PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE ENGENHARIA

FACULDADE DE INFORMÁTICA

ENGENHARIA DE COMPUTAÇÃO

DESIGN AND IMPLEMENTATION OF A STANDARD CELL LIBRARY FOR BUILDING ASYNCHRONOUS ASICS

MATHEUS TREVISAN MOREIRA

END OF TERM WORK

PROF. DR. NEY LAERT VILAR CALAZANS

ADVISOR

PORTO ALEGRE 2010

MATHEUS TREVISAN MOREIRA

DESIGN AND IMPLEMENTATION OF A STANDARD CELL LIBRARY FOR BUILDING ASYNCHRONOUS ASICS

End of Term work presented as part of the activities to obtain the degree of Computer Engineering at the Faculty of Engineering at the Pontifical Catholic University of Rio Grande do Sul.

PROF. DR. NEY LAERT VILAR CALAZANS

ADVISOR

Porto Alegre 2010

ACKNOWLEDGEMENTS

To my supervisor Prof. Dr. Ney Calazans, thanks for all his support and guidance in the development of this work. Thanks for the confidence deposited in me and for all the useful discussions we had. Thanks for the opportunities given to me and for always being available for help when I needed.

Thanks to the members of GAPH, to my friends and the people who contributed somehow in the development of this work. Thanks to my friends and colleagues Felipe de Magalhães, Thiago Raupp and Thales Chaves for proof reading. Thanks to Bruno Oliveira for his support in the development of this work.

Special thanks to my colleague and friend Julian Pontes for putting a large amount of time supporting me in the development of this work. Thanks for all the effort to teach me the foundations of microelectronics and for solving my CAD tools problems.

Thanks also go to Andrew Bardsley, from The University of Manchester, for promptly answering questions about the Teak System and for the support given by upgrading the tool to support technology mapping.

Finally thanks to my family and to my girlfriend, Gabriela, for everything they have done for me. Thanks for their love and patience. Without you this work would not have been possible.

Four Xeon based PCs running the Red Hat operating system were used for simulations and a Core 2 Quad based PC running Ubuntu was used for the remaining practical portion of the practical work described in this document.

ABSTRACT

Asynchronous ASIC design automation tools and standard cell libraries development lag behind their synchronous counterpart. Therefore, most asynchronous designs still make use of full-custom design approaches, which contributes to make asynchronous circuits less used. To help in solving this problem this work presents the design and implementation of a standard cell library for building asynchronous application specific integrated circuits (ASICs), called ASCEnD-ST65 (currently at version 0.1), developed for a 65nm gate length STMicroelectronics CMOS process. An abridged version of this library was designed to implement an asynchronous network on chip (NoC) router. Nevertheless, those cells presented elevated leakage power consumption. This characteristic was treated in ASCEnD-ST65, by proposing a new, robust design flow. This flow contains a novel method to determine the dimension of transistors in the CMOS implementation of C-Element gates, fundamental elements in asynchronous design. This work presents the details of the method and adopts it throughout the specification of the library cells. The designed standard cells count with several views: layout, schematic, symbol, Verilog and abstract. LEF and LIB files describe the physical and electrical characteristics for each cell, and the cells have been characterized for three different process corners, worst, typical and best cases. A total of 251 cells were designed. Among these are a large amount of implementations and variations of C-Elements, the main gate used in many asynchronous design styles. A tradeoff between three different C-Element CMOS implementations – Conventional, Symmetric and Weak Feedback – is presented in this work. Also, the library contains other cells, such as metastability filters. To validate the library, two different IC design flows were adopted. First, a typical flow consisting of VHDL sources synthesized and implemented through the Cadence Framework was used to generate and simulate an asynchronous NoC router. Next, a novel design flow, consisting of a Balsa description synthesized through the Teak System was used to generate the netlist of an asynchronous RSA based cryptography circuit. The layout of this circuit was obtained through Cadence Encounter and simulated through Spectre. The results prove the successful integration of this library and Teak System.

Keywords: Standard cells, ASIC, design libraries, asynchronous circuits, Balsa, Teak.

“Any intelligent fool can make things bigger and more complex… It takes a touch of genius – and a lot of courage to move in the opposite direction.”

Albert Einstein

Contents

List of Abbreviations

APT	Advanced Processor Technologies
ASIC	Application-Specific Integrated Circuit
BP	Best for low Power consumption
BS	Best for Speed
CAD	Computer-Aided Design
CeS	Cell Specifier
CMOS	Complementary Metal-Oxide-Semiconductor
DFM	Design For Manufacturability
DI	Delay-Insensitive
DRC	Design Rule Check
DSP	Digital Signal Processor
EDA	Electronic Design Automation
ELC	Encounter Library Characterization
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
GALS	Globally-Asynchronous Locally-Synchronous
GPSVT	General Purpose Standard Vt Transistor
HDL	Hardware Description Language
HVT	High Vt Transistor

I/O Input/Output

IC Integrated Circuit

LEF Library Exchange Format

LIB Liberty Library

LVS Layout Versus Schematic

NMOS N-Type Metal Oxide Semiconductor

NoC Network on Chip

PCHF Precharged Half Buffer

PMOS P-Type Metal Oxide Semiconductor

QDI Quasi-Delay-Insensitive

ROGen Ring Oscillator Generator

RTL Register Transfer Level

RTL Register Transfer Level

SDF Standard Delay Format

SI Speed-Independent

SoC System on a Chip

STM ST Microelectronics

SVT Standard Vt Transistor

UDP User Defined Primitives

VHDL VHSIC Hardware Description Language

VHSIC Very-High-Speed Integrated Circuit

VLSI Very Large Scale Integration

Vt Threshold Voltage

VTC Voltage Transfer Characteristic

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

1 INTRODUCTION

"The world is full of obvious things which nobody by any chance ever observes."

Sir Arthur Conan Doyle

1.1 Introduction

Throughout the evolution of electronic design techniques, some assumptions were adopted in order to overcome the increasing complexity of circuits. Classically, digital electronics assumes the discretization of information using two distinct voltage levels to represent the digits of a binary system, e.g.: high voltage (digit 1) and low voltage (digit 0). In this way, information can be manipulated through logic and arithmetic operations and digital devices are easily implemented. The evolution of the technologies used to fabricate integrated circuits (ICs) brought the possibility of implementing hundreds of millions of transistors in a single chip. Thus, modern ICs can contain all elements of a complex system in a single chip. Such devices are usually called Systems on a Chip (SoCs) [REI00].

At present, many SoC designs adopt the synchronous paradigm, which implies another assumption: a discrete notion of time. Synchronous digital circuits employ an externally generated control signal, usually called *clock*. The clock signal, distributed through the circuit and employed by all its storage components, synchronizes the overall circuit state changes. This abstraction of time significantly reduces the complexity of a digital circuit design and allows it to be more easily modeled using register transfer level (RTL) description languages, such as Very-High-Speed Integrated Circuits Hardware Description Language (VHDL) and Verilog [BRO05] [VAH10].

However, with the evolution of very large scale integration (VLSI) technologies, limitations of the synchronous design paradigm started to emerge. Synchronous design problems that were easily overcome in previous decades, such as clock skew and power consumption, became a complex task to solve in modern designs. The clock signal dissipates an average of 45% of the whole power in high performance processors and can reach as much as 70% of the power dissipation for such devices [AMD05]. This power budget makes complex synchronous designs less attractive for low power applications such as embedded systems. Therefore, it is necessary to consider new strategies as the complexity to design a synchronous SoC gets higher [ITR09].

An alternative is to use non-synchronous circuits, which make use of the voltage level discretization but where there is not a global clock signal [MYE01] [SPA01]. In this way, time is again considered a continuous variable. This characteristic makes such circuits more sensitive to temporal phenomena [PON08]. In contrast to the synchronous paradigm, in order to perform synchronization, communication and sequencing operations, asynchronous circuits often use explicit handshaking between their elements. Consequently, the circuit executes a computation only when and where it is needed [SPA01]. To illustrate this, Figure 1.1(a) shows an example of a synchronous circuit, where data flows through registers (*Reg.1*, *Reg. 2*, …) controlled by a global and common clock signal (*CLK*). Data transformation is accomplished by the combinational logic blocks (*CL3*, *CL4*) interspersed between registers. In an asynchronous counterpart, that Figure 1.1(b) displays, data flows by means of local communications between each pair of storage elements, using a handshake channel. Note that asynchronous design may employ level sensitive storage devices instead of the usually edge-triggered flip-flops used in synchronous design.

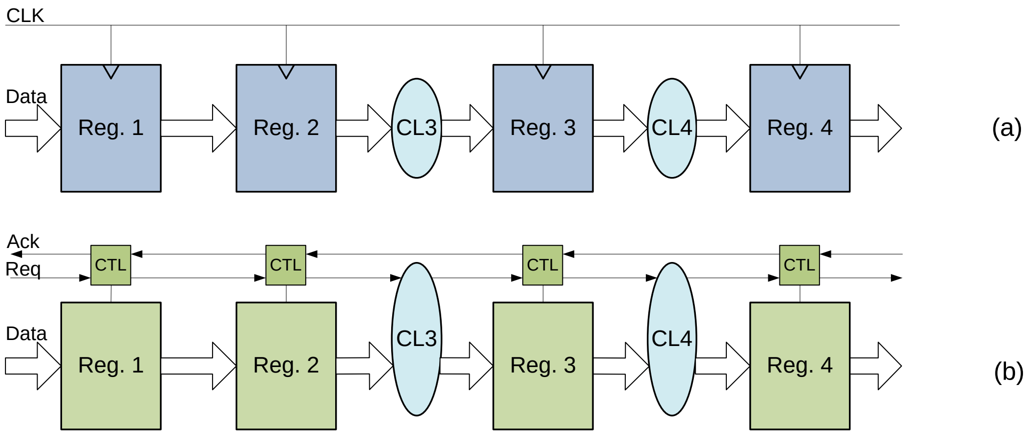


Figure 1.1 – (a) A synchronous circuit and (b) an asynchronous implementation, using local handshake control circuits (blocks called CTL) [SPA01].

In fact, the interest in non-synchronous circuits has been increasing since the last decade. The *International Technology Roadmap for Semiconductors* (ITRS) published in [ITR01] suggests that at 65nm technology nodes and below, communication architectures and protocols for SoCs would require a significant change, as it would become impossible to move signals across a large die within the period of a clock signal or without dissipating too much power. The ITRS 2008 Edition [ITR08] presents the need for asynchronous communication protocols for control and synchronization in ICs for the next decades, as depicted in Figure 1.2. Therefore, it seems inevitable a shift to asynchronous or globally asynchronous locally synchronous (GALS) design styles for deep submicron ICs.

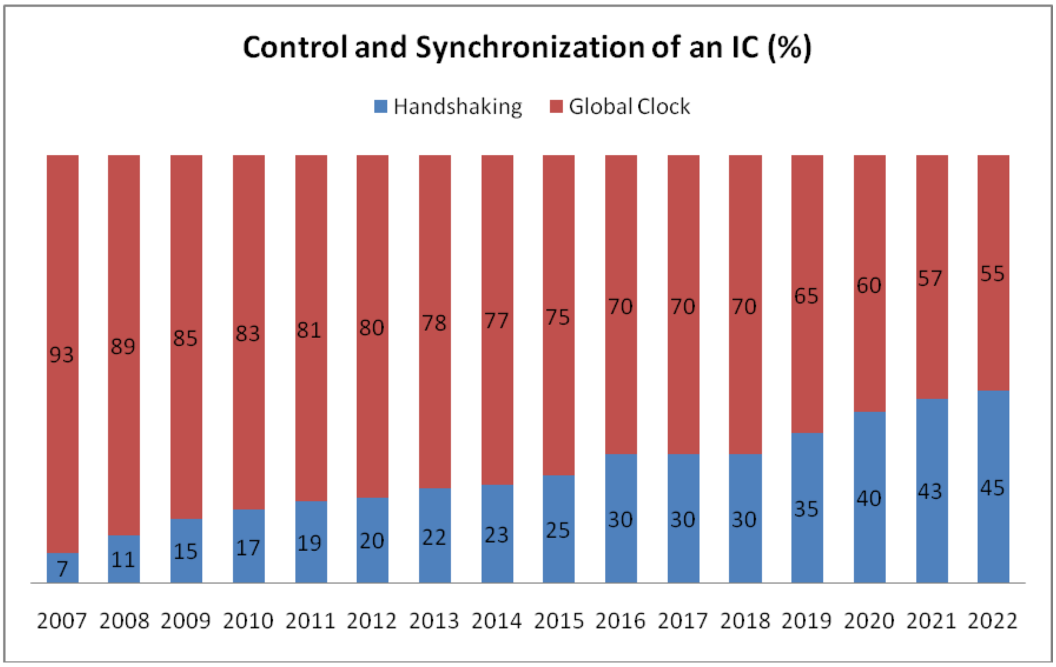


Figure 1.2 – Required percentage for next decades ICs driven by handshaking signaling according to [ITR08].

Conversely, the lack of adequate electronic design automation (EDA) tools imposes a great barrier in the design of asynchronous circuits. This is because the majority of commercial applications target the synchronous paradigm. According to the ITRS 2009 Edition [ITR09] further progress in the asynchronous paradigm depends on commercial tool support. The high complexity of modern ICs makes the task of ensuring that such designs will operate correctly on silicon nearly impossible without the use of adequate EDA software and consolidated design methodologies [MIC94] [REI00] [RAB03] [WON05]. Additionally, time to market constraints become increasingly stricter as the semiconductors industry develops new technologies and short design time remains an essential feature for successful products.

Current SoC designs use the standard-cell methodology extensively as an answer to design automation needs. This aims at fulfilling the constraints imposed by complex modern ICs. In fact, this concept (standard-cell based design) was invented to speed up the design phase of high performance ICs and is often referred as the key success factor for the rapid growth of integrated systems [JAM99] [ERI03] [HAS03]. The reason for that is the use of a standard library of pre-defined cells, provided by chip vendors, which contains a collection of gate-level elements standardized at logic or functional level [RAB03]. Such components are pre-designed and pre-verified, making the task of implementing an IC on silicon much easier to be automated by EDA tools. The latter support the assumption of using functional blocks to build the design [SAI02]. For instance, Figure 1.3 shows the layout of a standard-cell based design, depicting the use of different pre-implemented blocks interconnected in order to create a given functionality.

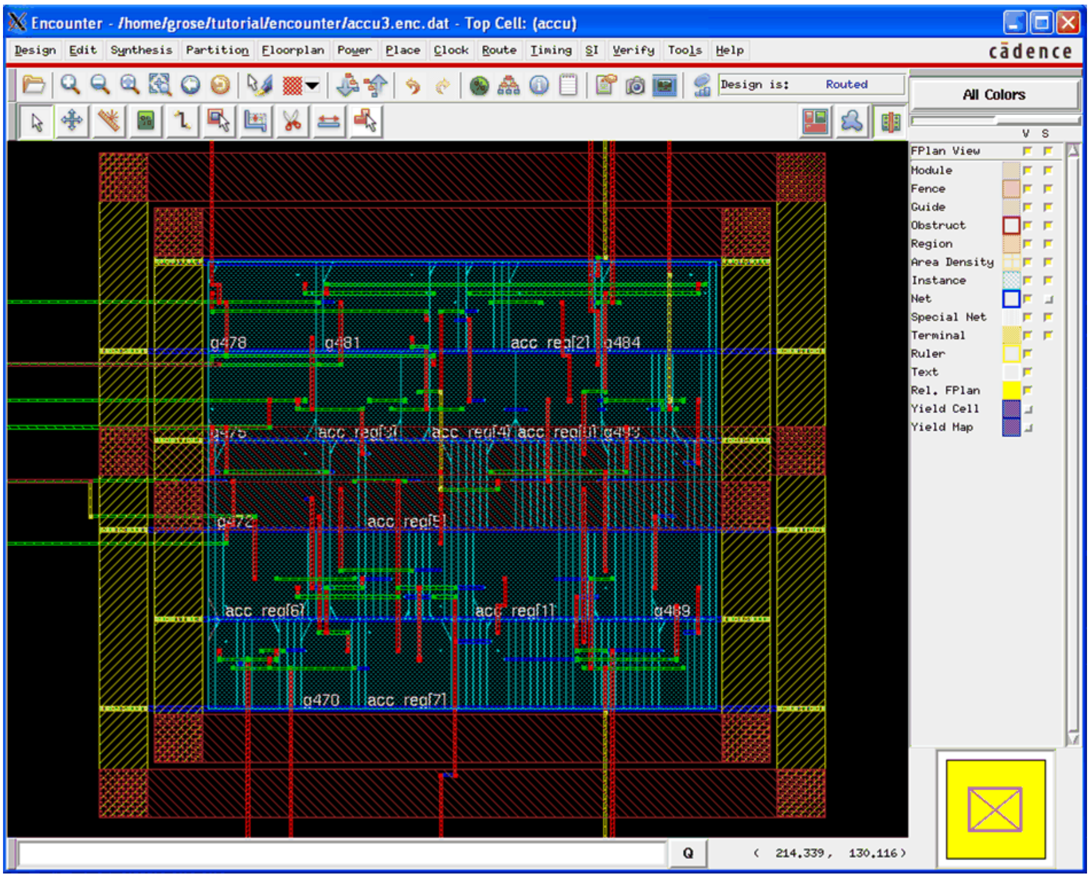


Figure 1.3 – Example of a partial vie w of a standard-cell based IC die, obtained with the Cadence Encounter layout editor tool [CAD10]. There are 5 horizontal stripes located at the center of the die containing each from 5 to 7 standard-cells. Vertical thin lines are wires interconnecting cells. Thin vertical rectangles are filler cells used to guarantee manufacturability characteristics of the design. The design is encircled by rectangles that represent global power and ground lines.

One major drawback for those intending to use standard-cell design methods for the asynchronous paradigm is that most libraries provided by vendors still contain only basic devices for synchronous design, such as logic gates and flip-flops. However, efficient asynchronous designs require other specific devices as well, as it will be explored in this document. The objective of this work is to supply these devices in order to enable the design of asynchronous standard cell based ASICs. To do so, a library containing over 200 typical asynchronous gates was designed for a 65nm gate length CMOS technology.

1.2 Contributions of This Work

This work presents ASCEnD, a standard cell library created to support the design of asynchronous ASICs. This library contains, in its present version, over 200 gates. Among the available standard cells there are different implementations of C-Elements and metastability filters. If designers associate ASCEnD to a regular standard cell library containing e. g. gates and flip-flops, this is sufficient to support asynchronous design styles such as QDI based on a 4-phase protocol with dual-rail encoding or bundled-data micropipelines [SPA01]. A novel flow to perform transistor sizing for C-Elements is presented and adopted in the design of the library. Moreover, this document also describes an alternative flow to implement asynchronous circuits and adopts it to validate ASCEnD. This flow employs asynchronous EDA tools from The University of Manchester Advanced Processor Technologies (APT) Group [APT10]. An asynchronous version of an RSA based cryptographic circuit was described in Balsa language and implemented in the designed library.

1.3 Work Structure

The remainder of this work is organized as follows. Chapter 2 provides relevant background information. Chapter 3 presents the state of the art in open source standard cell libraries and their support for asynchronous circuits. Chapters 4 , 5 and 6 present the developed work. The design method adopted, along with the required tools, is described in Chapter 4 . Next, Chapter 5 explores the designed library. Chapter 6 presents the validation of the library through the design and simulation of an asynchronous RSA cryptography circuit and a Network on Chip (NoC) router using ASCEnD. Finally, Chapter 7 explores conclusions and directions for future works.

