UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Ph.D. in Computing

Synthesis of variability-tolerant circuits with adaptive clocking

Alberto Moreno Vega

Advisor: Jordi Cortadella Fortuny

Barcelona, January 2019

Abstract

Improvements in circuit manufacturing have allowed, along the years, increasingly complex designs. This has been enabled by the miniaturization that circuit components have undergone. But, in recent years, this scaling has shown decreasing benefits as we approach fundamental limits. Furthermore, the decrease in size is nowadays producing an increase in variability: unpredictable differences and changes in the behavior of components. Historically, this has been addressed by establishing guardband margins at the design stage. Nonetheless, as variability grows, the amount of pessimism introduced by these margins is taking an ever-increasing cost on performance and power consumption.

In recent years, several approaches have been proposed to lower the impact of variability and reduce margins. One such technique is the substitution of a classical PLL clock by a Ring Oscillator Clock. The design of the Ring Oscillator Clock is done in such a way that its variability is highly correlated to that of the circuit. One of the contributions of this thesis is in the automatic design of such circuits. In particular, we propose a novel method to design digital delay lines with variability-tracking properties. Those designs are also suitable for other purposes, such as bundled-data circuits or performance monitors. The advantage of the proposed technique is based on the exclusive use of cells from a standard cell library, which lowers the design cost and complexity.

The other focus of this thesis is on state encoding for asynchronous controllers. One of the main properties of asynchronous circuits is their ability to, implicitly, work under variable conditions. In the near future, this advantage might increase the relevance of this class of circuits. One of the hardest stages for the synthesis of these circuits is the state encoding. This thesis presents a SAT-based algorithm for solving the state encoding at the state level. It is shown, by means of a comprehensive benchmark suite, that results obtained by this technique improve significantly compared to results from similar approaches.

Nonetheless, the main limitation of techniques at the state level is the state explosion problem, to which the sequential modeling of concurrency is often subject to. The last contribution of this thesis is a method to process asynchronous circuits in order to allow the use of state-based techniques for large instances. In particular, the process is divided into three stages: projection, signal insertion and re-composition. In the projection step, the behavior of the controller is simplified until the signal insertion can be performed by state-based techniques. Afterwards, the re-composition generalizes the insertion of the signal into the original controller. Experimental results show that this process enables the resolution of large controllers, in the order of 10^6 states, by state-based techniques. At the same time, only a minor impact in solution quality is observed, preserving one of the main advantages for state-based approaches.

# *Acknowledgements*

The journey that led to this thesis has not been easy, but it would have been impossible without the help of many people.

First and foremost I want to thank my advisor, Prof. Jordi Cortadella. He is the one who convinced me to follow this path and he has put a lot of effort into helping me getting here. His insight and expertise aided me in more occasions than I can count and his guidance and mentorship has helped me choose when faced with hard decisions.

I would also like to thank those who worked with me, specially in the early days. For the friendly support and help, Pedro López and Marc Lupon. For our discussions in asynchronous circuits Prof. Victor Khomenko and Danil Sokolov. I'm specially grateful to Antoni Roca, for bearing with me and providing great help while I was navigating the maze of commands, options and parameters of the EDA tools.

But this thesis would have been much harder without all the colleagues and people around me. I want to thank my friend Alex Vidal, who started this journey at the same time as me and helped me in many occasions, sometimes with technical insights and sometimes with jokes and light-hearted conversation. Thanks to Javier De San Pedro, for all the tips and tricks on C++ and other languages, as well as all the interesting conversations. Special tanks to all the people from the office S108 of Edifici Ω - Daniel Alonso, Jorge Muñoz, Josep Lluís Berral, Alberto Gutiérrez, Evelia Lizárraga, Eva Martinez, Carles Creus, Albert Vilamala, Alessandra Tosi, M. Àngels Cerveró, Lucas Machado, Tuomas Hakoniemi, Josep Sanchez. Meeting them was truly an experience and I count many of them among the most interesting people I have had the pleasure of meeting.

I cannot forget about the support provided by friends and family. They helped me stay positive and were always there when I just needed a break. I cannot fathom reaching this point without them. Finally, my most heartfelt thanks goes to Andreea Dragomir, for her love and for being there for me even in the most stressful moments. But also for helping me improve my English, in our conversations and, sometimes, even in my writings. Thanks!

This work has been partially supported by funds from the Spanish Ministry for Economy and Competitiveness and the European Union (FEDER funds) under grant TIN2017-86727-C2-1-R, the Generalitat de Catalunya (2017 SGR 786 and FI-DGR 2015).

Contents

Abstract **iii**

Acknowledgements **v**

1 Introduction **1**
1.1 Contributions of this thesis 3
1.2 Structure of this document 5

2 Variability and Ring Oscillator Clocks **7**
2.1 Variability 7
2.2 Static Timing Analysis 8
2.3 Dealing with variability 12
2.4 Adaptive Clocks 14
2.5 Ring Oscillator Clocks 15
2.6 Margins and Derating Factors in Ring Oscillator Clocks 18
2.7 Conclusions 19

3 Synthesis of Digital Delay lines **21**
3.1 Introduction 21
3.2 Nomenclature and overview 25
3.3 Algorithm for gate and wire selection 28
3.4 Cell placement 32
3.5 Configurable Delay Lines 36
3.6 Experimental Results 38
3.7 Conclusions 40

4 State encoding of asynchronous controllers **43**
4.1 Introduction 43
4.2 State encoding for logic synthesis 44
4.3 Overview of the method 48
4.4 Background 50
4.5 SAT formula for the signal insertion problem 56
4.6 Pseudo-Boolean formula for optimization 59
4.7 SAT-based optimization algorithm 63
4.8 Comparison with previous art 64
4.9 Experimental results 65
4.10 Conclusions 69

5 State encoding for large asynchronous controllers 71
5.1 Introduction 71
5.2 Overview 74
5.3 ALTS transformations 75
5.4 CSC resolution algorithm 78
5.5 Exploiting concurrency 84
5.6 Rip-off and re-encode 86
5.7 Experimental results 87
5.8 Conclusions 96

6 Conclusions 97

Bibliography 101

Chapter 1

Introduction

Computers have gone through a spectacular progress since their creation in the mid-20th century. This rate of progress has, arguably, allowed most of the technological changes that our society has undergone in the last decades.

It is impossible to talk about the evolution of computer technology without talking about *Moore's law*. In 1965, Gordon E. Moore presented what would become one of the most famous papers in computer engineering [1]. Based on observations of the achievements made by his company and others in the previous years, he estimated that the number of components per chip would double every year. This was later revised to 2 years and became known as Moore's law. This prediction proved to be accurate and became a fundamental part in the progress of computer technology.

The impact of increasing the number of components per chip, most notably transistors, is multiple. The larger amount of transistors enabled more complex designs with increased performance and capabilities. As the transistors became smaller, switching speed increased and voltage thresholds decreased. In particular, voltage reduction was an important feature that limited power consumption as designs became larger and faster. Possibly even more important, the cost per unit of area on the die remained largely unchanged between generations. In practice, this meant that the price per transistor was effectively halving every 2 years.

Figure 1.1 shows the evolution of performance for processors over the last 40 years. As can be seen, performance has followed an exponential growth since the early processors. Nonetheless, increases in performance started slowing down in the early 2000's and virtually stopped in the last few years. The main reason for this slowdown is attributed to the end of Moore's law.

It was known from the beginning that transistor miniaturization could not go on forever. The ultimate limit for Moore's Law is physics. Eventually, quantum effects on electrons dominate the behavior of transistors at the nanometer scale. But progress slowed

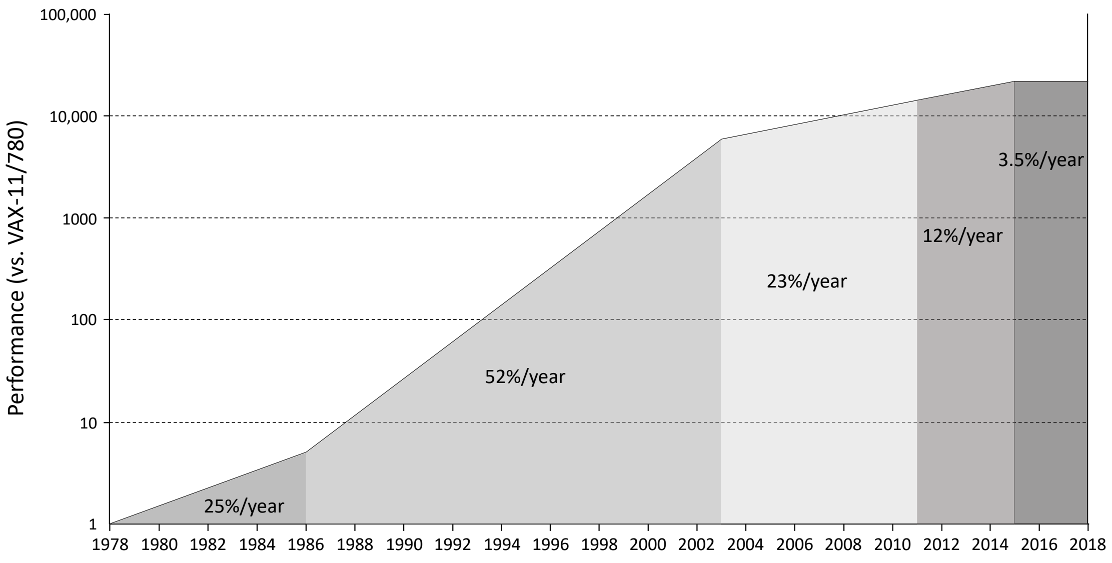


FIGURE 1.1: Growth in processor performance. Source [2].

down long before approaching those limits. When the nodes became small enough, several issues, which had had a very limited impact in the past, became problems that dominated important design decisions. Heat dissipation set a hard limit into the power budget. Transistors were increasingly harder to scale, eventually becoming more expensive than previous generations [3].

One of the biggest challenges over the last years has been variability. As transistors get smaller, the fabrication process becomes more unreliable and the differences with respect to the expected characteristics of components are larger and more frequent. Changes in the environment, like temperature, have an impact in the switching activity of transistors. As voltage thresholds become smaller, sensitivity to voltage supply increases. Unfortunately, voltage noise does not scale at the same pace, which causes supply to be comparatively more unreliable. This effectively prevents newer generations from reducing voltage, establishing power dissipation as a major limitation.

All of this has motivated a large number of research and engineering papers that try to deal with variability. One of such techniques, the substitution of classical PLL clocks by a Ring Oscillator Clock, is a focus of this thesis and is introduced in Chapter 2. An in-depth description and technical details about its implementation can be found in Chapter 3.

The other main focus of this thesis is on asynchronous circuit synthesis. Historically, most computer technology has been implemented using circuits that are synchronized with a global clock signal (synchronous circuits). On the other hand, an asynchronous circuit forgoes the use of a global synchronization signal in favor of localized signals between individual modules.

This class of circuits is not new, yet it presents several advantages with respect to the synchronous counterparts [4]. They typically present lower energy profiles, since the absence of long clock signal lines avoids an unnecessary energy cost. While similar results may be achieved by synchronous circuits, these require the use of complex clock gating techniques. Additionally, asynchronous circuits have much better electromagnetic emission spectrum. The presence of a clock signal in synchronous circuits induces spikes of activity pulses at every clock period, which causes significant correlation between signal edges. On the other hand, asynchronous circuits have a much flatter energy spectrum due to irregular signal patterns.

Besides energy, this class of circuits also have advantages in raw performance. One of the challenges for synchronous circuits is in choosing a clock period. For performance reasons, this period should be as short as possible. Yet, at the same time, the period needs to be large enough to accommodate all the components under any circumstances. This means that the clock cycle needs to be slower than the slowest component performing the longest operation under the worst variability conditions. On the other hand, the presence of local synchronization signals for asynchronous circuits allows every component to always work at its nominal speed. This is especially important in the presence of variability, where unpredictable changes in environmental conditions and voltage requires the presence of conservative margins for classical designs. An important characteristic for asynchronous circuits is that they are implicitly resistant to dynamic variations and always perform correctly, adapting the speed of its components to the current conditions.

1.1 Contributions of this thesis

This thesis addresses the design of circuits resistant to variability. The complexity of current systems makes automatic synthesis an essential part of the design process. This work presents novel techniques for automated synthesis in two fields. For every case, the problems are specified as combinatorial optimization problems and abstracted as graphs. A common characteristic between all the problems is the large search space and the need to implement efficient algorithms. The proposed techniques are shown to have advantages with respect to existing approaches or improve results over them.
The thesis is divided into two main contributions:

- Synthesis of digital delay lines and ring oscillators (Chapters 2 and 3).
- State encoding for asynchronous controllers (Chapters 4 and 5).

Now follows a summary for each of the contributions, as well as a list of publications related to them.

1.1.1 Synthesis of digital delay lines and ring oscillators

Delay lines allow to dynamically estimate the time needed to perform a computation. This becomes more important in the presence of variability, as it allows to sample, at any moment, an accurate approximation for the delay under current conditions. Chapter 3 presents a novel technique for synthesis of digital delay lines. Digital delay lines are designed by using exclusively standard cells from a cell library. On the other hand, more classical designs often make use of custom cells and especially tuned transistors. The advantage of the approach introduced lays in the reduced cost, as well as a simplification in the design and analysis, that the use of standard cells carries. This Chapter is based on the publication:

> [5] A. Moreno and J. Cortadella, "Synthesis of all-digital delay lines," in *23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2017, pp. 75-82

Delay lines can be used for a number of situations and approaches, but in this thesis we propose to build Ring Oscillator clocks. Using Ring Oscillators as clock substitutes allows to instantly react to variability changes, dynamically adapting the clock period to ensure correct operation without sacrificing performance. Chapter 2 describes the use of Ring Oscillator clocks and compares it against similar techniques. This is based on the publications [6, 7].

1.1.2 State encoding for asynchronous controllers

Asynchronous circuits have a completely different design and synthesis methodology to the more common synchronous circuits. An important stage for the synthesis of these circuits is state encoding. This step needs to be performed while maintaining certain notions of equivalence and correctness, such as *speed-independence*, *hazard-freeness* or *branching bisimilarity.* At the same time, different solutions may drastically yield different implementations, each with its own merits and issues.

State encoding is performed differently depending on the underlying model used to describe the behavior of the circuits. In this thesis, we focus on the most generic form for the input/output model by solving this problem for state-based models. These models explicitly represent all the interleaving of concurrent events. This allows finding solutions at a much finer grain than other, more concise, models. Thanks to this, it is often possible to find solutions where other techniques might fail. Additionally, increasing the search space can potentially lead to better solutions. Chapter 4 shows a SAT-based approach to solve this problem. It also serves as an example of the advantages of state-based techniques, as well as its disadvantages. The latter corresponds mainly to the potentially large execution time, due to the enormous size that these representations might have in some instances. This is because of the state explosion, typical for sequential representations of concurrency.

In order to deal with large models, Chapter 5 introduces a technique to simplify controllers which allows state-based techniques to solve them efficiently. This removes the main disadvantage for these techniques while keeping most of their benefits. Additionally, it is also shown how, in certain situations, it is possible to find solutions faster than techniques working on more succinct models, such as Petri nets. The chapters 4 and 5 are based on the publications:

[8] A. Moreno and J. Cortadella, "State encoding of asynchronous controllers using pseudo-Boolean optimization," in *24rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2018, pp. 9-16

[9] A. Moreno and J. Cortadella, "State-based encoding of large asynchronous controllers," *IEEE Access*, vol. 6, pp. 61503-61518, 2018

1.2 Structure of this document

This document is organized into 6 chapters. The current chapter acts as an introduction to the thesis.

Chapters 2 and 3 address the first subject of the thesis. In particular, Chapter 2 introduces some background on variability as well as related work on techniques to reduce its impact. Additionally, an introduction to Ring Oscillator clocks is included. The design of Delay Lines is discussed in depth in Chapter 3.

The second topic of this thesis, state encoding for asynchronous controllers, is discussed in chapters 4 and 5. Chapter 4 discusses a SAT-based technique for encoding, valid for small controllers. On the other hand, Chapter 5 introduces a novel technique that allows state-based techniques, such as the one from Chapter 4, to solve state encoding for large controllers.

Finally, Chapter 6 draws some conclusions about the work presented in this thesis.

Chapter 2

Variability and Ring Oscillator Clocks

This chapter sets the background for variability and discusses why it is such an important concept. There is a large number of techniques developed to reduce its impact and some of the more relevant ones are presented in the following sections. A special focus is given to two techniques: Adaptive clocks and Ring Oscillator Clocks. The latter is one of the central topics of this thesis and shares multiple similarities with the former. As such, a comparison between them is also included. The chapter concludes with a brief description of how derating factors affect the design of Ring Oscillator Clocks.

2.1 Variability

Variability refers to the variations in the properties between different devices or between the same device at different times. These differences appear because of uncertainties in multiple aspects, such as the manufacturing process or environmental changes. There are numerous taxonomies to classify sources of variability. Arguably the most important ones for design are locality (local or global) and variation speed (static or dynamic).

The classification by locality considers two components of variability, global and local. The global component of variability is the one that affects uniformly all the devices. These are differences with respect to the nominal values that every device suffers in a similar magnitude. On the other hand, local variability refers to the component of variability that has a different impact for each device. Nonetheless, some elements of local variability, such as voltage or temperature, often exhibit spatial correlation. This causes spatially close devices to suffer similar variability impact.

Variation speed refers to the pace at which variations occur, most notably static and dynamic variability. Static variability, as its name implies, is not altered with time. This type is often referred to as Process (P) variability, as it depends on the fabrication process. For example, because of imperfections in the manufacturing process, a specific transistor might exhibit a faster (or slower) switching speed than the expected magnitude. Nonetheless, this disparity with respect to nominal values remains constant throughout time. On the other hand, dynamic variability changes over time. These are changes that depend on the environment of the components. Dynamic variability can be further classified according to the source and speed of change. It is usually divided into three categories:

- Aging (A): It refers to the degradation of devices over time, as they become older. As such, it is the slowest changing source for dynamic variability.
- Temperature (T): The temperature that the device operates on is an important source of variation. While it does not change as slowly as aging, it is still considered slow (in the order of milliseconds).
- Voltage (V): Variations in voltage supply have an important impact on the behavior of devices. Voltage variations can happen slowly, in the range of milliseconds, or very fast, in the order of nanoseconds.

Voltage is the most complex source of variability and presents a diversity of components. On the one hand, it has DC components produced by static IR drops that can be either global (off-chip resistance) or local (on-chip power delivery network). On the other hand, voltage variability also has AC components determined by the activity of the system.

2.2 Static Timing Analysis

Static Timing Analysis (STA) is a technique used to verify timing in digital designs. The term *static* indicates that the analysis is done statically, without any dependence on the values propagated through the circuit. Furthermore, this analysis covers every possible path and scenario of a design at the same time, conforming an exhaustive and complete method of verification.

Another timing analysis technique is *simulation*. In this case, a stimulus is applied to input signals in order to verify the behavior. In contrast to STA, simulation requires a large number of test vectors to stimulate inputs, making this kind of analysis only as exhaustive as the number of paths exercised. In practice, this makes simulation suitable only for a limited number of paths or scenarios.

STA also has its own limitations, including false paths, reset sequences, X-handling, etc. [10] which prevents it from completely replacing simulation. Yet, current designs can have billions of gates, making the use of STA necessary for exhaustive verification.

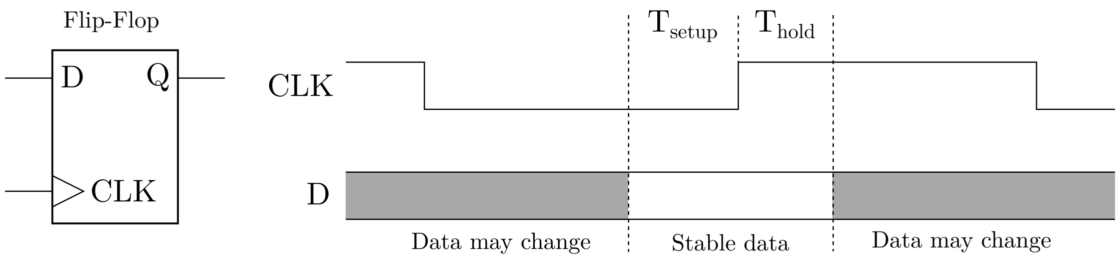


Figure 2.1: Hold and setup requirements for a flip-flop.

Given a design with an input clock and sequential elements, the purpose of STA is to guarantee that the design can operate properly. In particular, ensure that data propagates correctly across different sequential elements at the rated clock period. For this, a set of constraints are checked, where probably the most common ones are setup and hold constraints. The former checks whether data can arrive at a sequential element within the clock period. The latter ensures that the data is held for at least the minimum necessary time required to capture it.

These constraints must take into account requirements that sequential elements, such as flip-flops, have in order to properly work. Figure 2.1 shows a flip-flop, along with waveforms representing the input signals through time. Input D represents the data that the flip-flop captures while signal *CLK* represents the clock signal. When a rising edge of *CLK* is detected, the value of D is propagated to the output Q. But, in order to properly work, the flip-flop requires that the data in D stabilizes at least a T_{setup} time before the clock signal arrives. Similarly, a reliable capture of the data requires D to remain stable for a T_{hold} time after the clock signal rises.

STA must verify setup and hold constraints over all the possible paths between sequential elements. Consider Figure 2.2, which represents a small portion of a digital circuit design. The grayed rectangles represent flip-flops, which encase a combinational circuit. The flip-flops are connected to a clock tree, whose root is a clock generator shown as a box containing a Phase-Locked Loop (PLL). Two paths are highlighted in this figure: the *launch path* and the *capture path*. The launch path starts at the clock generator, traverses the clock tree and the *launch flip-flop*, goes through one of the paths in the combinational logic and ends at the *capture flip-flop*. The capture path starts at the clock generator and ends at the capture flip-flop. Setup and hold constraints check the timing relationship between these two competing paths.

For the setup constraint, STA ensures that a signal in the launch path arrives to the capturing flip-flop before the capture path has propagated the clock signal at the next cycle. This must include, additionally, the setup time of the flip-flop to ensure that data is captured. Mathematically, this can be expressed as:

$$T_{LCK} + T_{LFF} + T_{CP} + T_{setup} < \mathit{CapturePath} + \mathit{Period} \tag{2.1}$$

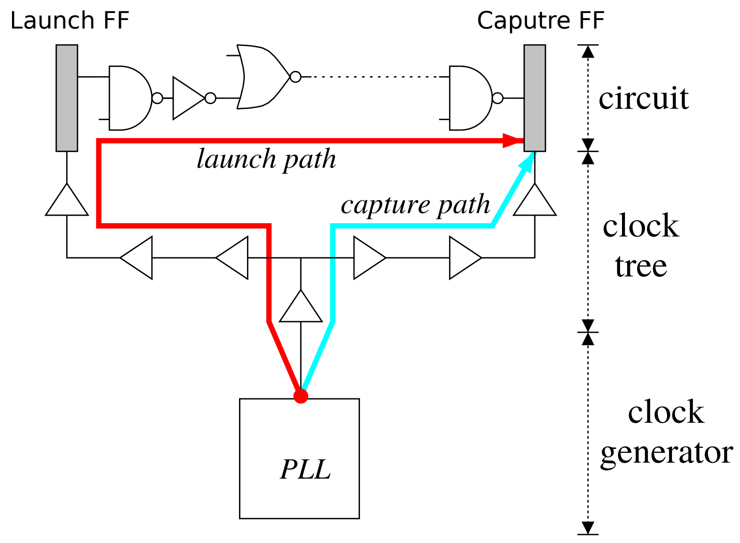


FIGURE 2.2: Paths involved in setup and hold constraints.

where T_{LCK} is the delay of the clock tree of the launch flip-flop, T_{LFF} is the delay of the launch flip-flop, T_{setup} is the setup time of the capture flip-flop and *Period* is the time between cycles of the clock signal (its period). T_{CP} represents the delay of the longest path in the combinational logic, commonly referred to as *critical path*. This last delay is important to guarantee that the setup constraint is honored regardless of what path is exercised in the combinational logic.

The hold constraint checks whether the clock signal propagated through the capture path arrives in time to capture the data from the launch path that started on the previous cycle. In particular, the clock signal must arrive before the data in the input of the capture flip-flop has been overwritten by the launch path of the current cycle, including a margin to account for the flip-flop hold time. This is mathematically expressed as:

$$T_{LCK} + T_{LFF} + T_{MIN} > T_{CCK} + T_{hold} \tag{2.2}$$

where T_{CCK} and T_{hold} are the delay of the clock tree and the hold time for the capture flip-flop respectively. The delay T_{MIN} represents the delay of the combinational logic. In this case, T_{MIN} corresponds to the delay of the shortest path in the combinational logic.

Of both constraints, setup is the only one that is concerned with the clock period and, because of that, it will be the focus of this thesis. Henceforth, we will refer to the setup constraint when we talk about STA, unless otherwise stated. Additionally, for simplicity in the discussions that follow, we introduce the variable *LaunchPath*, which includes all the delays of the launch path assuming the critical path in the combinational logic and adding the T_{setup} time of the capture flip-flop. Similarly, we will refer to the delay of the clock tree for the capture flip-flop as *CapturePath*. In particular:

$$\mathit{LaunchPath} = T_{LCK} + T_{LFF} + T_{CP} + T_{setup} \tag{2.3}$$

$$\mathit{Capture} = T_{CCK} \tag{2.4}$$

We can thus rewrite the setup constraint as:

$$\mathit{LaunchPath} < \mathit{CapturePath} + \mathit{Period} \tag{2.5}$$

The previous inequality must also take into account variability. Given that timing analysis cannot be performed under all possible operating conditions, the conventional approach for modern STA is to analyze the circuit in a discrete set of corners. Each corner defines the values for a set of parameters that model static (P) and dynamic variability (V and T). Henceforth, this parametrized analysis of variability will be referred to as PVT.

From the locality perspective, given a subset of global PVT operating conditions, the components of the circuit also suffer local (on-chip) variations. To cover on-chip variability (OCV for short), corner-based sign-off applies some derating factors to the timing paths of the circuit that scale the delays with regard to other competing paths in the timing constraints.

Finally, clock jitter and any pessimism derived from the inaccuracies and uncertainties of STA must also be modeled. Typically, they are modeled as a fixed margin in the timing constraints. In summary, in modern STA, variability is modeled using:

- **library corners** to model global variability.
- **derating factors** to model on-chip variability.
- **clock uncertainty** to model clock jitter and other inaccuracies.

Timing constraints must hold for all paths and corners under consideration. Given a library corner, the derating factors and clock uncertainty must be incorporated in the setup constraint:

$$\delta_L \cdot \mathit{LaunchPath} < \delta_C \cdot \mathit{CapturePath} + \mathit{Period} - \mathit{Jitter} \tag{2.6}$$

where $\delta_L \geq 1$ and $\delta_C \leq 1$ are the derating factors applied to the launch and capture paths, respectively. Clock jitter must be conservatively subtracted from the period.

A simplification of the model consists of making the derating factors symmetric and reducing the analysis to some ϵ such that:

$$\delta_L = 1 + \epsilon, \qquad \delta_C = 1 - \epsilon \tag{2.7}$$

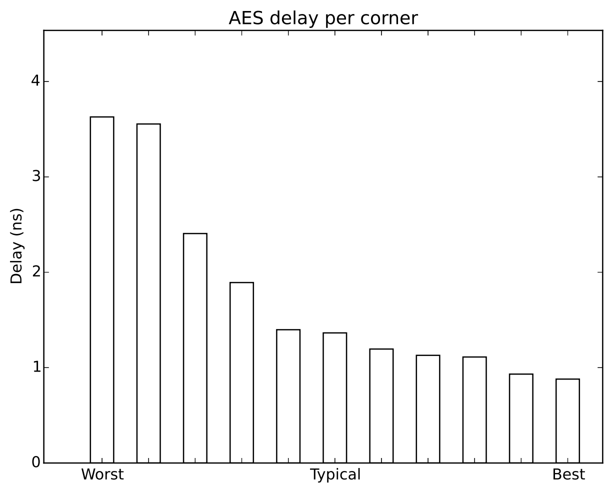


FIGURE 2.3: Critical path delay per PVT corner in AES circuit, implemented in 65nm.

The accuracy on how these derating factors model on-chip variability is crucial. Foundries usually provide conservative values, but more aggressive values can be used if designers have additional knowledge about the behavior and operating conditions of the circuit.

2.3 Dealing with variability

The presence of variability is the main driver for the use of derating factors and margins. This is often the only way to guarantee a valid behavior for a specific circuit. Unfortunately, these margins can have a severe impact on performance and power consumption. Furthermore, as process nodes become smaller, the effects of variability are more noticeable and larger margins need to be used. This effect is so prevalent that it might negate most of the benefits of process scaling.

In order to ensure correct operation, these margins need to account for the worst-case situations, even if they are extremely unlikely. This produces overly pessimistic designs, specified to operate in often unrealistic circumstances. As an example, consider Figure 2.3 which shows the delay of the critical path for each PVT corner for an AES (Advanced Encryption Standard) circuit implemented in a 65nm process. A variability-aware design requires the clock period to be larger than the worst-case corner, after applying derating factors and considering clock uncertainty. In the figure, any clock period smaller than 3.5ns will fail in the worst-case corner. Yet for normal operation, represented by the corner labeled *typical*, this period is extremely conservative. A clock with a period of just 2ns is enough to accommodate the majority of corners and, in particular, the most common ones. This evidences how conservative designs need to be in order to guarantee valid operation.

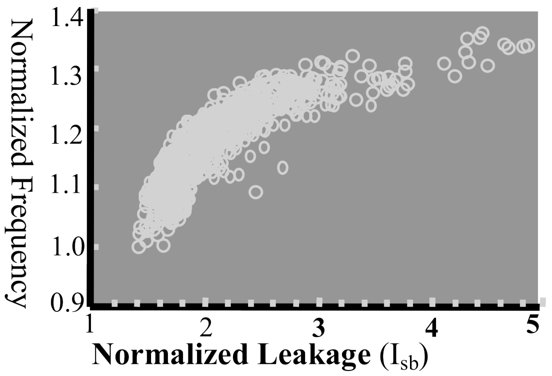


FIGURE 2.4: Process variations for dies in 130nm [11]. Each bubble represents a single die.

This pessimism presents multiple opportunities to decrease margins and thus improve performance or reduce power. Multiple techniques have been proposed in order to exploit these opportunities. Now follows a non-exhaustive list of techniques that aim to reduce pessimism. These are classified depending on whether they address static or dynamic variability. Note that this list is by no means exhaustive and it is only presented as a small sample.

2.3.1 Static variability

As stated earlier, static variability is caused by the manufacturing process. Even before taking into account dynamic variability, the process variations might be responsible for large differences in characteristics between devices. Figure 2.4 shows the distribution of operating frequency and leakage current over a high number of processor dies from Intel in a 130nm process. As can be seen, even a mature process can suffer from a high degree of frequency variation (more than 30%) and an even higher variation in leakage power.

These huge differences are responsible for some of the biggest margins at the design stage. A common technique to address them is *parametric binning* [11–13]. This technique is conceptually simple: after devices are manufactured, these are tested for some parameters, such as frequency (for speed binning) or voltage (for voltage binning). The data obtained from testing is then used to classify chips into different *bins*. For example, it is possible to categorize dies according to frequency and power leakage in Figure 2.4. This is done by defining bins with specific frequency and leakage ranges. Dies that fall below any bin are discarded as defective.

This classification allows identifying dies that have the desired process characteristics. For example, the fastest dies might be used in situations where performance is important, while the chips with less leakage can be directed to more power conscious use cases. On the downside, this technique requires an extensive and costly testing.

2.3.2 Dynamic variability

Multiple techniques have been proposed to mitigate dynamic variability. One of the most notable is Razor [14] and its variants, such as Resilient Circuits [15]. They propose to accommodate the clock period to the typical cases, as opposed to worst-case sign-off. This generates a number of errors which need to be detected and corrected.

