asynchronous inputs and scan functions. All these variations share the common structure described above: they consist of two latches, each in turn containing a positive feedback loop and a number of input and output buffers.

2.1.2 Timing Constraints

Flip-flops and logic gates are the building blocks of sequential logic and finite state machines. Sequential circuits can be classified as synchronous or asynchronous depending on how their state transitions take place. Synchronous sequential systems are those whose state transitions are performed at regular time intervals and are synchronized by a global clock signal that feeds to all memory elements. Asynchronous sequential systems, on the other hand, rely on localized component-based handshakes to trigger computations and update subsets of the machine's state register.

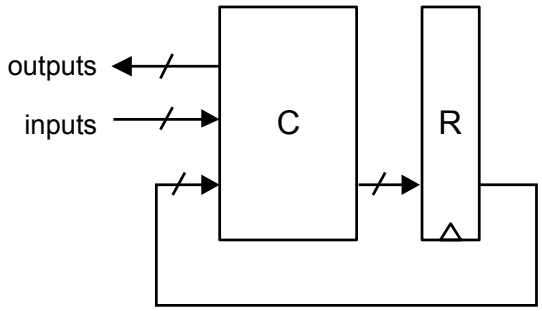


Figure 2.2: Mealy machine

Synchronous systems can be further divided into several categories depending on how they are clocked. Edge-based, pulse-based or two-phase clocking offer different combinations in the trade-offs between glitch-free operation, structural and timing constraints and the number of clock signals. Of these options, edge-based clocking is predominant in most modern digital systems. An edge-based sequential system can be represented by the generic Mealy machine shown Figure 2.2 where the *C* is the machine's combinational logic and the *R* is its state register.

For a synchronous design to behave correctly, a number of timing conditions must be satisfied. Two of these conditions govern the propagation of the state register outputs back to its inputs: the setup and hold time conditions. These conditions are of particular importance because their satisfaction must be guaranteed at design time, i.e. they are design constraints.

Setup Time

Synchronous machines are clocked at a fixed rate. At every clock edge, a subset of the machine's state bits are changed and the transitions of these bits propagate through series of logic gates (i.e. combinational logic paths) back to the state register inputs. The clock period must be long enough to let this process complete before the following clock edge. This is referred to as the setup time condition and must be satisfied for all the combinational paths between the machine's flip-flops. The setup condition of a path

between two flip-flops $FF1$ and $FF2$ can be expressed as:

$$t_{\text{clk-q}} + t_{\text{pd}} + t_{\text{su}} + t_{\text{skew}} < T \tag{2.1}$$

where $t_{\text{clk-q}}$ is the time between the occurrence of a clock edge and the availability of the latched value on FF1's output (this is referred to as the clock-to-q delay of the flip-flop), t_{pd} is the propagation delay of the path $FF1 \rightarrow FF2$ (the sum of all gate and interconnect delays on this path), t_{su} is the setup time of $FF2$, t_{skew} is the timing uncertainty in the arrival of clock edges at $FF1$ and $FF2$ and T is the clock period.

Hold Time

Although state bit transitions must arrive by at least t_{su} seconds before the sampling clock edge, they also cannot arrive too early. This is because a flip-flop's input must be held stable for a certain time t_{h} after the sampling clock edge. If state bit transitions travel very fast, they might arrive within t_{h} seconds of the clock edge and violate this requirement. To prevent this from happening, the delay of each combinational path between any two flip-flops must be large enough to satisfy the following condition:

$$t_{\text{clk-q}} + t_{\text{pd}} - t_{\text{skew}} > t_{\text{h}} \tag{2.2}$$

where t_{h} is the hold time of the destination flip-flop.

This is referred to as the hold time condition.

2.2 Metastability

2.2.1 Introduction

The purpose of enforcing the setup and hold time conditions on combinational paths is to constrain the input of every flip-flop: to ensure that it is held stable for at least t_{su} seconds before the clock edge and that it remains stable for no less than t_{h} seconds afterwards. By doing so, flip-flop outputs are guaranteed to behave in a predetermined

manner: they transition to the logic level of the input monotonically, with a nominal transition time and within a nominal clock-to-q delay. These properties are essential for the design of deterministic synchronous systems.

In some applications, however, the setup and hold times of a flip-flop's input cannot be always satisfied. For example, when the flip-flop is used to sample a real-time signal, input transitions can occur at any time relative to the clock edge. For a clock edge occurring at t_{clk}, if a transition occurs after $t_{\text{clk}} - t_{\text{su}}$ and before $t_{\text{clk}} + t_{\text{h}}$ (this interval is referred to as the setup-hold time window), the flip-flop may not behave in the predetermined manner described above. In other words, it may transition or not transition at all, it may transition after a long delay with a longer rise/fall time or it may produce multiple output transitions (behave non-monotonically).

Historically, flip-flops were not known to behave in this manner in the early days following their invention. It was believed that a flip-flop whose setup and hold time conditions were violated will either succeed or fail to capture the logic value of the input. The impact of these violations on the delay, transition time and monotonicity of the flip-flop output had not been foreseen. In consequence, multiple early synchronous computers which have included unconstrained flip-flops exhibited mysterious failures whose root cause was not identified until the first mathematical analysis of the problem was published in 1952 [4]. The anomalous behavior of unconstrained flip-flops was attributed to *metastability*: a pseudo-stable state in which a bistable element is neither logic high nor low but somewhere in between.

2.2.2 Historical References

An insight into this phenomenon was given when the sampling of a real-time signal was recognized to be a *decision process*. Specifically; one that involves mapping an analog quantity (the arrival time of a transition relative to the clock edge) into a discrete domain (logic high or low). Decisions of this nature have long been known to be vulnerable to indecisiveness hazards. An early example of these hazards appears in Aristotle's "On The Heavens" where a man, equally hungry and thirsty, dies as a consequence of not being able to choose whether to eat or drink first [5]. A similar example was given by Jean Buridan, a French philosopher, involving an ass that is placed at equal distances from

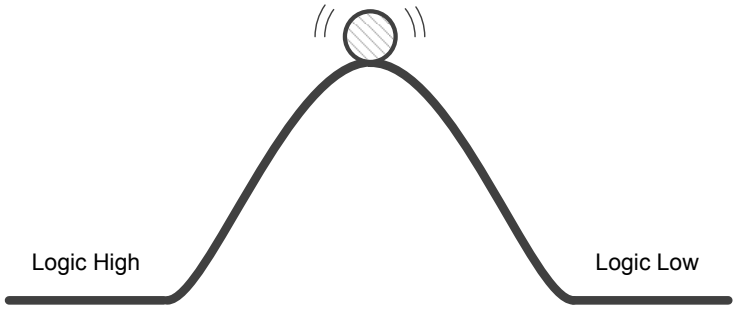


Figure 2.3: Ball and hill analogy of a bistable system

two hay stacks and is unable to decide between the two (Buridan's ass problem postdates Aristotle's but is more often quoted in metastability literature). A much older reference to the problem exists in the "The Incoherence of the Philosophers" by Al-Ghazali who argues, curiously, that an agent is able to choose between two identical courses of action by virtue of "the will" [6].

2.2.3 Problem Fundamentals

In all the scenarios pictured by the early philosophers, an agent takes a long time to arbitrate between two options based on an analog quantity (e.g. the desire for food/water or the distance to a hay stack). This is similar to a ball that is carefully placed at the top of a hill in a momentarily-stable position (Figure 2.3). The ball will take a longer time to escape this pseudo-equilibrium and roll to either side of the hill compared to another ball which is placed further from the top. In fact, the closer the ball is to the top, the longer it will take it to roll to either side.

If the ball and hill system were part of a larger mechanical system whose overall functionality depends on the ball reaching the bottom within a specified period of time, prolonged rolling can cause a "failure" of the parent system. Furthermore, if the rolling experiment was repeated a sufficient number of times with the initial ball position selected at random, failures of this sort are bound to occur.

Can the system be engineered in a such a way to prevent these failures? A number of ideas might rush to the mind of the unwary. For instance, if the hill was made asymmetric by increasing the slope of one of its sides relative to the other, would this

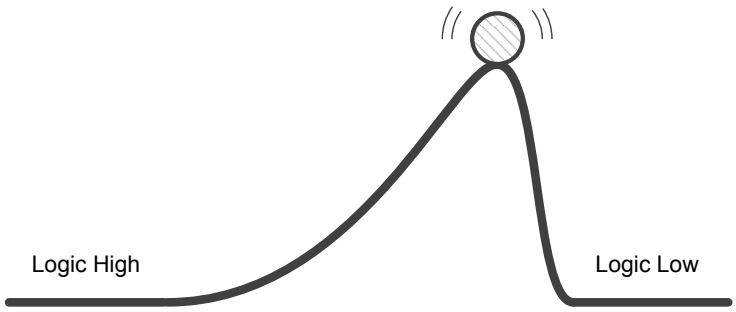


Figure 2.4: Biased bistable system

force a metastable ball to roll in that direction? The short answer is no: attempts at eliminating metastability by biasing the choice process are futile, they only result in the relocation of the tipping point but do not eliminate it (this is illustrated in Figure 2.4). What about adding a "detector" to signal when the ball has finally reached the bottom of the hill? Such a detector would incorporate its own decision process which can be modeled as a second ball-and-hill system (and hence be susceptible to the same problem on its own). A third intuitively-appealing solution might be to add a random source of perturbations that would push a metastable ball into freedom. Alas, such a source would also just as likely push a ball placed slightly away from the top into metastability. The non-existence of a solution for this situation is not due to any lack of ingenuity. Any system that attempts to map an analog quantity into a discrete domain in a finite amount of time is bound to experience failures. Flip-flops that have input signal arrival times that are not constrained to satisfy its setup and hold time constraints belong to this category: they attempt to map the arrival time of a data signal transition (an analog quantity) into a logical value (true or false). They are also required to do so in finite time (within a nominal clock-to-q delay) because their output transitions must have sufficient time to propagate to the following flip-flops. In consequence, applications that involve unconstrained flip-flops (e.g. real-time sampling and asynchronous communication) can not be completely guarded against flip-flop indecisiveness problems and must tolerate a finite probability of failure. Before this fact was carved in stone, there have been various attempts by digital designers to create metastability-free components or filtering circuits which are meant to eliminate the problem [7]. A survey of these attempts is provided

in [8]. All these claimed solutions were later shown to have only moved either the tipping point or the resulting failures to other parts of the system by means which have eluded the designers [8] [9].

2.2.4 Metastability in Latches

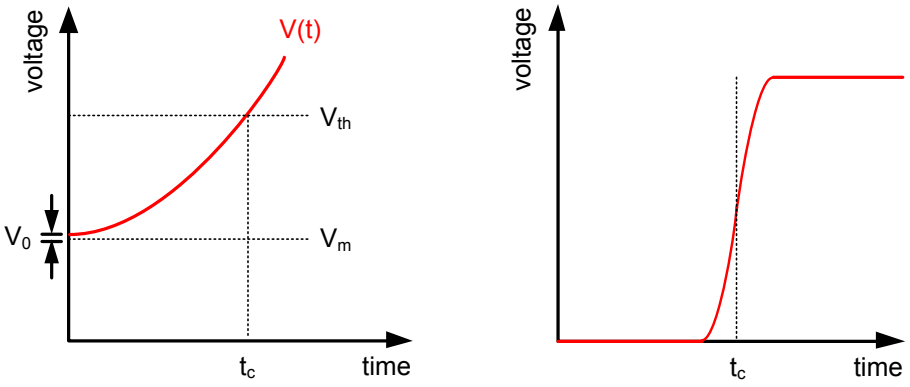
When a latch becomes transparent, its input buffers start pulling it towards the logic state of the input. This process stops when the input buffers are disabled and the latch becomes opaque. If this moment coincides with the transition of the input, the latch may not be pulled strongly to either logic state and the outputs of the cross-coupled inverting gates may get stuck at non-rail voltages for an arbitrary amount of time before they diverge (one to VDD and the other to ground). During this process, the output voltage $V(t)$ of each inverting gate can be expressed as [10] [11]:

$$V(t) = V_0 \times e^{t/\tau} \tag{2.3}$$

where V_0 is the initial output voltage of the inverting gate and τ is the metastability regeneration time constant. Both V and V_0 are expressed relative to a hypothetical voltage V_{m} at which the inverting gates will be perfectly-metastable state and take an infinite amount of time to resolve to a stable state.

In essence, the positive feedback loop at the core of every latch is analogous to the ball and hill system: it has two stable states and a metastable state somewhere in between. When the latch enable signal is de-asserted, the input buffers of the latch are disabled and the metastable node voltages (considered symmetric for the sake of simplicity) are set to V_0 volts. This corresponds to placing the ball on some position on the hill. The loop then amplifies this voltage exponentially to either rail level of the supply voltage (VDD or ground) in the same manner that the ball rolls to either side of the hill. Eventually, the voltage $V(t)$ crosses the switching threshold voltage V_{th} of the latch's output amplifier and a transition appears at the latch output [1] (Figure 2.5). This marks the end of the decision process similar to the ball crossing a finish line at the bottom of the hill.

[1] assuming that the newly latched logic state is different from the previous one



(a) Metastable node voltage

(b) Flip-flop output

Figure 2.5: Metastable voltage regeneration and delayed output transition

If V_{th} is expressed relative to V_{m}, the decision time t_{c} (the time it takes V to cross V_{th}) can be expressed using Equation 2.3 as the following:

$$t_{\text{c}} = \tau \times \ln(\frac{V_{\text{th}}}{V_0}) \tag{2.4}$$

From Equation 2.4, it can be seen that the decision time t_{c} increases linearly as the initial voltage V_0 approaches zero exponentially. The speed of this process depends largely on the value of τ which can be approximated as the inverse of the gain-bandwidth product of the cross-coupled inverting gates. The time constant τ is a characteristic feature of the metastability resolution performance of latches and is a function of both the latch design and its operating conditions (e.g. supply voltage and temperature).

Equation 2.4 also illustrates how the setup and hold time conditions prevent a latch from becoming metastable. Constraining the arrival time of input transitions, in effect, ensures that V_0 is always sufficiently-large such that it can be regenerated to V_{th}within a predetermined duration. This is similar to constraining the initial position of the ball in the ball and hill analogy. By ensuring that the initial position of the ball is no less than a certain minimum from the hill top, the ball is guaranteed to reach the finish line at the hill's bottom within a certain time period (Figure 2.6).

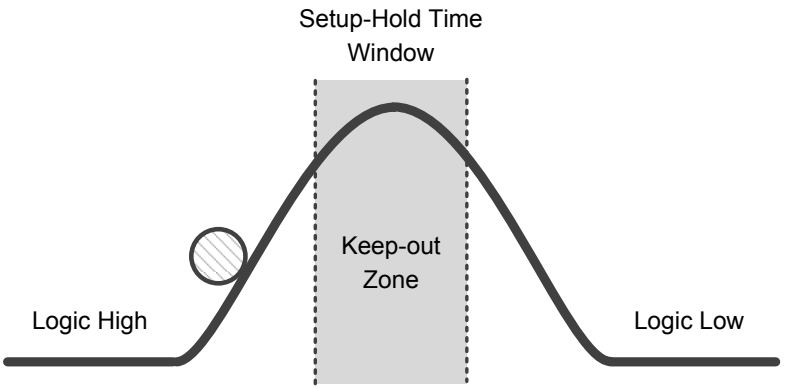


Figure 2.6: Setup and hold time conditions analogy

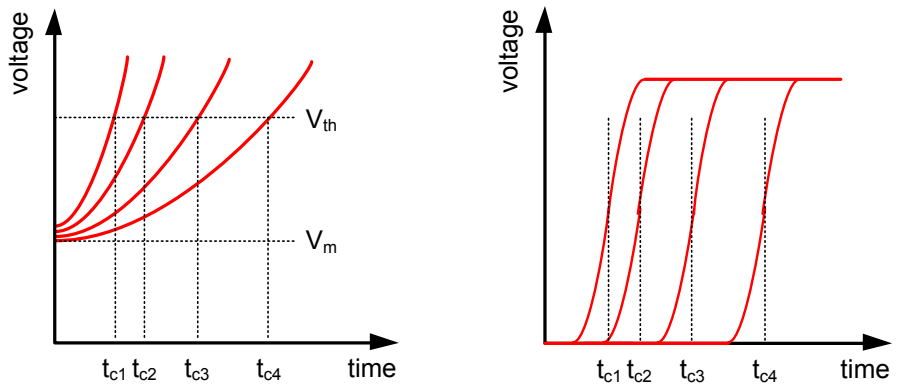
2.3 Metastable Flip-Flop Behavior

While metastability is primarily associated with long output delays, metastable flip-flops are also known to exhibit longer transition times, behave non-deterministically, non-monotonically or even oscillate. This section will examine these abnormal behaviors and the hazards they pose starting by taking a deeper look at how failures occur when flip-flop outputs take longer than expected to transition.

2.3.1 Prolonged clock-to-q Delay

The clock-to-q delay of a flip-flop is the sum of the decision time t_c of the master latch and the propagation delay of the slave latch during transparency. When the master latch becomes metastable, its prolonged decision time (Equation 2.4) can cause an increase in the clock-to-q delay of the flip-flop [12] [13]. This is illustrated in Figure 2.7.

Synchronous logic is designed such that state bit transitions have sufficient time to propagate to subsequent flip-flops by the time of the following clock edge. If one flip-flop k becomes metastable and produces a transition whose clock-to-q delay is longer than expected, this transition may not have sufficient time to reach all destination flip-flops. A subset of the destination flip-flops may capture the new (post-transition) value of k while others capture the old (pre-transition) value. This is essentially a *misinterpretation error*: the logic state of k is interpreted differently by different destination flip-flops



(a) Metastable node voltage

(b) Flip-flop output

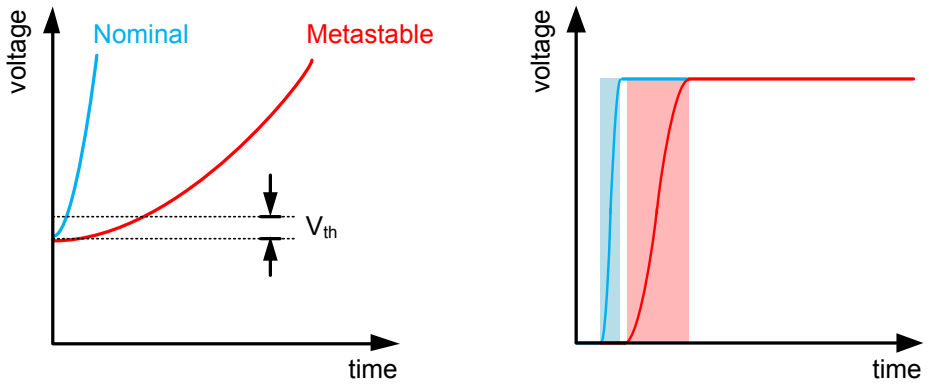
Figure 2.7: Prolonged clock-to-q delays

causing the following registers to latch incorrect (corrupt) values. The consequences of these events vary considerably depending on which registers are affected and how the system is structured. If the affected registers include some which hold important state information (such as program counters) then the system may transition into an unknown and possibly-unrecoverable state.

Prolonged clock-to-q delays also enable metastability to propagation from one flip-flop to another. If k's delayed transition arrives within the setup-hold time window of one of the destination flip-flops w, the latter may become metastable in the following cycle and exhibit a prolonged clock-to-q delay on its own. The metastable w may then induce a metastable state in one of its destination flip-flops z. This sequence can continue indefinitely but will happen at an exponentially diminishing probability for each subsequent flip-flop in the propagation chain.

2.3.2 Prolonged Transition Time

Transitions appear at the output of a metastable latch when the metastable node voltage $V(t)$ crosses the threshold voltage V_{th} of the output amplifier. If V_{th} is very close to V_{m} (the hypothetical metastable node voltage) the output amplifier may take longer than usual to transition [12] [13]. This is because amplifiers are not ideal devices and do not have a single threshold voltage point in practice. Instead, output amplifiers map a small range of input voltages (typically few millivolts for modern processes) into rail-



(a) Metastable node voltage

(b) Flip-flop output

Figure 2.8: Prolonged transition time

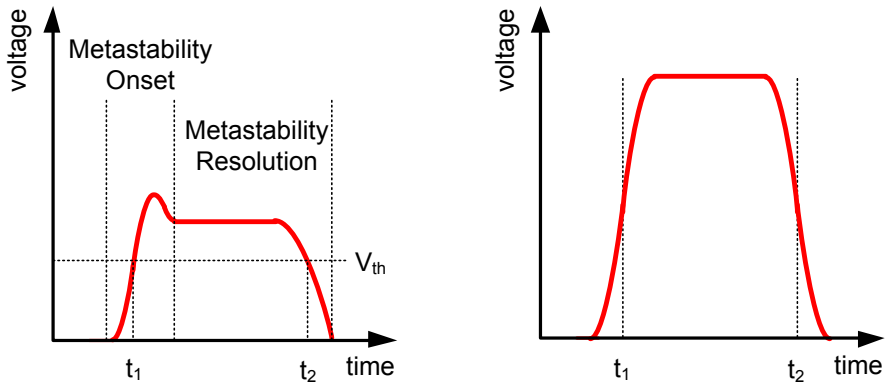
to-rail output voltages. If V_m is in the close vicinity of output amplifier's threshold range and the latch becomes metastable, the slow regeneration of the metastable node voltage (away from V_m) may cause a slow crossing of the amplifier's input voltage range and hence a slow output transition. This effect is illustrated in Figure 2.8.

The transition (rise and fall) times of flip-flops are pre-characterized and taken into consideration when calculating path propagation delays and designing synchronous systems. Slow transitions can violate timing constraints in the same manner that longer propagation delays do and so they can induce misinterpretation errors or cause subsequent flip-flops to become metastable.

Although it is impossible to prevent metastability from inducing prolonged clock-to-q delays, prolonged transition times can be prevented by elaborate circuit design. If the output amplifier is designed such that its threshold voltage V_{th} is well above (or below) V_m, output transitions will not occur unless the metastable node voltage has diverged sufficiently away from V_m. Thus, an upper bound can be placed on the time it takes the metastable node voltage $V(t)$ to cross the input voltage range of the amplifier. In consequence, the transition time of the flip-flop output can also be upper-bounded.

2.3.3 Non-monotonic Output Transitions

Flip-flops transition when the metastable node voltage crosses V_{th}. If the setup and hold time conditions are met, the metastable nodes of the master latch will be pulled strongly



(a) Metastable node voltage

(b) Flip-flop output

Figure 2.9: Runt pulse formation due to the onset and resolution of metastability

resulting in a single crossing of V_{th} and a single transition appearing at the flip-flop's output after a nominal clock-to-q delay [2]. However, if the metastable nodes were not pulled strongly and the master latch becomes metastable, multiple crossings of V_{th} might take place. The different mechanisms by which this can happen are described below, sorted by the likelihood of their occurrence in practice.

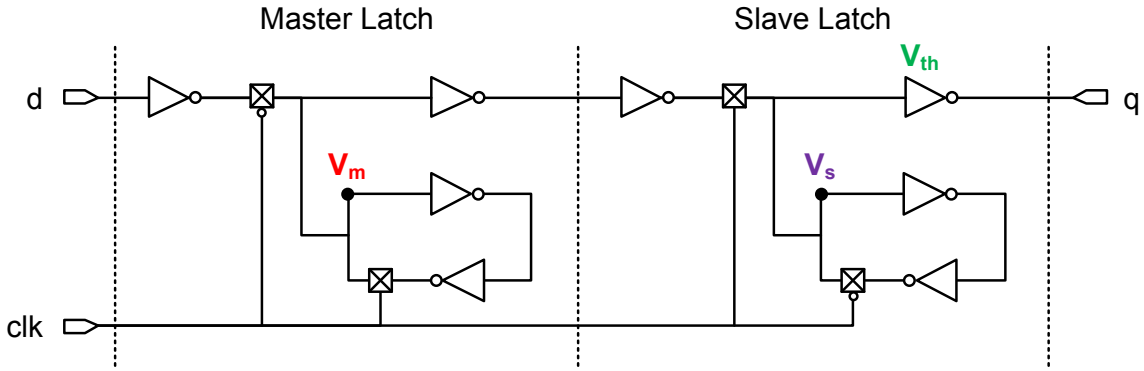
Metastability Onset and Resolution

If the initial transient that the flip-flop experiences before falling into metastability causes a crossing of V_{th} and then the metastable node diverges back to the previous state, a runt pulse will appear at the flip-flop's output as illustrated in Figure 2.9. This effect cannot be avoided by adjusting V_{th} since the metastable node voltage may fall into metastability either from VDD or ground. However, it is still possible to mitigate this non-monotonicity by avoiding sampling the flip-flop output too early. This is because, unlike the second transition, the first transition can be time-bounded.

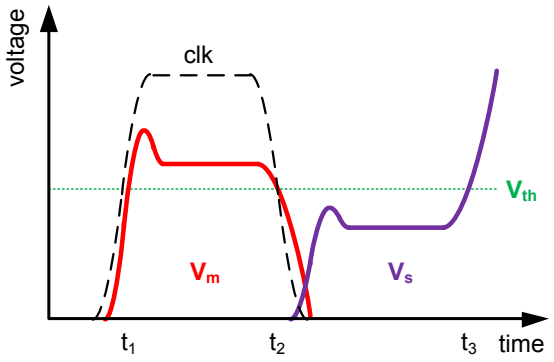
Asymmetry of Master and Slave Latches

Metastability can propagate from the master to the slave latches if the master latch resolves metastability very close to the non-active clock edge (the falling edge in a positive-edge-triggered flip-flop). If the master and slave latches have V_m values

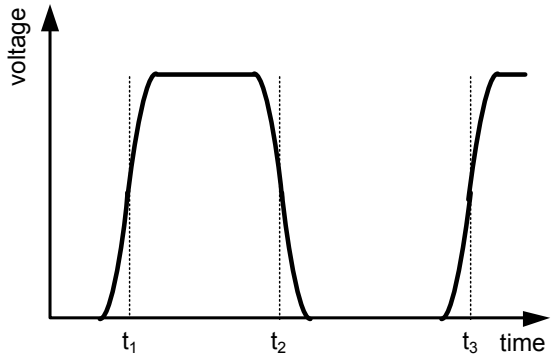
[2] assuming that the newly-latched state is different from the flip-flop's previous state



(a) Flip-flop schematics



(b) Metastable node voltages



(c) Flip-flop output

Figure 2.10: Metastability propagation and multiple flip-flop output transitions

that are on opposite sides of V_{th}, multiple output transitions may occur as metastability propagates from the master to the slave latches and then resolves [12]. An example of this behavior is demonstrated in Figure 2.10. Here, the master latch becomes metastable following the rising edge of the clock and drives the flip-flop output high at t_1. The master latch then resolves, driving the flip-flop output low at t_2 and bringing the slave latch into metastability. Some time later at t_3, the slave latch resolves and drives the flip-flop output logic high again. This form on non-monotonicity can be avoided by ensuring that V_{th} is either higher or lower than both master and slave V_{m} voltages.

Noise

A fourth (but much less likely) form of multiple output transitions may occur when V_{th} is very close to V_{m} and noise perturbs the metastable node voltage causing multiple

crossings of V_{th}. This effect can be mitigated by making V_{th} sufficiently larger or smaller than V_m such that an output transition does not occur unless the metastable nodes have diverged sufficiently away from V_m. Common library latches in modern processes cannot be pushed back into metastability once they have diverged by few millivolts [12]. Therefore, a difference of few millivolts between V_{th} and V_m is sufficient to prevent noise from inducing multiple transitions at the latch output.

Oscillation

If the transient pulse that proceeds the onset of metastability is shorter than the loop delay of the cross-coupled gates, the metastable node voltages may oscillate [14] [15] [16]. This is because the pulse will travel the loop and appear at the output periodically until the latch resolves to a stable state. Oscillation has been reported in older technologies such as TTL [16] [17] but is not typically observable in CMOS because common latches in modern processes have very short loop propagation delay.

2.3.4 Non-determinism

In Section 2.2.3 it was asserted that noise sources do not prevent a bistable system from becoming metastable. This is because any perturbation which may push the system away from the metastable point is also equally likely to push the system towards it. However, noise does determine which stable state that the system will settle to after escaping metastability. If noise was inherently stochastic then the final state of the metastable system will be non-deterministic.

In the case of flip-flops, non-deterministic supply voltage fluctuations and dynamic variability sources such as thermal noise and wire crosstalk perturb the value of V_0. Therefore, if a flip-flop enters a deep enough metastable state (i.e. one where V_0 is sufficiently small), its output will be non-deterministic.

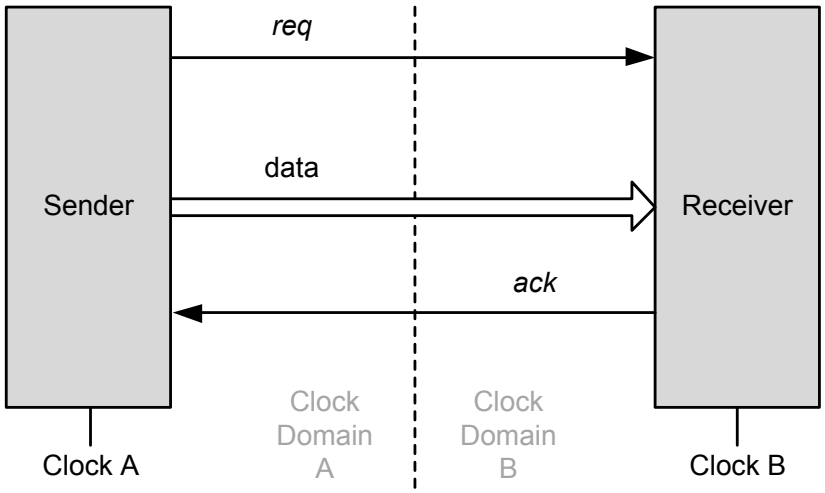


Figure 2.11: Clock domain crossing

2.4 Metastability in SoCs

Metastability is encountered in several areas in modern System-on-Chip (SoC) design. The behavior of metastable flip-flops is most often considered problematic and so different solutions have been proposed to mitigate its impact in the respective fields. There are also a few applications in which metastability is deliberately induced and exploited to perform a useful function. This section provides an overview of all these areas and describes its distinguishing aspects of metastability in each.

2.4.1 Clock Domain Crossing

Metastability failures were first noted in computers with multiple clock domains [17]. The passage of data between clock domains is referred to as clock domain crossing and is perhaps the application that is currently most associated with metastability failures.

