

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)
BarcelonaTech

**Synthesis of timing paths
with delays adaptable to
integrated circuit variability**

by

Alberto Moreno Vega

supervised by

Jordi Cortadella Fortuny

MASTER IN INNOVATION AND RESEARCH IN INFORMATICS
Advanced Computing

Facultat d'Informàtica de Barcelona (FIB)
Computer Science Department

Defence Data: 10th July 2015

DATOS DEL PROYECTO

Título del proyecto: Synthesis of timing paths with delays adaptable to integrated circuit variability

Nombre del estudiante: Alberto Moreno Vega

Titulación: Master in Innovation and Research in Informatics

Créditos: 30

Director: Jordi Cortadella Fortuny

Departamento: Ciències de la computació

MIEMBROS DEL TRIBUNAL *(nombre y firma)*

Presidente: Jordi Tubella Murgadas

Vocal: Jordi Petit Silvestre

Secretario: Enric Morancho Llena

CALIFICACIÓN

Calificación numérica:

Calificación descriptiva:

Fecha:

Abstract

Since the introduction of transistors and the first integrated circuits there has been a trend for increasing performance in the semiconductor industry. This has been primarily lead by the continuous shrinking in transistor size which enabled, roughly every two years, to double the number of transistors per area unit in a chip. This shrinking trend has become to be known as Moores law and it makes possible that, at each generation, transistors are faster, with lower power consumption and can be manufactured at lower costs.

Nonetheless, benefits from new generations of transistors have been diminishing over the last years and it is now argued that Moores law may not hold for much longer. One of the main causes for these diminishing returns is variability. Recent generations, or technology nodes, produce transistors so small that it is no longer possible to manufacture them with the exact desired properties. A small deviation in manufacture that would have once been negligible may today completely change the behaviour of a component. Furthermore, smaller transistors are much more susceptible to environmental changes, such as temperature or small voltage fluctuations.

Circuits are subject to strict constraints that set very definite operational ranges for its components in properties such as power and delay. Variations in the characteristics of transistors have the risk of causing violations in those constraints, producing a failure in the circuit. Modern design deals with this problem by using huge margins on the behaviour of components and often assumes worst case scenarios, effectively sacrificing great amounts of performance. Moreover, these worst case scenarios are rarely reached in practice, and, even if they are, it is only for very short periods of time.

In this work we propose a different way of dealing with variability. Instead of adding margins to the behaviour of transistors, it may make sense to *relax* constraints and make them change in the same way that variations over the rest of the system does. In fact, variability in circuits is a problem because, while silicon components change its behaviour, the clock that sets timing constraints is virtually agnostic to variability. Our proposal is the substitution of this clock for a small circuit that, while it still generates a

periodic signal, it is as susceptible to variability as the rest of the system. This approach gives the possibility of ignoring most of the margins applied at design stage and enables designing for normal operation rather than worst case scenarios.

One of the ways of implementing a circuit that gives a periodic signal is by using a Ring Oscillator. A Ring Oscillator is a path of gates with an odd number of inversions connected in a feedback loop. The period of the signal generated by such a circuit depends on the delay of the gates used. At the same time, the delay of the cells strongly depends on the variability sources at which their transistors are subject. The problem remains, though, of creating a sequence of gates with a delay long enough that it does not cause timing violations in the rest of the system. Furthermore, due to changes in variability, this must hold regardless of manufacture deviations, environmental changes or voltage fluctuations.

The aim of this project is, specifically, to generate paths for Ring Oscillators that are adaptable to variability in a way that do not violate timing constraints of the system. Moreover, periods of these circuits must be as small as possible, as this is directly related with performance. In order to accomplish this, we will present in this project different schemes for designing Ring Oscillators. These schemes may also present the possibility of dynamically changing the period as well as to monitor the current state of variability in the system, allowing for faster or safer operation modes.

The main contributions of this work lie in the techniques for designing the schemes presented here. It is not a simple task to create a path of gates that meets constraints for every possible scenario and we will formally prove that it is, in fact, a complex problem. A series of algorithms will be presented with the aim of generating Ring Oscillators that, while ensuring a correct behaviour of the system, perform as fast as possible. Finally, these algorithms will be analysed and benchmarked with a series of circuits in order to assess quality of the solutions.

This document starts by presenting variability in detail, as well as introducing necessary concepts such as process corners and static timing analysis. Afterwards, current state of the art in the field of reducing variability margins is presented and discussed. Schemes for Ring Oscillators and monitoring circuits are then be presented and its functionalities described in detail. With this, we formalize the problem of generating paths of gates adaptable to variability sources, including a discussion of its complexity. We then suggest a series of algorithms that solve path generation for different schemes of Ring Oscillators. These algorithms are finally tested in real circuits and the quality of their solutions discussed.

Contents

Abstract	v
List of Figures	xi
Glossary	xiii
1 Introduction	1
1.1 Introduction	1
1.2 Variability	3
1.2.1 Process Variations	4
1.2.1.1 Critical dimensions	4
1.2.1.2 Random dopant fluctuation	5
1.2.1.3 Gate Oxide Thickness	5
1.2.2 Voltage fluctuations	5
1.2.3 Temperature	6
1.2.4 Aging	7
1.3 Living with variability	7
1.4 Static Timing Analysis	8
1.4.1 Corners	9
1.4.2 Delay computation and propagation	11
1.5 Objectives	14
1.5.1 Ring Oscillator as a clock substitutive	15
1.6 Organization	16
2 State of the art	17
2.1 Introduction	17
2.2 Parametric binning	17
2.3 Razor	19
2.4 CRISTA and Trifecta	20
2.5 Performance Monitors	21
2.6 Adaptive Clocks	22
3 Ring Oscillator schemes	25
3.1 Introduction	25
3.2 Monitor	26
3.3 Ring Oscillator	28
3.3.1 Simple Ring Oscillator	29
3.3.2 Dual Ring Oscillator	30

3.3.3	Configurable Ring Oscillator	31
3.4	Summary	33
4	Path Generation	35
4.1	Introduction	35
4.2	Formal definition: Basic problem	35
4.3	Expanding the basic problem	39
4.3.1	Extensions on the problem	40
4.3.1.1	Additional cells	40
4.3.1.2	Wire models	40
4.3.2	Additional constraints	41
4.3.2.1	Maximum Slew	41
4.3.2.2	Length of the path	41
4.3.2.3	Inverted edge	42
4.3.2.4	Parity of the inversions	42
4.3.3	Variations on the problem	42
4.3.3.1	Monitor	42
4.3.3.2	Simple Ring Oscillator	43
4.3.3.3	Dual Ring Oscillator	44
4.3.3.4	Configurable path	45
4.4	Problem analysis	47
4.4.1	3-SAT polynomial reduction	48
4.4.1.1	Reduction	49
4.5	Solving the problem	51
4.5.0.2	Optimal solutions	52
4.5.0.3	Non-optimal solutions	53
4.6	Beam Search algorithm	54
4.6.1	Monitor algorithm	55
4.6.2	Simple Ring Oscillator Algorithm	57
4.6.3	Dual Ring Oscillator Algorithm	57
4.6.4	Configurable Path Algorithm	58
4.7	Cost Function	60
4.7.1	Variance over the normalized delay	61
4.7.2	Squared normalized distance	62
4.7.3	Mixed heuristic	63
4.7.4	Slew heuristic	64
4.8	Summary	64
5	Experimental results	65
5.1	Introduction	65
5.2	Heuristics and libraries	67
5.3	Simple and Dual Ring Oscillators	70
5.4	Monitor	71
5.5	Configurable Ring Oscillator	73
5.6	Place and Route experiment	75
5.7	Conclusions	77

6	Conclusions	79
6.1	Summary	79
6.2	Future Work	81

	Bibliography	83
--	---------------------	-----------

List of Figures

1.1	Transistor count evolution	2
1.2	Mask layout correction	5
1.3	Delay variations with voltage	6
1.4	AES critical path delay	8
1.5	Slew and transition edges	11
2.1	Process Variability	18
2.2	Razor error detection	20
2.3	CRISTA path isolation	21
3.1	Path representation	25
3.2	Regular clock system	26
3.3	System overview	26
3.4	Monitor Scheme	27
3.5	Thermometer representation	27
3.6	Thermometer measuring delay	27
3.7	Simple Ring Oscillator	29
3.8	Dual Ring Oscillator	30
3.9	C-Element	30
3.10	Multiplexer of Ring Oscillators	31
3.11	Ring Oscillator with Multiplexers	32
3.12	Ring Oscillator with multiple paths selected	32
4.1	Additional Cells	40
4.2	Multiple configuration paths target delay	45
4.3	Multiplexer gate configuration	46
4.4	Equivalent slews in different paths	46
4.5	3-SAT Reduction - Gate construction	50
4.6	3-SAT Example	51
4.7	Variance cost function	62
4.8	Squared distance cost function	63
5.1	Squared normalized distance heuristic	68
5.2	Squared normalized distance heuristic long delays	68
5.3	Variance over the normalized delay heuristic	69
5.4	Variance over the normalized delay heuristic long delays	69
5.5	Mixed heuristic	70
5.6	Mixed heuristic long delays	70
5.7	Simple and Dual RO	71

5.8	Simple and Dual RO long delays	71
5.9	Monitor	72
5.10	Monitor for long paths	72
5.11	Configurable Simple RO	74
5.12	Configurable Simple RO long delays	74
5.13	Configurable Dual RO	74
5.14	Configurable Dual RO long delays	75
5.15	RO for an AES circuit	76
5.16	Error AES RO	77

Glossary

AES Advanced Encryption Standard.

ASCI Application-Specific Integrated Circuit.

CNF Conjunctive Normal Form.

DDRO Design-Dependent Ring Oscillator.

ILP Integer Linear Programming.

IWLS International Workshop on Logic & Synthesis.

NLDM Non Linear Delay Model.

NTBI Negative-Bias Temperature Instability.

OPC Optical Proximity Correction.

PLL Phase-Locked Loop.

RCP Representative Critical Path.

RO Ring Oscillator.

RTL Register-Transfer Language.

SMT Satisfiability Modulo Theories.

SPICE Simulation Program with Integrated Circuit Emphasis.

SSTA Statistical Static Timing Analysis.

STA Static Timing Analysis.

TSMC Taiwan Semiconductor Manufacturing Company.

UMC United Microelectronics Corporation.

Chapter 1

Introduction

1.1 Introduction

It hardly makes sense to argue against the profound implications that semiconductor industry has had in society and people's life over the last half century. Since the appearance of the first integrated circuits in the late 1950's, electronic devices of all kind have been developed and introduced in the market, taking advantage of the versatility that transistors provide. This versatility has been intensified more and more by the speed at which transistors and integrated circuits in general have been evolving.

In 1965, Gordon E. Moore presented what would become one of the most famous papers in computer Engineering [1]. He predicted a trend of doubling the number of components per chip each year thanks to a shrinking in component size. Although the speed at which the doubling occurs was later slowed down to 18 months, this trend was fulfilled like a prophecy. It is commonly considered to be the main enabler for the evolution that electronic and digital devices have underwent in the last decades. And not surprisingly; this exponential growth in the number of transistors has made possible going from single transistors to billion-transistors microprocessors. Figure 1.1 shows this trend applied to Intel and AMD processors, in which we can see how transistor count has been continuously increasing.

This prediction is nowadays known as Moore's law and is a topic of hot controversy today. Since the early days of semiconductors industry, the shrinking of transistors had several benefits for electronic circuits:

- Smaller dimensions enabled a higher integration density.

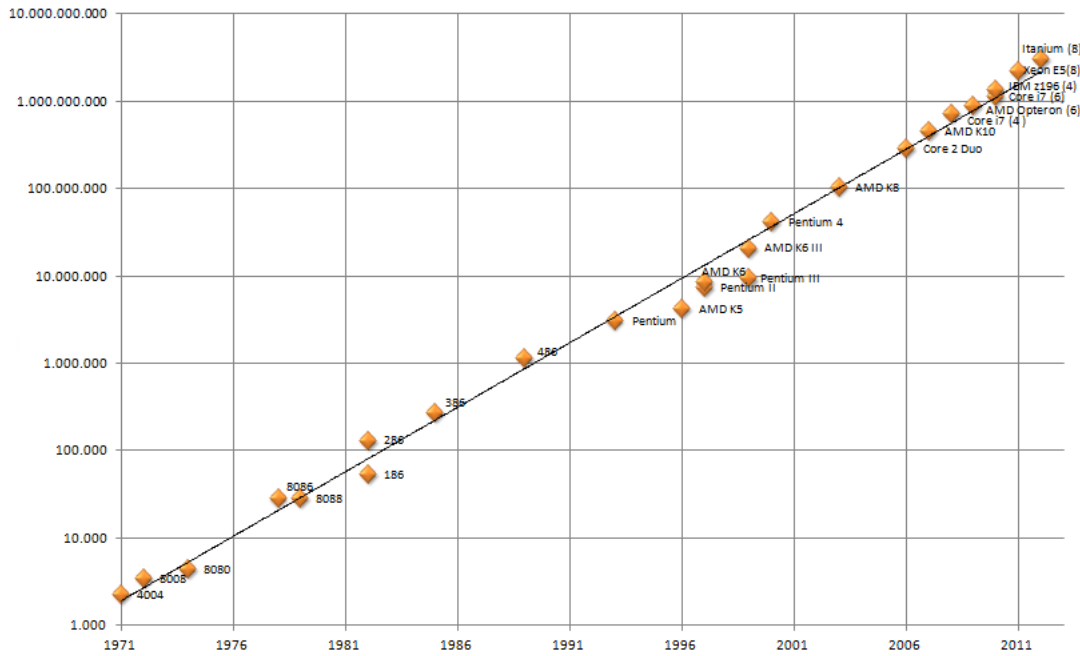


FIGURE 1.1: Transistor count evolution for Intel and AMD processors.

- Faster switching speed made possible to increase clock frequency, often doubling the previous frequencies.
- Lower voltage thresholds¹ increased the energy efficiency of transistors, making them only needing half the power.
- Increased reliability made circuits less likely to behave in unexpected ways.
- All of this without increasing the cost per area, meaning that the cost per transistor was halving at each new generation.

More importantly, all of these improvements occurred without virtually any tradeoff; a new generation of transistors was better than the previous one at almost every conceivable characteristic, and a new generation was appearing each 18 months.

But this exponential growth cannot continue eternally. If nothing else, there is a physical limit at which it is not possible to reduce the size of a transistor. Additionally, voltage fluctuations (noise) do not scale – in the last decade we ran into a limit in how much we could keep reducing voltage. Without voltage reductions, increasing the frequency of a transistor meant increasing its power consumption. Therefore, a trade off between speed and power appeared – we can have faster transistors or lower power, but not both. This put an end to the exponential growth for frequency, as increasing power consumption was causing the chips dice to literally melt. Moreover, current transistor

¹The voltage threshold of a transistor is the minimum voltage required to switch its logic state.

generations are more and more unreliable due to variability in fabrication process as well as environmental conditions once the chip is produced (which will be later discussed in depth), thus forcing the introduction of huge guard band margins in the design process.

On top of everything, it is now considered that the end for Moore's law will not really occur for physical reasons, but for economical ones. In the last generations, cost per transistor may have reversed its trend and is actually increasing for some manufacturers [2]. All of this is making some experts on the field believe that Moore's law will stop in the next few years [3].

1.2 Variability

One of the main reasons for the diminishing returns of transistor scaling in the last years is variability in the smallest technology nodes. In this section we introduce the most relevant variability sources.

One of the most common ways to categorize variability sources is by dividing them into two groups:

- Static variability: This variability accounts only for process variability. It is considered static because it does not change during the life of a chip.
- Dynamic variability: Sources of variability that change the properties of a chip in time. This category includes voltage, temperature and aging.

When talking about variability it is also important to distinguish between global and local variability.

- Global variations affect the whole chip or at least a major part. For example, the ambient temperature will affect equally all the chip, yet that does not mean that all the chip is at the same temperature.
- Local variations affect only a small part of the chip. The greater the distance between two points, the more unrelated local variations become. In the temperature example, there may be some components in the chip that become hotter due to switching activity, thus increasing the temperature in their vicinity.

This categorization of the locality of variations is important at design stage, as different parts of the chip may behave in different ways, but it becomes even more relevant when introducing some techniques that require monitoring variability. For example, knowing

the temperature in one part of the chip does not guarantee that the opposite side of the chip is subject to the same temperature. Below follows a detailed explanation for the main variability sources.

1.2.1 Process Variations

The chip manufacturing process is far from perfect. In the last technology nodes it has become more and more difficult to maintain homogeneity in the fabrication of chips, thus introducing differences in the characteristics of transistors. These variations from transistor to transistor in the manufacturing stage is what is known as *process variability*. As explained above, this type of variability is stable along the life of a chip and does not change with time.

Process variations come from a number of sources, the most important of which are enumerated below.

1.2.1.1 Critical dimensions

The manufacturing of a chip passes through a stage in which a source of ultraviolet light traverses a mask in order to “print” the layout. The wavelength of light used for this process has remained at $\lambda = 193$ nm since the 130 nm node. As we move towards smaller technology nodes, recently reaching 14 nm, this source of light is more and more unsuitable due to the diffraction phenomenon in which the waves are dispersed as they traverse openings smaller than their wavelength.

To cope with these limitations, some techniques are applied in order to correct the effects of diffraction. One of these techniques is optical proximity correction (OPC), which consists in using a *corrected* mask in order to force the light into projecting the desired layout. Figure 1.2 shows an example of this technique. On the top left is shown the intended layout which, if projected as is, will be deformed into the bottom left figure. In order to be able to print something similar to what we want, a correction algorithm is ran and the layout from top right is obtained. When this layout is projected, figure from bottom right is obtained. This last figure is very similar to what we needed, yet not perfectly identical to the indented shape.

These small variations with respect to what we originally intended, as well as deviations in both the creation of the corrected mask and its projection, affect in an unpredictable way the sizes of transistors and thus their properties, such as its drive current or voltage threshold.

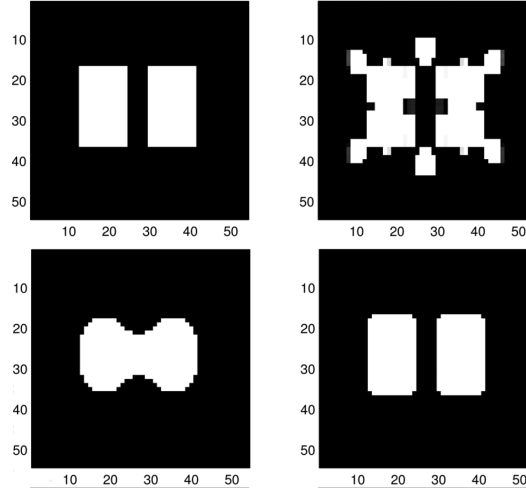


FIGURE 1.2: Mask layout correction from [4]. On the top left, the intended layout. Bottom left shows the printed layout if no correction is applied. Top right depicts the actual mask after correction, which results in the printed image at the bottom right.

1.2.1.2 Random dopant fluctuation

Another stage in the manufacturing process consists in doping the channel region of transistors with impurity atoms. These atoms are placed into the channel in order to alter its electrical properties and thus have a great impact in the threshold value. Older technologies required the random insertion of thousands of atoms, so deviations of several atoms had a negligible impact. In the latest nodes, with only a few tens of atoms, these deviations of several atoms are much more disruptive.

1.2.1.3 Gate Oxide Thickness

One of the parts that composes a transistor is the gate oxide. It separates the gate terminal from the underlying source and drain terminals as well as the conductive channel that connects source and drain when the transistor is switched on. This layer can be grown with an accuracy of 1-2 inter atomic layers [5]. In older technologies, with thickness in the order of tens of inter atomic layers, variations in the growth for the gate oxide were negligible, but in the most recent nodes the oxide thickness may be as small as 5 inter atomic spacings. The differences in the gate oxide thickness further contributes to the voltage threshold uncertainty.

1.2.2 Voltage fluctuations

Fluctuations in the voltage source are caused mainly by a combination of IR drops and di/dt noise. The former is caused by the current flow over the parasitic resistance of the

power grid and is a direct cause of Ohm's law ($V = i(t) \cdot R$, hence the name IR). The latter is a consequence of the parasitic inductance in combination with capacitance and resistance of the power grid, causing the voltage to “bounce”. When both effects are superposed, they may cause the voltage to not only drop, but to overshoot (going over the nominal voltage value). It is especially notorious that these effects are fast changing, having typically values of time in the range of nano to micro seconds.

The voltage variations in a gate or a path of gates has a great impact in their delay. As it can be seen by Figure 1.3, a voltage variation of just 33% may have an impact of 3x over the delay in a critical path.

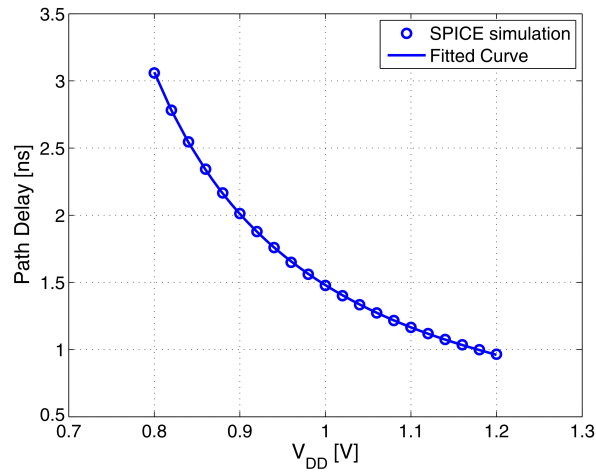


FIGURE 1.3: Relation between supply voltage and logic delay, illustrated by the most critical path of a multiplier circuit in 65 nm technology. Source from [5].

1.2.3 Temperature

Temperature on a chip fluctuates mainly due to power dissipation of its components. These changes in temperature may be global, due to ambient temperature or power dissipation, but it can also be local in regions of high-activity switching. Typically, temperature fluctuations have time constants in the range of milliseconds, being much slower than voltage fluctuations.

Increasing the temperature in the chip usually leads to slower delays due to reduced carrier mobility and increased interconnect resistance. In conditions of low voltage, this trend may be reverted when the voltage threshold decrease exceeds the mobility degradation. This phenomenon is known as temperature inversion.

1.2.4 Aging

Aging of a circuit is the wear out of the same along time. Aging becomes only significant over weeks or months of use. This makes it the variability source that changes the slowest with time. Aging occurs for a number of causes, such as *Hot Carrier Injection*, *Negative-Bias Temperature Instability* and *Electromigration* that we explain below.

- **Hot Carrier Injection:** This phenomenon mostly occurs during the switching of logic gates. The acceleration of carriers in the lateral field makes them gain enough energy to be injected into the gate dielectric. This effect changes the switching characteristics of the transistors.
- **Negative-Bias Temperature Instability:** Negative-Bias Temperature Instability (NTBI) is a phenomenon in the gate-oxide and occurs in negative gate voltages or in elevated temperatures. It has the effect of permanently changing the voltage threshold value of transistors over time.
- **Electromigration:** Electromigration is the transport of material caused by the gradual movement of ions in a conductor. This effect causes the degradation of wires and may cause the eventual failure of a circuit.