In order to detect errors, a *shadow latch* is added to each delay-critical flip-flop (those that might exhibit timing violations). The shadow latch uses a phased clock signal that is delayed enough to ensure that it always captures the correct data, even in worst-case situations. The values captured by the flip-flop and the shadow latch are compared and, if found different, an error is flagged. When that happens, the pipeline must be stalled, the incorrect data purged and the correct value (stored in the shadow latch) propagated. Since the occurrence of an error has a slight performance impact, there is a need to balance the error rate against the clock period. This method can be further enhanced by adding dynamic voltage scaling and regulating the voltage to produce an optimum error rate.

In a similar way, Tribeca [16] also reduces the clock period to work with nominal conditions. The difference is in the use of ECC-protected data to detect errors, as well as local recovery mechanisms.

These techniques produce benefits of over 30% of power reduction. The main drawback is the significant increase in area needed for the detection and correction of errors. Furthermore, they need intricate schemes to cope with the metastability that might occur. Blade [17] reduces the overheads of Razor by incorporating reconfigurable delay lines, error detecting latches and asynchronous structures, yet it still requires intrusive modifications in the circuitry.

More related to the work of this thesis are *Adative Clocking* [18–24] and Ring Oscillator Clocks [6, 7]. These deserve a more detailed discussion and are presented in the following sections.

2.4 Adaptive Clocks

As process nodes are miniaturized, voltage droops in the supply are becoming the most severe dynamic variation. In this context, adaptive clocks were proposed for detection and mitigation [18]. These techniques must be able to anticipate the arrival of voltage droops. When detected, the clock period is increased [19, 20] or altogether stalled [21] until the voltage stabilizes. This helps prevent timing violations, while avoiding conservative margins at the design stage.

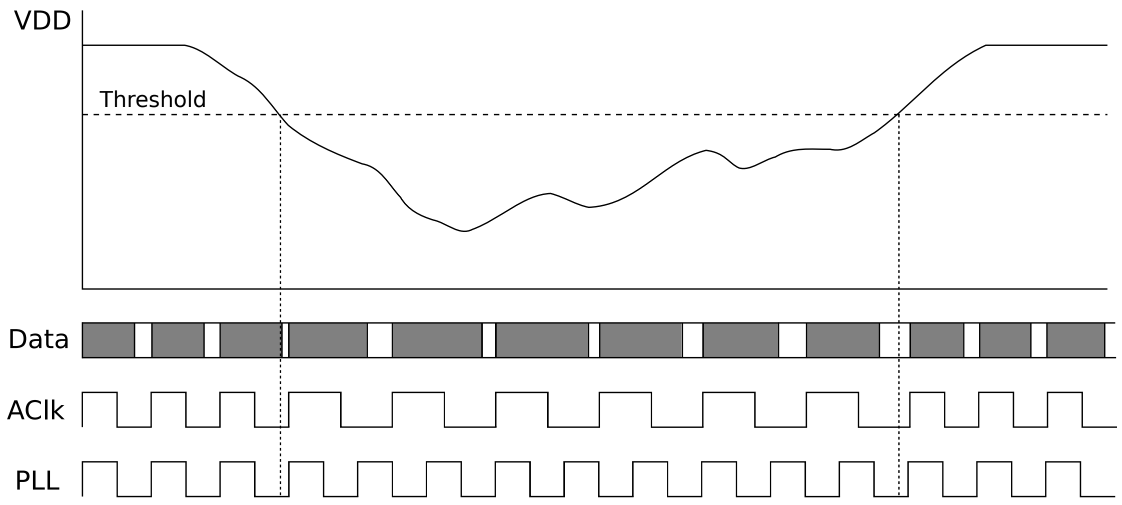


Figure 2.5: Adaptive clock scheme. The shadowed boxes in the Data row show the computation in a critical path. AClk represents the adaptive clock pulses. A rigid PLL is also shown as reference.

Figure 2.5 depicts the basic idea for adaptive clocks. The top part represents the evolution of voltage supply along time. Immediately below, labeled by Data, the computation time evolves with VDD variations. Shadowed boxes correspond to busy logic and empty boxes correspond to idle logic. Near the bottom, AClk represents the pulses of an adaptive clock. For reference, PLL at the bottom shows a standard clock with constant frequency. As the voltage supply is reduced, e.g. due to a droop, the critical path delay increases (shadowed boxes become larger). When the voltage falls under a specific threshold, an adaptive clock increments the clock period to accommodate the increased execution time. The normal frequency is resumed after the voltage raises over the threshold.

The droop detection may be done by perceiving differences or timing violations in delay lines or critical path monitors. The modulation of the clock period can be done directly in the clock generation block, such as using a PLL [18]. Alternatively, it can be done in the clock tree, while the edge is propagating [22, 23].

A common limitation of these schemes is the inability to deal with the full droop spectrum. Usually, they target only the first droop [18, 22] and, sometimes, the second [24]. But the main limitation is the reaction latency to modify the clock frequency. This is addressed by increasing the margins in order to buffer the increasing delays, which can have a considerable impact on performance.

2.5 Ring Oscillator Clocks

Ring Oscillator Clocks (ROCs) [6, 7] share the main characteristic of adaptive clocks: they dynamically change the period in order to account for variability. The difference is

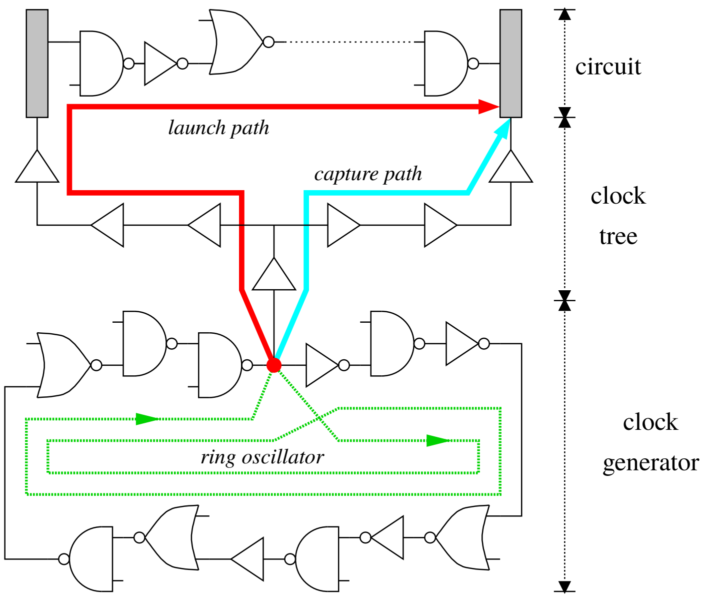


FIGURE 2.6: ROC scheme.

that, for the case of ROCs, the clock generator and the sensing circuit are the same. Because of that, an ROC does not need to anticipate voltage droops or any other variability change. Instead, they react to those changes in the same way and at the same time as the rest of the circuit. This simplifies the design and, more importantly, eliminates the main drawback of adaptive clocks: the reaction latency is always instantaneous.

Figure 2.6 shows the ROC scheme. Notice that the only difference with respect to Figure 2.2 is the substitution of the PLL by a Ring Ocillator (RO) circuit. In this case, a closed loop circuit (the RO) generates a periodic signal in a similar fashion to a classic PLL. Yet, since the RO is constructed out of the same gates than the rest of the circuit, it is subject to the same variability sources.

In general, when designing a classical clock source, it is important to reduce the jitter in order to keep margins small (see constraint (2.6)). This is necessary because jitter is uncorrelated to circuit variability. On the other hand, ROCs intentionally generate jitter that is closely correlated to the variability of a circuit [6]. Thanks to this, a variation that increases the delay in the critical path will similarly increase the period of the clock. It is thus important to maximize the correlation between variability in the RO and in the rest of the circuit. Chapter 3 discusses in detail how to achieve this correlation and describes the design of ROCs.

As an example, Figure 2.7 shows a comparison between a classical PLL and an ROC. The waveforms have been obtained by SPICE simulations in [6] and show a power fluctuation of 30% for illustrative reasons.

As can be seen, the ROC dynamically modifies frequency as the voltage changes. Higher voltages produce higher frequencies and lower voltages reduce, accordingly, the frequency. This allows the ROC of the example to keep an average frequency of 1.55 GHz.

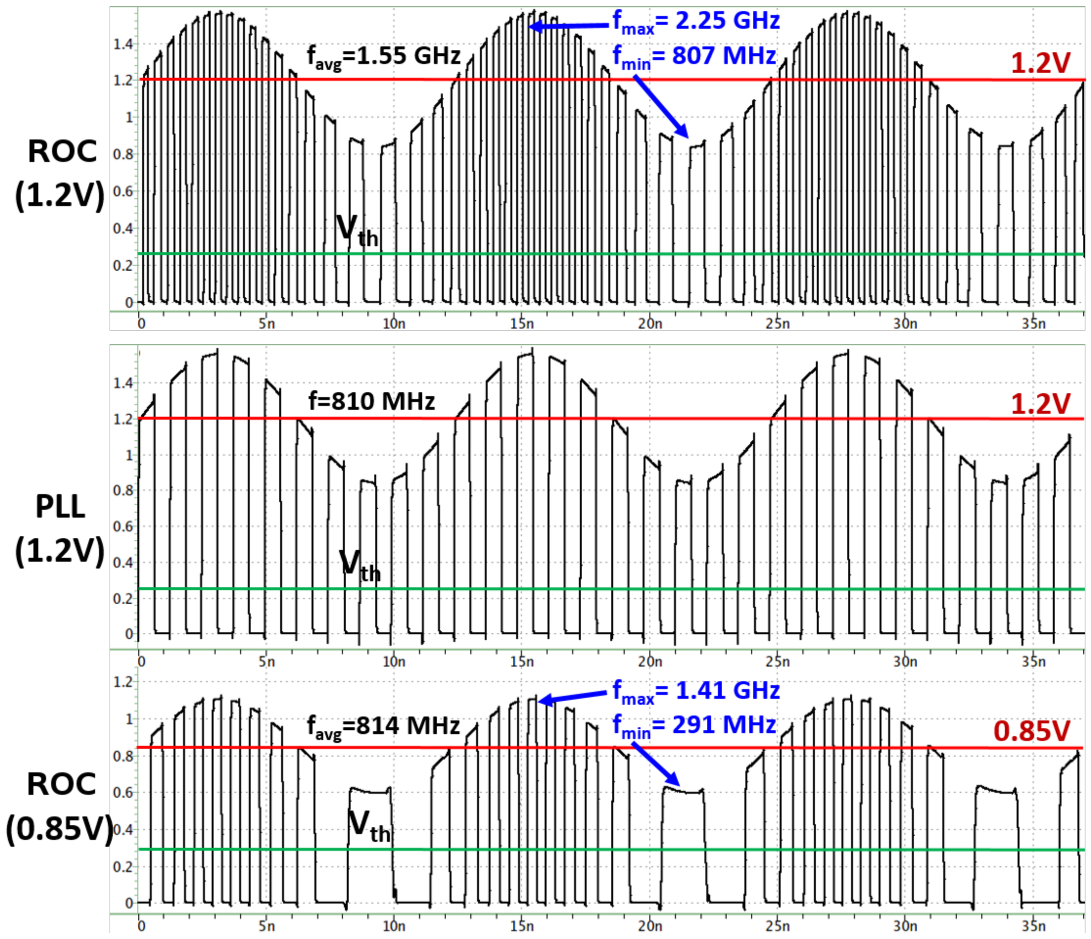


Figure 2.7: Clock generation with PLL and ROC. Source [6].

On the other hand, the PLL needs to be designed for the worst-case frequency of 810 MHz, in order to maintain correct operation. Alternatively, the ROC can be tuned to track a similar average frequency of 814 MHz, but this time with a much lower voltage (0.85v vs 1.2v). This shows how an ROC can be used both for increased performance at iso-power or reduced power at iso-performance.

When comparing ROCs with Adaptive clocks, it is important to take into account that the latter responds differently to fast and slow variability. In particular, fast changes in variability, such as those produced by voltage droops, require low latency to reduce clock frequency or assume larger margins. Figure 2.8, from [6], shows a comparison between ROC and Adaptive clocks (AClk). In order to account for different variations of voltage noise, the figure represents multiple frequencies of noise. Additionally, the reaction time for Adaptive clocks is represented by the number of cycles they needs to modulate the period (1, 2 or 3). It is interesting to note how the performance for adaptive clocks degrades with the higher noise frequency and latency. On the other hand, the ROC reacts immediately to changes in voltage and thus does not require conservative margins.

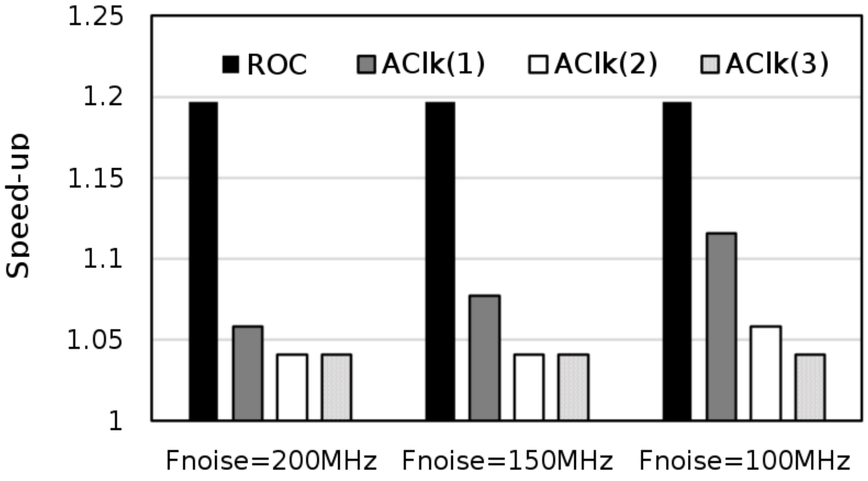


FIGURE 2.8: Speed-ups for AClk and ROC on different frequencies of voltage noise and adapting latencies. Source [6].

2.6 Margins and Derating Factors in Ring Oscillator Clocks

As previously shown, ROCs are well suited to track global variability. Nonetheless, local variability still requires the use of margins. This section presents a comparison between derating factors of PLLs and ROCs. A more in-depth analysis can be found in [7].

Let us first adapt the constraint (2.6) to an ROC. Note that the term $Period - Jitter$ must be substituted by the delay of the RO:

$$\delta'_L \cdot LaunchPath < \delta'_C \cdot (CapturePath + RO) \tag{2.8}$$

with $\delta'_L = 1 + \epsilon'$ and $\delta'_C = 1 - \epsilon'$ being new derating factors.

In this case, the derating factor δ'_C is also applied to the delay of the RO. This is necessary because the RO must be treated as a conventional timing path, which experiments the same sources of variability as the other components of the circuit. In contrast, the jitter disappears from the equation, as it is now part of the delay of the RO.

The derating factors in (2.6) can be different from those in (2.8) since δ'_C and δ'_L must also take into account the spatial correlation between the critical paths and the RO. In particular, the derating factors ϵ for a PLL can be expressed as:

$$\epsilon = \frac{M_{PLL}}{D_L + D_C}$$

with D_L and D_C representing the delay for the capture and launch path, respectively, and M_{PLL} being the margin required to cover on-chip variations for the PLL.

Similarly, the derating factor ϵ' required when performing timing sign-off in an ROC can be defined as:

$$\epsilon' = \frac{M_{RO}}{2D_L + M_{RO}}$$

with M_{RO} being the margin required to cover on-chip variations for the RO.

As discussed in [7], M_{RO} has higher values than M_{PLL}. This is because it needs to cover for on-chip differences in process variability of the RO itself (which is not applicable to PLLs, since they do not suffer process variability). Nonetheless, derating factors applied to ROs are smaller than the ones required by the PLL. Thus, it is possible to use derating factors provided by the foundry, which are valid but conservative from the ROC's point of view.

2.7 Conclusions

Variability and, more importantly, dynamic variability, has a significant impact in performance and power consumption. Multiple approaches have been proposed to deal with this phenomenon.

A promising technique, Ring Oscillator Clocks, presents important advantages with respect to classic PLLs. These advantages come from the correlation in variability between the RO and the rest of the circuit. This allows derating factors to be significantly reduced, as well as giving the capacity to adapt to dynamic variability. Even when comparing with the similar approach of Adaptive clocks, Ring Oscillator clocks can obtain better results due to immediate reaction times.

Chapter 3

Synthesis of Digital Delay lines

A delay line is a device that produces a specific delay in the transmission of a signal. A properly designed delay line can have a high correlation between its variability and the variability of another circuit. This property makes them ideal for, among many other things, the design of Ring Oscillators Clocks.

This chapter introduces an algorithmic approach for the synthesis of delay lines with accurate variability-tracking properties. Additionally, the delay lines are *all digital*, they use exclusively conventional standard cells. Finally, the technique allows the design of configurable lines that can be tuned at runtime.

3.1 Introduction

Delay lines (DLs) have been used in different contexts to track the increasing variability of integrated circuits as CMOS advances to smaller technology nodes. The main goal of a variability-tracking DL is to have a circuit that generates a delay highly correlated with the longest timing path of the system. DLs are often used for post-silicon tuning [25–28], thus enabling the reduction of guardband margins.

One of the potential uses for DLs is in bundled-data (BD) asynchronous circuits [29] where DLs are inserted in the paths of the handshake signals (req/ack) that synchronize different modules of the system. For a correct operation, delays need to be longer than the critical path yet as small as possible to prevent performance degradation.

The notion of Representative Critical Path (RCP) is used in [25] for the synthesis of a DL highly correlated with the circuit delay. Two algorithms are proposed for designing RCPs based statistical static timing models for variability rather than using the more conventional static timing analysis (STA).

Delay monitors, such as canary paths, are also built with DLs [30]. In [31], a comprehensive survey can be found. An algorithmic technique is also introduced for designing Ring Oscillators (RO) for circuit performance monitoring. The approach of [31] simplifies the design of DLs by considering only blocks of identical gates and specific interconnect

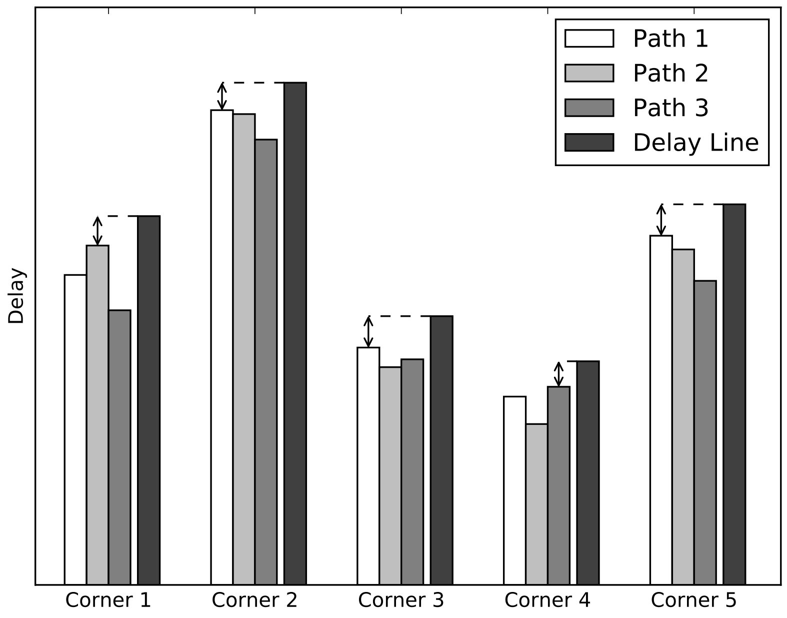


FIGURE 3.1: Several timing paths and delay line at different PVT corners.

lengths as the basic building element. This allows to ignore variations in slew propagation and capacitance between blocks. With this simplification, the problem can be modeled by an integer linear program, at the cost of losing flexibility and precision.

DLs can also benefit from post-silicon tuning to reduce margins after chip manufacturing by adjusting the delays. There are several ways of accomplishing this, including analog and digital techniques. On the analog side, voltage-controlled delay elements are typically used [32–34]. Digitally-controlled delay elements are also possible, for example, by interleaving multiplexers in the DL [28, 35].

Additionally, DLs can be used for the design of Ring Oscillator Clocks (ROCs) [6] introduced in Chapter 2.

All these schemes share the need to accurately match the delay of a DL with timing paths that exhibit PVT variability. Using the terminology of STA, we can say that different timing paths may have different criticality at different PVT corners. Therefore, designing a DL by simply replicating a timing path of the circuit is not always a good approach for delay matching.

A typical situation of time criticality is depicted in Figure 3.1. The histogram shows the delay of three different paths (Path 1-3) at five different PVT corners (Corner 1-5). Due to the different sensitivities to PVT variations, none of them can be taken as a representative of the time criticality of the circuit.

In general, the number of critical paths (with small slack) tends to be extremely large. The main reason is that physical design tools amortize the available time slacks to reduce power by undersizing non-critical gates. In this context, synthesizing a DL that is, at the same time, reliable and accurate at all corners is a challenging problem. The figure also illustrates the desirable properties for a DL:

- It must be longer than the longest delay at any corner (within a certain guardband margin).
- It must be as short as possible to minimize performance degradation.

It is also desirable that DLs can be synthesized and analyzed using conventional standard cell libraries and design automation flows. In this way, the use of DLs can be leveraged in a broader spectrum of application domains.

All the previous requirements pose a challenge for the design of DLs that must address several aspects:

- How to extract the timing characteristics of a circuit at all PVT corners without enumerating all critical paths?
- How to build a chain of heterogeneous standard cells that mimic the timing behavior of the circuit under different PVT conditions?
- How to take into account the variations introduced by the interconnect components (wires)?
- How to make the DL configurable?

In this chapter we propose algorithmic techniques for the synthesis of all-digital DLs with the following characteristics:

- The DLs only contain cells from a standard cell library. No custom cells or analog components are used.
- The timing of the DLs is analyzed by conventional STA tools using library corners and derating factors to model PVT variability.
- The design of DLs includes physical synthesis. In particular, an algorithm for cell placement and derivation of routing constraints for interconnects is proposed.
- The DLs include configurable delays for post-silicon tuning.

The area and power consumption of the DLs can be considered negligible when used for coarse-grain control, e.g., large clock domains or complex functional units.

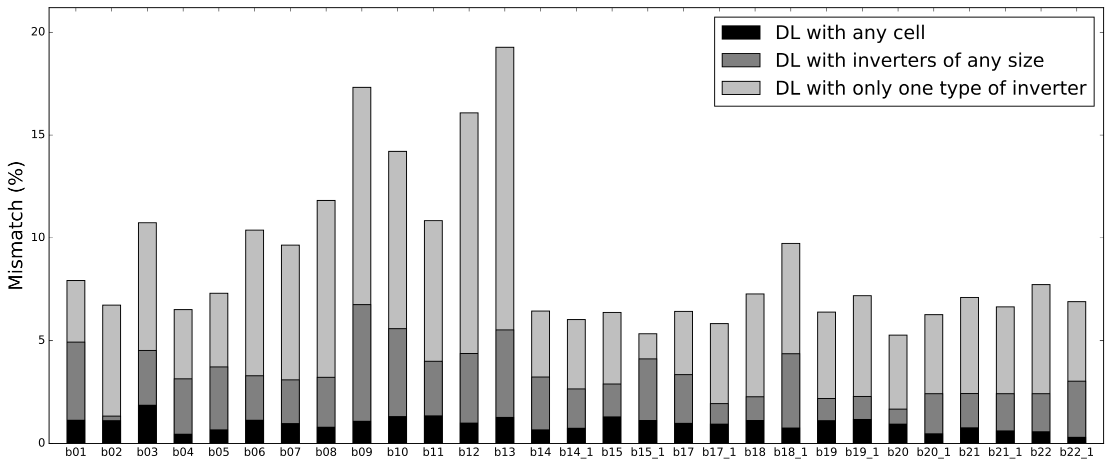


FIGURE 3.2: Accuracy of a DL when using only inverters or any cell in the library. The Y-axis represents average mismatch.

Relevance of the problem

Figure 3.2 illustrates the importance of designing DLs with a mixed combination of gates and wires to accurately track variability at different operating conditions. The algorithm proposed in this chapter was used to generate DLs for the I99T benchmarks from ITC99 [36]. For the selection of the DL cells, three scenarios were considered: (1) only using one type of inverter (i.e. all the cells are identical), (2) using a mix of inverters of different size and (3) using a mix of combinational cells in the standard cell library. The algorithm tried to find the best match for each scenario.

A commercial 65nm library was used to map all reported circuits. Variability was modeled by considering 22 different PVT corners with temperatures in the interval $[-40^oC, 125^oC]$, power supply in the range $[0.9V, 1.32V]$ and process parameters including SS, TT and FF models for transistors. The $RC_{\min}$ and $RC_{\max}$ corners were used to model the variability of the interconnect layers.

The figure depicts the average discrepancy (mismatch %) of the DLs with regard to the delay of the I99T circuits [36] mapped onto the library. The average was calculated over the delays reported by STA (Synopsys PrimeTime [37]) at all available corners of the library (more details in Section 3.6).

It can be observed that matching delays with only one type of inverter may result in a large mismatch (e.g., 20% for b13). Using a mix of inverters with different size may mitigate the mismatch significantly (6% for b13). Finally, the use of a mix of gates with large diversity may contribute to obtaining a good match at all corners (1% for b13).

Table 3.1 also reports the usage of each cell type in the DLs when any type of cell was used for synthesis. We can observe that more than half of the gates are not inverters. It is precisely this diversity what allows a better matching at different operating conditions. It is important to emphasize that the DLs do not only select a mix of gates, but also

Table 3.1: Gate type usage in delay lines

Gate	**Usage**	**Gate**	**Usage**	**Gate**	**Usage**
INV	42.2%	CKND2	4.1%	AO221	0.4%
NAND3	18.6%	NAND4	1.9%	XNOR2	0.4%
NOR2B1	13.5%	NOR2	1.0%	OAI222	0.4%
CKINV	6.6%	NAND2B1	0.6%	OA211	0.3%
NAND2	4.9%	AOI21B20	0.5%	Others	4.6%

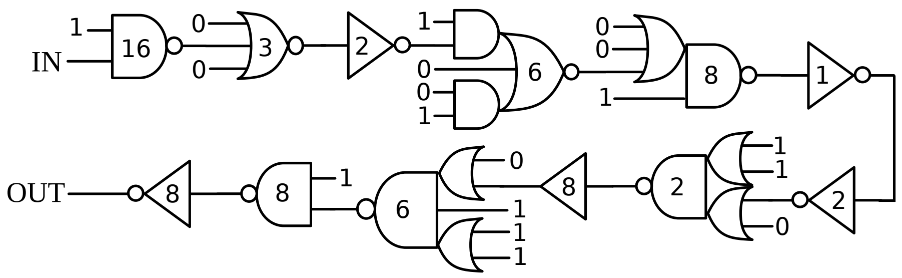


Figure 3.3: DL obtained for matching the delay of b05.

a mix of wire lengths between neighboring cells to account for interconnect variability. The details will be described later in this chapter.

Figure 3.3 depicts an example of DL synthesized to match the delay of one of the experimental circuits (b05). The picture shows the diversity of gates and sizes used in the DL that contribute to mimic the delay of the circuit more accurately at different operating conditions[1].

3.2 Nomenclature and overview

The problem we want to solve is the synthesis of a DL that matches the delay of a circuit under any potential operating condition. In our context, variability is modeled using the same PVT corners and derating factors used during conventional STA to model global variability and on-chip variability (OCV) and previously discussed in Chapter 2.

Using STA, the delay of the most critical path at each corner is obtained. However, any information about the particular critical path that generates the longest delay is disregarded, bearing in mind that each corner may exhibit different critical paths and the particular structure of each critical path is irrelevant. We will call $Dmax_c$ the longest delay at corner c.

[1]The numbers inside the gates indicate the size of the cells.

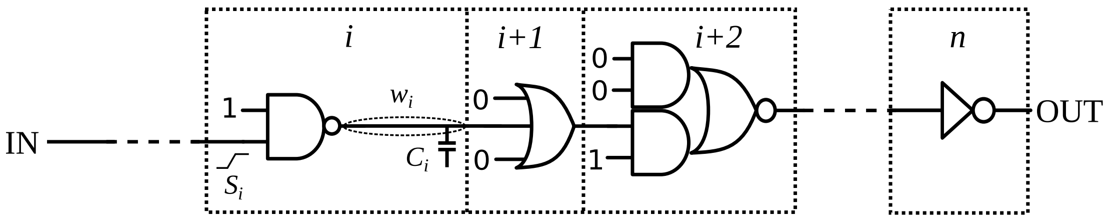


FIGURE 3.4: Stages of a delay line.

TABLE 3.2: Delay line stage parameters

c	Corner from the set of CORNERS
$d_{c,i}$	Delay of stage i at corner c
$C_{c,i}$	Output capacitance of stage i at corner c
$S_{c,i}$	Input slew of stage i at corner c
$w_{c,i}$	Wire delay of stage i at corner c

With this information, and the use of OCV derating factors, a set of target delays T is derived. This set contains, for each corner c, the ideal delay $\tau_c \in T$ of the DL for that corner. Formally:

$$\tau_c = \delta \cdot Dmax_c \tag{3.1}$$

with $\delta > 1$ being the OCV derating factor[2].

Figure 3.4 shows a representation of a DL, which is a sequence of gates and wires. Each pair gate/wire will be referred to as a stage of the DL. Each stage i has an output capacitance C_i, an input slew S_i and a delay d_i. For the sake of simplicity in the nomenclature and the description of the algorithm, we will not distinguish between falling and rising delays. However, they are considered in the actual algorithms and results reported in this chapter.

Each stage i is characterized by the parameters defined in Table 3.2, where c represents the PVT corner at which the parameters are measured. The delay for stage i is computed as the sum of the gate and wire delays. The gate delay and the output slew are functions of the input slew and output capacitance:

$$d_{c,i} = GateDelay_c(S_{c,i}, C_{c,i}) + w_{c,i}$$
$$S_{c,i+1} = Slew_c(S_{c,i}, C_{c,i})$$

The output capacitance for stage i is the sum of the input capacitance for stage $i+1$ and the wire capacitance of stage i.

[2]For simplicity, we assume a unique δ for all corners. However the proposed approach can be easily extended to different values of δ for each corner.

The delay of a DL of n stages at corner c is obtained by adding the delays of all stages:

$$\mathit{delay}_c(DL) = \sum_{i=1}^{n} d_{c,i}$$

Given a set of target delays $\{\tau_c\}$, we can define the *delay mismatch* of a DL at each corner c:

$$\mathit{Mismatch}_c(DL) = \mathit{delay}_c(DL) - \tau_c$$

It is important to notice that the mismatch is computed on a delay that has already been derated to take into account on-chip variability (equation (3.1)). For the algorithm, it is also convenient to define a normalized version of the mismatch:

$$\mathit{NormMismatch}_c(DL) = \frac{\mathit{Mismatch}_c(DL)}{\tau_c} \tag{3.2}$$

Delay constraint: For a DL to be correct, it should be always longer than the target delay. Therefore, the following property must hold for any valid DL:

$$\forall c \in \textsc{Corners}: \quad \mathit{Mismatch}_c(DL) > 0 \tag{3.3}$$

Cost function: A cost function is needed to guide the exploration of the DL structure during the execution of the synthesis algorithm. The cost function is responsible for reducing the mismatch between the DL and the delay of the circuit at different corners. Depending on the context, various cost functions may be considered. Here we present a generalized formulation that can be customized for different application domains:

$$\mathit{Cost}(DL) = \sum_{c \in \textsc{Corners}} \omega_c \cdot \mathit{NormMismatch}_c(DL)^{\alpha} \tag{3.4}$$

with ω_c being a set of weights associated to each corner and α being a constant to control the mismatch diversity. For example, if the designer would prefer to minimize the mismatch at the typical corner, at the expense of having more mismatch at other corners, then the weight ω_{typ} should be increased. If α has a small value (e.g., $\alpha = 1$), then the cost function will guide the exploration towards minimizing the average mismatch over all corners. Instead, if a large value is used (e.g., $\alpha = 3$), the cost function will guide towards minimizing the maximum mismatch over all corners.

The algorithm presented in this chapter is independent of the cost function used for optimization. Therefore, the designer can propose her own customized cost function.