Components that run in different clock domains do not share a common time reference and must communicate via handshakes [18] as illustrated in Figure 2.11. Typically, the two communicating entities, the sender and the receiver, coordinate the passage of data across their clock domain boundary using two signals *req* and *ack*. The sender makes data available on the data bus and asserts *req* and the receiver latches

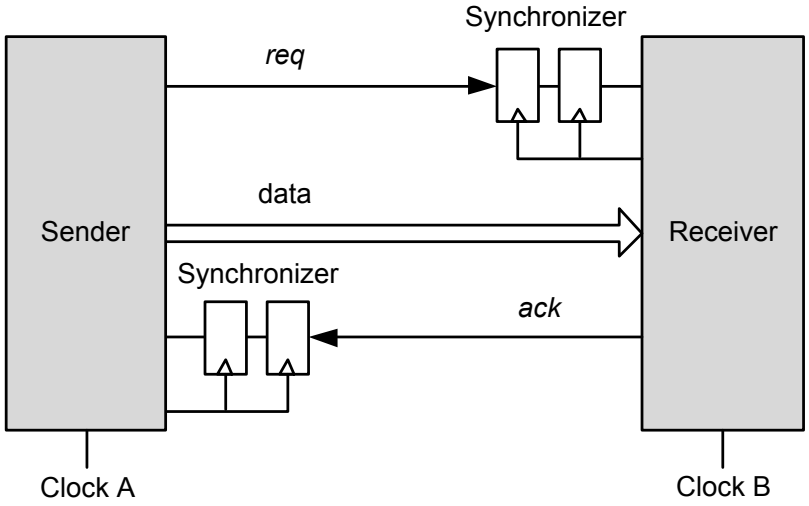


Figure 2.12: Synchronizers

the data and replies by asserting *ack*. The sender then de-asserts *req* and the receiver follows by de-asserting *ack*. This is referred to as a four-phase handshake. In a faster variation of this protocol, a two-phase handshake, the sender signals the availability of data by toggling *req* and the receiver acknowledges its consumption by toggling *ack*. Using two individual signals to communicate requests and acknowledgments is the dominant method of handshaking but is not the only one. The request signal can also be embedded in the data itself by using special encoding schemes such as dual-rail or one-hot encoding. Also, while handshaking has been described as a method to coordinate the transfer of data across clock domains, it can also be used to trigger events that do not involve the transfer of data.

In all the different handshaking forms described above, the *req* and *ack* signals (either implemented individually or decoded from data) are not synchronized to the clocks of their recipients. Therefore, these signals may transition at any time relative to the clock edge of their destination flip-flops. If a transition occurs within the setup and hold time window of these flip-flops, the latter may become metastable and induce catastrophic failures in their parent sub-systems. To guard against these events, a chain of flip-flops (typically two) is used on each end of the communication channel to allow any metastable states to resolve safely without inducing failures in the remaining part of the system. These flip-flop chains are known as synchronizers (Figure 2.12).

Synchronizers do not prevent metastability from occurring. As noted in Subsection 2.2.3, metastability can propagate from one flip-flop to another and this is fundamentally impossible to prevent. Instead, synchronizers reduce the probability of these failures to an acceptable level. The probability of a synchronization failure can be expressed as the probability of the metastable node voltage V not reaching the output amplifier transition threshold V_{th} in an allocated resolution time t_s or:

$$P[\text{failure}] = P[V(t_s) < V_{\text{th}}] \tag{2.5}$$

Given Equation 2.3, Equation 2.5 can be re-written as:

$$P[\text{failure}] = P[V_0 < V_{\text{th}} \times e^{-t_s/\tau}] \tag{2.6}$$

In other words, synchronization will fail if the value of the initial metastable node voltage V_0 is smaller than the voltage window ($V_{\text{th}} \times e^{-t_s/\tau}$). This is because, for V_0 values within this window, metastability resolution will exceed the allocated time t_s. For very small V_0 values, and given that the dynamics of latch circuits are linear near the metastable point [19], V_0 can be considered linearly proportional to the input transition time t_{in} relative to the clock edge. This relationship can be expressed as:

$$V_0 = k \times t_{\text{in}} \tag{2.7}$$

Given the linear relationship above, it is possible to map the voltage window $V_{\text{th}} \times e^{-t_s/\tau}$ into a corresponding input transition arrival time window $T_w \times e^{-t_s/\tau}$ where $T_w = V_{\text{th}} \times k$. This mapping is illustrated in Figure 2.13. Synchronization failures can now be expressed as the probability of t_{in} falling within this window or:

$$P[\text{failure}] = P[t_{\text{in}} < T_w \times e^{-t_s/\tau}] \tag{2.8}$$

Put differently, if a transition arrives very close to the clock edge such that t_{in} is extremely small (smaller than the time window $T_w \times e^{-t_s/\tau}$) then the induced V_0 will be smaller than the voltage window $V_{\text{th}} \times e^{-t_s/\tau}$ and metastability resolution will take longer than the allocated time t_s.

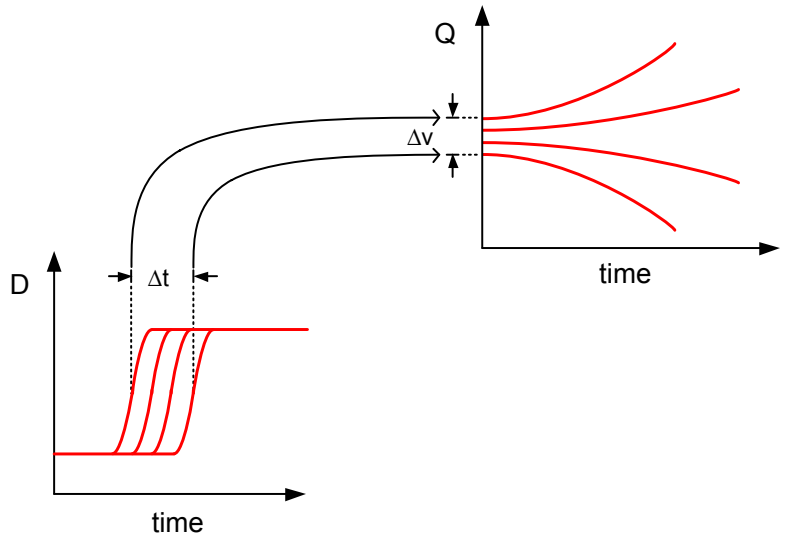


Figure 2.13: Linear mapping between Δt_{in} and ΔV_0

If t_{in} is evenly distributed across the clock period T, then:

$$P[\text{failure}] = \frac{T_w \times e^{-t_s/\tau}}{T} \tag{2.9}$$

Now, if $T = 1/f_c$ and the rate at which synchronization is performed is equal to the asynchronous data arrival rate f_d, synchronization failure rate can be expressed as:

$$\text{Failure Rate} = f_d \times P[\text{failure}] = f_d \times f_c \times T_w \times e^{-t_s/\tau} \tag{2.10}$$

It is more common to express synchronization failure rate in terms of the Mean Time between Failures (MTBF). Therefore:

$$\text{MTBF} = \frac{1}{\text{Failure Rate}} = \frac{e^{t_s/\tau}}{f_d \times f_c \times T_w} \tag{2.11}$$

Although Equation 2.11 is derived from the small-signal model of a single latch, it is also used to characterize multi flip-flop synchronizers by taking t_s as the sum of the resolution time provided by all flip-flops in the chain [3]. It is possible to connect flip-flops in a number of configurations to obtain different resolution times (typically of integer multiples of half the clock period) as shown in Figure 2.14. The resolution

[3] an error analysis of this approximation is presented in [12]

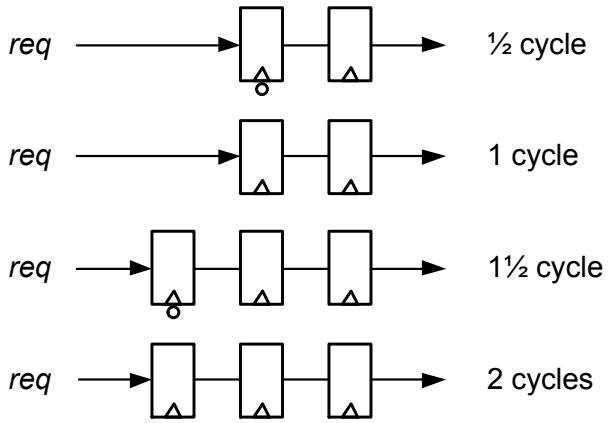
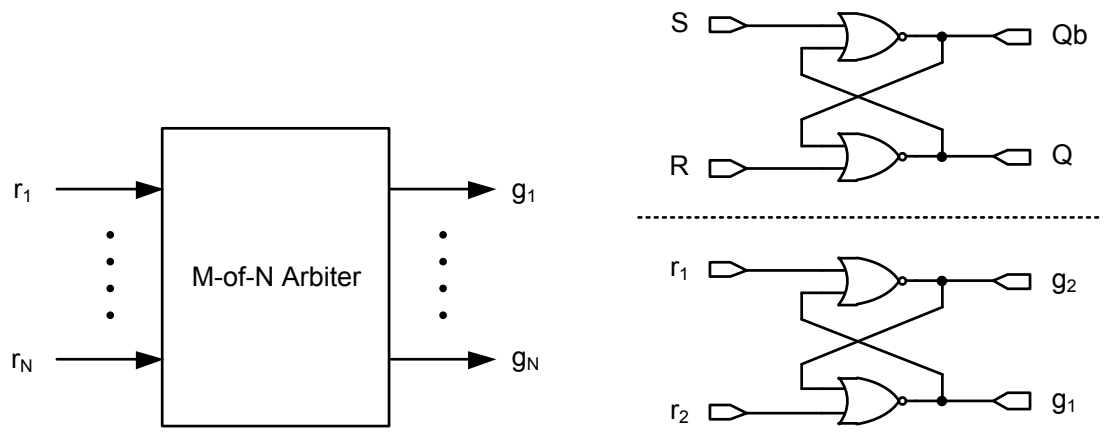


Figure 2.14: Synchronizer chains of different latencies

time t_s represents the only design choice in Equation 2.11 (f_c and f_d are system-specific parameters while τ and T_w are latch-specific).

Increasing the allocated metastability resolution time t_s increases the MTBF of synchronization exponentially but also increases the latency of processing the synchronized signal. Therefore, synchronizer design presents a reliability versus latency trade-off. A two flip-flop synchronizer provides a metastability resolution time of one clock period which is often sufficient to maintain a MTBF in the order of thousands of years in modern technologies. For example, taking $f_c = f_d = 1$ Ghz, $T_w = 1$ns and a typical τ value of 20ps, a two flip-flop synchronizer will provide a MTBF in excess of 100 thousand years. This might seem like a conservative figure but it is not. The reason is that the MTBF is exponentially dependent on τ which, in turn, is a function of the latch design and its operating conditions. Under non-nominal operating conditions (higher temperature or lower supply voltage), small variations in τ might induce order-of-magnitude changes in synchronization MTBF. In the example above, if τ increases by 50%, the MTBF will drop to less than 4 days. Designs that operate under extreme conditions or high process variability might require up to four flip-flops to perform reliable synchronization [20].



(a) M-of-N Arbiter

(b) SR Latch (top) and 1-of-2 Arbiter (bottom)

Figure 2.15: Asynchronous arbiters

2.4.2 Arbitration and Resource Allocation

Another area in which metastability is encountered is the design of circuits that regulate the access to shared resources by multiple clients (i.e. arbiters). Controlling access to shared resources is necessary to prevent multiple clients from attempting to access the same resource at the same time and induce errors in the process. For example, if two processors attempt to write to a single-port memory simultaneously, the data and address bits may become corrupt and an unknown word may be written into an unknown memory location. To prevent these access collisions, arbiters are employed as an intermediary between clients and the shared resource(s). Arbiters receive access requests from multiple clients and grant access to one at a time. When the client whose request has been granted finishes accessing the resource, it de-asserts its request and the arbiter may then allocate the resource to another client. The schematic of an M-of-N arbiter (an arbiter which regulates access to M resources by N clients) is shown in Figure 2.15a.

Arbiters can operate synchronously or asynchronously. Asynchronous arbiters coordinate access between clients with different time references and must handle the arrival of requests at any moment. When a 1-of-2 asynchronous arbiter receives two requests at nearly the same time, it may become metastable and take a long time to issue

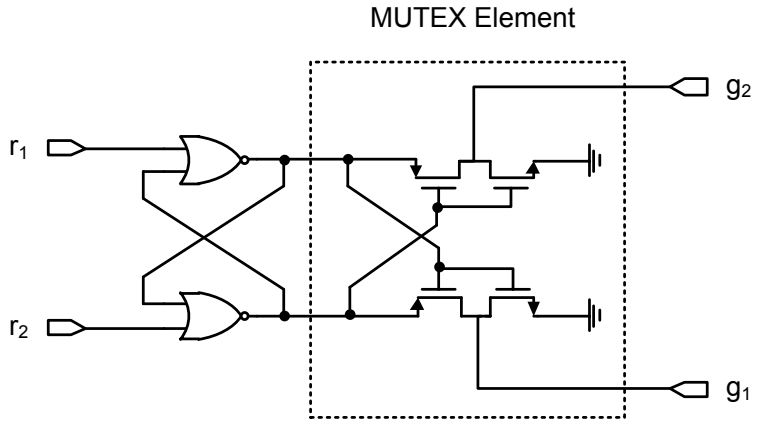


Figure 2.16: An arbiter with a MUTEX element

a grant. This is not an unexpected property because a 1-of-2 asynchronous arbiter is functionally-equivalent to an SR (Set-Reset) latch [21]. This equivalence is demonstrated in Figure 2.15b. Since all latches (including SR variants) have a metastable point, the same applies to a 1-of-2 asynchronous arbiter. In fact, all asynchronous arbiters have one or multiple metastable points and cannot arbitrate in a finite amount of time with zero probability of failure [22].

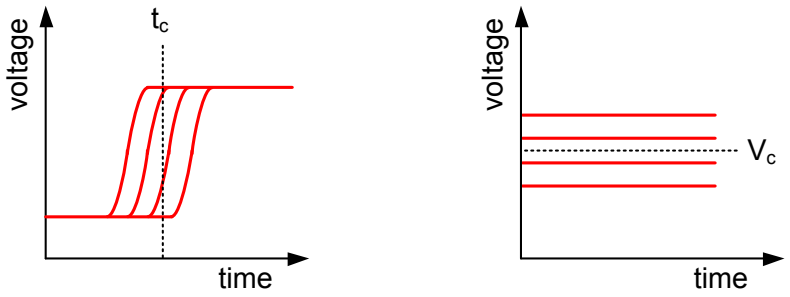
What happens internally inside a metastable arbiter is no different from what happens in a metastable flip-flop; a pair of cross-coupled gates inside the arbiter are brought to intermediate non-rail voltages and take additional time to resolve to a stable state. During the process, the arbiter may exhibit any of the problematic behaviors described in Section 2.3: it may take additional time to grant requests, output transitions with lower slew rate or behave non-monotonically. Asynchronous circuits operate under relaxed timing constraints and so prolonged grant times do not pose a reliability issue for asynchronous arbiters (nor do they affect performance because metastable states occur relatively rarely). However, the same cannot be said about non-monotonic output responses. If the arbiter produces multiple output transitions, multiple clients might be granted access to the same resource at the same time and a failure might occur. More complex arbiters which serve more than two clients may also exhibit other forms of failure such not granting any requests, violating the implemented priority scheme [23] or producing oscillating outputs [24]. To mitigate this, arbiter circuits are designed

to *contain* metastable states, i.e. to ensure that their outputs do not transition until any metastable state is resolved internally. This does not violate any of the principal tenets of metastability because the choice delay remains boundless. Metastable states are contained using a *mutex element* [21] [25] such as the one shown in Figure 2.16. The mutex element will filter any non-monotonic behavior that may occur during the onset and resolution of metastability because its outputs will not transition unless the metastable node voltages have diverged sufficiently away from each other.

2.4.3 Analog-to-Digital Conversion

Analog-to-Digital Converters (ADCs) play an integral part in interfacing mixed-signal blocks and can be considered the synchronizers of the analog-to-digital boundary. ADCs attempt to map analog voltages into a discrete domain in a finite sampling time and so they have a metastable region of operation similar to synchronizers and arbiters [26]. However, unlike synchronizers and arbiters, ADCs become metastable because of the unconstrained input voltage and not its transition time. This difference is illustrated in Figure 2.17. Synchronizers and arbiters enter metastability when their inputs transition very close to a hypothetical transition time t_c (this is near the clock edge for synchronizers and near the transition time of another request for an arbiter). An input transition occurring at t_c exactly will bring the synchronizer or arbiter into a hypothetical perfectly-metastable state that will take an infinite amount of time to resolve ($V_0 = 0$). On the other hand, a latch which is used as a one-bit flash ADC enters metastability when its input voltage level V_{inp} is very close to a hypothetical voltage V_c. The latch will enter the hypothetical perfectly-metastable state when $V_{\text{inp}} = Vc$.

Conventionally, ADC metastability is characterized by the Bit Error Rate (BER) of the digitizing latch. Similar to the case of time-induced metastability, the linear behavior near the metastable point permits a linear mapping between V_{inp} and V_0. Hence, it is possible to calculate a window of input voltages ΔV_{inp} that will map to the window of metastable node voltages ΔV_0 in which the latch will take longer than an allocated time



(a) Transition time (b) Voltage level

Figure 2.17: Different forms of input variations that cause metastability

t_s to produce an output transition. The size of the window ΔV_{inp} can be expressed as:

$$\Delta V_{\text{inp}} = \frac{V_{\text{th}}}{A} \times e^{-t_s/\tau} \tag{2.12}$$

where A is gain of all the preamplifier stages between the input voltage source and the metastable nodes (including the latch input buffer).

If V_{inp} is uniformly distributed across the range [0, VDD], then the BER of the latch can be expressed as [27] [28] [29]:

$$\text{BER} = \frac{\Delta V_{\text{inp}}}{\text{VDD}} = \frac{V_{\text{th}}}{A \times \text{VDD}} \times e^{-t_s/\tau} \tag{2.13}$$

Time-induced and voltage-induced metastable states are not different phenomena. What is different is just the way metastability is induced (the input buffers' current waveforms that pull the metastable nodes half-way between VDD and ground). However, the two are often considered independently because of two contextual differences [10]. First, while reliability is the key concern in the design of metastability-hardened synchronizers and arbiters, the equivalent in the ADC world is the magnitude of digitization error, characterized by the Signal-to-Noise Ratio (SNR). Existing work in literature reports that metastability bit flips have surpassed quantization error and are now the upper bound of the SNR of ADCs with high sampling rates [30] [31] [32]. The relationship between metastability BER and the SNR is not easy to establish and depends significantly on the topology of the ADC [33] [34]. For example, a single flip in the Least Significant

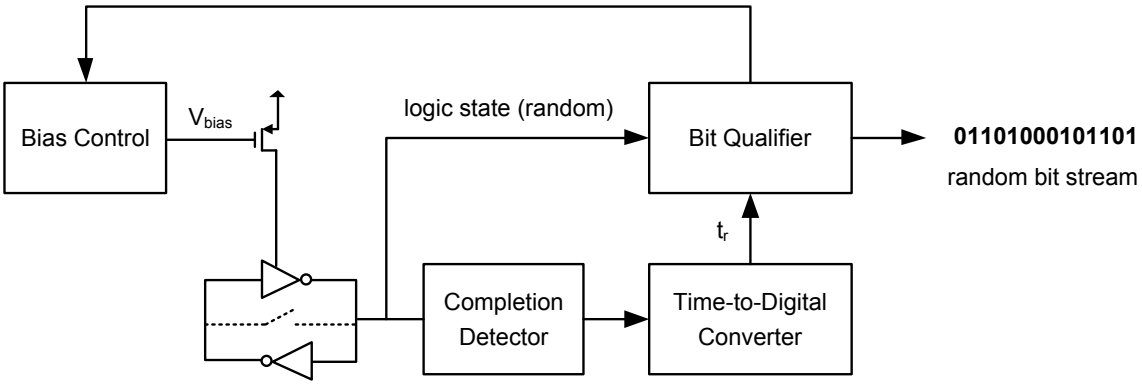


Figure 2.18: Metastability-based TRNG (based on [1])

Bit (LSB) of a flash ADC has a small impact on the output word while a bit flip in a successive approximation ADC will cause the search routine to diverge. To capture the impact of metastability on the SNR, the Signal-to-Metastability-Noise Ratio (SMNR) has been proposed and analyzed for several ADC topologies [26] [30] [29]. Second, the rate of entering metastability in ADCs can be lowered by investing more power in pre-amplification. This option is not available in synchronization because the rate of entering metastability (represented by the expression $T_w \times f_c \times f_d$) is fixed. Therefore, ADCs can trade power for metastability error rates in a fixed supply voltage environment but synchronizers and arbiters cannot.

2.4.4 Random Number Generation

Random number generators are used in several applications including cryptography, statistics and simulations. The random requirements of some of these applications can be met by pseudo-random bit sequences created by Pseudo Random Number Generators (PRNGs) such as linear feedback shift registers. Other applications (particularly cryptography) rely critically on the random quality of the generated data. The latter class of applications require True Random Number Generators (TRNGs) which harvest the inherent randomness of non-deterministic physical sources such as thermal noise and wire crosstalk. The non-deterministic behavior of metastable latches has been recognized as a mean of tapping into these physical phenomena and has thus been at the heart of several novel TRNG designs [35] [36] [37].

Table 2.1: Duality between two metastability applications

Application	**Determinisim**	**Non-determinism**
Random Number Generation	Vulnerability (*undesired*)	Entropy (*desired*)
Physical Parameter Sensing	Sensitivity (*desired*)	Noise (*undesired*)

A metastability-based TRNG from [1] is shown in Figure 2.18. The generator extracts random bits from the final state of a latch that is consistently brought into metastability. To ensure that the output bit stream is of good random quality, only the bits that are generated when the latch becomes deeply metastable are used. The deepness of the induced metastable states is assessed by quantifying the time t_r it takes the latch to resolve metastability (this is done using a completion detector, i.e. a mutex element, and a time-to-digital converter). If t_r is larger than a certain minimum, the final state of the latch is appended to the output bit stream. Otherwise, the induced metastable state in considered not deep enough and the bit is discarded. Filtering out the bits that result from non-deep metastable states improves the randomness of the output bit stream by making the ratio of 1's to 0's closer to 1:1. The generator also becomes more resilient to malicious attacks that attempt to bias the latch [1].

To guarantee a consistent rate of deep metastable states, metastability-based TRNGs include a closed-loop feedback mechanism to bias the latch near the metastable point. This is necessary because even closely matched cross-coupled inverting gates are unlikely to equally resolve to 1's and 0's in practice. Deterministic supply voltage and temperature biases compounded by process variations and even active attacks all contribute to pushing the latch consistently towards either logic state. To counteract these deterministic effects, the transconductance of either (or both) inverting gates is adjusted dynamically based on the ratio of 1's to 0's. A number of metastability-based TRNG implementations [38] (including Intel's [39]) substitute t_r quantification with statistical filters that attempt to "correct" the latch bias. However, these methods do not improve the randomness of the generated bit stream since they attempt to remove deterministic biases by equally-deterministic means.

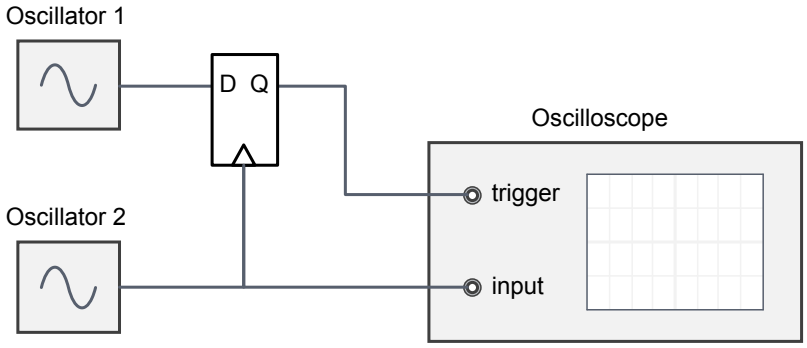


Figure 2.19: Basic metastability characterization setup

2.4.5 Physical Parameter Sensing

The generation of random data by tapping physical sources such as thermal noise can be alternatively described as a sensing process. Although the focus of random number generation is the "sensing" of non-deterministic sources and minimizing the influence of deterministic ones, a setup which does the opposite (maximize deterministic biases and minimize non-deterministic ones) can be used as a sensor for systemic changes in the latch's environment. This duality is illustrated in Table 2.1. In Chapter 5, the sensing of physical parameters is introduced as a new application of flip-flop metastability.

2.5 Metastability Characterization

The value of the metastability resolution time constant τ (which affects the MTBF of synchronization exponentially) is not taken into consideration during the design of common library flip-flops. Optimizing conventional flip-flop performance metrics such as the nominal clock-to-q delay or the setup and hold times often results in worse metastability resolution performance (higher τ) [16] [40]. The introduction of supplementary flip-flop features such as scan chains was also shown to have a similar effect [41]. Typical cell library flip-flops are therefore poorly-optimized for metastability resolution and have considerably different τ values. Hence, significant research effort has been invested in characterizing the metastability resolution performance of library flip-flops in existing technologies. This section surveys some of the work in this area.

Experimental setups were devised to bring flip-flops into metastable states and characterize their resolution time t_s against the size of the critical input window (represented by the expression $T_w \times e^{-t_s/\tau}$) [42]. These setups were primarily aimed at determining the resolution characteristics of flip-flops (the parameters τ and T_w) but have revealed other significant effects and gave further insight into the phenomenon. Jex et.al. [43] used uncorrelated oscillators to generate the clock and data signals to drive a flip-flop into metastability and plot its clock-to-q delay histogram. The same basic arrangement (Figure 2.19) was used in several other investigations including [44] where the non-nominal but deterministic region of operation was distinguished from the true non-deterministic metastable region. The latter study showed that measurements in the deterministic region can yield τ values that are deceptively higher or lower than the actual value in deep metastability. Foley [45] proposed the usage of "masks" (regions on the flip-flop output plots) to standardize the definition of metastability failures or *violations* such that reports from different experimental setups can be compared alongside each other. Kinniment et.al. [46] designed a setup that uses a feedback loop to lock on the tipping point of flip-flops and consistently bring them into metastable states. This has enabled the characterization of deep metastable events corresponding to MTBF values of up to 3 years. In consequence, two effects impacting synchronizer reliability in deep metastability were observed experimentally. First, the change in failure rates as a result of the crossing of metastability from the master to the slave latches at the falling edge of the clock (the clock back-edge effect). Second, the different τ values of the master and slave latches, commonly causing very significant over-estimation of the MTBF. The same scheme was later implemented on-chip [47] achieving further extension in the observed clock-to-q delay range. Other investigations looked into the performance of multi-synchronous and adaptive synchronizers [48] and more recently the scaling of the parameter τ with technology generations [49].

On the simulation side, bisection search [12] is the dominant method of characterizing synchronizer behavior and dates back to 1975 [50]. The values of τ and T_w obtained via bisection can be trusted to the accuracy of the simulator and that of the device and circuit models. One difficulty with bisection is that the limited numerical resolution of simulators prevents representing the extremely-small input transition time differ-

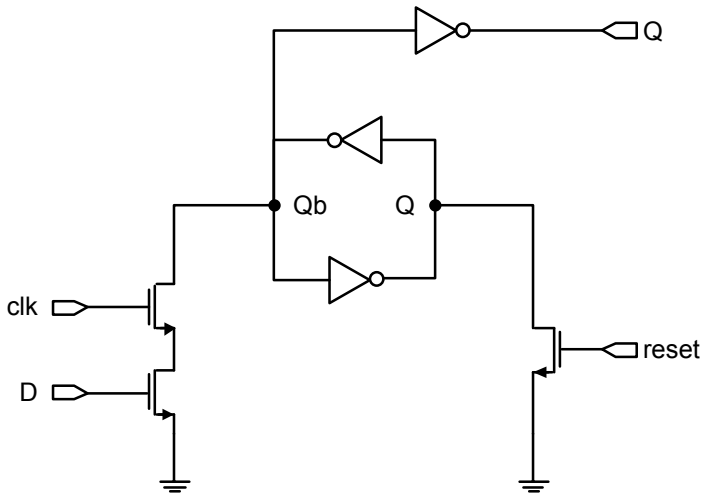


Figure 2.20: Jamb Latch

ences required to induce very deep metastable states. Greenstreet et.al. [19] proposed a simulation technique that overcomes this difficulty by combining small and large signal analysis. Another method to characterize metastability via simulation is to short-circuit the metastable nodes, let them diverge and then calculate τ as the divergence time constant [44]. This can be done in one transient simulation and so it is easier to implement and faster to execute compared to bisection. However, care must be taken because this method does not capture the large transient effects that occur during the onset of metastability and its propagation between the master and slave latches.

On a third front, there have also been attempts to construct flip-flops with smaller τ values that are particularly optimized for synchronization. One of the earliest of these is the Jamb latch [44] (Figure 2.20). It is a minimalistic latch that consists of a cross-coupled inverters (as opposed to a pair of larger inverting gates) and a small set of pull transistor networks to minimize the capacitance load on the metastable nodes. A better design, the robust synchronizer, was proposed in [51] and uses additional transistors to increase the regenerative loop gain when a metastable state is detected. Another improved design was presented in [41] demonstrating lower susceptibility to supply voltage variation. The metastability-resolution performance of latches has also been investigated in the subthreshold region [52] and under process variations [3] [53].

2.6 The Bundling Constraint

In addition to allowing a sufficient amount of time for synchronization, asynchronous communication must satisfy an important condition: data bit transitions must arrive sufficiently-earlier than the accompanying transition in the request signal. This is referred to as the bundling constraint and is necessary to ensure that the receiver latches correct data values. The constraint can be expressed as:

$$t_{\text{req}} - t_{\text{data}} > t_{\text{b}} \tag{2.14}$$

where t_{req} and t_{data} are the arrival times of the request and data signals at their destination registers and t_{b} is a safety margin.

All the combinational path delays between the sender and receiver, wire propagation delays, clock jitter and other uncertainties must be taken into account when satisfying Equation 2.14. This can be done by inserting a delay element that is equal to the sum of these delays and uncertainties (plus a safety margin) between the flip-flop issuing the request signal (at the sender) and the first synchronizer flip-flop (at the receiver).

The delay t_{b} can be implemented in a number of ways. One of these is to have the sender account for all the propagation delay differences between the request and data paths (up to the destination flip-flops) on its own. However, this is not a practical solution because the sender would require knowledge of the internal propagation delays of the other party. The receiver would also be unable to satisfy the criterion on its own for the same reason. Therefore, a practical solution would be to define partial delay requirements on both parties by protocol. For example, assuming t_{b} = 100ps, one suitable protocol would be:

1. The sender must output *req* after *data* by 50ps
2. The receiver must delay *req* relative to *data* by a further 50ps.
3. The Inter-block wiring delay of *req* must be equal or larger than the inter-block wiring delay of *data*

2.7 Definitions

2.7.1 Terminology

This subsection introduces four terms to describe the behavior of unconstrained flip-flops. The reader is advised to become familiar with the proposed terminology before proceeding to read the following chapters of the thesis, particularly Chapter 3.

If the transition of the flip-flop's input occurs close to the clock edge, the flip-flop may succeed or fail to copy the post-transition value of the input. The transition is "captured" if the flip-flop successfully copies the new (post-transition) value before the following clock edge (even after becoming metastable for a while). Otherwise, the transition is said to have "not been captured". These two scenarios can be broken into four depending on whether metastability occurs or not:

Case 1 (Safely Captured): If the post-transition input state is copied to the flip-flop output within a nominal clock-to-q delay, the transition is said to have been *safely captured*.

Case 2 (Unsafely Captured): If the process takes longer than the nominal clock-to-q delay (but the output is copied nonetheless), the transition is said to have been *unsafely captured*.

Case 3 (Safely not Captured): If the flip-flop does not capture the transition but its output stabilizes (does not change) after the clock-to-q delay, the transition is said to have been *safely not captured*.