2 CONCEPTS

"It was easier to know it than to explain why I know it."

Sir Arthur Conan Doyle

This Chapter presents basic concepts of microelectronic design styles and a typical design flow for standard cell libraries. Also, it describes some basic concepts of asynchronous circuits and addresses the Balsa System, which comprises a language and a framework for describing and implementing such circuits in ASIC and FPGA technologies.

2.1 Microelectronic Design Styles

Different methods are used to design integrated circuits (ICs). Usually these are roughly classified into full-custom or semi-custom design styles, or methods. In the former, both the functional and physical design are handcrafted, which implies a large effort of the design team to achieve maximum efficiency of each single feature of the circuit. Semi-custom designs, on the other hand, aim at reducing fabrication costs and time as well as design complexity, by using predesigned and pre-validated gates or even pre-diffused ICs. Figure 2.1 provides a classification that can be used to encompass available microelectronic design styles.

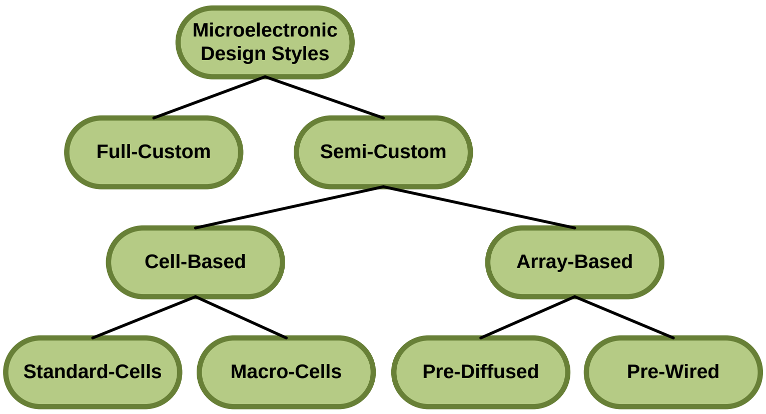
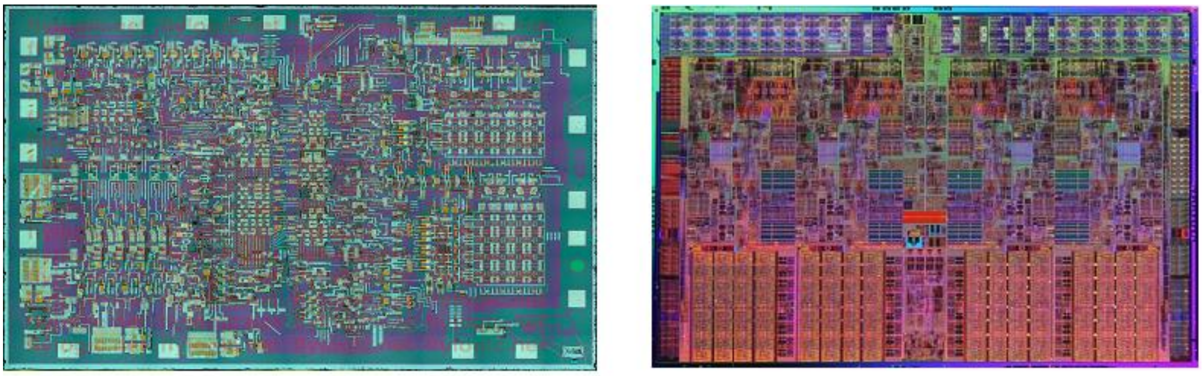


Figure 2.1 – Microelectronic design styles. Adapted from [MIC94] and [RAB03].

In the full-custom approach, every transistor has its functional and physical design optimized for excellence. This methodology involves detailed manipulation of physical and electrical characteristics, through layout design and simulation. However, the increasing complexity of current ICs, added to strict time-to-market constraints, has confined custom design techniques to specific cases, where such extremely costly effort pays off. For instance, Figure 2.2 shows an example of contemporary designs in comparison with previous decades. The Intel 4004, Figure 2.2 (a) was fabricated in the early 70's and counted roughly with 2.300 transistors. In contrast, the Intel i7, Figure 2.2 (b), in the market since 2009, contains around 800,000,000 transistors. It is simply impossible to design such an IC, as the latter, using a full-custom design approach and respect time to market constraints. Hence, another method is necessary to supply the fast transitioning and high demanded IC market: the semi-custom design style.



(a)

(b)

Figure 2.2 – Die photo of (a) Intel 4004 and (b) i7 microprocessors [INT10].

Semi-custom design styles can be divided in two subgroups: cell-based and array-based. The former aims to reduce fabrication costs and time. Semi-custom designs can even completely avoid to the end user the need to get involved in the IC manufacturing process, as in the case of field programmable gate arrays (FPGAs), an off-the-shelf IC fabricated as a pre-diffused array-based design. The cell-based approach can still be subdivided into macro cell and standard cell designs. Macro cell designs, in short, often make use of computer programs to synthesize modules. Such modules are implemented as circuitry that is too complex to be handcrafted but still presents some degree of regularity that can be exploited by a parameterized automatic generation process. Structures such as multipliers, data paths, memories, embedded microprocessors and digital signal processors (DSPs) are examples of commonly used macro cells [RAB03].

The semi-custom standard cell approach increases the grain size of design to logic gates, in contrast to designing each transistor, as in a full-custom method. When using standard cells, a library is provided containing a vast number of logic gates over a range of fan-in and fan-out with different physical, electrical and logical characteristics. With the support of EDA tools, the layout of a circuit with arbitrary complexity can be automatically generated either from a schematic, containing exclusively cells from the library, or from a higher level specification like a hardware description language (HDL) description, which is then synthesized to the desired technology and mapped according to the library elements [MIC94]. Additionally, a design does not need to be limited to a single method and it is common to have ICs designed with compatible methods. For instance, the Intel Pentium 4, which die appears in Figure 2.3, was designed mostly with automated semi-custom approaches but some of its performance-critical modules, such as clock buffers, were designed manually [RAB03].

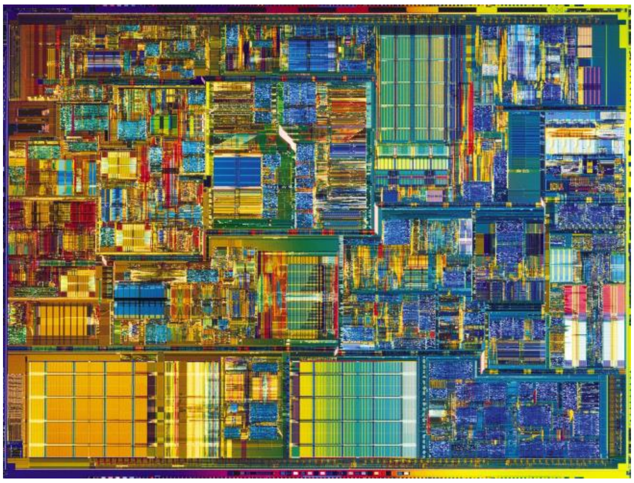


Figure 2.3 – Intel Pentium 4 die photo [INT10]. The patterns and rectangular shapes in the chip, makes clear the use of a semi-custom design style.

The choice for a design style is often based on a tradeoff between performance and design cost and time. For instance, a full-custom approach may only be justified over a very high volume, due to its high cost. The advantages and drawbacks of the design styles mentioned here are summarized in Table 2.1.

Table 2.1 – Tradeoffs between microelectronic design styles, from [MIC94].

	Full-Custom	**Cell-based**	**Prediffused**	**Prewired**
Density	Very High	High	High	Medium-Low
Performance	Very High	High	High	Medium-Low
Flexibility	Very High	High	Medium	Low
Design time	Very Long	Short	Short	Very Short
Manufacturing time	Medium	Medium	Short	Very Short
Cost – low volume	Very High	High	High	Low
Cost – high volume	Low	Low	Low	High

2.2 A Typical Design Flow of a Standard Cell Library

The typical design flow of a standard cell library consists in designing a set of logical gates at the transistor level for a given technology process. Figure 2.4 presents an overview of the basic steps required to design each element of a standard cells library. The design of each gate starts from a specification, defining logical and electrical parameters like propagation delay and power consumption. From these, the physical view of each gate is designed. In a typical standard cell based design, cells are placed side by side by an automated place and route tool. Therefore, rules are needed. In order for the gates to fit side by side, every element must have the same height with a variable, albeit predictable, width. Moreover, the input and output terminals of each gate must be placed in pre-determined physical regions.

From the metal layers of the physical view, an abstract view is generated, which specifies internal nodes and input/output (I/O) terminals. Then, the extracted circuit is obtained through an automated extraction tool. This circuit contains not only the drawn transistors but also every parasitic device generated from the physical layers. In this way, a more realistic view is obtained, and with it simulations are performed to characterize the electrical behavior of the designed gate. With the electrical characteristics of each gate, a circuit can be synthesized from a high level description. With the abstract views the place and route tool can easily assemble the IC from these predesigned blocks.

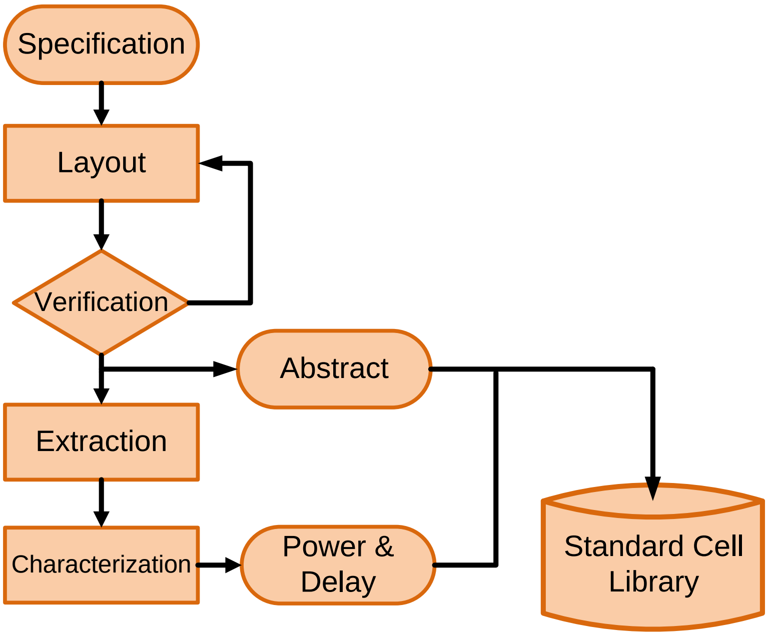


Figure 2.4 – Typical design flow of each gate of a standard cell library based on [CHO92] [SAI02] [HAS03] [RAB03] [YEO09]. Inputs and outputs are given by rounded corners boxes, actions by boxes, decisions by diamonds and the repository as a cylinder.

2.2.1 Layout

Once the circuit schematic is defined, it is possible to draw its physical view through a layout editor tool. The basic function of this tool is to place polygons representing the different layers used to fabricate an IC in a given technology. However, to make it possible for automated tools to place and connect the cells, rules are needed. Thinking about Lego, the reason why one can build bigger structures with them is that all of these bricks fit with each other. Moreover, all the blocks have standard sizes and standard places for the connection bumps. In order to design a standard-cell library, it is necessary to respect the same rules for each cell [SAI04]. In this way, the gates will fit together in predictable patterns and it is possible to automate the process of generating circuit layouts from abstract descriptions, such as RTL modeling. Figure 2.5 shows a basic set of rules for standard cell libraries.

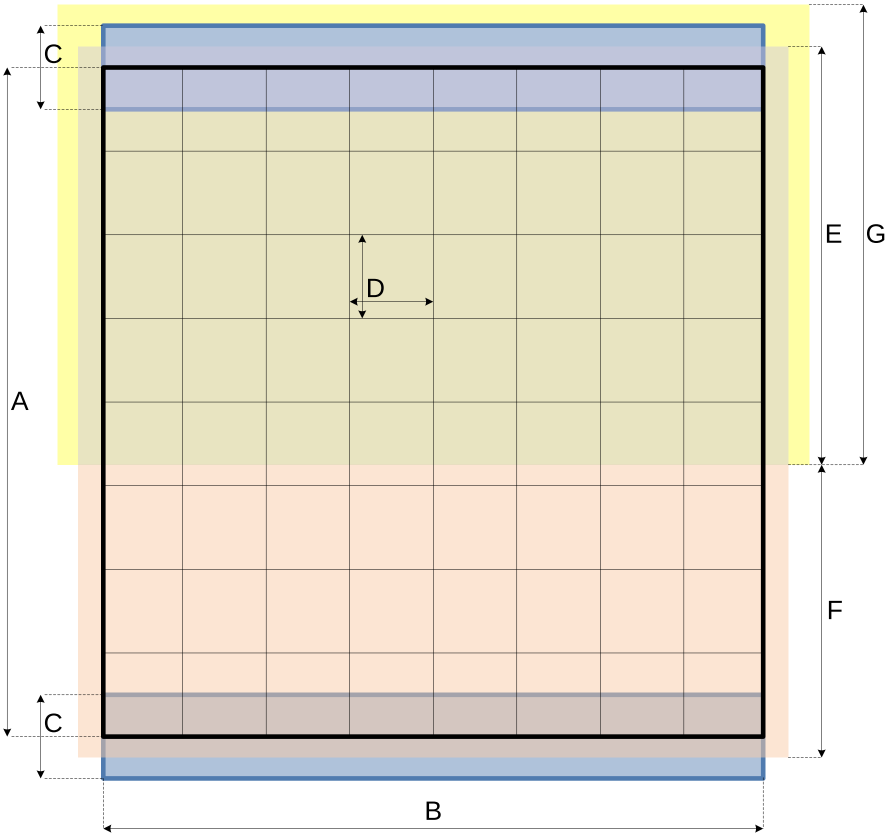


Figure 2.5 – A basic set of rules for a standard cell library. *"A"* and *"B"* are the cell height and width, respectively. *"C"* is the power lines width and *"D"* is the routing grid. *"E"*, *"F"* and *"G"* are the height of the implant layers [SMI97] [SAI02] [RAB03] [SAI04].

Because the generation of the layout of a standard cell-based IC is an automated process, a place and route tool must be used to build the circuit from the gates provided by the library. This tool places the cells side by side and then connects these to implement the logic of the circuit. It does help a lot the automation of this process if the cell height, represented by

"A" in Figure 2.5, is the same for every gate, facilitating the arrangement of the power lines. The cell width however, depicted as *"B"* in the figure, can vary. These dimensions are given as a function of the metal pitches, which are defined by the manufacturer [CAD09]. Usually, the metal pitch represents the minimum distance between the centers of two adjacent layers of a same metal line at minimum size with minimum spacing between them. Figure 2.6 depicts an example of a metal1 pitch.

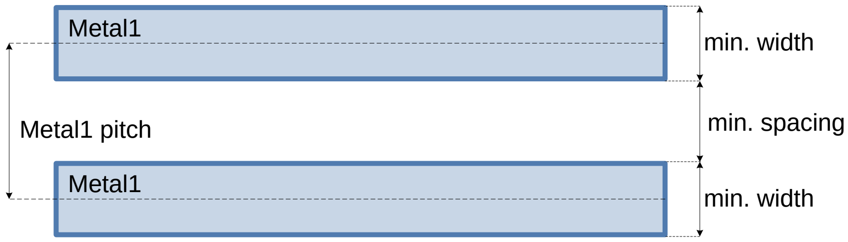


Figure 2.6 – Example of metal1 pitch, the minimum distance between the centers of two adjacent metal1 layers.

Generally, the height of a cell must be an even multiple of the metal1 pitch while the width must be an even multiple of the metal2 pitch. Equation 2.1 defines these relations. In (a) *"P_{m1}"* is the metal1 pitch and in (b) *"P_{m2}"* is the metal2 pitch. The metal pitches also define the routing grid, represented by *"D"*. Input and output pins are placed according to this grid, usually exactly in the middle of the region where the horizontal axis override the vertical axis, which are given by the metal2 and metal1 pitches respectively. Another standardized dimension is the height of the implant layers, *"E"*, *"F"* and *"G"*, due to the fact that the cells are usually placed side by side.

Equation 2.1 – Standard cell height (a) and width (b).

$$\left\langle A = n \times P_{m1} \middle| n \in \mathrm{N}^{*} \right\rangle \qquad \text{(a)}$$

$$\left\langle B = n \times P_{m2} \middle| n \in \mathrm{N}^{*} \right\rangle \qquad \text{(b)}$$

Finally, there is the manufacture grid. This grid defines the metric unit to design a layout. For instance, if the manufacture grid is 0.010nm, every measure in the layout must be an even multiple of 0.010nm in both axes, X and Y, in order to have a manufacturable layout. This grid can usually be defined in the configuration options of the layout editor tool.

2.2.2 Verification

To guarantee that an IC using standard cells is manufacturable, the designed layers must respect some rules stated by the semiconductor foundry. The basic rule classes for layers are spacing, width, enclosure and extension rules, all illustrated in Figure 2.7. From these classes it is possible to define a vast set of rules, specifying relations between different or identical layers. This makes the task of designing a single cell harder, as its complexity increases. To ensure that a given layout respects all the rules for a specific process, it is necessary to use a design rule check (DRC) tool. This tool verifies if the generated layout can be properly manufactured in a specific foundry and technology process. However, ensuring that the circuit can be correctly implemented in silicon does not mean that it will operate correctly. In order to ensure that the layout implements the specified logic, a layout versus schematic (LVS) test must be performed through an LVS tool, which compares the schematic with the layout and checks if the circuits are functionally equivalent.

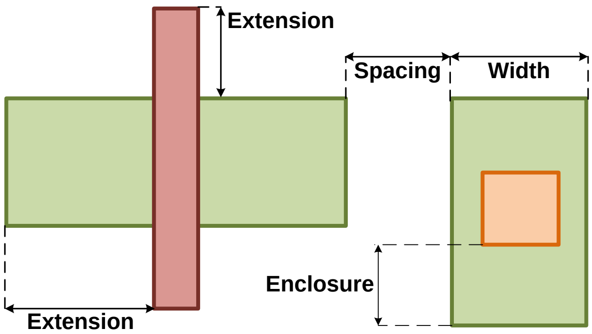


Figure 2.7 – Basic set of DRC rule classes [SAI02] [RAB03].

2.2.3 Abstract

Once a cell is fully verified, an abstract view can be generated. An abstract view is a high-level representation of a layout and is generated based on the extraction of the physical layout of a cell for a given technology [CAD07]. This view contains the metal layers of a circuit and its terminals along with boundaries, which is crucial in the silicon circuit assembly automation. From this view, the place and route tool will obtain the necessary information to place each gate and connect its terminals.

2.2.4 Extraction

Following the cell abstraction step, a circuit extractor tool must be used to scan the different layers of the physical design and reconstruct the circuit schematic from this geometrical description. The generated circuit contains not only the transistors that implement the logic of the gate, but also every parasitic element, like the capacitance between the gate and the bulk. This allows a more accurate simulation and analysis of the designed circuit [RAB03]. Figure 2.8 shows parasitic elements for a physical implementation of an NMOS transistor.