1.3 Living with variability

As we have seen, there are many sources of variability, each one affecting a given design in different ways. In order to have working circuits, it is necessary to account for all these variations in the characteristics of its components. For this reason, it has been historically a common practice to design for the worst case combination (usually referred to as worst-case sign off). This approach ensures that circuits work correctly in every condition with a low risk of failure.

Yet as mentioned before, newer technology nodes have more sources of variability and, more importantly, variability sources have a greater influence. This fact is making worst-case sign off increasingly expensive, to a point where it no longer makes sense to simply consider the worst possible combination. In addition, worst-case scenarios are extremely unlikely to occur in practical scenarios.

Figure 1.4 shows the delay for the critical paths of an AES (Advanced Encryption Standard) circuit. Each bar of the figure represents a combination of variability sources, called corners (explained in detail latter in Section 1.4.1), for a 65 nm process. The leftmost bar represents the worst-case combination, the bar labelled *typical* represents

the most frequent combination and the rightmost one the ideal case. As it can be seen, there is a huge gap of almost 3x between typical and worst case.

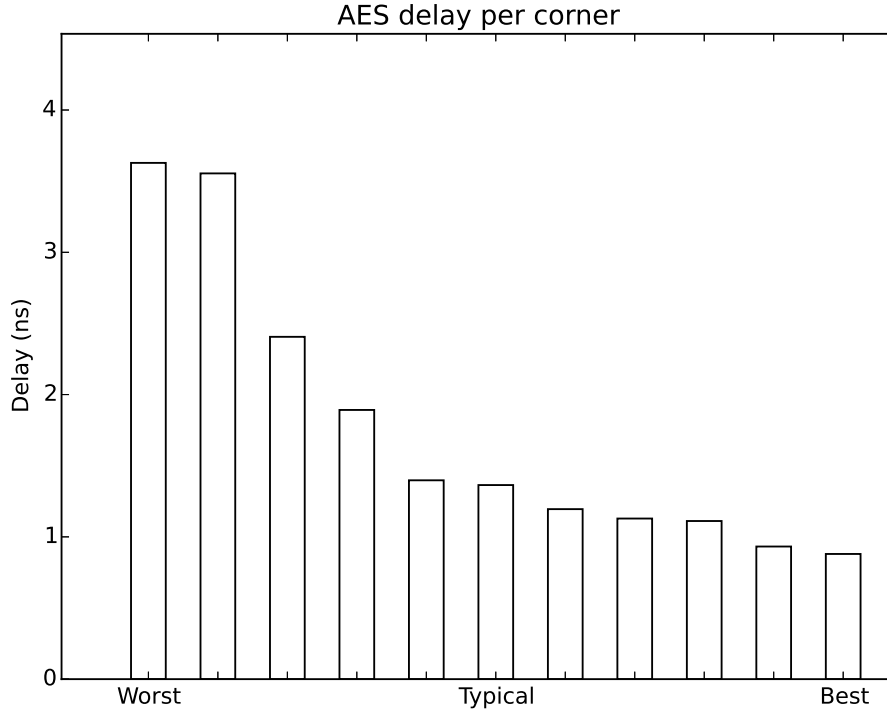


FIGURE 1.4: AES module critical path delay for corners. Delays were obtained for a 65 nm commercial library.

It is this growing difference between typical and worse conditions what justifies research in variability and, in particular, motivates this work.

1.4 Static Timing Analysis

The process of designing a circuit involves a number of stages. It usually starts with a requirement specification, not unlike in the software, and then continues with several stages until a functional design is obtained. This design, usually expressed in some RTL (Register-transfer language), describes the operation of the circuit in a way that can be synthesized and implemented into a physical circuit. But before this implementation can be done, it is important to perform a series of analysis and validations regarding its properties, such as logic functionality (does the logic design behave as expected?) or physical considerations (when manufactured, will it still behave as expected?).

One of the most important analysis that must be done is the timing analysis, in which we verify whether a circuit meets all of its timing requirements or constraints. These

constraints include, for example, checking if a certain path within a circuit has enough time to finish its computation before a clock pulse arrives.

There are two types of timing analysis:

- **Dynamic Timing Analysis:** The timing constraints are checked with a limited set of input test vectors. The behaviour of the circuit is simulated with high accuracy at the cost of high execution times. Additionally, it checks logical functionality. Not all possible scenarios are checked, as only a limited set of inputs can be analysed.
- **Static Timing Analysis:** Checks timing constraints for all possible inputs. Uses accurate models for timing, yet does not test functionality. While models are less accurate than simulation, runtimes are much lower and tests are more thorough, checking all possible *worst-case* scenarios.

Static Timing Analysis can be divided into 3 main steps:

- Design is divided into timing paths
- For each path, signal propagation delay is calculated
- Check for violations of timing constraints

As we saw in previous sections, variability gravely affects the delay in a circuit or, more specifically, in a path. One of the most common ways to take into account variability in STA is with the use of corners. The rest of this section will show how variability is included into delay computations and how delay is finally propagated along a path.

1.4.1 Corners

As we saw earlier, there are many sources of variability in a circuit, increasing in number and relevance as we move on to smaller technology nodes. These variability sources change the way a given cell behaves altering properties such as delay, capacity or slew². The corner paradigm tries to delimit the properties of a cell by setting limits on the values of the variability sources. These limits that enclose the way cells can behave is what we call corners.

Before going into a more accurate definition, it is important to first enumerate the sources of variability modelled. In this work we consider PVT/RC corners (Process, Voltage, Temperature / Resistance, Capacitance):

²The slew, or transition time, is the measure of the time it takes for a signal to go from a logic 0 to a logic 1 (rise transition) or from a logic 1 to a logic 0 (fall transition).

- **Process shifts:** Due to the fabrication process of a chip, transistors and wires may have different properties than expected. For example, a transistor in a certain chip may be faster or slower than the same transistor in a different chip, or a wire may be wider than expected, causing it to have more capacitance and less resistance. Moreover, chips are constructed by layers, and different types of transistors (PNP and NPN transistors) may have different properties. The process is thus divided into two categories:
 - **Transistors:** The model for process for transistors can be Fast, Typical or Slow. Due to the construction process in layers, cells containing both types of transistors may be modelled as Fast-Fast (FF), Typical-Typical (TT), Slow-Slow (SS) or any possible mixed combination of the three such as FS or SF.
 - **Interconnects:** The fabrication process can also alter the properties of interconnect elements such as wires and vias. In the case of interconnects, we consider models with maximum and minimum values for resistance and capacitance, providing four extreme RC (Resistance-Capacitance) corners plus one typical corner (RC-typical) for the wires and for the vias. Note that wires and vias are independent, so it is possible that one has RC minimum and the other one maximum.
- **Voltage:** A circuit may work with different voltage sources, caused either by controlled fluctuations (such as reducing the voltage to save power) or by uncontrolled variations (such as voltage droops or noise in general). The voltage is usually modelled by maximum, typical and minimum values.
- **Temperature:** The temperature at which the chip is working greatly affects the way it behaves. It is usually defined by maximum, typical and minimum.

Note that, while some of the variability sources may have small correlations (for example, higher voltage usually produces higher temperatures), nothing prevents for any single combination to happen in the correct circumstances (for example, it is possible to have a chip working at the maximum voltage with the minimum temperature).

All these variability sources define the PVT/RC space, whose limits are defined by the combination of extreme situations. For example, a point in a vertex of the PVT/RC space is a process with FS transistors, wires and vias with maximum capacity and minimum resistance, working at maximum voltage and minimum temperature. It was those combinations of extreme values that were at first defined as corners in the PVT/RC space, when the number of variability sources was much smaller. However, the idea behind this characterization is to find the extreme values for the delay (and possibly

nominal ones) for the cells and nets. For this reason, only some of these combinations are useful. A more accurate and current definition, which can be found in [6], is:

Corner is a point in the PVT/RC space where the cell/net delays have extreme (and optionally nominal) values - all cell delays are the maximum or the minimum and all net delays are the maximum or minimum independently.

It must be noted that obtaining these corners is not trivial, but it is outside the scope of this work to describe this process. It suffices to say that each foundry generates a set of libraries of cells for each technology node containing all this information that allows corner-based signoff.

1.4.2 Delay computation and propagation

In this section we will explain how to compute the delay of a net based on commercial corner-based libraries. But first, let's define what we meant with delay: in digital circuits, propagation delay or gate delay is the lapse of time that starts when the input to a logic gate becomes stable and finishes when the output of that logic gate is stable. Similarly, we may talk about delay of a path of gates by considering the inputs and outputs of the path.

Also important concepts that we will use are the slew, or transition time, and transition edges, represented by Figure 1.5. The slew represents the lapse of time that a signal needs for switching logic state. This transition of logic states may be a rising transition if the signal switches from 0 to 1 or a falling transition if it switches from 1 to 0.

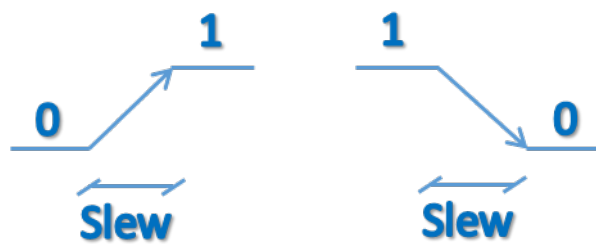


FIGURE 1.5: Slew, or transition time, and transition edges. On the left, a rising edge. On the right, a falling edge.

Before starting with the computation of delay, it is convenient to describe what information relevant to our problem is contained in the libraries. A library contains a set of corners and, for each corner, there is an exhaustive list of all available cells. For each cell, besides other data, there is the following information:

- Basic information: Such as name, area or footprint.

- Pins: An exhaustive list of pins both input and output.
- Capacitance: Maximum and minimum values for the capacitance of each of the input pins of the cell. Also, the capacitance of a cell depends on whether the input signal is a rising or a falling edge.
- Delay tables: For each pin, for each possible assignment of inputs on the pins and for both transition options (rise and fall), there exist a table with output capacitance and input slew (or transition time) which combination indexes a wide range of delays. This is used to compute the delay and will be explained in more detail latter on this section.
- Transition tables: For each delay table there is an additional transition table used to compute the output slew. Again, we will see a more detailed explanation on this latter.
- Unateness: For each pin and assignment of signals for the rest of pins, the libraries may optionally contain the unateness that can be either positive unate³ or negative unate⁴. Note that, while a cell may implement a binate function⁵, by fixing the assignment of signals we enforce unateness.
- Function: Each output pin contains the function that implements (with respect to the input pins).

Besides that information, each cell has other important information such as power or leakage values that are not relevant to our work at this stage. Also, the library contains models for wires (called *Wire Models*) that allows for a rough estimation of its capacitance and resistance before the routing stage of the signoff. This usually gives estimate values depending on the fan-out of the cells and are only useful for a first notion of the actual values.

As specified above, the delay and slew of a cell are defined by a table whose inputs are slew and capacitance. This table contains several discrete values for slew and capacitance, but needs to be accessed for any continuous value, so some sort of interpolation is needed. In our work, we are using the NLDM (Non Linear Delay Model) to interpolate the delay, as this is a common approach, which consists simply in a bilinear interpolation.

³A positive unate function indicates that a rise transition in the input can only cause a rise transition in the output and, symmetrically, a fall transition in the input causes exclusively a fall transition in the output.

⁴The opposite of positive unate, a negative unate function indicates that the output transition is the inverse of the input transition (i.e. a rise transition in the input causes a fall transition in the output and, conversely, a fall transition in the input causes a rise transition in the output).

⁵A binate function indicates that the behaviour of transitions at the output with respect to some input depends on the value of other inputs.

As an example of how to compute delay by the NLDM model consider Table 1.1. In the leftmost column are the indices for capacitance and the topmost row contains indices for slew. In this case, table values represent delays but transition time tables are identical otherwise. Let's suppose that we are interested in the delay for an input transition time of 0.15 ns and an output capacitance of 1.16 pF. Note that Table 1.1 has removed all values not relevant for this example and only shows the indices that contain our capacitance and slew values (i.e. 0.15 is contained within the range 0.1-0.3).

	...	0.1	0.3	...
...
0.35	...	0.1937	0.2327	...
1.43	...	0.7280	0.7676	...
...

TABLE 1.1: An example of a NLDM table for delay.

We will call x_1 and x_2 the index values for slew (0.1 and 0.3 in this example) and y_1 and y_2 the index values for capacitance (0.35 and 1.43 in this example). The corresponding table values will be denoted as T_{11} , T_{12} , T_{21} and T_{22} respectively. If the table lookup is required for (x_0, y_0) (0.15 and 1.16 in this example), the lookup value T_{00} obtained by interpolation is given by:

$$T_{00} = x_{20} \times y_{20} \times T_{11} + x_{20} \times y_{01} \times T_{12} + x_{01} \times y_{20} \times T_{21} + x_{01} \times y_{01} \times T_{22}$$

where

$$\begin{aligned} x_{01} &= (x_0 - x_1)/(x_2 - x_1) \\ x_{20} &= (x_2 - x_0)/(x_2 - x_1) \\ y_{01} &= (y_0 - y_1)/(y_2 - y_1) \\ y_{20} &= (y_2 - y_0)/(y_2 - y_1) \end{aligned}$$

By making the appropriate substitutions, the delay we are looking for is computed as:

$$T_{00} = 0.75 \times 0.25 \times 0.1937 + 0.75 \times 0.75 \times 0.7280 + 0.25 \times 0.25 \times 0.2327 + 0.25 \times 0.75 \times 0.7676 = 0.6043$$

By knowing the delay of all the cells in a netlist, we can easily compute the delay of a path (i.e. a sequence of interconnected cells) by simply adding the values. The problem remains, though, on how to find the input values for the delay tables, namely the capacitance and the slew.

The input slew depends on the output slew of the previous cell, which is computed in a similar way to the delay by using tables. We also need to know the direction of the edge depending on the function implemented by the previous cell. In our particular case in which we are only interested in a simple path of cells, this function can be simplified by looking only at the unateness of the pins connected. Finally, the capacitance on the output of the cell depends on the capacitance of the cells connected to its output pin. It suffices with adding the capacitance of the input pins connected to the fanout of the cell (note that the capacitance also depends on the edge direction). If we are using a wire model, we also need to take into account the increase in capacitance or, if we have routing values for the wires, we must add the wire capacitance to the cells capacitance.

All these computations will yield the delay of a path for a given corner. This must be repeated with each corner of the library and so we will obtain as many delays as corners for the path.

It is important to understand that the delay of a cell within a path of cells is then dependent on the previous cells and on the following cells. Furthermore, it also depends on the value of the pins not connected to the path as they can alter which table must be used for delay and slew and they can even modify the unateness of the function. This interdependency will become relevant later when the problem is defined, as it is the main cause for complexity in the problem.

1.5 Objectives

The main objective of this project is, in short, to propose a scheme that takes variability into account in such a way that margins can be reduced. There is significant previous research on this topic, some of which is presented in Chapter 2, that usually tries to avoid variability by limiting it (such as *parametric binning*) or adapting to it (for example, *adaptive clocks*). Our proposal is to *embrace* variability in such a way that the system behaves differently when subject to different conditions, yet still does so correctly.

By analysing variability, we can see that the problem is not so much that characteristics for components in a circuit may change from one moment to another as much as that they do so in an uncorrelated way. For example, a voltage drop will make the circuit work slower, yet a common clock is virtually agnostic to variability and so the requirements for speed remain mostly unchanged. This way, a voltage drop that slows the circuit but does not change the clock period may cause a timing violation, resulting in a possible failure that can only be avoided by adding margins to the period. Yet still, when working

in regular conditions, these additional margins are unnecessary and potential speed and energy is lost.

For that reason, we propose a different clocking scheme that is as susceptible to variability as the rest of the circuit, in such a way that its timing characteristics always match that of the system. In the previous example, when a voltage drop slows down the circuit, the clock would also slow down thus increasing the period and avoiding a timing violation *without* the need for margins. In order for this to work even for fast voltage variations, which may change in the order of nanoseconds, this adaptation must be instantaneous.

1.5.1 Ring Oscillator as a clock substitutive

A Ring Oscillator (RO) is a sequence of gates in a feedback loop with an odd number of inversions. For example, 5 inverter gates connected one after the other with the last gate also connected to the first one will form a RO. The properties of such a circuit are that it *oscillates* periodically, much like a PLL⁶ (Phase-Locked Loop), yet it may be constructed with the same gates and transistors that constitute the rest of the circuit. By using the same components, the RO behaves similarly to the circuit in the presence of variability.

In order to accomplish it, the RO must at least have the following two properties:

- Its variability must be closely related to that of the critical path. Not all gates in the circuit behave the same way with different variation sources, so the RO must be constructed in such a way that it always matches the critical path in the design.
- The period of the RO must never fall below the timing requirements of the circuit. While the period should be as small as possible, it must never be so small that it makes the circuit fail.

Constructing a Ring Oscillator with those properties is not a trivial matter, yet any path generated must follow them. This proposes an interesting problem: the second property may be directly converted into a constraint, while the first property poses an optimization goal. The main contribution of this work is the generation of paths with those two properties. This will be done by presenting schemes for constructing such RO, but also proposing an algorithm to solve this optimization problem. Finally, experimental tests will be conducted and results will be discussed.

⁶A PLL is a control system whose output signal phase is related to the input signal phase. These systems generate a periodic signal and are commonly used as a clock for circuits.

1.6 Organization

The rest of this thesis is organized as follows. In Chapter 2 there is a discussion about the state of the art and related work for systems that, similarly to ours, try to reduce variability margins. This is followed by Chapter 3 in which our schemes will be introduced and explained in detail. Chapter 4 will first formalize the problem that we intend to solve and then propose a series of algorithms to solve the different design schemes presented in Chapter 3. Chapter 5 shows results from our implementation of the algorithms for a benchmark of several circuits. Finally, we present in Chapter 6 our conclusions, as well as expose some of the future work about this topic that we intend to conduct.

Chapter 2

State of the art

2.1 Introduction

As mentioned in the previous chapter, our main objective is to avoid the large margins used nowadays in worst case signoff. But we are far from being alone in this quest. In fact, there is a significant amount of work done and papers published on this topic with different approaches, some of which are being actively used in industry.

In this chapter we will introduce some of those works that aim at a similar objective. We will focus on the most relevant for either their extensive application (such as parametric binning) or their similarity with our own work.

2.2 Parametric binning

Parametric binning is the oldest and most widely used of the technologies that we introduce in this document, but it can only address static variability.

This technique is highly related to *yield*, which can be briefly described as the probability of a certain design to meet the target delay. More precisely, the yield depends on the design, the foundry and the technology and it represents the proportion of devices in the wafer that are considered to perform properly.

For example, we may have a design targeted to work at 1 GHz of frequency but, due to process variability, some of the dies may not work at that speed without incurring in timing violations.

Parametric binning tries to *classify* the dies in groups or *bins* after they are manufactured depending on certain parameters such as frequency. This may be taken advantage of

while designing by not considering the worst case in process but aiming instead to the expected or *typical* process values. After the fabrication, dice will undergo intensive testing to determine whether the timing constraints are met, discarding those that do not. In summary, parametric binning trades static margins in the design stage for a reduction in yield in the fabrication stage.

To improve profits, manufacturers often include a greater number of bins rather than just accepting or discarding dice. This may be done by classifying each die into different parameter values and then pricing and marketing them accordingly. For the example of the design targeted at 1 GHz, some of the dice may correctly work at 0.8 and 0.9 GHz and thus can be sold at lower prices. There may also be dice that met delay constraints at 1.1 and 1.2 GHz and that can then be sold at higher prices as premium products. The way that chips are classified will ultimately depend on the intended use and market in which they will be sold. Figure 2.1 shows an example of different dice with respect to frequency and power leakage.

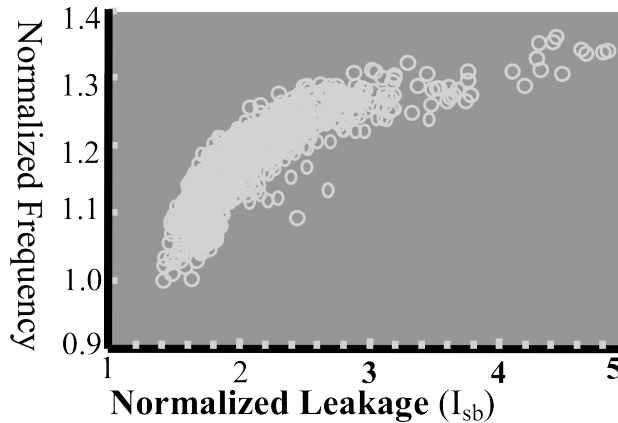


FIGURE 2.1: Process variations for dies in 130 nm. Each bubble represents a single die. Source from Intel.

This classification into *bins* may be done for two parameters:

- **Speed binning:** Speed binning [7, 8] classifies the dice with respect to its frequency. A die that does not meet timing constraints at a certain frequency *bin* is placed in a less restrictive *bin*.
- **Voltage binning:** In voltage binning [9] the classification is done with respect to the voltage at which the die has to work in order to meet delay constraints. Increasing the voltage in a circuit reduces the delay of its gates at the cost of an increased power consumption, allowing for dice that would fail at the target frequency to work correctly. The acceptance or rejection of dice and, possibly, its pricing is then done accordingly to power consumption.

2.3 Razor

A different technique that also addresses dynamic variability is *Razor* [10]. This technique is based on the idea of dynamically tuning the supply voltage of the circuit while monitoring the error rate. The voltage is lowered until the error rate reaches certain thresholds and increased when the number of errors grows due to variability sources. In order to guarantee valid execution, error correcting circuits are also included so that they may recover the data at a (small) time and power cost.

In essence, *Razor* tries to make the circuit work at the lowest possible voltage so that the circuit consumes as little power as possible (note that voltage has a quadratic influence over the power). This can also be potentially used for increasing the frequency of a circuit, while maintaining the power consumption with respect to a circuit without *Razor*.

With this idea, we can identify three parts on the *Razor* design:

- Error detection: The main circuit addition of *Razor* is the inclusion of *shadow latches*: each delay-critical flip-flop (such as the ones that are at the end of a critical path) is augmented with a *shadow latch* controlled by a delayed clock. This clock must be delayed enough so that it will always capture the correct value of the signal even in low voltage operations.

The error detection may be done by comparing the output of both the flip-flop and the *shadow latch*, but in timing violations in flip-flops there is always the possibility of incurring in metastability¹ issues. For that reason, a metastability detector is also included. Figure 2.2 depicts a simplification of the error detection hardware.

- Error correction: The error correction is already partly done in the error detection stage, as the correct value of a given stage is always in the *shadow latch*. The problem remains, though, of not propagating the faulty signal through the pipeline. There are different ways of doing this, the simplest one being stalling the pipeline one cycle whenever one error is detected by using clock gating. In the next cycle, the flip-flop will capture and propagate the correct value. Any number of errors can happen at the same time and the execution will always continue regardless of the error rate, never executing slower than half the speed.

¹A digital circuit may stay for an unbounded amount of time, yet with an exponentially decreasing probability over time, in a metastable state. In this state, the circuit is unable to settle into a logic 1 or 0 and acts in unpredictable ways.