Case 4 (Unsafely not Captured): If the flip-flop becomes metastable but eventually rolls back to the pre-transition input state (even after swinging to the post-transition state momentarily) then the transition is said to have been *unsafely not captured*.

At any clock edge occurring at time t_{clk}, the flip-flop behavior is determined by the arrival time t_{in} of the flip-flop input. If t_{in} is sufficiently smaller than t_{clk}, the setup condition of the flip-flop is met and the transition is captured safely. If t_{in} is within the setup-hold time window of t_{clk}, the transition may be unsafely captured or unsafely not

Table 2.2: Flip-flop behavior cases and terminology

Case	Condition	Flip-Flop Metastable
Safely captured	$t_{in} < t_{clk} - t_{su}$	No
Safely not captured	$t_{in} > t_{clk} + t_h$	No
Unsafely captured	$t_{clk} - t_{su} < t_{in} < t_{clk} + t_h$	Yes
Unsafely not captured	$t_{clk} - t_{su} < t_{in} < t_{clk} + t_h$	Yes

Table 2.3: Flip-flop logical primitives

Name	Observed Output Response	Inferred Input
Primitive 1	Input transition is captured (safely or unsafely)	$t_{in} < t_{clk} + t_h$
Primitive 2	Input transition is not captured (safely or unsafely)	$t_{in} > t_{clk} - t_{su}$

captured depending on which state the metastable flip-flop finally resolves to. If t_{in} is sufficiently larger than t_{clk}, the transition is safely not captured. These cases and the corresponding conditions are listed in Table 2.2.

2.7.2 Logical Primitives

Table 2.2 lists four logical primitives in the form $p \rightarrow q$ (p *implies* q) where p is an input condition and q is an output response. For example, if $t_{in} < t_{clk} - t_{su}$ then the transition is captured safely. It is also possible to construct similar logical primitives that enable us to deduce input conditions given an observable output response. Of these, two are of particular interest. First, if the input is captured (safely or unsafely) then $t_{in} < t_{clk} + t_h$. This is because, referring to Table 2.2, no input transitions arriving later than $t_{clk} + t_h$ can be captured. Second, if the input is not captured (safely or unsafely) then $t_{in} > t_{clk} - t_{su}$. This is because all input transitions arriving before $t_{clk} - t_{su}$ are captured. These two primitives are summarized in Table 2.3 and form the base of the proofs in Chapter 3.

Chapter 3

Hiding Synchronization Latency by Speculation

3.1 Clock Domain Crossing

Many-core SoCs are now prevalent. At the time of writing this thesis, SoCs consisting of quad-core processors and as many as eight GPU cores have already made their way into the tablet device market (e.g. Tegra 3 in Google Nexus 7). Multi-core processors have been dominant in desktop and laptop computing for a few technology generations and single-core processors are now confined to the racks of legacy systems.

The shift towards many-core computing was a natural consequence of the scaling of CMOS. The trend of making faster devices and packing them in larger densities has continued until maintaining synchrony across an entire chip became an intractable task. The huge integration density of modern systems also meant that, from a development point of view, designing the whole system from elementary logic components also became impracticable. Instead, just as the scale of software has exploded with the increase in memory capacity that it became necessary to use libraries at one point in computer history, so did the scale of today's hardware. Thus, Intellectual Property (IP) modules emerged as "compiled" hardware components which are developed

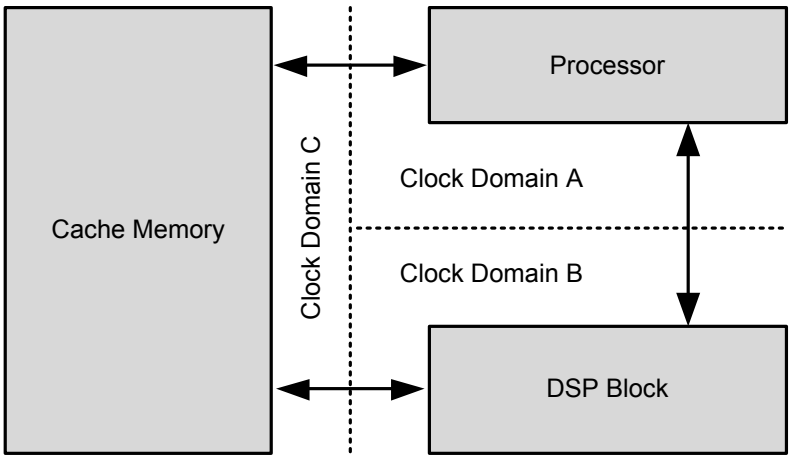


Figure 3.1: Clock domain crossing

and optimized independently and later integrated to create complete systems. IP cores enabled designers to create systems with better performance and unprecedented scalability. Example commercial products include ClearSpeed CSX700 (192 cores), Ambric Am2045 (336 cores) and the prospective 4096-core chip from Adapteva.

In effect, CMOS has began to transition towards Globally Asynchronous Locally Synchronous (GALS) systems [54]. Making the shift to GALS requires overcoming a number of difficulties. On the hardware level, designing asynchronous interfaces for clock domain crossing (Figure 3.1) is one of the major challenges. Asynchronous communication requires synchronizers to guard against the catastrophic impact of metastable states. However, synchronizers introduce latency and degrade the performance of inter-core communication links. This trade-off is becoming more detrimental as the number of on-chip synchronous islands increases and the portion of on-chip communication bandwidth that crosses clock domain boundaries grows larger.

This chapter describes two novel schemes that hide synchronization latency by overlapping synchronization with few speculative computation cycles. What remains of this section surveys two categories of synchronization-free clock domain crossing solutions with the aim of establishing a baseline for comparing the proposed schemes.

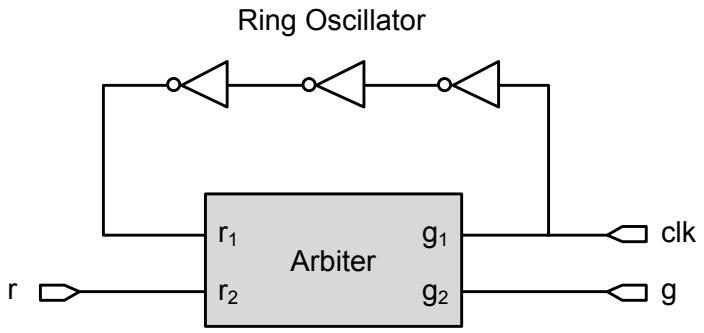


Figure 3.2: Pausable clock generator

3.1.1 Pausable Clocking

Synchronization failures occur because it is impossible to make a binary decision based on an analog value in a finite amount of time with zero probability of failure. A work-around solution to this problem is to remove the "finite response time" constraint and allow a theoretically-unbounded metastability resolution time. This solution was first proposed in [55] and is applied by allowing the clock to be paused until any occurring metastable states are resolved. The scheme is often referred to as pausable clocking.

Figure 3.2 demonstrates a simple implementation of a pausable clock generator. Here, a local Ring Oscillator (RO) generates a clock signal and an arbiter pauses it every time a request *r* is asserted. If *r* transitions at nearly the same time as the following clock edge the arbiter may become metastable but will not grant either request until metastability is resolved. This generator can be coupled with an asynchronous input port to latch asynchronous data without latency and without experiencing failures. The two are commonly implemented as an asynchronous wrapper [56] as illustrated in Figure 3.3. The input port communicates with the asynchronous sender via two handshake signals *ack* and *req*. When *req* is asserted indicating the validity of *data*, the port asserts *pause_req* requesting the generator to pause the clock. Any metastable state arising due to the assertion of *pause_ack* near the clock edge is resolved internally within the generator. When the clock is paused, the generator asserts *pause_ack*. Subsequenty, the input port makes *data* available at the synchronous module input, asserts *valid* and then de-asserts *pause_req*. The generator resumes clocking then and the synchronous module latches

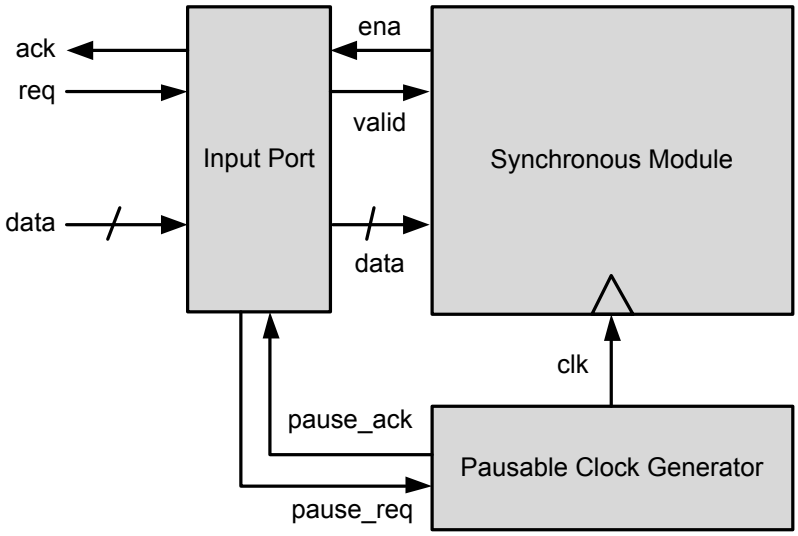


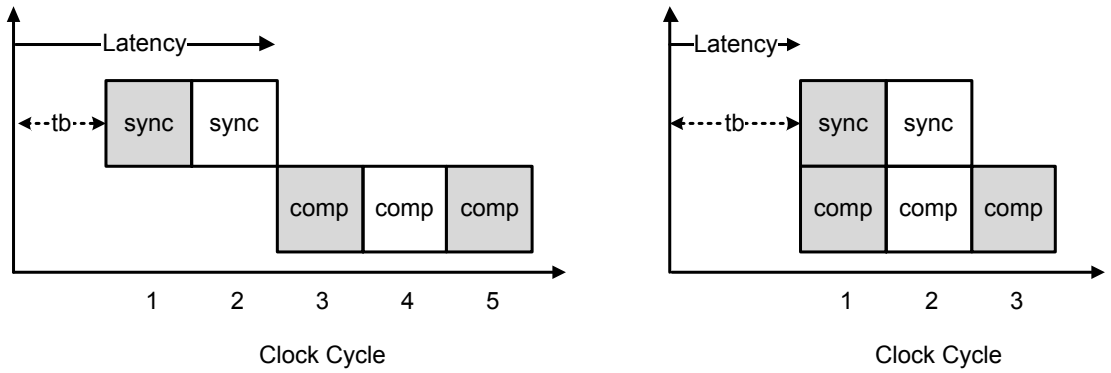
Figure 3.3: Asynchronous wrapper implementation of pausable clocking

data safely without any delays. This technique can be generalized to multiple ports [57] and exist in different variations (a comprehensive review of which is presented in [58]).

Pausable clocking has been demonstrated to work correctly in silicon [57] [59] but is not used in practice. This is because substituting an external crystal clock with a local RO is not an option that most designers are comfortable with. Locally-generated clocks have poor stability, high sensitivity to process, voltage and temperature variations and cannot be tuned easily. Existing solutions to mitigate these limitations exist but incur area, power and complexity costs that are commonly considered unacceptable [18].

3.1.2 Correlated Clocks

On-chip clocks are sometimes derived from the same source crystal and may share some timing relationships. Synchronization can be avoided in these cases by anticipating and avoiding the conflicts between the local clock and asynchronous requests. Several latency-free and reliable interfaces have thus been proposed for mesochronous clocks [60], closely-matched (plesiosynchronous) clocks [61], rationally-related clocks [62] and periodic clocks [63]. The solution provided in [61] can also be generalized to arbitrary clocks by prior establishment of their timing relationship.



(a) Conventional synchronization

(b) Speculation

Figure 3.4: Hiding synchronization latency by speculation

3.2 Introduction to Speculation

An alternative strategy to mitigate synchronization latency is to use redundant hardware to perform speculative computations during synchronization cycles. This "hides" synchronization latency by overlapping it with an equivalent number of computation cycles. If computing an output based on an asynchronous input requires n synchronization cycles and m computation cycles, this method yields a processing time of $\max(m, n)$ cycles as opposed to $m + n$ for conventional synchronization. This reduces the total latency to $t_b + T \times \max(m, n)$ where t_b is the bundling delay (see Section 2.6) and T is the clock period. The difference between the two cases is illustrated in Figure 3.4.

Speculation offers several advantages over conventional solutions to the problem of synchronization latency. First, this approach is entirely architectural and does not target the synchronization process itself. Therefore, it does not rely on any assumptions about the relationship between the communicating clocks and does not require fast metastability-resolving flip-flops. Second, trading reliability and low-latency with duplicated hardware will be an increasingly-affordable option in future technologies because of the continuous growth of available design area. This is in contrast to the metastability-resolution performance of flip-flops which deteriorates with supply voltage scaling [51] and growing process variations [3] and also the relative timing relationships which are becoming increasingly difficult to verify [61].

Despite these advantage, speculative synchronization has received little attention. Kinniment et.al. have proposed the only speculative form of synchronization to appear in the literature, a technique they referred to as Speculative Synchronization [2]. They argue that metastable states occur relatively rarely compared to handshake requests and that incurring two cycles to synchronize each individual handshake is thus unwarranted. Their scheme involves using a single flip-flop k as a synchronizer and speculating that it does not become metastable. A detector circuit can then reliably identify, n cycles later, whether k has actually become metastable. If this was the case, each register in the synchronous block is restored to a backup copy which is kept in an n-level stack. Using this form of speculation, the latency of processing the asynchronous request is reduced to a single cycle only (plus t_{b}). The cost is that each register needs to be duplicated n times. What remains of this chapter presents two novel forms of speculation that reduce latency to zero cycles (plus t_{b}) and have lower hardware duplication costs.

3.3 Datapath Unfolding

3.3.1 Overview

Speculation is the use of either time or resource redundancy to perform potentially useful work. Modern digital systems employ speculation at different abstraction levels. For example, memory management speculatively populates cache hierarchies with prefetched data to reduce the impact of slow memory access on processing speed [64]. Also, processors that use branch prediction execute the instructions following branches speculatively to increase throughput [65]. At the software level, speculative multithreading delegates branch instructions to idle processing cores as separate threads [66]. The latter is also facilitated by speculation-aware compilation frameworks [67].

Performing speculative computations in pipelined systems is particularly easy. This is because restoring the state of a pipeline in the case of misspeculation is trivial. For example, when a branch condition in a pipelined processor is evaluated, invalid instructions in the fetch and decode stages can be discarded by flushing these stages. On the other hand, speculative computations cannot be "reversed" in a similarly

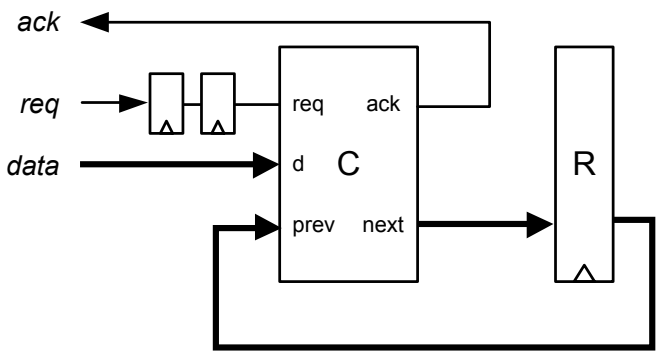


Figure 3.5: A Moore machine with an asynchronous port

straight-forward manner in non-pipelined systems. This is because non-pipelined systems have loop dependencies (i.e. feedback paths) such as the one represented by the expression $x \leftarrow x + 1$. The existence of loop dependencies can corrupt the system state in the case of misspeculation (pipelined systems are free from such dependencies by definition). Nevertheless, arbitrary designs can be converted into functionally-equivalent pipelines by *unfolding* [68]. Unfolding eliminates loop dependencies by instantiating design duplicates. For example, a design represented by $x \leftarrow x + 1$ can be unfolded into $x_c \leftarrow x + 1$ where x_c is a copy of x. By duplicating the register x, the design is converted into a functionally-equivalent two-stage pipeline. Unfolding is widely employed by compilers [69] [70] [71] and schedulers [72] [73] to increase execution throughput. It is also equivalently-capable of resolving loop dependencies in hardware implementations; it is used extensively in digital signal processors [74] [75] and has been proposed for general purpose synthesis [76].

Although pipelining a design by unfolding is used primarily to increase throughput, it can also be used to perform speculative computations during synchronization cycles. To demonstrate how, consider the generic synchronous module shown in Figure 3.5. The module is represented by a Moore machine consisting of the state register R, the combinational block C and the asynchronous port [*req*, *d*, *ack*]. To maintain reliability, two flip-flops are added to synchronize *req*. The latency introduced by this chain can be "hidden" by speculatively computing what the machine state would have been if *req* changed two cycles earlier. An arrangement which does this is shown in Figure 3.6.

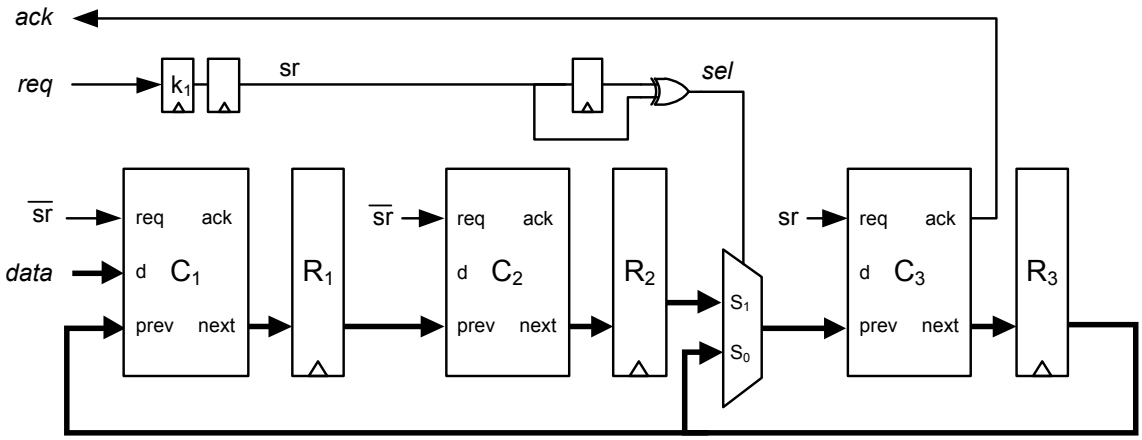


Figure 3.6: Unfolded Moore machine

Here, a Moore machine $\{C_3, R_3\}$ operates identically to the one in Figure 3.5 before the arrival of data. Two duplicates ($\{C_1, R_1\}$ and $\{C_2, R_2\}$) are used to compute what the state of R_3 would have been if *req* was toggled two cycles earlier [1]. When an actual *req* toggle appears at the synchronizer output, *sel* is asserted for one cycle and the machine uses the speculative state in R_2 to "jump" to the state corresponding to the third cycle after data consumption. Afterwards, the machine resumes computations and acknowledges the sender upon completion.

Note that the arrival of *data* may violate R_1's setup time condition and cause it to latch a corrupt (or even metastable) state. However, these states are discarded because not every speculative state computed by the pipeline is actually used (hence the adjective "speculative"). If *req* was asserted after *data* by a sufficient delay (bundling constraint - see Section 2.6) then the assertion of *sel* will imply that the setup condition of R_1 has been met two cycles earlier. This premise is proved logically in the following subsection.

[1] We assume that the machine latches *data* only on the first cycle after *req* toggles and so *data* is connected to C_1 only

3.3.2 Proof of Correctness

The state held by R_2 is used to compute the subsequent state of R_3 only when $sel = 1$. The assertion of *sel* implies that the first synchronizer flip-flop (denoted k_1) captured a transition of *req* two cycles earlier (premise p). Now, p implies [2]:

$$t_{\text{req}} < t_{\text{clk}} + t_{\text{h}} \tag{3.1}$$

where t_{req} is the arrival time of *req*, t_{h} is the hold time of k_1 and t_{clk} is the clock edge.

The transition of *data* satisfies the setup condition of R_1 when:

$$t_{\text{data}} < t_{\text{clk}} - t_{\text{su}} \tag{3.2}$$

where t_{su} is the setup time of R_1.

If $t_{\text{req}} - t_{\text{data}} > t_{\text{su}} + t_{\text{h}}$ (Constraint 1) then Inequality 3.1 will imply Inequality 3.2 (the transition of k_1 will imply the satisfaction of the setup condition of R_1). This can be shown by re-writing Constraint 1 as:

$$t_{\text{req}} > t_{\text{data}} + t_{\text{su}} + t_{\text{h}} \tag{3.3}$$

Now, from Inequality 3.1 we know that $t_{\text{clk}} + t_{\text{h}}$ is larger than t_{req}. Therefore, we can replace the left-hand side of the inequality above with $t_{\text{clk}} + t_{\text{h}}$ as follows:

$$t_{\text{clk}} + t_{\text{h}} > t_{\text{data}} + t_{\text{su}} + t_{\text{h}} \tag{3.4}$$

Finally, simplifying the above we get:

$$t_{\text{clk}} > t_{\text{data}} + t_{\text{su}} \tag{3.5}$$

which is equivalent to Inequality 3.2.

[2] using Primitive 1 (Subsection 2.7.2)

With Constraint 1, the behavior of the system can be described as follows. If *sel* goes high at a cycle n then k_1 has captured a transition at cycle $n-2$ which in turn implies that the setup condition of R_1 has been been met at cycle $n-2$. In practice, Constraint 1 can be met by inserting a delay element in the combinational path of the request signal. This will delay the arrival of the request transition relative to the data bit transitions.

3.3.3 Behavioral Constraints

The presented approach can be generalized to a Moore machine with any number of synchronous inputs and outputs. Input connections must be duplicated to the blocks C_1 and C_2 while output connections are derived from the block C_3. There is however a condition that the machine must satisfy: a change in *req* should not affect the behavior of the machine's outputs during the following 2 cycles. This is necessary to ensure that the sudden state jump performed by the machine when the speculative state is committed remains invisible to its environment. As a consequence of the latter property, there are no restrictions on what can be observed by the environment on the output of R_3.

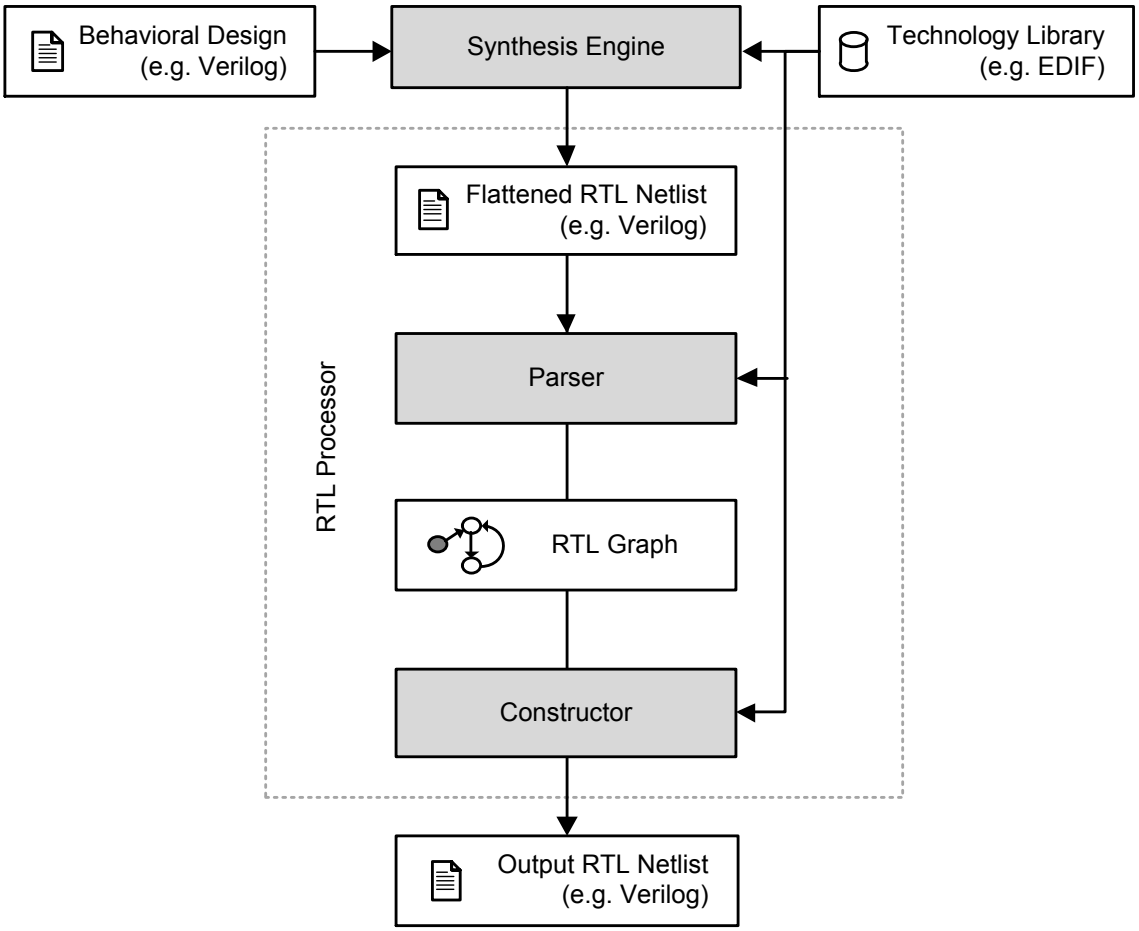


Figure 3.7: Modified design flow

3.3.4 RTL Automation

Datapath unfolding requires no more than gate-level manipulation and so it can be automated by an RTL tool (a netlist processor) which is integrated into the design flow as a post-synthesis step (Figure 3.7). In essence, the function of the tool is to generate a design in the form shown in Figure 3.6 given the design in Figure 3.5 as an input. If the delay and behavioral constraints are met, the generated design will behave identically to the original (with the exception of the hidden latency of the asynchronous port).

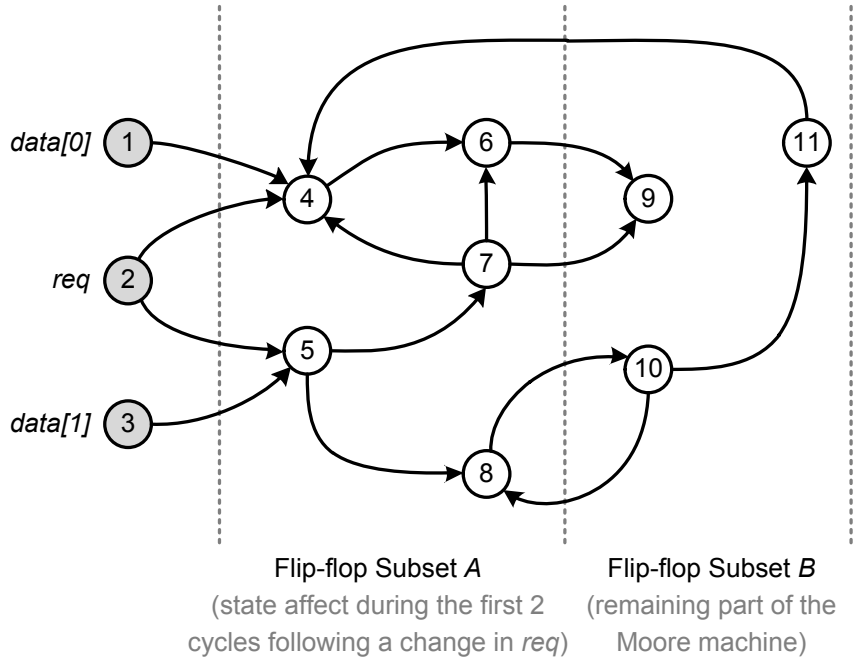


Figure 3.8: Graph representation of an RTL netlist

3.3.5 Cost Analysis

The previous discussion of datapath unfolding involved instantiating n machine duplicates for n synchronization cycles. If the machine represents a large design then duplicating it entirely will incur unacceptable area and power costs. Fortunately, this is not actually required and only a subset of the machine's flip-flops and combinational logic needs to be duplicated.

To illustrate why this is the case, consider the directed graph representation of the Moore machine shown in Figure 3.8. Here, the graph vertices represent individual flip-flops while the edges represent combinational logic paths. The shaded vertices represent the asynchronous flip-flops (2-bit *data* and *req*). Let A denote the subset of flip-flops that are within a 2 edge distance of *req* (A is located by performing a breadth-first search with a depth of 2 starting from *req*). Further let B represent the remaining flip-flops in the Moore machine. In essence, subset A includes the flip-flops whose value may depend on *req* during the first two cycles of a change in *req*. The flip-flops in subset B will not be affected by the change in *req* during this period. Hence, there is no need to speculatively

compute what the state of B would have been if *req* changed two cycles earlier: it will be the same regardless. In consequence, while $\{C_1, R_1\}$ and $\{C_2, R_2\}$ have been initially described as duplicates of the entire Moore machine, they need to contain the flip-flops in subset A and their input combinational logic only.

3.3.6 Synthesis Results

The RTL tool described in Subsection 3.3.4 was implemented in Java and used to analyze a number of designs from OpenCores [77]. The designs were synthesized using the Faraday 65nm commercial library. For each design, the RTL tool was used to apply datapath unfolding to a selected input port and calculate the associated duplication costs. The area costs are compared against the periodic synchronizer presented in [63] [3]. The purpose of this comparison is to demonstrate that datapath unfolding, besides not requiring correlated clocks or requiring design modifications, is more cost effective than existing synchronization solutions.

The results (Table 3.1) demonstrate that applying datapath unfolding to communication controllers incurs, on average, only 8.3% of the baseline area cost. To investigate how these costs compare to those of generic logic blocks, the technique has also been applied to four processors and a DSP core. The costs for the latter designs were found to be significantly higher (having an average of 268% of the baseline). This supports the conclusion that data communication and protocol handling logic is inherently more suitable for hiding latency by state speculation. This is because, in general, communication protocols and early data consumption logic often perform trivial operations that involve a limited subset of flip-flops and logic gates.