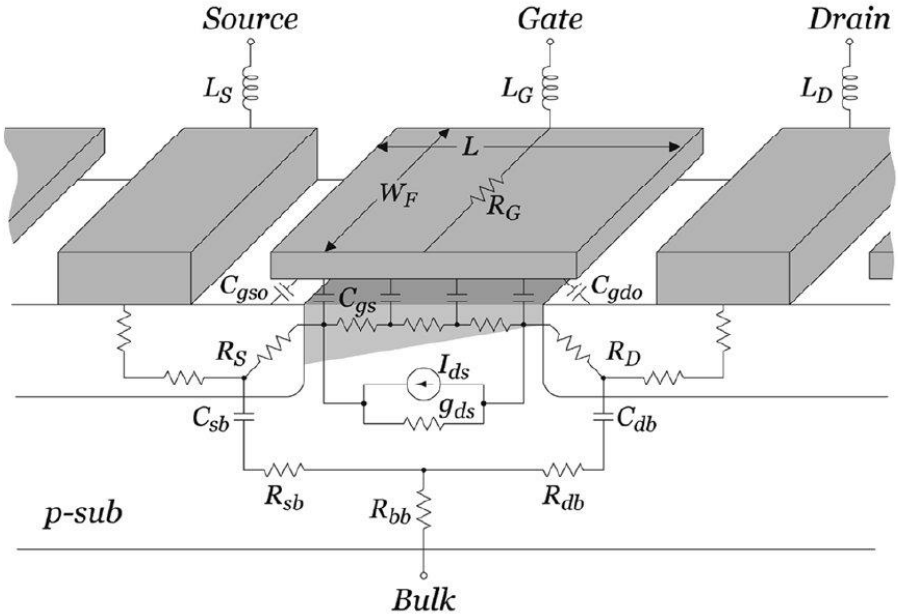


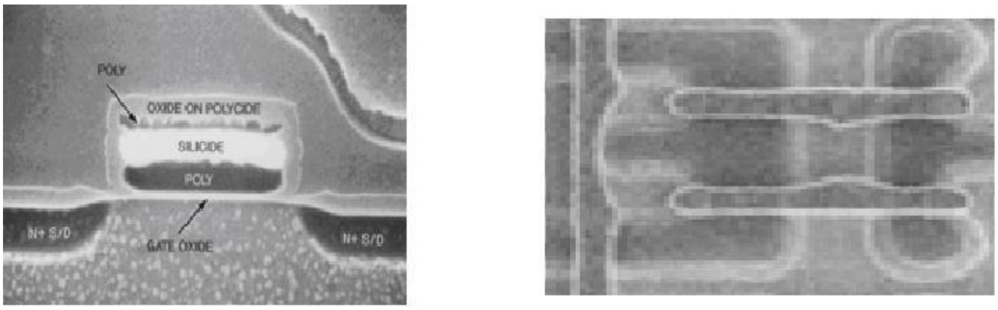
Figure 2.8 – Physical model of an NMOS transistor, extracted from [DOA05].

2.2.5 Electrical Characterization

A robust physical characterization of a standard cell library can guarantee that a design can be properly manufactured using the gates of the library, but it does not ensure that the design will work correctly on silicon. Hence, just logical and physical characteristics of the gates – functionality, size and terminal positions – are not enough. The quality of the result of a synthesis tool depends strongly on the level of detail of the library's electrical characterization process. This process consists in specifying the delay and power consumption of each gate as a function of the fan-out and the input slopes [RAB03].

Due to variations in the fabrication process, gates must be characterized at different manufacturing parameters, which will reflect their real behavior for different fabrication process variations. In this way, it is possible to verify that a design will operate correctly and perform to its required timing specification even in worst case conditions. In addition, there is also the influence of operational conditions, temperature and voltage, which can make the delay of the cells vary as well. For instance, Figure 2.9 (a) shows a microphotograph of a fabricated transistor cross section. In the photograph it is possible to identify the possible source of process variations on a channel length, on the distance between doped regions (left and right bottom darkened areas). It is possible to note for example, that this later distance is not the same for different depths of the channel. Figure 2.9 (b) shows how variations in the gate and source/drain of a transistor may occur, due to manufacturing process variations. Such phenomena can, when not adequately controlled result in faulty ICs.

A robust standard cell characterization is clearly a challenge, especially in contemporary deep submicron technologies, where any slight variation can compromise a whole chip. Fortunately, there are tools that automate the task of exhaustively simulating all necessary conditions to ensure the reliability of a characterization process. Once robust information about electrical, physical and functional characteristics of a cell is obtained, this can become part of a library.



(a) (b)

Figure 2.9 – CMOS microphotograph showing the possible origins of process corners in (a) a transistor of an Intel 386 processor, and (b) a gate of an Intel SDRAM [INT10].

2.3 Asynchronous Circuits

A digital circuit is (fully) asynchronous when no clock signal is used to control any sequencing of events. Instead, asynchronous modules use handshaking between its components to synchronize, communicate and operate [SPA01] [MYE01]. In “synchronous terms” the resulting behavior is registers being clocked only when and where this is needed [BAR00] [SPA01]. Such characteristic presents advantages over the use of a global clock signal in modern technologies. According to [BER99] [SPA01] [MYE01] [BOU07] [CHO07] and [CRI10], asynchronous circuits may present:

- Lower dynamic power consumption: When not computing, the circuit is quiescent, i.e. consumes only leakage current. When requested, the circuit starts to compute and starts to consume energy. However, after the computation of a task, the circuit returns to a quiescent until next request.
- Higher operating speed: The operating speed is given by local latencies rather than a global signal. Asynchronous circuits operate at average case delays, while their synchronous counterpart are designed based on worst case delays.
- Better composability and modularity: Since synchronization is given by local handshakes rather than a global signal.
- Low electromagnetic emissions: Registers do not switch all at the same time. Instead, they switch at random points of time through local requests.

- High robustness: The circuit continues to operate correctly for a larger range of variation in the power supply. Moreover, since no assumptions are made about the inter-cell communication delays, the circuit is much more tolerant to process, voltage and temperature variations.

Yet, there are also drawbacks concerning the use of asynchronous control logic, such as the obvious area penalty, and in some cases circuit speed and power consumption may be compromised [BER99] [SPA01] [MYE01] [KWE07].

2.3.1 Muller C-Element

The Muller C-element [MUL57] is a fundamental component in asynchronous circuits design and is extensively used [SPA01] [MYE01]. This is because the C-element operates as an event synchronizer. Its output will only switch when all inputs have the same logical value. Its truth table is represented in Table 2.2. When inputs (*A* and *B*) have the same value, the output (*Q*) assumes this same value. However, when the inputs are different, the output keeps its previous logic value. Hence, the C-element can be viewed as an AND function for transitions [BAR00]. C-elements are often required in asynchronous circuits to allow the synchronization of control inputs. Its properties make C-elements crucial components in the implementation of asynchronous circuits control.

Table 2.2 –C-Element truth table.

A	**B**	**Q_i**
0	0	0
0	1	Q_{i-1}
1	0	Q_{i-1}
1	1	1

The C-Element can also have some variations concerning to the symmetry of its inputs. Table 2.3 (a) shows the truth table of an asymmetric C-Element with a normal input (*B*) and a rising[1] input (*A*). When both inputs are high, the output is high. However, if *B* is low, the output is low, because *A* is a rising input. On the other hand, in Table 2.3 (b), the truth table of an asymmetric C-Element with a normal input (*A*) and a falling input (*B*) is showed. When both inputs are low, the output is low. However, since *B* is a falling input, only *A* needs to be high for the output to be high. Additionally, the C-Elements can have reset and/or set inputs. A reset input sets the output to low regardless the value of the other inputs, while the set input

[1] A rising input is an input that drives only the rising logical switching of an output. It is represented in symbols with “+”. On the other hand, a falling input is an input that drives only the falling logical switching of an output. This inputs are represented with “–“ in symbols.

sets it to high.

Table 2.3 – Asymmetric C-Element truth tables. In (a) a C-Element with a normal input (*B*) and a rising input *(A)*. In (b), a C-Element with a normal input (*A*) and a falling input (*B*).

A	B	Q_i
0	0	0
0	1	Q_{i-1}
1	0	0
1	1	1

(a)

A	B	Q_i
0	0	0
0	1	Q_{i-1}
1	0	1
1	1	1

(b)

From the presented cells, different versions of C-Elements can be implemented, providing a more flexible library. For instance, in [YAH06] different versions of registers are implemented and compared. In order to implement these versions, different variations of C-Elements are required. Figure 2.10 shows 3 buffer implementations from logic gates. In (a), a weak conditioned half buffer (WCHB) is designed with a NOR gate and two 2 input C-Elements with a reset pin (*C1* and *C2*). In (b), a pre charged half buffer (PCHB) is implemented with two NOR gates, two asymmetric 3 input C-Elements with one rising input and a reset pin (*C1* and *C2*) and one asymmetric 2 input C-Element with a rising input (*C3*). In (c), a pre charged full buffer is implemented with two NOR gates, one inverter, two asymmetric 3 input C-Elements with one rising input and a reset pin (*C1* and *C2*), one asymmetric 3 input C-Element with a rising and a falling input (*C3*) and one asymmetric C-Element with a rising input (*C4*).

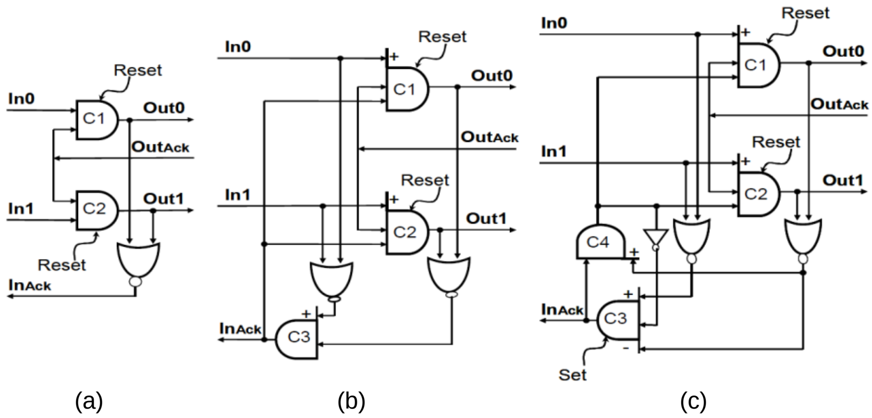


Figure 2.10 – Different buffer implementations. Weak Conditioned Half Buffer (WCHB) circuit diagram in (a), Pre-Charged Half Buffer (PCHB) in (b) and Pre-Charged Full Buffer (PCFB) in (c). Picture taken from [YAH06].

The designed library counts with all the C-Element variations described above. In this way, different implementations can be obtained by using it to build circuits.

2.3.2 Delay Models

Asynchronous circuits can be classified according to several criteria. The most used classification criterion is based on the delays of wires and gates. The most robust and restrictive delay model is the delay-insensitive (DI) model. Circuits in this class operate correctly regardless of gate and wire delays. Unfortunately, these circuits are restricted to modules composed of C-elements and inverters only [SPA01]. Referring to the circuit depicted in Figure 2.11, a DI circuit would operate correctly with arbitrary delay values d_A, d_B, d_C, d_1, d_2 and d_3.

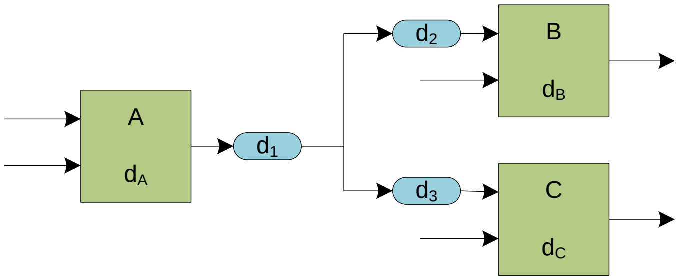


Figure 2.11 – Circuit fragment with gate and wires delays, adapted from [SPA01].

Asynchronous circuits can also be implemented with the addition of an assumption on the wire delays in some carefully indicated forks. This kind of implementation is called quasi-delay-insensitive (QDI). In this delay model, signal transitions must occur at the same time at all end points of the mentioned forks. Let us say that the circuit in Figure 2.11 is QDI and it contains a single fork. Therefore, Equation 2.2 must be true, where λ is negligible, and d_A, d_B, d_C and d_1 are arbitrary [SPA01].

Equation 2.2 – QDI delay assumption for the circuit depicted in Figure 2.11.

$$|d_2 - d_3| = \lambda$$

There are also speed-independent (SI) circuits. Such circuits assume positive, unknown delays in the gates and ideal wire delays. For instance, in the circuit shown in Figure 2.11 this means arbitrary delay for d_A, d_B and d_C and zero-delay for d_1, d_2 and d_3 [SPA01]. Obviously, the assumption of ideal wires in contemporary semiconductor processes is impractical [RAB03]. Therefore the most used classes are the QDI and DI, the latter being more often applied to higher levels of abstraction, where the grain is not at gate level, due to its nature [BAR00]. More delay models and more detailed information can be found in [SPA01].

2.3.3 Handshaking

Handshaking is the most used communication protocol to synchronize and transfer data through asynchronous elements of a given circuit [SPA01]. It consists, in its most basic form, of two control signals in opposite directions, request and acknowledge. Given that the mechanism can be used with the sole purpose of control, to synchronize elements only, a data channel is optional. The protocol consists of an active element sending a request to synchronize with a passive element, which issues an acknowledgement, when it is ready to communicate. Figure 2.12 (a) shows an example of two components that use handshaking to synchronize through the signals *req* and *ack*, representing request and acknowledge respectively. If the components are to transfer data, there are two possible configurations: using push or pull channels. The former consists of two handshaking elements that transfer data, where the data flows from the active to the passive. In the latter, on the contrary, the data flows from the passive to the active. Examples of push and pull data channels are depicted in Figure 2.12 (b) and (c), respectively.

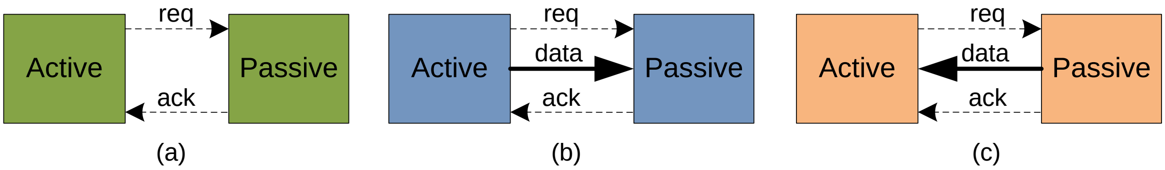


Figure 2.12 – Example of communication through handshaking. A pure control channel (a), a push data channel (b) and a pull data channel (c) [TOM06].

There are two possible ways to implement handshaking protocols, transition and level signaling [SPA01] [TOM06]. In the latter, also known as four phase protocol, the logic level of the request and acknowledge signals are used to control the handshake. Figure 2.13 (a) shows an example of a four phase handshake communication. In this example, the active element starts with a requisition to communicate, rising the *req* signal. If any data is to be transferred through a push channel it must be valid at this moment. The passive element reads, processes the request and asserts the *ack* signal. In a push channel the passive element reads the data made available by the active element. In a pull channel, data must be valid when the acknowledgment is signaled. When the active element reads the acknowledgement it also reads the data, if any, and sets the *req* signal to low. At any moment from there a new communication can take place. Since the level signaling has four handshaking phases, different validity schemes can be used. For simplicity sake, they will not be described in the present work. In [TOM06] each scheme is discussed in detail.

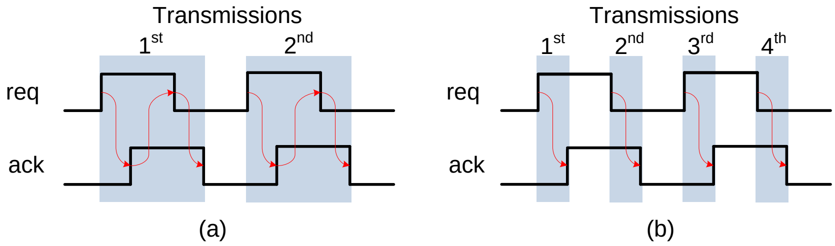


Figure 2.13 – Operation of the four phase (a) and two phase (b) handshake protocol [PON08].

The transition signaling is very similar to the level signaling but its transitions are reduced to half. In this protocol, as shown in Figure 2.13 (b), the active element starts with a request, switching the logical level of the *"req"* signal. If a push channel is implemented the data must be valid at this moment. When the passive element reads the request, it processes it, reads the data (if any) and, then, sends an acknowledgement, switching the logical level of the *"ack"* signal. If a pull channel is implemented, data must be valid when the passive issues the acknowledgement.

There are also the timing restrictions when dealing with data transfer handshaking. For instance, the setup time between the request signal transition and the valid data in a push channel. Every relationship between control and data signals must respect their respective time constraints.

2.3.4 Data Encoding

Asynchronous circuits use synchronization protocols in order to its elements to communicate and the handshake protocol is the most common approach, as explained before. In addition to the way the data is transferred, the way how it is encoded is also an important decision in a design. For instance, in synchronous circuits, data is usually encoded in *2n* distinct symbols represented by Boolean logical values (0 or 1), on n wires and the validity of these data signals is given by a clock signal along with control signals [BAR00].

In asynchronous designs there are different ways to encode data. Bundled-data protocols implement a situation where data signals use normal Boolean encoding to carry information and their validity is given by bundled request and acknowledge signals. Communication can then be implemented using four or two phases protocols [SPA01]. As Figure 2.14 (a) shows, a four phase bundled data communication starts with sender rising the request signal (*req*). At this moment the data must be valid. The receiver acknowledges the communication by rising the acknowledge signal (*ack*). After that, the sender sets the request low and the receiver subsequently sets the acknowledge signal low. At this point, a new transmission can take place. A similar operation takes place in the two phase protocol, the difference is that it avoids the unnecessary return to zero transitions of the control signals present in the four phase. In a two phase bundled data communication, every transition of the request and acknowledge represents a transmission, as Figure 2.14 (b) depicts.

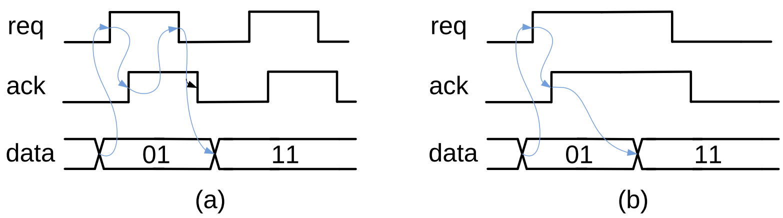


Figure 2.14 – Four phase (a) and two phase (b) bundled data transmission.

However, the assumption of bundled control signals requires extra care with the time constraints between data and control signals. Fortunately, DI codes present a solution that is robust to wire delays. The dual rail protocol encodes the request signal into the data signal making use of two wires per bit. Hence, there are no timing constraints for the data channel. However, the request must be computed from it and, therefore, demand extra hardware. In the dual-rail protocol each single bit requires two wires to be represented. Moreover, this codification can use either two or four phase communication protocols.

As it is shown in Table 2.4, in the four phase dual rail protocol logical 0 is represented by a low *d.t* and a high *d.f* while logical 1 is the opposite. The request is computed when *d.t* and *d.f* assume different logical values and to signal a low transition of the request, a spacer is used where *d.t* and *d.f* are set to logic 0. If *d.t* and *d.f* are logic 1 the data is invalid [SPA01]. Figure 2.15 (a) shows an example of a four phase dual rail data transmission. Firstly, the sender put valid data, "*01*" in the data channel. The receiver computes the request and acknowledges setting the *ack* signal high. When the sender reads the acknowledgment, it transmits a spacer to finish the handshaking. The receiver computes the spacer and set *ack* low. The sender can then start a new transmission.

Table 2.4 – Four phase dual rail codification for 1 bit of data [SPA01].