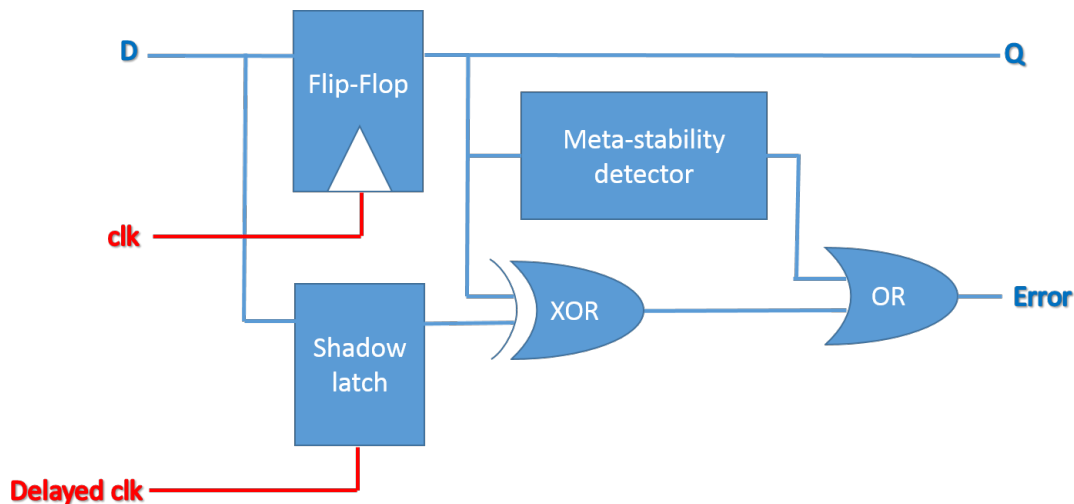


FIGURE 2.2: Razor error detection scheme. The regular input D feeds a common Flip-Flop and the shadow latch. The output from the Flip-Flop is compared with the output from the shadow latch and if different or if a metastable state is detected the *Error* output is fired.

- Supply voltage control: The voltage control is configured in a way that it tries to maintain a certain error rate. A low error rate indicates that the computation is finishing too quickly and that there is an opportunity for energy savings, so the voltage is lowered. When the error rate grows over an error rate reference, the voltage is increased to avoid losing performance or power due to recovery overhead.

Since the apparition of *Razor*, multiple optimizations have been presented, such as in [11], which proposes ways to better deal with metastability issues.

2.4 CRISTA and Trifecta

A different approach for dealing with dynamic variability is the one implemented by *CRISTA* [12] (short for CRitical path ISolation for Timing Adaptiveness) and *Trifecta* [13]. The basic ideas of these works are:

- Identify the set of paths that may cause timing violations in presence of variability (critical paths).
- In synthesis stage, it must be ensured that they are rarely activated.
- Those critical paths are given an extra cycle whenever they are activated.

This identification of critical paths is called *isolation* by [12], and it must be possible to predict their activation as a function of the input. In this way, the delay on these

paths may be much less restrained than regular paths, as they will have twice the time to complete. In fact, this is called *path isolation* due to the difference in delay with regular paths, as can be seen in Figure 2.3. The *isolation* of paths may be done by using a number of synthesis techniques, such as partitioning or gate sizing.

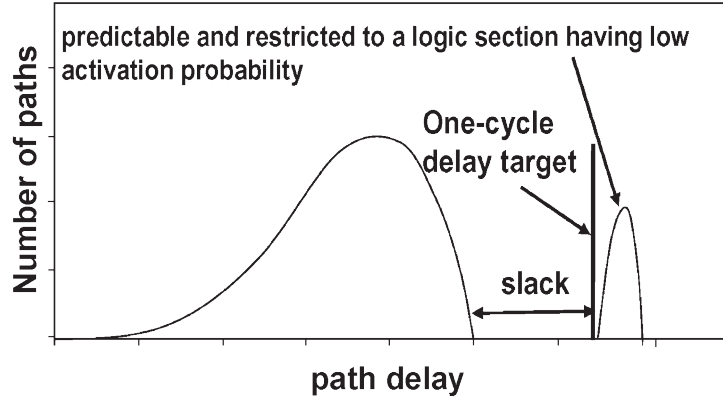


FIGURE 2.3: Delay distribution of paths from [12]. Critical paths have a clearly higher delay, differentiating them (*isolation*) from the regular paths.

Of course, these approaches only make sense if it can be ensured that the critical paths are only exercised in few circumstances. In those cases, a simple circuit dynamically identifies their activation and sends a signal to stall the pipeline for one cycle.

Additionally in [12] it is also proposed the use of a *Temperature-Adaptive Pipeline Design*. This technique tries to deal with the variability caused by the temperature in a circuit by including three different voltage supply sources that must be tuned for working at different temperature ranges. A simple sensor will then determine which of the three voltage supply units to use in function of the temperature at which the circuit is working and will switch between them as needed.

2.5 Performance Monitors

While not so directly correlated with reducing variability margins as the previous techniques, Performance Monitors aim at quantifying those margins in runtime. There are numerous articles on this topic proposing different implementations, only a few of which will be introduced here, but it is important to mention them for two reasons:

- They often use Ring Oscillators for implementing monitors. While their use has a different objective than ours, their designs share several similarities.
- Performance monitors are also an important part of our work. As shown later in Chapter 3, we use Monitors to check margins.

One way of classifying these Monitors is by considering whether they are designed for generic circuits or for a particular circuit design. The former may consist of simple inverter-based RO or more complex process-sensitive ones. In our case, we are more interested about design-dependent monitors, as they share more similarities with our work.

This last type may be further classified by the way they are designed. In [14] they propose the generation of a single representative critical path (RCP) designed for being slower than all the critical paths of the circuit. While this approach is similar to our own, the RCP is designed with statistical static timing analysis methods (SSTA) [15] and it only takes into account process variability without addressing dynamic variability.

Another interesting approach introduced by [16] consists on synthesizing 5 different delay paths that can be monitored either individually or combined, each path with different types of gates. The monitor includes a 12-bit thermometer² for measuring the delay that works in a very similar way to the one we propose in Chapter 3. With the help of the thermometer for measuring time margins, the delay path or combination of paths can be configured to approximate the actual critical path.

A different set of techniques estimate performance by placing *in situ* monitors [17, 18]. These monitors measure delay directly at critical paths, gaining in accuracy but resulting in a high area overhead. It is possible to reduce that overhead by selecting only a representative set of nets or even intermediate nets along critical paths as proposed by [18]. The problem remains, though, that *in situ* monitors may interfere with the actual critical paths thus increasing the complexity in the design stage.

The last type of monitors that we will be introducing are the Design-Dependent Ring Oscillators (DDROs) [19]. In their paper, they propose to run cluster algorithms over all the critical paths and design a RO for the centroid of each cluster. Note that the number of clusters may be as small as 3 or 4, so that the area for ROs remains small. Their technique also allows for both post-silicon tuning and real time performance monitoring.

2.6 Adaptive Clocks

Adaptive Clocks refer to a wide range of work and research in schemes that dynamically change the frequency of the clock [20–25]. In summary, they monitor the current conditions on the circuit and *adapt* the clock period accordingly. There are numerous schemes that differ in some way or another between themselves, making it hard to mention all

²A thermometer is introduced and explained later in Chapter 3.

the possibilities. Nonetheless, they share similarities and there is a set of characteristics that are often present:

- The set of frequencies to which the clock can be modulated is discrete.
- Voltage droops are usually the main focus and, often, the only source of variability that is alleviated, not taking into account, for example, temperature.
- Voltage droops may occur in the order of nanoseconds (high frequency voltage droops), yet adapting the clock is not instantaneous. Every scheme has its own way to deal with this problem, but it often requires the introduction of further margins.
- In order to know when to adapt the clock, it is common to use monitors such as the ones presented in Section 2.5.

Some of the Adaptive Clock schemes have been successfully used in commercial designs, such as the ones introduced in [21] or in [25].

Chapter 3

Ring Oscillator schemes

3.1 Introduction

The work in this thesis focuses on the design of a novel clocking scheme that adapts to variability in order to avoid margins in design stage. More specifically, we use a Ring Oscillator as substitutive for the PLL that is designed for tracking variability for critical paths in the system. Additionally, we will also include a *monitor* circuit, similar to the ones introduced by Section 2.5. This monitor, while not strictly necessary, will be convenient for a possible accuracy tuning as well as margin checks. It is especially interesting when used in conjunction with configurable ROs that are explained later in this chapter.

Along this chapter, we will talk about paths of gates. In order to simplify representation, we will use the convention shown in Figure 3.1. Any path of arbitrary gates, such as the topmost picture of Figure 3.1, will be depicted in the rest of this document as either a sequence of triangles or an elongated bubble. Note that, while they represent a path of gates, we are only interested in the delay they cause, rather than the gates themselves.

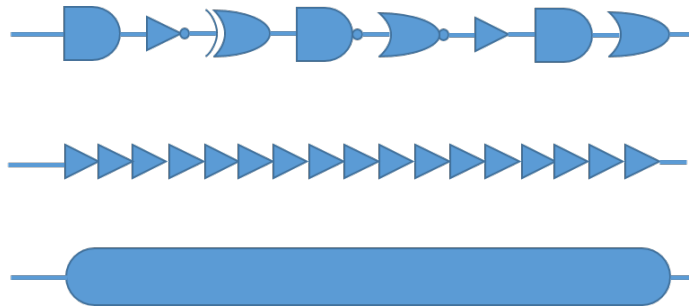


FIGURE 3.1: Topmost figure shows an arbitrary path of gates. This will be represented by the mid and bottom figures in later pictures.

As an overview of the system, consider figures 3.2 and 3.3. In Figure 3.2 we may see a regular clocking scheme, in which a periodic signal is feed to the Flip-Flops in the system. Note that the only changes done by a RO system is the substitution of this clock for a RO that fulfils the exact same role. A monitor circuit may also be present in the system.

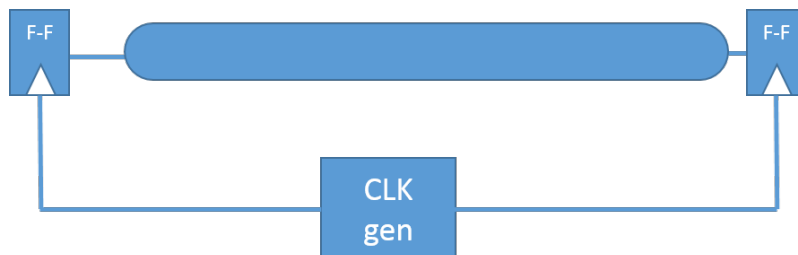


FIGURE 3.2: A regular clocking scheme. The clock generates a periodic signal that feeds to the Flip-Flops in the system.

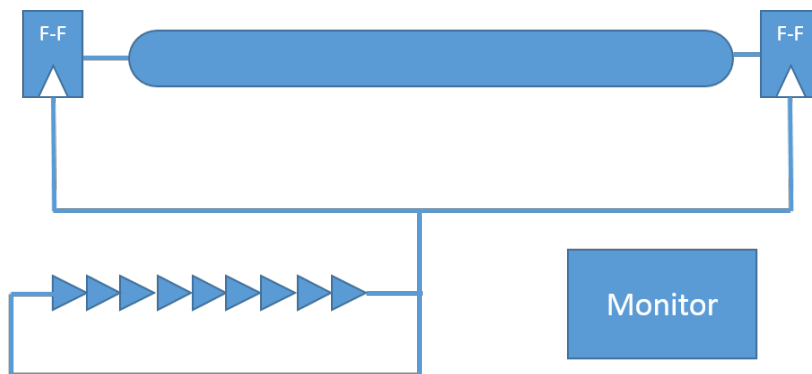


FIGURE 3.3: Simplified overview of a RO system. The clock is substituted by a RO, yet the signal is still directed towards the Flip-Flops. Monitor circuits are physically close to the paths they check.

The following sections will go into detail as to how the RO and monitors are designed. We will also talk about different schemes for RO and discuss their advantages.

3.2 Monitor

The Monitor is a circuit that keeps track of the variability in the system. This circuit shares similarities with the RO in that it is a path built with regular gates subject to the same sources of variability as the rest of the chip. The difference is that it does not need to generate a periodic signal but it needs to somehow measure what is the current delay of the path.

Figure 3.4 shows a basic scheme of the two main parts of the Monitor. It is first composed by a path of gates that gives a delay subject to variability, followed by a component called

thermometer. This thermometer measures how far has the signal travelled through the path before the clock signal arrives, being just composed by a sequence of gates with Flip-Flops interleaved at regular intervals, as can be seen by Figure 3.5.

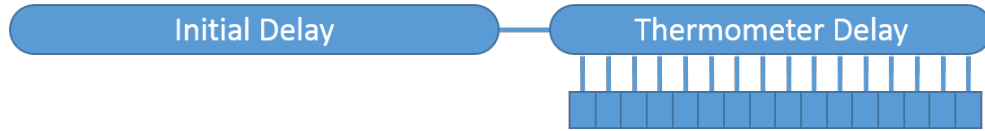


FIGURE 3.4: Basic scheme of a Monitor. There is an initial delay, followed by the Thermometer section which is used to measure the number of flip-flops reached by the signal before the clock edge arrives.

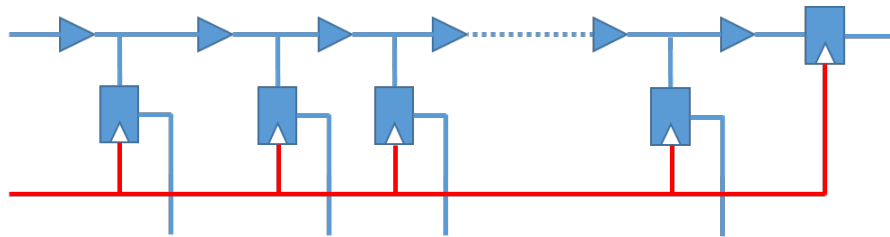


FIGURE 3.5: Scheme of a Monitor Thermometer. The blue triangles represent any possible gate or sequence of gates, the blue squares are flip-flops and the red signal indicates the clock.

The way this circuit works is simple: at the start of the clock period, a signal pulse is generated from the initial Flip-Flop and travels along the path, storing a value in each of the thermometer Flip-Flops it passes through. The thermometer is also connected to the clock, so when the next clock signal arrives, the Flip-Flops that had a value in its input will now have it in its output, while the ones that were not reached in time by the signal maintain their previous value. This can be read by a simple circuit showing how far the signal has travelled. Figure 3.6 shows a representation of this functionality.

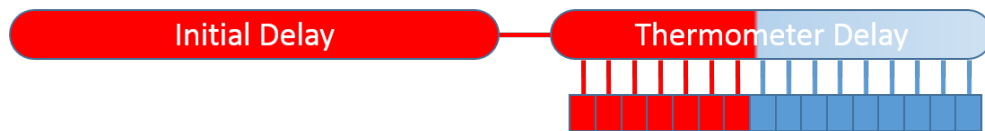


FIGURE 3.6: Representation of a monitor measuring the delay before the clock signal arrives. In red is shown the reach of the input signal when the clock edge activates the flip-flops.

The Monitor has some restrictions and there are some desirable characteristics that it should maintain:

- The length of the initial delay should be configurable (at design stage), so that it may be adaptable to each system.
- The total length of the path, including the thermometer, must always be greater, for each corner, to the critical path delay of the circuit that we are monitoring.

- The number of Flip-Flops in the thermometer needs to be a configurable parameter in design stage.
- The slew through the whole monitor should be relatively small, as bigger slews render the path more susceptible to variability and this could pose a problem.

3.3 Ring Oscillator

The Ring Oscillator is the most important circuit that we are designing, but also the most difficult one. As basic properties, it needs to generate a periodic signal that cannot be faster than the critical path at each corner. For simplicity, we will refer from now on to the timing restrictions as *target delay*.

Because of the periodicity of the signal, the RO circuit must be connected in a loop. A simple example of RO is an *odd* number of NOT gates in sequence, with the last gate connected to the first gate. It is important to note that there must be an odd number of inversions, as one traversal of the signal through the path will generate the positive edge and the next will generate the negative edge. This also means that the delay for both negative and positive signals must be greater than *half* the target delay.

Regardless of that simple example, the possible gates are not restricted to inverters and the scheme for a RO does not need to be that simple either. Latter on in this section we will discuss the two schemes that we are considering for this work, but before going into details, let us enumerate the main characteristics that an RO should have:

- It must provide a periodic signal, yet the period does not need to be constant. There may be, and in fact it is desirable, some jitter¹ as the period adapts to the variability sources.
- For each corner, the delay of the RO must be greater than the delay of the critical path for that corner.
- While maintaining the previous restriction, it should have a delay as small as possible.
- It is desirable that the slew is contained within a certain interval for avoiding an excess of sensitivity to variability.
- Optionally, it is interesting that the RO can be dynamically configurable, so that it can change the length of the path. This is useful when used in conjunction with

¹Jitter is the deviation from true periodicity of a presumed periodic signal.

the Monitor to allow for a fine grain tweak once the chip is fabricated permitting the delay to be reduced for performance or increased for safety.

During the rest of this section we will explain the two different oscillator schemes that we are considering in this work, as well as a way to deal with configurable paths.

3.3.1 Simple Ring Oscillator

The simple RO is the most straightforward strategy and it is similar to the example posed at the beginning of this section. As can be seen in Figure 3.7, this scheme is just composed by a sequence of gates (any normal gate) that invert the signal as represented by the small ball (this can be done, as stated before, with any odd number of inversions) and are connected in a feedback loop.

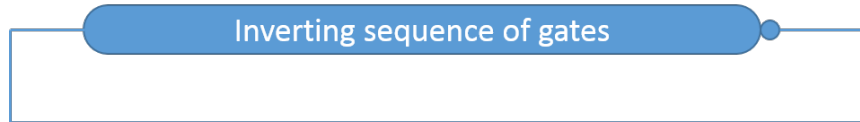


FIGURE 3.7: A simple Ring Oscillator connected in a loop. The signal is inverted at each iteration.

It is important to mention at this point that we are interested in a symmetric duty cycle – the time of the signal at level 0 is equal to the one at level 1. This kind of clock signal is very common and is the one we will focus on.

Note that, at any one period, the signal will traverse each cell twice; once as a rising edge and once as a falling edge. Now remember that the delay of a cell depends on the edge direction, so the effective delay of a cell will not be constant through a period. If this were not the case, we would only need to build a path that has a delay half of the target delay, but considering a real model greatly increases the difficulty of the problem.

When designing the path, the selection of gates will have to be such that the rising edge takes the same time than the falling edge to traverse it. This does not necessarily imply that each gate must have the same delay for rising and falling edges, as one cell can compensate for the next one. In any case, this is the biggest disadvantage of this approach, as it increases the complexity of the optimization problem to solve and, potentially, reduces the quality of the solution (i.e. greater margins).

3.3.2 Dual Ring Oscillator

This scheme for the RO aims to mitigate the problem of optimizing for two edges at the same time. It is based in the idea of making one path optimized for the rising edge and one path optimized for the falling edge.

Figure 3.8 depicts a general view of this scheme. It can be seen how it has two alternate paths joined together by a special cell named *C-element*. This C-element, depicted in Figure 3.9, is a sequential gate that only propagates the signal when the first edge arrives from any of the two paths. This means that, if the signal from path *A* arrives before the signal from path *B*, the C-element will propagate that signal and feed it back to both paths.

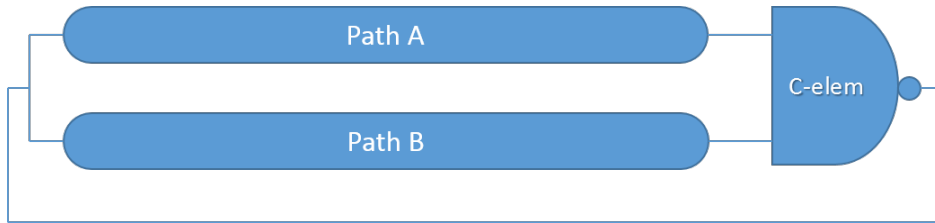


FIGURE 3.8: A Dual Ring Oscillator depicting two possible paths. One of them is optimized for the rising edge and the other one for the falling edge. As with the simple RO, the signal is inverted once per iteration.

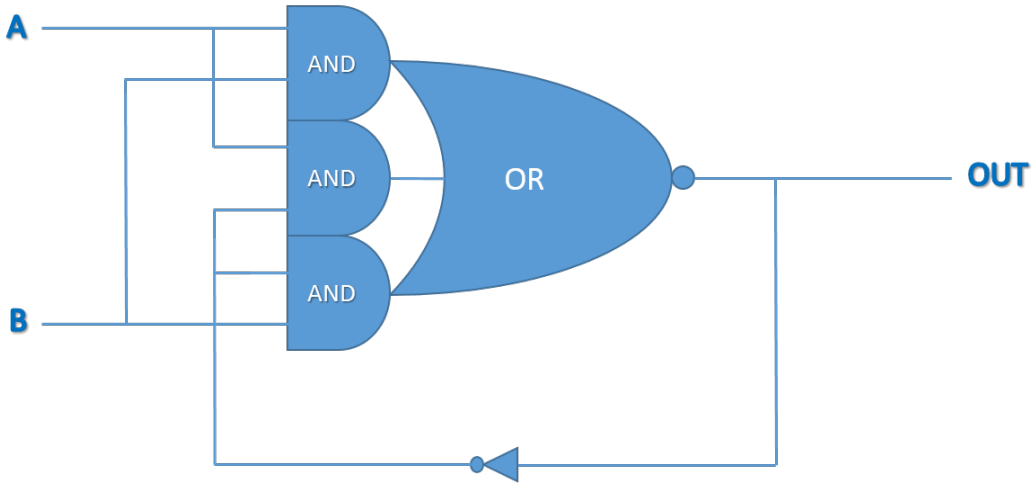


FIGURE 3.9: A possible implementation for a negated C-Element with standard gates. It has three AND gates and one NOR gate connected as shown, with two inputs and one output.

It should be noted that, while this allows for independently optimizing each path for one edge, it is still necessary to maintain the condition that the non-optimized edge must be slower (i.e. have higher delay) than the optimized edge from the opposite path. That means that, if path *A* is optimized for rising edges, the falling edge in *A* must be

slower than in B . While this may sound a harsh condition, it is in fact more lax than the condition from the simple oscillator scheme.

3.3.3 Configurable Ring Oscillator

As mentioned before, one of the optional yet desirable characteristics for a Ring Oscillator is the capacity for reconfiguration in runtime. This allows, with the help of the Monitor, to cut safety margins that were put in the design stage to attest for variability without incurring in timing violations. Alternatively, this can also be used to increase variability margins if deemed necessary. In any case, this is a tool that gives a degree of control over the performance of the whole chip once implemented.

Until now, a RO was composed by just a sequence of gates and so there was only one possible path. When introducing the possibility of configuration we need to somehow create alternative paths that have different lengths and so different delays, while still working within a loop. In this thesis, we consider two different approaches that accomplish this.

Probably the most obvious way to accomplish this is by creating multiple ROs and then selecting the one we are interested in by using a multiplexer, as can be seen by Figure 3.10. This has the advantage of not adding complexity to the problem of creating an RO, we just need to design as many of them as we want. The problem with this approach is that we only have one configuration per RO, so adding more configuration increases linearly the area (i.e. ten configurations require roughly ten times the area of a single RO). If we are interested in a big number of configurations this solution may become inefficient.