[3] taking $b = 10$, $w = 256$ and $M = 1$ and scaling area by 2 to map from 45nm to 65nm

Table 3.1: Synthesized designs and associated duplication costs

Design	Asynchronous Input	Gates	Flip-Flops	Design Area (μm^2)	Duplicated Logic (μm^2)	Area Cost (relative to [63])
Communication Controllers						
Simple RS232 UART	csr_we	225	96	1142.7	1459.2	12%
I2C Slave	scl	364	125	1513	78.8	1%
SPI core	stb_i	236	131	1562.2	2378.2	19%
MDI receiver	madi_clk_in	363	77	1848	1222.7	10%
PS/2 Host Controller	send_req	170	66	789.4	521	4%
JTAG Master	Shift_Strobe	341	100	1521	511.4	4%
Processors						
AVRtinyX61core	En	2134	383	8909.4	16853.2	133%
HPC16	ACK_I	2810	471	9625.3	19057.2	151%
ae18	iwa_ack_i	3359	1091	18130.6	31644.2	250%
miniMIPS	n0	12065	1938	47329	93941.2	743%
DSP Cores						
PID Controller	i_wb_stb	1764	473	8264	8039	64%

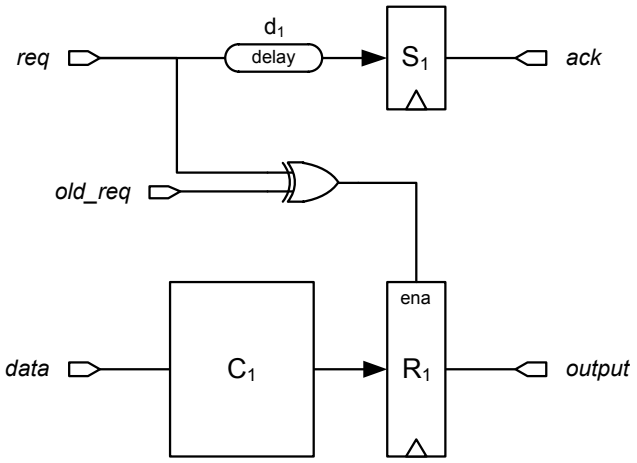


Figure 3.9: Pipeline stage

3.4 Sequenced Latching: Pipelined Designs

3.4.1 Overview

This section introduces another speculative scheme that requires less duplication than datapath unfolding. In short, the scheme employs the synchronizer as a state machine to sequence a series of speculative latching operations. The synchronizer is constrained such that it fails to capture the following state when the setup condition of the data registers is not met. Therefore, corrupt register data is overwritten by correct values on the following cycle. This section describes the scheme for pipelined-designs while Section 3.5 generalizes it to non-pipelined designs.

The technique can be illustrated by referring to the pipeline stage in Figure 3.9. The stage consists of a generic combinational block C_1, a register R_1 and one synchronizing flip-flop S_1. Assume that *req* and *data* are generated by an asynchronous sender that uses a two-phase handshake protocol and that *old_req* is the value of *req* before the beginning of the handshake (this value is stored in a synchronous flip-flop). Further assume that the sender always asserts *req* after *data* by a sufficient time margin (bundling constraint - see Section 2.6). When *req* transitions, R_1 is enabled and latches $C_1(data)$ on the following clock edge.

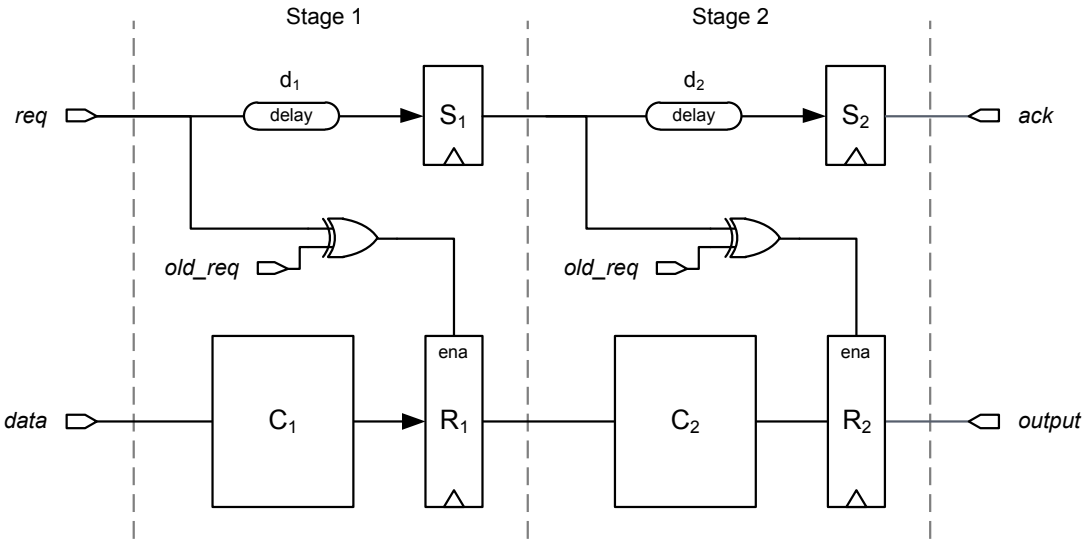


Figure 3.10: Two-stage pipeline

A sufficiently-long delay is introduced by the delay element d_1. This delay guarantees that every time S_1 captures a transition of *req*, R_1 does the same, safely. Therefore, the behavior of the stage can be described as follows. At any cycle, if S_1 captures a transition of *req* then R_1 captures both *data* and *req* safely.

If multiple such stages were connected in series (and assuming that each flip-flop S_i behaves monotonically [4]) then all stages will behave in the same fashion. Therefore, the behavior of stage 1 can be generalized as follows:

Lemma 3.1. *If d_i is sufficiently long and S_{i+1} captures a transition of S_i then R_{i+1} captures the same transition, safely.*

Now, given that a change in the state of S_k implies a change in the state of S_{k-1} in a previous cycle, the behavior of the pipeline can be summarized as:

Theorem 3.1. *If the state of S_n changes then the setup conditions of $R_i \forall i \in \{1...n\}$ have been met in succession in previous cycles.*

Note that this pipeline does not prevent metastability nor the resulting failures from occurring for that is impossible. Each synchronizer flip-flop S_i can still exhibit prolonged clock-to-q delays that may corrupt the data latched by pipeline register R_{i+1}. However,

[4] This assumption is examined in Subsection 3.4.5

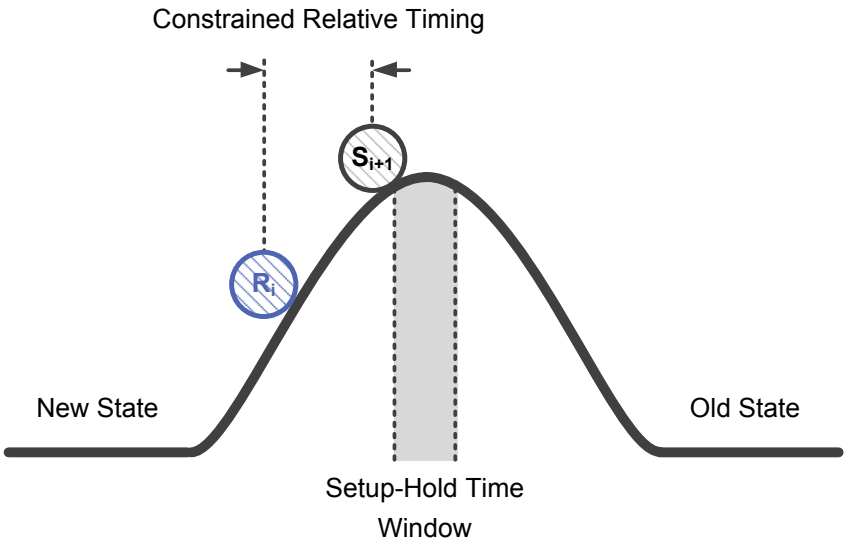


Figure 3.11: Ball and hill analogy of sequenced latching

every S_i transition that is not safely captured by R_{i+1} will be too late to be captured by S_{i+1}. When any synchronizer flip-flop S_i exhibits such a late transition, the synchronizer will remain in the same state for the following cycle and the pipeline will "stall" allowing R_{i+1} to re-latch its input correctly before the latching sequence proceeds. These events happen relatively rarely and so the average number of cycles required to complete n latching operations remains approximately equal to n.

The behavior of each pipeline stage can be represented by a system of a hill and two balls as shown in Figure 3.11. The initial position of the balls represent the arrival time of S_i's transition at R_i and S_{i+1}. Due to the arbitrary clock-to-q delay of S_i, the balls can be initialized at any position on the hill. However, their *relative* placement is constrained such that for all the initial positions of S_{i+1} that will cause S_{i+1} to roll to the new state, R_i will also roll to the new state. The relative displacement constraint is large enough such that even if S_{i+1} becomes metastable before rolling to the new state, R_i will roll to the new state in a nominal time. In other words, the setup condition of R_i will be met even if the synchronizer flip-flop S_{i+1} captures the transition unsafely,

In essence, this method uses the synchronizer as a state machine to control/sequence the flow of data through a pipeline and to overcome corrupt latching by inducing re-latch (stall) cycles. In what follows, it is referred to as *Sequenced Latching*.

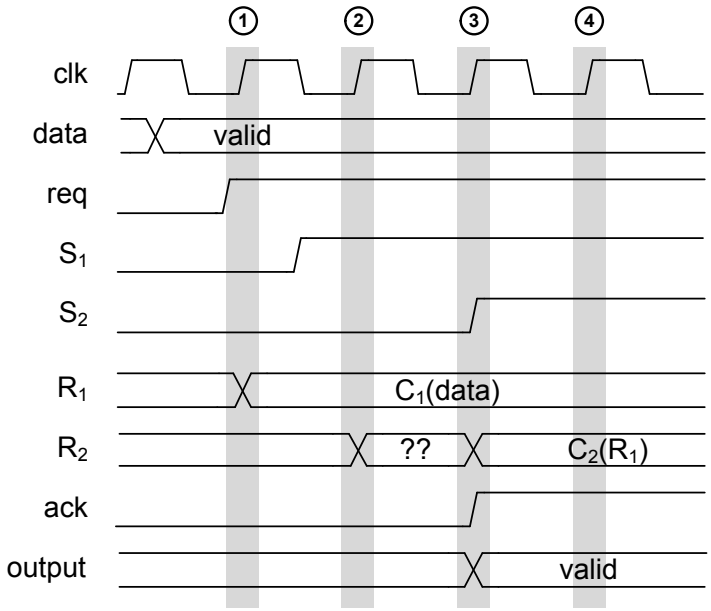


Figure 3.12: Handshake example 1

3.4.2 Example

Figure 3.12 shows an example of how this pipeline behaves. In this handshake, the sender makes *data* available on the bus and asserts *req* a sufficient time later (bundling constraint). *req* arrives close to the clock edge and causes S_1 to become metastable. In the meanwhile, *data* arrived sufficiently earlier and is latched by R_1 correctly. S_1 produces a delayed output transition which does not have sufficient time to propagate to all flip-flops in R_2. In consequence, R_2 latches a corrupt value of $C_2(R_1)$. However, the late transition of S_1 is arrives too late and is not captured by S_2 (because of the sufficiently-long delay d_2). In the following cycle, R_2 re-latches $C_2(R_1)$ correctly and the *ack* output is asserted.

Figure 3.13 shows the state diagram of the pipeline. Note that state transition conditions are expressed in terms of both *req* and the satisfaction of the setup time constraints of R_1 and R_2. The transition from state 00 to state 11 (or vice versa) necessarily implies the satisfaction of the setup conditions of R_1 and R_2 in succession. Violations of any of these setup conditions causes a stall cycle and prevents a state change.

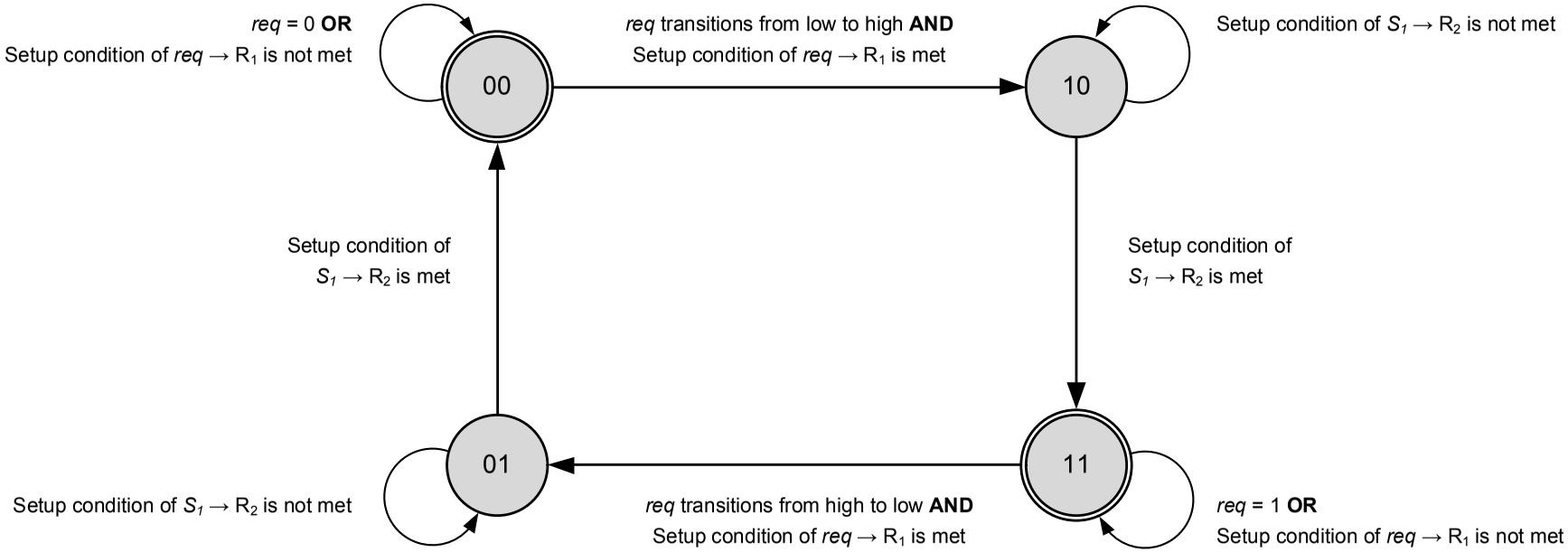


Figure 3.13: Pipeline state diagram (state encoding = S_1, S_2)

3.4.3 Proof of Correctness

At any cycle, if flip-flop S_{i+1} captures the transition of S_i at t_{S_i} then [5]:

$$t_{S_i} + t_{pd}(S_i \rightarrow S_{i+1}) < t_{\text{clk}} + t_{\text{h}}(S_{i+1}) \tag{3.6}$$

where t_{clk} is the time of the clock edge, $t_{\text{h}}(k)$ is the hold time of flip-flop k and $t_{pd}(k_1 \rightarrow k_2)$ is the propagation delay of the path $k_1 \rightarrow k_2$.

Now, t_{S_i} meets the setup constraint of R_{i+1} when:

$$t_{S_i} + t_{pd}(S_i \rightarrow R_{i+1}) < t_{\text{clk}} - t_{\text{su}}(R_{i+1}) \tag{3.7}$$

where $t_{\text{su}}(k)$ is the setup time of flip-flop k.

If the delay element d_{i+1} is adjusted such that:

$$t_{pd}(S_i \rightarrow S_{i+1}) - t_{pd}(S_i \rightarrow R_{i+1}) > t_{\text{h}}(S_{i+1}) + t_{\text{su}}(R_{i+1}) \tag{3.8}$$

then Inequality 3.6 will imply Inequality 3.7 (using the same logic in Subsection 3.3.2). Therefore, the change in the state of S_{i+1} will imply the satisfaction of the setup condition of R_{i+1} (Lemma 3.1).

Inequality 3.6 also implies the transition of S_i at a previous cycle. Using the same logic above, the latter implies the satisfaction of the setup condition of R_i at that cycle. By applying this argument recursively, Inequality 3.6 implies the satisfaction of the setup conditions of $R_j \forall j \in \{1...i\}$ in succession (Theorem 3.1).

[5] using Primitive 1 (Subsection 2.7.2)

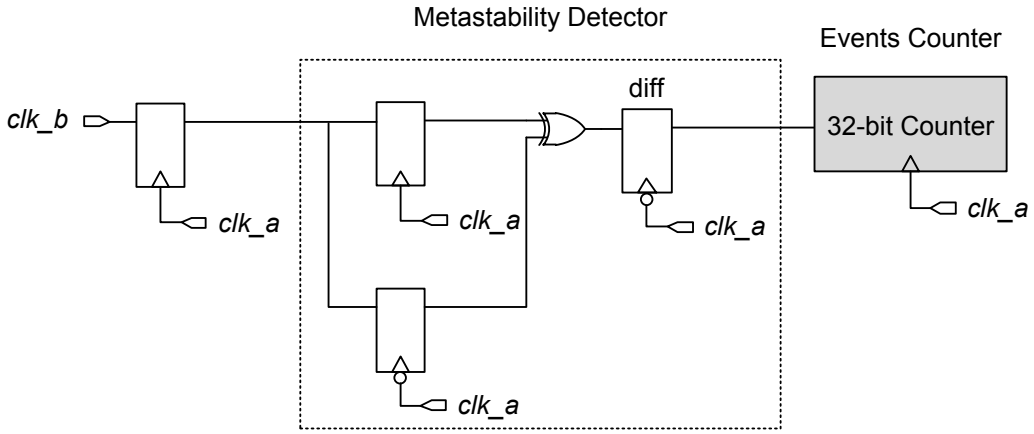


Figure 3.14: Tau characterization circuit

3.4.4 FPGA Verification

A benchmark system was implemented in an Altera Cyclone II FPGA to verify the correctness of sequenced latching (Theorem 3.1) and to demonstrate that the relative number of stall cycles is small.

Tau Measurement

Before running the benchmark, the parameter τ of the used FPGA device was measured to obtain rough estimates of failures rates and tweak the system accordingly. This measurement was performed by implementing a flip-flop with an asynchronous toggling input and a delayed-transition detection circuit (Figure 3.14). The detector captures and compares the output of a flip-flop at the falling and rising edges of the clock. When the two samples are different, a metastable event is flagged. The settling time for the flip-flop under test is determined by the time between the rising and falling edges of the clock (the duty cycle). To obtain fine control over the latter, the clock signal for the benchmark was generated using a precision Agilent 81133A Pulse Generator capable of 1ps resolution.

The clock's duty cycle was adjusted in small increments, each time recording the number of metastable events counted by the counter in a set period of time. Figure 3.15 shows a semi-log plot of the collected data. The straight-line segment on the plot resembles the exponential drop in the number of metastable events as the the resolution

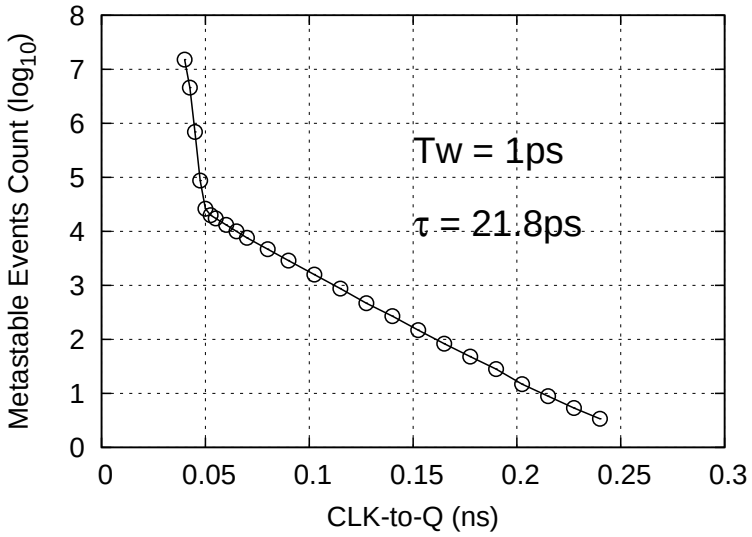


Figure 3.15: Tau chararization results

time of metastability is increased (Equation 2.11). The parameter τ is calculated as $-1/slope$ (where $slope$ is the slope of the straight-line segment) and has been obtained as 21.8ps for the FPGA device.

Verification of Functional Correctness

Following the evaluation of τ, the benchmark circuit shown in Figure 3.16 (resembling the pipeline in Figure 3.10) was implemented and tested. The circuit consists of a two-stage pipeline, a 4-phase handshake controller and several delay elements implemented as series of buffer-configured Lookup-Tables (LUTs). The circuit was connected to a simple sender (not shown in the figure) consisting of a 4-phase handshake controller and generating an array of *data* values using an asynchronous clock. The propagation delay constraint expressed by Inequality 3.8 is met by inserting the delay elements d_1 and d_2 whose delays are about 300ps each (the timing constraints for all paths were verified using TimeQuest, the static timing analyzer of the Altera FPGA software suite).

Several profiling sub-circuits and counters were added to the benchmark system to characterize various events. Counters C_1 and C_2 count the number of times S_1 and S_2 become metastable [6]. The clock's duty cycle was adjusted to detect clock-to-q transitions

[6] metastability is detected using the same detector circuit used to characterize τ

beyond 0.1ns. Counters C_3 and C_4 count the mismatches between any values written to the pipeline registers and corresponding reference values computed using the same array of *data* that is generated asynchronously. The mismatch events counted by C_3 occur when the improper arrival time of *req* violates the setup condition of the path $req \rightarrow R_1$. Similarly, the mismatch events counted by C_4 occur when the prolonged clock-to-q delay of S_1 violates the setup condition of the path $S_1 \rightarrow R_2$. Finally, the events counted by C_5 are the actual pipeline errors, i.e. the mismatches in R_2 after the transition of *req* appears at the output of S_2. Theorem 3.1 states that the pipeline output is correct every time synchronization is complete and consequently it is expected that $C_5 = 0$.

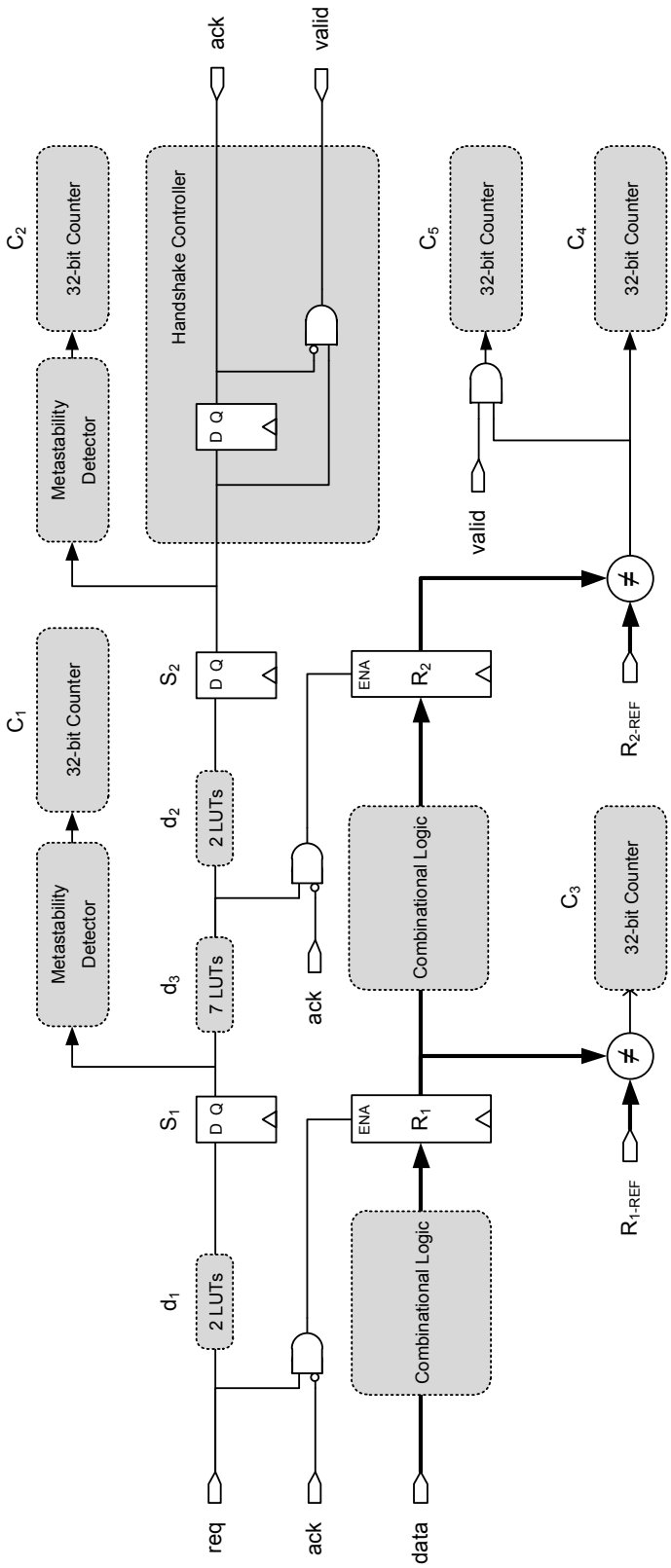


Figure 3.16: Benchmark system

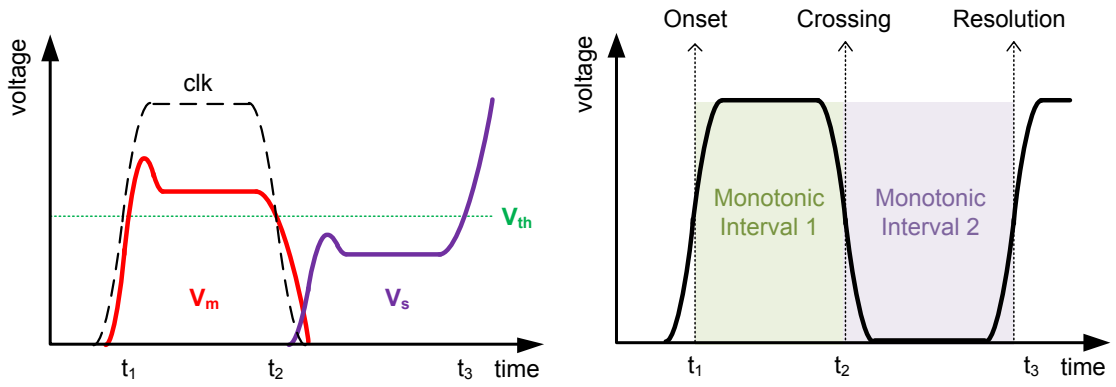
Table 3.2: Counter values after benchmark

Counter	Description	Value
C_1	prolonged clock-to-q delays of S1	220
C_2	prolonged clock-to-q delays of S2	5
C_3	corrupt data latched by R1	13187293
C_4	corrupt data latched by R2	120
C_5	actual pipeline errors	0

One difficulty with this characterization is that τ is very small and so a two-cycle synchronization at $f_c = 400$ MHz amounts to a MTBF of greater than 10^{30} years. This means that it would have been practically impossible to observe any metastable events at the output of S_2 or data corruption events in R_2. To circumvent this difficulty (i.e. to increase the number of metastable events), another delay element d_3 was inserted between S_1 and S_2. The delay d_3 takes away considerably from the settling time of S_2 without affecting the functionality of the pipeline (it can be considered internal to S_1). By carefully adjusting d_3 and the clock period, the settling time of S_2 was minimized and the MTBF of the synchronizer was reduced to a few seconds.

Table 3.2 presents the values of the counters C_1 through C_5 after one particular benchmark run whose length is 25 seconds ($f_c = 370.4$ MHz, data rate $= 150$ MHz). The results demonstrate that, despite the prolonged clock-to-q delays of S_1 and S_2 (C_1, $C_2 \neq 0$) and the resulting data corruption (C_3, $C_4 \neq 0$), the pipeline output is correct every time synchronization is complete ($C_5 = 0$). Also, the number of data corruption events (C_3 + C_4) represents a small fraction of the number of handshakes performed in this benchmark:

$$\text{Corruption Events (\%)} = \frac{13187293 + 120}{25 \times 150 \times 10^6} \times 100\% = 0.35\% \tag{3.9}$$



(a) Master and slave latch voltages

(b) Flip-flop output

Figure 3.17: Monotonic intervals

3.4.5 Monotonicity

The pipeline behavior described by Theorem 3.1 requires each synchronizer flip-flop S_i to behave monotonically. Section 2.3 discusses different scenarios in which metastable flip-flops do not behave as such. Nonetheless, monotonicity can still be preserved by elaborate flip-flop design and by confining the sampling period of the flip-flop output to certain "safe" intervals.

Unlike the flip-flop output transition that occurs due to the resolution of metastability, the transitions due to its onset and propagation to the slave latch have predictable timing. Therefore, a monotonic response can be obtained by simply ensuring that the sampling interval of the flip-flop output does not contain these predictable transition points. This is illustrated in Figure 3.17. In the given example, the onset of metastability causes a transition at t_1, its crossing to the slave latch causes a second transition at t_2 and finally its resolution causes a third transition at t_3. While the final transition time t_3 is arbitrary, t_1 and t_2 are independent of the deepness of the induced metastable state and can be pre-determined. Therefore, there exists two intervals in which the flip-flop output is monotonic: $[t_1, t_2]$ and $[t_2, T]$ (where T is the clock period). Any series of flip-flop output samples collected within one of these intervals will have at most one transition. In summary, while constraining the final transition time is fundamentally impossible, predicting the others is just a design problem and a technological difficulty.

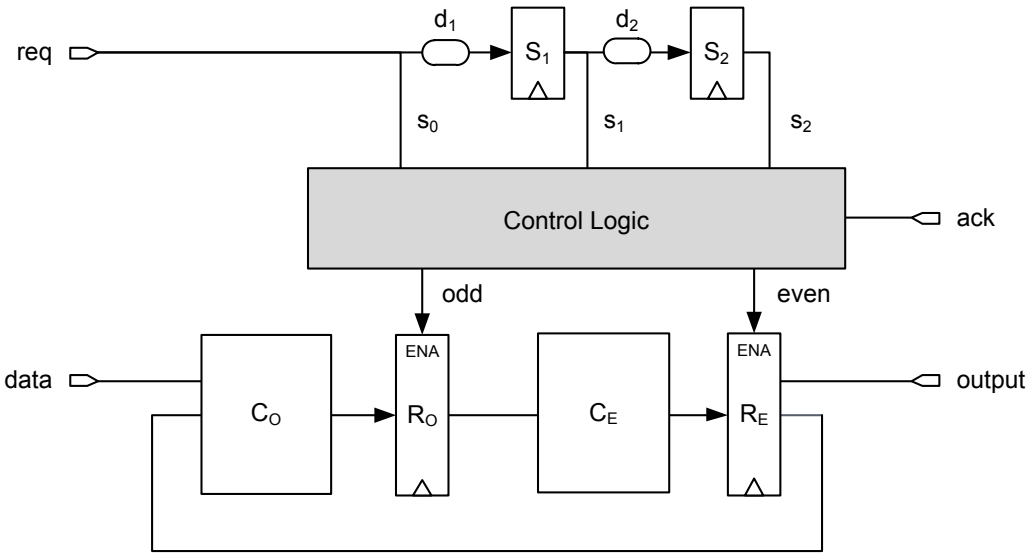


Figure 3.18: Cyclic pipeline and sequenced latching control logic

3.5 Sequenced Latching: Non-pipelined Designs

3.5.1 Overview

The previous section describes a scheme to sequence a series of pipelined latching operations reliably during synchronization cycles. The scheme cannot be applied to non-pipelined designs because the intermediate latching failures can cause irreversible state corruption. This difficulty can be evaded by unfolding the design into an n-stage pipeline for n synchronization cycles. However, since corrupt pipeline stage data are automatically re-latched before enabling the next stages, an n-stage pipeline is not actually necessary. Instead, a 2-stage cyclic pipeline is sufficient.