Problem statement: The synthesis problem consists of finding a sequence of gates and wires to build a DL with the following goal:

$$\begin{aligned} \text{minimize:} \quad & Cost(DL) \\ \text{subject to:} \quad & \text{Constraint (3.3)} \end{aligned}$$

Exploration space: The space of potential configurations for a DL is determined by the number of gates in the library (G) and the set of wire configurations for each stage (W). Unfortunately, W is infinite: any sequence of segments of different length using different layers could be potentially used to connect two consecutive gates. To prune the search space, only a small subset of wire configurations is defined a priori to cover a reasonable spectrum of wire lengths.

As an example, the results presented in this thesis have been obtained by considering wires with length 5, 12, 25, 50 and $100\mu m$ (the height of a standard cell is $1.8\mu m$). More details about the gate and wire delay models will be given in Section 3.3.1.

Still, with G and W being finite, the possible set of configurations of a DL with N stages is $(|G| \times |W|)^N$, which makes an exhaustive exploration impractical, bearing in mind that N is unknown and can potentially be a large number (e.g., $N > 50$ in some of the examples reported in Section 3.6).

Overview of the DL synthesis flow: The algorithmic strategy to generate a DL is decomposed into four steps:

1. Selection of gates and wire lengths that will constitute the DL (algorithm presented in Section 3.3).

2. Physical placement of the gates (Section 3.4).

3. Routing of wires using conventional EDA tools.

4. Timing sign-off with STA tools. If some timing violation is produced, the target delay is slightly adjusted and steps 1-4 are executed again until no violation occurs.

Steps 1 and 2, described later in this chapter, use simplified delay models to synthesized the DLs. Step 4 ensures that DLs will always meet constraint (3.3) using the same timing models as the STA tools.

3.3 Algorithm for gate and wire selection

The synthesis of a DL is a combinatorial optimization problem. In this chapter we present a heuristic algorithm based on the *Beam Search* paradigm [38]. Beam Search is based on a constant parameter β (beam width) and explores a search tree by keeping β partial solutions at each level selected from all the solutions generated from the previous

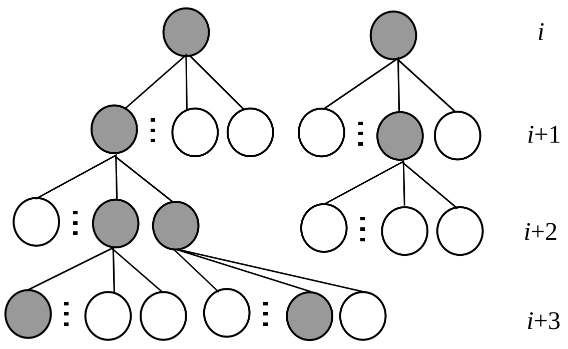


FIGURE 3.5: Beam Search with $\beta = 2$ showing the search levels $i \ldots i+3$. The selected candidates are shadowed.

level. A heuristic cost function is used to select the β best solutions. Figure 3.5 shows a search example with $\beta = 2$.

For the synthesis of DLs, each tree level i stores partial solutions with i gates. When all the generated solutions meet constraint (3.3), the search is aborted and the best solution is delivered.

For the details of the algorithm, it is important to define two new concepts:

- **Partial delay line** (PDL): any DL with zero or more stages.
- **Final delay line** (FDL): any PDL that meets constraint (3.3).

Algorithm 1 shows the main loop of the synthesis algorithm. Initially, the set of PDLs is initialized with a 0-stage DL (level 0 of the search tree) and the set of FDLs is empty. At each iteration of the main loop, each element in PDL is extended by one stage and the β best solutions are stored, according to the cost function described later in Algorithm 3. The extension is performed by the function EXTENDDELAYLINES, described in detail by Algorithm 2.

Algorithm 1: BEAMSEARCH(β)

```
begin
    dl = DL with 0 stages
    FDL = ∅                                          // Set of FDLs
    PDL = {dl}                                       // Set of PDLs
    while not Empty(PDL) do
        // Generate next level of DLs
        PDL, FDL = extendDelayLines(PDL, FDL)
        PDL = select the β best DLs from PDL
    return the best DL in FDL
```

The function EXTENDDELAYLINES generates the next level of the search tree by adding a new gate g and a wire w to the PDLs generated in the previous level. *Wires* contains a discrete variety of wire lengths. The number of new solutions is

$|\text{PDL}| \times |Gates| \times |Wires|$, from which the Beam Search algorithm will select the β best solutions. If any of the new solutions meets constraint (3.3), it is stored in the set of final solutions (FDL).

Algorithm 2: extendDelayLines(PDL, FDL)

```
input  : A set of PDLs and FDLs stored in PDL and FDL, respectively
begin
    newPDL= $\emptyset$                                // Stores next level of the tree
    foreach $dl \in$ PDL do
        foreach $g \in Gates$ do
            foreach $w \in Wires$ do
                dl' = addStage(dl, g, w)
                if dl' meets constraint (3.3) then
                    FDL = $FDL \cup \{dl'\}$
                else
                    newPDL = $newPDL \cup dl'$
    return newPDL, FDL
```

Finally, Algorithm 3 shows the function that computes the cost of each PDL. The function estimates the accuracy of a PDL if the current delays would be scaled linearly to meet constraint (3.3). First, a scaling factor s is calculated that corresponds to the smallest factor required to meet constraint (3.3) at each corner. Next, the normalized mismatch is computed for each corner using the scaled delays. Finally, the cost of the DL is estimated using the scaled mismatches and the cost function (3.4).

Algorithm 3: Cost(*dl*)

```
begin
    // $s'$ is a vector of scaling factors
    foreach $c \in$ Corners do
        $s'[c] = \tau_c / delay_c(dl)$
    $s = \max(s')$                                  // scale factor
    // Vector of scaled normalized mismatches
    foreach $c \in$ Corners do
        $NormMismatch[c] = (s \cdot delay_c(dl) - \tau_c)/\tau_c$
    // Apply the cost function (3.4)
    return CostFunction(NormMismatch)
```

3.3.1 Gate and wire delay models

The models used during the synthesis of DLs are identical to the ones used for STA. Each library uses one or more delay models (e.g., NLDM, CCS, ECSM). One of the simplest is NLDM, which is the one used in this thesis for the experiments. However, the delay model is only used in the evaluation of the cost function and the heuristic exploration can easily adopt any other model. Furthermore, timing sign-off can be done using the preferred model of the user, regardless of the model selected for the design.

For NLDM, each timing arc defines, for each transition direction, a transition time (slew) and a delay table. These tables are indexed by the output capacitance and input slew. The delay and output slew are calculated by a bilinear interpolation.

Libraries also include wire models. The main parameters that affect wire delays are capacitance, resistance and crosstalk. For a set of technological parameters (e.g., resistance/capacitance per unit length), resistance mainly depends on wire length, whereas capacitance and crosstalk are heavily influenced by surrounding wires.

DLs have three interesting properties that simplify delay analysis: (1) the nets do not have glitches, (2) the time windows of the nets do not overlap, and (3) all nets have single fanout[3]. In this way, simple delay models can be used and crosstalk can be ignored by simply isolating or shielding the DL.

In order to simplify the analysis of interconnect delays, the following routing constraints for the DLs are defined:

- Only a small set of metal layers is used. This limits the range of resistivity coefficients and increases the correlation between delay and wire length, regardless the layers used during routing. In our experiments, only three layers were used.
- All the wires must have the same width.
- Large spacing rules between wires are defined. This dramatically reduces coupling capacitance.
- The DL must be isolated from the rest of the circuit, preventing crosstalk.
- The routing algorithm must minimize length. This is important for predicting wire length during placement.

With the previous constraints, wire delay mostly depends on wire length. Thus, simple delay models can be generated by randomly synthesizing DLs and learning a simple statistical prediction model. Figure 3.6 shows a linear regression to estimate capacitance from a set of wires extracted from synthesized DLs, where each point represents a net. A high correlation between capacitance and wire length can be observed ($R^2 = 0.98$). A similar correlation is observed for wire delay predictions.

3.3.2 Implementation details

In the previous sections, it was assumed that the gate delay of a stage only depends on the input slew and output capacitance. In a real scenario, delay also depends on the

[3]Property (3) is not fully complied when synthesizing configurable DLs with muxes (see Section 3.5).

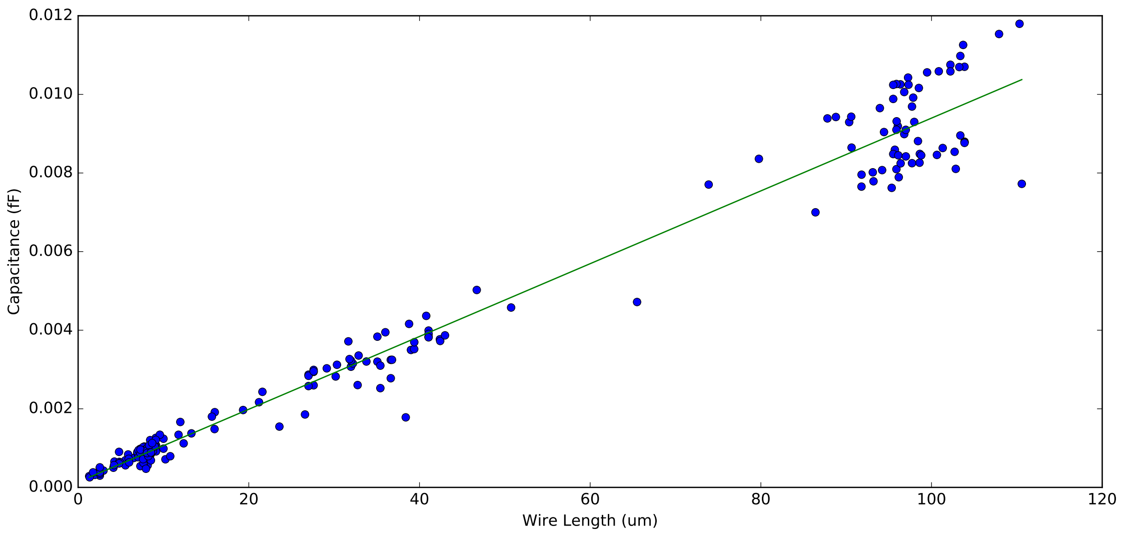


FIGURE 3.6: Linear regression to estimate capacitance as a function of wire length.

transition direction (rising or falling). The previous algorithm can be easily extended to take into account the delays in both directions and select the most convenient.

Each combinational gate may also have multiple input pins and each one may be eligible for the connection with the previous stage. Each input pin and transition direction corresponds to a different timing arc in the gate with different characteristics in slew, capacitance and delay.

The search algorithm can be easily extended to explore any input pin of each combinational gate with both transitions, rising and falling. In fact, any library gate could be considered as a family of gates in which a different pin and transition is selected for the exploration.

The non-selected input pins must be connected to constant values in such a way that the selected input pin is sensitized (e.g., the remaining pins of a NAND gate must be connected to 1).

The DL is treated as a black box during physical design. Therefore, space for the DL must reserved a priori and used for placing its cells, as explained in Section 3.4.

Finally, the algorithm for DL synthesis assumes that the driver of the first gate and the output capacitance of the last gate are known in advance. For example, if the DL implements a delay monitor, there will be flip-flops at the input/output of the DL. In handshake circuits, there might be C-elements.

3.4 Cell placement

The last step for the synthesis of DLs is physical synthesis (placement and routing). Routing is delegated to the existing routing tool in the design flow, but imposing the constraints described in Section 3.3.1.

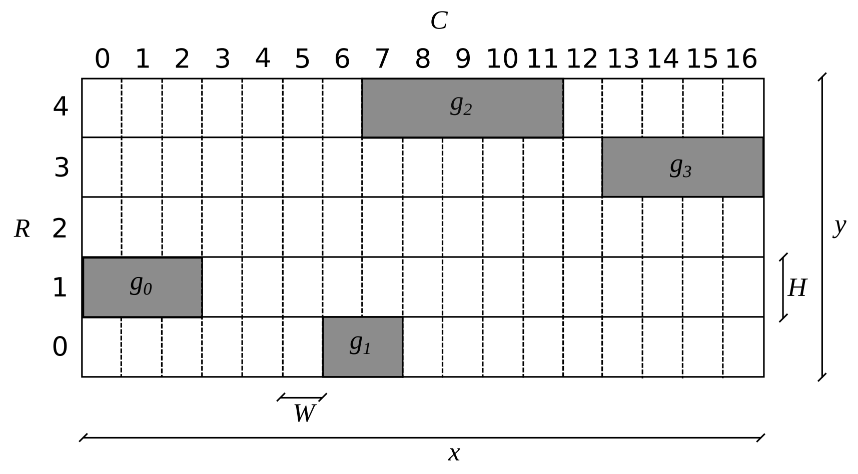


FIGURE 3.7: Placement area for a delay line discretized into a grid.

This section proposes a SAT formulation for the placement step. The SAT formula is guided by the wire lengths of each stage selected during the synthesis step (see Algorithm 2).

Given the routing constraints defined for the wires, that push for the minimization of wire length, it is reasonable to assume that the nets will have a length close the half-perimeter of their bounding boxes. Therefore, the half-perimeter wire length (HPWL) model can be used as a good estimator.

The input of the placement formulation is a DL:

$$g_1 \xrightarrow{l_1} g_2 \xrightarrow{l_2} \cdots \xrightarrow{l_{i-1}} g_i \xrightarrow{l_i} g_{i+1} \cdots \xrightarrow{l_{n-1}} g_n \tag{3.5}$$

where g_i represents the gate at stage i and l_i represents the required wire length from g_i to g_{i+1}.

The gates must be placed in an pre-defined area of the circuit. Figure 3.7 depicts a placement area with width x and height y, divided in R rows and C columns. The height of each row is H and corresponds to the height of the standard cells. The width of each column is W and must be a multiple of the minimum routing granularity specified in the cell library. Hence,

$$R = y/H, \qquad C = x/W$$

Placement problem statement: Given a DL as defined in (3.5), place the gates $g_1 \ldots g_n$ in a gridded area such that:

$$\forall i \in \{1, \ldots, n-1\} : \quad |\text{MANH}(g_i, g_{i+1}) - l_i| < m \tag{3.6}$$

where MANH(g_i, g_{i+1}) represents the Manhattan distance between g_i and g_{i+1}, and m is a tolerance factor between the actual distances and the required distances (ideally, m should be small).

Given that the number of gates is relatively small (few dozens at most), finding an optimal solution may be affordable. We first propose an iterative approximation based on the fact that a SAT formulation can be built for a given value m. The SAT formula is satisfied for all placement solutions for which (3.6) holds.

Main algorithm:

1. A small margin m is defined.
2. A SAT formulation is generated for m.
3. The formula is solved by a SAT solver.
4. If not satisfiable, increase m and go to 2)

The model that satisfies the SAT formula determines the location of each gate.

3.4.1 SAT formulation of the placement problem

We next define the set of variables and clauses of the SAT formula. We assume that each gate g occupies a set of adjacent slots in the grid. We call *size*(g) the number of slots occupied by g (for example, gate g_2 occupies 5 slots in Figure 3.7).

Variables: For every gate g, every row r and every column c, the variable $P^g_{r,c}$ indicates the fact that the leftmost slot of gate g is placed at the grid location (r, c).

Clauses: For simplicity in the representation, a number of definitions follow before describing the clauses.

- The function *Overlap*(g, c) returns, for gate g and column c, the set of columns occupied by g if placed at column c. More specifically:

$$\mathit{Overlap}(g, c) = \{c' : c \leq c' < c + \mathit{size}(g)\}$$

- The function $\textsc{Manh}(r_1, c_1, r_2, c_2)$ returns the Manhattan distance between the grid cells (r_1, c_1) and (r_2, c_2).
- The predicate *validDist*(l, r_1, c_1, r_2, c_2) is true when

$$|\textsc{Manh}(r_1, c_1, r_2, c_2) - l| < m$$

  This predicate is useful to describe all the grid cells that are at a certain distance from another cell. As an example, the darkest cell in the center of Figure 3.8 represents the location of a gate g_i. The shadowed halo around it represents the set of valid locations for gate g_{i+1} assuming that the required wire length is l_i. The width of the halo is determined by the tolerance factor m. This width is represented

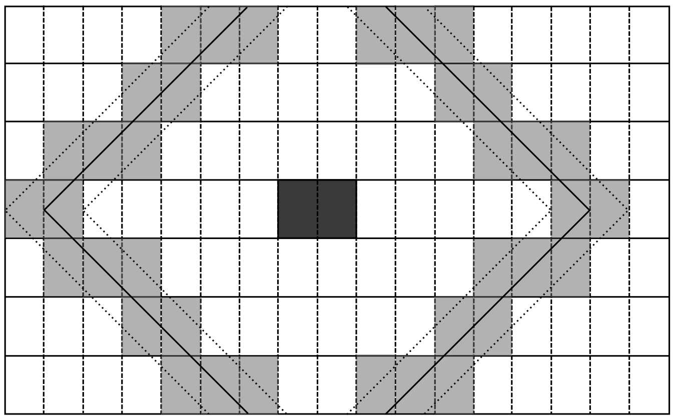


FIGURE 3.8: Valid positions for a gate connected to the one in the middle, as represented by the shadowed boxes.

in the figure as slashed lines and increase the amount of valid locations, allowing slightly closer or more distant positions for g_{i+1}.

We next describe the set of clauses of the SAT formula:

- **Every gate must be placed:** A clause for each gate g with the disjunction of all the possible grid locations, ensuring that it is placed at least in one of them:

$$\forall g : \quad \bigvee_{r,c} P^g_{r,c}$$

- **Every gate can only be placed in one location at most:**

$$\forall g, r_1, c_1, r_2, c_2 \ s.t. \ (r_1, c_1) \neq (r_2, c_2) : \quad P^g_{r_1,c_1} \Rightarrow \neg P^g_{r_2,c_2}$$

- **Gates cannot overlap:**

$$\forall g, g', r, c, c' \ s.t. \ g \neq g', c' \in \mathit{Overlap(g,c)} : P^g_{r,c} \Rightarrow \neg P^{g'}_{r,c'}$$

- **Valid distance for consecutive gates:** For any pair of consecutive gates, g_i and g_{i+1}, the Manhattan distance between them must be close to l_i (within the tolerance factor m), i.e.,

$$\forall g_i, g_{i+1}, r, c, r', c' \ s.t. \ \neg \mathit{validDist}(l_i, r, c, r', c') : P^{g_i}_{r,c} \Rightarrow \neg P^{g_{i+1}}_{r',c'}$$

It is interesting to realize that all clauses have two literals except those that enforce every gate to be placed. The proliferation of 2-literal clauses implies that a lot of decisions are taken without branching (unit propagation). This aspect makes SAT solving more computationally efficient.

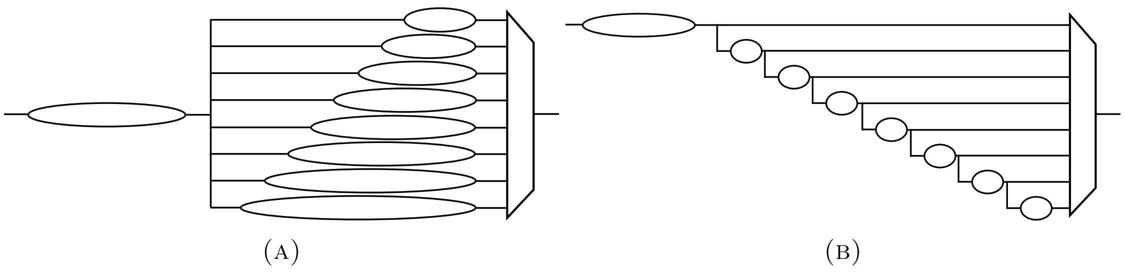


FIGURE 3.9: Mux-based configurable RO architectures.

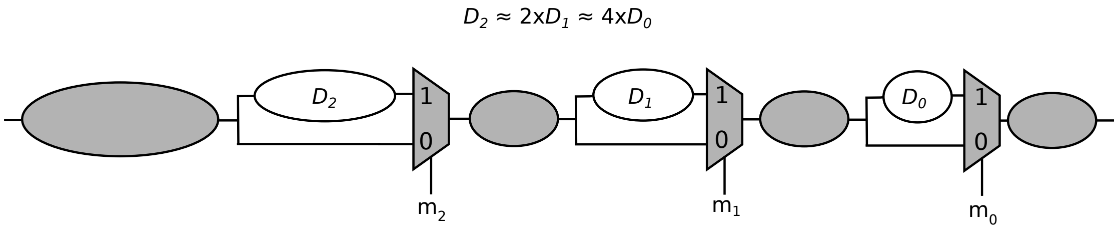


FIGURE 3.10: Distribution of delays in a configurable DL with 3 muxes.

3.5 Configurable Delay Lines

Delay models are just *approximations* of the reality used during synthesis and verification. But reality is only known after manufacturing. Therefore, post-silicon calibration is essential to adjust DLs to the actual delays of the circuit.

Various techniques exist for calibration such as current starved inverters or voltage-controlled delay elements. In our work we propose all-digital solutions that use multiplexers (muxes) that can be found in the cell library. Calibration is performed by a set of *codewords* that control the muxes. It is desirable that the different configurable delays are uniformly distributed across codewords.

Figure 3.9 depicts two possible schemes for configurable DLs. Each of them has a minimum delay shared by all possible configurations. The one in Figure 3.9b is more area efficient but gives less flexibility in synthesizing the delay for each configuration. Another interesting and area-efficient solution commonly used for delay lines is shown in Figure 3.10 (e.g., [28]). For N codewords, this scheme requires $M = \lceil \log_2 N \rceil$ 2-input muxes.

For the synthesis of configurable DLs, two new parameters are introduced:

- The number of codewords (N), usually a power of two.
- The configuration interval, $CI = (CI_{\min}, CI_{\max})$, that defines the range of configurable delays as coefficients over the target delay τ_c at each corner c. For example, $CI = (0.9, 1.1)$ indicates that N different delays must be configured in the interval $(0.9 \cdot \tau_c, 1.1 \cdot \tau_c)$.

In this thesis we focus on the scheme shown in Figure 3.10 as it is the smallest of the three schemes. The synthesis for other schemes requires simple modifications with regard to this one.

The configuration step Δ of the DL is the expected delay difference between two adjacent codewords for a uniform delay distribution. Hence,

$$\Delta_c = \frac{\tau_c \cdot (CI_{\max} - CI_{\min})}{N-1}, \qquad \text{for each } c \in \textsc{Corners}$$

and the delay D_i associated to each mux with control signal m_i is:

$$D_{i,c} = \Delta_c \cdot 2^i, \qquad \text{for } i \in \{0, \ldots, M-1\}, c \in \textsc{Corners}$$

The process of synthesizing a configurable DL is as follows:

- Synthesize a regular DL with target delay $CI_{\min} \cdot \tau_c$, for each corner c, in which M cells are enforced to be 2-input muxes. To mitigate the impact of slew propagation, it is also enforced that there are at least 5 gates between muxes (see the discusison about slew problem at the end of this section). This DL is represented by the shadowed components in Figure 3.10. After this step, D_0, D_1 and D_2 are simply wires.
- The two inputs of each mux cell are connected to the output of the previous cell. One of the inputs will be selected to implement the delay D_i, whereas the other will remain intact.
- Implement each delay D_i as a DL using the same algorithm for a conventional DL. Insert the delay in front of one of the inputs of the mux.

The synthesis of configurable DLs requires some small modifications of the SAT formulation of the placement.

The slew problem. Using muxes introduces a new problem in the synthesis of DLs. The output slew of a mux depends on which input is selected. This effect is multiplicative, as the number of potential slew values at the output of a chain of muxes grows exponentially with the number of muxes.

This problem can be solved using the following property: for a sufficiently long path of gates, the output slew at the last gate is independent from the input slew at the first gate. Typically, and for reasonable slew values, a chain of 5 gates is sufficient to make the output slew virtually independent from the input slew [31].

The synthesis algorithm for configurable DLs guarantees that a minimum number of gates is inserted between two adjacent mux stages, as shown in Figure 3.10. The delay of these gates is accounted within the minimum delay of the DL.

3.6 Experimental Results

DLs have multiple uses, including matched delays for bundled-data asynchronous circuits, canary paths or Ring Oscillators (ROs). This section will focus on using DLs to implement ROs, which implies some particular modifications on the algorithms previously described. A direct application of ROs is in the generation of Ring Oscillator Clocks (ROCs) that was previously discussed in Chapter 2.

An RO is a DL connected in a feedback loop. Few aspects must be considered for the synthesis of an RO:

- A new constraint for the DL algorithm is needed to ensure an odd number of inversions.
- The RO period consists of two oscillations, one for the rising and another for the falling transition. Thus, the period is the sum of the rising and falling delays at each stage.
- The output capacitance of the last cell is the input capacitance of the first cell. Similarly, the input slew of the first cell is the output slew of the last cell.

The experiments have been performed by synthesizing ROCs for several circuits. All the circuits have been implemented in a 65nm commercial library with 22 corners: 11 PVT corners $\times$ 2 interconnect corners (RC_{max} and RC_{min}). Timing results have been obtained by Synopsys PrimeTime [37].

The I99T subset from the ITC99 benchmark suite [36] has been selected for the experiments. Circuits have been divided into two categories: small circuits (`b01-b13`), with size up to a thousand gates, and processors (`b14-b22`) with size up to a few hundred thousand gates [36].

The methodology for the experiments is as follows:

- Layout synthesis has been performed using Synopsys EDA flow.
- PrimeTime has been used to calculate the target period (τ_c) at each corner.
- ROCs have been generated by running the synthesis algorithms for DLs presented in this thesis.
- The reported results have been obtained after layout synthesis using PrimeTime.

The values reported at the tables and charts in this section correspond to the normalized mismatch (in percentage) of the ROC with regard to the target delay of the circuit at each corner (τ_c), as defined in equality (3.2). In the case of configurable ROCs, the mismatch has been calculated for each possible configuration of the delay.

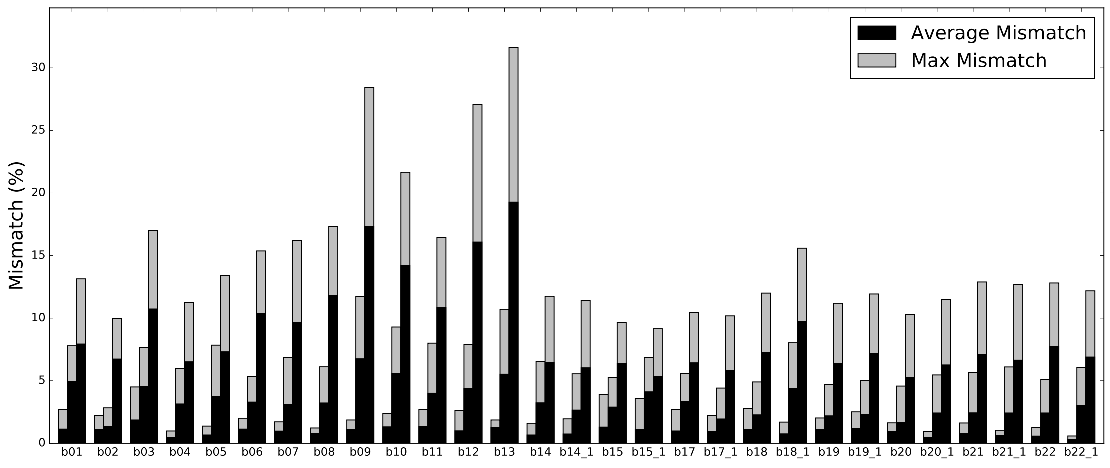


FIGURE 3.11: Accuracy of DLs synthesized with any cell in the library (left bar), inverters of any size (middle bar) and inverters of one size (right bar).

Table 3.3 shows the results for ROCs without muxes. The column *Size* indicates the number of gates of the ROC. Column *Max* reports the maximum mismatch for all corners, whereas *Avg* reports the average mismatch across the 22 corners. *Typ* shows the mismatch at the typical PVT corner, bearing in mind that most dies will fall around this corner after manufacturing. The method guarantees that the mismatch is never negative.

The maximum mismatch is usually below 3% while the average mismatch is around 1% in most cases. This shows that a single DL can track circuit variability very accurately.

Figure 3.11 gives more detailed information about the one shown in Figure 3.2. It can be observed that, when restricting the set of gates used in the DLs, the capability of tracking variability is highly degraded. When only using one type of inverter, the average and maximum mistmatches can go up to 20% and 30%, respectively (see b09, b12 and b13). The inverter used in this experiment corresponds to the most used cell in all synthesized DLs. Even when using all inverters in the library, the mismatch is still substantially larger than when allowing all cells.

Table 3.4 reports results for configurable ROCs with 1, 2 and 3 muxes (M), respectively. In this case, the maximum mismatch corresponds to the one achieved with any of the possible configurations. The average mismatch is the one over all configurations and corners. The mismatch at typical is the average over all the configurations at the typical PVT corner. Only circuits with DLs longer than 25 gates have been synthesized for this case. Small circuits are not appropriate for configurability given that the delay of a single gate is often longer than the minimum configuration step Δ. The configuration intervals used in the experiments were as follows:

TABLE 3.3: Ring Oscillator delay mismatch (%), no muxes.

Circuit	Size	Max	Avg	Typ	Circuit	Size	Max	Avg	Typ
b01	5	2.70	1.13	1.01	b15	27	3.90	1.29	1.65
b02	5	2.23	1.11	1.09	b15_1	26	3.56	1.12	0.85
b03	5	4.50	1.86	0.75	b17	33	2.68	0.98	0.32
b04	20	0.98	0.45	0.40	b17_1	32	2.21	0.94	0.92
b05	12	1.37	0.66	0.70	b18	49	2.77	1.12	0.55
b06	5	2.00	1.13	1.51	b18_1	54	1.69	0.75	0.98
b07	8	1.71	0.97	0.64	b19	79	2.02	1.11	0.92
b08	9	1.22	0.79	0.78	b19_1	65	2.51	1.17	0.62
b09	6	1.86	1.08	0.97	b20	44	1.63	0.94	0.54
b10	6	2.38	1.31	1.81	b20_1	64	0.95	0.47	0.31
b11	13	2.69	1.34	0.88	b21	47	1.62	0.76	0.54
b12	13	2.61	0.99	0.86	b21_1	56	1.04	0.61	0.97
b13	8	1.86	1.27	1.27	b22	46	1.24	0.57	0.32
b14	41	1.60	0.66	0.66	b22_1	59	0.58	0.30	0.40
b14_1	49	1.95	0.74	0.39	**Aver**	**30.55**	**2.07**	**0.95**	**0.81**

	$CI_{\min}$	$CI_{\max}$
M=1	0.975	1.025
M=2	0.925	1.075
M=3	0.825	1.175

The results are reported in Table 3.4. As expected, the mismatch increases with the addition of muxes, since the requirement for introducing muxes reduces the flexibility to find gates that properly track the variability for all configurations. Still, the average mismatch is maintained around 1-2% in most cases, which is a remarkable achievement. This confirms the effectiveness of the synthesis algorithms to find very accurate mixtures of gates even with a large number of configurations.

As an example, Figure 3.3 shows the DL generated for b05 according to the results shown in Table 3.3. In this particular case, an ROC was constructed by connecting the input and the output of the DL.

3.7 Conclusions

The synthesis of DLs for tracking variability is one of the emergent topics as technologies move towards nanometric dimensions. For a widespread use of DLs, it is necessary to provide design automation and schemes that can use the components of the cell libraries.

Table 3.4: Ring Oscillator delay mismatch (%) with 1, 2 and 3 muxes.