Value	d.t	d.f
Spacer	0	0
0	0	1
1	1	0
Invalid	1	1

The two phase dual rail protocol avoids the avoids the unnecessary return to zero transitions in the acknowledge signal and present a different data codification. In this protocol, logical 1 is represented by a transition in *d.t.* while switching *d.f* characterizes logical 0. The request is computed when every bit switch one of its wires. Figure 2.15 (b) presents an example of a two phase dual rail data transmission. Firstly, the sender issues a request putting "*01*" in the data channel. When the receiver computes the request it switches the acknowledge signal. From this point a new transmission can take place. For more detailed information about data encoding protocols one can refer to [BAR00] [SPA01] [TOM06].

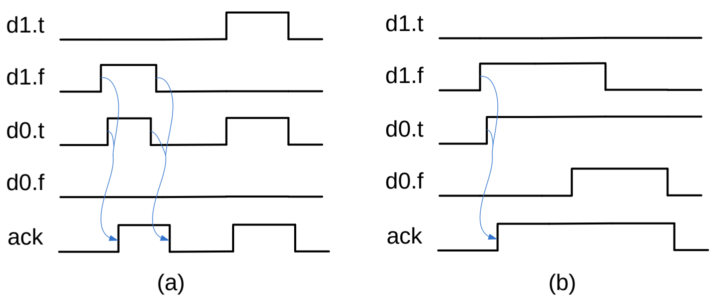


Figure 2.15 – Four phase (a) and two phase (b) dual rail transmission.

2.3.5 Metastability

Bistable circuits have two stable states that represent logical 1 and 0 values. In this scope, a static memory can be implemented by using cascaded inverters. An example of a static memory composed by two cascaded inverters appears in Figure 2.16(a). Figure 2.16(b) shows the voltage transfer characteristic (VTC) of this circuit in three graphs. As it can be seen in the last graph, the circuit has three possible states (A, B and C). Assume the value of V_{o2} is sampled at a given instant in time. If in this instant the circuit is stable in A or B, the sampled value will be logical 0 or 1, respectively. However while the circuit is transitioning, this value passes by point C, which is neither at logical 1 nor at logical 0. If a digital circuit uses (samples) this signal in this moment, for example to transfer its value to another bistable circuit, it may either store a logical 0, or a logical 1 or simply transfer the logically undefined value to the destination circuit. If this later situation happens, one must remember that physically this corresponds to N and P transistors simultaneously operating in a state of conduction at the destination of the sampled signal. The signal will eventually fall back to a stable state, but this can take an arbitrarily long time to occur. This is called a *metastable state*, and can imply devastating situations for the circuit, including transitory or permanent circuit failures [RAB03].

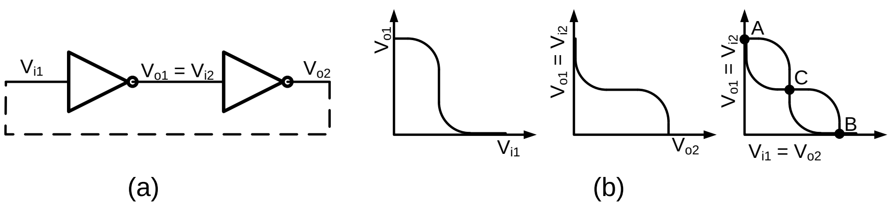


Figure 2.16 – Two cascade inverters (a) and their VTCs (b), based on [RAB03].

2.4 Balsa

Balsa is both a language to describe asynchronous circuits and a framework to synthesize such circuits [SPA01] [EDW06]. It was developed as a response to the need of a language to replace the Philips Tangram system [BER91] [KES01] in an existing Tangram application in the University of Manchester [BAR98]. In fact, the Balsa language can be viewed as an extension of the Tangram language. The approach adopted by the language is the syntax directed compilation into handshaking components. Balsa was developed and is maintained by the Advanced Processor Technologies Group of The University of Manchester [APT10].

The compilation of a Balsa description is transparent, language constructs are directly mapped into handshake circuits. The advantage is that it is relatively easy for the user to visualize the circuit-level architecture described by a balsa description. This way, language level changes reflect in predictable changes in the resultant circuit. Additionally, different technologies can be used to synthesize Balsa circuit descriptions, generating silicon or field programmable gate array (FPGA) netlist implementations [BAR00]. The back-end generated gate level netlists can then be imported into computer-aided design (CAD) systems to implement the described circuit. At present, Balsa generates netlists fully integrated with consolidated CAD systems [SPA01] including: Xilinx FPGA design tools [XIL10] and Cadence Design Framework [CAD10].

Figure 2.17 shows an example of a balsa description of a 2 place buffer. The input/output channels (I/O) of the buffer are declared in line 2, an input (*i*) and an output (*o*) of one byte. Two one byte internal registers are declared in line 3, *x1* and *x2*. The behavior of the circuit is given in lines 4-10. The circuit waits for a request in the input channel *i* and when it is asserted, the data in the channel is stored in register *x1*. Next, the data stored in register *x1* is copied to register *x2*. Finally, the circuit issues a request in the output channel and waits to transfer the data stored in register *x2*. Once the output communication finishes, the circuit will wait for a new input communication, due to the loop declared in line 5 and finished in line 9.

```
1  import [balsa.types.basic]
2  procedure buffer2 (input i : byte; output o : byte) is
3   variable x1, x2 : byte
4  begin
5   loop
6    i -> x1;   --input communication
7    x2 := x1;  --implied communication
8    o <- x2    --output communication
9   end
10 end
11
```

Figure 2.17 – Example of a two place buffer described in Balsa, from [EDW06].

The sequentiality of the handshakes is given by the operator *";"*. Therefore, in the given example, the input data follows through register *x1*, *x2* and then to the output, in this order. Figure 2.18 presents a waveform for the described two place buffer with random input values.

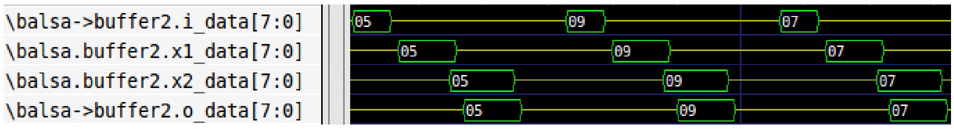


Figure 2.18 – Waveform for the circuit described in Figure 2.17 with random input values.

A more complex circuit description is give in Figure 2.19. The described circuit is a multiplier with parameterizable I/O channels size. Three internal registers, which width is variable as well, are required: *result*, *data_a* and *data_b*. The operation of the circuit consists in successive sums. Initially, the circuit waits for a request in the input channels in order to load its internal registers. A parallel operator *"||"* is used in line 14, in this way the two input channels, *data_in_a* and *data_in_b* wait for a request through the operator *"->"*. Once both of the channels are requested, and the internal registers loaded, the computation may start. The internal register *result* is reseted, in line 16, and the successive sums start through the line 17 loop. The partial result is added with the value of *data_a* for *data_b* times. After that, a request is issued in the output channel and the data is transferred. From there on, a new multiplication operation may start when both the multiplier and the multiplicand are loaded through the input channels once more. In line 25 a 16 bits multiplier, *mult16x16* is instantiated.

```
1  import [balsa.types.basic]
2  procedure multiplicador (
3         parameter size_a : cardinal;
4         parameter size_b : cardinal;
5         input data_in_a : size_a bits;
6         input data_in_b : size_b bits;
7         output data_out : size_a + size_b bits
8  ) is
9  variable result : size_a + size_b bits
10 variable data_a : size_a bits
11 variable data_b : size_b bits
12 begin
13  loop
14   data_in_a -> data_a || --load a
15   data_in_b -> data_b; --load b
16   result := (0 as size_a + size_b bits); --load result with 0
17   loop while data_b > (0 as size_b bits) then --"b" iterarions of sum
18    result := ((result + data_a) as size_b + size_a bits); --sum partial results
19    data_b := ((data_b - 1) as size_b bits) --decrement index
20   end; -- loop
21   data_out <-result --write result
22  end --loop
23 end -- multiplicador
24
25 procedure mult16x16 is multiplicador( 16, 16)
26
```

Figure 2.19 – Example of a parameterizable multiplier described in Balsa.

Figure 2.20 presents the waveform of a simulation of the 16 bit multiplier of line 25 of Figure 2.19 with an input of 2_{10} and 9_{10}. Firstly, the values are loaded in the internal registers and the partial result is reseted. Next, the circuit computes 2 times the sum of the partial result and 9. Finally, a request is issued in the output channel and the resultant value is made available.

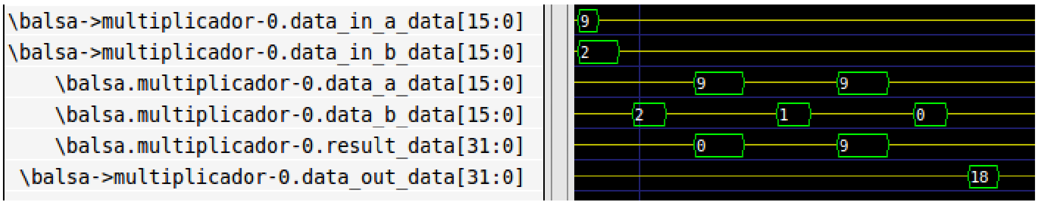


Figure 2.20 – Waveform of a 16 bit multiplier described in balsa for the inputs 2_{10} and 9_{10}.

2.5 Teak

Another option to synthesize Balsa descriptions is the open source back-end system designed in the University of Manchester: Teak [BAR09]. It consists in a new target parameterizable component set and synthesis scheme that aims the improvement of circuits described in the Balsa language. The tool optimizes Balsa descriptions synthesis by replacing data-less activation channels with separate control channels. Albeit the Balsa System allows different data encodings over two- or four-phase protocols, Teak implementations are typically QDI four-phase dual rail asynchronous circuits. Hence, circuits synthesized through this tool are limited to that choice of protocol and data encoding.

The components used in the Teak synthesis are abstract data and control flow manipulation elements as displayed in Figure 2.21. The Steer component conditionally distributes an input value to exactly one of its output. The fork splits a channel in *n* parts unconditionally. This is a delay insensitive form to send different parts of some information to distinct destinations. The Join element, on the other hand, unconditionally joins *n* channels in one single information flow and synchronizes information appearing at distinct instants on all its input channels. A Merge component selects one of *n* mutually exclusive inputs and let it available in a single output. Variables are used for storing values and have separate write and read sections. Operators represent any data transforming operations. Finally, a buffer is used for data storage and channel handshaking decoupling. A better explanation of these components can be found in [BAR09].

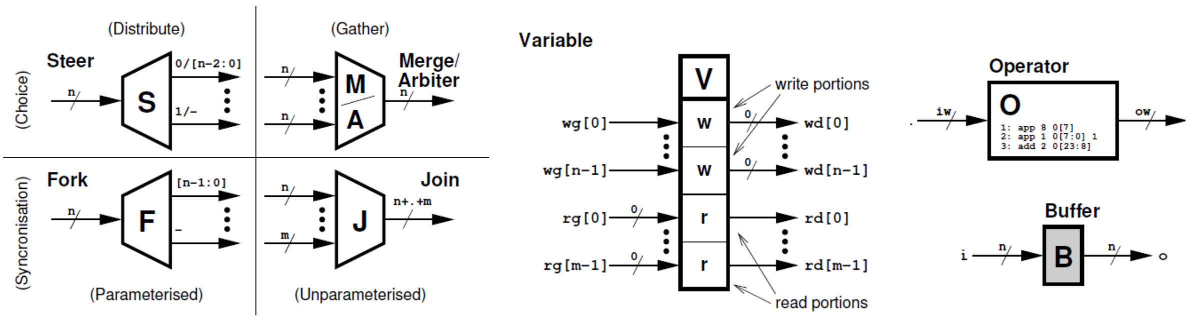


Figure 2.21 – Teak components, captured from [BAR09].

To make clear how Teak conducts the transformation of an asynchronous Balsa description into a diagram of components, this Section now depicts the processing of an example. A 4-bit sum and subtraction unit was described in Balsa language, as Figure 2.22 shows. First, two values must be loaded through *data_a* and *data_b*. Next, the desired operation is signaled through *ctr*. Depending on the logical value of *ctr* the circuit sums or subtracts the loaded values and copies the result to *data_out*.

```
import [balsa.types.basic]
procedure sum_sub (
        input data_in_a, data_in_b  : 4 bits;
        input ctr : 1 bits;
        output data_out : 8 bits ) is
variable data_a, data_b : 4 bits
begin
 loop
  data_in_a -> data_a ||
  data_in_b -> data_b;
  ctr -> then
   if ctr = 0 then data_out <- (data_a + data_b as 8 bits)
   else data_out <- (data_a - data_b as 8 bits) end
  end
 end
end
```

Figure 2.22 – Balsa description of a 4-bit sum and subtraction unit.

The diagram of the Teak netlist obtained through Teak synthesis is depicted in Figure 2.23. This netlist shows typical handshaking components, which are used by Teak: Merge (*M*), Fork (*F*), Join (*J*), Steer (*S*), Variable (*data_a-data_b*) and Operators (+, - and =). The C-Element is a fundamental primitive gate for implementing all these components. For instance, a 1 bit fork uses two buffers and one 2-input C-Element. A 1 bit Steer uses seven buffers, two OR gates and three 2-input C-Elements.

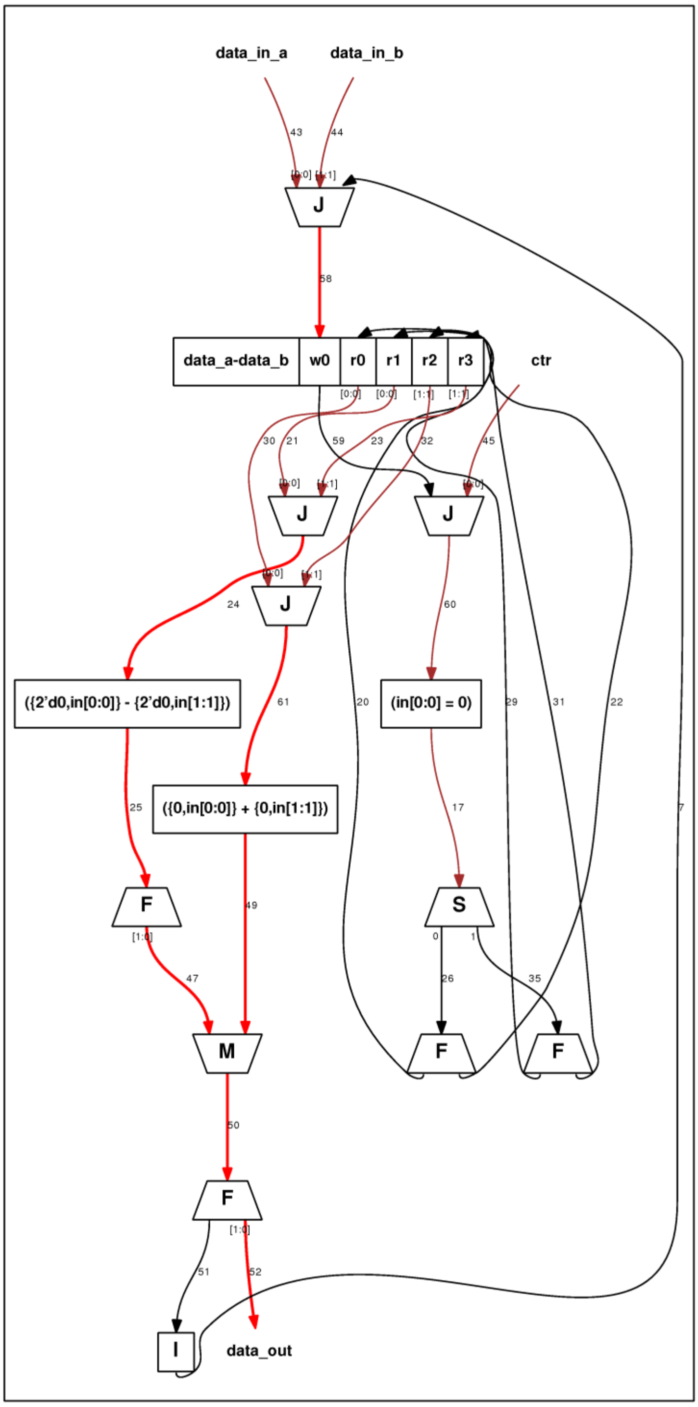


Figure 2.23 – Diagram of the Teak-generated netlist for the sum and subtraction unit.

The circuits generated by Teak do not perform very well in terms of performance when compared to the ones generated by the Balsa System [BAR09]. This is due to the fact that the system is still in its early stages of development and some functionalities of the tool do not present satisfactory results. However, due to the restricted set of standard cells that this system requires to generate a complete netlist for any circuit described in Balsa, the developed library will initially support Teak synthesis only, as well as flows starting with

VHDL descriptions helped by several manual operations. Future work includes expanding the set of gates in order to support Balsa System synthesis as well.

3 State of the Art in Open Source Standard Cell Libraries and Support for Asynchronous Circuits

"What one man can invent, another can discover."

Sir Arthur Conan Doyle

At present, the use of standard-cells in the design of ICs is a consolidated design style. Therefore, a wide variety of standard-cell libraries is on hand for different technologies, different processes and different purposes. Moreover, even free of charge libraries are available for educational institutions. However, asynchronous elements are usually not present in contemporary standard cell libraries. Designs that demand asynchronous elements usually design their own gates in a full-custom approach or use typically synchronous elements to build asynchronous ones.

Another major draw-back of the asynchronous paradigm is the enormous gap between the development of EDA tools for asynchronous circuits and their synchronous counterpart. Therefore, the design and implementation of asynchronous ASICs through typically synchronous EDA tools have also been proposed as an alternative.

This chapter presents an overview of current standard-cell libraries and design and implementation of asynchronous circuits in present.

3.1 Grad and Stine Standard Cell Library [GRA03]

Grad and Stine present a public domain Standard Cell Library for educational proposes that supports a fully automated design flow. The library contains combinational cells, such as NANDs and NORs, with various drive strengths and a positive edge Flip-Flop implementation for sequencing operations. The cells were designed for the American

Megatrends INC (AMI) [AMI10] 0.5µm process, which allows fabrication through MOSIS [MOS10] educational program. The design flow of the library used tools from the Cadence Framework [CAD10] and its physical and electrical characteristics are described through LEF and LIB text-based files, respectively.

A fully automated Semi-Custom IC design flow is also distributed through a set of script templates. The design flow uses tools that are widely available in the market and educational institutions. Synopsys [SYN10] Design Compiler is used for synthesis and Cadence Silicon Ensemble for place and route. Moreover, the scripts are highly parameterizable in order to provide full control over the final layout to the designer.

This library was not designed to support asynchronous design styles.

3.2 Djigbenou and Ha Standard CMOS Library [DJI07]

The library presented by Djigbenou and Ha, from Virginia Tech VLSI for Telecommunications (VTVT) Lab, targets the TSMC [TSM10] 0.25 µm process with a supply of 2.5 Volts. It was designed using commercial design and simulation tools and provides various primitive gates, multiplexers and flip-flops with a variety of drive strengths. In total, the library contains 84 cells.

In order to provide a glance of the real world design environment, the presented library is available for educational purposes free of charge. The supported Semi-Custom IC design flow uses a VHDL or Verilog behavioral description that is synthesized and simulated using Synopsys tools. The synthesized design is then imported into Cadence Framework to generate the IC layout through place and route tools. Still, this library does not have any typical asynchronous element.