This section extends the sequenced latching scheme to arbitrary designs by presenting a method to reliably sequence a set of latching operations in 2-stage cyclic pipeline. The solution involves unfolding the design into a functionally equivalent cyclic pipeline consisting of two stages and enabling them alternately as demonstrated in Figure 3.18. In the example, a generic Moore machine $\{C, R\}$ has been unfolded into a cyclic pipeline consisting of two instances $\{C_O, R_O\}$ and $\{C_E, R_E\}$ (referred to as the odd and even instances respectively). Two signals, *odd* and *even* are derived from the intermediate

Table 3.3: Truth table of *odd* and *even* (two-phase handshake)

S_0 (*req*)	S_1	S_2	*odd*	*even*	Synchronization Cycle
0	0	0	0	0	-
1	0	0	1	0	1
1	1	0	0	1	2
1	1	1	0	0	-
0	1	1	1	0	1
0	0	1	0	1	2
0	0	0	0	0	-

synchronizer nodes: *odd* is asserted during the odd cycles of synchronization while *even* is asserted during the even cycles (Table 3.3). For a two-phase handshake protocol, the functions *odd* and *even* are implemented as:

$$odd = \sum_{i \in (0,2,\ldots,n)} (S_i \oplus S_{i+1}) \tag{3.10}$$

$$even = \sum_{i \in (1,3,\ldots,n-1)} (S_i \oplus S_{i+1}) \tag{3.11}$$

where $S_0 = req$.

While a *req* transition propagates through the synchronizer, *odd* and *even* are asserted in alternating cycles. The design instances are thus enabled alternately and complete a number of state transitions equal to the number of synchronization cycles. The propagation delays of the paths $S_i \rightarrow odd$ and $S_i \rightarrow even$ are constrained relative to $S_i \rightarrow S_{i+1}$ to satisfy two conditions. First, if the delayed transition of a synchronizer flip-flop S_i corrupts the following state, the synchronizer will not change in that cycle. This will cause a stall and allow the cyclic pipeline to re-latch the following state. Second, the existing state (in the other instance) is not corrupted during a stall cycle.

What remains of this subsection will give an intuitive interpretation of the behavior described above (using ball-and-hill analogy), describe an example handshake and then present a logical proof of the correctness of the scheme.

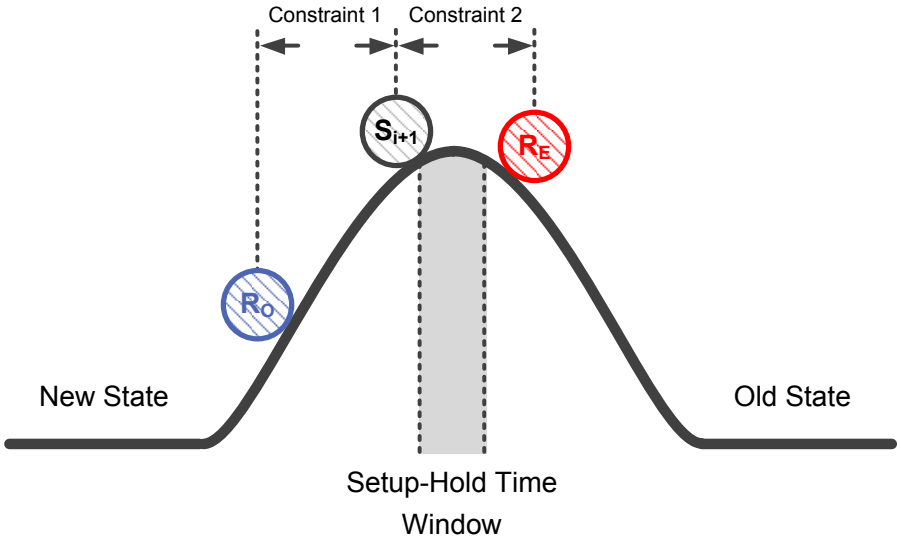


Figure 3.19: Ball and hill analogy of the decision in an odd cycle

The synchronizer decision in an odd cycle (even i) is illustrated graphically using the ball and hill system in Figure 3.19. Here, the initial position of three balls represent the arrival time of S_i's transition at each of R_O, S_{i+1} and R_E on every odd cycle.

The behavior of the cyclic pipeline in odd cycles can be described using the lemmas:

Lemma 3.2. *When the difference between the arrival time of S_i's transition at S_{i+1} and R_O is constrained* [7]*: if S_{i+1} captures S_i (even i) then R_O does the same, safely.*

Lemma 3.3. *When the difference between the arrival time of S_i's transition at S_{i+1} and R_E is constrained* [8]*: if S_{i+1} does not capture S_i (even i) then R_E does the same, safely.*

Thus, at every odd cycle, the synchronizer can either transition to an even state or not [9]. If it does, the setup condition of R_E is met and the new state latched by R_E is correct. If not, the setup condition of R_O is met and the existing state re-latched by R_O is correct.

Using the ball and hill analogy, if S_{i+1} rolls to the new state then so does R_O while if it rolls to the old state then so does R_E.

[7] this constraint is placed making the propagation delay of the path $S_i \rightarrow S_{i+1}$ sufficiently larger than that of $S_i \rightarrow R_O$
[8] this constraint is placed making the propagation delay of the path $S_i \rightarrow R_E$ sufficiently larger than that of $S_i \rightarrow S_{i+1}$
[9] it does when the transition of S_i is captured by S_{i+1}

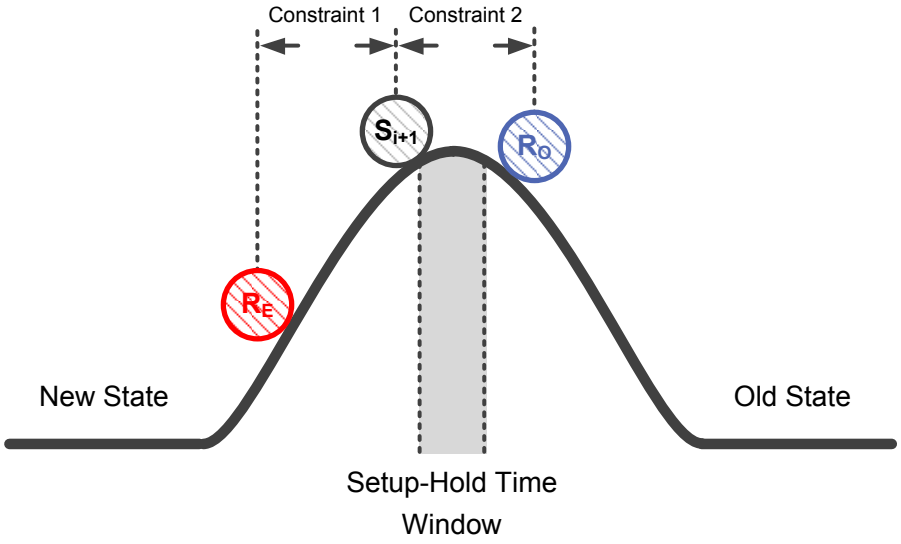


Figure 3.20: Ball and hill analogy of the decision in an even cycle

A symmetrical decision is made in even cycles (odd i) as illustrated in Figure 3.20. Again, the initial position of three balls represent the arrival time of S_i's transition at each of R_O, S_{i+1} and R_E on every even cycle.

The behavior of the cyclic pipeline in even cycles can be described using the lemmas:

Lemma 3.4. *When the difference between the arrival time of S_i's transition at S_{i+1} and R_E is constrained* [10]*: if S_{i+1} captures S_i (odd i) then R_E does the same, safely.*

Lemma 3.5. *When the difference between the arrival time of S_i's transition at S_{i+1} and R_O is constrained* [11]*: if S_{i+1} does not capture S_i (odd i) then R_O does the same, safely.*

Thus, at every even cycle, the synchronizer can transition to an odd state or not [12]. If it does, the setup condition of R_O is met and the new state latched by R_O is correct. If not, the setup condition of R_E is met and the existing state re-latched by R_E is correct.

Again, using the ball and hill analogy, if S_{i+1} rolls to the new state then so does R_E while if it rolls to the old state then so does R_O.

[10] this constraint is placed making the propagation delay of the path $S_i \rightarrow S_{i+1}$ sufficiently larger than that of $S_i \rightarrow R_E$
[11] this constraint is placed making the propagation delay of the path $S_i \rightarrow R_O$ sufficiently larger than that of $S_i \rightarrow S_{i+1}$
[12] it does when the transition of S_i is captured by S_{i+1}

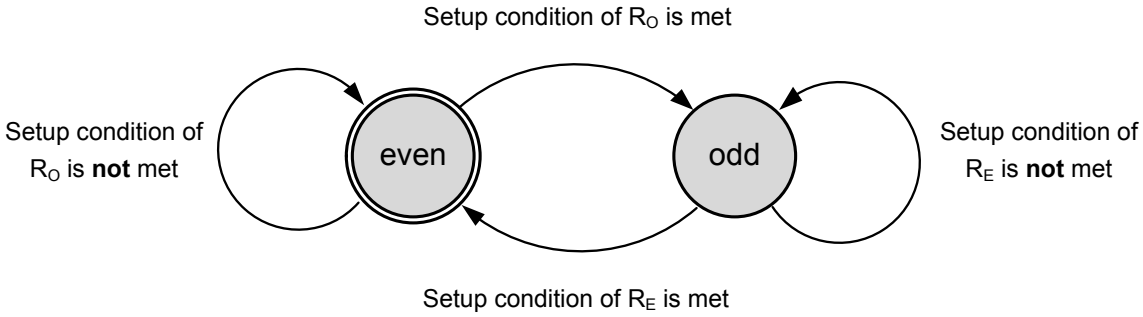


Figure 3.21: Cyclic pipeline state diagram

The behavior of the cyclic pipeline in all cycles can now be described using Lemmas 3.2, 3.3, 3.4 and 3.5 as follows:

Theorem 3.2. *At any cycle following the transition of req, a single flip-flop S_i may be metastable. For even i, the pipeline attempts to use the existing state in R_E to compute R_O (odd cycle): if S_{i+1} captures the new (post-transition) value of S_i, the new state is latched by R_O correctly. If not, the old state is latched by R_E correctly. For odd i, the pipeline attempts to use the existing state in R_O to compute R_E (even cycle): if S_{i+1} captures the new (post-transition) value of S_i, the new state is latched by R_E correctly. If not, the old state is latched by R_O correctly.*

This behavior is captured by the state diagram in Figure 3.21.

The presented implementation assumes that the number of state transitions per handshake (m) is even. Therefore, the machine state is always stored in the same register every handshake (R_E in the proposed notation). For odd m, the control block must keep track of where the machine state is kept at the end of each handshake (either R_E or R_O). The presented implementation also assumes that the machine latches *data* on the first cycle after the transition of *req*. Therefore, *data* is connected to the C_O only. If this is not the case, *data* must be connected to C_E. This does not affect the correctness of the method.

3.5.2 Example

The handshake example in Figure 3.22 illustrate how the cyclic pipeline in Figure 3.18 behaves when a metastable state manifests. In this handshake, the sender makes *data* available on the bus and asserts *req* a sufficient time later (bundled data constraint). *req*

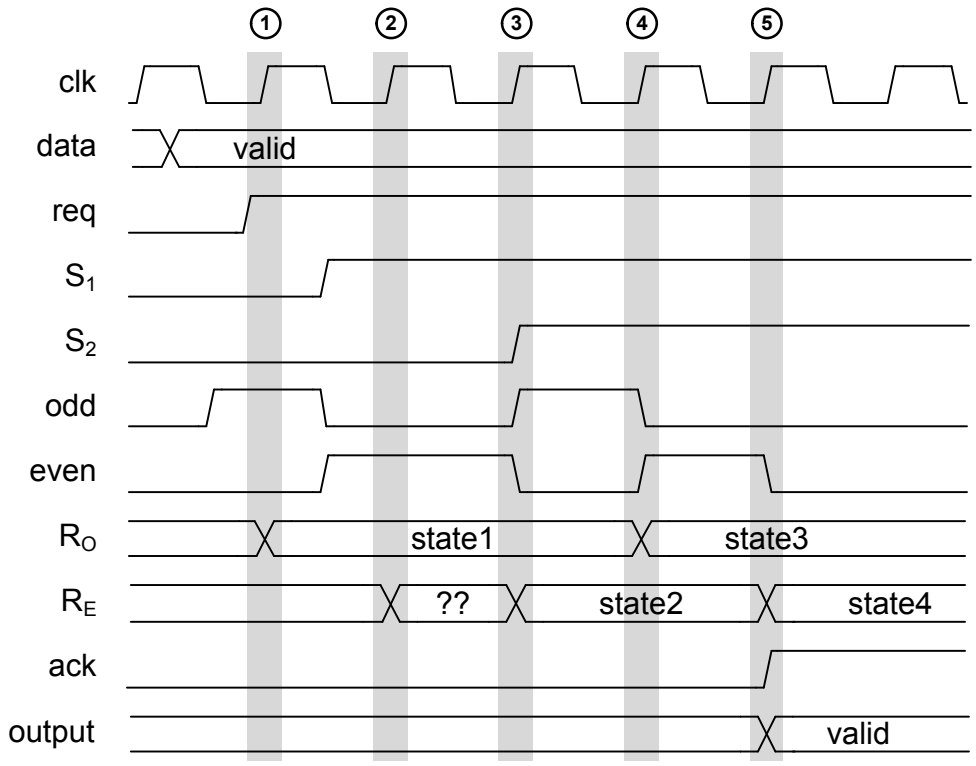


Figure 3.22: Handshake example 2

arrives close to clock edge 1 and causes S_1 to become metastable. In the meanwhile, *data* has arrived sufficiently earlier and is latched by R_O correctly.

During the first synchronization cycle (following clock edge 1), the prolonged clock-to-q delay of S_1 causes a violation of the setup condition of the path $S_1 \rightarrow R_E$ and corrupts the state latched by R_E. However, this delayed transition is not captured by S_2 on clock edge 2 (Lemma 3.4). Consequently, the synchronizer remains in the state ($\{S_0, S_1, S_2\} = 110$) and does not transition to the state ($\{S_0, S_1, S_2\} = 111$). Also, the delayed transition of S_1 is safely not captured by R_O (Lemma 3.5). Therefore, the state of R_O (*state*1) remains unchanged. In the following cycle, *even* remains asserted and R_E re-latches *state*2 correctly. In the two subsequent cycles, *state*3 and *state*4 are latched by R_O and R_E respectively. The handshake is then complete; *ack* is asserted by the control block and $output$ is valid.

3.5.3 Proof of Correctness

Lemmas 3.2, 3.3, 3.4 and 3.5 are very similar to Lemma 3.1 whose proof is provided in Subsection 3.4.3. For the sake of completeness, their proofs are listed below in full:

Lemma 3.2

For even i, if flip-flop S_{i+1} captures the transition of S_i at t_{S_i} then [13]:

$$t_{S_i} + t_{pd}(S_i \rightarrow S_{i+1}) < t_{\text{clk}} + t_{\text{h}}(S_{i+1}) \tag{3.12}$$

where t_{clk} is the time of the clock edge, $t_{\text{h}}(k)$ is the hold time of flip-flop k and $t_{pd}(k_1 \rightarrow k_2)$ is the propagation delay of the path $k_1 \rightarrow k_2$.

Now, t_{S_i} meets the setup constraint of R_O when:

$$t_{S_i} + t_{pd}(S_i \rightarrow R_O) < t_{\text{clk}} - t_{\text{su}}(R_O) \tag{3.13}$$

where $t_{\text{su}}(k)$ is the setup time of flip-flop/register k.

If the delay element d_i is adjusted such that:

$$t_{pd}(S_i \rightarrow S_{i+1}) - t_{pd}(S_i \rightarrow R_O) > t_{\text{h}}(S_{i+1}) + t_{\text{su}}(R_O) \tag{3.14}$$

then Inequality 3.12 will imply Inequality 3.13 (using the same logic in Subsection 3.3.2). Therefore, a change in the state of S_{i+1} (for even i) will imply that R_O captured the transition of S_i safely.

[13] using Primitive 1 (Subsection 2.7.2)

Lemma 3.3

For even i, if flip-flop S_{i+1} does not capture the transition of S_i at t_{S_i} then [14]:

$$t_{S_i} + t_{pd}(S_i \rightarrow S_{i+1}) > t_{\text{clk}} - t_{\text{su}}(S_{i+1}) \tag{3.15}$$

Now, R_E does not capture the transition of S_i, safely, when:

$$t_{S_i} + t_{pd}(S_i \rightarrow R_E) > t_{\text{clk}} + t_{\text{h}}(R_E) \tag{3.16}$$

If the delay element d_i is adjusted such that:

$$t_{pd}(S_i \rightarrow R_E) - t_{pd}(S_i \rightarrow S_{i+1}) > t_{\text{h}}(R_E) + t_{\text{su}}(S_{i+1}) \tag{3.17}$$

then Inequality 3.15 will imply Inequality 3.16 (using the same logic in Subsection 3.3.2). Therefore, a no-change in the state of S_{i+1} (for even i) will imply that R_E did not capture the transition of S_i, safely.

Lemma 3.4

For odd i, if flip-flop S_{i+1} captures the transition of S_i at t_{S_i} then [15]:

$$t_{S_i} + t_{pd}(S_i \rightarrow S_{i+1}) < t_{\text{clk}} + t_{\text{h}}(S_{i+1}) \tag{3.18}$$

Now, t_{S_i} meets the setup constraint of R_E when:

$$t_{S_i} + t_{pd}(S_i \rightarrow R_E) < t_{\text{clk}} - t_{\text{su}}(R_E) \tag{3.19}$$

If the delay element d_i is adjusted such that:

$$t_{pd}(S_i \rightarrow S_{i+1}) - t_{pd}(S_i \rightarrow R_E) > t_{\text{h}}(S_{i+1}) + t_{\text{su}}(R_E) \tag{3.20}$$

then Inequality 3.18 will imply Inequality 3.19 (using the same logic in Subsection 3.3.2). Therefore, a change in the state of S_{i+1} (for odd i) will imply that R_E captured the transition of S_i safely.

[14] using Primitive 2 (Subsection 2.7.2)

[15] using Primitive 1 (Subsection 2.7.2)

Lemma 3.5

For odd i, if flip-flop S_{i+1} does not capture the transition of S_i at t_{S_i} then [16]:

$$t_{S_i} + t_{pd}(S_i \rightarrow S_{i+1}) > t_{\text{clk}} - t_{\text{su}}(S_{i+1}) \tag{3.21}$$

Now, R_O does not capture the transition of S_i, safely, when:

$$t_{S_i} + t_{pd}(S_i \rightarrow R_O) > t_{\text{clk}} + t_{\text{h}}(R_O) \tag{3.22}$$

If the delay element d_i is adjusted such that:

$$t_{pd}(S_i \rightarrow R_O) - t_{pd}(S_i \rightarrow S_{i+1}) > t_{\text{h}}(R_O) + t_{\text{su}}(S_{i+1}) \tag{3.23}$$

then Inequality 3.21 will imply Inequality 3.22 (using the same logic in Subsection 3.3.2). Therefore, a no-change in the state of S_{i+1} (for even i) will imply that R_O did not capture the transition of S_i, safely.

3.6 Comparison of Speculative Techniques

3.6.1 What is speculated?

Speculative synchronization [2], datapath unfolding (Section 3.3) and sequenced latching (Sections 3.4 and 3.5) are different speculative methods for hiding synchronization latency. In each of these techniques, an underlying speculation – an *assumption* – enables the asynchronous receiver to begin data processing immediately without waiting for the handshake request to be synchronized. The assumptions hold in most synchronization attempts and so the speculative techniques provide near-zero average latency.

Since metastability is a fundamental attribute of asynchronous communication, all three methods require hardware duplication to tolerate the inevitable cases when metastability causes data/state corruption. However, each method does so differently. This section compares the three speculative techniques both qualitatively and quantitatively and starts by answering the following questions for each case: *what is speculated?* and *what happens when a misspeculation takes place?*.

[16] using Primitive 2 (Subsection 2.7.2)

Speculative Synchronization

This method uses a single flip-flop k to synchronize the asynchronous handshake and a detector to reliably identify, few cycles later, whether k has become metastable. This is true for most handshakes and so the average latency is reduced to little above 1 cycle. An n-level stack is added to the machine state register to keep state backups. When a metastable state is identified (a misspeculation), the machine is restored to a previous correct state and few cycles are wasted in re-computation. This approach can be summarized as "assume, execute, verify then correct if necessary".

Datapath Unfolding

In datapath unfolding, additional instances of the entire machine (both combinational logic and state register) are used to speculatively compute the machine states following the arrival of data. The assumption used here is that the value of the asynchronous data bus was valid n cycles earlier. No cost is incurred in the case of misspeculation because speculative states are not committed into the actual machine unless the assumption has been known to hold. This approach can be summarized as "assume, verify then execute".

Sequenced Latching

Unlike the other two speculative methods, sequenced latching makes an individual assumption on each synchronization cycle. The assumption is that the transition of the synchronizer flip-flop S_i is captured by its successor S_{i+1}. The delays between the synchronizer flip-flops and the sequenced pipeline stages are constrained such that data moves through the pipeline safely when this assumption holds. In the case of misspeculation, the pipeline is stalled for an additional cycle to re-latch the register that contains corrupt data with correct data values.

3.6.2 Area, Power and Reliability Costs

The speculative techniques are compared in Table 3.4. For a Moore machine composed of a combinational block C, a state register R and an n-stage synchronizer, datapath unfolding requires n machine duplicates ($n + 1$ instances), sequenced latching requires

only one duplicate (2 instances) while speculative synchronization requires n duplicates of R but none of C.

Although datapath unfolding and sequenced latching hide synchronization by overlapping it with computation cycles, there is a hidden latency cost due to introducing the combinational block C_1 at the input of the first data register R_1. The bundling delay t_{b} must increase by the worst propagation delay through the block C_1 (let this be t_{C1}).

The dynamic power costs of datapath unfolding and speculative synchronization are proportional to the amount of duplicated resources. On the other hand, the duplication power overhead of sequenced latching is negligible because the odd and even instances of R are enabled alternately and so their power consumption is equal to that of a single instance (assuming the average switching activity remains the same). However, there is an additional power overhead of re-latching the corrupt states. An upper bound $P_{\text{re-latch}}$ on this overhead can be expressed as:

$$P_{\text{re-latch}} = P_s \times \frac{t_{\mathrm{su}}(R_1) + t_{\mathrm{h}}(S_1)}{T} \tag{3.24}$$

where P_s is the average power consumed by a state transition.

This is because the overhead of re-latching is dominated by failed attempts in the first stage (R_1) which occur when R_1 captures *req* but S_1 does not. In reality, the actual power overhead of re-latching is much smaller than the upper bound ($< 0.4\%$ in the presented benchmarks).

The usage of delay elements in sequenced latching decreases synchronization time and the MTBF of synchronization. The latter can be expressed as:

$$\mathrm{MTBF}_{\mathrm{SL}} = \frac{e^{(ts-td)/\tau}}{f_c \times f_d \times T_w} \tag{3.25}$$

where td is the sum of the delays inserted between the synchronizer flip-flops.

The time td is of the order of few gate delays and does not take away much from ts. Therefore, the MTBF drop is small and is not expected to violate common MTBF requirements. This assertion is investigated quantitatively in Subsection 3.6.3.

Table 3.4: Cost complexity comparison of speculative methods

Method	Hardware Duplication Costs				MTBF Cost	Misspeculation Penalty	Latency
	Combinational		Sequential				
	Area	Power	Area	Power			
Datapath Unfolding	$O(n)$	$O(n)$	$O(n)$	$O(n)$	0	0	$t_{\mathrm{b}} + t_{C1}$
Speculative Synchronisation	0	0	$O(n)$	$O(n)$	0	n cycles	$t_{\mathrm{b}} + T$
Sequenced Latching	$O(1)$	negligible	$O(1)$	negligible	small	0	$t_{\mathrm{b}} + t_{C1}$

3.6.3 Synthesis Results

This section presents a quantitative comparison between the area, power and reliability costs of the three speculative techniques. Cost figures are drawn from synthesizing and applying the three techniques individually to each of the datapaths listed in Table 3.5. The designs were synthesized using the Nangate 45nm Open Cell library [78] for a target clock frequency of 1 GHz at 1.1 V supply voltage and 25 ° C junction temperature.

Figure 3.23 compares the area and power overheads of the speculative techniques. Speculative synchronization achieves significant savings over datapath unfolding in both area and power, particularly for datapaths of large combinational resources. Sequenced latching achieves further savings in area and eliminates the power overhead of speculation. Furthermore, the overheads of both datapath unfolding and speculative synchronization increase with the number of synchronization cycles. In contrast, sequenced latching has fixed overheads and thus becomes more cost-effective for larger numbers of synchronization cycles. This is demonstrated in Figure 3.24.

Table 3.5: Benchmark datapaths

Datapath	Description	Area (μm^2)			Dynamic Power (μW)		
		Combinational	Sequential	Total	Combinational	Sequential	Total
`counter`	8-bit binary counter	46.6	36.2	82.8	10.8	21.4	32.2
`multi`	8-bit by 8-bit multiplier	1138.1	108.5	1246.8	424.1	72.7	496.8
`crc8`	4-bit CRC, 8-bit data item	185.9	149.2	335.1	74.6	112.6	187.2
`crc16`	4-bit CRC, 16-bit data item	316.1	257.8	573.9	99.0	250.7	349.7
`fir3`	3^{rd} order FIR filter, 8-bit data item	381.2	149.2	530.4	110.1	82.7	192.8
`fir5`	5^{th} order FIR filter, 8-bit data item	909.4	221.6	1131	281.1	118.9	400
`fsm16`	finite state machine with 16 states	13.1	22.6	35.7	4.7	28.2	32.9
`fifo16`	16-slot FIFO, 8-bit data item	364.9	578.8	943.7	51.0	274.7	325.7

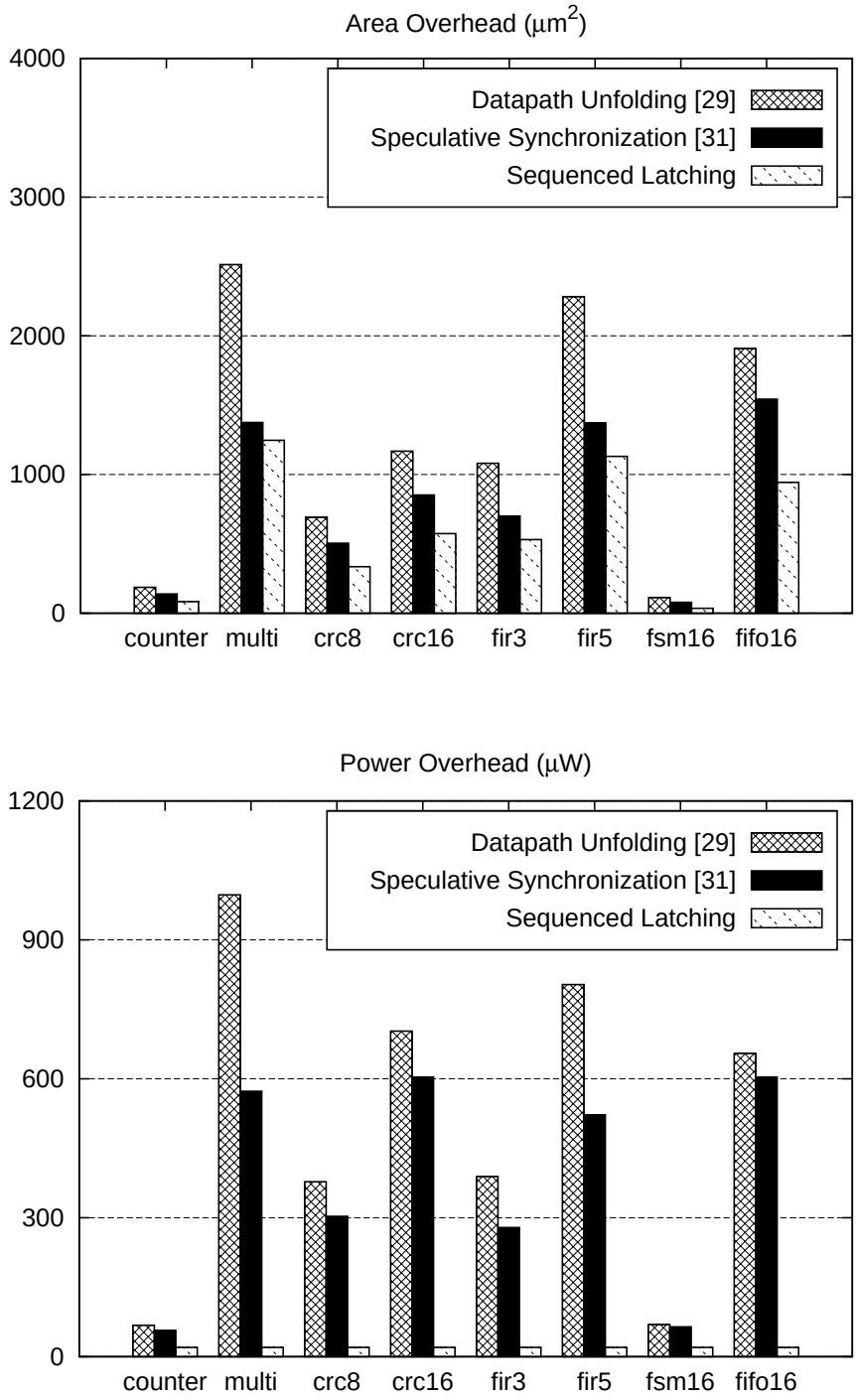


Figure 3.23: Area and power overhead comparison (2 synchronization cycles)

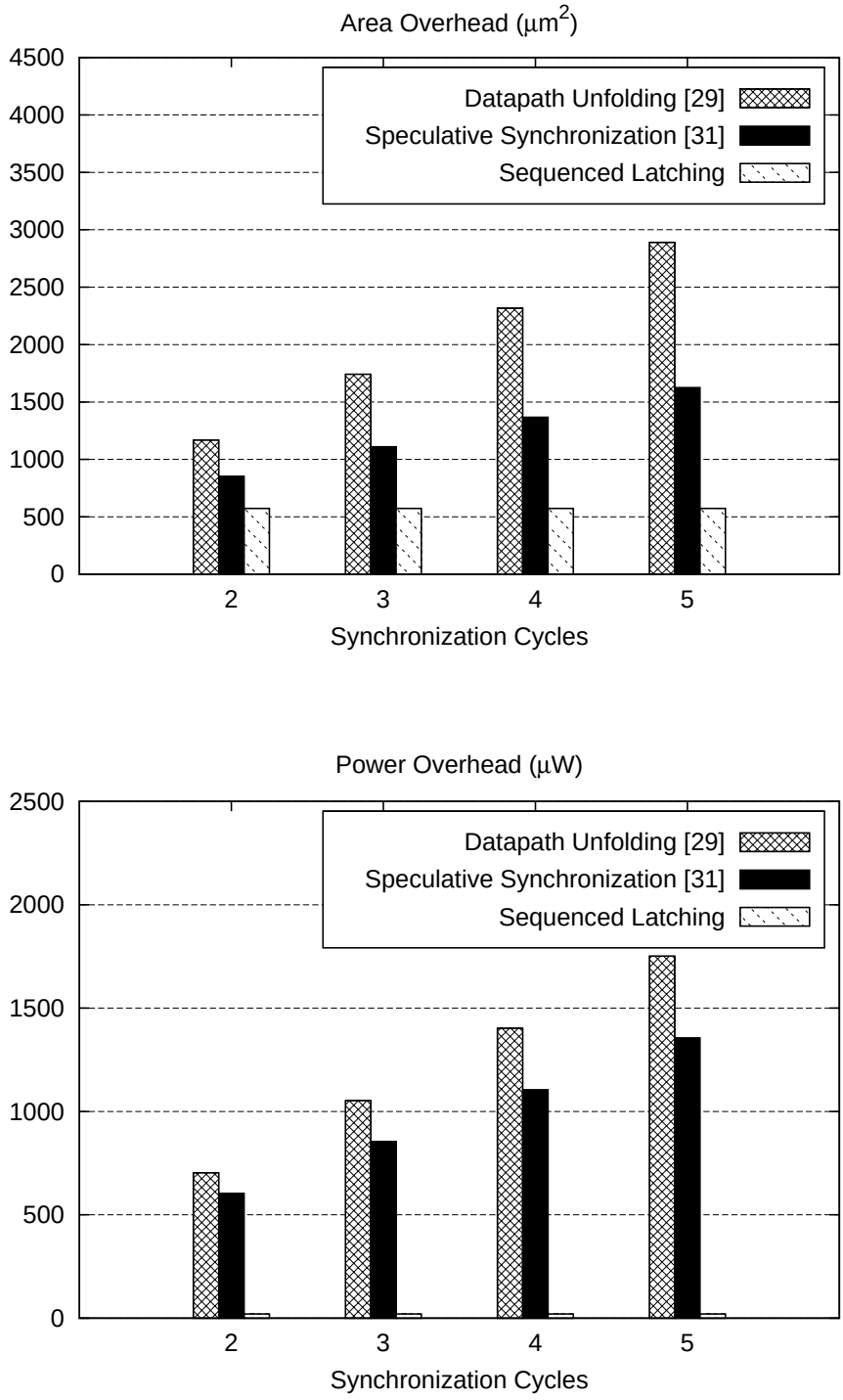


Figure 3.24: Area and Power overhead complexity comparison (datapath = `crc16`)

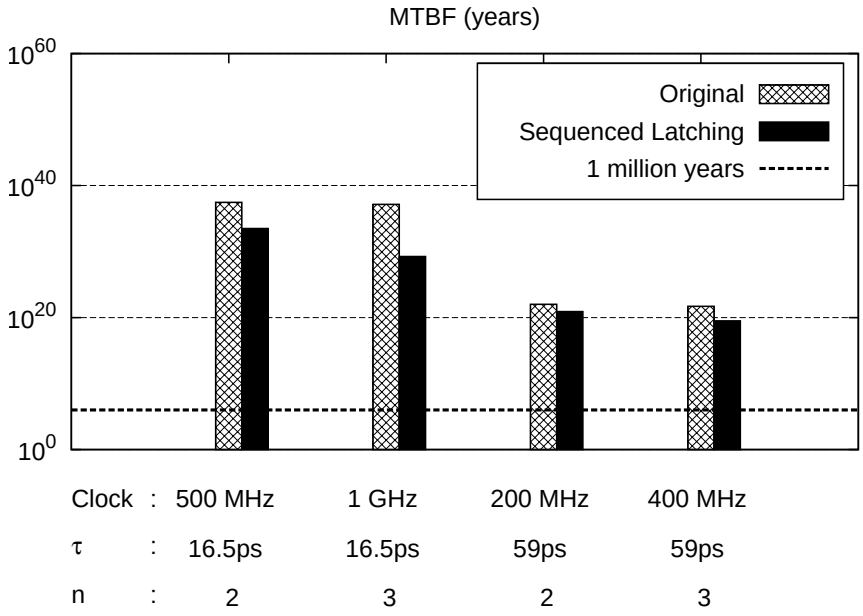


Figure 3.25: MTBF of sequenced latching

Figure 3.25 compares the MTBF of synchronization before and after adding the delay elements required to meet the sequenced latching propagation delay constraints. In each case, the MTBF has been calculated taking $f_d = 100$ MHz and $T_w = 1$ns and using delays of 150ps between the synchronizer stages. The values of τ used in this evaluation are those of the Nangate library flip-flop `DFFR_X1` under the typical and slow process corners.