	Max mismatch			Avg mismatch			Mismatch @typ		
Circuit	M=1	M=2	M=3	M=1	M=2	M=3	M=1	M=2	M=3
b14	2.06	2.46	3.44	1.04	1.19	2.03	1.00	1.14	1.99
b14_1	2.38	2.03	2.93	1.23	0.92	1.20	0.72	0.56	0.88
b15	3.27	4.82	6.30	1.56	2.50	3.38	1.19	2.02	3.37
b15_1	3.58	4.89	6.85	1.10	1.76	2.96	0.73	1.42	2.53
b17	2.46	4.32	4.94	0.88	2.31	1.97	0.40	1.92	1.78
b17_1	2.77	2.91	2.73	1.46	1.55	1.27	1.40	1.41	1.15
b18	3.73	3.05	4.55	1.81	1.22	1.80	1.42	0.73	1.20
b18_1	1.87	2.26	3.04	0.96	1.02	1.48	1.15	1.09	1.69
b19	2.90	3.65	3.93	1.54	2.25	2.18	1.01	1.80	1.48
b19_1	2.45	2.92	3.53	1.05	1.24	1.78	0.63	0.77	1.29
b20	1.39	1.73	2.04	0.75	0.85	1.08	0.44	0.50	0.77
b20_1	1.55	1.88	2.35	0.70	1.00	1.17	0.39	0.72	0.90
b21	2.14	3.31	3.04	0.96	1.51	1.47	0.81	1.29	1.33
b21_1	1.40	2.09	2.93	0.65	1.13	1.85	0.87	1.25	2.18
b22	2.01	2.54	3.05	1.09	1.49	1.92	0.78	1.04	1.55
b22_1	1.88	2.33	3.60	1.03	1.02	1.71	1.03	0.84	1.64
Aver	**2.36**	**2.97**	**3.70**	**1.11**	**1.44**	**1.83**	**0.87**	**1.16**	**1.61**

This chapter has presented algorithmic techniques to tackle the synthesis of DLs, both at the logic and physical level. Using a variety of gates and wires in the same DL has proved to be essential for an accurate tracking of delays under the presence of variability.

We expect the incorporation of DLs, either playing the role of sensors or clock generators, to be a growing trend in the future. DLs can be used to monitor the potential fluctuations of delays at runtime and adapt the circuit to the varying operation conditions without requiring conservative guardband margins.

Chapter 4

State encoding of asynchronous controllers

This chapter shifts the focus of the thesis towards asynchronous controllers. In particular, it introduces a method to perform state encoding at the state level. This technique leverages the use of SAT in order to encode the problem and find the solution, if it exists. An additional process of optimization guarantees that the solutions are optimal with respect to a cost function.

4.1 Introduction

State encoding is one of the critical problems during the synthesis of asynchronous control circuits. Several methods have been proposed in the past, either for circuits working in fundamental mode [39] or input/output mode [40], among others. In the latter case, the concurrency between input and output events imposes more severe constraints on the insertion of internal signals to disambiguate encoding conflicts. What makes encoding difficult is the preservation of the implementability properties of the specification (e.g., consistency and persistence) after the insertion of new events.

In this thesis we will face the encoding problem in its most generic form, i.e., using state-based models (state graphs) in which all possible interleavings of concurrent events are explicitly represented. State graphs (SGs) can be derived from higher level formalisms such as Signal Transition Graphs (STGs) or Burst-Mode (BM) machines.

The space of configurations for state encoding is huge and similar solutions may result in significantly different logic complexity. One of the challenges in solving the problem is finding low-complexity correct solutions.

This Chapter proposes an approach based on satisfiability (SAT) with two main features: (1) all possible solutions for the encoding problem are represented by one

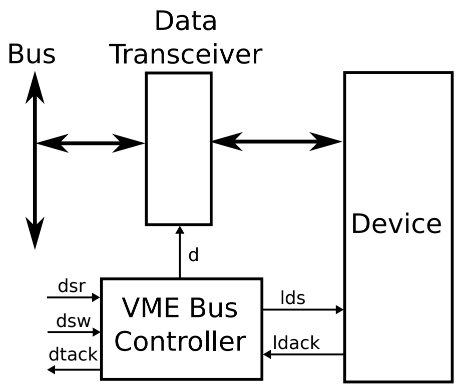


FIGURE 4.1: VME bus controller interface diagram.

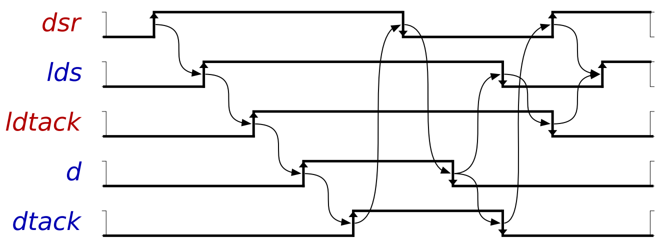


FIGURE 4.2: VME bus controller timing diagram.

Boolean formula and (2) simple estimators of logic complexity are added to the formula in such a way that high-quality solutions can be obtained by Pseudo-Boolean optimization.

The work goes beyond a previous SAT-based approach presented in [41], both in the space of explored solutions and in the estimation of logic complexity. The results obtained by our method shows that still a tangible margin for improvement was left by the best previous approaches implemented in petrify [40] or MPSAT [42].

4.2 State encoding for logic synthesis

State encoding is a necessary step of logic synthesis. It is relevant to recall the full process in order to contextualize the proposed work. In this section, we describe and summarize all the basic steps for logic synthesis of asynchronous controllers.

A very comprehensive and detailed explanation for logic synthesis can be found in [4]. In this overview, we will make use one of the classical examples from [4], the VME bus controller. Figure 4.1 shows a block diagram representing a VME bus controller.

The role of this controller is to open and close the data transceiver according to a protocol for reading and writing data on a device. The arrows shown in Figure 4.1 represent signals that go into and out of the controller. Input signals conform the information that the circuit has of the outside, usually referred to as *environment*. On the other hand, output signals need to be generated by the controller. Figure 4.2 shows a timing diagram for the controller that describes the behavior of the read operation.

$$\begin{tikzcd}
 & s_1 \arrow[dl, "lds^+"'] & s_{14} \arrow[l, "dsr^+"'] & s_{13} \arrow[l, "dtack^-"'] \\
s_2 \arrow[d, "ldtack^+"] & s_{12} \arrow[u, "ldtack^-"'] & s_{11} \arrow[l, "dsr^+"'] \arrow[u, "ldtack^-"'] & s_{10} \arrow[l, "dtack^-"'] \arrow[u, "ldtack^-"'] \\
s_3 \arrow[d, "d^+"] & s_9 \arrow[u, "lds^-"'] & s_8 \arrow[l, "dsr^+"'] \arrow[u, "lds^-"'] & s_7 \arrow[l, "dtack^-"'] \arrow[u, "lds^-"'] \\
s_4 \arrow[r, "dtack^+"] & s_5 \arrow[r, "dsr^-"] & s_6 \arrow[ur, "d^-"'] &
\end{tikzcd}$$

FIGURE 4.3: VME bus controller LTS for the read operations.

At first, the controller is in standby. The input signal *dsr* is raised from *low* to *high* to indicate that a read request is being made. This is followed by a request with signal *lds* for the device to perform a data transfer. When the device is ready, it acknowledges the request with *ldtack*. The controller can now safely open the data transceiver by raising signal *d*. While this signal remains on high, the device is directly connected to the bus. The controller now needs to indicate that the read operation is ready to be performed by raising signal *dtack*. The finalization of the operation is signaled when input signal *dsr* is lowered. This allows the controller to close the data transfer by setting *d* to low. Now the controller can signal that the operation is over with signals *dtack* and *lds*. This can be done concurrently or in any order. Lowering *dtack* also indicates that the controller is ready to perform another read operation. From the device perspective, the ending of the operation still needs to be acknowledged by lowering *ldtack*. Until this happens, no new requests can be performed from the controller.

The logic synthesis process is endeavored in going from the specification into a logic description of a circuit, that generates the appropriate output signals from the input signals it receives. In order to do that, we first need to specify the behavior in one of the models for asynchronous synthesis. In this thesis we make use of Labeled Transition System (LTS), which explicitly represents every signal interaction.

Figure 4.3 shows the LTS for the previous specification. In this model, every arrow represents one of the signal events, or transitions, and how the state of the model changes with them. Every label in a transition indicates in which way a signal changes, with the symbol + indicating a rising edge and the symbol − representing a falling edge. Notably, concurrency between events that occur after signal *d* is set to low must be represented by explicitly enumerating all the valid combinations of causality.

As mentioned, every transition implies a change of state for a signal. For example, the transition lds^+ between states s_1 and s_2 implies that the state for *lds* in s_1 is *low* or

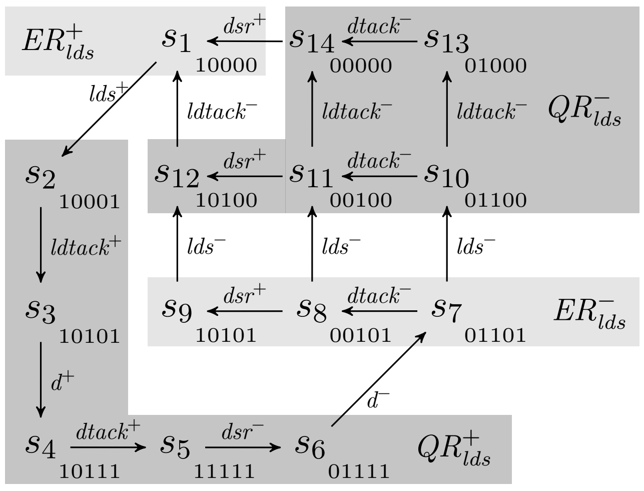


FIGURE 4.4: Binary encoding of states with the vector (*dsr, dtack, ldtack, d, lds*). Shadowed areas indicate regions for *lds.*

0, and the state in s_2 is *high* or 1. Notice that no other signal changes its state between s_1 and s_2. Making use of this, it is possible to infer a binary encoding for every signal and every state. If such an encoding is unique and there are no contradictions, we say that the model is *consistent.* A consistent encoding can be seen in Figure 4.4.

Transitions also allow us to divide the set of states into four regions. For this, we are only interested in output signals, such as *lds*:

- Positive excitation region (ER^+): Those states in which there is a *rise* transition for *lds.*

- Negative excitation region (ER^-): Those states in which there is a *fall* transition for *lds.*

- Positive quiescent region (QR^+): Those states in which there is no transition for *lds*, but its encoding is 1.

- Negative quiescent region (QR^-): Those states in which there is no transition for *lds*, but its encoding is 0.

This division into regions is represented, for signal *lds*, by shadowing states in Figure 4.4. Such a division enables a powerful way to analyze the behavior of the model. In particular, every positive region for *lds* implies that its next state is 1. Conversely, every negative region implies a 0 for the next state. We can easily make use of this by defining *next state functions*, depending on the binary encoding of every state and their region for a given signal.

Figure 4.5 shows the Karnaugh map for the next state function of signal *lds* in which every state is represented by its encoding. The function describes the value that *lds* must transition into at every state. Note that some of the encodings are not defined, *lds=0*

ldtack,d \ *dsr,dtack*	00	01	11	10
00	0	0	-	1
01	-	-	-	-
11	-	-	-	-
10	0	0	-	0

lds=1

ldtack,d \ *dsr,dtack*	00	01	11	10
00	-	-	-	1
01	-	-	-	-
11	-	1	1	1
10	0	0	-	1/0

FIGURE 4.5: Karnaugh map of the next state function for *lds*.

and in these cases the value of the signal is not relevant. These empty slots in the map are called *don't care*, since it does not matter whether they are 0 or 1. By using logic minimization techniques, it is possible to infer a Boolean formula describing a well formed function, and thus the specification that the model implements. Yet, the formula from Figure 4.5 is not well formed. In this case, one of the slots in the map has both 0 and 1 at the same time. Going back to Figure 4.4, it is possible to identify which states are responsible by looking at the encoding. Specifically, s_3 and s_9 share the same encoding, but belong to regions of *lds* with different polarities. This is called a Complete State Coding (CSC) conflict, which causes an irreconcilable ambiguity in the formula.

Intuitively, a CSC conflict indicates that the system lacks enough *memory* to remember the state. In some cases, this can be fixed by adding a new signal to act as additional memory. An example of such a *signal insertion* can be seen in Figure 4.6, along with the new encoding. Note that now every state has a unique encoding and so the next state function is well defined for every signal. After a step of logic minimization, the Boolean formula for every output signal can be obtained and implemented as a circuit. The resulting formula after minimization for all the output signals is:

$$\begin{aligned}
&lds = d + x \\
&dtack = d \\
&d = ldtack \cdot x \\
&x = dsr \cdot (x + \overline{ldtack})
\end{aligned}$$

An important thing to note is that there are multiple ways to insert a signal in order to solve CSC conflicts. Where a signal is inserted can have a potentially dramatic impact on the size and performance of the circuit that implements the formula. Furthermore, there are many restrictions about where a signal can be inserted. The main focus of this and the next chapters of the thesis is on finding the best place to insert a signal. The following section illustrates this and overviews the proposed method.

$$\begin{tikzcd}
\underset{100001}{s'_1} \arrow[d, "lds^+"] & \underset{100000}{s_1} \arrow[l, "x^+"'] & \underset{000000}{s_{14}} \arrow[l, "dsr^+"'] & \underset{010000}{s_{13}} \arrow[l, "dtack^-"'] \\
\underset{100011}{s_2} \arrow[d, "ldtack^+"] & \underset{101000}{s_{12}} \arrow[u, "ldtack^-"] & \underset{001000}{s_{11}} \arrow[l, "dsr^+"'] \arrow[u, "ldtack^-"] & \underset{011000}{s_{10}} \arrow[l, "dtack^-"'] \arrow[u, "ldtack^-"] \\
\underset{101011}{s_3} \arrow[d, "d^+"] & \underset{101010}{s_9} \arrow[u, "lds^-"] & \underset{001000}{s_8} \arrow[l, "dsr^+"'] \arrow[u, "lds^-"] & \underset{011010}{s_7} \arrow[l, "dtack^-"'] \arrow[u, "lds^-"] \\
\underset{101111}{s_4} \arrow[r, "dtack^+"] & \underset{111111}{s_5} \arrow[r, "dsr^-"] & \underset{011111}{s_6} \arrow[r, "x^-"] & \underset{011110}{s'_6} \arrow[u, "d^-"]
\end{tikzcd}$$

FIGURE 4.6: LTS after signal insertion. Encoding after the signal insertion is represented by the vector (*dsr, dtack, ldtack, d, lds, x*).

$$\begin{tikzcd}
s_1 \arrow[r, "r_0^+"] & s_2 \arrow[r, "r_1^+"] & s_3 \arrow[r, "a_1^+"] & s_4 \arrow[r, "r_1^-"] & s_5 \arrow[r, "a_1^-"] & s_6 \arrow[d, "r_2^+"] \\
s_{12} \arrow[u, "a_0^-"] & s_{11} \arrow[l, "a_2^-"] & s_{10} \arrow[l, "r_2^-"] & s_9 \arrow[l, "r_0^-"] & s_8 \arrow[l, "a_0^+"] & s_7 \arrow[l, "a_2^+"]
\end{tikzcd}$$

FIGURE 4.7: Example sequencer.

4.3 Overview of the method

Let us consider the LTS from Figure 4.7 that models the behavior of a controller with $\{r_0, a_1, a_2\}$ and $\{a_0, r_1, r_2\}$ as input and output signals, respectively.

The states s_2 and s_6 share the same encoding. This is evidenced by the complementary subsequence of events $(r_1^+\ a_1^+\ r_1^-\ a_1^-)$ that transition from s_2 to s_6. Solving the encoding problem requires the insertion of a signal x with an event that breaks this subsequence.

In order to break this subsequence, a new event (e.g., x^+) needs to be inserted between r_1^+ and a_1^-. Given that a_1 is an input signal, x^+ can only be inserted immediately before r_1^- in order to maintain the handshaking protocol with the environment. Still, there is some freedom for the insertion of the complementary event x^-. Let us consider three different solutions found in Figure 4.8.

A well-established estimator of the complexity of a logic circuit is the number of literals of the Boolean equations after logic minimization. We use the same criterion in this thesis.

The state encoding problem faces a vast space of solutions. The challenge is to find the ones that lead to simpler circuits without resorting to logic minimization during the exploration.

$$\begin{tikzcd}
s_1 \arrow[r, "r_0^+"] & s_2 \arrow[r, "r_1^+"] & s_3 \arrow[r, "a_1^+"] & s_3' \arrow[r, "x^+"] & s_4 \arrow[r, "r_1^-"] & s_5 \arrow[r, "a_1^-"] & s_6 \arrow[d, "r_2^+"] \\
s_{12}' \arrow[u, "a_0^-"] & s_{12} \arrow[l, "x^-"'] & s_{11} \arrow[l, "a_2^-"'] & s_{10} \arrow[l, "r_2^-"'] & s_9 \arrow[l, "r_0^-"'] & s_8 \arrow[l, "a_0^+"'] & s_7 \arrow[l, "a_2^+"']
\end{tikzcd}$$

(A)

$$\begin{tikzcd}
s_1 \arrow[r, "r_0^+"] & s_2 \arrow[r, "r_1^+"] & s_3 \arrow[r, "a_1^+"] & s_3' \arrow[r, "x^+"] & s_4 \arrow[r, "r_1^-"] & s_5 \arrow[r, "a_1^-"] & s_6 \arrow[d, "r_2^+"] \\
s_{12} \arrow[u, "a_0^-"] & s_{11} \arrow[l, "a_2^-"'] & s_{10}' \arrow[l, "r_2^-"'] & s_{10} \arrow[l, "x^-"'] & s_9 \arrow[l, "r_0^-"'] & s_8 \arrow[l, "a_0^+"'] & s_7 \arrow[l, "a_2^+"']
\end{tikzcd}$$

(B)

$$\begin{tikzcd}
s_1 \arrow[r, "r_0^+"] & s_2 \arrow[r, "r_1^+"] & s_3 \arrow[r, "a_1^+"] & s_3' \arrow[r, "x^+"] & s_4 \arrow[r, "r_1^-"] & s_5 \arrow[r, "a_1^-"] & s_6 \arrow[d, "r_2^+"] \\
s_{12} \arrow[u, "a_0^-"] & s_{11} \arrow[l, "a_2^-"'] & s_{10} \arrow[l, "r_2^-"'] & s_9 \arrow[l, "r_0^-"'] & s_8' \arrow[l, "a_0^+"'] & s_8 \arrow[l, "x^-"'] & s_7 \arrow[l, "a_2^+"']
\end{tikzcd}$$

(C)

Figure 4.8: Valid signal insertions for sequencer of Figure 4.7.

This chapter proposes a SAT-based approach in which the main contribution is the incorporation of logic complexity estimators in the same formula. The most important estimator used in this thesis is the number of *essential literals*. Informally, if the encoding of two states, s_1 and s_2, only differs in one signal value (e.g., $z = 1$ in s_1, $z = 0$ in s_2), and s_1 and s_2 belong to the on- and off-set of the next-state function for signal x, respectively, then z is essential for x, i.e., z must be in the support of x. The important aspect is that the presence of essential literals is a local property (between pairs of states) that can be efficiently encoded in a Boolean formula. Moreover, the number of essential literals can be minimized by using Pseudo-Boolean optimization [43].

We have observed that there is a very high correlation between the number of essential literals and the final literals of a function represented as a factored form. Figure 4.9 depicts a plot comparing essential vs. literals after logic synthesis for a large number of controllers. The solid line represents the ideal prediction (essential = actual). The red dashed line represents a linear regression ($R^2 = 0.91$), that indicates that the number of essential literals is a good estimator.

The following table reports the logic equations for the previous solutions of Figure 4.8. The number of essential literals is represented in brackets and is a lower bound (and a good estimator) of the number of literals of the equations.

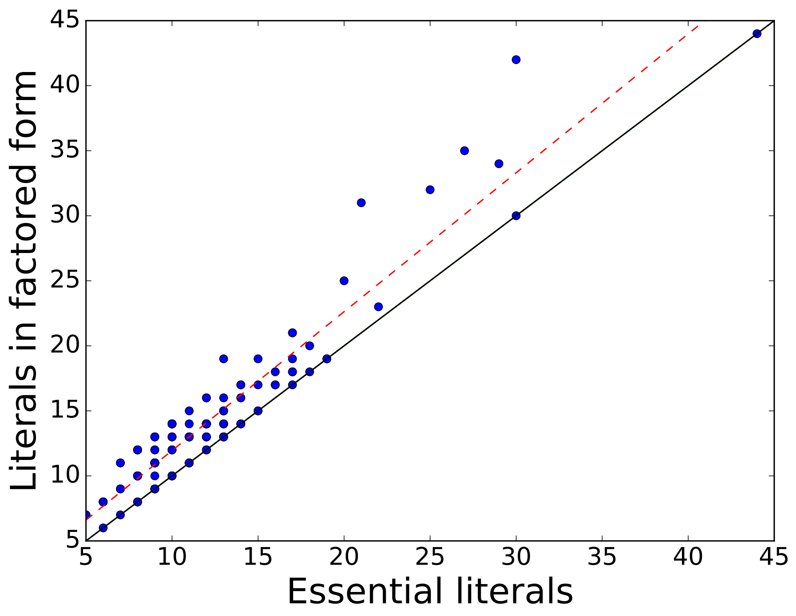


FIGURE 4.9: Essential literals vs. literals in factored form.

	Solution (4.8a)	Solution (4.8b)	Solution (4.8c)
$r_1 =$	[2] $r_0\bar{x}$	[2] $r_0\bar{x}$	[2] $r_0\bar{r}_2\bar{x}$
$r_2 =$	[3] $r_0\bar{a}_1x$	[2] $\bar{a}_1x$	[3] $\bar{a}_1x + r_0a_2$
$a_0 =$	[2] $a_2 + a_0x$	[1] a_2	[2] $a_2\bar{x}$
$x =$	[3] $a_1 + a_2 + \bar{a}_0x$	[3] $a_1 + r_0x$	[3] $a_1 + a_2x$

Besides essential literals, there are other estimators that also have some correlation with the complexity of the logic: size of the don't care set and number of entry points of the excitation regions. These estimators will be discussed later in this chapter.

4.4 Background

This section reviews some known concepts on Boolean functions, asynchronous LTSs and speed-independent circuits. Additionally, it revisits the notion of branching bisimilarity to characterize systems that are behaviorally equivalent. Some of the following definitions only become important in Chapter 5, but are included here as reference.

4.4.1 Boolean Functions

An incompletely specified function (ISF) is a functional mapping $F : \mathbb{B} \rightarrow \{0, 1, -\}$, where $\mathbb{B} = \{0, 1\}$ and '$-$' represents the *don't care* (DC) value. The subsets of $\mathbb{B}^n$ in which F has the 0, 1 and DC values are called the OFF-, ON- and DC-set, respectively.

Let $F(x_1, x_2, \ldots, x_n)$ be a Boolean function of n Boolean variables. The set $X = \{x_1, x_2, ..., x_n\}$ is the *support* of the function F. A variable $x_i \in X$ is *essential* for function F if there exist at least two elements of $\mathbb{B}^n$, v_1 and v_2, that only differ on the value of x_i, such that $F(v_1) = 0$ and $F(v_2) = 1$.

4.4.2 Asynchronous Labeled Transition System

Most works about state-based encoding for asynchronous circuits use State Graphs (SG). In this thesis we prefer to use the name Asynchronous Labeled Transition System, as it better conveys the notion that they are based on the LTS formalism. Either way they can be derived from higher-level formalisms such as STGs or BM machines. An *Asynchronous Labeled Transition System* (ALTS) is a 4-tuple $A = (S, \Sigma, T, s_0)$ where:

- S is a finite non-empty set of states.
- $\Sigma = \text{In} \cup \text{Out} \cup \text{Int}$ is the set of signals, with In, Out and Int being disjoint sets of input, output and internal signals, respectively.
- $T \subset S \times \mathcal{L}_\tau(\Sigma) \times S$ is the set of transitions, with
  - $\mathcal{L}(\Sigma) = \Sigma \times \{+, -\}$
  - $\mathcal{L}_\tau(\Sigma) = \mathcal{L}(\Sigma) \cup \{\tau\}$
  - For every $(s, a, s') \in T$, $s \neq s'$
  - At most one transition $(s, a, s') \in T$ exists between s and s'.
- s_0 is the initial state.

Henceforth, we will also assume that all states in S are reachable from s_0. The label τ is used to represent a silent (non-observable) event. A τ-free ALTS is an ALTS in which there is no transition with label τ. This is an important property for state-based encoding tools, such as the one presented in this chapter. These tools require either a τ-free ALTS or all τ transitions to be *inert*, i.e., can be hidden while preserving the behavior of the specification.

We denote $(s, a, s') \in T$ by $s \xrightarrow{a} s'$, where $a \in \mathcal{L}_\tau(\Sigma)$ is an event (possibly silent). Rising and falling transitions of signal $a \in \Sigma$ between states s and s' are represented by $s \xrightarrow{a^+} s'$ and $s \xrightarrow{a^-} s'$, respectively. We will sometimes refer to $s \xrightarrow{a^\pm} s'$ as a generic transition of signal a.

We will refer to events that possibly have arbitrarily many τ events interleaved. We use $s \overset{a}{\Rightarrow} s'$ as a possibly empty (ϵ) sequence of transitions with the trace $\tau^* a$. In particular, if $s \overset{\epsilon}{\Rightarrow} s'$ (empty transition) then $s = s'$. Additionally, $\alpha \in \mathcal{L}_\tau(\Sigma)^*$ denotes a sequence of (possibly empty) events, with $\alpha = a_1 a_2 \dots a_n$ and $s \xrightarrow{\alpha} s'$ the sequence of transitions that leads from s to s' by following the events of α. If $s \overset{\alpha}{\Rightarrow} s'$, then τ events may be interleaved between events in the form $\tau^* a_1 \tau^* a_2 \tau^* \dots a_n$.

An event a is *enabled* in state s if there is a transition $s \xrightarrow{a} s'$ for some s'. Furthermore a signal a is enabled in s if $s \xrightarrow{a^{\pm}} s'$ for some s'. A sequence of events $\alpha \in \mathcal{L}_{\tau}(\Sigma)^*$ is enabled in state s if $s \xrightarrow{\alpha} s'$ for some s'.

4.4.3 Branching bisimilarity

Milner proposed *observational equivalence* [44] (or weak bisimilarity) as a branching time semantics to classify systems according to their capability of being distinguishable by an external observer under the presence of unobservable events. *Branching bisimilarity* was later introduced as a stronger equivalence that preserves the branching structure of processes [45]. The difference between both equivalences is very subtle and irrelevant in most practical cases.

Given an ALTS $A = (S, \Sigma, T, s_0)$ we call a relation $R \subseteq S \times S$ a branching bisimulation relation if for all $s, t \in S$ such that sRt, the following conditions hold for all $a \in \mathcal{L}_{\tau}(\Sigma)$ [46]:

- If $s \xrightarrow{a} s'$, then
  - either $a = \tau$ and sRt', or
  - there is a sequence $t \overset{\tau^*}{\Longrightarrow} t'$ such that sRt' and $t' \xrightarrow{a} t''$ with $s'Rt''$.
- Symmetrically, if $t \xrightarrow{a} t'$, then
  - either $a = \tau$ and sRt', or
  - there is a sequence $s \overset{\tau^*}{\Longrightarrow} s'$ such that $s'Rt$ and $s' \xrightarrow{a} s''$ with $s''Rt'$.

Two states s and t are branching bisimilar, denoted by $s \approx t$, if there is a branching bisimulation R such that sRt. Two ALTSs A_1 and A_2 are branching bisimilar, denoted by $A_1 \approx A_2$ if their initial states are branching bisimilar.

4.4.4 State encoding

Signals in an ALTS implicitly assign binary codes to the state. Thus, $s(a) = 1$ or $s(a) = 0$ represent the fact that a has value 1 or 0 in state s, respectively. In particular, $s \xrightarrow{a^+} s'$ implies $s(a) = 0$ and $s'(a) = 1$. Similarly, $s \xrightarrow{a^-} s'$ implies $s(a) = 1$ and $s'(a) = 0$. If $s \xrightarrow{b} s'$, with $b \in \Sigma \cup \{\tau\}$, for any $b \neq a$, then $s(a) = s'(a)$. An ALTS is said to be *consistent* if these rules can be applied to every signal and state without any contradiction. In a consistent ALTS with $\Sigma = \{a_1, a_2, ..., a_n\}$, a code can be assigned to every state: $code(s) = (s(a_1), s(a_2), ..., s(a_n))$.

The positive and negative *excitation regions* of signal a, denoted ER_a^+ and ER_a^- respectively, are the sets of states in which a^+ (for ER_a^+) and a^- (for ER_a^-) are enabled. The positive and negative *quiescent regions* of signal a, denoted QR_a^+ and QR_a^- respectively, are the sets of states in which a is not enabled and has value 1 (for QR_a^+) and 0 (for QR_a^-). For convenience we also define $ER_a = ER_a^+ \cup ER_a^-$ and $QR_a = QR_a^+ \cup QR_a^-$. When referring to individual states, $ER_a^+(s)$, $ER_a^-(s)$, $QR_a^+(s)$ and $QR_a^-(s)$ denote that s belongs to ER_a^+, ER_a^-, QR_a^+ and QR_a^- respectively.

We define $ON_a = ER_a^+ \cup QR_a^+$ and $OFF_a = ER_a^- \cup QR_a^-$. The next-state function of a signal defines its future value in the next stable state. Thus, an enabled signal toggles its value, whereas a stable signal maintains its value. The next-state function for signal a is an ISF defined as follows:

$$\begin{aligned} ONset(a) &= \cup_{s \in ON_a} code(s) \\ OFFset(a) &= \cup_{s \in OFF_a} code(s) \\ DCset(a) &= \mathbb{B}^n \setminus (ONset(a) \cup OFFset(a)) \end{aligned}$$

An ALTS satisfies the *Unique State Coding* (*USC*) property if every state is assigned a unique binary code, i.e.,

$$\forall s, s' \in S : s \neq s' \implies code(s) \neq code(s')$$

An ALTS satisfies the *Complete State Coding* (*CSC*) property if the next-state function for any non-input signal is well defined, i.e.,

$$\forall s, s' \in S, \forall a \in \text{Out} \cup \text{Int} : (s \in ON_a \wedge s' \in OFF_a) \implies code(s) \neq code(s')$$

The *CSC* property is a necessary condition for a specification to be implementable as a circuit. If the previous condition does not apply for the states s, s' and signal a, we say that there is a CSC conflict between s and s'. Furthermore, we say that a has a CSC conflict in s, s' when:

$$CSC_a(s, s') \implies code(s) = code(s') \wedge (s \in ON_a \wedge s' \in OFF_a)$$

Finally, the number of CSC conflicts for signal a is defined as the number of pairs of states s, s' such that a is in CSC conflict.

4.4.5 Speed independence and conflicts

From [44], an ALTS $A = (S, \Sigma, T, s_0)$ is weakly deterministic if, for every state $s \in S$ and for every sequence of events $\alpha \in \mathcal{L}_\tau(\Sigma)^*$, whenever $s_1 \overset{\alpha}{\Rightarrow} s_2$ and $s_1 \overset{\alpha}{\Rightarrow} s_3$ then $s_2 \approx s_3$. For the rest of the thesis, the term *determinism* will refer to weak determinism.

A signal a triggers another signal b if there is a transition $s \overset{a\pm}{\rightarrow} s'$ such that b is enabled in s' and not enabled in s. Conversely, a disables b if b is enabled in s and not in s'. An *ALTS* is said to be *output persistent* if for any pair of signals a and b such that a disables b, then both a and b are input signals.

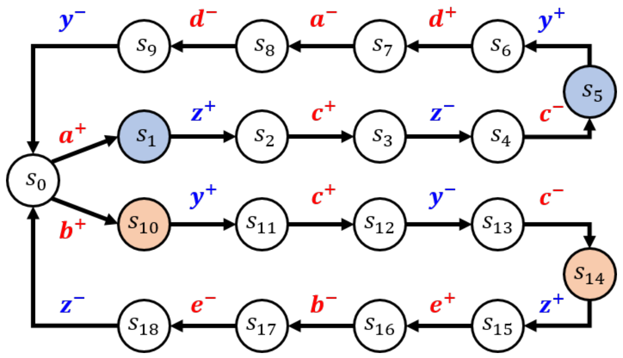


FIGURE 4.10: ALTS with CSC conflicts.

An *ALTS* is said to be *commutative* if for any state s in which $s \xrightarrow{ab} s'$ and $s \xrightarrow{ba} s''$, then $s' = s''$.