3.3 Chong, Gwee and Chang Design Methodology [CHO07]

The lack of appropriate support to design asynchronous ASICs is one of the main drawbacks for this paradigm in present. Hence, Chong, Gwee and Chang present a methodology of designing asynchronous circuits using conventional IC design tools and standard cell libraries.

In the design methodology proposed in [CHO07], the circuit is implemented as a latch-based pipeline and an asynchronous latch controller, composed of conventional elements, is proposed for the synchronization of each stage. To synthesize the design, the architecture of the circuit is divided in three: data path circuits, asynchronous latch controllers and state machines. The first, is basically synthesized through a normal combinational logic synthesis. The latch controllers, on the other hand, cannot be synthesized because the synthesis tool would end up generating a very different circuit through its optimization approach. Finally, the state machines are synthesized through conventional RTL synthesis with one delay assumption: the delay for a change of state has to be lower than the worst delay of the combinational logic.

To validate their design methodology, an asynchronous finite impulse response (FIR) filter was implemented in a 0.35 µm CMOS process. The design flow counted with conventional commercial EDA tools from Synpsys and Cadence.

3.4 Cortadella et al. Desynchronization Paradigm [COR06]

Cortadella et al. present a paradigm for automating the design of asynchronous circuits from synchronous specifications. In [COR06], different protocols for desynchronization are addressed and a desynchronization flow is proposed. This flow consists, in essence, of switching each register of a pipeline for two latches and implementing a control for each local handshaking, in order to eliminate the global clock. Additionally, combinational logic delays are matched in their respective synchronization lines.

The work validates the proposed paradigm through solid implementations, including a desynchronized version a DES core and a DLX processor [HEN02]. A SoC containing the DLX processor and two on chip memories was synthesized, through typically synchronous libraries and tools, fabricated, and tested. The chip supported multiplexed clock that enables it to operate in both the synchronous and the desynchronized mode. A comparison between the two implementations proved that the presented paradigm did not introduce significant penalties in the design and, more importantly, achieved benefits of asynchronous circuits, such as lower electromagnetic emission and lower power consumption. However, for its nature, this paradigm provides just a taste of the advantages of asynchronous circuits.

3.5 Ferretti Standard Cell Library [FER04] [FER06]

In these works, Ferreti presents a Single-Track Full-Buffer Standard-Cell library that was designed to support high-speed area-efficient asynchronous nonlinear pipeline design. The library was designed for the TSMC [TSM10] 0.25 µm process and validated through silicon implementations. Each cell of the presented library employs an asynchronous communication channel that can help automate the implementation of local handshaking.

The designed STFB Standard-Cell is freely available through the MOSIS Educational

Program [MOS10]. As far as the Author could verify in the available literature this library is the only freely available standard-cell library for asynchronous ASIC design. The library supports dual-rail and 1-of-3 data encoding and the back-end design flow using the presented library can be done through Synopsys and Cadence tools.

3.6 Ozdag and Beerel QDI Templates [OZD04] [OZD06]

Ozdag and Beerel present the implementation of an asynchronous low-power high-performance sequential decoder along with the design flow of a QDI precharged half buffer (PCHB) standard cell library. The design flow utilized to generate the gate-level cell library counts with the following steps: Cell Specification, Cell Design and Cell Abstract. For each cell it was designed the physical layout, transistor-level schematic view, symbol and behavioral (Verilog) views according to its specification. Moreover, spice simulations were used to characterize the electrical behavior of each of the cells.

The resulting cell library, which is available through MOSIS [MOS10] for the TSMC [TSM10] 0.25 µm process, counts with a set of 40 dynamic cells, to implement function blocks, and 10 cells to implement control logic. However, the cells are significantly limited when it comes to output driving strength and many of the cells have a short range of output load capacitance. These limitations are due to the fact that the designed library was generated in order to implement a specific application and there was no fine grain optimization regarding the design of the gates.

3.7 TIMA and LETI asynchronous standard cell library [MAU03]

The only fairly way to experience asynchronous ASICs is to have an asynchronous standard cell library available. From this assumption, TIMA and LETI (both French laboratories) developed a library to support such designs. A 130nm gate length version of this library, called TAL-130 is presented in [MAU03]. In this work, the flow used to design the library is detailed and compares the results of implementing QDI circuits using TAL-130 and using a standard synchronous library (through AO222 gates).

At present, a 65nm transistors gate length version of this library has also been designed. This library is called TAL-65 and is fully validated through characterization process and silicon implementation. Two sets of cells compose the library, a set of C-Elements and a set of PCHB templates. In the former, different variations of C-Elements are available. Among these are two, three and four input cells and asymmetric cells.

However, this library is not freely available. The information obtained about it was courtesy of TIMA and LETI.

4 LIBRARY DESIGN METHOD AND TOOLS

"To let the brain work without sufficient material is like racing an engine. It racks itself to pieces."

Sir Arthur Conan Doyle

The main goal of this work is to describe the design and implementation of a standard cell library called ASCEnD ST65, specific for building asynchronous ASICs. In particular, this Chapter presents the adopted design flow proposed and used to develop ASCEnD ST65. The contents of the library are the subject of Chapter 5 .

This flow can be divided in three main sections, with the dependency relations among them depicted by Figure 4.1.

Figure 4.1 – Dependency between the main steps of the adopted design flow for the standard cell library.

4.1 Physical Design Guidelines for the Library

As it was explained in Chapter 2 , the standard cell approach to design ASICs makes use of pre-designed and pre-verified functional blocks in order to build complex ICs. These blocks can be seen as Lego bricks and, in order for them to fit side by side, rules are needed to define physical characteristics like cell height and terminals positioning. Moreover, a PMOS-to-NMOS ratio must be defined to obtain balanced cells, with rise and fall propagation times the more similar the possible. Finally, different designs of a same gate are usually demanded to support different requirements, such as design density or circuit performance. Therefore, a metric to define the driving strength of cells must be specified as well.

4.1.1 Tightened Physical Design Rules for Manufacturability

The stance of the doping layers, power rails and connection pins along with the width of the power rails and the height of the cell were standardized for each cell of the designed library. The width of the cells varies, albeit respecting the routing grid. Moreover, in deep submicron technology nodes, minimum DRC rules are not enough to guarantee a significant die yield. To do so, design for manufacturability rules (DFM) are essential in 65nm nodes and below, where a slight variation of process parameters may ruin a whole die [WON09].

This work proposed a set of layout guidelines in order to obtain designs that are more likely to respect all DFM principles and improve yield. A fundamental choice is not to use minimum size, but at least 2 manufacture grid points bigger geometries. The set of adopted DFM rules is summarized in Figure 4.2, where *A* represents the polysilicon extension from the gate region, which was defined to be 4 manufacturing grid points bigger than the minimum size rule. When the source/drain runs in parallel with the overlapping polysilicon, this size must be even bigger (6 manufacturing grid points) to avoid unwanted short circuits. In patterns, the distance between lanes of a same layer, represented as *C*, must be at least 4 manufacturing grid points bigger than the minimum size rule. Contact redundancy is recommended whenever possible to reduce failure by faulty contacts, as depicted in *D*. The same is recommended for vias to interconnect different metal layers. Finally, the enclosure of contacts (*E*) must be at least 2 metal pitches bigger than the recommended. Once more, the same rule applies to vias. All the DFM rules adopted were obtained from the design rules specification in the design kit manuals.

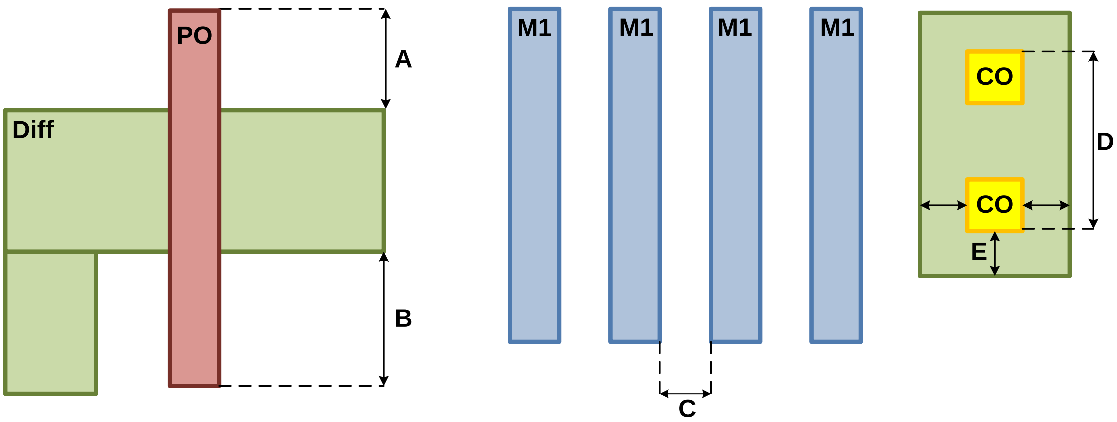


Figure 4.2 – Adopted set of DFM rules concerning IC layers. PO is polysilicon, Diff is diffusion, CO is contact and M1 metal 1 layers.

The architecture of each standard cell has also been defined in order to be compatible with the library provided by the foundry. In this way an IC can be implemented through the use of cells from both libraries. Figure 4.3 presents the defined architecture for every gate.

The height of the gates is fixed in 2.6µm, while the width can vary, albeit it must be an even multiple of 0.2µm. The power rails have standard widths of 0.56µm and the doping layers have predefined sites and extension over the standard cell.

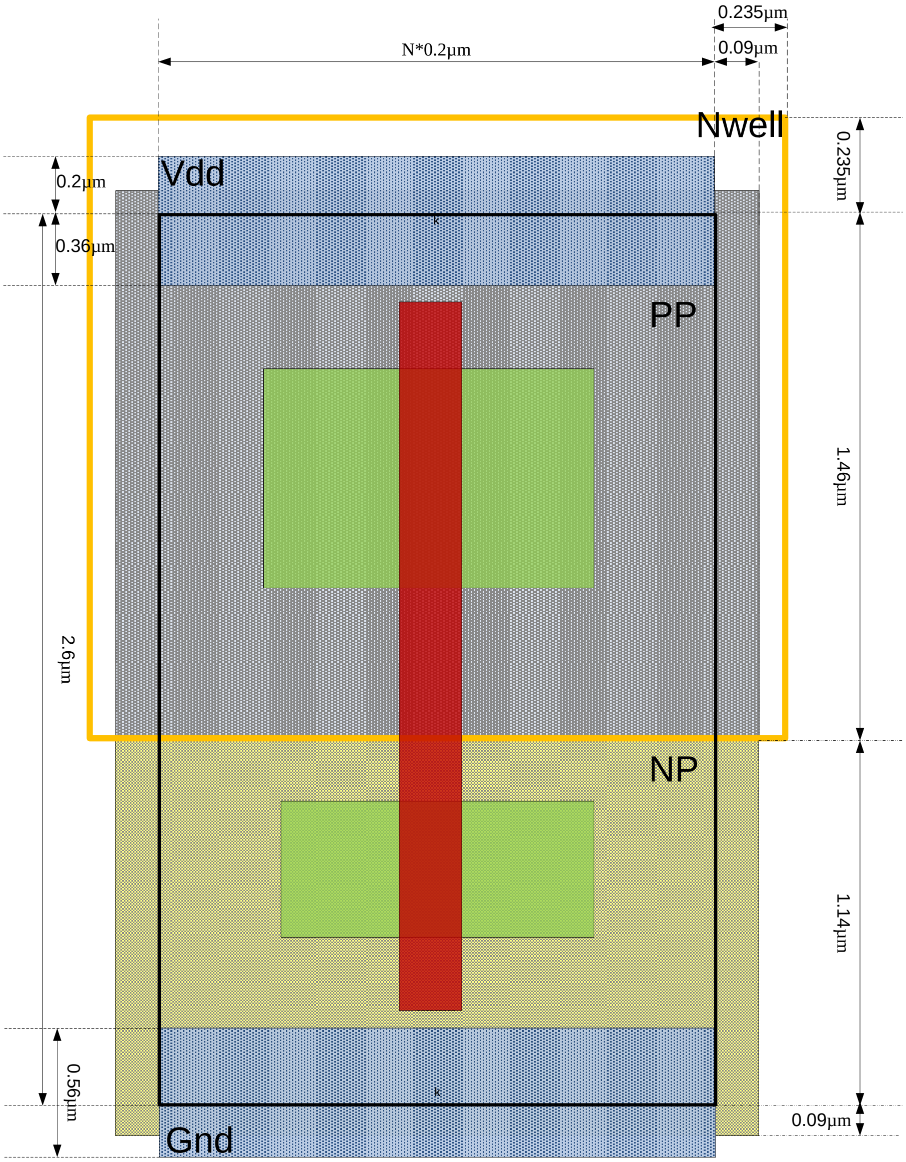


Figure 4.3 – Standard layout architecture. The red polygon is polysilicon and the green rectangles are diffusion. NP and Nwell are the negative doping layers and PP is the positive doping layer.

Another important definition for the physical design of the library is the fact that all the cells will be tapless, where a tapless cell consists of a standard cell without connections to bulk silicon. This is done for compatibility sake with the basic library of standard cells provided by the foundry. The use of tapless cells enables the design of hybrid ICs, using cells from the asynchronous standard cell library and the typical library provided by the foundry. In the approach described here as in the basic 65nm library of ST the polarization of the substrate is not given by each gate, but by special elements called *tapping cells*. These cells are already available in the library provided by the foundry and when generating the layout of an IC, they are placed in stripes separated by predefined distances.

4.1.2 Defining the PMOS-to-NMOS Transistors Ratio

The PMOS-to-NMOS transistors ratio is a constant that defines the size (more precisely, the width, since the length is fixed) of the PMOS as a function of the size of the NMOS transistor, as Equation 4.1 establishes. The method adopted by this work to define a PMOS-to-NMOS ratio consisted in describing the schematic of an inverter, the most elementary logic gate, using the SPICE language and simulating it in a test circuit. Cadence Spectre was the simulator used [CAD10]. This choice is a reflex of its availability and widespread usage. The test circuit consisted of an inverter with an output of 0.5 pF and an input slope of 0.033 ns. These values were obtained from the design kit manual, where it is specified typical simulation scenarios.

Equation 4.1 – Size of PMOS transistor (W_p) as a function of the size of the NMOS transistor (W_n) and the PMOS-to-NMOS ratio (β).

$$W_p = \beta \times W_n$$

Firstly, an initial PMOS-to-NMOS ratio was obtained by setting the NMOS to its minimum size and varying the PMOS from its minimum size to a maximum size of two fingers[1] (an extreme transistor size for the aimed technology process), respecting the defined maximum cell height and doping layers stances. Different simulations were performed for each PMOS size variation and a ratio of 2.19 was obtained from the simulation results. As Figure 4.4 shows, the best PMOS-to-NMOS ratio was found for a PMOS of 0.295µm with a minimum sized NMOS (0.135µm). A total of 244 simulations were performed, each varying by 5nm the PMOS size for one finger implementations and by 10nm for two fingers.

[1] Since the maximum cell height and doping layers stances restrict the maximum size of a single transistor, the only way to implement a transistor bigger than this maximum size is to have transistors in parallel. In these cases, the widespread folding technique can be applied to obtain bigger transistors, inflicting a smaller area penalty than implementing two separated parallel transistors. The technique consists in utilizing the drain/source of the transistor to generate a next "finger" and connecting the drain/source nodes together to implement two parallel transistors. Finally, the gates are connected to implement the equivalent folded bigger transistor.

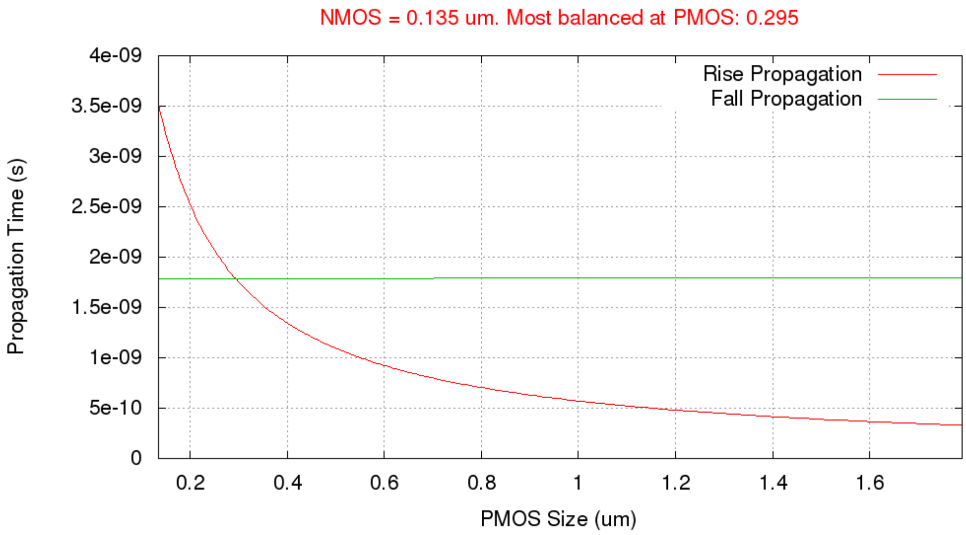


Figure 4.4 – Simulations results for an inverter with a minimum sized NMOS transistor (0.135µm) with the PMOS transistor size varying from minimum to a maximum size of two fingers. A total of 244 simulations were performed, obtaining the best PMOS-to-NMOS ratio of 2.19.

In order to obtain a more reliable ratio, the same flow used to obtain the most balanced PMOS for a minimum sized NMOS, concerning to rise and fall propagation delay, was performed for different NMOS sizes. Appendix A presents the simulation results for all performed variations of the NMOS size. Table 4.1 summarizes the obtained PMOS-to-NMOS ratio results. Finally, an average of PMOS-to-NMOS ratio of 2.11 was obtained through the performed simulations.

Table 4.1 – Summarized results for PMOS-to-NMOS ratio simulation different NMOS sizes.

NMOS size (µm)	PMOS size (µm)	PMOS-to-NMOS ratio
0.135	0.295	**2.19**
0.2	0.43	**2.15**
0.3	0.645	**2.15**
0.4	0.855	**2.14**
0.5	1.03	**2.06**
0.6	1.24	**2.07**
0.7	1.45	**2.07**
0.8	1.67	**2.09**

4.1.3 Output Driving Strength

Different designs of a same gate are usually required to assist different needs, such as density or speed. The former need entails to build smaller transistors, so that the final design will use less area and consume less power, while the latter need leads to larger and, consequently, faster transistors with good driving capability, albeit with an area and power consumption penalty [MIC94] [JAM99] [RAB03]. Simulation results prove that bigger transistors provide faster, although more power consuming, cells. Figure 4.5 shows the results of multiple simulations for a ring of 11 identical inverters with different transistors size for a 65nm gate length technology process. The ratio used to dimension the transistors was 2.11, i.e. an inverter with a PMOS 2.11 times bigger than the NMOS, which in average provides the best balance between rise and fall cell delays.