The data shows that the MTBF of synchronizers in modern technologies is exceptionally high. Hence, the deduction of a relatively small delay (150ps) of synchronization time per cycle does not cause a violation of a typical MTBF requirement (10^6 years).

3.7 Design for Speculative Synchronization

This chapter presented two novel speculative methods that exploit hardware duplication to hide synchronization latency. Section 3.3.5 presented a method to identify the subset of the machine's state register whose value needs not be speculated and whose logic can thus be excluded from duplication. While this has been suggested in the course of discussing the first speculative technique, datapath unfolding, it can equally be applied to the second, sequenced latching.

In what preceded of discussions, the synchronous Moore machine whose state is being speculated during synchronization is assumed immutable. However, if speculation was taken into consideration during the design of the machine, further reductions in hardware duplication costs may be achieved by deliberately structuring the design to support state speculation. Recall that the duplication cost reduction method described in Section 3.3.5 involves traversing the flip-flop dependency graph of the design and locating all combinational and sequential components within a 2-cycle distance of the asynchronous request input. If large resources (memory units and complex arithmetic circuits) were moved out of this subset (by avoiding their utilization during the early cycles of data arrival), these resources need not be duplicated and the overall duplication cost is reduced. The designer can thus pro-actively design the machine to support state speculation in what can be referred to as *Design for Speculative Synchronization*.

Of course a synchronous machine cannot be restructured freely since component interconnectivity is derived from the machine's functional specifications. For example, a large lookup table cannot be moved out of the "speculation zone" if it needs to be accessed immediately following the arrival of data. Therefore, the room for modification at the circuit-level is usually limited. Instead, design for speculation can be practiced more effectively at higher abstraction levels, particularly during the formulation of the design specifications. For example, all other things being equal, a Network-on-Chip (NoC) designer can favor routing algorithms with trivial early data consumption operations to support speculation and enable the whole NoC to benefit from low-latency communication at a small duplication cost.

3.8 Conclusion

The growing number of asynchronously-clocked cores in modern systems means that the negative performance impact of clock domain crossing latency is likely to increase. Existing solutions are limited to the cases where the communicating clocks have dependable timing relationships or rely on pausible locally-generated clocks which have poor stability and require design modification. This chapter presented two novel architectural solutions (datapath unfolding and sequenced latching) that are free from these limitations. The proposed methods leverage hardware duplication to speculatively compute the first few system states following a change in the asynchronous input. This allows a system to hide synchronization latency by overlapping it with the computation of the first few data-dependent states.

Synthesis results drawn from automating datapath unfolding via an RTL tool demonstrate that the duplication costs for a number of benchmarks are significantly smaller than the area of a periodic synchronizer. More importantly, the method outperforms existing clock domain crossing approaches by being *seamless* and *transparent*. In other words, it does not require modifying other steps of the design flow nor the behavioral description of the processed design.

The second method, sequenced latching, uses the synchronizer state to sequence the latching of data during synchronization cycles and automatically re-latch any data that had been corrupted. The method has been verified in practice by implementing it on an FPGA and demonstrating that it results in correct behavior under persistent manifestation of metastable states. In comparison with datapath unfolding and another speculative method which appears in the literature, sequenced latching is superior in several aspects: it provides shorter latency, smaller area overhead (which also does not increase with the number of synchronization cycles) and negligible power overhead.

Chapter 4

Adaptive Synchronization for DVFS

4.1 Synchronization under DVFS

Simple brute-force synchronizers remain the most popular method of interfacing multiple clock domains in practice. Latency-insensitive designs or those consisting of few clock domains favor synchronization over more complex schemes such as pausable clocking or correlated clocks. However, despite the abundance of metastability characterization and latch performance reports in literature, direct recommendations for the length of a "reliable" synchronizer chain are non-existent. The reason for this is that the answer is very sensitive to the technology particulars and the working conditions of the design (e.g. clock frequency, supply voltage, temperature and process variations).

Designers are ultimately interested in knowing the MTBF of the basic synchronizer chains (Figure 2.14) in their technology and working conditions so they can pick the optimal (lowest latency) chain that meets their MTBF criteria. However, this is almost never an easy task. First, cell libraries do not provide this information. Second, results from characterization reports of other or similar technologies are informative but can not be mapped with absolute certainty. Third, obtaining this information via simulation or on-chip measurements is both an expensive and a slow process.

The situation is more complex in systems that operate under non-nominal conditions such as lower supply voltages. Conventional rules of thumb that help designers produce rough MTBF estimates can not be used in the design of these systems. An example rule of

thumb which has been shown to be inaccurate at lower supply voltages is approximating τ as the FO4 delay of technology [79]. Without even these rough guidelines, synchronizer reliability is hard to estimate and must be explicitly evaluated [80]. Two further complications arise in designs that have multiple operating points such as those that support Dynamic Voltage and Frequency Scaling (DVFS). First, synchronizer reliability is sensitive to changes in the operating point. Second, this scaling is highly-dependent on flip-flop design and cannot be predicted without elaborate analysis (e.g. the latches proposed in [79] [81] have better voltage scaling characteristics than typical designs). Therefore, while conventional designs face the difficulty of evaluating synchronizer reliability at a single operating point, DVFS systems must do the same at several operating points.

This chapter presents a cost-effective and practical solution for optimizing the length of a synchronizer chain in a DVFS system. The solution relies on evaluating the ratio τ/FO4 dynamically after every change in the operating point and using this information to select the minimum-latency synchronizer that meets the MTBF criterion of the system from four built-in synchronizers.

4.2 The Scaling of Synchronizer Reliability

The dynamic scaling of voltage and frequency is one of the most ubiquitous methods of reducing power consumption, particularly in tablet and mobile phone SoCs which run on limited supplies and have highly-variable workloads. Reducing the supply voltage and frequency linearly results in cubic reduction in dynamic power consumption following the relationship ($P = \alpha C V^2 f$). Supported systems dynamically transition between multiple Voltage/Frequency (VF) points which are pre-defined based on the supply voltage to propagation delay relationship of the critical path of the system.

VF scaling affects several of the parameters in the synchronizer MTBF formula (Equation 2.11) but its impact on the exponential term t_s/τ is the most significant. Synchronizer chains provide a settling time t_s which is a multiple m of the clock period T. In a DVFS system, T is constrained by the critical path delay of the design at all VF points and so it can be expressed as a fixed multiple n of the FO4 delay of the technology. Thus,

synchronization time can be expressed as:

$$t_s = m \times n \times \text{FO4} \tag{4.1}$$

Since t_s is, in fact, a design and synchronizer-specific multiple of the FO4 delay, the ratio t_s/τ is also a multiple of the ratio FO4 / τ and has the same supply voltage dependency. To evaluate how the ratio FO4/τ scales with the supply voltage V, consider the small-signal models of both the FO4 delay and τ. Assuming square law devices, the FO4 delay can be expressed as [10]:

$$\text{FO4} = \frac{C_L V}{I} \propto \frac{V}{(V - V_{\text{th}})^2} \tag{4.2}$$

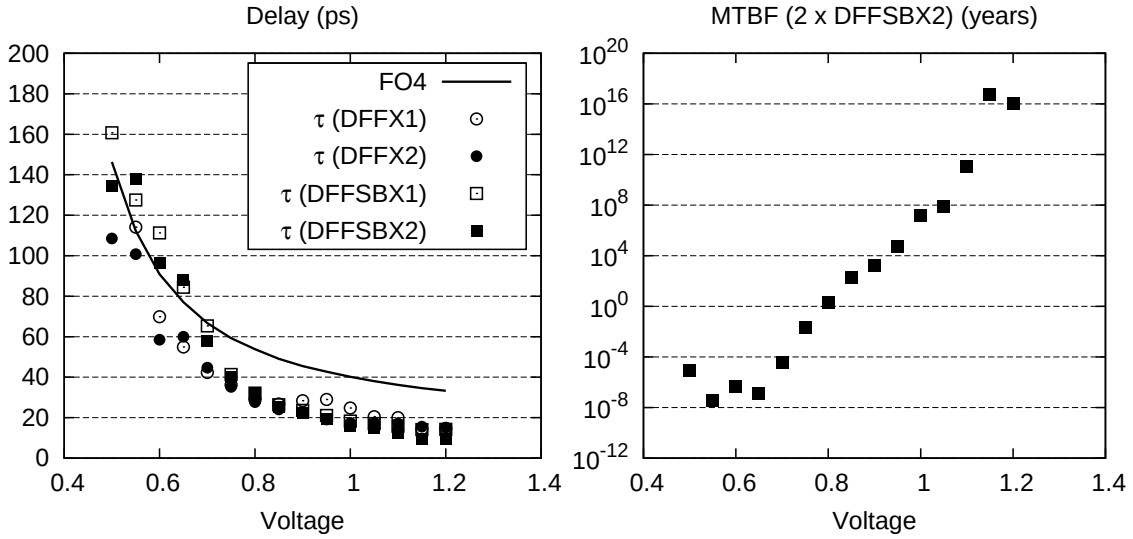
where C_L is the input capacitance of an inverter, I is the drive current of a 4X-smaller inverter and V_{th} is the threshold voltage of the technology.

Similarly, for a cross-coupled inverter pair [10]:

$$\tau = \frac{C_m}{g_m} \propto \frac{1}{(V - 2V_{\text{th}})} \tag{4.3}$$

where C_m is the bistable node capacitances and g_m is the transconductance of the cross-coupled inverters.

Equations 4.2 and 4.3 show that the FO4 delay and τ do not scale proportionately with the supply voltage. The FO4 delay function has a pole at $V = V_{\text{th}}$ while τ has one at $V = 2V_{\text{th}}$. This is because metastability resolution depends on the small-signal characteristics of the latch near the metastable point (roughly $V/2$) while gate transitions occur at the full magnitude of the supply voltage. Therefore, the relative increase of τ at lower supply voltages supersedes that of the FO4 delay leading to a decrease in the ratio t_s/τ. It has been noted in [82] and [83] that the increase in propagation delay at lower voltages compensates for the increase in τ. However, this is true only for supply voltages well above $2V_{\text{th}}$. At lower voltages, synchronizers have exponentially smaller MTBF due to this effect.



(a) Scaling of τ and FO4

(b) MTBF Degradation

Figure 4.1: Impact of VF scaling on synchronizer MTBF

To evaluate the practical severeness of this effect, simulation was used to calculate τ of four flip-flops in a 90nm library and compare it against the FO4 delay. The flip-flops consist of two sizes of a typical data flip-flop DFF and the equivalent sizes of a variant DFFSB which supports asynchronous set. Two observations can be made from the collected data (Figure 4.1a). First, the value of τ of all flip-flops increases more significantly than the FO4 delay at lower supply voltages (which supports small-signal analysis). The plot in Figure 4.1b demonstrates how this effect can reduce the MTBF of a typical 2 flip-flop synchronizer from an extremely conservative figure (10^{16} years) at nominal supply voltage to as low as 1 second at near-threshold voltages. Thus, while one synchronization cycle is sufficient to meet a MTBF criterion of 10^4 years at nominal supply voltage, up to three cycles are required to maintain this figure across the entire supply voltage range. Second, the performance of different flip-flop designs does not scale evenly and so it is difficult to devise a general rule to counteract this degradation.

The disproportionate scaling of the FO4 delay and τ has been investigated in [79] [81] [10] [84] from a technology-scaling perspective and as a performance metric for comparing different latches. However, the impact of this effect on synchronization MTBF in DVFS applications appears not to have been recognized.

4.3 Proposed Clock Domain Interface

Not being able to characterize τ or its scaling characteristics at design time leaves the designer with little choice except for implementing a long synchronizer chain to ensure that required MTBF is obtained across all VF points. This section presents a dynamic interface that serves as an alternative to this conservative strategy. The interface contains four synchronizers of different latencies and a sensor circuit to evaluate the ratio FO4/τ dynamically after every shift in the VF point. The measured value of FO4/τ is used to determine and switch to the minimum latency synchronizer that meets the system's MTBF criterion.

4.3.1 Principle of Operation

The presented design exploits the fact that satisfying $t_s > R\tau$ (where R is a constant) is sufficient to meet a MTBF criterion without explicit knowledge of either t_s or τ. To illustrate, let P_0 denote the quantity $T_w f_c f_d$. The MTBF expression (Equation 2.11) can now be re-written as:

$$\text{MTBF} = \frac{\exp(t_s/\tau)}{P_0} \tag{4.4}$$

Now assume that $P_0 = 10$ MHz, if $R = 40$ then:

$$\text{MTBF} = \frac{\exp(40)}{10^6} = 746 \text{ years} \tag{4.5}$$

Using Equation 4.1, the Inequality $(t_s > R\tau)$ can be expressed as $(m \times n \times \text{FO4} > R\tau)$ which can be re-arranged into:

$$\frac{\text{FO4}}{\tau} > \frac{R}{m \times n} \tag{4.6}$$

In other words, it is possible to determine whether a synchronizer whose latency is m clock cycles satisfies the MTBF of a particular system (with given R and n) by simply checking if the ratio FO4/τ is larger than $R/(m \times n)$.

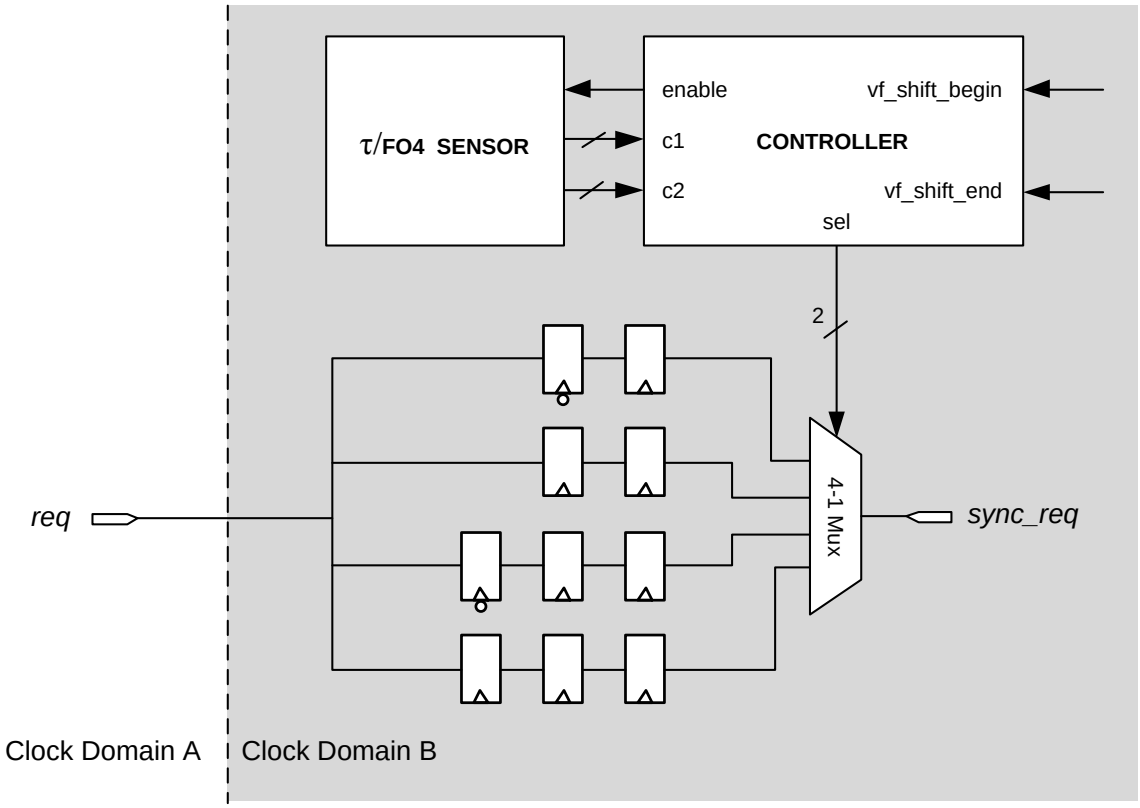


Figure 4.2: Adaptive clock domain interface

The presented interface, shown in Figure 4.2, exploits this relationship. The interface includes four synchronizers of latencies equal to 0.5, 1, 1.5 and 2 clock cycles. After every shift in the VF point of the clock domain, the interface uses a built-in sensor to dynamically evaluate the ratio FO4/τ and, using Inequality 4.6, determine if each of the four synchronizers meets the MTBF requirement of the system. The minimum latency synchronizer from the matching group is then selected and used to synchronize the asynchronous input until the next VF shift.

The selection criteria based on Inequality 4.6 for the implemented synchronizers ($m = \{0.5, 1, 1.5, 2\}$) are listed in Table 4.1.

Table 4.1: Synchronizer selection criteria

$\frac{\text{FO4}}{\tau} > \frac{2R}{3n}$	$\frac{\text{FO4}}{\tau} > \frac{R}{n}$	$\frac{\text{FO4}}{\tau} > \frac{2R}{n}$	Minimum Sync. Cycles (m)	sel
1	1	1	0.5	0
1	1	0	1.0	1
1	0	0	1.5	2
0	0	0	2.0	3

4.3.2 FO4/Tau Sensor

This subsection discusses how the FO4/τ sensor is used to determine if the value of FO4/τ is higher than the thresholds $\frac{2R}{n}$, $\frac{R}{n}$ and $\frac{2}{3}\frac{R}{n}$ which correspond to the typical synchronizer chains of 0.5, 1, 1.5 and 2 latency cycles respectively.

The schematics of the FO4/τ sensor are shown in Figure 4.3. In principle, this circuit measures the relative increase in the MTBF of a synchronizing flip-flop due to allowing $d\times$FO4 extra time for resolving metastable states (where d is a circuit constant). From Equation 2.11, increasing t_s by $d\times$FO4 will scale the MTBF by a factor of $\exp(d \times \text{FO4}/\tau)$ which, given the value of d, can be used to calculate FO4/τ.

The circuit consists of a flip-flop FF1 which samples the output of a ring oscillator. Assuming that the output frequency f_{osc} of the oscillator is asynchronous to the sampling clock of FF1 (clk), then FF1 will become metastable. Two flip-flops FF2 and FF4 sample the output of FF1 at two different times that shortly follow the positive edge of clk. In particular, FF2 samples the output of FF1 after t_{pd1} seconds while FF4 samples it after $t_{\text{pd1}} + t_{\text{pd2}}$ seconds. A much later sample is captured by a fourth flip-flop FF3 at the negative edge of clk.

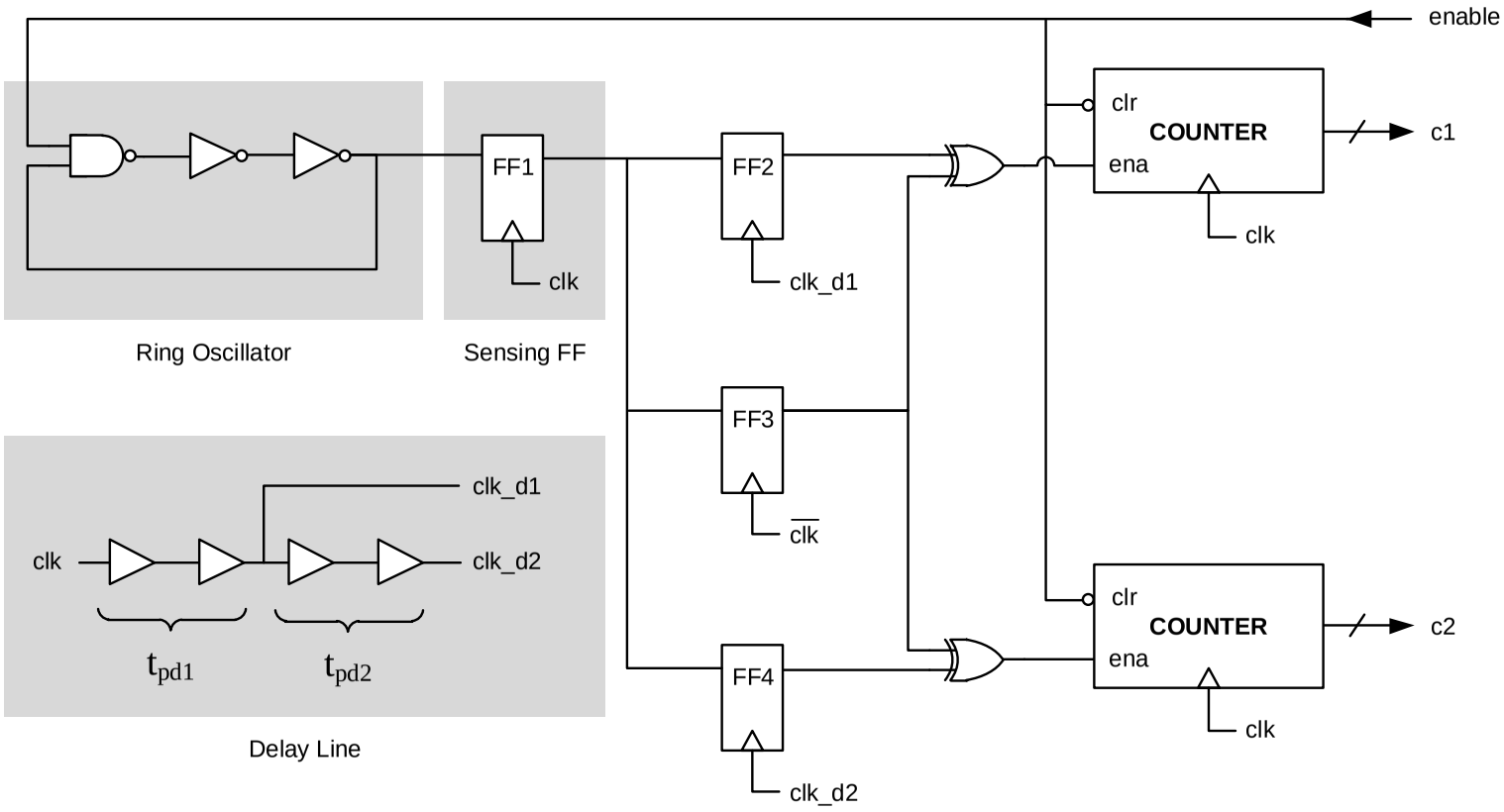


Figure 4.3: Schematics of FO4/τ sensor

Due to the occurrence of metastable states, FF1 will exhibit prolonged clock-to-q transitions. The clock period is assumed long enough such that transitions later than the sampling time of FF3 ($T/2$ seconds after the positive edge of clk) are relatively rare and can be ignored. When a late transition is not captured by FF2, the values of FF2 and FF3 will differ and a counter c1 is incremented. Similarly, when a transition is not captured by FF4, the values of FF4 and FF3 will differ and a counter c2 is incremented.

In essence, the chains FF1-FF2 and FF1-FF4 act as synchronizers whose failures are counted by c1 and c2 respectively. Thus, after enabling the counters c1 and c2 for a fixed period of time t, their values can be derived from Equation 2.11 as:

$$\text{c1} = t \times 2 \times T_w f_c f_{\text{osc}} \times \exp(-(t_{\text{pd1}} - t_{\text{cq}})/\tau) \tag{4.7}$$

$$\text{c2} = t \times 2 \times T_w f_c f_{\text{osc}} \times \exp\big(-(t_{\text{pd1}} + t_{\text{pd2}} - t_{\text{cq}})/\tau\big) \tag{4.8}$$

where t_{cq} is the nominal clock-to-q delay of FF1.

From Equations 4.7 and 4.8:

$$\frac{\text{c1}}{\text{c2}} = \exp(t_{\text{pd2}}/\tau) \tag{4.9}$$

Now, let t_{pd2} represent a pre-determined multiple d of the FO4 delay. Thus:

$$\text{c1} = \text{c2} \times \exp(d \times \text{FO4}/\tau) \tag{4.10}$$

If the counters are enabled till c2 reaches a pre-defined value, c2 will become a design constant and the only dynamic parameter that will influence c1 will be the ratio $\text{FO4}/\tau$. Based on this monotonic relationship, it is possible to pre-determine the c1 values that correspond to the $\text{FO4}/\tau$ threshold values $\left\{\frac{2R}{n}, \frac{R}{n}, \frac{2}{3}\frac{R}{n}\right\}$ and use them to determine when these thresholds have been crossed. These c1 thresholds are referred to as $\{k1, k2, k3\}$ respectively and are calculated from Equation 4.10 as follows:

$$k1 = c2 \times \exp(d \times 2R/n) \tag{4.11}$$

$$k2 = c2 \times \exp(d \times R/n) \tag{4.12}$$

$$k3 = c2 \times \exp(d \times 2R/3n) \tag{4.13}$$

```
while (1)
{
        while (! vf_shift_begin );  // wait until VF shift begins
        sel=3;                      // select most conservative synchronizer
        while (! vf_shift_end );    // wait until VF shift ends
        enable=1;                   // enable performance sensor
        while (c2!=1024);           // wait until measurement is complete
        enable=0;                   // disable performance sensor
        // select optimum synchronizer:
        if       (c1>k1) sel=0;   // FO4 / Tau > (2R/n)
        else if  (c1>k2) sel=1;   // FO4 / Tau > (R/n)
        else if  (c1>k3) sel=2;   // FO4 / Tau > (2R/3n)
        else             sel=3;
}
```

Listing 4.1: Controller psuedocode

4.3.3 Controller Behavior

Subsection 4.3.1 described how the minimum-latency synchronizer can be determined by comparing the value of FO4/τ with the pre-computed thresholds $\left\{\frac{2R}{n}, \frac{R}{n}, \frac{2}{3}\frac{R}{n}\right\}$. Subsection 4.3.2 then described how the latter task can be achieved by enabling the FO4/τ sensor till c2 reaches a pre-defined value and then comparing the value of c1 with three corresponding pre-computed thresholds $\{k1, k2, k3\}$. This subsection now explains how the interface controller implements the previously described behavior following every shift in the VF point of the clock domain.

The two signals `vf_shift_begin` and `vf_shift_end` are issued by the environment to notify the interface when VF shifts begin and end respectively. Initially, both `vf_shift_begin` and `vf_shift_end` are de-asserted and the controller is idle. When the domain's DVFS controller is about to initiate a change to a new VF point, it asserts `vf_shift_begin`. As soon as `vf_shift_begin` is asserted, the interface controller switches to the most conservative synchronizer (`sel`$= 3$) immediately. This is necessary because the minimum-latency synchronizer at the new VF point is unknown at this stage and the interface must not permit a MTBF violation under any circumstances.

When the shift is complete, `vf_shift_end` is asserted and the controller enables the FO4/τ sensor by asserting `enable`. The controller then waits for the sensor measurement process to complete (this happens when c2 reaches a pre-defined value, chosen to be 1024 in the proposed design). Subsequently, the value of c1 is compared against the three pre-determined threshold $\{k1, k2, k3\}$ and the lowest-latency synchronizer is selected according to the criteria in Table 4.1. This behavior is summarized in Listing 4.1.

4.3.4 Average Latency

The proposed design uses the most conservative synchronizer during VF shifts and the subsequent FO4/τ measurement process. If these time periods represent a significant fraction of the runtime of the system, the average latency of the interface will be higher than optimum. To mitigate this problem, a lookup-table can be used to store the lowest-latency synchronizer setting after measuring FO4/τ at each VF point. In subsequent shifts to pre-characterized VF points, the optimum synchronizer is selected directly based on the table records.

4.3.5 Variability

The proposed design assumes that the sensing flip-flop FF1 has the same τ as the synchronizer flip-flops and that t_{pd2} accurately represents a fraction d of the critical path delay of the system. In practice, these quantities differ due to process variability and so the sensing components cannot be assumed identical to the components they represent. Therefore, sufficient margins must be allowed when computing the thresholds $\{k1, k2, k3\}$ to leave room for component mismatch errors. Allocating these margins to accommodate for component variability will not increase the average latency if the ratio FO4/τ is sufficiently-far from the pre-computed thresholds at all VF points. In all cases, the average latency of the proposed design will be lower than that of the worst-case synchronizer chain.