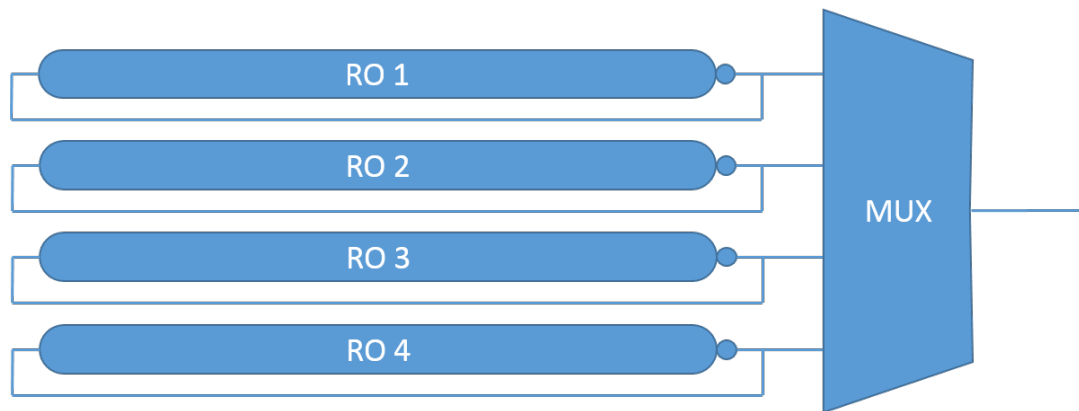


FIGURE 3.10: A multiplexer selects which Ring Oscillator to use.

Another way of implementing this may be by including multiplexers within the RO. For example, if we include a multiplexer at half the path that can select between the rest of the path and a straight wire to the end of the RO, we can choose with a single bit

whether we want the total delay or just half of it. This can be easily extended with more multiplexers. Furthermore, if we place the multiplexers always at half of the remaining length we may have an exponential number of configurations with respect to the number of multiplexers. An example of this can be seen in Figure 3.11, where we have a fixed delay and a variable delay in which each multiplexer selects a path with half of the delay of the previous one.

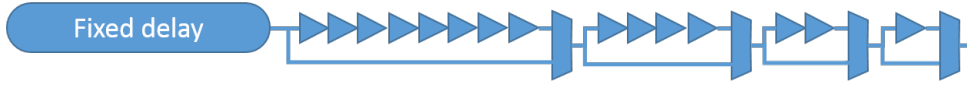


FIGURE 3.11: A Ring Oscillator with four MUX, allowing for 2^4 configurations. The triangles symbolize in this case the length of each path so that 2 triangles is twice the delay of one.

This approach is, contrary to the first one, very efficient in terms of area. Doubling the number of possible configurations only requires adding one more multiplexer. Nonetheless, there is a big problem: including a multiplexer inside the path greatly increases the complexity of generating the RO. This is caused by the slew. Consider Figure 3.12 that depicts a RO with 2 multiplexers and 4 different paths. If we choose the path shown in 3.12(a) the signal will traverse all the gates and go back to the first gate again in the loop back with a certain slew. If we were to select the path of 3.12(b), we would skip some of the gates in the middle and so the slew when arriving to the last gates would be different than what it was in 3.12(a), effectively changing the delay of those cells. Moreover, if we select the paths of 3.12(c) or 3.12(d), a different slew would then arrive for the loop back to the first gate and so modifying the delay for the next iteration. This problem must be solved when designing the RO and, as can be seen, it increases the complexity of the problem with respect to the number of multiplexers considered, potentially reducing the quality of the solutions obtained.

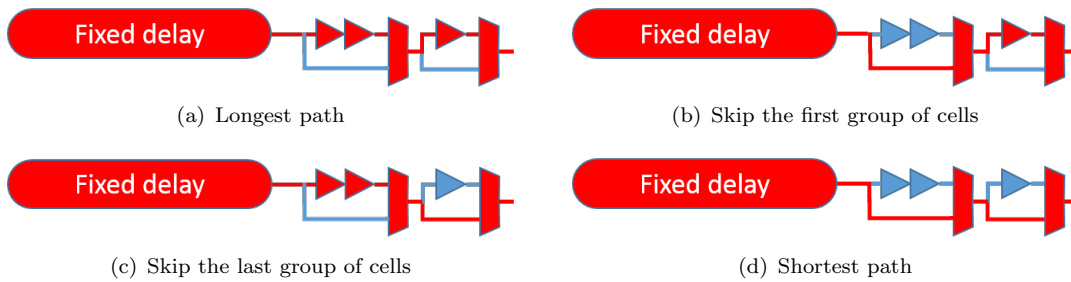


FIGURE 3.12: Ring Oscillator with four possible paths. In red are marked the paths that the signal will follow on each configuration.

Besides those two different ways of implementing configurations, it is also possible to mix them to gain the benefits of the two, at the cost of incurring a bit in the drawbacks of both. For example, if we are interested in 16 configurations we can use two RO with

three multiplexers. This will have twice the area of an RO with 4 multiplexers, but it will be significantly easier to design and will probably be more accurate.

3.4 Summary

In this Chapter we introduced the different circuits that we aim to design, with some alternatives for the case of the Ring Oscillator. It is not clear which of those different implementations for a RO is better, so they need to be analysed in the experiments chapter. We also presented the monitor circuit that, while in principle not necessary, enables the possibility of configurable ROs. This characteristic may be used for either performance gains or for safer operations modes. The following chapter will give more information in how both circuits are designed.

Chapter 4

Path Generation

4.1 Introduction

In this chapter we will give a more formal definition of the problem that we need to solve. We will start by proposing a basic version that will be later expanded. Each of the different schemes presented in Chapter 3 will be a variant of the problem. After the definition, an analysis on the complexity of the problem will follow, including a polynomial reduction from the well-known 3-SAT problem.

With a more formal definition of the problem, we will be able to discuss different alternatives for algorithms, including optimal and non-optimal schemes. A thorough definition of the algorithm that we implemented will follow, with specific versions for each of the problem variations. Finally, we will conclude the chapter with a discussion on the cost functions used by the algorithm.

4.2 Formal definition: Basic problem

This section presents a formal definition and nomenclature for the problem. We first start with a basic version of the problem, which will be expanded in the next section in order to encompass all the different alternative schemes. For reference, table 4.1 shows all the nomenclature that will be used in the basic definition of the problem.

We part from a library that can be divided into c corners that form the set corners $c \in C$. In the library we also have cells or gates. Remember that a gate may have different pins and input values, as well as different output pins. We consider that a gate g that exists in the set of gates G contains one input pin, one output pin and a specific assignment in the rest of the input pins. So for example, an *AND2* cell with two input

L	Library
C	Set of corners
c	Corner
G	Set of gates
g	Individual gate, including specific pin and signal configuration
\mathcal{P}	Path of gates
P	Ordered sequence of gates
p_i	i th gate for path P
E	Edge, either r (rise) or f (fall)
E_i	Output edge for gate i or input edge for gate $i + 1$
U	Unateness, either p (positive) or n (negative)
$\mathcal{D}_{g,c}$	Delay function for corner c and gate g
$\mathcal{S}_{g,c}$	Slew function for corner c and gate g
Q	Capacity
Q_i	Set of output capacitances for gate i or input capacitances for gate $i + 1$
$Q_{i,c}$	Output capacitance for the i th cell or input capacitance for the $i + 1$ cell for corner c
S	Slew
S_i	Set of output slews for gate i or input slew for gate $i + 1$
$S_{i,c}$	Output slew for the i th cell or input slew for the $i + 1$ cell for corner c
T	Set of target delays
t_c	Target delay for corner c
$\delta_{i,c}$	Delay for the i th cell for corner c
$d_{i,c}$	Accumulated delay for i th cell for corner c
Λ	Set of normalized delays
$\lambda_{i,c}$	Normalized delay for cell i and corner c

TABLE 4.1: Nomenclature used for the basic problem definition.

pins (A and B) and one output pin (Z) that implements the logic function $A \cdot B = Z$ actually represents four gates in our nomenclature:

- Pin A as input pin, pin Z as output pin, $B = 0$.
- Pin A as input pin, pin Z as output pin, $B = 1$.
- Pin B as input pin, pin Z as output pin, $A = 0$.
- Pin B as input pin, pin Z as output pin, $A = 1$.

Each gate $g \in G$ has associated an input capacitance $Q(E)$, that depends on the signal edge $E = \{r, f\}$ (r for rising edge, f for falling edge) and a unateness $U = p, n$ (p for

positive, n for negative¹). The library also associates, for each corner c and gate g , the following functions:

- The delay function $\mathcal{D}_{g,c}(Q, S, E)$, with Q being the capacitance connected to the output pin, S the slew in the input pin and E the edge of the input signal.
- The slew function $\mathcal{S}_{g,c}(Q, S, E)$, where Q , S and E are defined as in the delay function.

The capacity Q at the output pin of a cell is equivalent to the capacity of the input pin of the next cell. The slew S comes from the slew function from the previous cell. Finally, the edge E is function of the unateness and input edge of the previous gate.

We summarize the definition of a library L as the tuple containing:

- Set of corners C .
- Set of gates G .
- Delay function $\mathcal{D}_{g,c}(Q, S, E)$.
- Slew function $\mathcal{S}_{g,c}(Q, S, E)$.

Along with a library we also have a set of *target delays* T . Each target delay $t_c \in T$ has associated a corner c . We also define a path of gates \mathcal{P} associated with a library L as the tuple containing:

- A sequence of gates P , where p_i represents the i th gate with $1 \leq i \leq n$, and n is the length of the path.
- The input slew for the first gate S_0 .
- The input edge for the first gate E_0 .
- The output capacitance for the last gate Q_n .

For convenience, we also define S_i , E_i and Q_i , with $0 \leq i \leq n$ as the slew, edge direction and capacitance at the output of the i th cell or at the input of cell $i + 1$.

¹A positive unateness will maintain the input edge at the output, while a negative unateness will invert the input edge at the output. Also note that, in general, a gate may not beunate (for example, an *XOR* gate). In our case, though, we ensure that our definition of gate is alwaysunate by predefining the state for all the input pins.

For a given path \mathcal{P} we call $\delta_{i,c}$ the delay of the i th cell in P for corner c and we define the accumulated delay $d_{i,c}$ as:

$$d_{i,c} = \sum_{j=1}^i \delta_{j,c} \quad (4.1)$$

With $\delta_{i,c}$ computed as

$$\delta_{i,c} = \mathcal{D}_{g,c}(Q_{i,c}, S_{i-1,c}, E_{i-1}) \quad (4.2)$$

The basic goal of the problem is, given a library L , a set of target delay T and, possibly, an input slew S_0 , an input edge E_0 and an output capacitance Q_n , find a path \mathcal{P} that satisfies

$$\forall t_c \in T : \quad d_{n,c} \geq t_c \quad (4.3)$$

With n being the last cell of the path. This defines the main constraint for the problem and any path that does not satisfy it is not considered a valid solution. But this problem is in fact an optimization problem, so we lack the last ingredient, the optimization goal.

First lets define the normalized delay $\lambda_{i,c} = \frac{d_{i,c}}{t_c}$, with $\lambda_{i,c} \in \Lambda$. Note that the previous condition is satisfied if and only if $\forall c \in C : \lambda_{n,c} \geq 1$. In short, we want the values in Λ to be as small as possible. Different possible optimization goals are:

- Minimize *maximum*(Λ): Minimize the maximum of all the $\lambda \in \Lambda$, so that the worst case is as good as possible.
- Minimize *average*(Λ): Minimize the average of all the normalized delays.
- Minimize λ_{ty} : Minimize the delay of the corner corresponding to the typical corner.
- Minimize *pondered average*(Λ): Minimize a pondered average so that we may give specific *weights* to each of the corners.

It is not clear at this point which one is the best optimization goal, so we will leave the discussion for the section 4.7, in which we describe possible cost functions. For now, let us just assume that our goal is the minimization of the cost function:

$$\text{Minimize : } \quad \text{CostFunction}(T, \mathcal{P}) \quad (4.4)$$

4.3 Expanding the basic problem

Besides the basic definition of the problem exposed in the previous section, there are some other properties and constraints that we may need. Also, variations over the simple path exposed in Chapter 3, such as monitors or configurable paths, require slightly different definitions.

The following subsections will formalize some extensions over the problem. Note that most of them are not mutually exclusive and, in fact, are often used in conjunction. Table 4.2 summarizes the new nomenclature introduced in this section. Any variable not found in this table will be found in the basic nomenclature Table 4.1.

\mathcal{P}'	Identical to \mathcal{P} but with the initial input edge inverted
\mathcal{P}^i	i th path of gates
P	Sequence of vectors of gates $P = \{\{p_{1_1}, p_{1_2}, \dots, p_{1_{m_1}}\}, \{p_{2_1}, p_{2_2}, \dots, p_{2_{m_2}}\}, \dots, \{p_{n_1}, p_{n_2}, \dots, p_{1_{m_n}}\}\}$
P^i	Sequence of vectors of gates for the path \mathcal{P}^i
P_i	Vector of cells in the i th position of the path
p_{ij}	Cell in the i th position of the path in the j th position of the vector
p_j^i	j th vector of cells for the i th path
$W_c()$	Wire model for corner c
Q_{ij}	Capacitance of the cell in the i th position of the path and the j th position of the vector
Q_j^i	Set of capacitances for the vector of gates in position the j th position in the i th path
S_{max}	Maximum slew
S_j^i	Set of slews for the vector of gates in the j th position in the i th path
ty	Typical corner
n_{min}	Minimum length of the path
n_{max}	Maximum length of the path
n^i	Length of path i
F	Number of flip-flop gates in the thermometer
f	Flip-flop gate instance
$d'_{n,c}$	Accumulated delay for the last cell, corner c , with the initial input edge inverted w.r.t. $d_{n,c}$
$d'_{j,c}$	j th accumulated delay for path i and corner c
$\lambda'_{n,c}$	Normalized delay for the last cell, corner c , with the initial input edge inverted w.r.t. $d_{n,c}$
Λ'	Set of inverted normalized delays
M	Number of multiplexers in a configuration path
Δ	Vector of coefficients for target delay
δ_i	i th coefficient from Δ

TABLE 4.2: Nomenclature used for the extended problem definition.

4.3.1 Extensions on the problem

In order to allow for modelling some of the variations over the problem, as well as gain in accuracy, we include the following extensions over the basic problem.

4.3.1.1 Additional cells

We extend the definition of path of cells to include the possibility of connecting a cell to more than one cell. A path $P = \{p_1, p_2, \dots, p_n\}$ previously contained exactly n cells in sequence. Now, we define p_i as a vector of cells of the form $p_i = \{p_{i_1}, p_{i_1}, \dots, p_{i_{m^i}}\}$, where p_{i_j} represents an instance of a cell. Only the first cell, denoted p_{i_1} , has its output pin actually connected to the following cells, as shown in Figure 4.1.

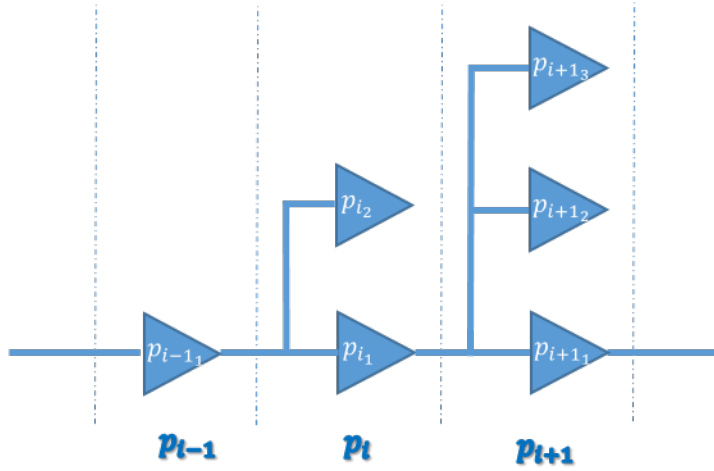


FIGURE 4.1: Each triangle represents an arbitrary cell. Only the first cell of each set has its output pin connected and so is the only one traversed by the signal.

The rest of the cells p_{i_j} with $1 < j \leq m^i$ affect the delay by changing the capacity. Thus the capacity is now computed as follows:

$$Q_{i,c} = \sum_{j=1}^{m^i} Q_{i_j,c} \quad (4.5)$$

with $m^i = |p_i|$ representing the number of cells in p_i .

4.3.1.2 Wire models

For now we have omitted the presence of the wires that interconnect the cells, but they also have an impact on the delay by modifying the capacitance. Libraries often include approximations for the influence of wires in capacitance in the way of *wire models*. These

models use the fanout² of the cell in order to estimate a capacitance. The computation of capacity having on account *wire models* is as follows:

$$Q_{i,c} = W_c(m^i) + \sum_{j=1}^{m^i} Q_{i_j,c} \quad (4.6)$$

with $W_c()$ being the *wire model* function for corner c obtained from the library.

4.3.2 Additional constraints

We introduce some additional constraints to the previous only constraint. These new constraints exist to either give more control over the resulting path or to allow for variations in the problem that will later be presented.

4.3.2.1 Maximum Slew

The slew of a signal is one of the main factors in the delay computation of a cell. It has also a great impact in the sensitivity to variability of the cells it drives. In some occasions we may be interested in limiting the maximum values that the slew can reach in order to have better control over variability.

As the slew for all the corners is correlated (high slew in one corner also gives a high slew in the rest of corners) it is enough with checking that the slew at the *typical corner* does not surpass some input specified maximum slew S_{max} . The new constraint is as follows:

$$\forall i \in [1, n] : \quad S_{i,ty} < S_{max} \quad (4.7)$$

4.3.2.2 Length of the path

Another property we may need is a specific length for the path measured in number of gates. We also consider a certain margin over the length. We define n_{min} as the minimum length of the path and n_{max} as the maximum length and add the following constraint:

$$n_{min} \leq |P| \leq n_{max} \quad (4.8)$$

²The fanout of a cell is the number of elements to which the output pin is connected to.

4.3.2.3 Inverted edge

One of the variations of the problem, the Dual Ring Oscillator, requires that the propagation of the signal along the path for a specific input edge is faster than the inverted edge. This adds the following constraint:

$$\forall c \in C : d_{n,c} > d'_{n,c} \quad (4.9)$$

with $d \in \mathcal{P} = \{P, S_0, E_0, Q_n\}$ and $d' \in \mathcal{P}' = \{P, S_0, \bar{E}_0, Q_n\}$.

4.3.2.4 Parity of the inversions

It is often required to control the parity of the inversions in the path. For example, if we need that the signal is inverted at the end of the path with respect to the input edge, we need an odd number of inversions. An even number of inversions will produce an output edge similar to the input edge. Thus we have two different possible constraints.

Inverting path,

$$E_0 \neq E_n \quad (4.10)$$

Non-inverting path,

$$E_0 = E_n \quad (4.11)$$

4.3.3 Variations on the problem

The definition of the problem at this point allows to model paths of gates that trace variability. Still, we are actually interested in Ring Oscillators and Monitors, as well as the possibility of dynamically selecting a path. This section defines the different problems that we actually need to solve.

4.3.3.1 Monitor

The monitor path problem is very similar to the basic problem. It only introduces a new constraint, subject to new inputs. For the monitor path we are interested in building a *thermometer* (see Section 3.2) at the end of the path, so we need to know the amount of flip-flops that we want to use. We will call this number F . We also require a specific flip-flop gate f .

The only difference between the Monitor path and the basic problem path is that the last F gates have as an additional cell a flip-flop. Thus we can define the only additional constraint as:

$$\forall i \in [n - F, n] : \quad p_{i_2} = f \quad (4.12)$$

4.3.3.2 Simple Ring Oscillator

This RO is the simplest one and we have most of the ingredients to specify it. We only lack one more optimization goal that has not yet been defined and modify some constraints. Remember from Section 3.3.1 that this kind of RO needs to have a similar delay for both raising and falling input edges. We cannot add a constraint that forces to have both edges equally, as a solution that satisfies it may not even exist. We can settle with a more relaxed version:

$$\forall c \in C : \quad d_{n,c} \approx d'_{n,c} \quad (4.13)$$

with $d \in \mathcal{P} = \{P, S_0, E_0, Q_n\}$ and $d' \in \mathcal{P}' = \{P, S_0, \bar{E}_0, Q_n\}$.

Now this constraint is very unspecific as to what “ \approx ” means. Defining that constraint with some margins is a possibility, yet then choosing those margins would be tricky and can lead to worse solutions. We instead decided to modify the optimization goal to reflect this constraint without specifying margins, as we will see below.

The Simple RO also needs to change some constraints. Remember that the signal has to traverse the path of gates twice each period, one for the rising edge and one for the falling edge. This forces us to add the inverting edge constraint 4.16 and half the target delay constraints 4.14 and 4.15 – we are now interested in only half of the target delay for each traversal of the path.

Another thing to take into account is that we are now defining a Ring Oscillator that is connected by a feedback loop. This means that the output capacitance for the last cell will actually be the input capacitance for the first cell. Consequently, the output slew for the last cell is also the input slew for the first cell, yet enforcing this as a constraint may render the problem unsolvable. We will only add constraint 4.17 for the capacitance and constraint 4.18 for the slew of the inverted edge.

In previous sections we defined the normalized delay as $\lambda_{i,c} = \frac{d_{i,c}}{t_c}$, with $\lambda_{i,c} \in \Lambda$ and $d \in \mathcal{P} = \{P, S_0, E_0, Q_n\}$. We extend this definition to include the inverted edge delay

and slew $\lambda'_{i,c} = \frac{d'_{i,c}}{t_c}$, with $\lambda'_{i,c} \in \Lambda'$ and $d' \in \mathcal{P}' = \{P, S'_0, \bar{E}_0, Q_n\}$. The new constraints are:

$$\forall c \in C : \quad \lambda_{n,c} \geq 0.5 \quad (4.14)$$

$$\forall c \in C : \quad \lambda'_{n,c} \geq 0.5 \quad (4.15)$$

$$E_0 \neq E_n \quad (4.16)$$

$$Q_0 = Q_n \quad (4.17)$$

$$S'_0 = S_n \quad (4.18)$$

And the optimization goal:

$$\text{Minimize :} \quad \text{CostFunction}(T, \mathcal{P}) + \text{CostFunction}(T, \mathcal{P}') \quad (4.19)$$

4.3.3.3 Dual Ring Oscillator

The Dual RO is actually two different paths. For simplicity, we will divide the problem in two different problems that can be solved independently. We will call these two problems \mathcal{P}_1 and \mathcal{P}_2 . The main difference between them is the input edge E_0 :

$$\mathcal{P}_1 = \{P^1, S_0^1, E_0, Q_n\} \quad (4.20)$$

$$\mathcal{P}_2 = \{P^2, S_0^2, \bar{E}_0, Q_n\} \quad (4.21)$$

Note that the paths P^1 and P^2 are different, as well as the input slew S_0^1 and S_0^2 , but the output capacitance must be the same. In fact, this capacitance must be the addition of capacitances of input cells plus the wire model and this will be modelled as constraint 4.25. This is because the paths are actually connected together. Similarly, the output slew for one path is the input slew for the other. As in the case for the simple RO, enforcing this may not be possible, so we add the more relaxed constraint 4.26.