A *Well-Formed* ALTS (WF-ALTS) is an ALTS such that is deterministic, commutative and output persistent. An important result on speed independence is the following [40]:

> *A WF-ALTS that satisfies the CSC property is implementable as a speed-independent circuit.*

An additional important property is *input-properness.* An ALTS is input-proper if no internal signal triggers any input signal. This guarantees that the behavior of the environment does not depend on any unobservable signal of the circuit.

A signal a is said to be *in conflict* if there is another signal b such that either a disables b or b disables a. We say that σ is a conflict-free set of signals if every signal $a \in \sigma$ is not in conflict.

Solving the state encoding problem is based on inserting new signals to disambiguate CSC violations. The insertion of new signals proposed in this thesis preserves the conditions for speed-independence and input-properness.

4.4.6 Example

Figure 4.10 depicts an ALTS with five input signals $(a, \ldots, e)$ and two output signals $(y,\, z)$. The pairs of states (s_1, s_5) with code $abcdeyz = 1000000$ and (s_{10}, s_{14}) with code 0100000 are in CSC conflict, since the states of each pair share the code but differ in the onset for y and z.

There are two signals in conflict, a and b, since they disable each other at state s_0. The ALTS is a WF-ALTS since it is deterministic, output persistent and commutative.

4.4.7 Signal Insertion

The insertion of a new *internal signal* is now described. This transformation is always applied to a τ-free WF-ALTS. In this chapter we assume that an ALTS is a τ-free

WF-ALTS whenever we insert a new signal. Signal insertion was proposed in [40, 47] and proved to preserve trace equivalence when the new inserted signal is silent. Since WF-ALTS are also deterministic, signal insertion also preserves branching bisimilarity [48].

Henceforth, the new inserted signal will be named $x \notin \Sigma$, whereas the signals from the original $ALTS$ will be named $a, b \in \Sigma$. The signal insertion process requires all states in S to be partitioned into four sets[1]: ER^+, ER^-, QR^+ and QR^-. These sets will determine the future ERs and QRs of x.

After inserting signal x, some transitions will be delayed (triggered) by x. These are the transitions that *exit ER*:

$$EXIT = \{s \xrightarrow{a} s' \mid (ER^+(s) \wedge \neg ER^+(s')) \vee (ER^-(s) \wedge \neg ER^-(s'))\}$$

Some other transitions will become concurrent with x. These are transitions that will remain inside ER:

$$CONC = \{s \xrightarrow{a} s' \mid (ER^+(s) \wedge ER^+(s')) \vee (ER^-(s) \wedge ER^-(s'))\}$$

The set of new states created by the insertion of x is called $\hat{S}$. For every state $s \in ER$ a new *sibling* state $\hat{s} \in \hat{S}$ is added. New transitions are also added with the new states. In particular, the new sets of transitions are:

$$\begin{aligned}
T_x &= \{s \xrightarrow{x^+} \hat{s} : s \in ER^+\} \cup \{s \xrightarrow{x^-} \hat{s} : s \in ER^-\} \\
T_d &= \{\hat{s} \xrightarrow{a} s' : s \xrightarrow{a} s' \in EXIT\} \\
T_c &= \{\hat{s} \xrightarrow{a} \hat{s'} : s \xrightarrow{a} s' \in CONC\}
\end{aligned}$$

with T_x referring to the transitions between siblings, T_d to the delayed transitions and T_c to the concurrent transitions.

The new $ALTS$ (S', Σ', T', s'_0), obtained after the insertion of x in the original $ALTS$ (S, Σ, T, s_0) is defined as:

- $s'_0 = s_0$
- $S' = S \cup \hat{S}$
- $T' = (T \cup T_x \cup T_d \cup T_c) \setminus EXIT$
- $\Sigma' = \Sigma \cup \{x\}$

Figure 4.11 shows an example of signal insertion on a fragment of an ALTS. On the left, the figure shows the ALTS before signal insertion in which every state has been

[1]When no subscript is specified in the sets, they are assumed to refer to the new inserted signal.

$$\begin{tikzcd}
ER^+\ s_1 \arrow[d, "a^+"'] & s_4\ ER^- & & s_1 \arrow[r, "x^+"'] & \hat{s}_1 \arrow[d, "a^+"'] & s_4 \arrow[r, "x^-"] & \hat{s}_4 \\
QR^+\ s_2 \arrow[r, "b^-"] & s_3\ ER^- \arrow[u, "c^+"] & \Longrightarrow & & s_2 \arrow[r, "b^-"] & s_3 \arrow[u, "c^+"] \arrow[r, "x^-"] & \hat{s}_3 \arrow[u, "c^+"']
\end{tikzcd}$$

Figure 4.11: *ALTS* before and after signal insertion.

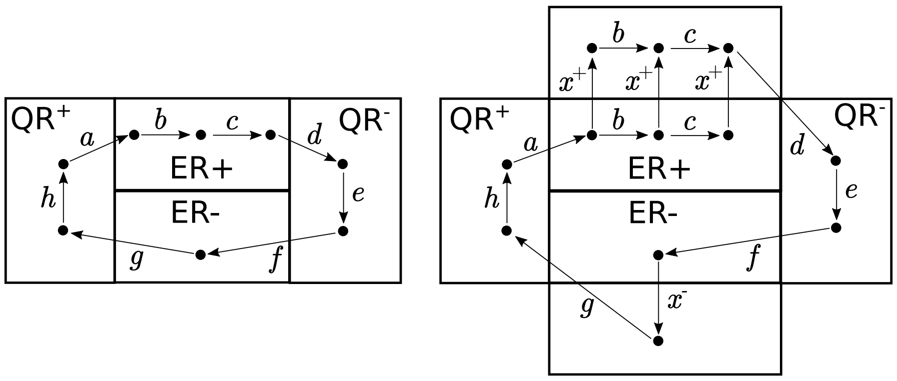


Figure 4.12: Partitioning of the state space into the ER and QR regions of x before (left) and after (right) the insertion.

tagged with one of the *ER*s or *QR*s of x. On the right, states in the ER of x have been duplicated and the new transitions defined accordingly.

A generic view of signal insertion is depicted in Figure 4.12. On the left, the partition of S into the four ER/QR regions of x is shown. On the right, the state space after adding the sibling states is shown.

4.5 SAT formula for the signal insertion problem

The SAT formulation is inspired by the work in [41]. The main difference with respect to the proposed technique is that the CSC problem is solved by inserting signals sequentially rather than inserting all signals at once. This strategy explores a larger space of solutions, since it allows one internal signal to trigger another internal signal. This enables the generation of solutions that cannot be found by the approach in [41].

Signal insertion is based on partitioning the set of states into four subsets as described in the previous section. The SAT formula encodes this partitioning. Additionally, it also encodes the properties for speed-independent implementability: consistency, persistence and input-properness. It is assumed that the original ALTS are τ-free WF-ALTS.

$$\begin{tikzcd}
ER+ \arrow[loop left] \arrow[r] \arrow[dr, bend left] & QR+ \arrow[loop right] \arrow[d] \\
QR- \arrow[loop left] \arrow[u] & ER- \arrow[loop right] \arrow[l] \arrow[ul, bend left]
\end{tikzcd}
\qquad
\begin{tikzcd}
ER+ \arrow[d, "/" marking] & QR+ \arrow[l, "/" marking] \arrow[dl, bend left, "/" marking] \\
QR- \arrow[r, "/" marking] \arrow[ur, bend left, "/" marking] & ER- \arrow[u, "/" marking]
\end{tikzcd}$$

FIGURE 4.13: Consistent (left) and inconsistent (right) transitions.

4.5.1 Boolean variables

Two variables are defined for every state s: $v_1(s)$ and $v_2(s)$. They encode the membership of s to one of the ER/QR regions of x. The encoding used in this work is:

$$\begin{aligned}
ER^+(s) &= v_1(s) \wedge v_2(s) & ER^-(s) &= v_1(s) \wedge \neg v_2(s) \\
QR^+(s) &= \neg v_1(s) \wedge v_2(s) & QR^-(s) &= \neg v_1(s) \wedge \neg v_2(s)
\end{aligned}$$

The total number of variables is $2 \times |S|$. Additional variables, will be required for optimization purposes (see Section 4.6).

4.5.2 Consistency

Constraints to ensure the consistency of x (i.e., x^+ and x^- alternate) must be included in the SAT formula. That means that all paths across the ALTS must visit the insertion regions in the order[2] $ER^+ \to QR^+ \to ER^- \to QR^- \to ER^+ \to \cdots$. Figure 4.13 shows the legal transitions between sets (left) and the illegal transitions (right). The constraint can be formulated as:

$$\begin{aligned}
&\forall s_1 \to s_2 \in T: \\
&\quad \neg(QR^-(s_1) \wedge QR^+(s_2)) \wedge \neg(QR^-(s_1) \wedge ER^-(s_2)) \wedge \\
&\quad \neg(QR^+(s_1) \wedge QR^-(s_2)) \wedge \neg(QR^+(s_1) \wedge ER^+(s_2)) \wedge \\
&\quad \neg(ER^+(s_1) \wedge QR^-(s_2)) \wedge \neg(ER^-(s_1) \wedge QR^+(s_2))
\end{aligned}$$

4.5.3 Persistence

The insertion of a new signal must guarantee that no new non-persistence is introduced. For that, it suffices to look at *diamonds* of concurrent transitions [40, 41]. Figure 4.14 (left) depicts a diamond with a possible assignment of ER/QR regions for signal insertion. On the right, the result after signal insertion is shown. It can be noticed that this insertion does not maintain persistence, e.g., a is enabled in s_1 but not in s_3.

[2]Transitions $ER^+ \to ER^-$ and $ER^- \to ER^+$ are also possible.

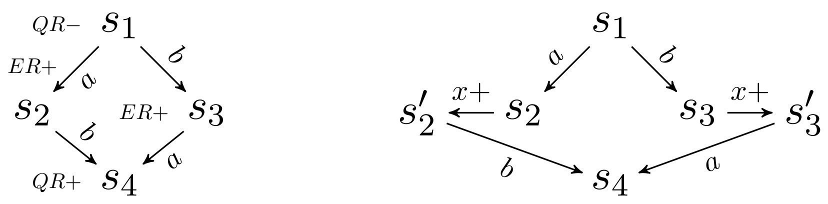


FIGURE 4.14: Non-persistent signal insertion.

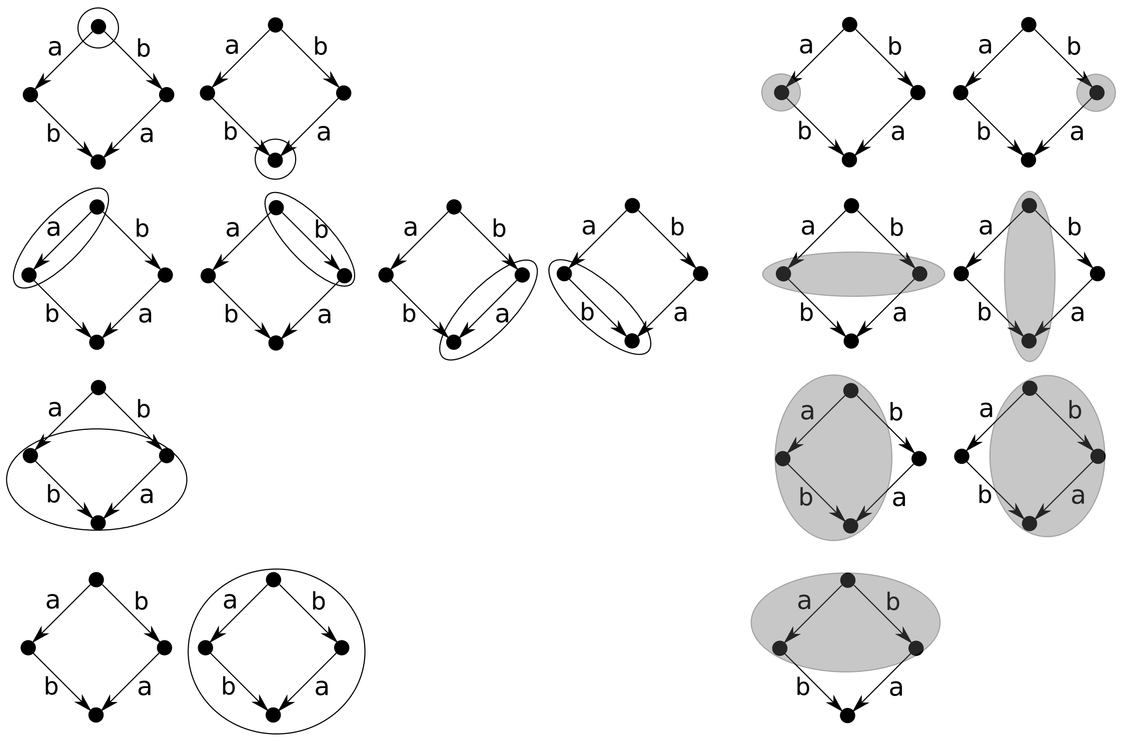


FIGURE 4.15: Persistent insertions are circled on the left. Non-persistent insertions are shadowed in gray.

Figure 4.15 shows all possible allocations of ERs in a diamond. The circled states identify the ER for insertion. Circled regions preserve persistence, whereas shadowed regions do not.

For each diamond $s_1 \xrightarrow{a} s_2 \xrightarrow{b} s_4$ and $s_1 \xrightarrow{b} s_3 \xrightarrow{a} s_4$, persistence can be formulated with the following three constraints:

$$\begin{aligned}
ER^+(s_1) \wedge ER^+(s_4) &\Rightarrow ER^+(s_2) \wedge ER^+(s_3) \\
\neg ER^+(s_1) \wedge \neg ER^+(s_4) &\Rightarrow \neg ER^+(s_2) \wedge \neg ER^+(s_3) \\
ER^+(s_2) \wedge ER^+(s_3) &\Rightarrow ER^+(s_4)
\end{aligned}$$

Similar clauses apply for ER^-.

4.5.4 Input-properness

Input-properness is guaranteed by forbidding x to trigger an input signal, i.e., not allowing any input transitions to exit ER. Formally, for each $s_1 \xrightarrow{a} s_2$, such that a is an input event:

$$(ER^+(s_1) \Rightarrow ER^+(s_2)) \ \wedge \ (ER^-(s_1) \Rightarrow ER^-(s_2))$$

4.6 Pseudo-Boolean formula for optimization

This section introduces the optimization part of the SAT formula for generating high-quality solutions. Optimization is performed by defining a cost function as a linear combination of Boolean variables. This function biases the explored solutions towards disambiguating CSC conflicts with low logic cost. Methods for Pseudo-Boolean optimization can be used to formulate the problem with a linear cost function and still using SAT solving engines [43].

4.6.1 Reduction of CSC Conflicts

After the insertion of a signal x, some of the CSC conflicts will be solved and some will not. We next propose a formulation to quantify the remaining conflicts after the insertion.

Let us call *CSCpairs* the sets of pairs of states with CSC conflicts. For each pair (s_i, s_j) in the previous set we define a new variable $c_{i,j}$ that denotes whether a CSC conflict remains after signal insertion.

A CSC conflict is solved for (s_i, s_j) if the two states have a different value for x. This requires both states to be in different QRs of x. Notice that the presence in some ER means that sibling states would be created that would inherit the original CSC conflict. Thus, for any $(s_i, s_j) \in \mathit{CSCpairs}$:

$$\neg c_{i,j} \ \Leftrightarrow \ [QR^-(s_i) \wedge QR^+(s_j)] \vee [QR^+(s_i) \wedge QR^-(s_j)]$$

USC conflicts may also become CSC conflicts after signal insertion (they are called *secondary conflicts*). They occur when two states still have the same code and x becomes enabled in one of them but not in the other one. While this can be easily modelled as a Boolean formula, these conflicts have a very minor impact and can be ignored in practice.

The total number of conflicts (minus secondary conflicts) that will remain after inserting x can be easily computed as:

$$\mathrm{Conf} = \sum_{(s_i,s_j)\in \mathit{CSCpairs}} c_{i,j}.$$

4.6.2 Estimation of logic: essential literals

The encoding for essential literals is the most elaborate of the ones presented here. Before giving the final encoding, we first need to introduce a series of predicates. Henceforth, the suffix $\pm$ is used to indistinctly refer to the positive and negative regions.

The next predicate indicates that a transition in the original SG is delayed by x after its insertion:

$$\mathrm{Del}_x(s,a) \;\equiv\; \exists s \xrightarrow{a} s' : s \rightarrow s' \in EXIT$$

It can also be interpreted as "x is a trigger of a in s". The following predicates encode the ERs and QRs in the new ALTS based on the original one. The $\widehat{\ }$ symbol indicates that the region refers to the new ALTS after inserting signal x:

$$\begin{aligned}
\widehat{QR}_a^{+}(s) &= QR_a^{+}(s) \vee (ER_a^{-}(s) \wedge \mathrm{Del}_x(s,a))\\
\widehat{QR}_a^{-}(s) &= QR_a^{-}(s) \vee (ER_a^{+}(s) \wedge \mathrm{Del}_x(s,a))\\
\widehat{QR}_a^{\pm}(\hat{s}) &= QR_a^{\pm}(s)\\
\widehat{ER}_a^{\pm}(s) &= ER_a^{\pm}(s) \wedge \neg\mathrm{Del}_x(s,a)\\
\widehat{ER}_a^{\pm}(\hat{s}) &= ER_a^{\pm}(s) \wedge ER_x^{\pm}(s)\\
\widehat{QR}_x^{\pm}(s) &= QR_x^{\pm}(s)\\
\widehat{QR}_x^{\pm}(\hat{s}) &= ER_x^{\pm}(s)\\
\widehat{ER}_x^{\pm}(s) &= ER_x^{\pm}(s)\\
\widehat{ER}_x^{\pm}(\hat{s}) &= \text{False}
\end{aligned}$$

We will use the predicates $\widehat{ON_y}(s)$ and $\widehat{OFF_y}(s)$ to denote the fact that state s belongs to the on- and off-set of signal y, respectively, after the insertion of signal x. They are defined as:

$$\begin{aligned}
\widehat{ON_y}(s) &= \widehat{ER}_y^{+}(s) \vee \widehat{QR}_y^{+}(s)\\
\widehat{OFF_y}(s) &= \widehat{ER}_y^{-}(s) \vee \widehat{QR}_y^{-}(s)
\end{aligned}$$

The following predicates define the encoding of x in a state s after signal insertion:

$$\begin{aligned}
\widehat{ONE_y}(s) &= \widehat{ER}_y^{-}(s) \vee \widehat{QR}_y^{+}(s)\\
\widehat{ZERO_y}(s) &= \widehat{ER}_y^{+}(s) \vee \widehat{QR}_y^{-}(s)\\
\widehat{EQ_y}(s_1,s_2) &= \widehat{ZERO_y}(s_1) \wedge \widehat{ZERO_y}(s_2) \vee \widehat{ONE_y}(s_1) \wedge \widehat{ONE_y}(s_2)
\end{aligned}$$

The Hamming distance between the binary encodings of the encoding of two states, s_1 and s_2, before the insertion of signal x is defined as:

$$d(s_1, s_2) = \sum_{a \in \Sigma} (s_1(a) \neq s_2(a)).$$

Moreover, we define the Boolean predicate $\widehat{d_1}(s_1, s_2)$ to be true if the Hamming distance after the insertion of x is one. This is defined as:

$$\widehat{d_1}(s_1, s_2) \equiv \begin{cases} \mathit{False} & \text{if } d(s_1, s_2) > 1 \\ \widehat{EQ_Y}(s_1, s_2) & \text{if } d(s_1, s_2) = 1 \\ \neg\widehat{EQ_Y}(s_1, s_2) & \text{if } d(s_1, s_2) = 0 \end{cases}$$

This predicate can also be extended and used for sibling states, e.g., $\widehat{d_1}(s_1, \hat{s_2})$.

The basic condition for a signal z becoming an essential literal for signal y is as follows: there must be a pair of states $s_1 \in \widehat{ON_y}(s)$ and $s_2 \in \widehat{OFF_y}(s)$, such that $\widehat{d_1}(s_1, s_2)$ and $s_1(z) \neq s_2(z)$. We can also distinguish between positive and negative essential literals depending on the polarity of the essential literal z with regard to y.

We can now define the basic predicate that represents the fact that two states (or their siblings) with Hamming distance one can be at the on/off-set of y after the signal insertion:

$$\begin{aligned} \mathit{D1}(s_1, s_2, y) \equiv\ & (\widehat{d_1}(s_1, s_2) \wedge \widehat{ON_y}(s_1) \wedge \widehat{OFF_y}(s_2)) \vee \\ & (\widehat{d_1}(\hat{s_1}, s_2) \wedge \widehat{ON_y}(\hat{s_1}) \wedge \widehat{OFF_y}(s_2)) \vee \\ & (\widehat{d_1}(s_1, \hat{s_2}) \wedge \widehat{ON_y}(s_1) \wedge \widehat{OFF_y}(\hat{s_2})) \vee \\ & (\widehat{d_1}(\hat{s_1}, \hat{s_2}) \wedge \widehat{ON_y}(\hat{s_1}) \wedge \widehat{OFF_y}(\hat{s_2})) \end{aligned}$$

Next, the constraint for essential literals is defined, where $E^+_{z\to y}$ and $E^-_{z\to y}$ are new Boolean variables that represent the fact that z is a positive and negative essential literal for y, respectively.

$$\begin{aligned} & \forall s_1, s_2 \in S : \\ & \quad \big(\mathit{D1}(s_1, s_2, y) \wedge s_1(z) = 1 \wedge s_2(z) = 0 \Rightarrow E^+_{z\to y}\big) \ \wedge \\ & \quad \big(\mathit{D1}(s_1, s_2, y) \wedge s_1(z) = 0 \wedge s_2(z) = 1 \Rightarrow E^-_{z\to y}\big) \end{aligned}$$

The number of essential literals after the insertion of x can now be computed as:

$$\text{EssLit} = \sum_{y,z \in \Sigma \cup \{x\}} E^+_{z\to y} + E^-_{z\to y}$$

4.6.3 Don't Care set

A large DC-set increases the opportunities for logic minimization. After the insertion of the new signal, the size of the DC-set depends on the amount of new sibling states, which is determined by the size of the ERs for signal x. A simple way for estimating their size is to count the signals that are concurrent with x after the insertion.

The variables $conc_{a+}$ and $conc_{a-}$ indicate whether there is a transition $a+$ or $a-$, concurrent with x. The following predicates represent the concurrent events with x:

$$\forall s_1 \xrightarrow{a+} s_2 : (ER^+(s1) \wedge ER^+(s2)) \vee (ER^-(s1) \wedge ER^-(s2)) \Rightarrow conc_{a+}$$
$$\forall s_1 \xrightarrow{a-} s_2 : (ER^+(s1) \wedge ER^+(s2)) \vee (ER^-(s1) \wedge ER^-(s2)) \Rightarrow conc_{a-}$$

The number of concurrent signals, highly correlated with the size of the ER_x, is thus computed as:

$$\text{ERsize} = \sum_{a \in \Sigma} (conc_{a+} + conc_{a-})$$

4.6.4 Entry points

We say that s is an entry point (EP) for ER_x^+ if $s \in ER_x^+$ and all its predecessor states are outside ER_x^+ (similarly for ER_x^-). The events leading to EPs determine the trigger signals of x. Thus, reducing the number of EPs also contributes to reduce the causality relations with the remaining signals of the circuit. We have observed that penalizing the amount of EPs helps to find solutions with simpler logic.

For each state s, we define the variable $ep(s)$ that determines whether s is an EP for x:

$$\forall s_i \to s_j : \left(\neg ER^+(s_i) \wedge ER^+(s_j)\right) \vee \left(\neg ER^-(s_i) \wedge ER^-(s_j)\right) \implies ep(s_j)$$

The number of entry points can now be computed by:

$$\text{numEP} = \sum_{s \in S} ep(s)$$

4.6.5 Cost function

The multiobjective cost function used to estimate the quality of a solution is defined as:

$$\text{Cost} = \alpha \cdot \text{Conf} + \beta \cdot \text{EssLit} + \gamma \cdot \text{numEP} + \delta \cdot \text{ERsize} \tag{4.1}$$

with $\alpha, \beta, \gamma, \delta$ being adjustable coefficients.

This function needs to be encoded as a SAT formula. The larger the coefficients, the more complex the formula. This affects the runtime dramatically and limits the range of values that can be used in practice.

We found that weights ≤ 3 produce good results with reasonable execution times.

Having a diversity of cost functions with different coefficients also contributes to a wider exploration of solutions. In our experiments we have also generated results by exercising a small set of cost functions and selecting the best solution. This strategy will be further discussed in Section 4.9.

4.7 SAT-based optimization algorithm

The optimization algorithm iteratively tries to insert new signals (one at a time) into the ALTS until CSC is solved or no satisfiable solution is found. The core of the algorithm is the function *findModelForOneSignal*, which returns a model that encodes the definition of the $ER^{\pm}/QR^{\pm}$ regions for the insertion of a new signal.

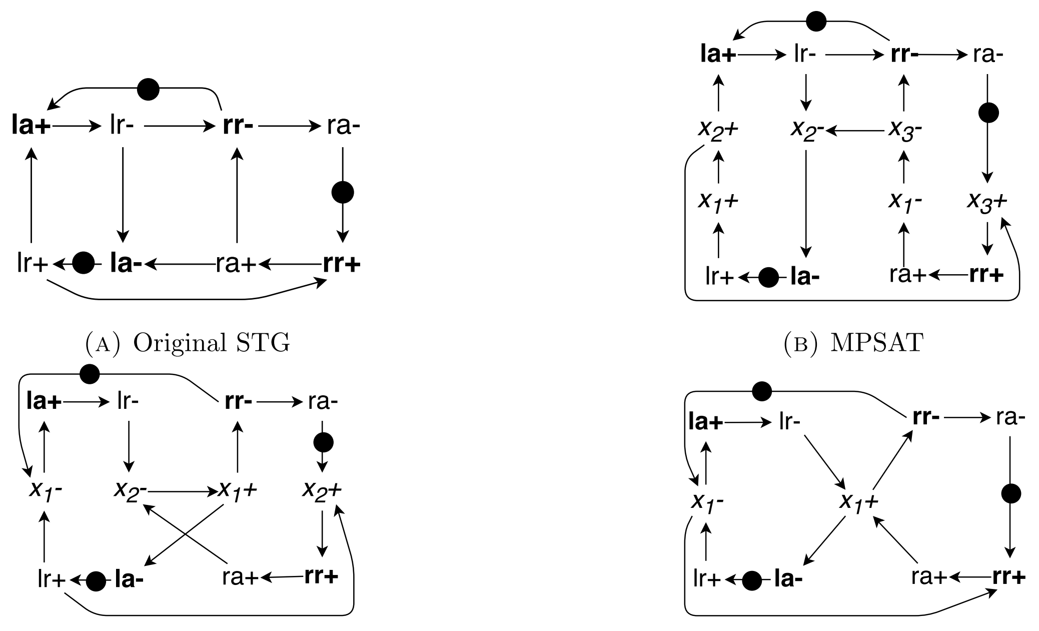
Algorithm 4 sketches the procedure to find a solution for signal insertion using pseudo-Boolean optimization. The cost function (4.1) is encoded as a set of SAT clauses [49]. The function is minimized by iteratively constraining the formula until it becomes unsatisfiable. If a model with $Cost = k$ is found in one iteration, the constraint $Cost < k$ is encoded and added for the next iteration. This strategy speeds-up the optimization by taking advantage of the clauses learned by the SAT solver from the previous iterations [49].

A binary search on the value of k could also be possible, but it cannot take advantage of the learned clauses. We have not observed a clear benefit when using binary search.

Algorithm 4: FINDMODELFORONESIGNAL(G)

```
input  : An SG with CSC conflicts.
output: A SAT model for signal insertion.
begin
    $CNF$ = encodeCSCconstraints($G$)
    $model$ = SATsolver($CNF$)
    $bestModel = model$
    while $isSatisfiable(model)$ do
        $k$ = getCost($model$)
        addClausesForCost($CNF$, $Cost < k$)
        $model$ = SATsolver($CNF$)
        if $isSatisfiable(model)$ then $bestModel = model$
    return $bestModel$
```

The PBLib [50] toolkit was used for the encoding of Pseudo-Boolean constraints and solving the SAT formulas. Internally, PBLib uses Minisat [51] as SAT solver.



(A) Original STG

(B) MPSAT

(C) Petrify

(D) PBASE

FIGURE 4.16: 4-phase latch controller L220oR2242 (from [53]). State encoding solutions obtained by different tools.

4.8 Comparison with previous art

We next discuss the main differences with the most relevant approaches proposed for asynchronous controllers working in input/output mode. We can distinguish two main categories:

- Structural methods working at Petri net level, such as MPSAT [42] (based on unfoldings) and structural methods using integer-linear programming [52].
- State-based methods, such as petrify [40] and a previous SAT-based approach [41].

We will use the example of Figure 4.16, depicting one of the 4-phase latch controllers presented in [53], to discuss the differences among tools. This figure includes the approach presented here, that will from now on be referred to as PBASE. The logic equations for each solution are the following:

	MPSAT	Petrify	PBASE
$la =$	$x_2(\overline{rr}+\bar{x}_1)+x_3$	$\bar{x}_1$	$\bar{x}_1$
$rr =$	$x_2\ lr\ \bar{x}_1+x_3$	$x_2+rr\ \bar{x}_1$	$(\bar{x}_1\ \overline{ra})+rr\ \bar{x}_1$
$x_1 =$	$\bar{x}_2\ lr+x_1(\overline{ra}+\bar{x}_3)$	$(\bar{x}_2\ rr)+x_1+\overline{lr}$	$(ra\ rr\ \overline{lr})+x_1(rr\ \overline{lr})$
$x_2 =$	$(x_3+x_1)+x_2\ lr$	$\overline{ra}(\bar{x}_1+lr)+x_2\ lr$	
$x_3 =$	$(x_2x_1\overline{ra})+x_3\bar{x}_1$		

Regarding the exploration of insertion points for the new signals, the main limitation of the structural methods is that the original specification acts as a *corset*. The new events must be *anchored* in existing nodes of the Petri net (or its unfolding). If two different Petri nets have the same reachability graph, the space of solutions is also different and a subset of the solutions available at ALTS level. Moreover, the insertion must be done in such a way that the causality relations can be expressed with the semantics of a Petri net. In Figure 4.16, the MPSAT solution requires three new signals and 22 literals. The reader can intuitively perceive that the new events have simple causality relations. This phenomenon also occurs for the ILP-based method proposed in [52].

Petrify is a special case. The insertion of signals is done at state level, however the sets of states for insertion are built based on combinations of regions (that correspond to Petri net places). Petrify only uses simple combinations of regions that prevent the exploration of intricate solutions that could potentially be better. It requires two signals and 13 literals.

Pbase provides the most efficient solution, with only one signal and 11 literals. Notice that the two new events have multiple causality relations (two input and two output arcs). Although the figure shows a Petri net, these relations are naturally found at state level ignoring the model of the original specification. In this particular case, the solution was representable as a nice Petri net.

With regard to the estimation of logic, structural methods are mostly based on finding trigger relations between events. This gives a lower bound on the number of literals, although it is less accurate than the estimation given by essential literals.

The SAT-based approach presented in [41] has two main limitations. First, all new signals are inserted simultaneously and cannot have mutual trigger relations between them. Second, the approach is simply based on finding valid solutions without any estimation of the logic cost. The solutions provided by this approach are significantly worse than the ones generated by the other tools discussed in this section.

4.9 Experimental results

This section shows the experimantal results for Pbase and a comparison with Petrify and MPSAT. Additionally, we have re-implemented the approach from [41] (referred to as SAT), and included it as a baseline.

We have used a large diversity of benchmarks from the literature and all the 4-phase latch controllers presented in [53] (127 out of 137 had CSC conflicts). The solutions for all benchmarks can be found in [54].