Table 4.2: Cost comparison of adaptive interfaces*

Interface	Latency Control	Area (μm^2)	Power (μW)
[3]†	Fine	625000	1500
Proposed	Coarse	588	61

*cost figures drawn from synthesis in a 90nm technology library

†using a 25k lookup table for log

4.4 Conclusion

The disproportionate scaling of propagation delay and τ with the supply voltage means that the optimum number of synchronization cycles in a DVFS system can vary depending on the voltage/frequency operating point. Common design flows rely on black-box flip-flop models which do not enable characterizing τ and so it is difficult to mitigate this problem without relying on high-latency synchronizers to accommodate for worst-case performance. This chapter presented an adaptive interface that can optimize synchronization latency dynamically by evaluating flip-flop synchronization performance after every shift in the operating point. The proposed design relies on pre-computed thresholds and does not require arithmetic circuits. This makes it more practical than similar adaptive approaches such as [3] where computing the MTBF of synchronization explicitly incurred large area and power overheads (a cost comparison is presented in Table 4.2).

Chapter 5

Physical Parameter Sensor for FPGAs

5.1 Physical Parameter Sensing

The chapter introduces a new application area for flip-flip metastability, namely the sensing of intra-chip physical parameters. In Chapter 4, the sensitivity of the metastability resolution time constant τ to changes in the supply voltage is treated as a reliability problem. Here, the same effect is exploited to build a soft FPGA sensor that converts intra-chip physical quantities such as supply voltage and temperature into digital counts which can be interpreted by the FPGA application. This section starts by motivating physical parameter sensing and discussing its applications in FPGAs.

Online monitoring of VLSI systems using on-chip sensors can provide a variety of useful information for self-awareness, adaptivity and performance profiling. The reconfigurability of Field Programmable Gate Arrays (FPGAs) offers a unique opportunity to exploit such sensors to counteract process variations, aging effects and within-die uneven distribution of supply voltage and thermal activity. For example, variation-aware FPGA CAD flows [85] [86] use variation maps that are collected by sensor arrays to compute optimized component placement. Such flows were demonstrated to achieve up to 19.3% reduction in critical path delays [87]. Several studies have also investigated the use of on-chip sensors to characterize thermal activity. In [88], adaptive thermal regulation improved the performance of a benchmark system by a factor of 4. Another form of thermal management was presented in [89] where thermal-aware

thread-mapping in a multi-core system was used to balance temperatures across the chip. Sensor readings were also used to evaluate thermal simulators and models in [90] [91]. Similar support is provided by physical parameter sensing for the development of IR-drop management [92], power-aware CAD flows [93] and wear-leveling techniques [94].

Sensing intra-chip physical parameters in FPGAs is particularly challenging due to the digital nature of their components. Some FPGAs are equipped with analogue sensors (e.g. Xilinx System Monitor) but these sensors have fixed locations within the chip and cannot be used to collect spatial data. An alternative to embedded sensing is to use external equipment to perform non-intrusive characterization. An example of this approach is presented in [95] where an external probe is used to measure frequency inside an FPGA using electromagnetic analysis. Similarly, infrared imaging has been used in [96] to characterize thermal activity. Such methods eliminate the need for embedded sensors but require additional hardware, more complex measurement procedures and are more difficult to interface with the target FPGA application. The shortcomings of built-in and external sensors are overcome by those that can be realized using reconfigurable components, i.e. *soft sensors*. In particular, designs based on Ring Oscillators (ROs) are prevalent in intra-chip FPGA parameter sensing literature.

Emerging profiling and dynamic management applications require versatile and high performance sensors. This translates to a number of requirements. First, sensors must not consume significant device resources. Even when few are utilized, sensors must still be compact to enable them to fit into the available resources within the implemented system. This is particularly important when sensors are instantiated dynamically after the deployment of the target FPGA application [97] or when their placement must be constrained for optimal sensing [98]. Second, sensors must be accurate and precise to minimize measurement error. Third, sensors must have a small measurement time to support sensing applications that require high sampling rates.

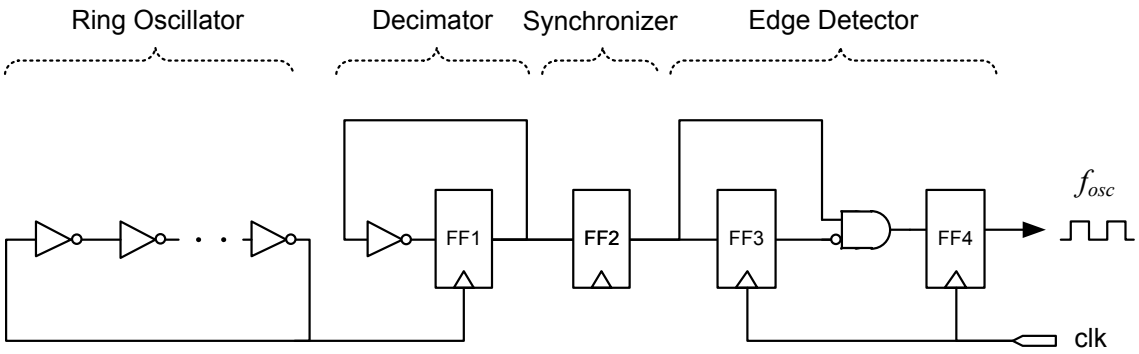


Figure 5.1: Ring oscillator sensor

5.2 Background

This section covers necessary background pertaining to the sensing of intra-chip physical parameters in FPGAs. First, the functionality, resource utilization, accuracy and precision of RO-based sensors are discussed to provide a baseline for comparing the proposed design. Second, a few notes are made regarding the modeling and relationships between the different intra-chip physical parameters.

5.2.1 Ring Oscillators

An RO is a loop of logic gates with a negative net gain. When the loop is powered up, nondeterministic circuit noise induce transitions which are initially amplified and then continue to propagate through the loop. The loop thus oscillates at a frequency that is inversely proportional to its element count n and element delay t_d or:

$$f_{\text{osc}} = \frac{1}{2 \times n \times t_d} \tag{5.1}$$

The loop is commonly created by connecting inverter chains and may include a single NAND gate to provide an enable control for the oscillating behavior. Intra-chip physical parameters affect the propagation delay of logic cells and can thus influence the loop's output frequency. This relationship is exploited by sensors such as the one shown in Figure 5.1, to map physical parameter changes to frequency variations which are easy to interpret digitally.

Resource Utilization

To ensure reliable counting, the output frequency of a RO must be smaller than the sampling clock frequency (f_c) by at least a factor of 2 (Nyquist criterion). One way to do this is to add more elements to the loop until a sufficiently-long period is obtained. However, this method requires a considerably-large number of inverting elements. For example, assuming $t_d = 150$ ps and $f_c = 100$ MHz, the required element count is 67. A more economical solution is to use a clock decimator, similar to the one shown in Figure 5.1. Each stage of the decimator, consisting of a single flip-flop and an inverter, scales down the RO frequency by a factor of 2. Thus, the output frequency of a RO with 3 elements (= 1.1 GHz taking $t_d = 150$ ps) can be down-scaled to below 50 MHz using only 5 decimation stages. The total element count using this method is only 8 inverters and 5 flip-flops as opposed to 67 inverters in the previous example.

The output frequency of a RO can be measured using an edge detector and an event counter to count the number of periodic oscillations occurring during a fixed amount of time. Note that the RO output is asynchronous to the sampling clock and so it cannot be used to drive the counter logic directly. Instead, the edge detector must be preceeded by at least one flip-flop to synchronize the RO output.

The event counter can be implemented in a number of ways. Binary counting logic consumes significant resources compared to the other parts of the sensor and so alternative implementations are often used. In [99], the authors describe a highly-efficient implementation of event counters based on shift registers only and requiring few integer operations to decode. Shift registers are abundantly present in common FPGA architectures (reconfigurable M4K blocks in Altera and LUT Shift Registers in Xilinx). Their use to implement counters greatly reduces the overhead of implementing on-chip sensors.

Accuracy and Precision

The amount of measurement error exhibited by a sensor is characterized by its accuracy and precision. Accuracy is a measure of the deviation of the mean of samples from the actual value of the measured physical quantity while precision is a measure of the spread of samples. Measurement error is introduced when physical quantities other

than the one being measured contribute to the sensor output. These quantities can be classified as either deterministic or stochastic and affect measurement accuracy and precision respectively.

In the case of on-chip measurement, examples of error sources include wire crosstalk and thermal noise. The oscillating behavior of ROs makes them highly susceptible to the influence of such sources. Variations in propagation delays cause a build up of jitter in the oscillation period of a RO. Jitter resulting from stochastic variations accumulates with the square root of time while that from systematic variations accumulates linearly [100]. It has been suggested that larger ROs offer better precision because their output counts exhibit lower relative variations [101]. However, larger ROs also have higher sensitivity and so their measurement variation (precision) is the same as that of smaller ROs. This is demonstrated empirically in Section 5.4.

5.2.2 Parameter Mapping

Intra-chip physical parameters are strongly correlated and knowledge of few is usually sufficient to infer others. For example, lower voltages and higher temperatures decrease transistor switching time and hence increase propagation delays. Therefore, given a fixed voltage, propagation delay measurements can be used to calculate temperature. In [99], the authors describe methods to use ring oscillator measurements to calculate voltage drop, component variations, leakage, dynamic power and temperature. Other studies investigated relationships between thermal activity and power [96], temperature and process variations [102] and temperature and supply voltage [101]. The methods of sensing all these parameters are essentially the same; ROs are used to transform changes in propagation delay that arise due to physical changes into frequency variations which can be measured digitally.

5.3 Proposed Sensor

5.3.1 Overview

Figure 5.2 depicts the proposed sensor. A flip-flop FF1 is used to latch an asynchronous input and exhibits frequent prolonged clock-to-q delays as a becoming metastable. Two flip-flops, FF2 and FF3, capture the output of FF1 on the following falling and rising clock edges respectively. When the transition delay of FF1 is excessively long, FF2 and FF3 will capture different values and a Metastable Event (ME) is flagged (the cases where long transitions fail to be captured by FF3 are ignored because the clock period is assumed to be long enough to render the probability of such events extremely small. By adjusting the duty cycle of the clock, it is possible to maintain a fixed rate of these events. A buffer gate BUF1 is inserted between FF1 and FF2 to enable the sensor to maintain high event rates without requiring an impractically-low clock duty cycle. Small-signal analysis shows that the rate of MEs is exponentially dependent on the metastability regeneration time constant (τ) of FF1. The time constant τ is a delay metric and is affected by variations in intra-chip physical parameters similar to gate propagation delays. Therefore, changes in τ affect the rate of MEs experienced by the sensor. To quantify these changes, an event counter is incremented whenever a ME is flagged. After enabling the sensor for a set period of time, the counter output is mapped to the physical quantity of interest using calibrated models.

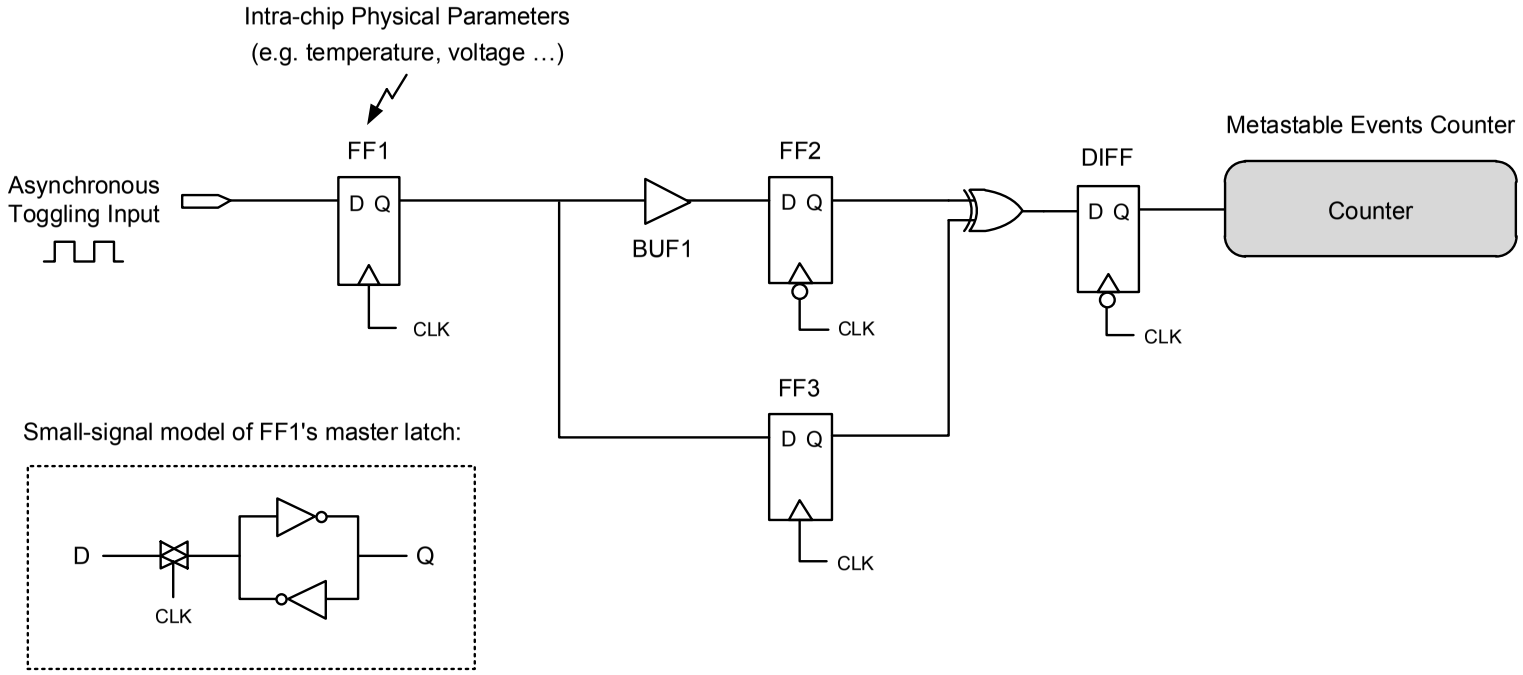


Figure 5.2: Proposed metastability-based sensor

5.3.2 Small-signal Model

After the occurrence of a rising clock edge, the master latch of FF1 becomes opaque and attempt to decide whether the input is logic high or low. During this process, the latch behaves as a regenerative amplifier whose output $Q(t)$ grows exponentially with time [11]:

$$Q(t) = Q' \times e^{t/\tau} \tag{5.2}$$

where Q' is the initial output voltage after the input is latched and τ is the regeneration time constant. Both Q and Q' are expressed relative to a hypothetical critical voltage at which the latch will be in a perfectly-metastable state.

The ME counter is incremented whenever the output of FF1 transitions late enough not to be captured by FF2. This can be expressed numerically as Q not reaching a certain threshold voltage Q_{th} by the time it is sampled by FF2. Thus:

$$\begin{aligned} \text{Count} &= P[Q(t_r) < Q_{\text{th}}] \times n \\ &= P[Q' < Q_{\text{th}} \times e^{-t_r/\tau}] \times n \end{aligned} \tag{5.3}$$

where t_r is the time available for FF1's output to regenerate before it is sampled by FF2 and n is the measurement duration in cycles.

The arrival time of FF1's input relative to the clock edge is assumed to be evenly-distributed. Hence, Q' is assumed to be evenly distributed across the range [0, V_{DD}]. Therefore:

$$\text{Count} = \frac{Q_{\text{th}}}{V_{DD}} \times e^{-t_r/\tau} \times n \tag{5.4}$$

The regeneration time t_r is the propagation delay slack of the path FF1 $\rightarrow$ FF2. This is equal to the time between the rising and falling clock edges (denoted t_{high}) minus the propagation delay of BUF1 (t_{BUF1}).

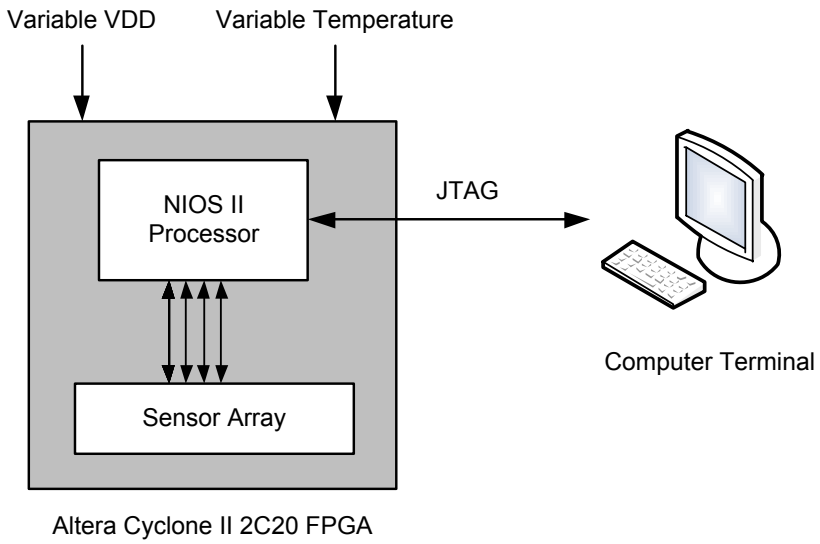


Figure 5.3: Sensor characterization system

Two assumptions are now made. First, since τ is a delay metric, it is assumed to be inversely proportional to the measured physical quantity (denoted p), similar to the case with propagation delay (Equation 5.1). Therefore:

$$p = \frac{k_1}{\tau} \tag{5.5}$$

where k_1 is a constant.

Second, the propagation delay of BUF1 is assumed to scale proportionately with τ over the measurement range or:

$$\frac{t_{\text{BUF1}}}{\tau} = k_2 \tag{5.6}$$

where k_2 is a constant.

Equation 5.4 can now re-written in the form:

$$\text{Count} = n \times K \times e^{Sp} \tag{5.7}$$

where K and S are sensor-specific constants as follows:

$$K = \frac{Q_{\text{th}}}{V_{\text{DD}}} \times e^{k_2} \tag{5.8}$$

$$S = \frac{-t_{\text{high}}}{k_1} \tag{5.9}$$

Following Equation 5.7, the proposed design establishes an exponential relationship between the MEs counter output and the value of the physical quantity p. This mapping substitutes the functionality of ROs in sensing intra-chip physical parameters.

The assumptions represented by Equations 5.5 and 5.6 were necessary for the derivation of Equation 5.7. In Section 5.4, both assumptions are validated by showing empirically that the relationship between the counter output and p is exponential to a high degree of accuracy.

5.3.3 Count Adjustment

The nominal count for the proposed sensor is adjusted by varying the clock's duty cycle. The latter should be adjusted to to achieve counts in the range $[10^1 : 10^7]$ for optimal sensing using a 32-bit counter. Count adjustment needs to be done only once during the calibration process.

5.4 FPGA Measurements

The proposed design and 3 RO-based sensors were implemented on an Altera Cyclone II FPGA. Table 5.1 lists these sensors and their resource utilization. The characterization system used for sensor evaluation (illustrated in Figure 5.3) supports voltage and temperature control and aims at comparing sensor response, precision and accuracy.

The FPGA device used was mounted on a Terasic DE1 development board which has been modified in two ways. First, the FPGA core voltage pin was disconnected from the on-board supply and connected it to an external source. Second, a heat sink with two soldered power resistors were mounted on top of the FPGA as shown in Figure 5.4

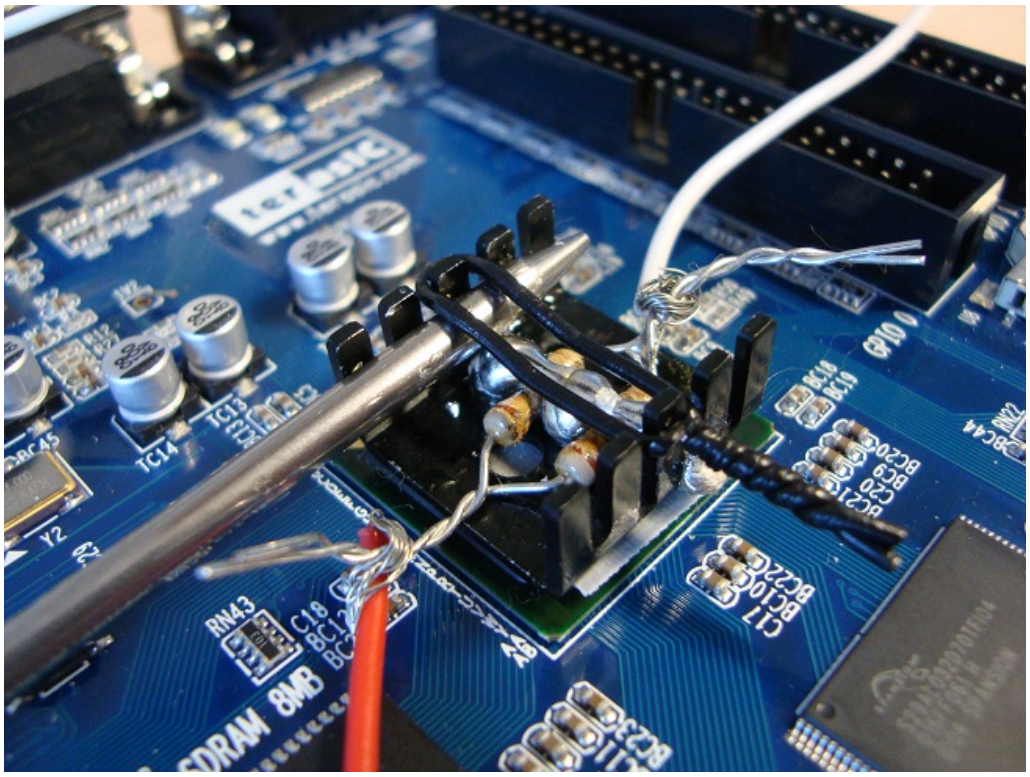


Figure 5.4: Temperature control and measurement setup

to sense and control its temperature. By driving and adjusting the current through the resistors, the heat sink temperature could be set in the range $25 \sim 70\,^{\circ}$ C with an accuracy of $\pm 0.5\,^{\circ}$C. An external temperature probe, connected to the heat sink, was used as a reference for calibration. Note that this setup does not require (nor attempt to perform) an accurate calibration of the FPGA's junction temperature. The purpose of the setup is to induce simple thermal gradients which are sufficient to compare the sensors as long as they are calibrated and compared under the same conditions.

Table 5.1: Resource utilization and models of characterized sensors

Sensor	Description	LUTs	FFs	Temperature Model (°C)	Voltage Model (volts)
`RO5`	5-stage RO with 3 decimation stages	9	5	$t = -1.01\text{e-}4 \times C + 3074$	$v = 2.02\text{e-}8 \times C + 0.58$
`RO11`	11-stage RO with 3 decimation stages	15	5	$t = -1.92\text{e-}4 \times C + 2846$	$v = 4.29\text{e-}8 \times C + 0.56$
`RO17`	17-stage RO with 3 decimation stages	21	5	$t = -2.84\text{e-}4 \times C + 2803$	$v = 6.55\text{e-}8 \times C + 0.55$
`MS`	metastability-based sensor (proposed)	2	4	$t = 122 \times \log_{10}(C) - 706$	$v = -0.030 \times \log_{10}(C) + 1.35$

5.4.1 Resource Utilization

The clock frequency (f_c) used in the reported experiments is 430 MHz. The most compact RO that can be sampled efficiently by this clock (on the used FPGA device) was determined via trial and error. It consists of 5 inverters and 3 decimation stages. Higher speed FPGAs and those operating at lower clock frequencies require more resources to produce RO frequencies which are adequately low for proper counting. The proposed design is free from this dependency and can be instantiated using 20% less flip-flops and 75% less LUTs compared to the most compact RO implementation. This excludes the logic needed to implement event counters since they can be instantiated very compactly as described in Section 5.2.

The proposed sensor has two further requirements over ROs. First, a toggling signal that is asynchronous to the system clock is needed to induce metastable states in FF1. In the described experiment, this was provided by an independent oscillator running at 450 MHz. Second, the clock's duty cycle needs to be adjusted. This was performed using an external clock generator although on-chip adjustment of the clock's duty cycle is equally suitable. Several FPGA families provide PLLs which support, among other things, adjusting the clock's duty cycle (e.g. the `ALTPLL` megafunction in Altera and Clock Management Tiles in Xilinx devices).

5.4.2 Response

The output counts of the implemented sensors were measured under a temperature gradient of $25 \sim 70$ °C. This process has been repeated for a voltage gradient of $1.1 \sim 1.3$ volts. The output counts of `RO5` and `MS` (the proposed sensor) are shown in Figures 5.5 and 5.6 against temperature and voltage scales respectively (to simplify the comparison, the responses of `MS` are shown in a semi-log plots).

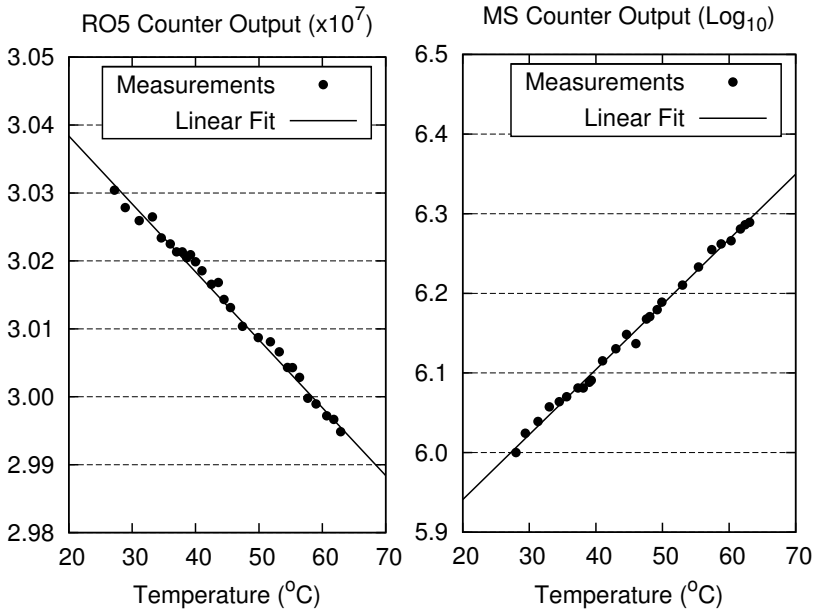


Figure 5.5: Temperature response

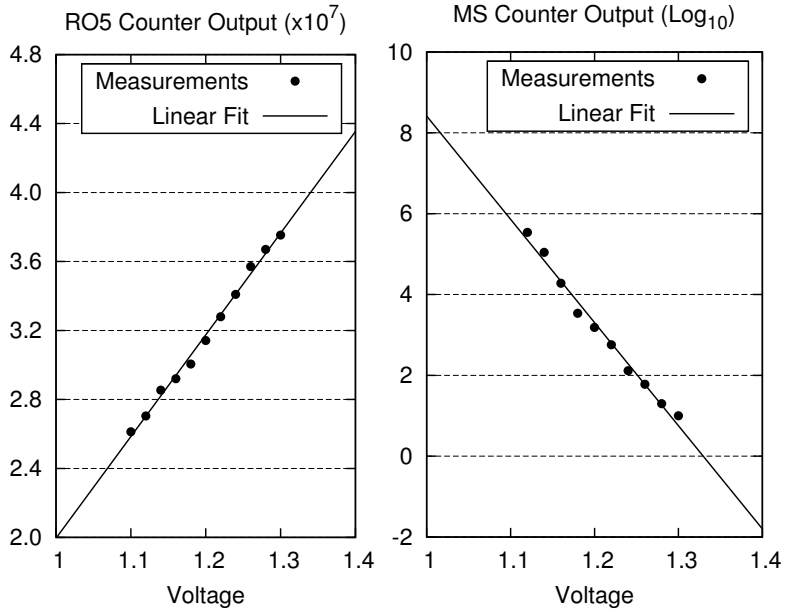


Figure 5.6: Voltage response

The output count response of MS accurately fits a straight line on a semi-log plot in both cases. This demonstrates a highly exponential relationship with temperature and voltage variations and validates the assumptions used in small-signal analysis in Subsection 5.3.2 (Equations 5.5 and 5.6).

5.4.3 Calibration

Calibrated temperature and voltage models were constructed for each of the implemented sensors (Table 5.1). Temperature and voltage experiments were performed independently so the models of each of the two parameters were derived while the other was held constant. The response of RO sensors was modeled using the linear form:

$$p = m \times C + b \tag{5.10}$$

where p is the measured physical parameter, C is the counter output after 1 second and m and b are model constants.

As for MS, its response was modeled using the exponential model form:

$$p = m \times \log_{10}(C) + b \tag{5.11}$$

The clock's duty cycle was set to 50% to obtain 10^6 MEs/sec in temperature experiments and to 70% to obtain $10^1 \sim 10^6$ MEs/sec in voltage experiments.

5.4.4 Precision

After calibration, 20 temperature readings were collected from each implemented sensor. This process was repeated 10 times for measurement durations ranging from 1 to 50 milliseconds. Figures 5.7 and 5.8 compare the standard deviation of sensor measurements against measurement duration.

The results demonstrate that MS offers an average precision improvement of 60% in temperature sensing and 173% in voltage sensing compared to RO sensors. The improvement is more significant at lower measurement durations (corresponding to sampling rates in excess of 100 Hz). These differences can be attributed to the buildup

of jitter and the instability of ring oscillator frequencies, particularly at small time intervals. Furthermore, the results show that all RO sensors have similar precision versus measurement duration profiles. This supports the conclusion that the lower relative frequency variations of large ROs are compensated for by their higher sensitivity. Hence, increasing the number of inverters does not increase the precision of RO sensors.

5.4.5 Accuracy

To compare the accuracy of the sensors, a set of 100 measurements was first collected from each sensor and used for calibration. An additional 100 measurements were then collected from each sensor and compared to the predictions of its calibrated model. The systematic error exhibited by each sensor was calculated as:

$$\text{Systematic Error} = \left|\mu - \frac{1}{100}\sum_{i=1}^{100} s(i)\right| \tag{5.12}$$

where μ is the parameter value predicted by the calibrated model and $s(i)$ is the ith measurement.

This process was repeated 10 times for each of the temperature and voltage sensing scenarios. The results, presented in Figures 5.9 and 5.10, do not demonstrate any significant differences between the implemented sensors.