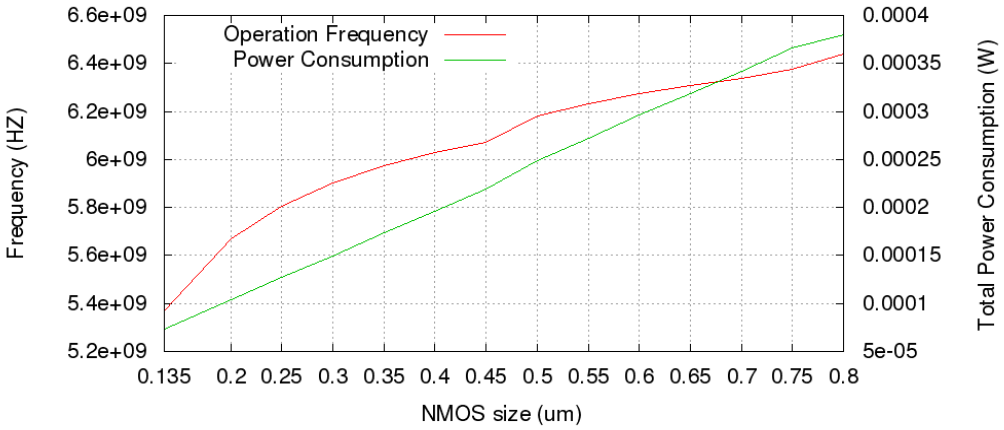


Figure 4.5 – Simulation results for different cell dimensions in a ring with 11 identical inverters, for a 65nm gate length technology process using a PMOS-to-NMOS ratio of 2.11.

Having different implementations of each cell improves the performance of the circuit that may achieve near full-custom performance designs [HAS03]. That happens because when the cell does not need to drive a big load, a small transistor version can be used, which leads to lower power consumption. On the other hand, for instances that need to drive a large load, a cell with larger transistors can be employed to improve performance. Therefore, an indispensable step for the design of a standard cell library is to define different output driving strengths.

The methodology adopted in this work is to define output driving strengths through simulations using Spectre. Firstly, it was to assumed that a X1 drive cell would drive the same load that a minimum sized inverter in a period of time of 1ns, a X2 would drive two times this value, a X3 three times and so on. Therefore, the first step was to define the maximum load (dis)charged by an inverter with minimum size transistors, respecting the defined PMOS-to-NMOS ratio. The test circuit consisted of a minimum sized inverter with an input slope of 0.033ns (as defined by the design kit user guide) and varying output loads, from 0.01pF to 0.40pF in 0.01pF steps, because it was verified by previous simulations that these values represent extreme values for a minimum sized transistor. The results showed that the circuit was capable to drive a maximum of 0.27pF in 1ns, as Figure 4.6 shows.

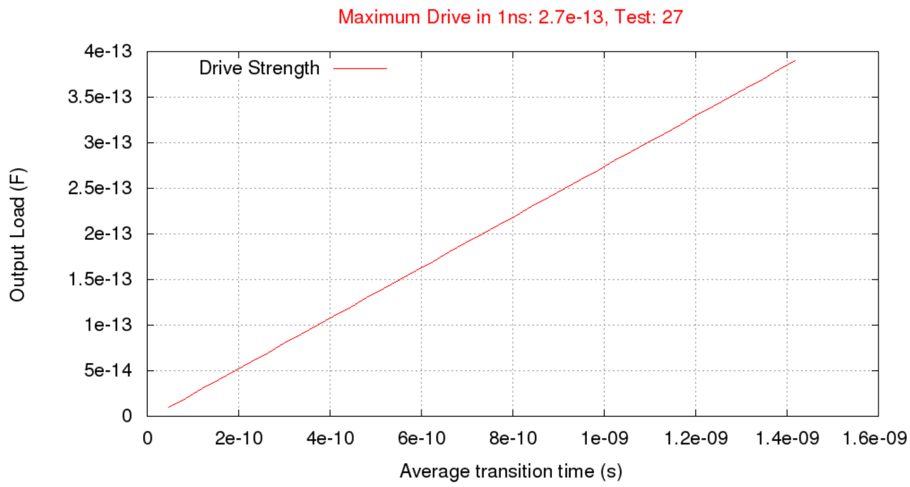


Figure 4.6 –Simulation results of a minimum sized inverter with varying output loads and an input slope of 0.033ns.

Next, different simulations were performed to define what transistor size would be capable to drive 2 times the X1 drive (0.54pF/ns). In order to do so, a test circuit composed by an inverter with a 0.54pF capacitance in its output and a slope of 0.033ns in its input was described in the Spice language. The inverter was simulated for different sizings, varying from minimum, 0.135µm NMOS and 0.285µm PMOS, to a maximum size of two fingers in the NMOS (1.4µm). The size of the PMOS was always fixed in 2.11 times the NMOS transistor size. In the results, depicted in Figure 4.7, the red line represents the time taken for each transistors dimension to drive the output load, while the green line represents the balance for rise and fall propagation delays. The point where the red graphics cross the 1ns line is the best inverter dimensions for that output drive. Moreover, the closer the green line is to 1ns, the more balanced is the cell. Therefore, it is determined that a X2 cell requires an NMOS of 0.285µm and a PMOS of 0.600µm.

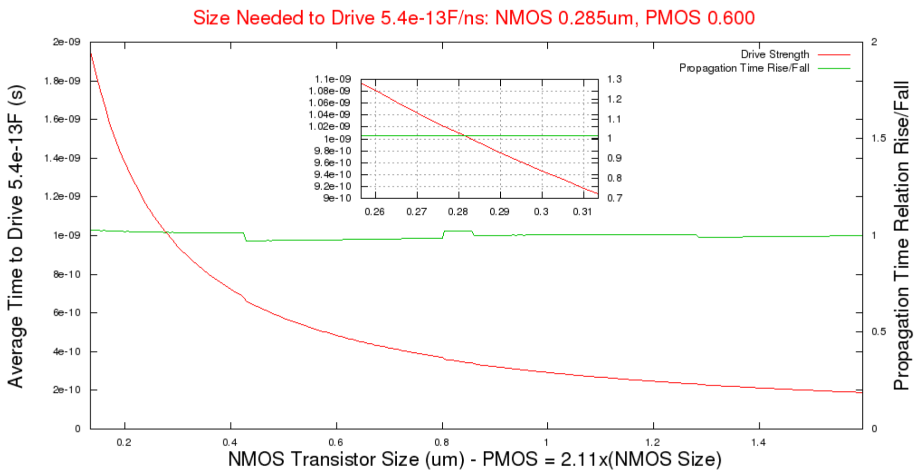


Figure 4.7 – Simulation results for an inverter with varying transistor sizes, an input slope of 0.033ns and an output load of 0.54pF. The results show that the transistor size required to drive the output load in 1ns is an NMOS of 0.285µm and a PMOS of 0.600µm.

The same method was used to specify the transistor sizing for X3, X4, X5, X6, X7 and X8 inverters. The complete set of results for each drive can be found in Appendix A. Table 4.2 presents the summarized results for the simulations performed to define the transistor sizing for each output drive.

Table 4.2 – Summarized simulation results for different output drives of an inverter.

Drive	NMOS size (µm)	PMOS size (µm)	Strength (pF/ns)
X1	0.135	0.285	0.27
X2	0.285	0.600	0.54
X3	0.430	0.905	0.81
X4	0.575	1.210	1.08
X5	0.725	1.530	1.35
X6	0.860	1.820	1.62
X7	1.000	2.100	1.89
X8	1.150	2.420	2.16

4.2 The Design Flow for Cells in the Library

Figure 4.8 shows a summary of the method adopted for the design portion of the flow proposed in this work to implement the standard cell library. This method consists of three main steps for each cell: Specification, Design and Validation. The rest of this Section gives an overview of the cell design flow, while Sections 0through 4.2.7 detail all the intermediate steps of the process.

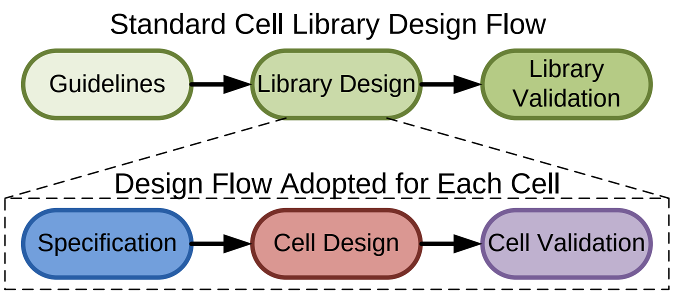


Figure 4.8 – Adopted flow to design each standard cell of the library.

Figure 4.9 depicts the flow in detail, along with the tools used at each step. The first step is to specify logical and electrical behavior, functionality along with requirements and constraints, of each cell. The schematic of the cell circuit is then described and exported to a text-formatted file in the Spice language. A specially designed tool, developed in the C++ language, called Ring Oscillator Generator (ROGen), automatically generates a simulation circuit in the form of a ring oscillator to test the cell for different transistor size combinations in a total of 1219 variations. Each ring implementation is simulated and some of its characteristics are measured. The results are analyzed and compared to define two versions[1] of the cell, performance driven (BS) and low power consumption driven (BP). This process is also fully automated by another specially designed tool, also implemented in C++ called Cell Specifier (CeS). From this point, the physical view of each version is obtained using the 65nm process STMicroelectronics design kit [STM10]. Each physical view is the result of a hand design using the Cadence Virtuoso layout editor. Next, equivalent circuits are extracted from the layout producing not only the drawn but also the parasitic devices. This step employs the Mentor Calibre tool. Each extracted circuit have its electrical behavior characterized, with Encounter Library Characterization (ELC), and if it meets the specification, an abstract view, a symbol view and a Verilog view are generated and the cell follows to the library. If the extracted circuit does not meet the requirements, its layout is redesigned and the design and validation steps are repeated.

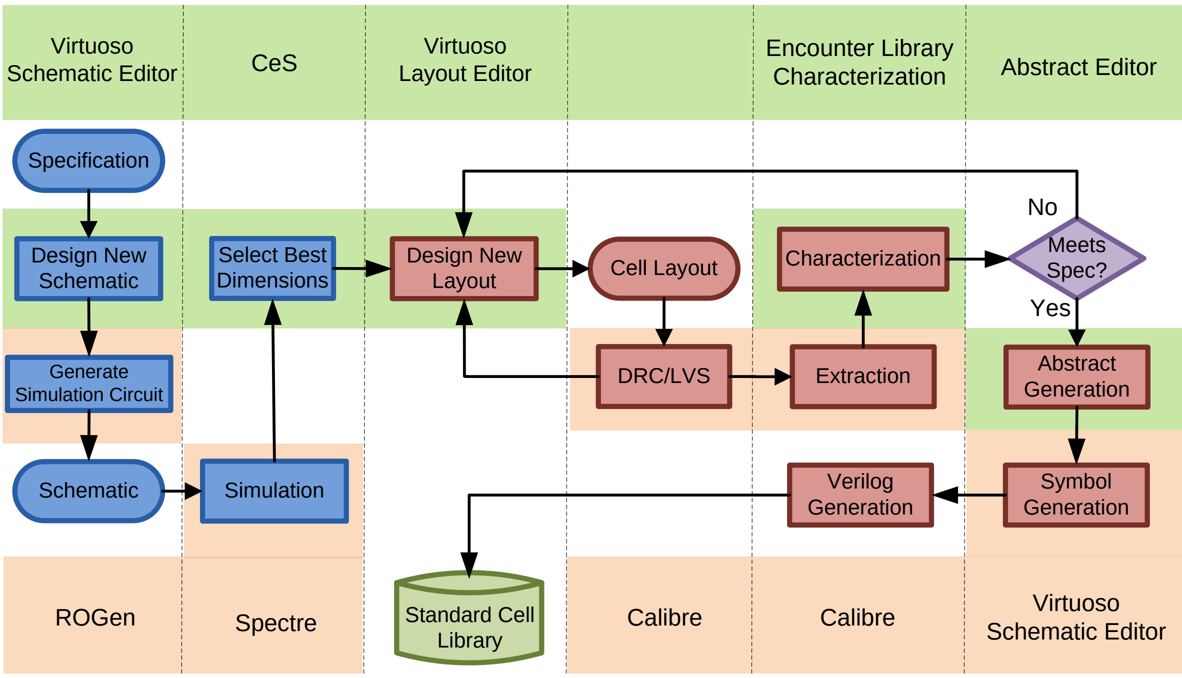


Figure 4.9 – Adopted design flow for each gate of the designed standard-cell library and required tools. The three main steps are: Specification (blue), (Physical) Design (red) and Validation (purple). Actions are represented by boxes, decisions by diamonds, descriptions as rounded corners boxes and repositories as a green cylinder. The tools used in each step appear in the top and bottom stripes.

[11] Every C-Element cell in the library will have two implementations, a low power driven and a performance driven ones. However other cells have only one implementation. This is due to the fact that, because of their nature, the approach used to specify C-Elements could not be implemented. Therefore, another approach was adopted, generating a single version of the cell, that is most balanced in terms of rise and fall propagation delays.

The tool used to describe the schematic of the cells and export them to symbols is the Virtuoso Schematic Editor, designed by Cadence. This choice is a reflex of the compatibility of the design kit with the Cadence Framework, as well as its availability in the resource set to develop this work. For compatibility sake, the chosen layout design tool is the Virtuoso Layout Editor, also part of the Cadence Framework. The tool used to verify the layout, design rule check (DRC) and layout versus schematic (LVS), is Mentor Calibre. Moreover, Calibre is also used to extract the circuits from the designed layouts. This choice derives from the possibility to seamlessly integrate the design kit and the Cadence Framework with this tool. The chosen simulation tool is Cadence Spectre for reasons identical to the choice of the layout editor. The tools used to automate the process of specifying the standard cells were designed by the author in C++ and are called ROGen, used to generate ring oscillators, and CeS, used to select transistors dimensions.

4.2.1 Specification

The first step of the adopted flow to design a standard cell is to specify its functionality and its electrical requirements. The functionality is given as a logical expression defining the output(s) as a function of the input(s) of the circuit. Equation 4.2 shows an example of a standard cell logical specification, in this case a 2 input C-Element.

Equation 4.2 – Example of logical expression defining the output of a 2 input C-Element as a function of its inputs. *Q* is the output and *A* and *B* the inputs.

$$Q_{i+1} = (A \wedge Q_i) \vee (A \wedge B) \vee (B \wedge Q_i)$$

The electrical requirements consist in the required speed for a cell to charge or discharge its output(s). Basically, the electrical specification of a standard cell is a tradeoff between area, power and speed [MIC94] [RAB03]. As seen in previous sections, high performance gates require larger transistors and, as a consequence, consume more power and demands more silicon area. Therefore, usually different cells are specified for a same logical function in order to support different output driving strengths.

Additionally, the definition of the name of the standard cell and its connection terminals is also part of its specification. For instance, say Equation 4.2 defines the functionality of the cell GPSVT_CMUL2, being *A* and *B* its inputs and *Q* its output. Assuming that the required output driving strength is the equivalent of a X4 cell, then the cell must drive 1.08pF in 1ns. Firstly, a schematic of the circuit that implements the specified logic is generated through Virtuoso Schematic Editor, depicted in Figure 4.10.

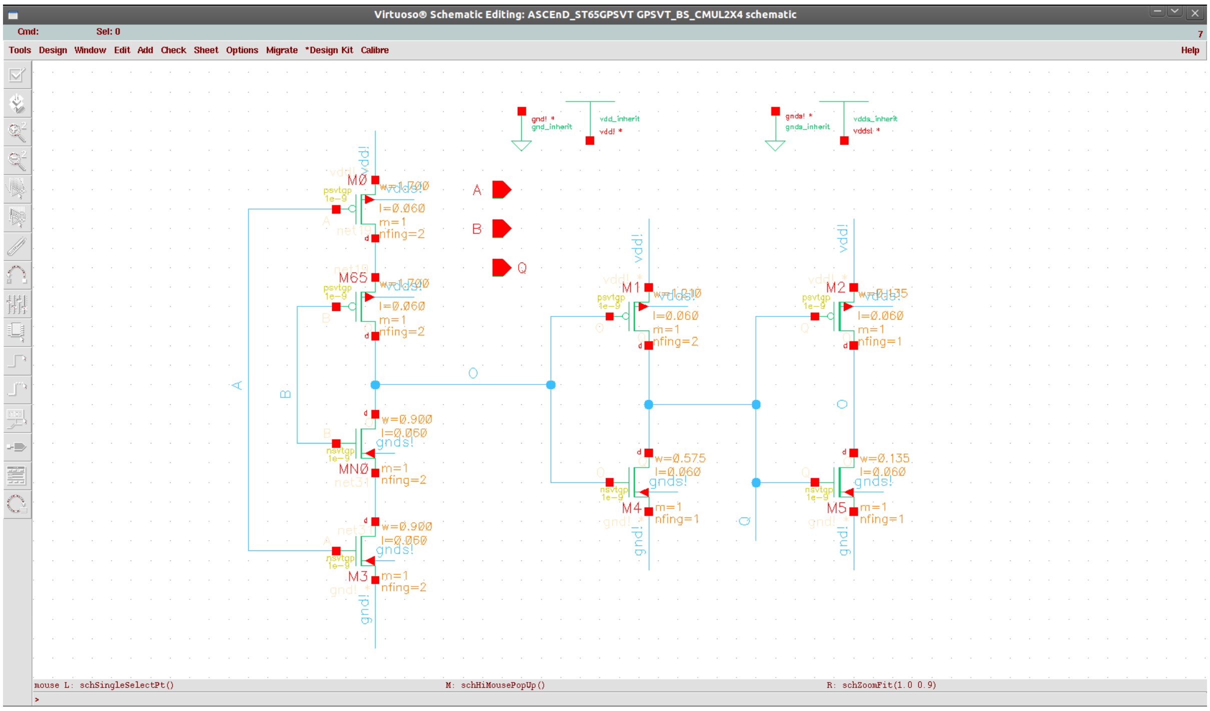


Figure 4.10 – Example of schematic design using Virtuoso Schematic Editor.

Depending of the characteristics of the circuit, two different flows can be implemented to define the transistors size. In this case, since the drive of a C-Element is given by an inverter (composed by the transistors *M1* and *M4*) the sizing of these transistors is fixed in 0.575µm for the NMOS and 1.21µm for the PMOS, as defined for X4 cells in previous sections. Moreover, the inverter composed by *M2* and *M5* is used only to keep a previous state. Therefore it is fixed in minimum size for the NMOS and the PMOS. The remaining transistors have a variable size, until a dimension that best drives the output inverter is defined. In this case, two versions are specified, one that drives as fast as possible without concerning with power consumption and another one that best drives it as a function of performance and power consumption.

In order to do so, the designed schematic is automatically exported to a Spice description. The ROGen tool is used to automatically generate an oscillator ring for the described circuit, which gate level view is represented in Figure 4.11, also in SPICE language. The generated description varies the size of the drive transistors through the *.alter* function from minimum size transistors to three fingers transistors at maximum size. The minimum transistor size is defined by the foundry and the maximum by the adopted standard cell height. Additionally, due to the nature of a C-Element gate, a NAND port is also included in the ring to make it oscillate. However, the effect of this gate over the ring circuit is normalized over the simulations and results comparison.

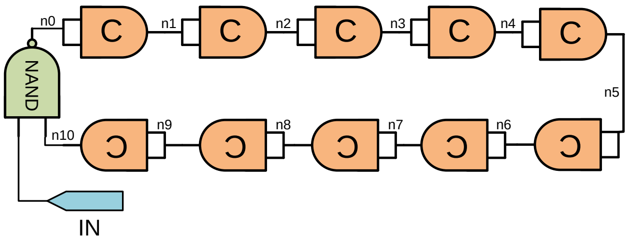


Figure 4.11 – Example of a gate level view of a simulation circuit automatically generated through a specially designed tool. When the *IN* pin is at logical 0, the circuit is static with all the nodes in logical 1. However when the input is asserted, the circuit starts to oscillate.