Table 4.1 shows the results for a variety of heterogeneous controllers. The column *Signals/Literals* reports the number of state signals that were inserted and the number of literals of the Boolean equations (in factored form) after logic synthesis. *CPU(sec)* reports the CPU time required to solve CSC. The number of states of the SG is in column

$|S|$. The *I/O* column contains the number of input/output signals of the SG. This table compares results between Petrify, MPSAT and two versions of PBASE, *single* and *multi*, using different versions of the $(\alpha, \beta, \gamma, \delta)$ coefficients for optimization function (4.1):

- PBASE(single): using the coefficients (2,1,3,2).
- PBASE(multi): using multiple different values for the coefficients and choosing the best solution. The set of coefficients were (0,1,1,1), (3,2,2,0), (3,1,1,0) and (1,0,1,1), besides the one used for PBASE(single).

PBASE(multi) explores a larger variety of solutions at the expense of computational time. It also uses a fast heuristic in the first iteration to be able to solve larger problems. A 10-minute timeout is set up and the best solution found when the timeout expires is returned. The combination of the fast heuristic with the timeout allows to solve problems that could not be solved with the simpler version.

In some cases, the tools were not able to complete the task. These cases are reported with one of the following codes:

- Unsf: Unsafe Petri nets. MPSAT is unable to solve them.
- Fail: The tool was unable to find a solution.
- Time: No solution found in less than 1 hour.

A summary of the results for Table 4.1 can be found in Table 4.2, including the results for SAT [41]. This table presents a comparison between PBASE(multi) and the other tools. Row *Solved* reports the number of solved instances. The remaining data in the table only report the total results for the benchmarks that were solved by both tools under comparison. Results for those not solved by both were ignored in the summary. The CPU time is divided into 3 groups as a function of problem size (see Table 4.3 for the group division). This puts into scale the amount of time used for the largest problems. The final row reports the ratio of literals obtained by any pair of tools taking the other tools as a reference.

SAT gives the lowest-quality solutions, as it does not include any quality estimator in the model, while PBASE outperforms the other methods, with an average improvement of 13% in the number of literals with regard to petrify. PBASE(multi) offers a tangible improvements with regard to PBASE(single). However, this comes at the expense of a higher computational cost. Section 4.9.1 discusses this problem.

Interestingly, one of the tiniest and most difficult problems for state encoding (`buf_unsafe.1`), was only solved by PBASE. It required 5 state signals and the SG was expanded from 12 states to 69 after signal insertion.

Table 4.4 reports the summary of results for the 127 4-phase latch controllers [53] without CSC. Even though all benchmarks were small, only Petrify and PBASE could

TABLE 4.1: Experimental results for Petrify (Pfy), MPSAT (MP), PBASE(single) (PB(s)) and PBASE(multi) (PB(m)).

			CPU(sec)				Signals/Literals			
Example	I/O	$\|S\|$	Pfy	MP	PB(s)	PB(m)	Pfy	MP	PB(s)	PB(m)
adc.buff1	0/2	6	0.4	0.9	**0.3**	5.4	**2/9**	**2/9**	**2**/11	**2/9**
adfast	3/3	44	1.2	**0.1**	5.6	16.0	**2/14**	**2**/21	**2/14**	**2/14**
alloc-outbound	4/3	17	0.2	**0.1**	0.8	7.3	**2/16**	**2**/17	**2/16**	**2/16**
buf2	0/2	8	**0.1**	Unsf	0.8	7.9	**3**/14	-/-	**3**/15	**3/13**
buf_dum.1	0/2	8	**0.1**	0.1	0.8	6.8	**3**/14	**3**/15	**3**/15	**3/13**
buf_unsafe.1	0/2	12	Fail	Unsf	**5.3**	23.3	-/-	-/-	**5/26**	**5/26**
c10	0/10	2046	Time	Fail	**32.7**	136.7	-/-	-/-	**1/31**	**1/31**
c6	0/6	126	4.2	**0.8**	1.1	6.9	**1/19**	**1/19**	**1/19**	**1/19**
csc-div1	0/2	8	**0.0**	0.1	0.1	4.7	**1/16**	**1/16**	**1/16**	**1/16**
duplicator	2/2	20	0.4	**0.1**	0.5	5.9	**2**/18	**2/13**	**2/13**	**2/13**
future	4/4	36	1.0	**0.2**	0.4	6.0	**1/18**	3/33	**1/18**	**1/18**
glc	2/1	17	0.1	**0.1**	0.1	4.9	**1/10**	**1**/11	**1/10**	**1/10**
ircv-bm	5/4	44	5.8	**0.4**	9.8	39.9	**2**/37	**2**/31	**2**/35	**2/28**
isend	4/3	36	4.2	**0.4**	4.4	23.8	3/48	3/34	3/**29**	**2/29**
lazy_ring.noncsc	5/3	160	1.7	**0.4**	27.6	53.9	**1**/24	2/29	**1**/22	**1/20**
master-read	6/7	8932	**54.6**	Fail	Time	Time	**8/68**	-/-	-/-	-/-
master-read2	0/13	8932	26.3	**15.5**	Time	Time	6/**70**	**5**/75	-/-	-/-
master-read.1098	6/7	1098	9.7	**3.3**	Time	537.3	**4**/57	6/43	-/-	5/**41**
mmu0	4/4	174	2.7	**0.1**	89.1	198.3	**3**/29	**3**/28	**3**/28	**3/26**
mmu1	4/4	82	1.1	**0.2**	8.1	27.1	**2**/32	**2**/25	**2**/25	**2/23**
mod4_counter	1/2	16	**0.1**	**0.1**	0.3	8.0	**2**/26	**2/25**	**2**/26	**2**/26
mr0	5/6	302	4.4	**0.4**	Time	600.2	**3**/45	4/**29**	-/-	4/33
mr1	4/5	190	3.4	**0.6**	91.6	201.7	4/35	4/31	**3**/26	**3/25**
nak-pa	4/5	56	0.7	**0.1**	0.7	9.0	**1**/18	**1**/18	**1**/18	**1/16**
nowick	3/2	18	0.2	**0.1**	0.2	5.4	**1/13**	**1/13**	**1/13**	**1/13**
par2	3/3	28	0.2	**0.1**	4.4	12.9	**2/16**	**2/16**	**2/16**	**2/16**
par4	5/5	628	3.9	**0.2**	Time	544.6	**4/32**	**4/32**	-/-	**4/32**
pla	0/3	12	**0.1**	0.1	0.2	4.5	**1/14**	2/16	**1/14**	**1/14**
ram-read-sbuf	5/5	36	1.8	**0.1**	1.2	10.3	**1/18**	**1**/19	**1**/22	**1/18**
read_write	7/4	322	2.0	**0.2**	79.1	164.8	**1/24**	**1**/26	**1/24**	**1/24**
sbuf-ram-write	5/5	58	5.0	**0.2**	9.1	24.1	**2**/22	**2**/31	**2**/23	**2/21**
sbuf-read-ctl	2/4	14	**0.1**	0.2	0.2	4.5	**1/15**	**1/15**	**1/15**	**1/15**
seq2	3/3	12	0.1	0.1	**0.1**	3.0	**1/8**	**1/8**	**1/8**	**1/8**
seq3	4/4	16	0.5	**0.1**	0.6	6.9	**2/14**	**2/14**	**2/14**	**2/14**
seq4	5/5	20	1.3	**0.2**	1.4	8.3	3/20	3/20	**2/19**	**2/19**
seq8	9/9	36	4.7	**1.1**	108.7	302.6	4/47	7/44	**3**/44	**3/37**
seq-mix	4/4	20	1.1	**0.2**	2.2	10.8	3/20	3/20	3/**18**	**2/18**
sis-master-read	6/7	1882	3.3	**0.3**	Time	309.3	**1/38**	**1**/40	-/-	**1**/39
trcv-bm	5/4	44	8.7	**0.3**	7.6	35.3	**2**/37	**2**/32	**2/31**	**2/31**
tsend-bm	5/4	40	4.6	**0.2**	7.9	21.0	2/39	2/**27**	3/34	**1**/28
vbe4a.nousc	3/3	58	1.4	**0.2**	5.1	21.1	**3**/26	4/23	**3**/18	**3/16**
vbe5a	3/3	44	0.9	**0.1**	4.2	15.0	**2/14**	**2**/21	**2/14**	**2/14**
vbe6a.nousc	4/4	128	1.1	**0.2**	41.0	114.4	3/31	**2/30**	**2/30**	**2/30**
vbe6x.nousc	3/3	48	0.4	**0.2**	4.4	17.3	**2/22**	**2/22**	**2**/23	**2/22**
vme_read	8/6	251	4.0	**0.1**	16.2	39.0	**1**/32	**1**/33	**1/30**	**1/30**
vme_read_write	3/3	28	0.3	**0.3**	1.0	8.6	**1**/23	2/27	**1/22**	**1/22**
vme_write	8/6	817	7.8	**0.2**	Time	602.1	**1**/38	**1**/38	-/-	**1/35**
vmebus	3/3	24	0.8	**0.2**	0.5	7.2	**1/19**	2/28	**1/19**	**1/19**

Table 4.2: Summary for the benchmarks in Table 4.1.

	Pfy	PB(m)	MP	PB(m)	SAT	PB(m)	PB(s)	PB(m)
Solved	46	46	44	46	43	46	41	46
CPU (small)	13	163	4	155	0	148	26	186
CPU (medium)	30	226	2	226	0	226	62	226
CPU (large)	53	3675	8	3675	73	3812	487	1218
Signals	88	83	97	80	72	78	78	74
Literals	1081	943	1042	930	1938	948	864	820
Ratio	**1.00**	**0.87**	**1.00**	**0.89**	**1.00**	**0.49**	**1.00**	**0.95**

Table 4.3: Average CPU time for different SG sizes.

			Avg. CPU (s)	
Size	**Condition**	n	**PB(s)**	**PB(m)**
small	$\|S\| < 40 \ \wedge \ \|\Sigma\| \leq 15$	22	1.2	8.5
medium	$40 \leq \|S\| < 100 \ \wedge \ \|\Sigma\| \leq 15$	10	6.2	22.6
large	$\|S\| \geq 100 \ \vee \ \|\Sigma\| > 15$	9(s)/14(m)	54.1	272.3

Table 4.4: Summary for the 127 4-phase latch controllers.

	Pfy	PB(m)	MP	PB(m)	SAT	PB(m)	PB(s)	PB(m)
Solved	127	127	72	127	85	127	127	127
CPU (sec)	41	928	11	347	1	430	118	928
Signals	231	207	111	84	110	110	207	207
Literals	1818	1550	952	778	1386	943	1593	1550
Ratio	**1.00**	**0.85**	**1.00**	**0.82**	**1.00**	**0.68**	**1.00**	**0.97**

solve all of them. Since many of them were specified as unsafe Petri nets, MPSAT could not handle them. SAT also failed in many examples due to the impossibility of inserting state signals with causality relations among them. This feature was essential for the other methods to solve some of the examples. The results for both tables show an average reduction of 14% in literals when compared to petrify.

4.9.1 Scalability

A major concern is scalability with the size of the SG. The main reason for the increase of the CPU time is the size of the SAT formula, which is mainly dominated by the clauses representing the cost function (Pseudo-Boolean constraints).

Table 4.3 reports the average execution times for the benchmarks classified in three categories according to the number of states ($|S|$) and signals ($|\Sigma|$) of the ALTS (n reports the number of instances in each class). While the runtime is low for small examples, it drastically increases for large ALTS.

Chapter 5 focuses on this particular problem by introducing a technique to reduce the size of the ALTS.

4.10 Conclusions

This chapter has introduced a novel approach for state encoding based on Pseudo-Boolean optimization. The approach allows to encode any valid solution as well as estimators of logic complexity. The results show a significant reduction in the number of literals with respect to the existing tools.

Scalability for large controllers poses a challenge that is addressed in the following chapter.

The exploration of solutions trading-off performance and complexity is an aspect that remains to be addressed to reduce the input/output response time of the controllers.

Chapter 5

State encoding for large asynchronous controllers

This chapter is devoted to address the challenges presented by large asynchronous controllers for state encoding techniques. Typically, encoding of large controllers requires turning to structural methods, which handle concurrency more efficiently than state-based approaches. In contrast, the method proposed here works exclusively at the state level. It enables state-based techniques, like the one presented in Chapter 4, to effectively deal with large spaces of states. This allows keeping most of the advantages of these methods, such as higher quality of solutions, while still performing in reasonable execution times, even in the presence of high concurrency.

5.1 Introduction

The existing methods to solve the state encoding problem can be divided into two categories: structural and state-based.

Structural methods have been proposed for STGs and exploit the properties of the underlying Petri nets to avoid an explicit enumeration of the state space [42, 52]. In state-base methods, the state space is enumerated explicitly by representing all possible interleavings of concurrent events [8, 40]. State-based methods enable a more accurate exploration of the space of solutions and can potentially lead to better circuits. However, they may suffer from the state explosion problem when the specification is highly concurrent. All of this is evidenced by results reported in Chapter 4: Pbase could yield the best results at the cost of the highest execution time.

The structural methods work directly on the graph representation of the specification (e.g., a Petri net) or some unfolded version. They have limitations about the type of acceptable representations, e.g., safe or free-choice Petri nets, and the locations where the new signals can be inserted. Even for structural methods, some controllers may be too large. For this reason, some techniques have been proposed to decompose a large controller into smaller ones that can be synthesized separately [55].

One of the main problems for decomposition techniques is the appearance of *irreducible conflicts* that cannot be solved while preserving implementability. Solutions for that problem are suggested in [55] by introducing a structure called *gyroscope* that inserts new signals with a high degree of concurrency. However, this structure aims at solving conflicts without paying attention at the cost of implementing the circuit, e.g., the complexity of the Boolean equations.

The work presented in this Chapter is encouraged by the following facts, observed through years of experience:

- State-based methods can be superior to structural methods for the state encoding problem. The main reason is that the exploration space for signal insertion is larger and better estimators for good-quality solutions can be used (see Chapter 4).
- Most of the controllers are designed manually by humans and the largest specifications usually have no more than 10^7 states.
- The best-quality state-based methods for encoding can manage up to 10^3 states with an affordable runtime.

Therefore, there is a gap of roughly 4 orders of magnitude between what is computationally affordable for state encoding and the size of large controllers.

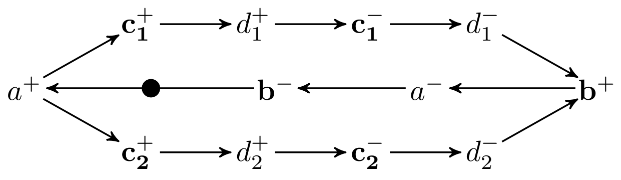
In previous work, the decomposition into smaller sub-controllers has been proposed [55]. Besides the requirement to insert the *gyroscope* structures to avoid irreducible conflicts, the resolution of conflicts at each sub-controller is agnostic on the behavior of the other sub-controllers. This may have a negative effect in the quality of the solutions.

In this chapter we propose a new approach that explicitly keeps track of the complete state space. The approach iteratively projects the behavior of the controller into subsets of relevant signals and partially solves the encoding problem on the projections. The new signals are incorporated into the original specification and the process is re-executed until all encoding conflicts have been solved.

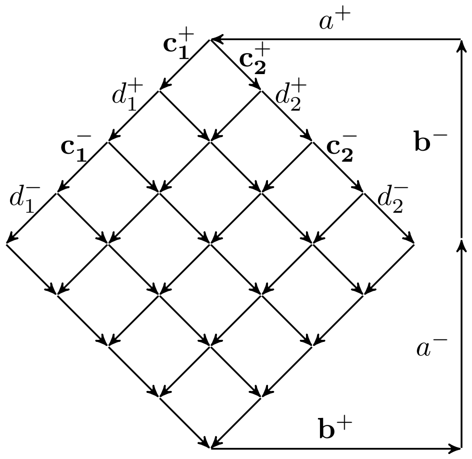
Unlike other decomposition techniques, irreducible conflicts do not pose any hurdle for the proposed method. While a projection might cause these kind of conflicts, the iterative nature of the projection and re-composition allows for these conflicts to be solved in subsequent iterations.

An important aspect of the method is that the projections can be calculated efficiently. Algorithms with complexity $\mathrm{O}(m \log n)$[1] to minimize labelled transitions systems up to some criterion of behavioral equivalence (branching bisimilarity) can be used [56].

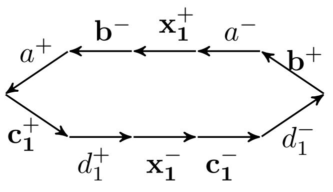
[1] n and m are the number of states and transitions, respectively.



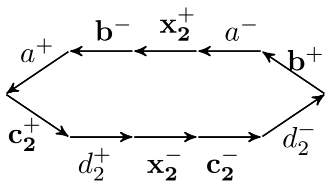
(A) STG of a parallelizer.



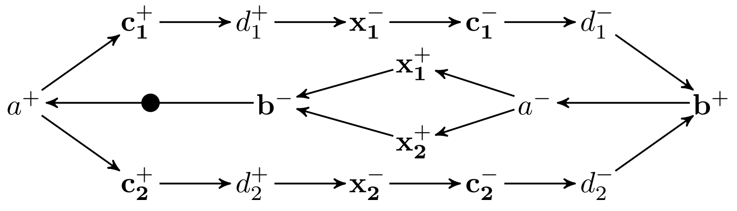
(B) LTS of the parallelizer.



(C) Projection onto channel 1 and insertion of signal x_1.



(D) Projection onto channel 2 and insertion of signal x_2.



(E) STG after state encoding.

FIGURE 5.1: State encoding for a parallelizer.

Thus, large controllers can still be manipulated and the state encoding problem solved in small controllers using SAT-based methods like the one introduced in Chapter 4.

The re-composition of the system with the new inserted signals can be done via synchronous products, which can have a quadratic runtime in the worst-case, but typically run in linear time due to the high similarity of the two components.

5.2 Overview

This section sketches the main features of the method proposed in this chapter. The example shown in Figure 5.1 will be used to illustrate the method. Figure 5.1a shows an STG specifying the behavior of a parallelizer, which is a controller used in handshake circuits to fork the execution of two asynchronous processes.

Signals a (input) and b (output) are the handshake signals of the channel that triggers the activity of the parallel processes represented by the handshake signals c_i (output) and d_i (input).

The controller can be represented by a Petri net in a very succinct way. Yet, due to the high level of concurrency, it suffers from the state explosion problem. This means that the number of valid markings, corresponding to states in a labeled transition system (LTS), grows exponentially with the number of channels. Figure 5.1b shows the LTS representation of the same parallelizer. To quantify the state explosion, the following table shows the number of states needed to represent a parallelizer with n processes:

Processes	2	3	4	5	6	7	8	n
Signals	6	8	10	12	14	16	18	$2n+2$
States	28	128	628	3K	16K	78K	391K	5^n+3

Let us now consider one of the channels, which follows the sequence $\langle c_i^+, d_i^+, c_i^-, d_i^- \rangle$. The reader will notice that the states before c_i^+ and after d_i^- have the same encoding. This conflict occurs at every channel. In this particular example, it is sufficient to focus on each channel individually to solve the corresponding encoding conflict. Thus, a channel can be freed from conflicts if a signal is inserted between d_i^+ and c_i^-. This example suggests that not all the information is relevant to find a solution for certain encoding conflicts.

In order to exploit this feature, we propose the following method:

1. Find a group of signals to be hidden and project the behavior onto the remaining signals. For the example, a good strategy is to hide every signal except a, b, and one of the channels (c_i and d_i). Initially, the signals c_2 and d_2 are hidden while c_1 and d_1 are maintained.

2. Insert new signals to solve the encoding conflicts of the simplified controller. Figure 5.1c shows the projected LTS after the insertion of signal x_1.

3. Recompose the full controller by doing a synchronous product between the original controller and the simplified one with the new inserted signals.

4. If not all conflicts have been solved, go to step 1 and repeat the process using the full controller with the new inserted signals.

In this example, the second iteration would generate the projection shown in Figure 5.1d, after hiding c_1, d_1 and x_1. After recomposing the original LTS, the behavior shown in the STG of Figure 5.1e would be obtained.

By *hiding* a well selected set of signals, an asynchronous controller can be simplified enough so that it is possible to use state encoding techniques that can handle the full state space.

In general, asynchronous controllers do not show behaviors as simple as the one of the parallelizer and the automation of the process requires smart strategies to calculate projections. This chapter presents a method to simplify arbitrary asynchronous controllers and obtain small projections that can be manageable by encoding tools working at state level.

5.3 ALTS transformations

This section describes a collection of transformations over WF-ALTSs. The purpose of these transformations is to provide an infrastructure to insert/hide signals and recompose the original specification with new signals that solve the CSC conflicts. All these transformations must satisfy two properties:

- The behavior of the system must be preserved (branching bisimilarity).
- The implementatibility conditions must hold.

The most important result of this section indicates that the projections of the specification should never hide signals in conflict. This strategy allows to work with τ-free WF-ALTSs when inserting new signals to solve CSC conflicts.

5.3.1 Signal Insertion

The insertion of new signals is described in detail in Chapter 4.4.7. For the purposes of this chapter, it is important to remember that signal insertion preserves branching bisimilarity.

5.3.2 Hiding signals

Given an LTS $A = (S, \Sigma, T, s_0)$, a set of signals σ can be *silenced*, denoted *silence*(A, σ), if every event of every signal $a \in \sigma$ is substituted by τ. Figure 5.2b shows an example of the silence operation on Figure 5.2a for signals a and b.

We are now interested in removing the new τ transitions that appear after a *silence* operation. This will yield a smaller ALTS. One of the ways of achieving this is by using τ-priorization and τ-compression operations described in [57].

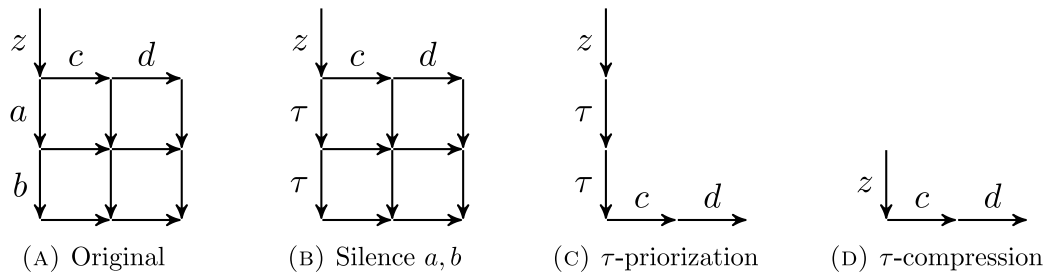


FIGURE 5.2: Process to hide signals a and b.

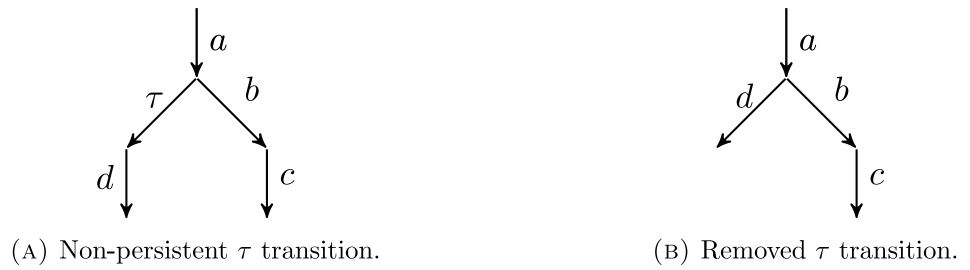


FIGURE 5.3: Branching bisimilarity is not preserved when removing a non-persistent τ transition.

The τ-priorization operation consists on the following: if there is a transition $s \xrightarrow{\tau} s'$, then any other transition $s \xrightarrow{a} s''$ is removed.

Intuitively, this operation assumes *zero-delay* τ-transitions and non-zero delay for the other transitions. This makes the model *prioritize* τ's over other transitions. Figure 5.2c shows the τ-priorization for the ALTS 5.2b.

The τ-compression is an operation aimed at removing sequences of τ transitions. If a state s has only one transition, and this transition is $s \xrightarrow{\tau} s'$, then s is *merged* with s'. An example for this operation can be seen in Figure 5.2d.

Important results about these operations can be found in [57]. In particular, these operations preserve *branch bisimilarity.* If the τ transitions are persistent, then applying both operations yields a τ-free ALTS.

Finally, given an ALTS $A_1 = (S_1, \Sigma_1, T_1, s_0^1)$, a set of signals σ is said to be *hidden* in $A_2 = (S_2, \Sigma_2, T_2, s_0^2)$, denoted $A_2 = \mathit{hide}(A_1, \sigma)$, if A_2 is the τ-compression of the τ-priorization of $\mathit{silence}(A_1, \sigma)$.

We can now define the concept of *branch bisimilarity with respect to a set of signals.* Let $A_1 = (S_1, \Sigma_1, T_1, s_0^1)$, $A_2 = (S_2, \Sigma_2, T_2, s_0^2)$ be two ALTS, then A_1 is branching bisimilar with respect to σ, denoted $A_1 \approx_\sigma A_2$, iff $\mathit{silence}(A_1, \sigma) \approx \mathit{silence}(A_2, \sigma)$.

With these definitions and results we can now state that, given the τ-free WF-ALTS A_1, and $A_2 = \mathit{hide}(A_1, \sigma)$, then $A_1 \approx_\sigma A_2$. Furthermore, if σ is conflict-free, then A_2 is τ-free.

On the other hand, hiding a signal in conflict does not yield a τ-free ALTS. Figure 5.3a shows an ALTS with a non-persistent τ transition. In this case, τ-priorization cannot be applied without breaking branching bisimilarity. The τ transition can still be removed, as shown in Figure 5.3b, but this ALTS only preserves *trace equivalence*, which is a weaker equivalence class.

5.3.3 Synchronous Product

The synchronous product of two LTSs can be defined as follows. Let $A_1 = (S_1, \Sigma_1, T_1, s_0^1)$, $A_2 = (S_2, \Sigma_2, T_2, s_0^2)$ be two ALTS. The synchronous product of A_1 and A_2, denoted by $A_1 \times A_2$ is another LTS (S, Σ, T, s_0) defined by:

- $s_0 = \langle s_0^1, s_0^2 \rangle \in S$
- $\Sigma = \Sigma_1 \cup \Sigma_2$
- $S \subseteq S_1 \times S_2$ is the set of states reachable from s_0 according to the following definition of T':
- Let $\langle s_1, s_2 \rangle \in S$:
  - If $a \in \Sigma_1 \cap \Sigma_2$, $s_1 \xrightarrow{a} s_1'$ in T_1 and $s_2 \xrightarrow{a} s_2'$ in T_2, then $\langle s_1, s_2 \rangle \xrightarrow{a} \langle s_1', s_2' \rangle$ in T'
  - If $a \in \Sigma_1 \setminus \Sigma_2$ and $s_1 \xrightarrow{a} s_1'$ in T_1, then $\langle s_1, s_2 \rangle \xrightarrow{a} \langle s_1', s_2 \rangle$ in T'
  - If $a \in \Sigma_2 \setminus \Sigma_1$ and $s_2 \xrightarrow{a} s_2'$ in T_2, then $\langle s_1, s_2 \rangle \xrightarrow{a} \langle s_1, s_2' \rangle$ in T'
  - No other transitions belong to T'
- $T \subseteq T'$ is the set of transitions between states in S that belong to T'.

It is necessary to note that the synchronous product preserves branching bisimilarity in some way. In particular, the synchronous product of two ALTS that are branching bisimilar with respect to their common signals will also be branching bisimilar to the original ALTSs with respect to their common signals. This becomes important later to ensure that the approach presented in this chapter preserves branching bisimilarity at all steps. This result is formally stated by the following theorem:

Theorem 5.1. *Let $A_1 = (S_1, \Sigma_1, T_1, s_0^1)$, $A_2 = (S_2, \Sigma_2, T_2, s_0^2)$ be two τ-free WF-ALTS, with $\sigma_1 = \Sigma_1 \setminus \Sigma_2$ and $\sigma_2 = \Sigma_2 \setminus \Sigma_1$. Let $A_3 = A_1 \times A_2$. If $A_1 \approx_{\sigma_1 \cup \sigma_2} A_2$ then $A_3 \approx_{\sigma_2} A_1$ and $A_3 \approx_{\sigma_1} A_2$.*

Proof.
We will denote the set of states of A_i as S_i and we will use $s_i, s_i', s_i'', \ldots$, to denote different states in S_i. By construction of $A_3 = A_1 \times A_2$, every state in S_3 is a pair $s_3 = \langle s_1, s_2 \rangle$ with $s_1 \in S_1$ and $s_2 \in S_2$.

Let us first prove that $A_3 \approx_{\sigma_2} A_1$. Consider $A_4 = \textit{silence}(A_3, \sigma_2)$. Then, it suffices to show that $A_1 \approx A_4$.

Since A_4 has the same states as A_3, we can also represent every state in S_4 as a pair $s_4 = \langle s_1, s_2\rangle$. Let us define a binary relation R between S_1 and S_4 as follows: for every state $s_4 = \langle s_1, s_2\rangle$, $s_1 R s_4$. It is trivial to see that this relation exists for every $s_4 \in S_4$. Similarly, since $A_1 \approx_{\sigma_1 \cup \sigma_2} A_2$, it follows that by construction of the synchronous product the relation R also exists for every $s_1 \in S_1$. We only need to prove that R is a branching bisimulation, i.e.,

1) Whenever $s_1 R s_4$ and $s_1 \xrightarrow{a} s_1'$, then either $a = \tau$ and $s_1' R s_4$, or there exists a path $s_4 \overset{\tau^*}{\Longrightarrow} s_4'' \xrightarrow{a} s_4'$ such that $s_1 R s_4''$ and $s_1' R s_4'$.

2) Whenever $s_1 R s_4$ and $s_4 \xrightarrow{a} s_4'$, then either $a = \tau$ and $s_1 R s_4'$, or there exists a path $s_1 \overset{\tau^*}{\Longrightarrow} s_1'' \xrightarrow{a} s_1'$ such that $s_1'' R s_4$ and $s_1' R s_4'$.

1) Since A_1 is τ-free, we know that $a \neq \tau$. If $s_1 \xrightarrow{a} s_1'$ and $s_4 = \langle s_1, s_2\rangle$ then the product also creates a state $s_4' = \langle s_1', s_2'\rangle$ and a path $s_4 \xrightarrow{a} s_4'$. In case $a \notin \Sigma_2$, then $s_2 = s_2'$. Therefore, τ^* is empty and $s_4 = s_4''$. By the definition of R, we have that $s_1 R s_4''$ and $s_1' R s_4'$.

2) Let us assume $s_4 = \langle s_1, s_2\rangle$. We need to consider two cases: $a = \tau$ and $a \neq \tau$. If $a = \tau$ then τ is hiding a signal in $\Sigma_2 \setminus \Sigma_1$. Therefore, the product generates the state $s_4' = \langle s_1, s_2'\rangle$, since A_1 does not move and, thus, $s_1 R s_4'$. If $a \neq \tau$ then A_1 and A_2 synchronize with a and the product generates the state $s_4' = \langle s_1', s_2'\rangle$. Therefore, the path $s_1 \xrightarrow{a} s_1'$ exists in A_1 and $s_1' R s_4'$. Notice also that $s_1'' = s_1$ and, thus, $s_1'' R s_4$.

By symmetry, $A_3 \approx_{\sigma_1} A_2$ can be proved identically. $\square$

5.4 CSC resolution algorithm

Intuitively, solving CSC conflicts is an iterative process with the following steps:

- Hide a subset of signals to reduce the size of the ALTS.
- Insert a new signal to solve some of the CSC conflicts of the remaining signals.
- Re-compose the ALTS by recovering the previously hidden signals.
- Reduce concurrency of the newly inserted signal.

This process is repeated until all CSC conflicts have been solved. A high-level description of the algorithm is shown in Algorithm 5.

The projection step (line 1) is necessary to reduce the ALTS to a size that is manageable by CSC solving algorithms. Preserving the relevant signals of the CSC conflicts is essential to derive good-quality solutions. The insertion of a new signal x_i (line 2) will solve only conflicts of the remaining signals. The details of the projection step are discussed in Section 5.4.1.