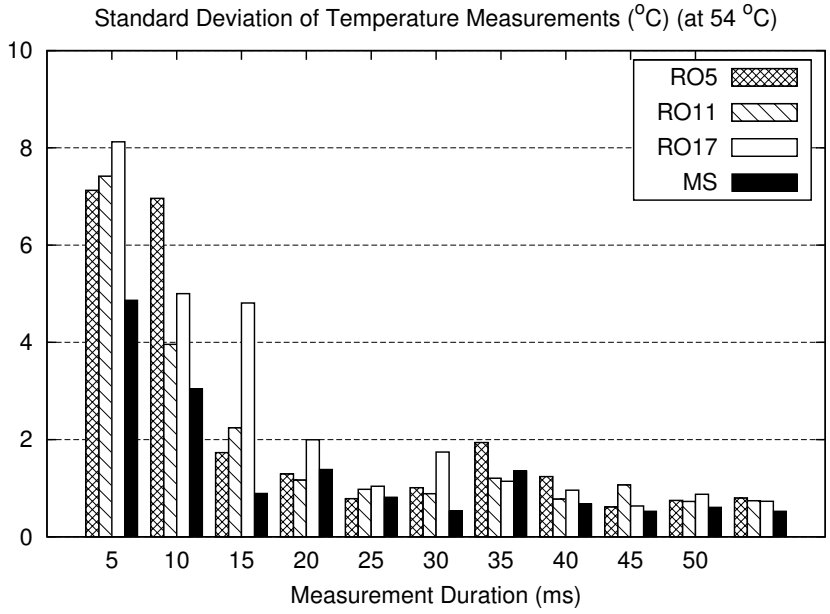


Figure 5.7: Temperature precision

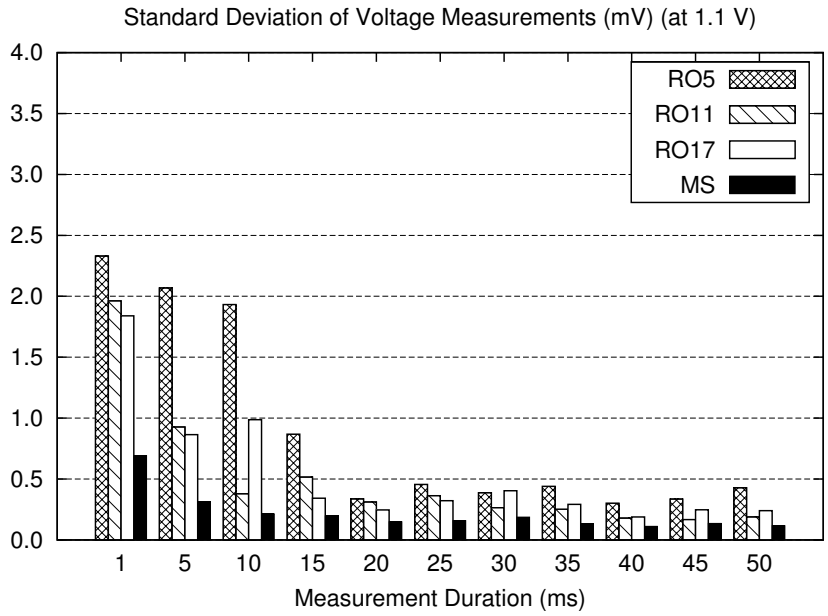


Figure 5.8: Voltage Precision

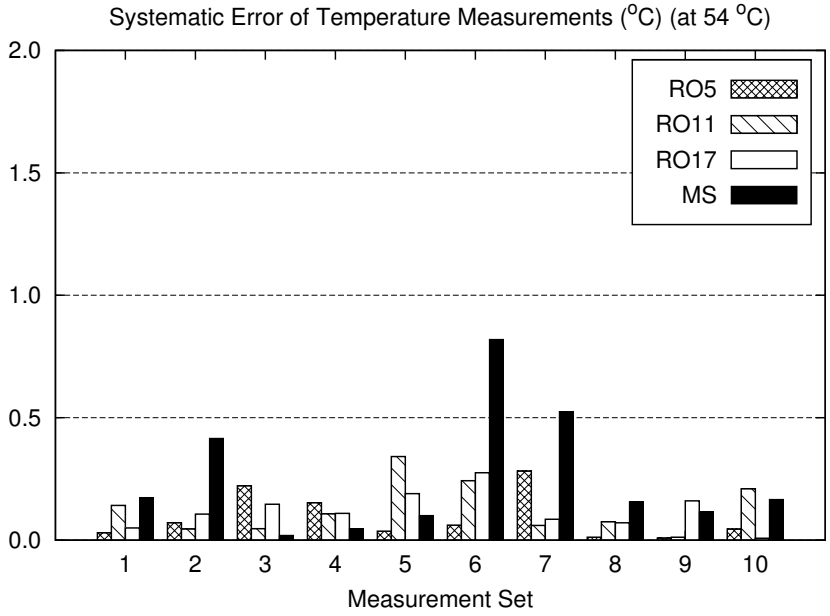


Figure 5.9: Temperature Accuracy

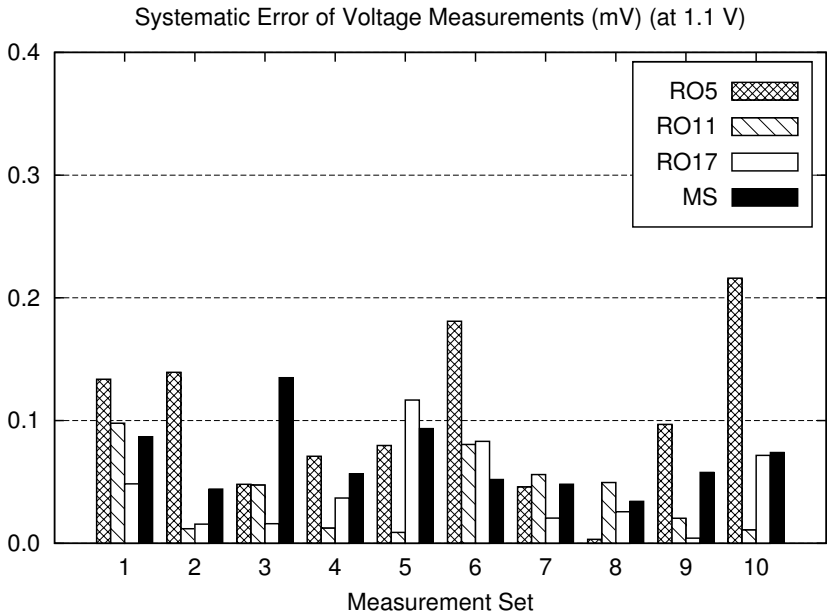


Figure 5.10: Voltage Accuracy

5.5 Conclusion

Metastable flip-flops are very sensitive to changes in their operating conditions. Small changes in the voltage or working temperature of a flip-flop have a significant impact on its metastability resolution parameter τ and consequently its failure rates. This chapter presented a novel sensor design that exploits this sensitivity to quantify changes in intra-chip physical parameters. Measurements from an Altera Cyclone II device demonstrated precision improvements of 60% in temperature sensing and 173% in voltage sensing compared to ring oscillators. The proposed sensor does not rely on oscillation and thus does not require a high clock frequency to sample oscillation periods. Furthermore, it consumes 20% fewer flip-flops and 75% less LUTs compared to the most compact ring-oscillator sensor in the setup making it more economical for implementing large arrays and easier to fit into existing FPGA applications.

Chapter 6

Conclusion

6.1 Summary of Contributions

The bulk of this thesis has been concerned with the problem of synchronizing the transmission of data (or control signals) between multiple clock domains. Specifically, two speculative solutions have been proposed to hide synchronization latency by performing an equal number of speculative data-dependent operations.

The first solution which has been referred to as Datapath Unfolding relies on loop unrolling to create duplicate state machines whose function is to compute speculative data-dependent states. These states are not used by the original machine until synchronization is complete and the validity of data (and hence the speculative states) is confirmed. It is shown that this approach is functionality correct and that it does not violate any of the principle tenets of the metastability problem. The solution is then extended by presenting a method to identify a subset of the state register whose flip-flops and input logic does not need to be duplicated; namely those whose values are independent of the value of the synchronized signal during synchronization cycles. The values of the these state bits remain the same during synchronization cycles regardless of whether a handshake is being synchronized or not. Therefore, their values need not be speculated. Finally, a design flow is presented to apply this transformation automatically to an RTL netlist representing an arbitrary Moore machine.

The second proposed solution which has been called Sequenced Latching relies on a different form of speculation that involves the individual synchronizer flip-flops. Here, the synchronizer is used as a state machine to alternately toggle two datapath instances which are connected in a cyclic pipeline. It is speculated that the potential prolonged clock-to-q delays of the synchronizer flip-flops do not corrupt the data latched by the cyclic pipeline. The synchronizer is constrained such that, when these events actually occur, its state will not change in the following cycle and the corrupted computation is automatically retried. This form of speculation involves an uncertainty span of a single cycle and so its duplication cost complexity is less than that of datapath unfolding whose uncertainty span equals the number of synchronization cycles.

The thesis also presented a practical solution for addressing the wide variability of flip-flop synchronization performance in the case of variable supply voltages. Choosing the optimal synchronizer chain length is particularly challenging in these cases because the common uncertainties in flip-flop metastability resolution performance are compounded. The presented solution involves using a minimal interface consisting mainly of a sensing circuit and a controller. The interface adjusts the length of the synchronizer chain in units of half a clock cycle depending on the dynamic metastability resolution speed of the library flip-flops. Although similar interfaces have been proposed and are able to fine-tune metastability resolution with higher resolution, the one presented in this thesis has the distinct advantage that it does not require any arithmetic circuits. Therefore its area and power costs are very small and represent a negligible fraction of those of similar designs.

Finally, the use of metastable flip-flops as sensors for quantifying intra-chip physical parameters such as voltage, temperature and parametric variation is identified as an application area of flip-flop metastability. The thesis presented a novel digital sensor that exploits the high sensitivity of metastable flip-flops to the intra-chip physical parameters that affect the gain of their cross-coupled inverting gates. On-chip measurements from an FPGA demonstrated precision and area cost improvements in comparison to conventional digital sensor designs.

6.2 Future Work

This thesis investigated speculation as a method of hiding synchronization latency and demonstrated that it has the practical potential of outperforming other clock domain crossing solutions. As outlined in Section 3.7, high level design restructuring can reduce speculation costs by excluding costly computational resources from the duplication set. An investigation in this area might reveal optimization guidelines and patterns that can be exploited by EDA tools without significant designer involvement.

The thesis also highlighted the compounded problem of determining synchronizer performance in DVFS systems. The proposed solution uses a dynamic circuit to sense metastability resolution performance and optimize synchronizer chain length dynamically. An alternative approach to tackle this problem is to attempt to optimize flip-flops to have a lower average τ/FO4 ratio. Although the value of τ at the nominal supply voltage is likely to suffer, the average latency of such flip-flops over the entire range of supply voltages can be lower.

Finally, in the domain of physical parameter sensing, the precision and sensitivity of the proposed metastability-based sensor can be improved by using high τ flip-flops. Such flip-flops will have higher metastable event rates and consequently lower relative random variation. It is also interesting to investigate whether the sensor's response can be optimized in favor of particular parameters over the others.

Bibliography

[1] C. Tokunaga, D. Blaauw, and T. Mudge, "True random number generator with a metastability-based quality control," in *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, Feb. 2007, pp. 404 –611.

[2] D. J. Kinniment and A. Yakovlev, "Low latency synchronization through speculation." in *PATMOS'04*, 2004, pp. 278–288.

[3] J. Zhou, D. Kinniment, G. Russell, and A. Yakovlev, "Adapting synchronizers to the effects of on chip variability," in *Proceedings of the 2008 14th IEEE International Symposium on Asynchronous Circuits and Systems*, ser. ASYNC '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 39–47. [Online]. Available: http://dx.doi.org/10.1109/ASYNC.2008.11

[4] S. Lubkin, "Asynchronous signals in digital computers (in automatic computing machinery; discussions)," *Journal of Math Tables and Other Aids for Computing*, vol. 6, no. 40, pp. 238–241, Oct. 1952.

[5] Aristotle and S. Leggatt, *On the Heavens, I and II*, ser. Classical texts. Aris & Phillips, 1995. [Online]. Available: http://books.google.co.uk/books?id=OrWAQgAACAAJ

[6] J. Sennett and D. Groothuis, *In Defense of Natural Theology: A Post-Humean Assessment*. InterVarsity Press, 2005. [Online]. Available: http://books.google.co.uk/books?id=UKSZeRnuyjAC

[7] E. Wormald, "A note on synchronizer or interlock maloperation," *Computers, IEEE Transactions on*, vol. C-26, no. 3, pp. 317 –318, March 1977.

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

[9] T. Chaney, "Comments on "a note on synchronizer or interlock maloperation"," *Computers, IEEE Transactions on*, vol. C-28, no. 10, pp. 802 –804, Oct. 1979.

[10] C. L. Portmann, "Characterization and reduction of metastability errors in CMOS interface circuits," Ph.D. dissertation, Stanford University, 1995.

[11] D. Kinniment, A. Bystrov, and A. Yakovlev, "Synchronization circuit performance," *Solid-State Circuits, IEEE Journal of*, vol. 37, no. 2, pp. 202 –209, Feb 2002.

[12] I. W. Jones, S. Yang, and M. Greenstreet, "Synchronizer behavior and analysis," in *ASYNC '09: Proceedings of the 2009 15th IEEE Symposium on Asynchronous Circuits and Systems (async 2009)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 117–126.

[13] *ispLSI/GAL Metastability Report*, 2001.

[14] A. Steininger, "Error containment in the presence of metastability," in *Fault-Tolerant Distributed Algorithms on VLSI Chips*, ser. Dagstuhl Seminar Proceedings, B. Charron-Bost, S. Dolev, J. Ebergen, and U. Schmid, Eds., no. 08371. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2009/1923

[15] ——, "Advanced digital design - metastability," University Lecture.

[16] J. Horstmann, H. Eichel, and R. Coates, "Metastability behavior of CMOS ASIC flip-flops in theory and test," *Solid-State Circuits, IEEE Journal of*, vol. 24, no. 1, pp. 146 –157, Feb 1989.

[17] T. J. Chaney and C. E. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," *IEEE Trans. Comput.*, vol. 22, no. 4, pp. 421–422, Apr. 1973.

[18] A. Martin and M. Nystrom, "Asynchronous techniques for system-on-chip design," *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1089 –1120, June 2006.

[19] S. Yang and M. Greenstreet, "Computing synchronizer failure probabilities," in *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, April 2007, pp. 1 –6.

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

[21] A. J. Martin, "Developments in concurrency and communication," C. A. R. Hoare, Ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990, ch. Programming in VLSI: from communicating processes to delay-insensitive circuits, pp. 1–64. [Online]. Available: http://dl.acm.org/citation.cfm?id=107155.107157

[22] M. Branicky *et al.*, "Why you can't build an arbiter," 1993.

[23] M. Valencia, M. Bellido, J. Huertas, A. Acosta, and S. Sanchez-Solano, "Modular asynchronous arbiter insensitive to metastability," *Computers, IEEE Transactions on*, vol. 44, no. 12, pp. 1456 –1461, Dec 1995.

[24] C. Van Berkel and C. Molnar, "Beware the three-way arbiter," *Solid-State Circuits, IEEE Journal of*, vol. 34, no. 6, pp. 840 –848, Jun 1999.

[25] F. El Guibaly, "Design and analysis of arbitration protocols," *Computers, IEEE Transactions on*, vol. 38, no. 2, pp. 161 –171, Feb 1989.

[26] J.-E. Eklund and C. Svensson, "Influence of metastability errors on SNR in successive-approximation A/D converters," *Analog Integr. Circuits Signal Process.*, vol. 26, no. 3, pp. 183–190, Mar. 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1008387223956

[27] C. Portmann and T. Meng, "Power-efficient metastability error reduction in CMOS flash A/D converters," *Solid-State Circuits, IEEE Journal of*, vol. 31, no. 8, pp. 1132 –1140, Aug. 1996.

[28] D. Kinniment, B. Gao, A. Yakovlev, and F. Xia, "Towards asynchronous A-D conversion," in *Advanced Research in Asynchronous Circuits and Systems, 1998. Proceedings. 1998 Fourth International Symposium on*, Mar. 1998, pp. 206 –215.

[29] T. Sundstrom, C. Svensson, and A. Alvandpour, "A 2.4 GS/s, single-channel, 31.3 dB SNDR at nyquist, pipeline ADC in 65 nm CMOS," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 7, pp. 1575 –1584, July 2011.

[30] J. Eklund and C. Svensson, "Metastability determines the noise in fast and accurate A/D converters," in *International Workshop on ADC modeling and testing*, ser. ADC Modeling anf Testing, M. L. Halttunen J., Daponte P., Ed. Helsinki; Finnish Society of Automation, June 1997, pp. 171–176.

[31] C. Mangelsdorf, "A 400-MHz input flash converter with error correction," *Solid-State Circuits, IEEE Journal of*, vol. 25, no. 1, pp. 184 –191, Feb 1990.

[32] P. Stubberud and E. Dagher, "Metastability requirements for a 2 GHz CMOS delta; sigma; modulator," in *Systems Engineering, 2005. ICSEng 2005. 18th International Conference on*, Aug. 2005, pp. 263 – 268.

[33] A. Hart and S. Voinigescu, "A 1 GHz bandwidth low-pass δ σ ADC with 20 - 50 GHz adjustable sampling rate," *Solid-State Circuits, IEEE Journal of*, vol. 44, no. 5, pp. 1401 –1414, May 2009.

[34] J. Cherry and W. Snelgrove, "Clock jitter and quantizer metastability in continuous-time delta-sigma modulators," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 46, no. 6, pp. 661 –676, Jun 1999.

[35] V. Suresh and W. Burleson, "Robust metastability-based TRNG design in nanometer CMOS with sub-vdd pre-charge and hybrid self-calibration," in *Quality Electronic Design (ISQED), 2012 13th International Symposium on*, March 2012, pp. 298 –305.

[36] ——, "Entropy extraction in metastability-based TRNG," in *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*, June 2010, pp. 135 –140.

[37] D. Kinniment and E. Chester, "Design of an on-chip random number generator using metastability," in *Solid-State Circuits Conference, 2002. ESSCIRC 2002. Proceedings of the 28th European*, Sept. 2002, pp. 595 –598.

[38] R. Brederlow, R. Prakash, C. Paulus, and R. Thewes, "A low-power true random number generator using random telegraph noise of single oxide-traps," in *Solid-State Circuits Conference, 2006. ISSCC 2006. Digest of Technical Papers. IEEE International*, Feb. 2006, pp. 1666 –1675.

[39] B. Jun and P. Kocher, "The intel random number generator," *Cryptography Research Inc. white paper*, 1999.

[40] M. Alshaikh, D. Kinniment, and A. Yakovlev, "On the trade-off between resolution time and delay times in bistable circuits," in *Electronics, Circuits, and Systems, 2009. ICECS 2009. 16th IEEE International Conference on*, Dec. 2009, pp. 355 –358.

[41] S. Yang, I. Jones, and M. Greenstreet, "Synchronizer performance in deep submicron technology," in *Asynchronous Circuits and Systems (ASYNC), 2011 17th IEEE International Symposium on*, April 2011, pp. 33 –42.

[42] G. Lacroix, P. Marchegay, and G. Piel, "Comments on "the anomalous behavior of flip-flops in synchronizer circuits"," *Computers, IEEE Transactions on*, vol. C-31, no. 1, pp. 77 –78, Jan. 1982.

[43] J. Jex and C. Dike, "A fast resolving BiNMOS synchronizer for parallel processor interconnect," *Solid-State Circuits, IEEE Journal of*, vol. 30, no. 2, pp. 133 –139, Feb 1995.

[44] C. Dike and E. Burton, "Miller and noise effects in a synchronizing flip-flop," *Solid-State Circuits, IEEE Journal of*, vol. 34, no. 6, pp. 849 –855, Jun. 1999.

[45] C. Foley, "Characterizing metastability," in *Advanced Research in Asynchronous Circuits and Systems, 1996. Proceedings., Second International Symposium on*, Mar 1996, pp. 175 –184.

[46] D. Kinniment, C. Dike, K. Heron, G. Russell, and A. Yakovlev, "Measuring deep metastability and its effect on synchronizer performance," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 9, pp. 1028 –1039, Sept. 2007.

[47] J. Zhou, D. Kinniment, C. Dike, G. Russell, and A. Yakovlev, "On-chip measurement of deep metastability in synchronizers," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 2, pp. 550 –557, Feb. 2008.

[48] Y. Semiat and R. Ginosar, "Timing measurements of synchronization circuits," in *Asynchronous Circuits and Systems, 2003. Proceedings. Ninth International Symposium on*, May 2003, pp. 68 – 77.

[49] S. Beer, R. Ginosar, M. Priel, R. Dobkin, and A. Kolodny, "An on-chip metastability measurement circuit to characterize synchronization behavior in 65nm," in *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, May 2011, pp. 2593 – 2596.

[50] T. J. Chaney, "Me and my glitch," March 2012.

[51] J. Zhou, D. Kinniment, G. Russell, and A. Yakovlev, "A robust synchronizer," in *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*, vol. 00, March 2006, p. 2 pp.

[52] J. Zhou, M. Ashouei, D. Kinniment, J. Huisken, and G. Russell, "Extending synchronization from super-threshold to sub-threshold region," in *Asynchronous Circuits and Systems (ASYNC), 2010 IEEE Symposium on*, May 2010, pp. 85 –93.

[53] M. Baghini and M. Desai, "Impact of technology scaling on metastability performance of CMOS synchronizing latches," in *Design Automation Conference, 2002. Proceedings of ASP-DAC 2002. 7th Asia and South Pacific and the 15th International Conference on VLSI Design. Proceedings.*, 2002, pp. 317 –322.

[54] D. M. Chapiro, "Globally-asynchronous locally-synchronous systems," Ph.D. dissertation, Stanford University, Oct. 1984.

[55] M. J. Stucki and J. Cox, J. R, "Synchronization strategies," in *Proceedings of the Caltech Conference On Very Large Scale Integration*, 1979, pp. 375–393.

[56] D. Bormann and P. Cheung, "Asynchronous wrapper for heterogeneous systems," in *Computer Design: VLSI in Computers and Processors, 1997. ICCD '97. Proceedings., 1997 IEEE International Conference on*, Oct 1997, pp. 307 –314.

[57] J. Muttersbach, T. Villiger, and W. Fichtner, "Practical design of globally-asynchronous locally-synchronous systems," in *Advanced Research in Asynchronous Circuits and Systems, 2000. (ASYNC 2000) Proceedings. Sixth International Symposium on*, 2000, pp. 52 –59.

[58] S. Dasgupta and A. Yakovlev, "Comparative analysis of GALS clocking schemes," *Computers Digital Techniques, IET*, vol. 1, no. 2, pp. 59 –69, March 2007.

[59] F. Gurkaynak, S. Oetiker, H. Kaeslin, N. Felber, and W. Fichtner, "GALS at ETH Zurich: success or failure?" in *Asynchronous Circuits and Systems, 2006. 12th IEEE International Symposium on*, March 2006, pp. 10 pp. –159.

[60] M. Greenstreet, "Implementing a STARI chip," in *Computer Design: VLSI in Computers and Processors, 1995. ICCD '95. Proceedings., 1995 IEEE International Conference on*, Oct 1995, pp. 38 –43.

[61] A. Chakraborty and M. Greenstreet, "Efficient self-timed interfaces for crossing clock domains," in *Asynchronous Circuits and Systems, 2003. Proceedings. Ninth International Symposium on*, May 2003.

[62] L. F. G. Sarmenta, G. A. Pratt, and S. A. Ward, "Rational clocking [digital systems design]," in *Proceedings of the 1995 International Conference on Computer Design: VLSI in Computers and Processors*, ser. ICCD '95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 271–278. [Online]. Available: http://dl.acm.org/citation.cfm?id=645463.655516

[63] W. Dally and S. Tell, "The even/odd synchronizer: A fast, all-digital, periodic synchronizer," in *Asynchronous Circuits and Systems (ASYNC), 2010 IEEE Symposium on*, May 2010, pp. 75 –84.

[64] Z. Wang, T. O'Neil, and E.-M. Sha, "Optimal loop scheduling for hiding memory latency based on two-level partitioning and prefetching," *Signal Processing, IEEE Transactions on*, vol. 49, no. 11, pp. 2853 –2864, Nov 2001.

[65] M. Younis, T. Marlowe, A. Stoyen, and G. Tsai, "Statically safe speculative execution for real-time systems," *Software Engineering, IEEE Transactions on*, vol. 25, no. 5, pp. 701 –721, Sep/Oct 1999.

[66] G. Lakshminarayana, A. Raghunathan, and N. Jha, "Incorporating speculative execution into scheduling of control-flow-intensive designs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, no. 3, pp. 308 –324, Mar 2000.

[67] A. Bhowmik and M. Franklin, "A general compiler framework for speculative multithreaded processors," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 15, no. 8, pp. 713 – 724, Aug. 2004.

[68] K. Parhi and D. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *Computers, IEEE Transactions on*, vol. 40, no. 2, pp. 178 –195, Feb 1991.

[69] K. Ebcioğlu, "A compilation technique for software pipelining of loops with conditional jumps," *SIGMICRO Newsl.*, vol. 19, pp. 36–41, September 1988. [Online]. Available: http://doi.acm.org/10.1145/62185.62191

[70] A. Aiken and A. Nicolau, "Optimal loop parallelization," in *Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation*, ser. PLDI '88. New York, NY, USA: ACM, 1988, pp. 308–317. [Online]. Available: http://doi.acm.org/10.1145/53990.54021

[71] M. Stoodley and C. Lee, "Software pipelining loops with conditional branches," in *Microarchitecture, 1996. MICRO-29. Proceedings of the 29th Annual IEEE/ACM International Symposium on*, Dec 1996, pp. 262 –273.

[72] L.-F. Chao and E. Hsing-Mean Sha, "Scheduling data-flow graphs via retiming and unfolding," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 8, no. 12, pp. 1259 –1267, Dec 1997.

[73] L.-F. Chao, "Scheduling and behavioral transformation for parallel systems," Ph.D. dissertation, Department of Computer Science, Princeton, NJ, USA, 1993, uMI Order No. GAX93-34171.

[74] L. Lucke, A. Brown, and K. Parhi, "Unfolding and retiming for high-level DSP synthesis," in *Circuits and Systems, 1991., IEEE International Sympoisum on*, Jun 1991, pp. 2351 –2354 vol.4.

[75] G. Goossens, J. Vandewalle, and H. De Man, "Loop optimization in register-transfer scheduling for DSP-systems," in *Design Automation, 1989. 26th Conference on*, June 1989, pp. 826 – 831.

[76] N. Park and A. Parker, "Sehwa: a software package for synthesis of pipelines from behavioral specifications," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 7, no. 3, pp. 356 –370, Mar 1988.

[77] OpenCores, http://opencores.org/.

[78] "Nangate 45nm open cell library," http://www.nangate.com.

[79] S. Yang, I. Jones, and M. Greenstreet, "Synchronizer performance in deep sub-micron technology," in *Asynchronous Circuits and Systems (ASYNC), 2011 17th IEEE International Symposium on*, April 2011, pp. 33 –42.

[80] J. Zhou, M. Ashouei, D. Kinniment, J. Huisken, G. Russell, and A. Yakovlev, "Sub-threshold synchronizer," *Microelectronics Journal*, vol. 42, no. 6, pp. 840 – 850, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026269211000887

[81] J. Zhou, D. Kinniment, G. Russell, and A. Yakovlev, "A robust synchronizer," in *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*, vol. 00, March 2006, p. 2 pp.

[82] T. Sakurai, "Optimization of CMOS arbiter and synchronizer circuits with submicrometer MOSFETs," *Solid-State Circuits, IEEE Journal of*, vol. 23, no. 4, pp. 901 –906, Aug 1988.

[83] J. Horstmann, H. Eichel, and R. Coates, "Metastability behavior of CMOS ASIC flip-flops in theory and test," *Solid-State Circuits, IEEE Journal of*, vol. 24, no. 1, pp. 146 –157, Feb 1989.

[84] S. Beer, R. Ginosar, M. Priel, R. Dobkin, and A. Kolodny, "The devolution of synchronizers," in *Asynchronous Circuits and Systems (ASYNC), 2010 IEEE Symposium on*, May 2010, pp. 94 –103.

[85] Y. Lin, M. Hutton, and L. He, "Special section on field programmable logic and applications - statistical placement for FPGAs considering," *Computers Digital Techniques, IET*, vol. 1, no. 4, pp. 267 –275, July 2007.

[86] H. Yu, Q. Xu, and P. Leong, "Fine-grained characterization of process variation in FPGAs," in *Field-Programmable Technology (FPT), 2010 International Conference on*, Dec. 2010, pp. 138 –145.

[87] L. Cheng, J. Xiong, L. He, and M. Hutton, "FPGA performance optimization via chipwise placement considering process variations," in *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, Aug. 2006, pp. 1 –6.

[88] P. Jones, J. Moscola, Y. Cho, and J. Lockwood, "Adaptive thermoregulation for applications on reconfigurable devices," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, Aug. 2007, pp. 246 –253.

[89] M. Happe, A. Agne, and C. Plessl, "Measuring and predicting temperature distributions on FPGAs at run-time," in *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, 30 2011-Dec. 2 2011, pp. 55 –60.

[90] P. Mangalagiri, S. Bae, R. Krishnan, Y. Xie, and V. Narayanan, "Thermal-aware reliability analysis for platform FPGAs," in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 722–727.

[91] S. Bhoj and D. Bhatia, "Thermal modeling and temperature driven placement for FPGAs," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, May 2007, pp. 1053 –1056.

[92] A. Kumar and M. Anis, "IR-drop management in FPGAs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 6, pp. 988 –993, June 2010.

[93] J. Lamoureux and S. Wilton, "On the interaction between power-aware FPGA CAD algorithms," in *Computer Aided Design, 2003. ICCAD-2003. International Conference on*, Nov. 2003, pp. 701 – 708.

[94] E. Stott and P. Cheung, "Improving FPGA reliability with wear-levelling," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, Sept. 2011, pp. 323 –328.

[95] F. Bruguier, P. Benoit, P. Maurine, and L. Torres, "A new process characterization method for FPGAs based on electromagnetic analysis," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, Sept. 2011, pp. 20 –23.

[96] A. N. Nowroz and S. Reda, "Thermal and power characterization of field-programmable gate arrays," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, ser. FPGA '11. New York, NY, USA: ACM, 2011, pp. 111–114.

[97] S. Lopez-Buedo, J. Garrido, and E. Boemo, "Dynamically inserting, operating, and eliminating thermal sensors of FPGA-based systems," *Components and Packaging Technologies, IEEE Transactions on*, vol. 25, no. 4, pp. 561 – 566, Dec 2002.

[98] R. Mukherjee, S. Mondal, and S. Memik, "Thermal sensor allocation and placement for reconfigurable systems," in *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on*, Nov. 2006, pp. 437 –442.

[99] K. M. Zick and J. P. Hayes, "On-line sensing for healthier FPGA systems," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, ser. FPGA '10. New York, NY, USA: ACM, 2010, pp. 239–248.

[100] B. Valtchanov, V. Fischer, A. Aubert, and F. Bernard, "Characterization of randomness sources in ring oscillator-based true random number generators in FPGAs," in *Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010 IEEE 13th International Symposium on*, April 2010, pp. 48 –53.

[101] J. Franco, E. Boemo, E. Castillo, and L. Parrilla, "Ring oscillators as thermal sensors in FPGAs: Experiments in low voltage," in *Programmable Logic Conference (SPL), 2010 VI Southern*, March 2010, pp. 133 –137.

[102] A. Bsoul, N. Manjikian, and L. Shang, "Reliability- and process variation-aware placement for FPGAs," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, March 2010, pp. 1809 –1814.