Information about electrical behavior of each variation of the circuit is obtained through the *.measure* function. Such information counts with: dynamic and leakage power of the whole ring, rise and fall propagation and transition delay of each cell and operating frequency of the whole ring. The leakage power was measured as the average power consumed from the power source while the circuit is quiescent. On the other hand, the dynamic power is the average power consumption from the source when the ring is oscillating.

The fall and rise propagation delay are measured as the delay of a signal to propagate through the cell under simulation, in this case a 2 input C-Element. For instance, from Figure 4.11, the rise propagation delay can be measured as the time that takes to node *n6* to rise after *n5* has risen. However, it is very important to measure nodes between the cells under simulation, in other words, one could lose precision if measuring nodes *n0* or *n10* due to the effect of the NAND port in the I/O of the cell to be dimensioned. Figure 4.12 shows an example of a measurement of the fall and rise propagation delays for a simulation of the oscillator ring. The adopted switching thresholds for rise and fall are 40% and 60% of Vdd, respectively. This definition is recommended in the manuals of the design kit.

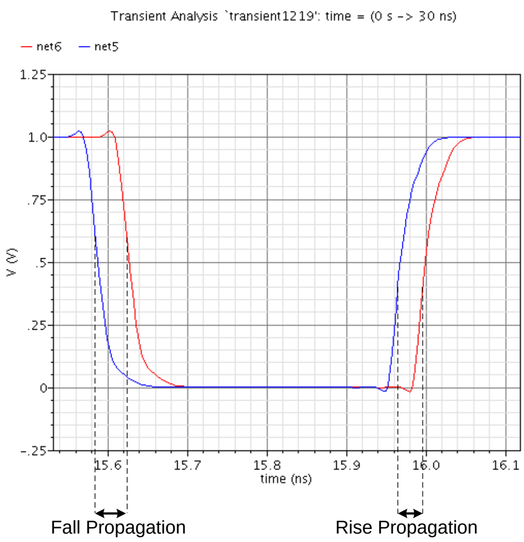


Figure 4.12 – Example of propagation delay measurement. The graphics were generated by Cadence Wavescan [CAD10].

The rise and fall transition delay, are the transient delays of the output of the cell to be dimensioned. For instance the rise transition is measured as the time that it takes for the output of the cell, say *n6* from Figure 4.11, to switch from logical 0 to logical 1 when it starts to rise. Figure 4.13 depicts the fall and rise transition of a simulation of the oscillator ring. The transient time is between 20% and 80% of Vdd, as recommended by the manuals of the design kit.

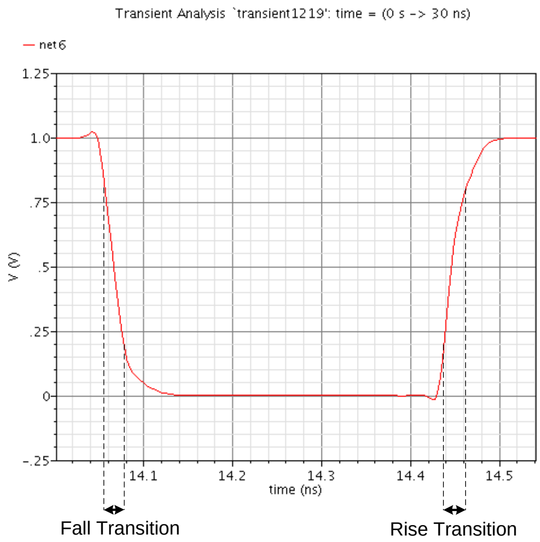


Figure 4.13 – Example of transition delay measurement.

The operating frequency is measured for the whole ring. It is a metric that defines how many times per second a signal switches to logical 1 (or logical 0). For instance, the time between two logical rises in n6 is the inverse of the operating frequency. Figure 4.14 shows an example of how to measure the frequency of the oscillator ring.

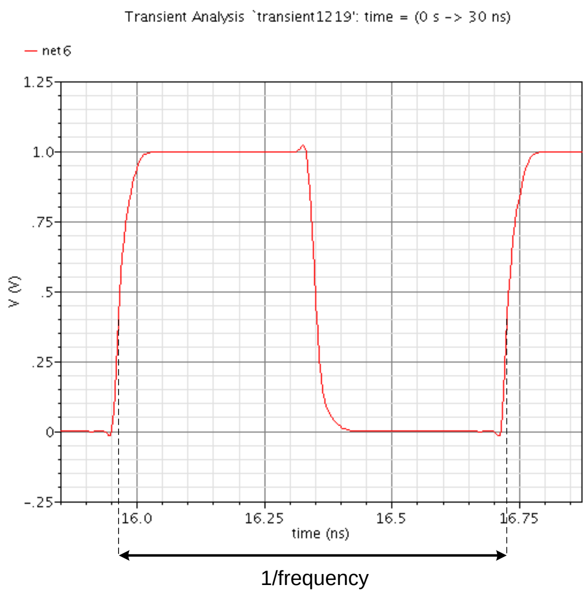


Figure 4.14 – Example of frequency measurement.

From a Spectre simulation, the measurements performed for each variation of the circuit are exported to a text based file. A second tool, CeS, is used to analyze the results and chose for a dimension of PMOS that best drives the cell for each NMOS transistor size variation. The result is the set of the best PMOS/NMOS combination for each sizing variation. From this set, the fastest combination is selected as a "best for speed" (BS) version and the sizing that consumes less power as a function of the performance is selected as a "best for low power" (BP) implementation. The described automated transistor sizing flow is represented in Figure 4.15. A total of 1219 simulations are performed and analyzed before selecting the best dimensions.

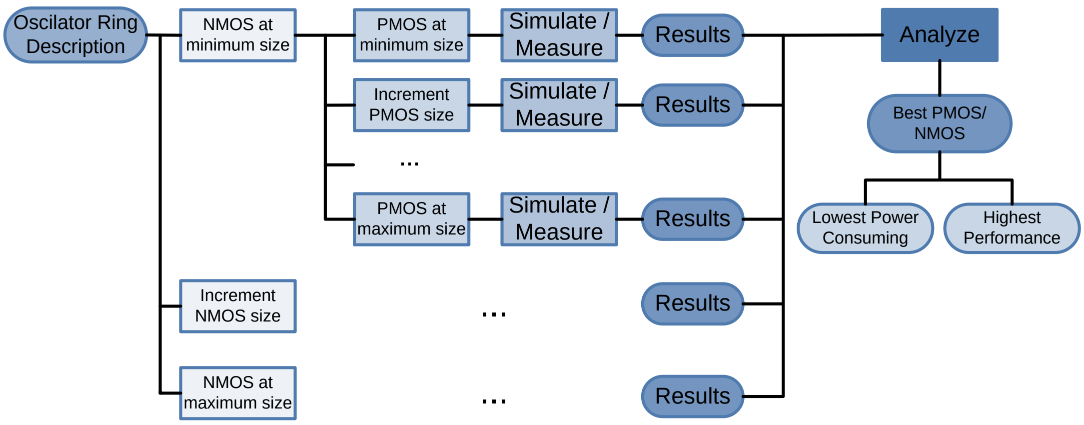


Figure 4.15 – Automated flow for simulation, analysis and specification of transistors size. A specially designed tool generates an oscillator ring from the designed schematic view, this ring is simulated for different dimensions of PMOS transistors over different dimensions of NMOS transistors and each variation has its electrical behavior measured. The results are analyzed by another specially designed tool that selects the best PMOS dimension for each variation in the NMOS size. From there, two implementations are selected, one for lower power consumption (BP) and one for high performance (BS).

The output of this flow is a set of graphics defining the electrical behavior of each variation in the dimension of the transistors. For simplicity sake, only two graphics will be presented here, for the full set of results of every cell of the library, one can refer to [ASC10]. The metric used to define the best sizing for a performance driven cell, as explained above, is the operating frequency. Figure 4.16 presents the operating frequency of the ring, measured for each combination of PMOS/NMOS sizes. As it is represented in the graphic, when the transistors are too small, poor operating frequency is obtained. As the dimensions get bigger, the operating frequency increases. However there is a point where the transistors start to get too large and their input capacitance starts to be bigger than the maximum load driven each cell and the performance starts to plummet. Therefore, the best dimension for a high performance cell is the peak of the graphic, in this case a NMOS of 0.9µm and a PMOS of 1.7µm.

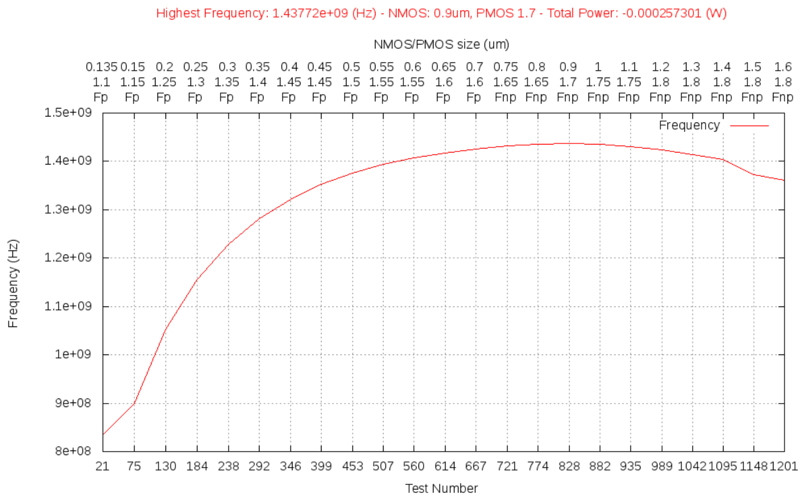


Figure 4.16 – Results obtained from operating frequency measurements for a ring of C-Elements with varying transistors size.

The metric used to define a best cell, concerning to low power consumption, is a function of the total power consumption and the operating frequency. The cell that presents the best cost-benefit, regarding Watts consumed per Hertz obtained, is the BP implementation. Figure 4.17 presents the result for this relation between power consumption and operating frequency. As the graphic shows for this example, when the size is too small, the W/Hz relation is not the best, that is due to the fact that this dimension of transistors offers poor performance, albeit consuming little power. Therefore, the result is that as the sizing is increased, a better cost-benefit is obtained. However, there is a point where the power consumed to increase the performance starts to get too large and the cost-benefit gets worst. In this case, the best cell for a BP implementation has NMOS transistors of 0.25µm and PMOS of 1.3µm.

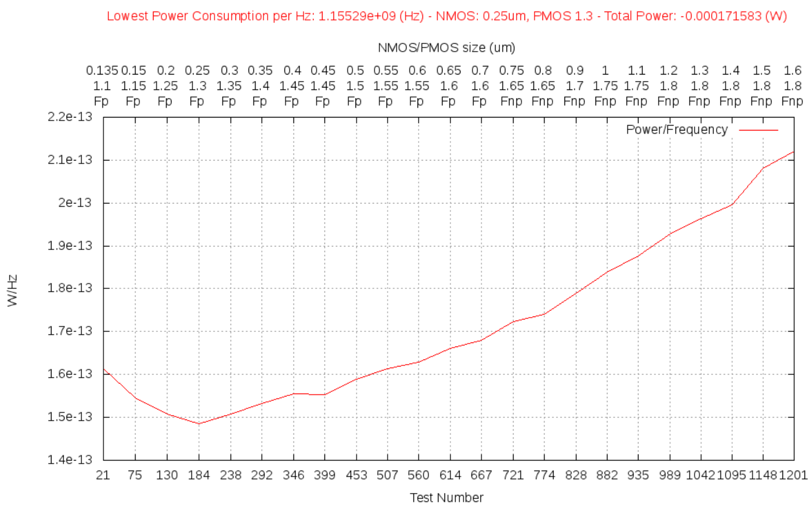


Figure 4.17 – Results obtained from operating frequency and power consumption measurements for a ring of C-Elements with varying transistors size. The graphic represent the cost-benefit of bigger transistors regarding to Watts consumed per Hertz obtained.

Therefore, two specifications of the cell are obtained, summarized in Table 4.3. The tradeoff between these implementations is clear when comparing the cost in area and power consumption to obtain a higher operating frequency.

Table 4.3 – Summarized specifications of a BP and BS implementation of an example gate.

Version	NMOS size	PMOS size	Operating Frequency	Power Consumption
BS	0.9μm	1.7μm	1.44GHz	0.257mW
BP	0.25μm	1.3μm	1.16GHz	0.172mW

However, when it is not an inverter that drives the output of a standard cell, a different method is used, albeit very similar and fully automated as well. For instance, assume that the functional specification for another cell is expressed by Equation 4.3 (a) and (b). For instance, assume that this cell must drive an equivalent load of the one driven by a X1 inverter, as determined in previous sections. Say this gate is called GPSVT_FILTER and its input ports are *RA* and *RB* and its output ports are *AA* and *AB*.

Equation 4.3 – Specification of a filter functionality for its two outputs: (a) AA and (b) AB.

$$AA = (RA \wedge \overline{RB}) \qquad \text{(a)}$$

$$AB = (RB \wedge \overline{RA}) \qquad \text{(b)}$$

In order to define the transistors size, the first step is to generate a schematic that implements the specified logic. Figure 4.18 shows the schematic designed for the specified filter. In this case the outputs are not driven by a simple inverter. Therefore the approach to define the best size for the transistors is slightly different.

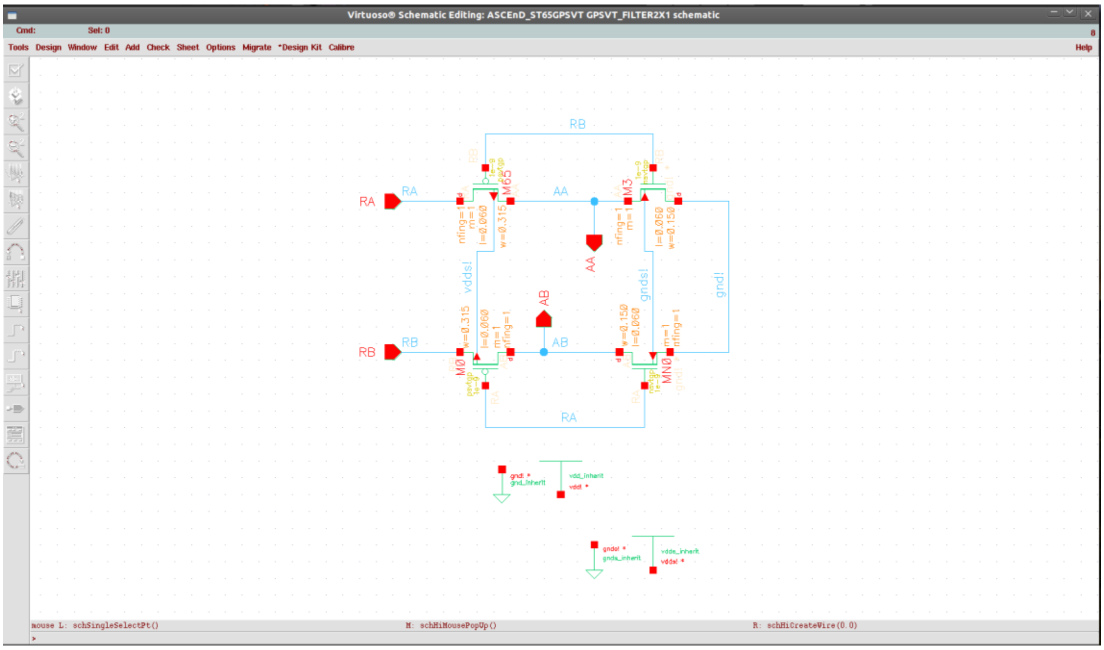


Figure 4.18 – Schematic that implements the logic of GPSVT_FILTER2X1 standard cell.

In this case, the whole circuit drives the outputs. Therefore, an approach similar to the one used to define the size of inverters for different output drives, in section 4.1.3 , must be used. The approach is fully automated and consists in fixing an output load in the output terminals of the cell and simulating the circuit for different NMOS and PMOS sizes respecting the ratio defined in section 4.1.2 . The sizes of the transistors vary from a minimum size of NMOS to a maximum of three finger NMOS. The size of the PMOS tracks the NMOS respecting the defined PMOS-to-NMOS ratio. For instance, for the GPSVT_FILTER2X1 cell, the best dimensions can be found by placing a capacitance of 0.27pF in the output terminals and measuring the time that each variation of transistors size takes to (dis)charge this load. The combination that gets closer to 1ns represents the best sizing for PMOS and NMOS transistors. Figure 4.19 presents the results of the simulations performed for different variations, concerning to transistors size, of the specified circuit. As the graphic shows, the sizing that got closer to 1ns was the minimum sizing able to drive the output load, a NMOS of 0.15µm and a PMOS of 0.315µm. Therefore, this is the sizing that will be used in the design of the physical view of this cell.

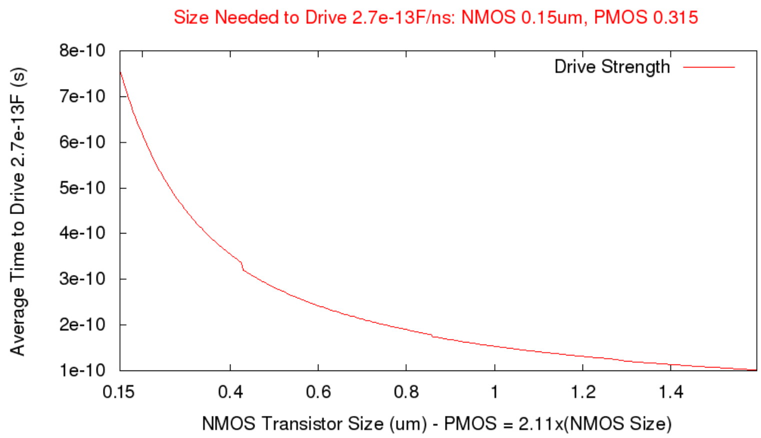


Figure 4.19 – Results for simulations of different variations, concerning to transistors size, of the GPSVT_FILTER2X1 cell with an output load of 0.27pF.

4.2.2 Layout

From the circuit schematic, the physical characteristics of the standard cell are usually designed with the use of a layout editor. The basic function of this tool is to place polygons representing the different layers used to fabricate an IC in a given technology [MIC94] [RAB03]. As it was mentioned in previous sections, the chosen tool to design physical views is Virtuoso Layout Editor. Figure 4.20 shows the layout of the 2 input C-Element specified in 0. In this example, the green blocks are the diffusion and the red ones are polysilicon. Every polysilicon that overlaps the diffusion forms a gate of a transistor and its side diffusion is the drain/source. The logic is then implemented by connecting the nodes together through metal

1, blue blocks. Contacts, used to connect different layers like polysilicon and metal 1, are represented by yellow squares. The doping layers are the NWELL, orange rectangle, P implant, pink rectangle and N implant, yellow rectangle.