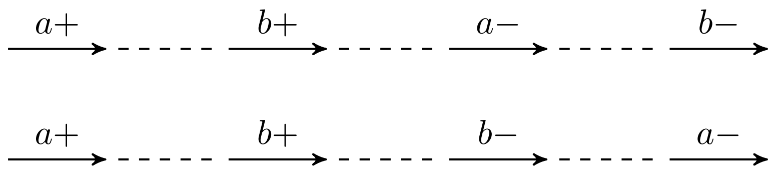


FIGURE 5.4: Top: lock relation between a and b (top). Bottom: a and b are not in lock relation.

The hidden signals are recovered by re-composing the original ALTS with the new inserted signal. This is achieved by computing a synchronous product (line 3) between the new ALTS (with the new inserted signal) and the original one.

Re-composition implicitly creates a high-degree of concurrency of the new inserted signal with the signals that were hidden by the projection. In particular, the CSC conflicts for the hidden signals are not solved by the new signal. To mitigate this effect, the concurrency of the new signal is reduced (line 4). As a side-effect, new CSC conflicts are solved and the size of the ALTS is also reduced.

An accurate description of concurrency reduction is out of the scope of this thesis. An in-depth discussion can be found in [58].

Algorithm 5: SOLVECSC(A)

```
   input  : An ALTS with CSC conflicts.
   output: An ALTS without CSC conflicts.
   begin
       while $A$ has CSC conflicts do
1          $B$ = Project($A$)      /* Hiding signals      */
2          $x_i$ =insertSignal($B$)  /* Solving CSC         */
3          $A = A \times B$          /* Re-composition      */
4          reduceConcurrency($A$, $x_i$)
       return $A$
```

The following subsections describe more details about projection and re-composition. Afterwards a discussion about some of the properties of this algorithm is presented.

5.4.1 Projection

The main objective of the projection step is to reduce the size of the ALTS by means of hiding signals. The only constraint is that none of the hidden signals can be *in conflict*. The reason is that τ events become *inert* if they are not in conflict [57] and, thus, they can be completely removed during state minimization. That means that any ALTS can become τ-free if no signals in conflict are hidden.

The set of signals to hide has an impact in the quality of the solution. Next, a set of concepts useful to define the criteria to select the signals are discussed.

$$\begin{tikzcd}
\bullet \arrow[r, "a+"] & {} \arrow[r, "b+"] & {} \arrow[r, "c+"] & {} \arrow[r, "a-"] & {} \arrow[r, "b-"] & {} \arrow[r, "c-"] & \bullet
\end{tikzcd}$$

FIGURE 5.5: Hiding signals a, b and c causes the CSC conflict represented by the dots to collapse.

- **Concurrency:** Hiding signals with high concurrency has a bigger impact on the size of the ALTS. It is thus convenient to hide signals with a large ER.
- **Lock relation:** Two signals a and b are in lock relation when, for every possible trace in the ALTS, there is a transition b between two transitions for signal a and vice versa. Figure 5.4 shows an example of lock relation. Signals in lock relation are helpful to solve conflicts [59].
- **Signals with CSC conflicts:** Hiding signals with many CSC conflicts generate ALTSs with fewer conflicts. This gives less information to the signal insertion process. In general, preserving signals with many conflicts leads to more informed decisions and better solutions.
- **Conflict collapse:** A CSC conflict between the states s and s' *collapses* when all the signals present in a path between s and s' are hidden. Figure 5.5 shows an example of a collapsing conflict. A collapsed conflict is not observed and cannot be solved in the projected ALTS. It is thus convenient not to hide signals that collapse conflicts, whenever possible.

Algorithm 6 shows a high level description of the projection algorithm. The first step is to choose the signal a with the largest amount of CSC conflicts. This signal will be the anchor of the new ALTS and will **not** be hidden. The objective is to solve as many conflicts as possible for a. Next, signals are iteratively hidden until the size of the ALTS is below a threshold. At each iteration, the best candidate signal for hiding is obtained.
The criteria (in priority order) to select the best candidates is as follows:

- From the set of signals that do not collapse conflicts, and are not in lock relation with a, the best candidate is the one with the largest ER (highest concurrency).
- In case of a tie, the signal with the smallest number of CSC conflicts is selected.
- If no such signal exists, the lock relation constraint is dropped and the best candidate is selected.
- In case of a tie, the signal that collapses the smallest number of conflicts is selected.

Algorithm 6: PROJECT(A)

```
input  : An ALTS with CSC conflicts.
output: An ALTS with a size under thresholdSize
begin
    $a$ = signalWithLargestCSCconflictNumber($A$)
    while size(A) > thresholdSize do
        $b$ = findSignalToHide($A$, $a$)
        hideSignal($A$, $b$)
    return $A$
```

5.4.2 Re-composition

Re-composition aims to re-introduce the hidden signals after signal insertion. This is achieved by calculating the synchronous product of the original ALTS with the projection after solving CSC.

The conflicts solved in the projected ALTS will also be solved in the re-composed ALTS, whereas the other ones will remain. If some conflicts were collapsed during signal hiding, they will also remain after re-composition.

Re-composition greatly increases concurrency, specially when a high number of signals were hidden during the projection. This may have the undesired effect of increasing the total number of CSC conflicts, thus precluding convergence of the algorithm. For this reason, concurrency reduction is an effective way of avoiding this effect.

5.4.3 Concurrency reduction

We resort to the concurrency reduction transformation proposed in [58]. A concurrency reduction operation over a signal a reduces the size of the ER for that signal and preserves commutativity, determinism and persistency. In particular, it also preserves branching bisimilarity with respect to a.

Concurrency reduction has two positive effects:

- The size of the ALTS is reduced.
- Additional CSC conflicts are solved.

This operation makes some states unreachable and, as a by-product, CSC conflicts are reduced if some of these states are involved in the conflicts. Furthermore, the reduction of states also increases the DC-set of the logic functions and the opportunities to simplify the Boolean equations.

There are multiple ways of performing concurrency reduction, each one deriving a different solution. Figure 5.6 shows an example with different valid reductions.

In this work, a greedy approach has been used to decide how concurrency must be reduced. At every state, the number of possible reductions may be potentially of

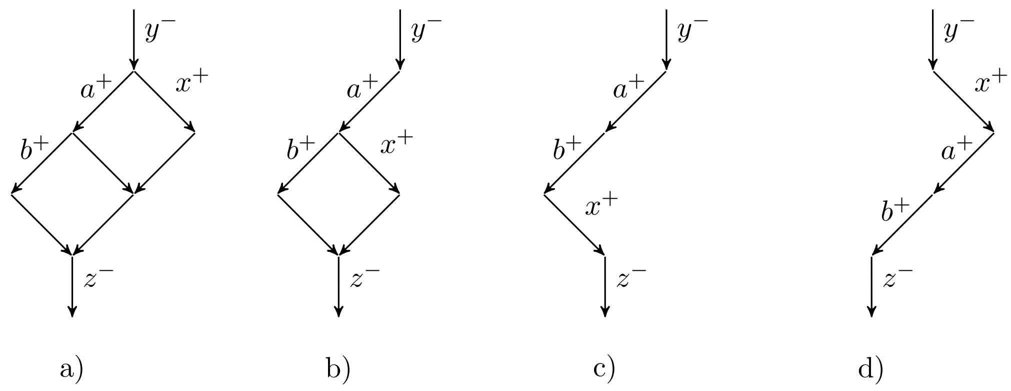


FIGURE 5.6: Concurrency reduction of x: a) full concurrency; b) no concurrency with a; c) no concurrency with a and b, b triggers x; d) no concurrency with a and b, x triggers a.

the order of 2^k, with k being the number of enabled signals at the state. To avoid a worst-case exponential cost in the exploration of highly-concurrent controllers, a limit is defined for the maximum number of solutions that are evaluated.

The following criteria are taken into account to estimate the quality of each solution:

- The number of CSC conflicts that disappear.
- The number of states that become unreachable.
- The number of new trigger signals that appear/disappear.

The number of CSC conflicts that disappear after the reduction is used to maximize the utility of the inserted signal. Furthermore, reducing the number of states as much as possible is important to prevent the size of the ALTS from growing excessively. Finally, the number of triggers is highly correlated with the number of essential literals. This is, at the same time, correlated with the number of literals after logic synthesis [8]. Ideally, the concurrency reduction operation would minimize the number of essential literals. This work proposes to use the trigger events as proxy for essential literals for the sake of performance.

5.4.4 Properties of the algorithm

There are two main properties that this algorithm must have to be an effective and valid technique:

- The computational complexity must be affordable.
- It must preserve the behavior of the specification (branching bisimilarity).

Complexity. For projection, hiding a signal is done by a step of *silencing*, followed by τ-priorization and τ-compression. Silencing can be trivially done in $O(|T|)$, whereas τ-priorization and τ-compression can also be solved in linear time [57].

The other important operation for the projection is selecting the signal to be hidden. The worst-case cost is dominated by the detection of pairs of conflicts that collapse. Theoretically, this operation is $O(|C| + |S|)$, with $|C|$ representing the number of CSC conflicts and $|S|$ the number of states. Although a theoretical upper bound for $|C|$ is $|S|^2$, in practice $|C| < |S|$ for realistic controllers. Since the projection step is repeated on the order of $O(|\Sigma|)$, the average complexity for projection is $O(|S| \times |\Sigma|)$.

The complexity of the re-composition step is the one of the synchronous product. However this is a singular synchronous product $A_1 \times A_2$ in which A_2 is a projection of A_1 with a newly inserted signal. If A_1 has $|S|$ states, the product will have at most $2|S|$ states, under the assumption that the new signal can be highly concurrent with the original specification. Thus, the synchronous product can run in linear time.

The cost of concurrency reduction is maintained as $O(|S| \times |\Sigma|)$ and guaranteed by the heuristic that explores a small amount of options at each state in which the new signal is enabled (discussed in Section 5.4.3).

In general, the average runtime for solving CSC of a large controller can be modeled as:

$$\text{Runtime(CSC)} = O\left(|X| \times (|S| \times |\Sigma| + \text{Runtime}(\text{CSC}_{proj}))\right)$$

where $|X|$ is the number of inserted signals. For every signal, the cost might be dominated by the projection/re-composition steps ($O(|S| \times |\Sigma|)$) or the runtime for solving CSC of the projected controllers. The dominating term will depend on the size of the projected controllers. If they are small, the effort will be dominated by the projection/re-composition. Conversely, a little effort in projection (hiding few signals) will result in larger controllers and a major effort in solving CSC. Defining the appropriate size of the projected controllers is a tuning parameter of the method.

Branching bisimilarity. We need to show that the insertion of a new signal x through the following transformations preserves branching bisimilarity:

$$A_1 \xrightarrow{hide(A_1,\sigma)} A_2 \xrightarrow{insert(A_2,x)} A_3 \xrightarrow{A_3 \times A_1} A_4 \approx_x A_1$$

Theorem 5.2. *Let* $A_1 = (S_1, \Sigma_1, T_1, s_0^1)$ *be a* τ*-free WF-ALTS,* $A_2 = (S_2, \Sigma_2, T_2, s_0^2)$ *such that* $A_2 = hide(A_1, \sigma)$*, with* σ *being a conflict-free set of signals,* $A_3 = (S_3, \Sigma_3, T_3, s_0^3)$ *such that* $A_3 = insert(A_2, x)$*, with* $x \notin \Sigma_1$*, and* $A_4 = (S_4, \Sigma_4, T_4, s_0^4)$ *such that* $A_4 = A_3 \times A_1$*. Then* $A_1 \approx_x A_4$*.*

Proof. First note that the hiding operation and the insertion operation preserves branching bisimilarity, so $A_1 \approx_\sigma A_2$ and $A_2 \approx_x A_3$. Since branching bisimilarity is transitive, $A_1 \approx_{\sigma \cup \{x\}} A_3$. Then, by Theorem 5.1, $A_4 \approx_\sigma A_3$ and $A_4 \approx_x A_1$. □

From [58] it can be easily observed that the basic transformation for concurrency reduction (forward reduction) is equivalent to a τ-priorization assuming the new inserted signal x is silent and confluent. Thus, branching similarity is also preserved when applying concurrency reduction.

5.5 Exploiting concurrency

Usually, large ALTSs show a high degree of concurrency. While the algorithm previously described has a disposition to hide signals that exhibit more concurrency, it is possible to exploit parallelism more explicitly. In this section we describe a pre-processing step for the projection algorithm that dramatically reduces the number of iterations by directly targeting concurrent signals first.

Let us first introduce the concept of *individual excitation region*. As a reminder, the concept of excitation region for a signal a was defined in Chapter 4.4 as the set of states in which a is enabled. This is divided into ER_a^+ and ER_a^- for states in which a^+ and a^- are enabled, respectively. Building on this, an individual excitation region for signal a, $er_a^i \subseteq ER_a$, is a subset of ER_a^+ or of ER_a^- where every state is at most at distance 1 from another state in er_a^i (i.e. there is at most one transition between them). In particular, if a is consistent there is at least two individual excitation regions, one for a^+ and one for a^-.

For the purposes of this thesis, two persistent signals a and b are concurrent if one or both of the following conditions holds:

- For every $er_a^i \subseteq ER_a$, there exists a state $s \in er_a^i$ such that b is enabled, or
- For every $er_b^i \subseteq ER_b$, there exists a state $s \in er_b^i$ such that a is enabled.

Figure 5.7a shows an example of concurrent signals. The partially shown ALTS on the figure depicts all the transitions for signals b and c. Assume that signal a has other persistent transitions that are not shown in the figure. As can be seen, every individual excitation region for signals b and c contains at least one state in which a is enabled. From our definition then, a and b are concurrent, as well as a and c.

A careful examination of the ALTS in Figure 5.7a reveals the presence of two conflicts: one between the states s_1 and s_5, and one between s_6 and s_{10}. Both of these are conflicts for signal b and are *mirrored* by the concurrency with signal a.

Let us assume that we are only interested in solving conflicts for signal b. In that context, signal a is not providing any distinction in the encoding of the states in conflict, so it can be hidden without losing information on the conflicts. This is depicted in Figure 5.7a. After hiding a, the two pairs of states in conflicts are merged into a single pair, s_1, s_5, and the space of states has been reduced. This conflict can be solved by inserting a signal between c^+ and b^-, as represented by Figure 5.7c. Finally, it is possible

$$\begin{tikzcd}
s_1 \arrow[r, "b^+"] \arrow[d, "a^+"] & s_2 \arrow[r, "c^+"] \arrow[d, "a^+"] & s_3 \arrow[r, "b^-"] \arrow[d, "a^+"] & s_4 \arrow[r, "c^-"] \arrow[d, "a^+"] & s_5 \arrow[d, "a^+"] \\
s_6 \arrow[r, "b^+"] & s_7 \arrow[r, "c^+"] & s_8 \arrow[r, "b^-"] & s_9 \arrow[r, "c^-"] & s_{10}
\end{tikzcd}$$

(A) Concurrency ALTS

$$\begin{tikzcd}
s_1 \arrow[r, "b^+"] & s_2 \arrow[r, "c^+"] & s_3 \arrow[r, "b^-"] & s_4 \arrow[r, "c^-"] & s_5
\end{tikzcd}$$

(B) Signal a hidden

$$\begin{tikzcd}
s_1 \arrow[r, "b^+"] & s_2 \arrow[r, "c^+"] & s_3 \arrow[r, "x^+"] & s_3' \arrow[r, "b^-"] & s_4 \arrow[r, "c^-"] & s_5
\end{tikzcd}$$

(C) Inserted signal x

$$\begin{tikzcd}
s_1 \arrow[r, "b^+"] \arrow[d, "a^+"] & s_2 \arrow[r, "c^+"] \arrow[d, "a^+"] & s_3 \arrow[r, "x^+"] \arrow[d, "a^+"] & s_3' \arrow[r, "b^-"] \arrow[d, "a^+"] & s_4 \arrow[r, "c^-"] \arrow[d, "a^+"] & s_5 \arrow[d, "a^+"] \\
s_6 \arrow[r, "b^+"] & s_7 \arrow[r, "c^+"] & s_8 \arrow[r, "x^+"] & s_8' \arrow[r, "b^-"] & s_9 \arrow[r, "c^-"] & s_{10}
\end{tikzcd}$$

(D) Conflicts solved

FIGURE 5.7: Exploiting concurrency to simplify an ALTS

to generalize the insertion into the original ALTS by a synchronous product operation in order to obtain the conflict-free ALTS of Figure 5.7d.

Note that no signal insertion between the states s_1 to s_9 would be able to solve any conflict for signal a due to the concurrency with other signals. If we only wanted to solve conflicts for signal a, hiding signals b and c before performing signal insertion would reduce the search space without affecting the space of solutions.

We propose then to identify all pairs of concurrent signals. This can be done with a worst-case complexity of $O(|S| \times |\Sigma|)$. After choosing to which signals are conflicts going to be solved, all other concurrent signals can be hidden. Algorithm 7 shows the improved projection step. Note that the only difference is the addition of a function *hideConcurrentSignals* that hides all signals concurrent to a.

Algorithm 7: PROJECTIMPROVED(A)

```
input  : An ALTS with CSC conflicts.
output: An ALTS with a size under thresholdSize
begin
    $a$ = signalWithLargestCSCconflictNumber($A$)
    hideConcurrentSignals($a$,$A$)
    while size(A) > thresholdSize do
        $b$ = findSignalToHide($A$, $a$)
        hideSignal($A$, $b$)
    return $A$
```

This process does not have any impact on worst-case complexity, but can dramatically reduce the number of iterations in the projection stage when dealing with highly concurrent ALTS. This is particularly important when considering that these are the ALTSs that suffer from the state explosion problem.

5.6 Rip-off and re-encode

Besides the base algorithm from Section 5.4 and the improvements of Section 5.5, there is an optional step that can be done as a post-processing stage. The idea is very simple: hide one of the inserted signals (rip-off) and find a different solution (re-encode). This process is repeated until no further improvements are observed. This step improves the quality of the results, at the expense of a cost in execution time.

This technique exploits the fact that signals are inserted sequentially and some CSC conflicts might be resolved by more than one signal. Typically, the first inserted signals are eager to resolve a large amount of conflicts. But some of the conflicts may also be resolved later by new inserted signals. By ripping-off some of the first signals and re-encoding, the constraints are relaxed, i.e., the number of CSC conflicts is smaller, and better solutions can be found. In some rare cases, it may even occur that ripping-off some signal does not introduce any CSC conflict, thus detecting that the signal was redundant.

For a fast estimation of the quality of the solutions, the cost function used is similar to the one presented in Chapter 4, which accounts for the number of essential literals, entry points and size of the excitation regions.

Algorithm 8 shows the strategy proposed in this chapter. The algorithm consists of two nested loops. The external loop repeats the process until no further improvements are found. The set of inserted signals (X) is visited in descending order of essential literals, which is an estimation of the logic complexity of the signal. The rationale behind this order is that signals with more literals offer more opportunities for improvement after logic synthesis.

The inner loop stops when some improvement has been detected. After that, the cost of the signals is re-evaluated and the outer loop starts again.

Finally, in order to estimate the quality of the solutions, an efficient algorithm for calculating essential literals is needed. Given a signal a, it is possible to efficiently compute the set of signals for which a is essential. This can be accomplished by grouping all the states that have the same encoding (minus the code for signal a) and checking, for each non-input signal, which ones meet the condition for a to be essential. If the encodings are stored in a hash table, the worst-case complexity is in the order of $\mathrm{O}(|S| \times |\Sigma|)$. Nonetheless, by exploiting bitwise and vectorial instructions in actual hardware, a linear cost $\mathrm{O}(|S|)$ can be obtained, as long as $|\Sigma|$ is on the order of the word size.

Algorithm 8: RipOffReencode(A)

```
input  : An ALTS with CSC property.
output: A re-encoded ALTS.
begin
    $C$ = costSolution($A$)
    do
        $improved$ = False
        $X$ = insertedSignals($A$)
        sortByEssentialLiterals($X$)
        /* in descending order                                   */
        foreach $x \in X$ do
            $B$ = hideSignal($A$, $x$)
            solveCSC($B$)
            $newC$ = costSolution($B$)
            if $newC < C$ then
                $improved$ = True
                $C = newC$
                $A = B$
                break
    while $improved$
    return $A$
```

To compute the essential literals for all signals, the previously described computation needs to be executed for every signal. The cost of finding all the essential literals is O($|S| \times |\Sigma|^2$), or O($|S| \times |\Sigma|$) if the size of $|\Sigma|$ is on the order of the word size.

5.7 Experimental results

This section presents experimental results for the projection and recomposition technique, henceforth called SEPR (State Encoding using Projection and Re-composition). The signal insertion step is performed with Pbase (in its single heuristic version), even though it is possible to use any other state graph based approach (like Petrify). Experimental results include a comparison of the method against Pbase as a baseline and MPSAT for large controllers. Versions with the rip-off and re-encode technique (SEPR-R) are also included.

In every case, the projection steps of the algorithm are performed until the ALTS satisfies all the following conditions:

- $|S| \times |\Sigma| < 500$.
- At least one signal has been hidden.

The last condition is included for the smallest controllers. Some of them are small enough to already satisfy the first condition. Hiding at least one signal guarantees that the technique is used in every instance.

The benchmarks are divided into three groups: small, medium and large. The following subsections present and discuss the results for the different groups, as well as experiments testing the scalability of the approach.

5.7.1 Small controllers

Controllers in this group correspond to the ones presented in Chapter 4 and have less than 1000 states (with the exception of c10). This experiment is performed to give a baseline comparison with PBASE (single heuristic version), since it cannot be used for larger controllers due to the execution time. Additionally, a comparison with the rip-off and re-encode technique is included.

Table 5.1 shows the results in the same format presented in Chapter 4.9. The table compares PBASE (PB), SEPR (SP) and SEPR-R (RR). In some cases, PBASE was not able to solve CSC in less than 10 hours. This is denoted as *Time* in the table.

A summary of the results for Table 5.1 can be found in Table 5.2, which presents a pairwise comparison between different techniques. Row *Solved* reports the number of solved instances. The remaining data in the table only reports the total results for the benchmarks that were solved by both techniques under comparison (i.e. ignoring controllers not solved by both). *Ratio* reports the average ratio of literals between every pair of techniques.

The comparison between PBASE and SEPR shows a significant difference in execution time, without hardly sacrificing quality: the number of literals only increases by 1%. The addition of the rip-off technique has a very minor impact on quality, while increasing execution time. The main reason is because the number of inserted signals is small (less than 3 in most cases), giving few opportunities to explore different re-encodings. Thus, the rip-off technique is not well suited for small controllers. Nonetheless, all controllers were solvable with SEPR-R.

Although the work of this chapter was not originally meant to be used for small controllers, the experiments show that the technique contributes to reduce runtime without having a significant impact on quality.

5.7.2 Medium controllers

In this experiment, the controllers have up to 14,000 states. This size is already out of the scope of the controllers manageable by PBASE. For this reason, MPSAT is used as reference.

The controllers come from different sources. Some of them (master-read versions) correspond to controllers from Chapter 4.9. The *art(m,n)* are parameterized controllers from [60]. They model a synchronization of m pipelines, as shown by the STG depicted

Table 5.1: Experimental results for small controllers. Comparing Pbase (PB), SEPR (SP) and SEPR-R (RR).

			CPU(sec)			Signals/Literals		
Example	I/O	$\|S\|$	PB	SP	RR	PB	SP	RR
adfast	3/3	44	5.6	**1.2**	1.3	**2/14**	**2/14**	**2/14**
alloc-outbound	4/3	17	0.8	**0.2**	0.3	**2/16**	**2/16**	**2/16**
c10	0/10	2046	32.7	**4.1**	15.4	**1/31**	2/32	**1/31**
c6	0/6	126	1.1	**0.2**	0.3	**1/19**	**1/19**	**1/19**
duplicator	2/2	20	0.5	**0.2**	0.2	**2/13**	**2/13**	**2/13**
future	4/4	36	0.4	**0.1**	**0.1**	**1/18**	**1/18**	**1/18**
glc	2/1	17	**0.1**	0.1	0.2	**1/10**	**1**/11	**1**/11
lazy_ring.noncsc	5/3	160	27.6	**0.9**	1.0	**1/22**	**1/22**	**1/22**
mmu0	4/4	174	89.1	**1.8**	2.9	**3/28**	**3**/29	**3**/29
mmu1	4/4	82	8.1	**1.0**	2.1	**2**/25	**2/24**	**2/24**
mod4_counter	1/2	16	0.3	**0.1**	0.2	**2/26**	**2/26**	**2/26**
mr0	5/6	302	Time	**1.2**	2.7	-/-	4/31	**3/30**
mr1	4/5	190	91.6	**7.5**	11.1	**3/26**	**3/26**	**3/26**
nak-pa	4/5	56	0.7	0.2	**0.2**	**1/18**	**1/18**	**1/18**
nowick	3/2	18	0.2	0.1	**0.1**	**1/13**	**1/13**	**1/13**
par2	3/3	28	4.4	**0.5**	0.8	**2/16**	**2/16**	**2/16**
par4	5/5	628	Time	**3.6**	10.7	-/-	**4/32**	**4/32**
pla	0/3	12	0.2	**0.1**	0.1	**1/14**	2/16	2/16
ram-read-sbuf	5/5	36	1.2	0.3	**0.3**	**1/22**	**1/22**	**1/22**
sbuf-ram-write	5/5	58	9.1	**1.9**	2.1	**2/23**	**2**/24	**2**/24
sbuf-read-ctl	2/4	14	0.2	**0.1**	**0.1**	**1/15**	**1/15**	**1/15**
seq2	3/3	12	**0.1**	**0.1**	0.1	**1/8**	**1/8**	**1/8**
seq3	4/4	16	0.6	**0.2**	0.3	**2/14**	**2/14**	**2/14**
seq4	5/5	20	1.4	**0.4**	0.6	**2/19**	**2/19**	**2/19**
seq8	9/9	36	108.7	**41.0**	56.6	**3**/44	5/**43**	5/**43**
seq-mix	4/4	20	2.0	**0.5**	0.9	**3/18**	**3**/20	**3**/20
vbe4a.nousc	3/3	58	5.1	**1.4**	2.2	**3/18**	**3/18**	**3/18**
vbe5a	3/3	44	4.2	**0.7**	0.9	**2/14**	**2/14**	**2/14**
vbe6a.nousc	4/4	128	41.0	**1.2**	1.7	**2/30**	**2/30**	**2/30**
vbe6x.nousc	3/3	48	4.4	**0.3**	0.3	**2**/23	**2/22**	**2/22**
vme_read	8/6	251	16.2	**0.7**	0.7	**1/30**	**1**/31	**1**/31
vme_read_write	3/3	28	1.0	0.5	**0.3**	**1/22**	**1/22**	**1/22**
vme_write	8/6	817	Time	**1.0**	1.0	-/-	**1/36**	**1/36**
vmebus	3/3	24	0.5	**0.2**	**0.2**	**1/19**	**1/19**	**1/19**

in Figure 5.8. These controllers have a high number of states and require a moderately high number of signals to guarantee CSC.

Another set of parameterized controllers is *PpArb(m,n)*, obtained from [61]. They model m pipelines synchronized with arbitration. Figure 5.9 shows an example for *PpArb(2,3)*. These controllers are highly concurrent and have a large set of states, but a comparatively small number of signals. They can be solved with few signal insertions.

The *ParMix(m,n)* controllers are based on the ones presented in [60]. These controllers show a handshake of sequencers, parallelizers and mixers, as represented by Figure 5.10. The original controllers in [60] did not have any CSC conflict. The ones presented here have been modified (by hiding internal signals) such that the sequencer and every parallelizer have conflicts. The result is a controller with a high number of

Table 5.2: Summary for the benchmarks in Table 5.1.

	PB	SP	PB	RR	SP	RR
Solved	31	34	31	34	34	34
CPU (sec)	459	68	459	104	74	118
Signals	53	57	53	56	66	64
Literals	628	634	628	633	733	731
Ratio	**1.00**	**1.01**	**1.00**	**1.01**	**1.00**	**1.00**

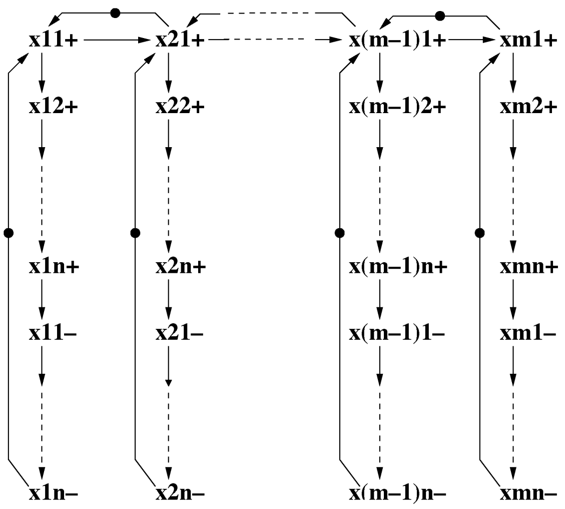


Figure 5.8: Art(*m*,*n*). Source: [60].

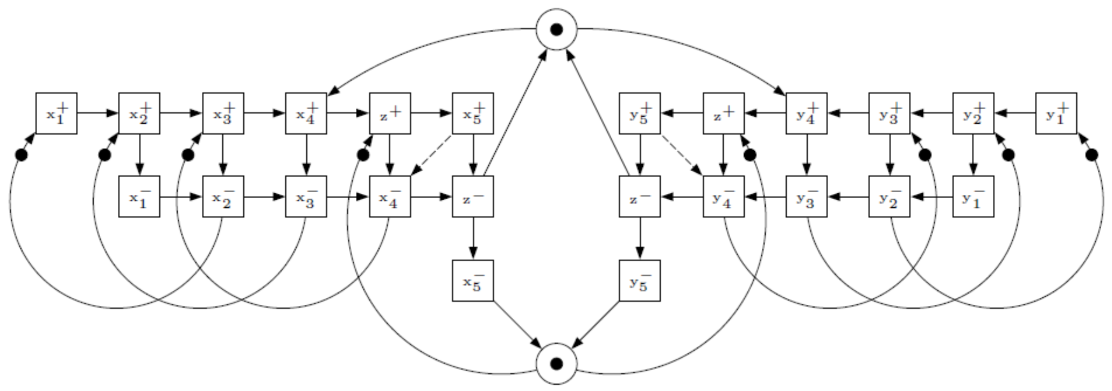


Figure 5.9: PpArb(2,3). Source [61].

signals and CSC conflicts.

Finally, the *SeqPar(n)* controllers are introduced in this work. Like the *ParMix(m,n)*, they represent a handshake of smaller controllers. A *SeqPar(n)* controller represents an n-level tree of alternating handshakes of sequencers and controllers. Figure 5.11 shows an example with three levels. Since every parallelizer and sequencer has CSC conflicts, this class of controllers also contains a high number of signals and CSC conflicts.

Table 5.3 shows results for this experiment. The codeword *Time* is used when a controller could not be solved in less than 10 hours. The codeword *Fail* marks an

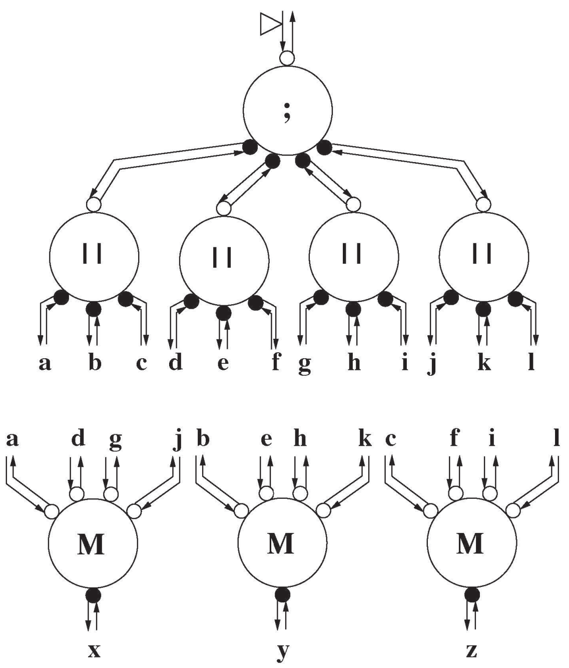


FIGURE 5.10: ParMix(4,3). Source [60].