Another constraint that both paths must follow is a fixed last cell. The last cell is a *C-element* cell, yet each path is connected to a different set of pins. We will call $p_{n1}^1 \in P^1$ the last cell for \mathcal{P}_1 and $p_{n2}^2 \in P^2$ the last cell for \mathcal{P}_2 . The C-element cell with different pins configurations will be ce_1 and ce_2 . This produces constraints 4.27 and 4.28.

Similarly to the Simple RO, each path traversal corresponds to half of the target delay. This also adds the constraint of having an odd number of inversions (constraint 4.23)

and halves the target delay for each path (constraint 4.24). Yet in this case, we also need the *inverted edge* constraint 4.22 explained in Section 4.3.2.3.

Finally, the optimization goal is independent for each path and so does not need to be modified. The new constraints for this problem are:

$$\forall c \in C : \quad d_{n,c}^1 > d_{n,c}^2 \quad (4.22)$$

$$E_0 \neq E_n \quad (4.23)$$

$$\forall c \in C : \quad \lambda_{n,c} \geq 0.5 \quad (4.24)$$

$$Q_n = Q_0^1 + Q_0^2 + W(|p_1^1| + |p_1^2|) \quad (4.25)$$

$$S_0^2 = S_{n^1}^1 \quad (4.26)$$

$$p_{n^1}^1 = ce_1 \quad (4.27)$$

$$p_{n^2}^2 = ce_2 \quad (4.28)$$

4.3.3.4 Configurable path

This variant of the problem addresses the optional extension of dynamic configuration of the path introduced in Section 3.3.3. Note that while this can (and should) be used in conjunction with other schemes, here we will define it as a separate problem.

The input for this problem needs to include the number of multiplexers M that will be added into the path. There must also be an input vector Δ of size $M+1$ with coefficients for the target delay, so that we know how long each path is. The first element of the vector Δ is the time for the fixed delay, the second element is the time between the first and second multiplexer, and so on. For example, $\Delta = \{0.5, 0.4, 0.3\}$ indicates that half of the target delay corresponds to the fixed delay, the path between the fixed delay and the first multiplexer is 0.4 times the target delay and the time between the first and second multiplexers is 0.3 times the target delay. Note that the sum of all the coefficients is not necessarily equal to 1. Figure 4.2 shows how the Δ vector is distributed.

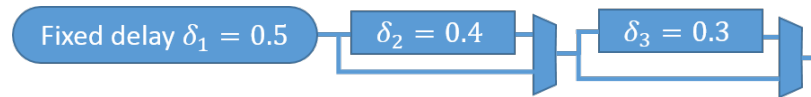


FIGURE 4.2: Distribution of the Δ vector of coefficients for target delay along the path.

The multiplexers may be any cell from the library that have a function that allows the selection of one of its inputs based on configuration signals. We consider that only one type of cell is used as multiplexer and defined by the input. There must also be two

different configurations: the fixed configuration, that skips the longest path, and the long configuration, that propagates the signal through the longest path. We will call the gates thus configured m_f and m_l respectively, as can be seen in Figure 4.3.

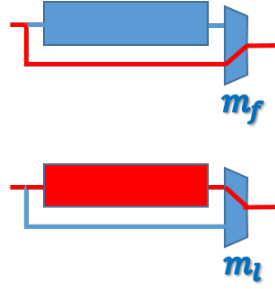


FIGURE 4.3: Naming convention for the two possible multiplexer configurations. We will refer to the configuration that selects the shortest path as m_f , represented at the top. At the bottom is depicted the longest delay configuration, which we will call m_l .

For modelling the configurable path, we will actually consider $M + 1$ different paths $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_{M+1}$. The first path \mathcal{P}_1 , that corresponds to the fixed delay, will have M multiplexers m_f at the end of the path, as expressed by constraint 4.30. The other paths all end with just one multiplexer m_l (constraint 4.31). Each path will have a different target delay based on the Δ vector of coefficients. In particular, path \mathcal{P}_i has target delay $T \cdot \delta_i$ with $\delta_i \in \Delta$ and $1 \leq i \leq M + 1$. This is modelled by constraint 4.29.

The input slew for paths other than the first one will also be dependant on the slew at the input of the corresponding multiplexer in the first path. This means that the input slew for path i , that we will call S_0^i , corresponds to the input slew for cell $p_{n^1-(M+1-i)}^1$ from the fixed delay path. Thus, for paths \mathcal{P}_i with $2 \leq i \leq M + 1$, the input slew S_0^i is equal to $S_{n^1-(M+2-i)}^1$. Constraint 4.32 shows this restriction. Figure 4.4 shows the distribution in different paths as well as the points from different paths that share slew.

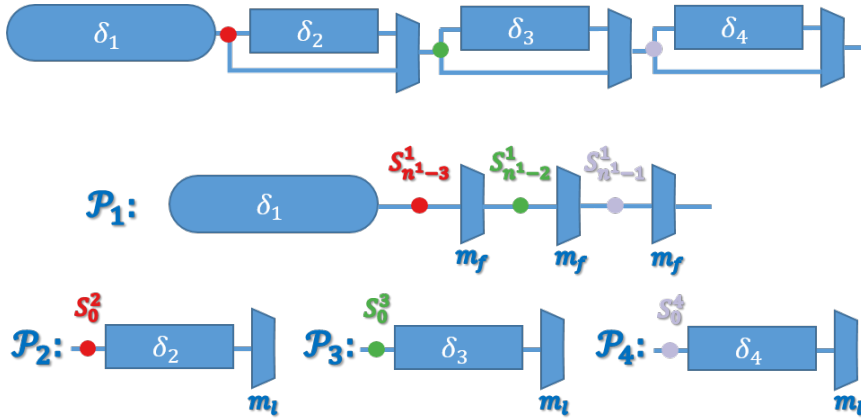


FIGURE 4.4: At the top is represented the path that we want to model. The path \mathcal{P}_1 at the middle represents the fixed delay, while the paths at the bottom represent the configurable delay. The circles marked at each path in a different color represent the points in the path that share slew.

This last constraint allows us to correctly model the propagation of a signal from the fixed delay path to exactly one of the longer paths. The moment a signal goes through one of the alternative paths the slew will make delays computations incorrect for the rest of the path. To palliate this effect we will introduce a new optimization goal that tries to match the slew at the end of the alternative paths to the slew at the output of the multiplexers in the fixed path. The objective of this new goal is to reduce the difference between the slew $S_{n^i}^i$ and the slew $S_{n^1-(M+1-i)}^1$, with $2 \leq i \leq M+1$, n^1 the length of the fixed delay path and n^i the length of the i th path.

The new target delay constraint for all the paths is:

$$\forall i \in [1, M+1], \forall c \in C : \quad \lambda_{n,c}^i \geq \delta_i \quad (4.29)$$

The constrain for path \mathcal{P}_1 is :

$$\forall i \in [n-M, M] : \quad p_i^1 = \{m_f\} \quad (4.30)$$

Constraints for paths \mathcal{P}_i with $2 \leq i \leq M+1$ are:

$$\forall i \in [2, M+1] : \quad p_{n^i}^i = \{m_l\} \quad (4.31)$$

$$\forall i \in [2, M+1] : \quad S_0^i = S_{n^1-(M+2-i)}^1 \quad (4.32)$$

The new optimization goals for the same paths need to reduce the difference between slews like so:

$$\text{Minimize :} \quad \forall i \in [2, M+1], \forall c \in C : \quad |S_{c,n^1-(M+1-i)}^1 - S_{n^i}^i| \quad (4.33)$$

Yet this is not a valid function. The actual optimization goal is, as previously, to minimize the cost function. This cost function will be latter discussed in Section 4.7.4.

4.4 Problem analysis

With the problem defined it is now possible to analyse it. In this section we will try to understand where the difficulty of this problem lies as well as give some proof and intuition about its complexity.

The first thing that should be noted are the dependencies for the delay computation for a cell. On the one hand, we have the capacity for the output pins in a cell. This capacity is dependent of the next cell (or set of cells, if they are connected in parallel), and thus the delay cannot be accurately known without looking at least one step ahead. On the other hand, the edge and slew depend on the previous cell. Furthermore, the output slew is also dependent on the input slew, so in practice the slew and delay for the i th cell in the path is dependent on all the previous cells from 1 to $i - 1$.

This entanglement of dependencies, especially the ones concerning the slew, makes it impossible to have any partial solution that is guaranteed to be optimal. For example, if we were to construct the path one cell at a time, we cannot at any point know for sure that the portion of the path already built is optimal. Even the first cell of the path will modify how the delay of the last cell is computed. This inconveniently prevents us from reaching an optimal solution by using a simple greedy algorithm that selects at each step the best candidate.

In order to give a formal proof about the hardness of the problem, we show below a polynomial reduction from 3-SAT to a decisional version of the basic problem. This will prove that the basic problem is at least as hard as 3-SAT, placing it into the NP-hard complexity class.

4.4.1 3-SAT polynomial reduction

The Boolean Satisfiability Problem, often abbreviated as SAT, is a decisional problem consisting in determining whether there exists an assignment of variables that satisfies a given boolean formula. SAT was also one of the first problems proven to be NP-complete. In the most general version the boolean formula may have any shape, but here we will only consider the 3-SAT version that is also proved to be NP-complete.

The 3-SAT problem is identical to the regular SAT problem, yet with restrictions for the input formula. In particular, the formula must be in conjunctive normal form (CNF), where each clause has at most three literals. Sometimes, it is required for 3-SAT that each clause has *exactly* 3 literals. Note that this does not affect the complexity, as auxiliary variables can be added to complete the restriction, requiring extra clauses as well to ensure that all solutions assign the new variables to false.

For simplicity in the proof, we will consider the 3-SAT version that requires exactly 3 literals per clause. We will also require that there are no duplicate clauses. This last restriction may be satisfied by a polynomial search over the formula, removing repeated clauses.

We also need to transform the basic problem to which we want to make the reduction into a decisional problem. For simplicity of the proof, we will simplify the basic problem and only consider one corner. We will call the decisional version of this problem PATH. Maintaining all the restrictions from the basic problem, except for the optimization goals, PATH is the problem of deciding whether there exist a path of gates such that $t \leq d < t + \epsilon$, with d being the accumulated delay of the path, t the target delay and ϵ a positive rational number.

4.4.1.1 Reduction

An instance from 3-SAT is composed by N variables and M clauses. Each variable is enumerated by appearance order from 1 to N . For each clause C_i , $1 \leq i \leq M$, a set of 7 gates G_i is created into the library L . Each of these 7 gates from G_i has the delay and slew functions constructed in base to the 7 possible assignments to C_i that satisfy the clause.

Consider for example the clause $(a \vee b \vee \bar{c})$. There are 7 assignments for the variables a , b and c that satisfy it, which are: $(a = 0, b = 0, c = 0)$, $(a = 0, b = 1, c = 0)$, $(a = 0, b = 1, c = 1)$, $(a = 1, b = 0, c = 0)$, $(a = 1, b = 0, c = 1)$, $(a = 1, b = 1, c = 0)$ and $(a = 1, b = 1, c = 1)$, but not $(a = 0, b = 0, c = 1)$. There will be one gate representing each of these possible assignments and we will refer individually to them as $g_{i,j}$, with $1 \leq i \leq M$ identifying the clause C_i and $1 \leq j \leq 7$ identifying the variable assignment. All the gates created are identical except for the slew and delay functions. A more graphic example of this gate creation can be seen in Figure 4.5.

Remember from the introduction to this section that the slew at any one point in the path may depend on all the previous cells. We will make use of this to encode information into the slew. Let's first define a possible encoding. The initial slew will be a natural number which most significant digit is a 1 followed by $N + M$ 0's. We will refer to this digits as Y_i , with $1 \leq i \leq N + M$, and Y_1 being the digit immediately to the right of the initial 1, Y_2 at the right of Y_1 and so on. We now define the codification:

- A 0 in Y_i , with $1 \leq i \leq N$, indicates that the slew has not traversed any gate from the set G_i , while a 1 indicates that the slew has traversed some gate from G_i .
- A 0 in Y_i , with $N + 1 \leq i \leq M$, indicates that the slew has not traversed any gate from a set G_j such that the clause C_j contains the variable $(i - N)$.
- A 1 in Y_i , with $N + 1 \leq i \leq M$, indicates that the slew has traversed some gate $g_{j,k} \in G_j$ such that the clause C_j contains the variable $(i - N)$ and the gate $g_{j,k}$ refers to an assignment where the variable $(i - N)$ is assigned to *FALSE*.

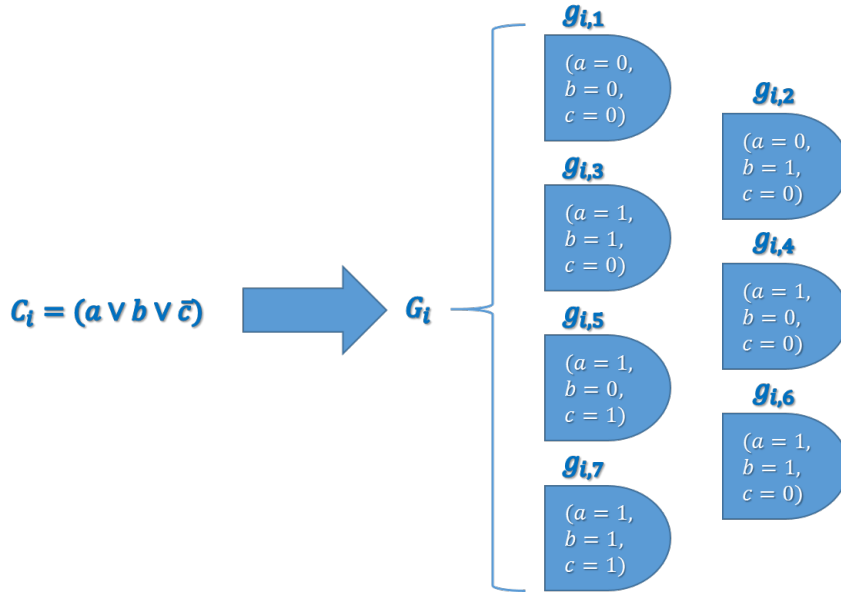


FIGURE 4.5: The i th clause generates a set of 7 gates G_i . Each gate $g_{i,j}$ in this set represents a satisfiable assignment of the clause, numbered from 1 to 7.

- A 2 in Y_i , with $N + 1 \leq i \leq M$, is similar to the previous case, but with an assignment to *TRUE*.

For example, consider a problem with $N = 4$ and $M = 9$. The slew 1 1010 021102100 indicates that only gates from the sets G_1 and G_3 have been seen. It also shows that the variable 2 and 6 have been assigned to true and the variables 3, 4 and 7 have been assigned to false. No other variables have been yet assigned.

It is now possible to define the slew function. For simplicity, the slew function will only depend on the input slew. Thus, for each cell $g_{i,j}$, with $1 \leq i \leq N$ and $1 \leq j \leq 7$ we can define the slew function $Sg_{i,j}(S_{in})$. The slew at the output is identical to the slew at the input S_{in} with the following changes:

- The i th digit of the slew is set to 1, indicating that clause C_i has already been traversed.
- By looking at the j th assignment for C_i we set the appropriate values for the variables with respect to the encoding previously defined.

Finally, we may also define the delay function $Dg_{i,j}(S_{in})$ for gate $g_{i,j}$, with $1 \leq i \leq N$ and $1 \leq j \leq 7$. Note that, like in the slew case, the delay is only function of the slew. Depending on the slew, the delay function outputs the following:

- If the i th digit of the input slew is a 1, the delay is $t + \epsilon$.

- If the encode of the input slew shows that any of the variables contained in clause C_i has already been assigned, and the assignment is different than the same variable in the j th assignment for clause C_i , the delay is $t + \epsilon$.
- Otherwise, the delay is t/N .

The value for the target delay and for ϵ may be any positive rational number, for example, $t = N$ and $\epsilon = 1$. Note that this construction can be done in polynomial time and nothing else is required for the reduction. If the formula is satisfiable, it is guaranteed that a path exists such that its accumulated delay d satisfies $t \leq d < t + \epsilon$. In this case, exactly N gates will be used, one for each group of cells, and their variable assignments will be compatible. For construction, we also know that all the variables will have a value assigned. If no such path exists, it can only be because there is no possible assignment of variables that satisfies the formula, thus the formula is unsatisfiable.

As an example, consider the formula $(a \vee b \vee \bar{c}) \wedge (a \vee \bar{b} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d)$ that has $N = 3$ clauses and $M = 4$ variables. The library for this formula would contain 3 sets of gates G with a total of 21 gates. A possible path that satisfies constraints can be seen in Figure 4.6. Note from that example that there is exactly one gate from each clause. The gates also contain compatible values for the variables and they correspond to a satisfiable assignment of the formula.

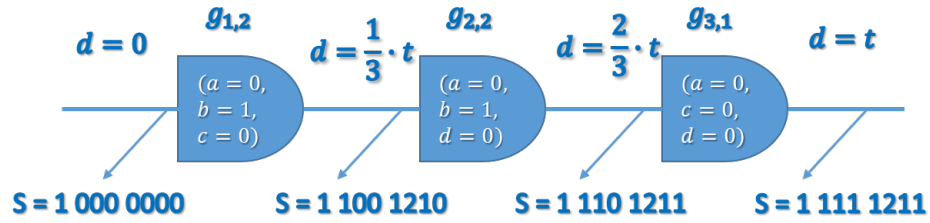


FIGURE 4.6: Correct path for the example formula. Slew at each point is shown below, while delay is shown above.

4.5 Solving the problem

As we have shown in the previous section, this is a hard problem. Even so, there are lots of possible strategies and schemes that may solve it in reasonable times, often with compromises regarding quality on the solution. In this section we will do a brief analysis for both approaches that yield optimal solutions and approaches that compromise quality as a trade-off for complexity and execution time.

4.5.0.2 Optimal solutions

We have seen that any optimal algorithm would need to pay an exponential cost unless $P = NP$. This is the case for lots of important problems that are, nonetheless, possible to solve by paying that cost for real life instances. We will now propose a couple of algorithms that try to do that and see if it would make sense to implement them.

First of all, it is always interesting to see what the naive algorithm would look like. Let us assume that we are trying to solve the basic problem without cells in parallel³. The naive algorithm would evaluate all the possible combinations of cells such that the delay is greater than the target delay. Solutions were removing one or more cells still gives a delay greater than the target delay for all the corners are not considered (this prevents an infinite amount of possible combinations). After each evaluation, the best solution so far is stored. The algorithm returns at the end the best solution stored.

If the longest possible path is N and the amount of cells in the library is M , it can be seen that the algorithm would have a complexity proportional to $\mathcal{O}(M^N)$. In a real-life instance of the problem, N is typically in the order of several tens and M in the order of several thousands. Thus this naive algorithm would rarely be useful for real instances.

By looking at the formal definition of the problem, it may seem somehow natural to propose a reduction to ILP (Integer Linear Programming). In fact, the constraints could be directly translated from the definition, including the optimization goals. While the ILP algorithms are still within the realm of the exponential, they have proven to be quite fast in practical applications. Yet in this case there is one catch: the delay and slew functions. These two functions would need to be translated as variables and, as explained earlier, both delay and slew depend on all the previous cells. It is necessary, then, to have one delay and slew variable for each possible combination of paths. We would need to pay the same cost of the naive algorithm just to generate the ILP instance, and then we may potentially need an exponential cost over it for solving it.

Similar problems arise when considering reductions to other kind of well-known problem solving frames such as SAT, constraint programming or SMT. This does not mean that there does not exist any optimal algorithm that solves this problem for some real-life instances in reasonable time, but may give an intuition of the difficulty of designing it. We were, in fact, unable to find any such optimal algorithm.

³With cells in parallel we would need to ensure either that the delay function is monotonic with respect to the capacity or fix a constant bound over the number of cells in parallel in order to have a finite amount of combinations. If not, the decisional version of this problem would not be decidable.

4.5.0.3 Non-optimal solutions

There is a huge number of ways for solving this problem if we relax the optimality requirement. Here we will consider two ways: simplifying the problem and solving it optimally or solving the complete problem with a non-optimal algorithm.

The first way allows for controlling how much precision you may have, by deciding in which way to simplify the problem. An example of this approach can be found in [19], where they solve a very similar problem to ours. In real-life instances, the slew function tends to converge to a certain value when applied to a given sequence of gates. For example, the slew at the end of a sequence of AND gates will eventually converge to a certain value, regardless of the input slew. In [19] they make use of this property to group sequences of several gates. They also only consider a relatively small set of groups as their atomic unit for building a path, and ignore the interaction of the slew between groups. With this, they can use an ILP algorithm to optimally solve the simplified problem.

There exist a wide range of algorithms and techniques for the non-optimal approach that range from simple greedy algorithm to more complex approximate algorithms. Greedy algorithms are commonly very fast at solving problems, but do not give any short of guarantee with respect to the quality. On the other end of the spectrum, approximate algorithms are typically not as fast as a greedy one, yet they give very specific boundaries for the quality of the solution. Regardless of that, approximate algorithms may not exist for every problem and, even if they do, they not always yield high quality solutions. In this work we decided to use a middle ground by choosing a metaheuristic algorithm.

Metaheuristic algorithms have been widely used since they appearance for solving complex optimization problems often yielding near-optimal solutions. They share a similarity with greedy algorithms in that they are often fast and do not give any guaranty respect the quality of the solution. Even so, they usually give high-quality solutions in short execution times and, sometimes, even optimal ones. There is a wide range of metaheuristic algorithms, each one usually working well for one type of problems and not so well for other types. In fact, even if sometimes there is some intuition as to what metaheuristic works well with what problem, the only way to know which is preferable is to actually test them.

In our work we decided after some tests to use a metaheuristic called Beam Search that performs a wide range of greedy algorithms in parallel, keeping a constant set of best solutions. This metaheuristic and, in particular, the implementation that we did for this problem, will be explained in the following section.

4.6 Beam Search algorithm

The Beam Search algorithm is a search algorithm that explores the space of solutions by expanding the most promising set of nodes. It is an extension for a greedy search algorithm that at each step selects the best possible solution. In the case of Beam Search, it always keeps a set of best solutions with a fixed size.

A fast and informal description of the algorithm is as follows:

- We part from a set (with fixed length) of empty paths of gates.
- At each step, we evaluate with a cost function our current paths with a new cell at the end. This is evaluated with all possible cells in the library and each path in the set.
- The best solutions found replace the ones in the set.
- This continues until all the solutions in the set have finished paths.
- The output is the best solution.

A more formal specification of the algorithm for the basic problem can be found in Algorithm 1. The input for this problem is a set of gates G and a target delay T that serves to compute the cost function. Additionally, we need an input slew S_0 and an input edge E_0 for the first cell as well as an output capacitance Q_{out} for the last cell. The output is a path of gates satisfying the constraints imposed by T .

In the first part of the algorithm we initialize the set of candidates with a fixed size K . This set of candidates will always contain the best K paths found so far, as well as the cost of each one. At the end of the initialization, the set *Cand* has the best K paths that have just one cell.

The core of the algorithm is a loop that goes on until all candidates contain a path with a delay greater than T . At each iteration, we evaluate the cost of adding any possible cell to any possible candidate path, including paths in which we do not add any cell. Note that this could stuck the algorithm into a fixed length. To solve this, the *getBestCandidates()* function will always select paths with one more cell unless a path already satisfies the constraints. After the core loop, the best cell found in the *Cand* set is returned.

It is possible to expand this core algorithm to solve the different variants of the problem. Additional constraints such as the maximum slew, length of the path, the inverted

Algorithm 1: Basic Beam Search algorithm

input : A set of gates G , a set of target delay T , input slew S_0 , input edge E_0 and output capacitance Q_{out}

output: A path of gates satisfying the basic problem constraints

begin

```

   $Cand \leftarrow \emptyset$  //Set of candidates
   $Test \leftarrow \emptyset$  //Set of paths tested so far
  foreach  $g \in G$  do
     $\mathcal{P} \leftarrow \{P = \{g\}, S_0, E_0, Q_{out}\}$  //Path with only one gate
     $cost \leftarrow \text{costFunction}(\mathcal{P}, T)$ 
     $Test \leftarrow \{Test | \{\mathcal{P}, cost\}\}$  //add new path with cost
   $Cand \leftarrow \text{getBestCandidates}(Test, K, T)$  //Assign best  $K$  candidates to Cand
   $L := 2$  //Minimum length for paths that do not satisfy constraints
  while not  $\text{allPathsSatisfyConstraints}(Cand)$  do
     $Test \leftarrow \emptyset$ 
    foreach  $c \in Cand$  do
       $Test \leftarrow \{Test | c\}$ 
      foreach  $g \in G$  do
         $\mathcal{P} \leftarrow \text{appendCell}(c, g)$  //Append gate  $g$  to the end of the path from  $c$ 
         $cost \leftarrow \text{costFunction}(\mathcal{P}, T)$ 
         $Test \leftarrow \{Test | \{\mathcal{P}, cost\}\}$ 
       $Cand \leftarrow \text{getBestCandidates}(Test, K, T, L)$ 
      /* Assign best  $K$  candidates to Cand. Unless a path already
         satisfies constraints, the candidates selected from  $Test$  must
         have at least length  $L$  */
     $L := L + 1$ 
  return:  $\text{bestCandidate}(Cand)$ 

```

edge or the parity of inversions can be dealt with by not accepting paths that do not satisfy them. This can be done by the *getBestCandidates()* function by controlling which paths are valid. The additional cells and wire models can be taken into account while evaluating the paths by the *costFunction()* function. There only remains the different variations of the problem, each of which will need an algorithm of their own.

4.6.1 Monitor algorithm

The monitor path problem only requires the addition of a *thermometer* at the end of the path with respect to the basic problem. This requirement may be taken into account by solving two problems and merging the solutions.

Algorithm 2 shows the algorithm used for this version of the problem. First note that we now require a new input parameter F for the length of the *thermometer* as well as

a parameter f for the type of flip-flop that the *thermometer* will use. We may consider two parts for this algorithm:

- The first part of the algorithm solves a path of length F , in which each cell that we add to the path has as an additional cell a flip-flop f . There are no constraints regarding the target delay T , but any other constraint required (such as max slew), may still be applied. At the end of this part, we will have a *thermometer* path with delay D_{th} and length f .
- The second part of the algorithm just solves the basic problem but with a target delay $T - D_{th}$, which output capacitance Q_{out} equals the first input capacitance of the *thermometer* path. If length constraints apply, the new required length will be $N - F$. After the basic algorithm finishes, we append the *thermometer* path to the best path and return it as output.

One thing to consider in this algorithm is that we are, in fact, ignoring the slew of the last cell of the path previous to the *thermometer*. This simplification takes advantage of the properties in real scenarios for slew, which converges to certain values with sequences of gates. Even so, there is some error introduced by this simplification, which may be palliated by the introduction of slew optimization goals if necessary.

Algorithm 2: Monitor Beam Search algorithm

input : A set of gates G , a set of target delay T , input slew S_0 , input edge E_0 , output capacitance Q_{out} , a natural number F for the length of the thermometer and a flip-flop cell f

output: A path of gates satisfying the monitor requirements

begin

```

    Thermometer  $\leftarrow$  generateThermometer( $G, T, S_0, E_0, Q_{out}, F, f$ )
    /* Thermometer with length  $F$ . The only difference of this algorithm
       w.r.t. the basic algorithm is that each cell that we add has a
       flip-flop  $f$  as an additional cell and that no constraints
       regarding  $T$  are taken into account */
     $D_{th} \leftarrow$  getDelay(Thermometer)
     $Q_{th} \leftarrow$  getInitialCapacitance(Thermometer)
    BasicPath  $\leftarrow$  basicBeamSearch( $G, T - D_{th}, S_0, E_0, Q_{th}$ )
    /* Call to the basic Beam Search Algorithm */
    Path  $\leftarrow$  appendPath(BasicPath, Thermometer)
    /* We append the path Thermometer to BasicPath */
    return: bestCandidate(Path)

```

4.6.2 Simple Ring Oscillator Algorithm

The problem variation for the simple RO does not require a huge modification over the basic algorithm. The special constraint required for this problem is limited to the parity of the inversions, which can be dealt by the function that selects best candidates. We also need to take into account the new optimization goal that requires similar delays for both possible input edges. This will then be encoded into the cost function. A straightforward way of implementing this is, as suggested by the formal definition of this variation of the problem, adding the costs from delays for both possible input edges.

The only thing that remains to specify is how to guarantee that the input capacitance and slew for the first cell are identical to the output capacitance and slew for the last cell. We will accomplish this by repeating several iterations of the basic algorithm. In the first iteration, the values for capacitance and slew used will be from the input, but subsequent iterations use the cells from previous paths. Ideally, this will be repeated until convergence (paths in two iterations are identical), but in practice this may never happen. This *unrolling* of the loop is thus repeated an input-fixed amount of times I .

Algorithm 3 shows more specifically how this is handled. A first execution of the basic algorithm provides us with capacitance and slew values for more accurate approximations. After we have an initial path, a loop iterates I times repeating the basic algorithm with capacitance and slew values from the previous iteration. After the number of required steps is performed, the algorithm returns the last path constructed.

Algorithm 3: Simple Ring Oscillator Beam Search algorithm

input : A set of gates G , a set of target delay T , input slew S_0 , input edge E_0 , output capacitance Q_{out} , amount of loop unrolls I

output: A path of gates satisfying the simple Ring Oscillator requirements

begin

$\mathcal{P} \leftarrow \text{basicBeamSearch}(G, T \cdot 0.5, S_0, E_0, Q_{out})$ //First path

while $I > 0$ **do**

$Q \leftarrow \text{getInitialCapacitance}(\mathcal{P})$

$S \leftarrow \text{getFinalSlew}(\mathcal{P})$

$\mathcal{P} \leftarrow \text{basicBeamSearch}(G, T \cdot 0.5, S, E_0, Q)$

$I := I - 1$

return: \mathcal{P}

4.6.3 Dual Ring Oscillator Algorithm

The Dual RO problem was defined as two separate paths joined by a C-element cell. Each of the separate problems only needed to have in common the output capacitance,

which was equivalent to the summation of input capacitances for the first cell. Besides this, each path is connected to a different configuration of a C-element cell that we called ce_1 and ce_2 . The only additional constraint besides these characteristics is the *inverted edge* constraint explained in section 4.3.2.3. This constraint can be taken into account by the *getBestCandidates()* function.

Before going into the algorithm for this problem, we need to expand the basic version of the algorithm represented by Algorithm 1. This extension will allow for having a path predefined by input appended at the end of the path that we construct. Algorithm 4 shows the new algorithm. Note that the only differences are:

- The input now requires a *partial path* that will be appended at the end of the best solution.
- We evaluate the cost function of each new cell appending the *partial path* at the end of the path, yet the path added to the set of candidate solutions does not include the *partial path*.
- After the best solution is found we append the *partial path* before returning it.

This new version of the algorithm will allow us to include the C-element cell at the end of the paths. Algorithm 5 shows the pseudocode for the Dual RO problem. The inputs are similar to those of the Simple RO problem, yet we now have the two possible C-Element configurations ce_1 and ce_2 . Like in the Simple RO algorithm, we include as input parameter the number of loop unrolls I , that will be used to satisfy the conditions for capacitance at the output of the paths, as well as propagate the slew between paths.

The algorithm starts by initializing the two paths with the input values. Note the calls to the Algorithm 4, that include the cells ce_1 and ce_2 . Also notice how the call for path \mathcal{P}_2 uses an inverted input edge with respect to \mathcal{P}_1 . After the initialization, we go into a loop with I iterations that recomputes output capacitance and input slew before each call to Beam Search. After the loop, the return value are the last two paths obtained.

4.6.4 Configurable Path Algorithm

Configurable paths are intended for substituting regular paths in the Dual and Simple RO. This algorithm is designed for substituting the calls to the basic algorithm in those problems to calls to this algorithm. As explained in the formalization for this problem, we will split it into one problem per multiplexer, plus one more for the fixed delay path.

Algorithm 4: Predefined partial path Beam Search algorithm

input : A set of gates G , a set of target delay T , input slew S_0 , input edge E_0 , output capacitance Q_{out} and a partial path P_p that is appended at the end of the path

output: A path of gates satisfying the basic problem constraints which last gates correspond to the ones in P_p

begin

```

   $Cand \leftarrow \emptyset$  //Set of candidates
   $Test \leftarrow \emptyset$  //Set of paths tested so far
  foreach  $g \in G$  do
     $\mathcal{P} \leftarrow \{P = \{g\}, S_0, E_0, Q_{out}\}$  //Path with only one gate
     $\mathcal{P}_{ext} \leftarrow \text{appendPaths}(\mathcal{P}, \mathcal{P}_p)$ 
     $cost \leftarrow \text{costFunction}(\mathcal{P}_{ext}, T)$ 
    /* The cost function evaluates the path with the partial path
       appended */
     $Test \leftarrow \{Test | \{\mathcal{P}, cost\}\}$  //We add the path without the partial path
   $Cand \leftarrow \text{getBestCandidates}(Test, K, T)$  //Assign best  $K$  candidates to Cand
   $L := 2$  //Minimum length for paths that do not satisfy constraints
  while not  $\text{allPathsSatisfyConstraints}(Cand)$  do
     $Test \leftarrow \emptyset$ 
    foreach  $c \in Cand$  do
       $Test \leftarrow \{Test | c\}$ 
      foreach  $g \in G$  do
         $\mathcal{P} \leftarrow \text{appendCell}(c, g)$  //Append gate  $g$  to the end of the path from  $c$ 
         $\mathcal{P}_{ext} \leftarrow \text{appendPaths}(\mathcal{P}, \mathcal{P}_p)$ 
         $cost \leftarrow \text{costFunction}(\mathcal{P}_{ext}, T)$ 
         $Test \leftarrow \{Test | \{\mathcal{P}, cost\}\}$ 
       $Cand \leftarrow \text{getBestCandidates}(Test, K, T, L)$ 
      /* Assign best  $K$  candidates to Cand. Unless a path already
         satisfies constraints, the candidates selected from  $Test$  must
         have at least length  $L$  */
     $L := L + 1$ 
   $best \leftarrow \text{appendPaths}(\text{bestCandidate}(Cand), \mathcal{P}_p)$ 
  return:  $best$  //Return the path with the appended partial path

```

Algorithm 6 shows the pseudocode for this problem. The inputs are similar to the basic algorithm, with a few additions: the number of multiplexers M , a vector of coefficients for the delay Δ , both multiplexer configuration m_f for the fixed delay and m_l for the long path and a partial path P_p for the case of Dual RO in which we will need to include C-Element cells.

We may consider two parts on this algorithm. First we create the fixed delay path, which includes a sequence of M multiplexers m_f at the end of the path. After this, we create M more paths, each with its target delay multiplied by the coefficient in Δ .

Algorithm 5: Dual Ring Oscillator Beam Search algorithm

input : A set of gates G , a set of target delay T , input slew S_0 , input edge E_0 , output capacitance Q_{out} , amount of loop unrolls I and two C-Element cells ce_1 and ce_2

output: A path of gates satisfying the Dual Ring Oscillator requirements

begin

```

 $\mathcal{P}_1 \leftarrow \text{predefinedPartialPathBeamSearch}(G, T \cdot 0.5, S_0, E_0, Q_{out}, \{ce_1\})$  //Initial
path 1
 $\mathcal{P}_2 \leftarrow \text{predefinedPartialPathBeamSearch}(G, T \cdot 0.5, S_0, \bar{E}_0, Q_{out}, \{ce_2\})$  //Initial
path 2
 $Q_1 \leftarrow \text{getInitialCapacitance}(\mathcal{P}_1)$ 
while  $I > 0$  do
     $Q_2 \leftarrow \text{getInitialCapacitance}(\mathcal{P}_2)$ 
     $Q \leftarrow Q_1 + Q_2$ 
     $S \leftarrow \text{getFinalSlew}(\mathcal{P}_2)$ 
     $\mathcal{P}_1 \leftarrow \text{predefinedPartialPathBeamSearch}(G, T \cdot 0.5, S, E_0, Q, \{ce_1\})$ 
    //Current path 1
     $Q_1 \leftarrow \text{getInitialCapacitance}(\mathcal{P}_1)$ 
     $Q \leftarrow Q_1 + Q_2$ 
     $S \leftarrow \text{getFinalSlew}(\mathcal{P}_1)$ 
     $\mathcal{P}_2 \leftarrow \text{predefinedPartialPathBeamSearch}(G, T \cdot 0.5, S, \bar{E}_0, Q, \{ce_2\})$ 
    //Current path 2
     $I := I - 1$ 
return:  $\mathcal{P}_1, \mathcal{P}_2$ 

```

The slew and capacitances for these paths can be obtained from the fixed path that we previously created. Finally, we return the set of paths.

4.7 Cost Function

In the previous section we talked about the algorithms that we implement for solving different variations of the problem. In every algorithm we mentioned a cost function that will somehow evaluate the quality of a path, yet anywhere did we explain anything about it. This section is dedicated to the cost function, as it is not straightforward to decide which is the best way to implement it. Furthermore, there was a lot of trial and error in the design for this function, so we will not be able to formally defend some of the decisions that we took.

In Section 4.2 we talked about different possibilities for the optimization goal, yet stated that it was not clear which was preferable. At first sight, it may make sense to implement the cost function directly from those optimization goals and use whichever seems preferable for our practical application. In our tests, we found that, often, directly applying

Algorithm 6: Configurable Path Beam Search algorithm

input : A set of gates G , a set of target delay T , input slew S_0 , input edge E_0 , output capacitance Q_{out} , amount of multiplexers M , vector of coefficients Δ with size $M + 1$, multiplexer cell for fixed path m_f and multiplexer cell for long path m_l and a partial path P_p that is appended at the end of the path

output: A configurable path of gates satisfying the constraints for configurable path problem

begin

```

   $Mux \leftarrow [m_f \times M]$  //A path with a sequence of  $M$  cells of type  $m_f$ 
   $\mathcal{P}_p \leftarrow \text{appendPaths}(Mux, P_p)$ 
   $\mathcal{P}_f \leftarrow \text{predefinedPartialPathBeamSearch}(G, T \cdot \Delta[1], S_0, E_0, Q_{out}, \mathcal{P}_p)$  //Fixed path
   $n := \text{lengthOfPath}(\mathcal{P}_f)$ 
   $Paths \leftarrow \{\mathcal{P}_f\}$  //Set of paths
   $i := 2$ 
  while  $i \leq M$  do
     $S_i \leftarrow \text{getSlew}(\mathcal{P}_f, n - (M + 2 - i))$ 
     $E_i \leftarrow \text{getEdge}(\mathcal{P}_f, n - (M + 2 - i))$ 
     $Q_i \leftarrow \text{getCapacitance}(\mathcal{P}_f, n - (M + 1 - i))$ 
     $\mathcal{P}_i \leftarrow \text{predefinedPartialPathBeamSearch}(G, T \cdot \Delta[i + 1], S_i, E_i, Q_i, \{m_l\})$ 
    //Long path  $i$ 
     $Paths \leftarrow Paths \cup \{\mathcal{P}_i\}$ 
     $i := i + 1$ 
  return:  $Paths$ 

```

those functions would yield low quality solutions with respect to the goals that we were trying to optimize. Besides the basic optimization goals, we also need in some occasions to minimize differences in slew. This requires the use of a different cost function that has to take into account slew as well as delay.

The following subsections will describe implementations for the cost function that were explored and found to give best quality for all optimization goals overall.

4.7.1 Variance over the normalized delay

The heuristic that implements this cost function is, as the name suggest, computed as the variance of the normalized delay. The function is defined as:

$$Cost = \sum_{c \in C} (\lambda_c - Avg)^2$$

$$Avg = \frac{(\sum_{c \in C} \lambda_c)}{|C|}$$

With C representing the set of corners, $\lambda_c = \frac{d_c}{t_c}$ the normalized delay for corner c , d_c the current accumulated delay of the path for corner c , and t_c the target delay for corner c .

Note that this function will penalize gates that give unbalanced normalized delays. A good visual representation of what this function tries to accomplish can be seen at Figure 4.7.

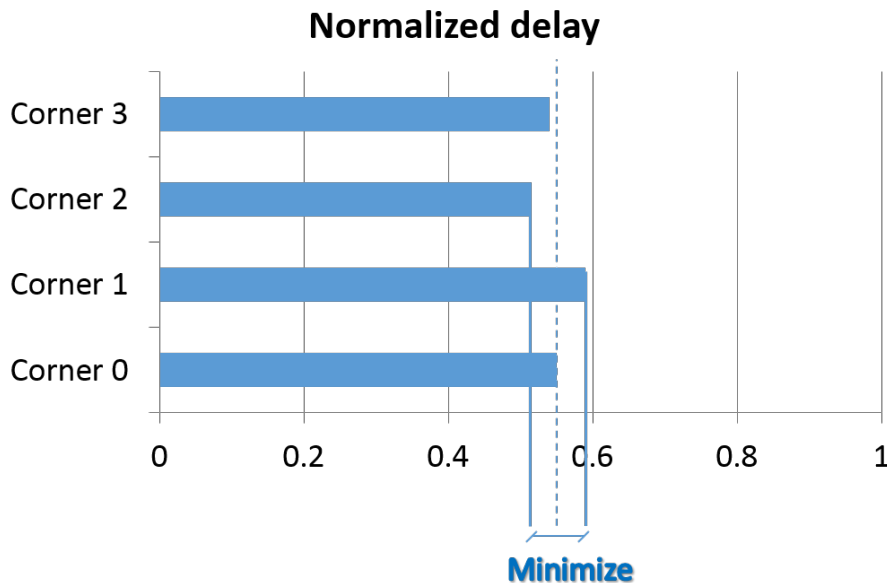


FIGURE 4.7: Graphical representation of the *variance over the normalized delay* heuristic. The y-axis represents different corners. The x-axis represents normalized delay values. The function tries to minimize the differences in normalized delays.

4.7.2 Squared normalized distance

This heuristic function uses the concept of *distance* of the delay to the target delay, penalizing cells that have, for some corner, a long difference between the normalized delay and 1. This function is simply defined by:

$$Cost = \sum_{c \in C} (\lambda_c - 1)^2$$

With C representing the set of corners, $\lambda_c = \frac{d_c}{t_c}$ the normalized delay for corner c , d_c the current accumulated delay of the path for corner c , and t_c the target delay for corner c .

A visual representation of this function is shown in Figure 4.8, in which we see how the heuristic tries to minimize long distances.

It should be noted that, in the same way that this function penalizes the smallest normalized delays, it also incentives big normalized delays. This may sometimes cause unbalances that may affect the quality of the solution. A way of solving this problem is to redefine the function in the following way:

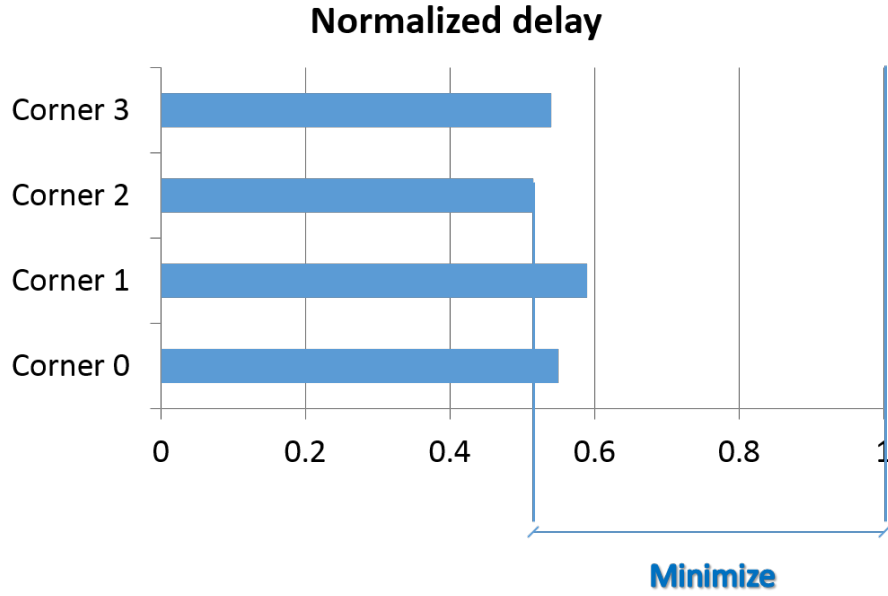


FIGURE 4.8: Graphical representation of the *squared normalized distance* heuristic. The y-axis represents different corners. The x-axis represents normalized delay values. The function tries to minimize the distance to the target delay.

$$Cost = \sum_{c \in C} (\lambda_c - \frac{n_i}{n})^2$$

With n_i being the current length of the path and n the total expected length for the path, as defined by the input.

This alternative version is only possible when solving the extension of the problem that includes a specific length for the final solution. The advantage of this modification is that it allows for a gradual construction of the path. For example, for $n = 20$ the heuristic will incentive the selection of gates that increase the delay for all corners for about $1/20$ of the target delay. In fact, experimental trial and error shows that using this heuristic is a far better way to implement the extension of the problem that specifies a length instead of adding a new constraint.

4.7.3 Mixed heuristic

Another possibility for the cost function is to mix the two previous heuristic functions. The first heuristic balances the delay of the corners, while the second heuristic penalizes differences with respect to the target delay. We experimentally found that it was better to use the *variance over the normalized delay* heuristic for the most part of the algorithm and then, in the final iterations, switch over to the *squared normalized distance* function.

The mixed heuristic raises a new problem: when is best to change heuristics. In our implementation, this change is done when the first complete solutions (that is, solutions that satisfy all the constraints) appear in the candidate set, but other options are possible. For example, this switch could be done when all the candidates reach some threshold either in length or normalized delay.

4.7.4 Slew heuristic

Some of the extensions previously discussed require the optimization of slew as well as the delay. Having two optimization goals is tricky, especially when they are uncorrelated. Each one may pull from different directions, gravely resenting the quality of the solution.

Luckily, the slew requirements only apply for the last cell. We can make use of this and only optimize for slew in the last stage. Thus, the last cell of each path will have a different cost function that will penalize slews different than the target slew TS .

A function that was found to work well is defined as:

$$Cost = DelayCost + \sum_{c \in C} \left(\frac{S_c}{TS_c} - 1 \right)^2$$

With $DelayCost$ being the cost from the heuristic for the delay, S_c the delay for corner c and TS_c the target slew for corner c .

4.8 Summary

In this chapter we formalized and analysed the problem that conforms the main focus of this work. We then discussed different approaches for solving it. Considering this, we decided to use a metaheuristic algorithm called Beam Search for constructing solutions. This approach surrenders optimality in the solutions in favour of reasonable runtimes. Corresponding sections then describe different implementations of the Beam Search algorithm for the diverse versions of the problem.