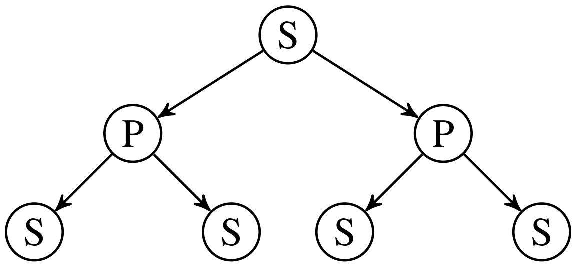


FIGURE 5.11: SeqPar(3).

instance in which a solution could not be found. A summary for Table 5.3 can be found in Table 5.4.

The results show that SEPR generates slightly better results than MPSAT, even before the rip-off technique. In general, the execution time is slightly higher than MPSAT, with the exception of the controller *ParMix(2,4)*. This controller biases the total execution time for MPSAT in Table 5.4. Nonetheless, this result is important because it hints at a trend in the *ParMix* and *SeqPar* controllers: MPSAT takes too long to solve these classes of problems and hits the 10-hour timeout for most of them. A possible explanation is later discussed in Section 5.7.4.

Another singularity is the *master-read* controller. This controller is the original specification of *master-read2*, including the inputs and outputs (all signals in *master-read2* are artificially declared as outputs). The presence of inputs reduces the space of valid solutions since the input properness property prevents the insertion of a signals triggering inputs. While MPSAT can solve *master-read2*, it fails to find a solution for

Table 5.3: Experimental results for medium controllers. Comparing MPSAT (MP), SEPR (SP) and SEPR with Rip-off (RR).

			CPU(sec)			Signals/Literals		
Example	I/O	$\|S\|$	MP	SP	RR	MP	SP	RR
art(3,4)	0/12	2048	**4.1**	16.5	21.6	6/54	**4/49**	**4/49**
art(3,5)	0/15	4000	**10.2**	12.9	16.8	6/57	**4/53**	**4/53**
art(3,6)	0/18	6912	38.7	**23.5**	28.0	6/60	**4/56**	**4/56**
art(4,3)	0/12	10368	**8.7**	39.5	81.8	10/70	**5/69**	**5/69**
master-read	6/7	8932	Fail	**42.9**	160.7	-/-	9/74	**6/59**
master-read2	0/13	8932	**15.6**	147.6	201.1	**5**/75	7/**69**	7/**69**
master-read.1098	6/7	1098	**3.6**	4.4	12.4	6/43	6/44	**4/39**
PpArb(2,3)	2/9	1088	0.3	**0.1**	0.2	**1/39**	**1**/42	**1**/42
PpArb(3,3)	3/13	14336	**0.3**	2.9	4.5	**2/61**	**2**/69	**2**/69
sis-master-read	6/7	1882	0.4	**0.4**	0.5	**1**/39	**1/37**	**1/37**
ParMix(2,4)	0/38	13852	766.8	**53.4**	99.2	**5**/123	6/**121**	6/**121**
ParMix(3,3)	0/46	3796	Time	**76.8**	134.3	-/-	**6/157**	**6/157**
SeqPar(4)	0/72	7452	Time	**194.5**	817.2	-/-	11/210	**9/195**

Table 5.4: Summary for the benchmarks in Table 5.3.

	MP	SP	MP	RR	SP	RR
Solved	10	13	10	13	13	13
CPU (sec)	849	301	849	466	615	1578
Signals	48	40	48	38	66	59
Literals	621	609	621	604	1050	1015
Ratio	**1.00**	**0.98**	**1.00**	**0.97**	**1.00**	**0.97**

master-read. This highlights the increased power of the state-based techniques to find intricate solutions in highly restrictive specifications.

Finally, the rip-off technique shows an overall reduction of 2% in the number of literals with respect to the base approach, at the cost of a higher execution time. The higher number of inserted signals with respect to the small controllers allows this technique to improve the SEPR solutions.

5.7.3 Large controllers

The last experimental results are for large controllers, which was the main motivation for work in this chapter. These controllers can have up to several million of states. The aim of this experiment is to show the scalability of the proposed approach.

The controllers in this test come from the same sources as the ones in the previous results. Table 5.5 reports the results for large controllers, which are summarized in Table 5.6.

SEPR can solve problems up to 4.5 million states in a reasonable time. The rip-off technique significantly increases the execution time, even more than in previous results. This is because there are more candidates to rip-off, which also increases the

Table 5.5: Experimental results for large controllers. Comparing MPSAT (MP), SEPR (SP) and SEPR with Rip-off (RR).

			CPU(sec)			Signals/Literals		
Example	I/O	$\|S\|$	MP	SP	RR	MP	SP	RR
art(4,4)	0/16	$0.3 \cdot 10^5$	**17.0**	38.3	129.7	9/76	**6**/74	**6/72**
art(5,4)	0/20	$5.2 \cdot 10^5$	**25.4**	677.7	1676.2	10/104	**7/100**	**7/100**
art(5,5)	0/25	$16 \cdot 10^5$	**225.8**	2210.3	6422.0	12/105	**8**/115	**8/102**
par8	9/9	$3.9 \cdot 10^5$	**9.9**	554.7	1833.7	**8/64**	**8/64**	**8/64**
PpArb(2,6)	2/15	$0.7 \cdot 10^5$	**1.6**	7.5	8.6	**1/69**	**1**/72	**1**/72
PpArb(2,9)	2/21	$44.6 \cdot 10^5$	**6.5**	808.0	826.8	**1/99**	**1**/102	**1**/102
ParMix(4,4)	0/86	$1.1 \cdot 10^5$	Time	**480.2**	1911.6	-/-	**11**/313	**11/298**
ParMix(5,4)	0/110	$2.2 \cdot 10^5$	Time	**812.9**	7126.0	-/-	16/411	**14/387**
SeqPar(5)	0/126	$2.4 \cdot 10^5$	Time	**892.2**	4505.0	-/-	12/396	**10/394**

Table 5.6: Summary for large controllers.

	MP	SP	MP	RR	SP	RR
Solved	6	9	6	9	9	9
CPU (sec)	286	4296	286	10897	6482	24440
Signals	41	31	41	31	70	66
Literals	517	527	517	512	1647	1591
Ratio	**1.00**	**1.02**	**1.00**	**0.99**	**1.00**	**0.97**

opportunities to generate better results. This last approach allows for solutions with higher quality than those of MPSAT. Every instance can be solved with the SEPR and SEPR-R.

Even though MPSAT uses structural methods, it solves CSC using a SAT formulation of the problem [42]. The runtime highly depends on the size of the SAT formula, which is mainly determined by the size of the unfolding and the number of signals. Although the unfolding can grow exponentially under the presence of multiple choices in the specification, in practice the number of signals is the one that has the largest impact on MPSAT runtime. The following section discusses the scalability of different approaches.

It is also important to note that only examples suitable for MPSAT have been selected, which need to have an underlying safe Petri net. These constraints do not apply for state-based methods.

5.7.4 Scalability

This section studies the scalability of SEPR with regard to MPSAT, with the goal of comparing a state-based method with a structural one. The experiments are performed with three suites of benchmarks: *Sequencer(n)*, *Art(m, n)* and *Parallelizer(n)*. The circuits have been scaled with the parameter n. In the case of *Art(m, n)*, m has been set at 3. The following table shows how these circuits grow with n:

	Seq(n)	*Art*(3,n)	*Par*(n)
Signals	$2n+2$	$3n$	$2n+2$
States	$4n+4$	$32n^3$	5^n+3

Signals grow linearly with n in all cases. The main difference is in the size of the set of states. For *Seq*, it grows linearly, whereas for *Art* and *Par* the growth is cubic and exponential, respectively.

Figure 5.12 reports the execution time of these benchmarks for MPSAT, SEPR and SEPR-R. The x-axis represents n and the y-axis represents the execution time in seconds (log scale). Table 5.7 reports the total sum of literals after logic synthesis for all controllers of every class.

TABLE 5.7: Total number of literals for controller classes.

	Seq(n)	*Art*(3,n)	*Par*(n)
MPSAT	1339	567	352
SEPR	1240	565	352
Rip-off	1220	521	352

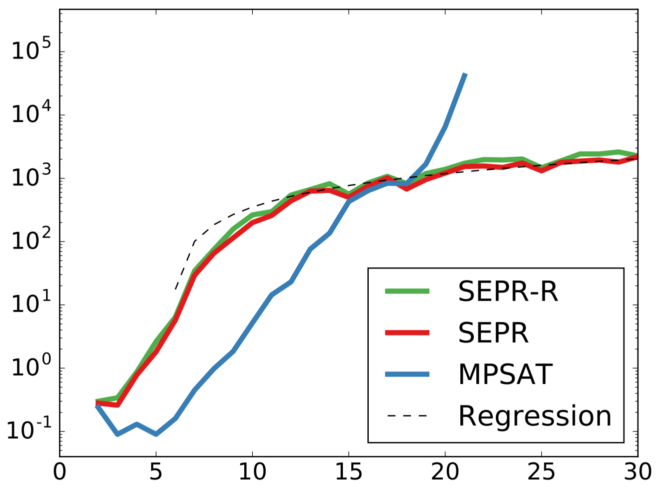
Figure 5.12a depicts the results for *Seq*. The dashed line represents a linear regression of SEPR, with $R^2 = 0.946$. SEPR and SEPR-R manifest a linear asymptotic behavior, whereas MPSAT hits a computational wall around $n = 20$. The main reason is that MPSAT does not scale well with the number of signals.

The results for *Art* are reported in Figure 5.12b. In this case, the dashed line is a cubic polynomial regression of SEPR, with $R^2 = 0.988$. This is consistent with the cubic polynomial growth of the number of states. MPSAT shows an exponential behavior, mostly dominated by the number of signals.

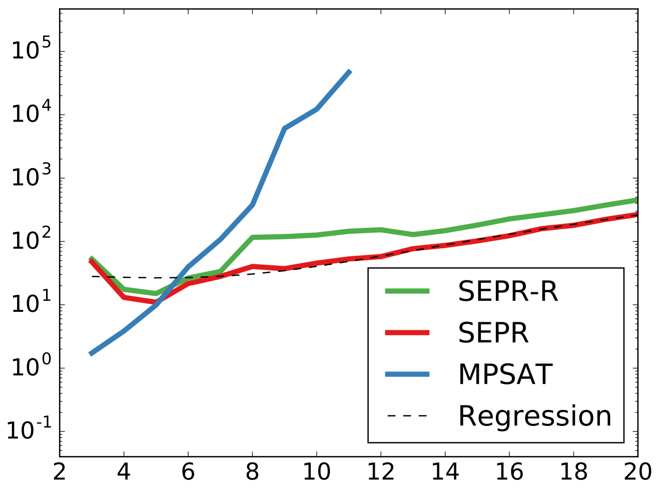
Finally, Figure 5.12c shows results for *Par*. In this case, the complexity of the ALTS is dominated by the number of states, rather than the number of signals. The dashed line represents an exponential regresion of SEPR[2]. Clearly, MPSAT overtakes the state-based methods since the number of states grows much faster than the number of signals. Working with the unfolding of a Petri net, rather than its reachability set, is a clear advantage in this case.

MPSAT is more scalable for large state spaces that can be succinctly represented by a Petri net. However, the runtime grows exponentially with the number of signals. The main reason is the way that MPSAT estimates the logic complexity of the circuit, using a quadratic number of SAT variables to encode the trigger relations between pairs of signals [42].

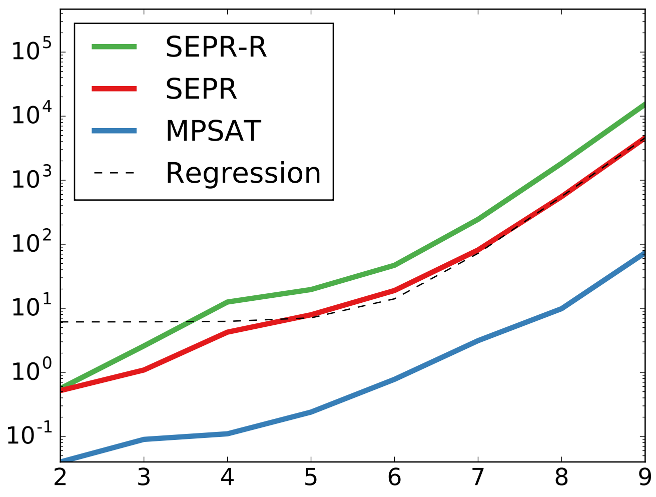
[2]The regression is on the order of 4.5^n (states grow on the order of 5^n). Given the small number of points and the dominance of the large values, the regression may not be sufficiently meaningful. However it helps to hypotesize the exponential relationship with the state space.



(A) Seq(n)



(B) Art (3,n)



(C) Par(n)

FIGURE 5.12: Runtime growth, in seconds (y-axis) with the size of the ALTS, defined by n (x-axis).

5.7.5 Final remarks

The results show a good picture of how SEPR scales. For small controllers, it reduces runtime while maintaining quality. For medium controllers, the quality of the solution and runtime are slightly better than the structural methods.

For controllers with a large number of signals, SEPR can go much beyond the complexity wall hit by other tools (e.g., MPSAT or petrify). The base version of the tool, SEPR, sometimes provides solutions with slightly lower quality than MPSAT, but the re-encoding strategy using rip-off gives an opportunity to improve the results, specially in those controllers that require a larger number of encoding signals. In fact, it generates the best results for most cases, with the exceptions of the *PpArb* class of controllers, which are solved with just one signal.

Finally, the strongest advantages of SEPR are in the number of problems solved and the scalability of the approach. Structural methods depend on the Petri net structure, which limits the solutions that can be found. In the case of MPSAT, for example, it cannot solve unsafe nets. But even when safe Petri nets are used (as in the case of the benchmarks presented here), some other limitations might arise. As for scalability, this approach grows linearly with the number of signals and states. In the case of state explosion typical of high concurrency, this limits the size of the controllers than can be solved (to the order of 10^6 states). But when the controllers have large number of signals, results show that SEPR still manages to grow linearly, as opposed to the exponential growth of MPSAT.

5.8 Conclusions

This chapter has presented a novel technique to address the problem of state encoding for large asynchronous controllers. The approach allows to project a large specification onto a subset of signals and obtain a smaller one suitable to be handled by state-based encoding techniques. The complete asynchronous controller is recovered by re-composing the original specification with the projected solution.

Results show that asynchronous controllers of several million states are now within reach of state-based encoding techniques. Furthermore, it can speed up the encoding for controllers of smaller sizes. This allows state-based techniques to effectively compete with structural methods and handle controllers that can be generated from different formalisms for which no encoding tools exist yet.

Chapter 6

Conclusions

This thesis contributes to the area of variability tolerant circuits. It proposes techniques in two fields: synthesis of Ring Oscillators Clocks, that adapt their period to variability conditions, and synthesis of asynchronous controllers, which are implicitly resistant to variability. While methods in both categories concur in their ultimate goals, their applicability and approaches are different enough to deserve a separate analysis. This chapter summarizes the contributions on each of these approaches and concludes this thesis.

Delay lines and Ring Oscillator Clocks

As technology scaling reaches its limits and variability grows, mitigating its impact is becoming more and more necessary. Several techniques exist for this purpose, yet most of them either offer limited improvements or require high implementation costs in terms of complexity or area. This thesis proposed, in Chapter 2, a novel technique that reduces most of the impact of global variability by substituting the PLL for a Ring Oscillator Clock (ROC). This method offers similar benefits to other, more aggressive, approaches such as Razor [14], while dramatically simplifying the design. In fact, an ROC can act as a drop-in replacement for a PLL. It is even possible to use both a classical clock and a ROC in the same design. Because of its size, there is no increase in complexity to speak off, and the cost in area is negligible.

In order to implement ROCs, Chapter 3 introduces an algorithmic technique to design all-digital delay lines (DL). DLs are designed with the purpose of having a delay representative of a specific circuit for all variability conditions (PVT corners). When connected in a loop, a DL can act as a RO whose period is affected by variability. Since the variability of a DL is correlated to that of a circuit, the period of an RO instantaneously adapts to changes in the environment, such as temperature or voltage. Besides building ROs, DLs have multiple uses in systems that require accurate tracking of variability. These include bundled-data circuits and performance monitors, which are often used by other techniques, such as adaptive clocks.

While a delay line is not a novel circuit in itself, this thesis proposed a new technique to design all-digital DLs. In this case, all the components for the DL are limited to standard cells from standard cell libraries. This considerably increases the ease and approachability of designing such circuits. In particular, no costly custom design is needed and conventional EDA tools can be used to verify them. Additionally, this thesis showed how DLs can be built to be configurable at the post-silicon stage. This further increases the utility and ability of DLs to reduce variability margins. Finally, a comprehensive series of experiments over a well-known benchmark suite, I99T from ITC99, shows very promising results about the accuracy that these circuits exhibit in tracking variability.

State encoding for asynchronous controllers

State encoding is one of the most challenging problems in the synthesis of asynchronous controllers modeled in input/output mode. There exist a few techniques that solve the problem in effective ways. But, as Chapter 4 shows, those approaches still have margin for improvement. This thesis presented Pbase, a SAT-based approach that solves the state encoding problem at the state level. By working at the state level, this technique leverages a larger search space with respect to structural methods that work at the Petri net level. Furthermore, the encoding in SAT guarantees that, if a solution for a single signal insertion exists, it will be found. These two properties give important advantages over previous approaches. Pbase can work with any type of correctly specified controller, bypassing restrictions of other techniques (e.g. unsafe nets). In some cases, solutions are found for controllers that no other tool could solve. Finally, results over a heterogeneous benchmark suite show that Pbase finds the best solutions, in terms of number of literals, for most of the circuits tested.

Unfortunately, these improvements do not come without an important drawback. Working at the state level means that controllers may fall into the state-explosion problem. This is a typical issue for models in which concurrent events are modeled by sequential interactions, causing an exponential growth in the number of states while in presence of concurrency. This large state space dramatically increases the execution time for circuits with a high degree of concurrency. Furthermore, since Pbase uses SAT to find solutions, a larger number of states increase the risk of falling into an exponential runtime. This effectively puts a limit on the size of controllers that can be solved by Pbase. Indeed, benchmarks show several instances in which Pbase requires so much runtime that execution is aborted after reaching a tiemout limit.

In order to overcome this limitation, Chapter 5 proposes SEPR, a method that allows state-based techniques to effectively solve large controllers. This is achieved by a process of projection and re-composition, in which a large part of a circuit's behavior is simplified, or hidden, until the size of the state space is suitable for solving. Once a solution has been found in this reduced controller, it is projected into the original circuit by means of a synchronous product. The process is then repeated until a full encoding is achieved. With the use of this technique, controllers with a size up to 10^6 states are solvable by PBASE or any other technique working at the state-level.

The results presented show how execution time may be dramatically reduced for PBASE when using SEPR for small controllers. In the case of large controllers, the execution time often remains higher than competing structural methods. Nonetheless, this is rewarded by an increased quality in the solutions found with respect to those same approaches. Furthermore, the other main advantage of PBASE, finding solutions when other approaches are unable to, remains present in SEPR. Finally, even though structural methods scale better with the number of states, results show that this approach can surpass structural methods for controllers with a large number of signals.

An aspect not addressed by this work is related to the way quality is measured. This thesis uses a well established metric to gauge the quality of a solution: the number of literals of the Boolean formula after synthesis. Oftentimes, the use case of asynchronous controllers is more concerned about latency than complexity. Using metrics to approximate delay and latency is left as future work. Another avenue left unexplored is power consumption. This is a metric that is often overlooked in synthesis of asynchronous controllers, due to their often relatively small size. Yet it may become relevant for larger instances or larger amounts of instances.

Bibliography

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics (magazine)*, vol. 38, no. 8, pp. 114–117, Apr. 1965.

[2] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach.* Elsevier, 2017.

[3] H. Jones, "Why migration to 20nm bulk CMOS and 16/14nm FinFETs is not best approach for the semiconductor industry," International Business Strategies, Los Gatos, CA, Tech. Rep., Jan. 2014.

[4] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, *Logic Synthesis of Asynchronous Controllers and Interfaces.* Springer-Verlag, 2002.

[5] A. Moreno and J. Cortadella, "Synthesis of all-digital delay lines," in *Asynchronous Circuits and Systems (ASYNC), 2017 23rd IEEE International Symposium on.* IEEE, 2017, pp. 75–82.

[6] J. Cortadella, L. Lavagno, P. López, M. Lupon, A. Moreno, A. Roca, and S. S. Sapatnekar, "Reactive clocks with variability-tracking jitter," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on.* IEEE, 2015, pp. 511–518.

[7] J. Cortadella, M. Lupon, A. Moreno, A. Roca, and S. S. Sapatnekar, "Ring oscillator clocks and margins," in *Asynchronous Circuits and Systems (ASYNC), 2016 22nd IEEE International Symposium on.* IEEE, 2016, pp. 19–26.

[8] A. Moreno and J. Cortadella, "State encoding of asynchronous controllers using pseudo-Boolean optimization," in *Asynchronous Circuits and Systems (ASYNC), 2018 24rd IEEE International Symposium on.* IEEE, 2018, pp. 9–16.

[9] A. Moreno and J. Cortadella, "State-based encoding of large asynchronous controllers," *IEEE access*, vol. 6, pp. 61 503–61 518, 2018.

[10] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs.* Springer, 2009.

[11] A. Datta, S. Bhunia, J. H. Choi, S. Mukhopadhyay, and K. Roy, "Speed binning aware design methodology to improve profit under parameter variations," in *Proceedings of the 2006 Asia and South Pacific Design Automation Conference.* IEEE Press, 2006, pp. 712–717.

[12] B. Cory, R. Kapur, and B. Underwood, "Speed binning with path delay test in 150-nm technology," *IEEE Design & Test of Computers*, vol. 20, no. 5, pp. 41–45, Sep. 2003.

[13] V. Zolotov, C. Visweswariah, and J. Xiong, "Voltage binning under process variation," in *Proceedings of the 2009 International Conference on Computer-Aided Design.* ACM, 2009, pp. 425–432.

[14] D. Ernst, S. N. S. Kim, Das, S. Pant, R. Rao, T. Pham, C. Zieslera, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: a low-power pipeline based on circuit-level timing speculation," in *IEEE Micro*, 2003, pp. 7–18.

[15] K. A. Bowman, J. W. Tschanz, N. S. Kim, J. C. Lee, C. B. Wilkerson, S. Lu, T. Karnik, and V. De, "Energy-efficient and metastability-immune timing-error detection and instruction-replay-based recovery circuits for dynamic-variation tolerance," in *International Solid State Circuits Conference*, 2008, pp. 402–403.

[16] M. S. Gupta, J. A. Rivers, P. Bose, G.-Y. Wei, and D. Brooks, "Tribeca: design for PVT variations with local recovery and fine-grained adaptation," in *Int. Symp. on Microarchitecture*, 2009, pp. 435–446.

[17] D. Hand, M. Trevisan, H. Hsin-Ho, C. Danlei, F. Butzke, L. Zhichao, M. Gibiluka, M. Breuer, N. L. V. Calazans, and P. Beerel, "Blade – a timing violation resilient asynchronous template," in *IEEE Int. Symp. on Asynchronous Circuits and Systems*, May 2015, pp. 21–28.

[18] N. Kurd, P. Mosalikanti, M. Neidengard, J. Douglas, and R. Kumar, "Next generation Intel core micro-architecture (Nehalem) clocking," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 4, pp. 1121–1129, 2009.

[19] K. Chae and S. Mukhopadhyay, "All-digital adaptive clocking to tolerate transient supply noise in a low-voltage operation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 59, no. 12, pp. 893–897, 2012.

[20] C. Lefurgy, A. Drake, M. Floyd, M. Allen-Ware, B. Brock, J. Tierno, J. Carter, and R. Berry, "Active guardband management in Power7+ to save energy and maintain reliability," *IEEE Micro*, vol. 33, no. 4, pp. 35–45, Jul. 2013.

[21] K. Bowman, C. Tokunaga, T. Karnik, V. De, and J. Tschanz, "A 22 nm all-digital dynamically adaptive clock distribution for supply voltage droop tolerance," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 4, pp. 907–916, Apr. 2013.

[22] K. L. Wong, T. Rahal-Arabi, M. Ma, and G. Taylor, "Enhancing microprocessor immunity to power supply noise with clock-data compensation," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 4, pp. 749–758, 2006.

[23] D. Jiao, J. Gu, and C. H. Kim, "Circuit design and modeling techniques for enhancing the clock-data compensation effect under resonant supply noise," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 10, pp. 2130–2141, 2010.

[24] A. Grenat, S. Pant, R. Rachala, and S. Naffziger, "5.6 adaptive clocking system for improved power efficiency in a 28nm x86-64 microprocessor," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*. IEEE, 2014, pp. 106–107.

[25] Q. Liu and S. S. Sapatnekar, "Synthesizing a representative critical path for post-silicon delay prediction," in *Proceedings of the 2009 international symposium on Physical design*. ACM, 2009, pp. 183–190.

[26] G. D. Carpenter, A. J. Drake, H. S. Deogun, M. S. Floyd, N. K. James, R. M. Senger *et al.*, "Circuit for dynamic circuit timing synthesis and monitoring of critical paths and environmental conditions of an integrated circuit," US Patent 7,576,569, Aug., 2009.

[27] L. Xie and A. Davoodi, "Representative path selection for post-silicon timing prediction under variability," in *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 386–391.

[28] A. Singhvi, M. T. Moreira, R. N. Tadros, N. L. V. Calazans, and P. A. Beerel, "A fine-grained, uniform, energy-efficient delay element for FD-SOI technologies," in *2015 IEEE Computer Society Annual Symposium on VLSI*, Jul. 2015, pp. 27–32.

[29] G. Heck, L. S. Heck, A. Singhvi, M. T. Moreira, P. A. Beerel, and N. L. V. Calazans, "Analysis and optimization of programmable delay elements for 2-phase bundled-data circuits." in *VLSI Design*, 2015, pp. 321–326.

[30] M. Bhushan, A. Gattiker, M. B. Ketchen, and K. K. Das, "Ring oscillators for CMOS process tuning and variability control," *IEEE Transactions on Semiconductor Manufacturing*, vol. 19, no. 1, pp. 10–18, Feb. 2006.

[31] T. B. Chan, P. Gupta, A. B. Kahng, and L. Lai, "DDRO: A novel performance monitoring methodology based on design-dependent ring oscillators," in *Thirteenth*

International Symposium on Quality Electronic Design (ISQED), Mar. 2012, pp. 633–640.

[32] M. Maymandi-Nejad and M. Sachdev, "A digitally programmable delay element: design and analysis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 5, pp. 871–878, Oct. 2003.

[33] W. Hua, R. N. Tadros, and P. Beerel, "2 ps resolution, fine-grained delay element in 28 nm FDSOI," *Electronics Letters*, vol. 51, no. 23, pp. 1848–1850, 2015.

[34] N. R. Mahapatra, S. V. Garimella, and A. Tareen, "An empirical and analytical comparison of delay elements and a new delay element design," in *IEEE Computer Society Workshop on VLSI, 2000. Proceedings*, 2000, pp. 81–86.

[35] J. Tschanz, K. Bowman, S. Walstra, M. Agostinelli, T. Karnik, and V. De, "Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance," in *Int. Symp. on VLSI Circuits*, 2009, pp. 112–113.

[36] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Design Test of Computers*, vol. 17, no. 3, pp. 44–53, Jul. 2000.

[37] Synopsys, "Synopsys PrimeTime," http://www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx.

[38] R. Bisiani, "Beam search," in *Encyclopedia of Artificial Intelligence*, S. Shapiro, Ed., 1987, pp. 56–58.

[39] R. M. Fuhrer, B. Lin, and S. M. Nowick, "Symbolic hazard-free minimization and encoding of asynchronous finite state machines," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, 1995.

[40] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "A region-based theory for state assignment in speed-independent circuits," *IEEE Transactions on Computer-Aided Design*, vol. 16, no. 8, pp. 793–812, Aug. 1997.

[41] P. Vanbekbergen, B. Lin, G. Goossens, and H. de Man, "A generalized state assignment theory for transformations on signal transition graphs," *Journal of VLSI Signal Processing*, vol. 7, no. 1/2, pp. 101–115, Feb. 1994.

[42] V. Khomenko, "Efficient automatic resolution of encoding conflicts using stg unfoldings," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 17, no. 7, pp. 855–868, 2009.

[43] P. Barth, “A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization,” Max Planck Institut für Informatik, Saarbrücken, Germany, Tech. Rep. MPI-I-95-2-003, 1995.

[44] R. Milner, *Communication and concurrency.* Prentice hall New York etc., 1989, vol. 84.

[45] R. J. V. Glabbeed and W. P. Weikland, “Branching time and abstraction in bisimulation semantics,” *Journal of the ACM*, no. 3, pp. 555–600, May 1996.

[46] J. Groote and M. Mousavi, *Modeling and Analysis of Communicating Systems.* The MIT Press, 2014.

[47] P. Vanbekbergen, B. Lin, G. Goossens, and H. De Man, “A generalized state assignment theory for transformations on signal transition graphs,” *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 7, no. 1-2, pp. 101–115, 1994.

[48] R. J. van Glabbeek, “The linear time - branching time spectrum,” in *CONCUR ’90 Theories of Concurrency: Unification and Extension*, J. C. M. Baeten and J. W. Klop, Eds. Springer Berlin Heidelberg, 1990, pp. 278–297.

[49] N. Eén and N. Sörensson, “Translating Pseudo-Boolean Constraints into SAT,” *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, pp. 1–25, 2006.

[50] T. Philipp and P. Steinke, “PBLib – A Library for Encoding Pseudo-Boolean Constraints into CNF,” in *Theory and Applications of Satisfiability Testing – SAT 2015*, ser. LNCS, M. Heule and S. Weaver, Eds. Springer, 2015, vol. 9340, pp. 9–16.

[51] N. Eén and N. Sörensson, “An Extensible SAT-solver,” in *6th Int. Conf. on Theory and Applications of Satisfiability Testing*, 2003, pp. 502–518.

[52] J. Carmona and J. Cortadella, “Encoding large asynchronous controllers with ILP techniques,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 20–33, 2008.

[53] G. Birtwistle and K. S. Stevens, “The family of 4-phase latch protocols,” in *Asynchronous Circuits and Systems, 2008. ASYNC’08. 14th IEEE International Symposium on.* IEEE, 2008, pp. 71–82.

[54] “Benchmark repository,” http://www.cs.upc.edu/~jordicf/petrify/benchmarks.

[55] D. Wist, R. Wollowski, M. Schaefer, and W. Vogler, “Avoiding irreducible CSC conflicts by internal communication,” *Fundamenta Informaticae*, vol. 95, no. 1, pp. 1–29, 2009.

[56] J. F. Groote, D. N. Jansen, J. J. A. Keiren, and A. J. Wijs, "An $O(m \log n)$ Algorithm for Computing Stuttering Equivalence and Branching Bisimulation," *ACM Trans. Comput. Logic*, vol. 18, no. 2, pp. 13:1–13:34, Jun. 2017.

[57] J. F. Groote and J. van de Pol, "State space reduction using partial τ-confluence," in *International Symposium on Mathematical Foundations of Computer Science*. Springer, 2000, pp. 383–393.

[58] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Automatic handshake expansion and reshuffling using concurrency reduction," in *Proc. of HWPN*, vol. 98, 1998, pp. 86–110.

[59] P. Vanbekbergen, G. Goossens, F. Catthoor, and H. J. De Man, "Optimized synthesis of asynchronous control circuits from graph-theoretic specifications," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 11, no. 11, pp. 1426–1438, 1992.

[60] J. Carmona, J.-M. Colom, J. Cortadella, and F. García-Vallés, "Synthesis of asynchronous controllers using integer linear programming," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 9, pp. 1637–1651, 2006.

[61] V. Khomenko, M. Koutny, and A. Yakovlev, "Detecting state encoding conflicts in STG unfoldings using SAT," *Fundamenta Informaticae*, vol. 62, no. 2, pp. 221–241, 2004.