An important idea from this chapter is the complexity of the problem. We do not know a way of solving it optimally and thus different approaches to its solutions may yield different quality levels. The next chapter will discuss how accurate the solutions are by providing experimental results.

Chapter 5

Experimental results

5.1 Introduction

In this chapter we will present experimental results from our implementation of the algorithms described in Chapter 4. Each of the following sections will be devoted to one aspect of the algorithm, such as comparison between heuristics or behaviour of different schemes. There will be, for each section, a simple description of the experiment, followed by the results and a brief discussion of them.

For the experiments, we used circuits from the IWLS-2005 benchmark, as it is widely used in research papers. In particular, we selected circuits from the *itc99* subset of IWLS-2005. Some information about these circuits used can be found in Table 5.1. From those circuits, we executed the STA in order to obtain *target delays* for our experiments. We used the following tools from Synopsys®:

- Synthesis with *Design Compiler*. This tool obtains, from the RTL design, a netlist of the circuit mapped to a specific library.
- Place and route with *IC Compiler*. Based on a previously obtained netlist, this tool maps cells to positions on the chip (placement) and to the wires that interconnect them (routing).
- Static Timing Analysis with *Prime Time*. Given a netlist, possibly after placement and routing (as is our case), this tool executes a STA and returns timing information for all the corners of the library. This is the information that we use for our experiments.

Name	Gates ¹	Primary Inputs	Primary Outputs	Flip-Flops
b01	49	2	2	5
b02	28	1	1	4
b03	160	4	4	30
b04	737	8	11	66
b05	998	1	36	34
b06	56	2	6	9
b07	441	1	8	49
b08	183	9	4	21
b09	170	1	1	28
b10	260	11	6	17
b11	770	7	6	31
b12	1,076	5	6	121
b13	362	10	10	53
b14	10,098	32	54	245
b14_1	6,900	32	54	245
b15	8,922	36	70	449
b15_1	13,098	36	70	449
b17	32,326	37	97	1,415
b17_1	39,665	37	97	1,415
b18	114,621	36	23	3,320
b18_1	108,482	36	23	3,320

TABLE 5.1: Basic description for benchmarked circuits.

In order to both run the experiments and synthesize the benchmark circuits we need to use standard cell libraries. We selected three libraries from three different technology nodes and foundries. In particular:

- Synopsys 32nm Generic Library. This library, obtained from Synopsys®, is a educational library for the 32 nm technology node and uses 10 corners. It is not suitable for actual silicon synthesis, but it adds diversity to our tests.
- UMC (United Microelectronics Corporation) standard cell library for the 65 nm node with 11 corners. This is a commercial library that can be used for chip manufacturing.
- TSMC (Taiwan Semiconductor Manufacturing Company) standard cell library for the 40 nm node with 15 corners. Another commercial library from one of the biggest semiconductor foundries in the world, also suitable for manufacturing.

For the experiments that follow in this chapter, we decided to split the circuits from the benchmark into two groups: circuits with *long critical paths* and *short critical paths*.

¹Number of gates from original source. This can be different when synthesising with our tools and may change for each library.

The reason for this classification is that most of the circuits from the itc99 benchmark are very small. This affects the algorithm, as smaller circuits do not give opportunity to adapt to delay. More importantly, in practice we are interested in longer critical paths that would appear in a processor or an ASCI (Application-Specific Integrated Circuit) such as the AES circuit that we analyse latter in this chapter. For that reason, we will consider that *short critical path circuits* are those that have less than half the delay of the AES circuit.

5.2 Heuristics and libraries

As can be seen by the multiple schemes and different alternatives for the problem and the algorithm, there is a big number of possible configurations to analyse. In order to keep things simple, we will devote this first section to compare both heuristics and libraries. After this section, we will focus on only one heuristic and one library.

All the results shown on this section use the Simple Ring Oscillator scheme with two loop unrollings. We use this scheme because it does not have input fixed cells, such as multiplexers, flip-flops or C-Elements. The nomenclature for naming heuristics is as follows:

- **H1**: Squared normalized distance, introduced in Section 4.7.2.
- **H2**: Variance over the normalized delay, introduced in Section 4.7.1.
- **H3**: Mixed Heuristic, introduced in Section 4.7.3

We will present most of the results in one type of line chart, such as the one presented in Figure 5.1. These charts show the normalized error on the *y-axis* and the circuits in the *x-axis*. The normalized error is the percentage that the delay of the resulting path *overshoots* the target delay. For example, a normalized error of 0% means that the delay of the path is equal to the target delay, while a 1% indicates that it is 1% greater. In particular, normalized error for one corner is computed by the following expression:

$$\frac{d_c - t_c}{t_c} \times 100$$

with d being delay, t target delay and c the corner.

For simplicity, we will show the averaged normalized error for all the corners. Each point in the line chart represents the normalized error for some particular circuit from the benchmark. We will often categorize these charts by library or scheme, as indicated

by a legend. Circuits in the x -axis will always be ordered by the average delay of the circuits. In particular, circuits with smaller delays are on the left, and circuits with bigger delays are on the right. Finally, these charts are very often split by a vertical line. This line always show, on the left, circuits with *short delays* and, on the right, circuits with *long delays*. This distinction is important, as we are mostly interested in circuits with long delays. In fact, short delay circuits are only present for reference, as the algorithm is not designed to work well with these.

Moving into the results themselves, Figure 5.1 shows a line chart for the *Squared normalized distance* heuristic for libraries TSMC, UMC and Synopsys. Note how circuits on the left of the partition have usually worse results, as the algorithm has more trouble tracking delays. Figure 5.2 shows a zoomed version of the previous chart, in which we can see more accurately the delay of the longer circuits. While there are differences between libraries, they are in the same range of errors, between 5 and 10% in average.

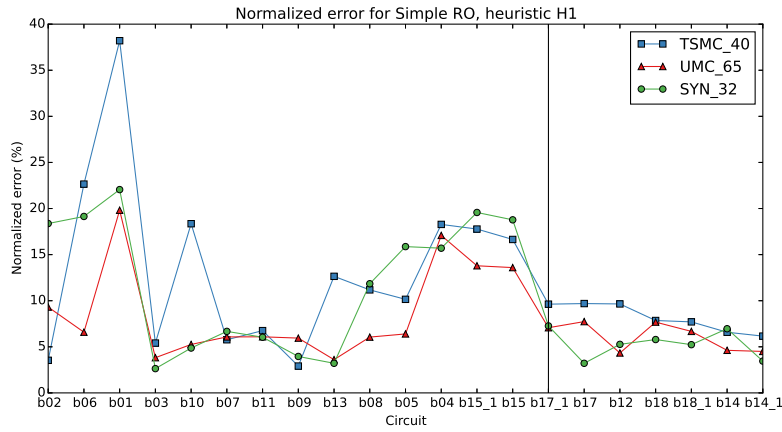


FIGURE 5.1: Results for the Squared normalized distance heuristic for all the libraries.

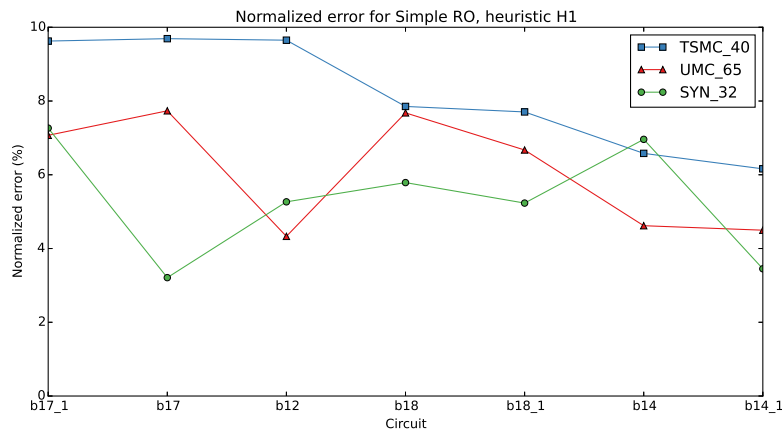


FIGURE 5.2: Results for the Squared normalized distance heuristic for all the libraries. Only circuits with long critical paths are shown.

Results for the *variance over the normalized delay* heuristic are depicted in Figure 5.3. It can be seen how, in this case, results are more stable for libraries TSMC and Synopsys, yet UMC shows more spikes in the short circuits. In any case, by looking in detail to the more interesting long delay circuits in Figure 5.4, it is obvious that results are much better than the previous heuristic.

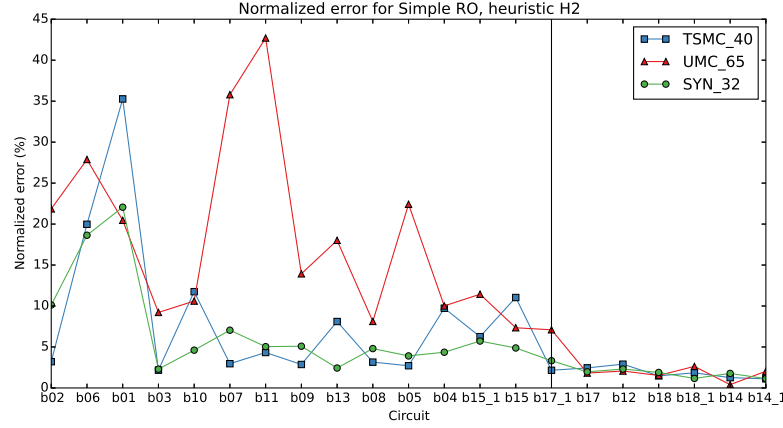


FIGURE 5.3: Results for the Variance over the normalized delay heuristic for all the libraries.

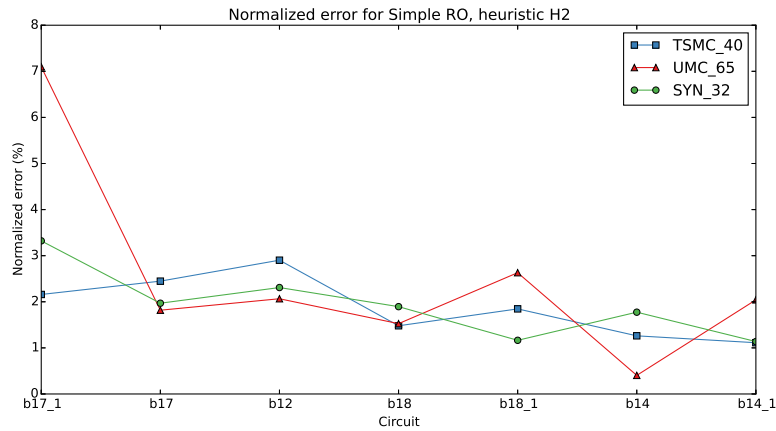


FIGURE 5.4: Results for the Variance over the normalized delay heuristic for all the libraries. Only circuits with long critical paths are shown.

Results for the last heuristic that we analyse, the mixed heuristic, can be seen in Figure 5.5. This heuristic tries to combine the benefits of the previous two, and it seems to succeed in improving results. As it can be seen, errors are much lower for all circuits, including the short ones. More importantly, long delay circuits, shown in Figure 5.6, have very small errors overall, regardless of the library.

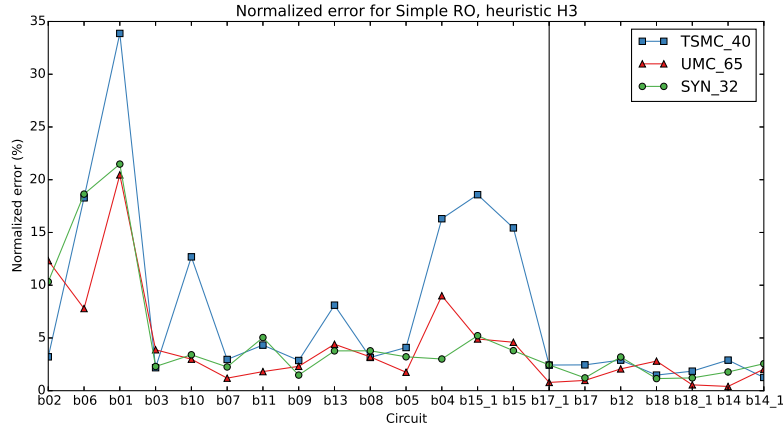


FIGURE 5.5: Results for the mixed heuristic for all the libraries.

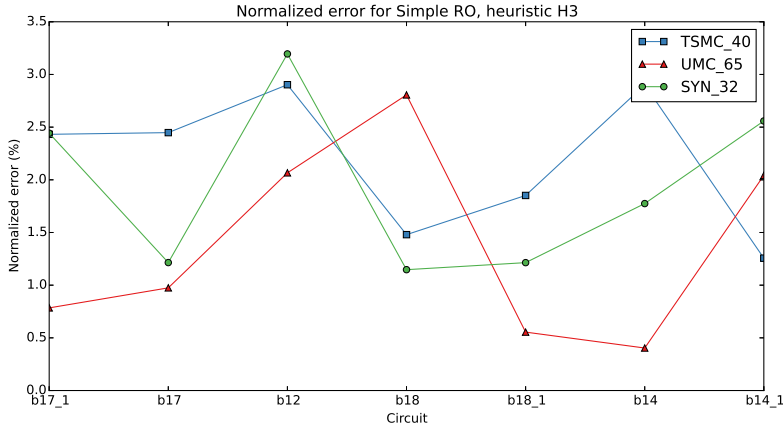


FIGURE 5.6: Results for the mixed heuristic for all the libraries. Only circuits with long critical paths are shown.

5.3 Simple and Dual Ring Oscillators

In this section we will discuss results for Simple and Dual Ring Oscillators. For simplicity, we will only compare the schemes with the UMC library and the mixed heuristic.

Figure 5.7 shows a line chart comparing Ring Oscillator schemes along the benchmark. We can see how the Dual RO has consistently better results than the Simple scheme, with very few exceptions. This can be achieved thanks to the more relaxed constraints for the Dual RO with respect to the inverted edge.

For the more interesting long delay circuits we can look at a zoomed view in Figure 5.8. It can be seen how in these circuits the Dual scheme has average errors always under the 1.5% mark.

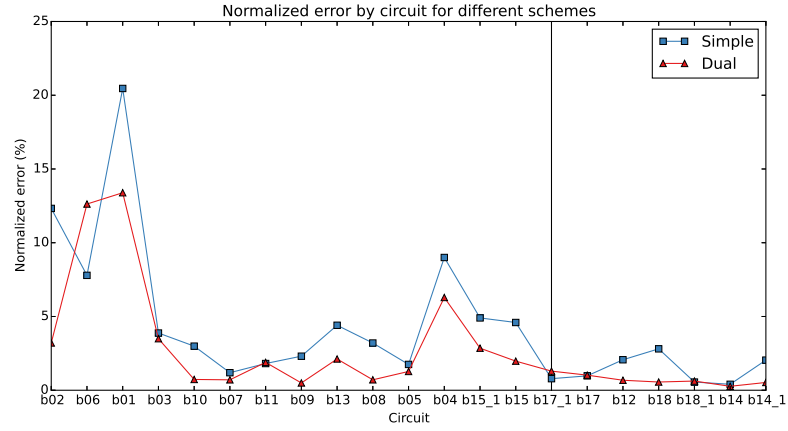


FIGURE 5.7: Comparison between Simple and Dual RO schemes for library UMC and mixed heuristic.

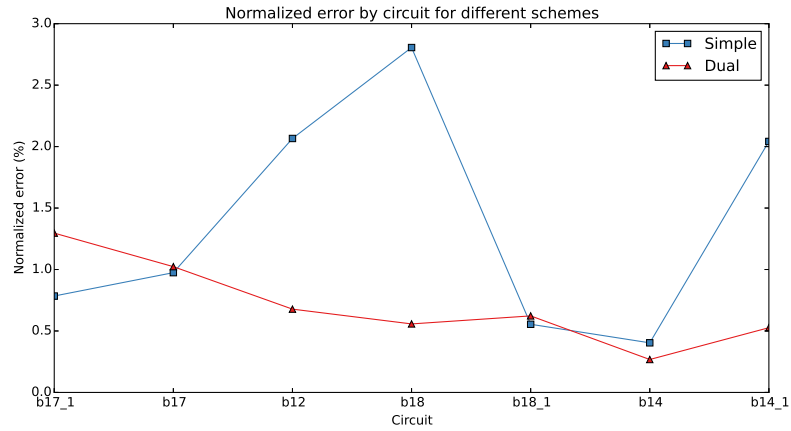


FIGURE 5.8: Comparison between Simple and Dual RO schemes for library UMC and mixed heuristic. Only circuits with long critical paths are shown.

Besides the results shown here, there is something important to point out. The Dual RO needs, as input parameter, a C-Element cell. There are lots of ways of implementing such a gate with standard cell libraries, and this implementation may have an impact on the solution. For this experiment we selected one option and stuck with it for the sake of consistency. This, nonetheless, puts the dual scheme at a disadvantage – the Dual RO has more room for tweaking and solutions may be further improved by playing with different C-Element cells and sizes.

5.4 Monitor

The Monitor is another of the schemes that were discussed in previous chapters. In this case, the signal does not oscillate, so there are less constraints for the problem. Nonetheless, this scheme is actually build by concatenating two paths: a regular path

and a thermometer path. Remember that the peculiarity of the thermometer path is that it has a fixed number of gates and there is a Flip-Flop attached to each gate. We used, in particular, 16 Flip-flops for the thermometer for all the benchmarks.

Figure 5.9 shows results for the benchmark with the UMC library and the mixed heuristic. Differences between long and short delays are very noticeable in the case of the monitor.

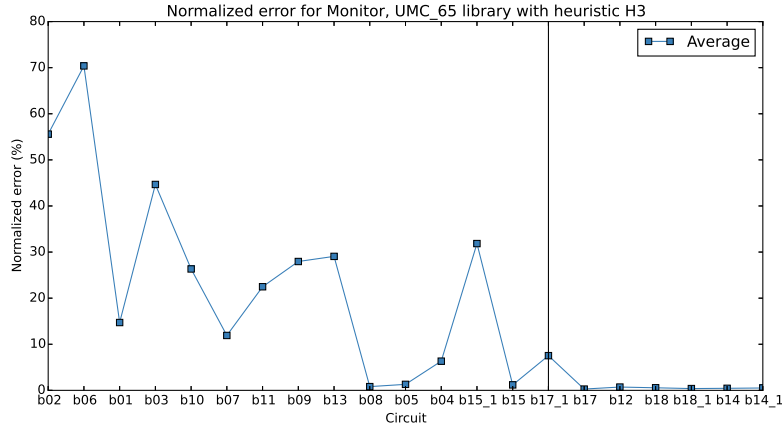


FIGURE 5.9: Results for Monitor scheme, library UMC and mixed heuristic.

On the one hand, circuits with short delays have a very high error rates. Note that, in order to construct the monitor, we force a minimum length of 16 cells with Flip-Flops. Most of these paths are actually built overly long due to the thermometer length constraints and, for some of them, the thermometer makes up the whole length of the path.

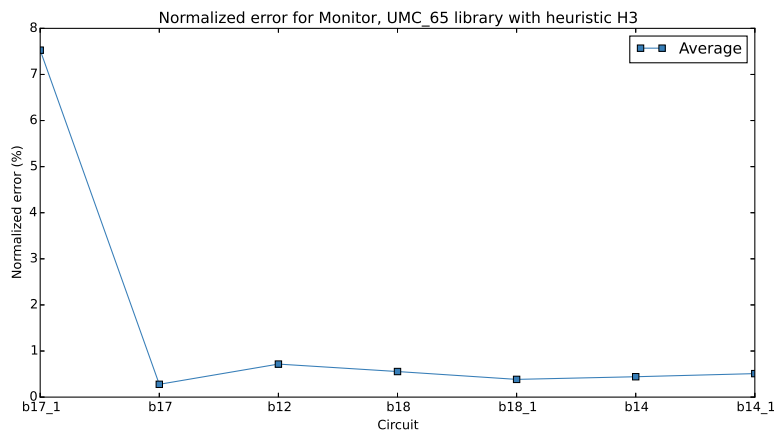


FIGURE 5.10: Results for Monitor scheme, library UMC and mixed heuristic. Only circuits with long critical paths are shown.

On the other hand, circuits with long delays have very low errors rate, as can be better seen in Figure 5.10. We can see how, in general, results stay below 1% error, as it is

easier to track variability if we ignore the inverted path. Even so, the problem is solved by first generating two different paths and then concatenating them. Note that the slew between the two paths is not taken into account when concatenating them. This has usually a very small impact. Even so, big slews at the end of the first path may potentially change the delay on the second path. This is reflected in the b17_1 circuit, which reaches an average error of around 7%.

Summarizing, the Monitor scheme tends to give very good results for circuits with long delays, which are the ones that we are interested in. Shorter paths would need fewer Flip-Flops, thus affecting the capacity to accurately monitor the state of the circuit.

5.5 Configurable Ring Oscillator

Configurable Ring Oscillators are the last of the schemes discussed in Chapter 3 that remains to be tested. In this section we will show results for configurable paths for both types of Ring Oscillators. Again, the algorithm was ran with the UMC library and the mixed heuristic was used.

In the case of configurable paths, each circuit has now 2^n target delays, with n being the number of multiplexers. We change for this section the way to show results in order to simplify things. Average values are now computed by averaging all the normalized errors for all the possible configurations in every corner.

Figure 5.11 shows average values for the Configurable Simple RO with up to three multiplexers. The first thing that should be noted is that increasing the number of multiplexers also increments the average error, as expected. Similarly to what happened in the case of the Monitor, the multiplexers are cells fixed by the input. In this case, multiplexer cells have often large delays and thus are inappropriate for tracking delays in small circuits. For that reason, we can also see how as we move to the left of the chart to circuits with smaller delays, solutions deteriorate much more than in previous schemes.

A zoomed version of the previous chart into the circuits with longer delay can be seen in Figure 5.12. It can be seen how errors are much greater than schemes without multiplexers, especially for 3 multiplexers.

Results for the Configurable Dual RO are shown in Figure 5.13. Similar trends to the Simple RO case can be seen for this scheme. Even so, error values are in general lower, as was the case without multiplexers.

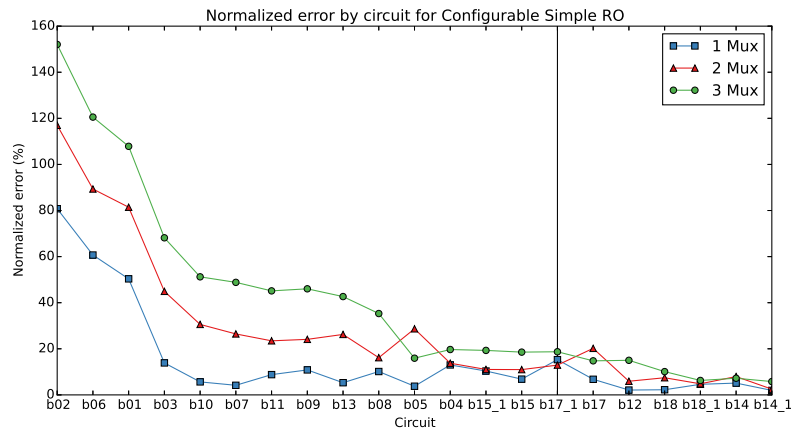


FIGURE 5.11: Results with the mixed heuristic for the Configurable Simple RO with up to 3 multiplexers.

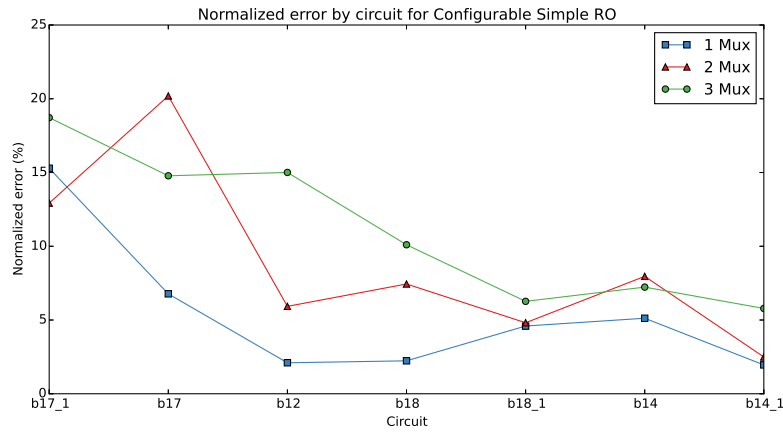


FIGURE 5.12: Results with the mixed heuristic for the Configurable Simple RO with up to 3 multiplexers. Only circuits with long critical paths are shown.

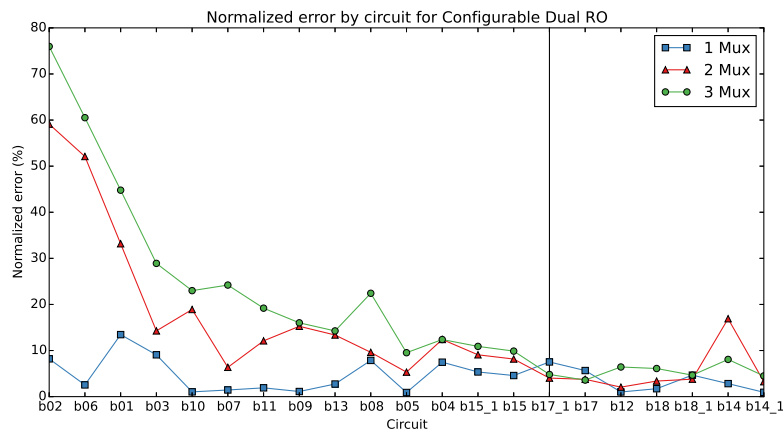


FIGURE 5.13: Results with the mixed heuristic for the Configurable Dual RO with up to 3 multiplexers.

Concluding results for the Configurable Paths, Figure 5.14 depicts only circuits with long delays. Again, with longer delays the algorithm accomplishes smaller errors, especially with one or two multiplexers.

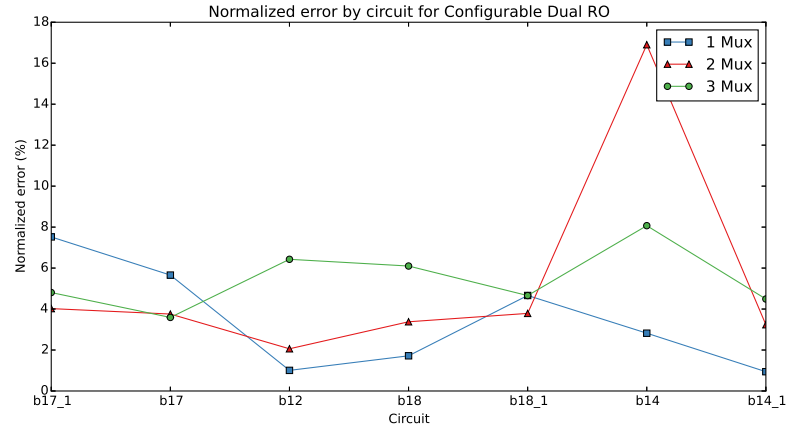


FIGURE 5.14: Results with the mixed heuristic for the Configurable Dual RO with up to 3 multiplexers. Only circuits with long critical paths are shown.

As noted by previous sections, these benchmarks were done all with the same parameters for consistency reasons, yet this algorithm greatly benefits from parameter tweaking for each instance. This is especially important for configurable paths, in which the decision of the multiplexer cell has a huge impact in the final delay. Circuits with longer delays are less influenced from the selection of multiplexer and thus some of the bigger circuits yield reasonable results. Even so, parameter tuning is necessary in order to obtain good results for configurable paths.

5.6 Place and Route experiment

Until now, we have been making one simplification for the results shown. While the target delays that we used come from circuits after the place and route stage, the delays that we present are obtained based solely on the wire models of the libraries. These wire models approximate wire capacitances, yet an actual placement and routing is still necessary for a correct Static Timing Analysis. The reason for this omission and simplification is that it would be necessary for the algorithm to execute itself the placement and routing when generating the path. While this is something interesting for future work, this characteristic is not currently implemented.

Nonetheless, it is possible to overcome this simplification by making use of an iterative process. In particular, this can be made by generating a set of very similar target delays, multiplied by some small coefficients. For example, if the target delay is T , we create

several target delays $\epsilon \cdot T$, with ϵ being values close to 1. Each of these target delays may then be solved and later processed through a place and route tool and a STA analysis. Of these results, the best solution that does not violate constraints is selected.

We now present an example of a Simple Ring Oscillator with timing results after a correct placement and routing. We selected for this experiment an AES (Advanced Encryption Standard) circuit. This case is also an example of results that may be obtained by fully tweaking the algorithm to optimize the solution, even if some precision is lost due to the placement and routing.

Figure 5.15 shows the delay for each corner of the UMC library. The delay for the AES circuit is represented by the white bars. On top of them, the delay of the Simple RO after place and route is shown in blue. As it can be seen, the difference between delays is actually very narrow for all corners. A better view of these differences may be seen in Figure 5.16. This last figure represents the normalized error for each corner of the library.

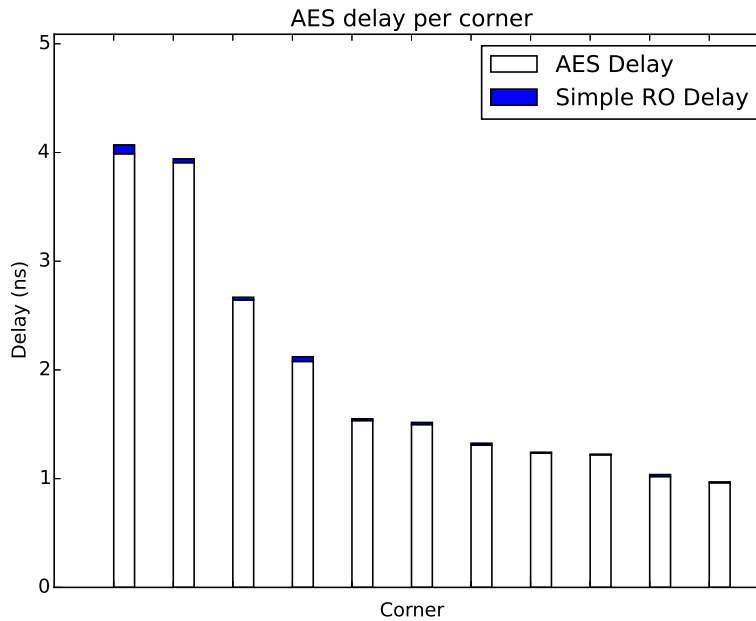


FIGURE 5.15: Simple Ring Oscillator delay values for an AES circuit. The timings of the Ring Oscillator were taken after place and route.

The main objective of this experiment was to show that, despite the limitations of the algorithm with respect to placement and routing, it is still possible to use it in real-life scenarios with the help of some iteration in the process. While this iterative process is not ideal, it must only be done once per design. Furthermore, results obtained are close in quality terms to those that were obtained without any placement and routing.

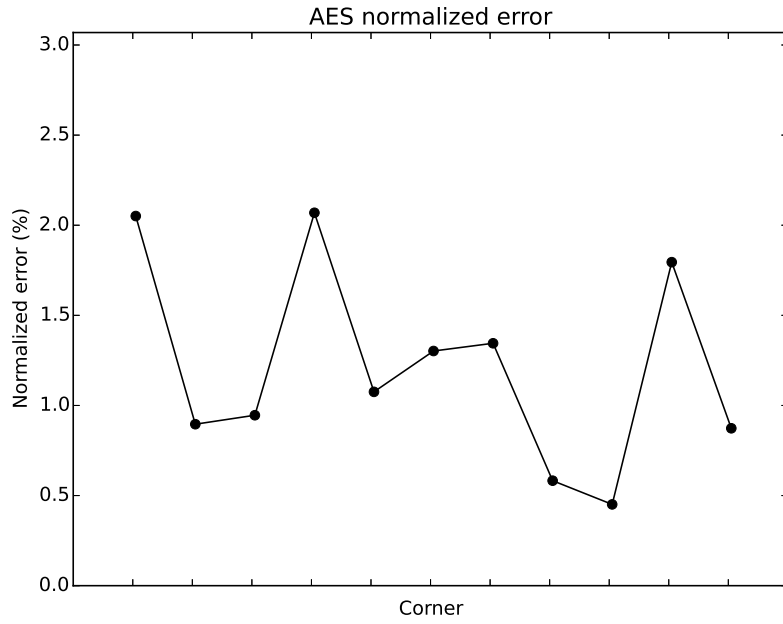


FIGURE 5.16: Error per corner for the AES Ring Oscillator.

5.7 Conclusions

Conclusions for each experiment have mainly already been discussed at the end of each section. we now present a summary of the basic conclusions:

- The library used may impact the accuracy of the algorithm.
- Heuristics used for the cost function gravely affects the quality of the solutions found. The first heuristic yields results consistently worse than the other two, especially than the mixed heuristic.
- Adequately tweaking the algorithm for each input is very important in order to obtain good results. Most of the results shown here would have presented better values have we tweaked them individually.
- Smaller target delays negatively affects quality of solutions. This is especially significant when using schemes that need input fixed cells such as Monitors and, most notably, configurable paths.
- The algorithm is valid for usage in real-life scenarios where placement and routing of the Ring Oscillator is needed.

Chapter 6

Conclusions

6.1 Summary

Since the introduction of the first transistors, the semiconductor industry has been profiting from a continuous technology node shrinking every couple of years. While this has allowed an exponential improvement of the characteristics of silicon devices for over fifty years, we are now facing a slowdown and, potentially, a complete stop of this trend.

During the last years, an important part of the improvements in semiconductor circuits comes from clever tricks and techniques rather than just technology improvements at transistor level. This will probably have more and more impact in the coming years, as finding new ways of better exploiting our current technologies becomes a necessity if we want to keep improving.

In this document we identified an opportunity for improvement in the way circuits are designed. In particular, recent technology nodes require huge guard band margins at design stage in order to overcome the variability at which circuits are subject. We proposed a clock scheme that aims to substitute common PLLs, in order to dynamically adapt timing constraints of a circuit to the actual requirements at each moment.

The main component of this scheme is a Ring Oscillator which suffers from the same variability sources than the rest of the system. This Ring Oscillator has the task of generating the clocking signal, so any changes that affect the way a circuit behaves will similarly affect the clock signal. In particular, we relax the periodicity of the clock signal so that it dynamically changes over time in order to adapt to variability.

The Ring Oscillator that we introduce in this work may be built in different ways. We presented two main ways: the Simple RO and the Dual RO. Furthermore, these circuits

may be enhanced with the possibility of dynamically configuring their length. This is what we called *configurable paths*. Configurable paths are constructed with the help of multiplexers, which allows the signal to *skip* sequences of gates in order to reduce the length of the path.

In order to give feedback about the actual state of variability of a circuit and allow for tweaking configurable paths, we also presented *monitor* circuits. This kind of circuits is not novel to this work and has been studied in other research papers, such as the ones we introduce in Section 2.5. Even so, we design them using a similar techniques to the ones we use for the Ring Oscillators.

The design of all these circuits was approached in an algorithmic way. First, we formalized the problem with a set of constraints and optimization goals. With a formalization we were able to analyse the difficulty of the problem and found that it was in the NP-hard complexity class. This, coupled with the dimensions of the input parameters that we need to solve, made us look for fast algorithms in exchange for optimality. We then presented a set of algorithms to solve the problem and each of its variations.

Finally, we tested an implementation of the algorithm with a benchmark of different circuits and libraries. The results showed that, for circuits with sufficiently long critical paths, designing a Ring Oscillator with almost identical characteristics to the circuit is possible, often with errors no bigger than a 1% or 2%. The Dual RO seemed to yield better results in general, thanks to the more relaxed constraints.

When considering configurable paths, the algorithm is not always able of maintaining accurate results and errors of 5-10% are more common. This is especially important with bigger number of multiplexers. Nonetheless, it is also possible to obtain configurable paths by using multiplexers *outside* the Ring Oscillator, in which case there is no penalization to accuracy. The limitation of this last approach is a linear cost in area over the number of possible configurations.

With respect to the monitor, results show that it is easier to track variability when no oscillating signal is involved. In fact, for most of the circuits that we were interested in, errors were often well under the 1% error rate.

As a final experiment, we also showed that the algorithm is also able to keep track of variability even after placement and routing. A Simple RO was designed to work for an AES circuit and was latter placed and routed with Synopsys® tools. Results show errors no bigger than 2% per corner in the worst cases and, very often, errors under 1%.

With all this, we can conclude that this is a reasonable approach to the design of the Ring Oscillators. Even so, further improvements are required in order to reduce the

tweaking dependency and, especially, to improve results when designing configurable paths.

6.2 Future Work

In this document we presented a problem and showed that the approach that we took to solve it was reasonable. Nonetheless, there are numerous ways to improve results from the algorithm, as well as simplify the design stage.

During this work we prioritized functionality over quality. As such, we designed different schemes and analysed them, yet we somehow left aside different algorithms testing. For that reason, possibly one of the most important things that remains to be tested are the potential improvements that may be achieved by using different algorithms.

One of the main drawbacks of our algorithm is its tendency to explore in depth only a few solutions, rather than exploring a more diverse range of solutions more superficially. While this is a good approach in some cases, it still has the risk that a wrong decision at an early stage cannot be corrected latter. There remains as future work to test different metaheuristic algorithms that may focus in the opposite and yield a wider range of solutions.

A strategy that was discussed in the document but not tested is the usage of optimal algorithms for solving a simplified version of the problem. While we proved that optimal algorithms may not be possible in polynomial time for solving the complete problem, finding a good simplification and then solving it optimally may still yield good results. In fact, as we discussed in Chapter 4, there has been work done in that direction for similar problems while designing monitors. Probably, a first good option for this approach would be to use ILP. The description as a linear programming problem seems to come very naturally from the formal definition that we presented in Chapter 3.

Another drawback of the approach presented here is the manual tweaking process. In particular, it is undesirable to leave to the user the responsibility of selecting multiplexer cells for configurable paths or C-Element cells for the Double RO. This is not only a problem of difficulty of usage, but can also be a problem of accuracy. The algorithm should automatically decide which are the best cells to use in these situations, possibly from a set of candidates that depends on each library.

Perhaps in part as an effect for letting the user select the multiplexer gates, delays from configurable paths often fail to accurately track variability. This is something that may be solved by incorporating into the optimization problem the selection of multiplexer

cells, but that is probably not enough. Configurable paths as we described here are subject to very harsh constraints due to slew propagation. Because of this, it may be possible that accurate solutions do not even exist. More research in this topic will be necessary to either find a way of obtaining better solutions or deciding for a different scheme. In any case, it is always possible to create an array of Ring Oscillators and multiplex each one individually, if we are willing to pay the cost in area.

Finally, something that we left as a small experiment in the results chapter is what happens about the placement and routing of these Ring Oscillators. In last instance, the only timing values that are useful for real life instances are those after a placement and routing. This is something that is not currently incorporated in the tool. For now, we know thanks to our experiments that our proposed way of designing RO is able to track variability even after a placement and routing, but this is not enough. One of the most important things that remains to be done is, then, to somehow take into account accurate values for wires after the place and route.

Possibly the most obvious way of accomplishing this is by executing the placement and routing at the same time that the Ring Oscillator is constructed. Even so, there may be other possibilities that incorporate iteration, such as generating a path first, then placing it, and then using the capacity values for the wires to modify the path. This may be iterated a few times until convergence is reached. Deciding for one of these implementations or even some other third option is, again, something that we leave as future work.

Bibliography

- [1] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics (magazine)*, 38(8):114–117, April 1965.
- [2] H. Jones. Why migration to 20nm bulk CMOS and 16/14nm FinFETs is not best approach for the semiconductor industry. Technical report, International Business Strategies, Los Gatos, CA, January 2014.
- [3] C.A. Mack. Fifty Years of Moore’s Law. *IEEE Transactions on Semiconductor Manufacturing*, 24:202–207, May 2011. ISSN 0894-6507. doi: 10.1109/TSM.2010.2096437.
- [4] A. Poonawala and P. Milanfar. Mask Design for Optical Microlithography - An Inverse Imaging Problem. *IEEE Transactions on Image Processing*, 16(3):774–788, March 2007. ISSN 1057-7149. doi: 10.1109/TIP.2006.891332.
- [5] Martin Wirnshofer. Sources of Variation. In *Variation-Aware Adaptive Voltage Scaling for Digital CMOS Circuits*, volume 41, pages 5–14. Springer Netherlands, Dordrecht, 2013. ISBN 978-94-007-6195-7, 978-94-007-6196-4.
- [6] Alexander Tetelbaum. Corner-based timing signoff and what is next. (*white paper*): http://abeliteda.com/wpcontent/uploads/2012/01/Corner_based_Signoff_paper_Jan_2014.pdf.
- [7] Bruce D Cory, Rohit Kapur, and Bill Underwood. Speed binning with path delay test in 150-nm technology. *Design & Test of Computers, IEEE*, 20(5):41–45, 2003.
- [8] Animesh Datta, Swarup Bhunia, Jung Hwan Choi, Saibal Mukhopadhyay, and Kaushik Roy. Speed binning aware design methodology to improve profit under parameter variations. In *Design Automation, 2006. Asia and South Pacific Conference on*, pages 6–pp. IEEE, 2006.
- [9] Vladimir Zolotov, Chandu Visweswariah, and Jinjun Xiong. Voltage binning under process variation. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 425–432, 2009.

- [10] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, et al. Razor: A low-power pipeline based on circuit-level timing speculation. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 7–18. IEEE, 2003.
- [11] Keith A Bowman, James W Tschanz, Nam Sung Kim, Janice C Lee, Chris B Wilkerson, SL Lu, Tanay Karnik, and Vivek K De. Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance. *Solid-State Circuits, IEEE Journal of*, 44(1):49–63, 2009.
- [12] S. Ghosh, S. Bhunia, and K. Roy. CRISTA: A New Paradigm for Low-Power, Variation-Tolerant, and Adaptive Circuit Synthesis Using Critical Path Isolation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(11):1947–1956, November 2007. ISSN 0278-0070. doi: 10.1109/TCAD.2007.896305.
- [13] P. Ndai, N. Rafique, M. Thottethodi, S. Ghosh, S. Bhunia, and K. Roy. Trifecta: A Nonspeculative Scheme to Exploit Common, Data-Dependent Subcritical Paths. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(1), January 2010. ISSN 1063-8210. doi: 10.1109/TVLSI.2008.2007491.
- [14] Qunzeng Liu and S.S. Sapatnekar. Capturing Post-Silicon Variations Using a Representative Critical Path. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(2):211–222, February 2010. ISSN 0278-0070. doi: 10.1109/TCAD.2009.2035552.
- [15] Jiayong Le, Xin Li, and Lawrence T. Pileggi. STAC: Statistical Timing Analysis with Correlation. In *Proceedings of the 41st Annual Design Automation Conference, DAC '04*, pages 343–348, New York, NY, USA, 2004. ACM. ISBN 1-58113-828-8. doi: 10.1145/996566.996665.
- [16] A.J. Drake, R.M. Senger, H. Singh, G.D. Carpenter, and N.K. James. Dynamic measurement of critical-path timing. In *IEEE International Conference on Integrated Circuit Design and Technology and Tutorial, 2008. ICICDT 2008*, pages 249–252, June 2008. doi: 10.1109/ICICDT.2008.4567288.
- [17] Lin Xie and A. Davoodi. Representative path selection for post-silicon timing prediction under variability. In *2010 47th ACM/IEEE Design Automation Conference (DAC)*, pages 386–391, June 2010.

- [18] Liangzhen Lai, Vikas Chandra, Robert Aitken, and Puneet Gupta. SlackProbe: A Low Overhead in Situ On-line Timing Slack Monitoring Methodology. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE '13*, pages 282–287, San Jose, CA, USA, 2013. EDA Consortium. ISBN 978-1-4503-2153-2.
- [19] Tuck-Boon Chan, Puneet Gupta, Andrew B Kahng, and Liangzhen Lai. Synthesis and analysis of design-dependent ring oscillator (DDRO) performance monitors. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 22(10):2117–2130, 2014.
- [20] Aaron Grenat, Sanjay Pant, Ravinder Rachala, and Samuel 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*, pages 106–107. IEEE, 2014.
- [21] Nasser Kurd, Praveen Mosalikanti, Mark Neidengard, Jonathan Douglas, and Rakesh Kumar. Next Generation Intel[®]; Core[™]; Micro-Architecture (Nehalem) Clocking. *IEEE Journal of Solid-State Circuits*, 44(4):1121–1129, April 2009. ISSN 0018-9200. doi: 10.1109/JSSC.2009.2014023.
- [22] Keith A. Bowman, Carlos Tokunaga, Tanay Karnik, Vivek K. De, and James W. Tschanz. A 22 nm All-Digital Dynamically Adaptive Clock Distribution for Supply Voltage Droop Tolerance. *IEEE Journal of Solid-State Circuits*, 48(4):907–916, April 2013. ISSN 0018-9200, 1558-173X. doi: 10.1109/JSSC.2013.2237972.
- [23] Charles Lefurgy, A. Drake, M. Floyd, M. Allen-Ware, B. Brock, Jose Tierno, J. Carter, and R. Berry Jr. Active Guardband Management in POWER7+ to Save Energy while Maintaining Reliability and Performance. *IEEE Micro*, August 2013.
- [24] Kwanyeob Chae and Saibal Mukhopadhyay. All-Digital Adaptive Clocking to Tolerate Transient Supply Noise in a Low-Voltage Operation. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 59(12):893–897, December 2012. ISSN 1549-7747, 1558-3791. doi: 10.1109/TCSII.2012.2231033.
- [25] Kathryn Wilcox, Robert Cole, Harry R. Fair III, Kevin Gillespie, Aaron Grenat, Carson Henrion, Ravi Jotwani, Stephen Kosonocky, Benjamin Munger, Samuel Naffziger, Robert S. Orefice, Sanjay Pant, Donald A. Priore, Ravinder Rachala, and Jonathan White. Steamroller Module and Adaptive Clocking System in 28 nm CMOS. *IEEE Journal of Solid-State Circuits*, 50(1):24–34, January 2015. ISSN 0018-9200, 1558-173X. doi: 10.1109/JSSC.2014.2357428.