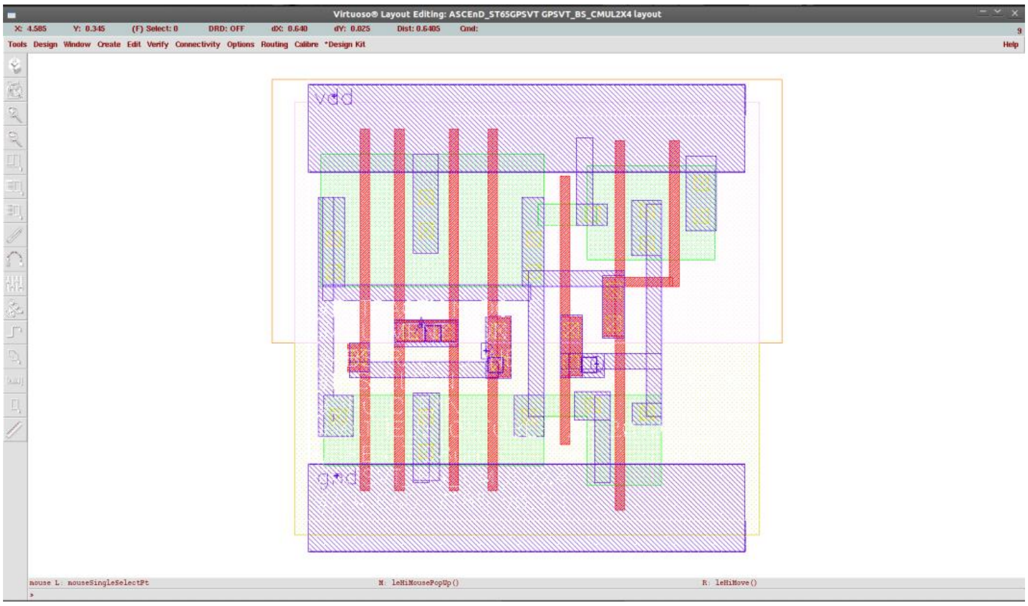


Figure 4.20 – View of the Virtuoso Layout editor. The physical design of a 2 input C-Element.

The layout must respect DRC and sometimes DFM rules. The present work defined a set of DFM rules through a layout guideline, in 0. The adopted design flow for the physical view is to handcraft each transistor of the specified circuit and verify it until it meets the specifications and the design rules, as it is shown in Figure 4.21.

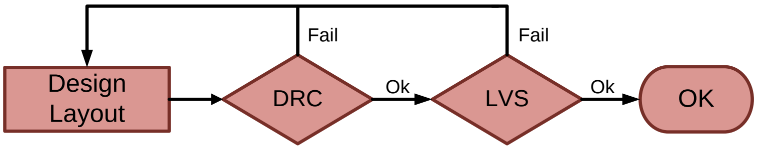


Figure 4.21 – Adopted physical view design flow.

The tool used to perform DRC and LVS verification is Calibre, developed by Mentor. Once the view succeeds in both verifications, it can be extracted. Calibre is also the tool used for extraction. The process of extracting a circuit from a layout consists in scanning the designed physical layers and reconstructing the circuit. The difference is that the extracted circuit will contain not only the projected devices, P and N type transistors, but also parasitic elements. In this way a more accurate analysis over the circuit is possible.

However, due to the fact that the designed standard cells are tapless and have no bulk polarizations, a tapped version of the cell must be generated in order to extract the circuit. Therefore, a circuit consisting of the designed gate and a tapping cell is designed. The distance between the tap and the gate is the maximum distance recommended by the design kit manuals. Figure 4.22 shows an example of a tapped cell, in this case the 2 input C-Element from Figure 4.20.

In the adopted flow, the extraction tool performs a RC extraction, *i.e.* extracts parasitic capacitors and resistors. Moreover, the circuit is extracted for 3 process corners, typical, minimum and maximum parasitic devices. From these, 3 descriptions of the circuit are obtained, based on the design kit manuals recommendations:

- Best Case: minimum parasitic devices for a temperature of -40C;
- Typical: typical parasitic devices for a temperature of 25C;
- Worst Case: maximum parasitic devices for a temperature of 125C.

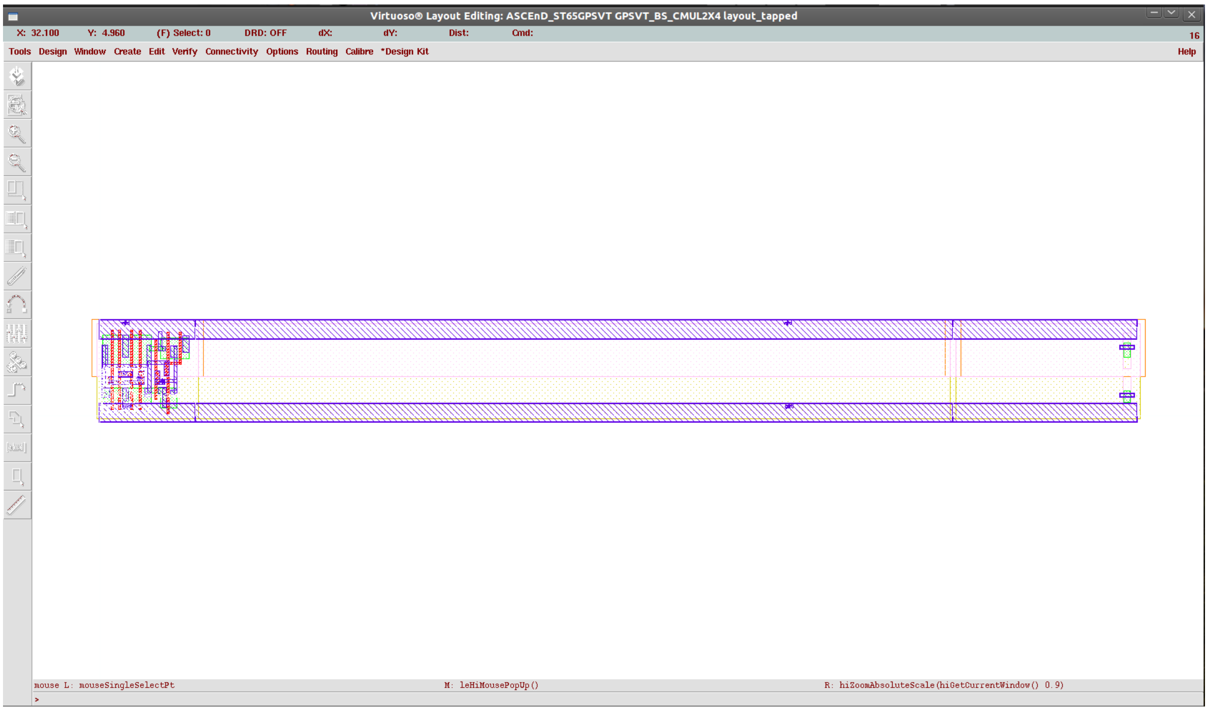


Figure 4.22 – Example of a tapped cell, required for circuit extraction.

4.2.3 Electrical Characterization

To design an IC using a standard-cell library, a detailed documentation about the library is required. That is due to the fact that the design flow of a complex standard-cell based design is virtually impossible without the aid of automated tools. Such tools must rely on the referred documentation in order to synthesize higher level descriptions of circuits, such as RTL sources. Therefore, a robust library must have each standard cell electrical behavior characterized for different process variations, output loads and input slopes.

As mentioned before, the tool used to characterize each gate of the library is Encounter Library Characterization (ELC) [CAD10b]. This tool makes use of non linear table delay models to characterize circuits. These models consist of tables defining input slopes and output loads and each combination represents a simulation scenario. Therefore a limited number of simulations are performed, constrained by the number of elements in the table, and less computation is required to characterize a standard cell. However, the obtained results are not a continuous function of input slopes and output loads and any missing value is defined by linear interpolation. In this way less computation is required to characterize a standard cell. In fact it the recommended number of values for these models is from 5 to 7 [RAB03]. Therefore the adopted flow will use non linear tables with 7 values of input slopes and 7 values of output loads. Figure 4.23 shows an example of a non linear model input table used to characterize X4 drive cells.

```
Index DRIVEX4_T {
  Bslew =  0.002N 0.008N 0.022N 0.050N 0.080N 0.180N 0.300N ;
  slew =  0.002N 0.008N 0.022N 0.050N 0.080N 0.180N 0.300N ;
  load =  0.001P 0.005P 0.020P 0.070P 0.150P 0.500P 0.800P ;
};
```

Figure 4.23 – Example of non linear model input table.

Figure 4.24 depicts the electrical characterization flow using ELC. Firstly, a logic model is generated from a transistor-level circuit described in spice format, usually exported from a schematic description. From this logic model, test vectors are automatically generated in order to simulate every possible transition as well as each possible state of the cell. Then, a characterization environment is formed, specifying the operational conditions and the process corners along with a non linear table. From there, different SPICE simulations are performed for each combination of the input table for each generated test vector. From the simulations, the electrical behavior of the circuit for each combination of input slope and output load for the specified process corner is determined.

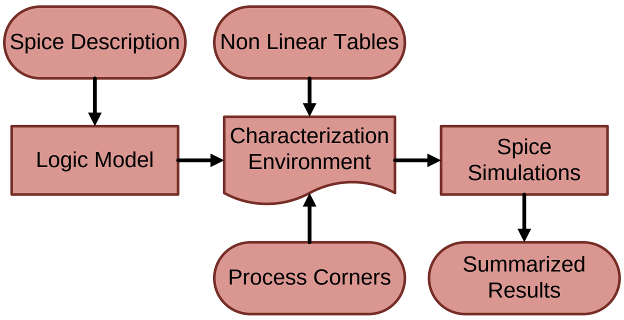


Figure 4.24 – Library characterization flow using ELC [CAD10b].

The aspects characterized by ELC are: slew rate, propagation time, static and dynamic power consumption, input pins capacitance and maximum load for each output pin. The propagation time is characterized pin-to-pin, which means that every possible path in the cell will have its propagation delay characterized. In fact, the propagation delay represents the time that it takes a change in an input to affect an output. For the adopted design flow, as defined by the design kit manuals, the switching threshold is at 40% for rise transitions and 60% for fall transitions. For instance, Figure 4.25(a) shows the propagation delays for an inverter where t_f and t_r represents the fall and rise propagation delays, respectively.

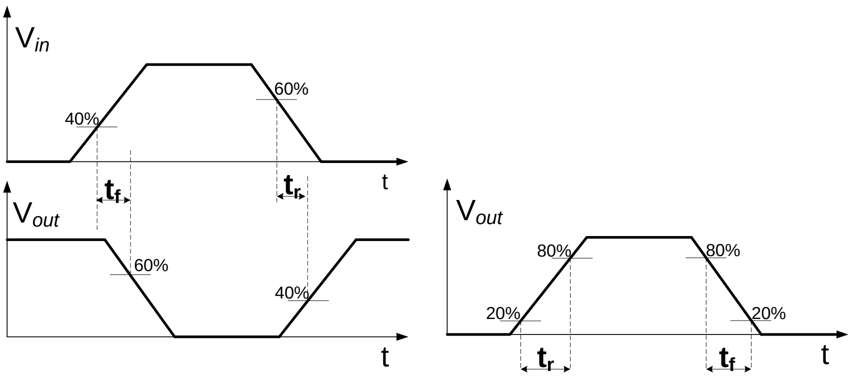


Figure 4.25 – Example of propagation delays (a) and transition delays (b) for an inverter.

The example of an inverter is rather simple and there is a direct effect from a single input to the only output of the gate. However, more complex cells, where the propagation time between an input and an output port can depend on the status of other input ports, are not that trivial to be characterized. This aspect is called state dependency. ELC automatically detects state dependency and generate test vectors to verify every possible state of the cell [CAD10b].

Along with the propagation delay, the time that an output takes to switch its logical level for a given input slew and output load must also be characterized. Such time is called transition time. To avoid the uncertainty of when a transition actually starts or ends, it is defined that the transition times occur between 20% and 80% of the output voltage. Such assumption is defined in the design kit manuals. Figure 4.25(b) shows an example of rise (t_r) and fall (t_f) transition delays.

To characterize the dynamic power consumption, the transient power and the power needed to charge the circuit, output load and parasitic capacitors, must be defined. For instance assuming that the NMOS and the PMOS of an inverter are never on simultaneously its transition circuits can be represented as in Figure 4.26. The output load is represented by the capacitor C_L and the circuits (a) and (b) represents the high to low and the low to high transition circuits, respectively. During the low to high transition of the output, the capacitor C_L is charged through the PMOS and an amount of energy is drawn from the power supply, albeit part of this energy is dissipated in the PMOS device. When the output switches once more, high to low transition, the load in C_L is discharged through the NMOS and part of this stored energy is also dissipated in the NMOS.

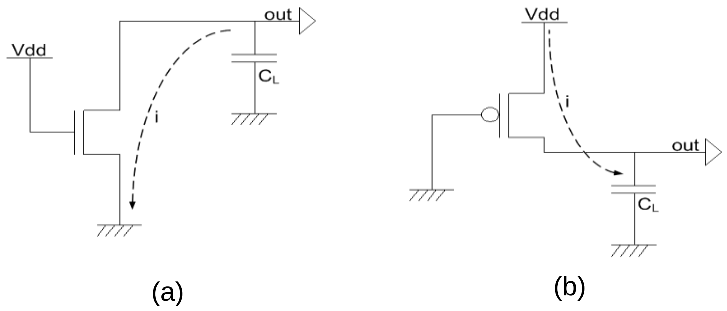


Figure 4.26 – Equivalent transition circuits for an inverter, (a) high to low transition and (b) low to high transition, from [RAB03].

However, in actual designs, the assumption that the PMOS and the NMOS are never on in the same instant is not correct. For a short period of time, there is a transient current between the power rails. That is due to the fact that there is a range of values where the logical value is undefined, recalling Figure 4.25(b), between 20% and 80%. Hence, during this period of time the current flows from the power supply to ground, reaching its peak in half of the signal excursion as it is depicted in Figure 4.27. Moreover, there are also the parasitic capacitances that are inevitable because of the proximity of the components of the circuit. The energy required to load these capacitances also represents power consumption, because each rise transition of the cell require energy to load them. Together with the power consumed during the transient period of the inputs and the power consumed to charge the output load, it represents the dynamic power consumption.

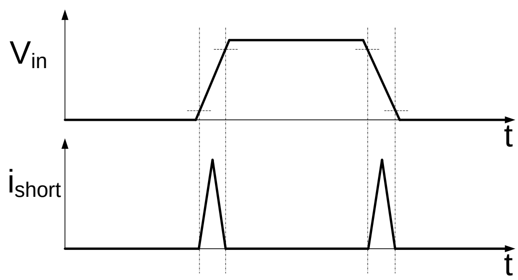


Figure 4.27 – Transient current, adapted from [RAB03].

The static power dissipation of a circuit is expressed by Equation 4.4, where I_{stat} is the current flowing between the power rails when there is no activity in the circuit. When the circuit's inputs are static and its state is well defined, the PMOS and the NMOS should not be on at the same time. However there is a leakage current flowing through the transistors between the substrate and the source, or drain. This current represents power dissipation for a static state of the circuit.

Equation 4.4 – Static Power Dissipation [RAB03]

$$P_{stat} = I_{stat} V_{DD}$$

The capacitance of the input pins and the maximum load for the output pins are essential information for the electrical characteristics of a library. The input capacitances rely on the size of the transistors. A large transistor has a large gate and, consequently, a large input capacitance. The output maximum load, on the other hand, is the maximum load that the drive transistors were capable to drive in a certain period of time. For the present work, the cells are characterized with a maximum simulation time of 0.5ns. Therefore, the maximum output load represents the maximum load that a standard cell can drive in 0.5ns.

All the described process, of characterizing a cell with ELC, is automated. Therefore, usually, one may just supply the spice circuit, the input look-up table, the process corners and the operational conditions in order to obtain the electrical characteristics of a cell for the given parameters. As it was explained before, different process corners and operational conditions must be characterized in order to assure the reliability of a library. For the present work, three characterizations are performed:

- Best Case: using a best case extracted circuit, a fast NMOS and fast PMOS (FF) process corner and a temperature of -40C;
- Typical Case: using a typical extracted circuit, a typical NMOS and a typical PMOS (TT) process corner and a temperature of 25C;
- Worst Case: using a worst case extracted circuit, a slow NMOS a slow PMOS (SS) process corner and a temperature of 125C.

All the information generated by ELC characterization process is exported to a Liberty Library (LIB) file. The LIB is a powerful high-level description that can represent a wide variety of characteristics of a circuit. It was developed and is maintained by Synopsys [SYN10].

4.2.4 Abstract

The abstract view is automatically generated through Cadence Abstract Editor. Figure 4.28 shows the abstract view of the layout depicted in Figure 4.20. The filled blue rectangles represent the metal 1 connections of I/O terminals, which can be used for the cell routing. On the other hand the empty blue polygons represent metal blockage, due to internal routing of the gate. This view is then automatically exported to a text based library exchange format (LEF) [CAD09]. This format was designed and is maintained by Cadence.

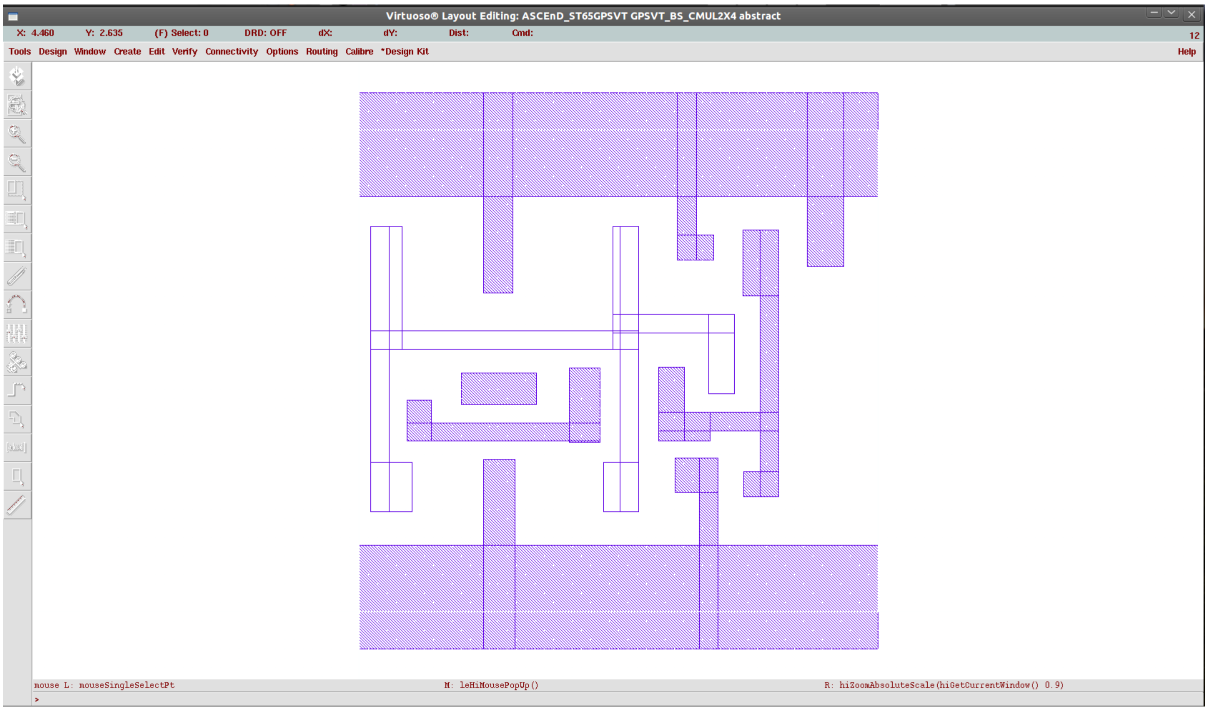


Figure 4.28 – Example of abstract view automatically generated through Cadence Abstract Generator. Abstract view of the layout showed in Figure 4.20.

4.2.5 Symbol

Next, a symbol must be generated for the standard cell. This view is automatically exported from a schematic by Virtuoso Schematic Editor and can be used to gate level circuit descriptions. Figure 4.29 shows an example of a symbol view for the schematic showed in Figure 4.10Figure 4.18.

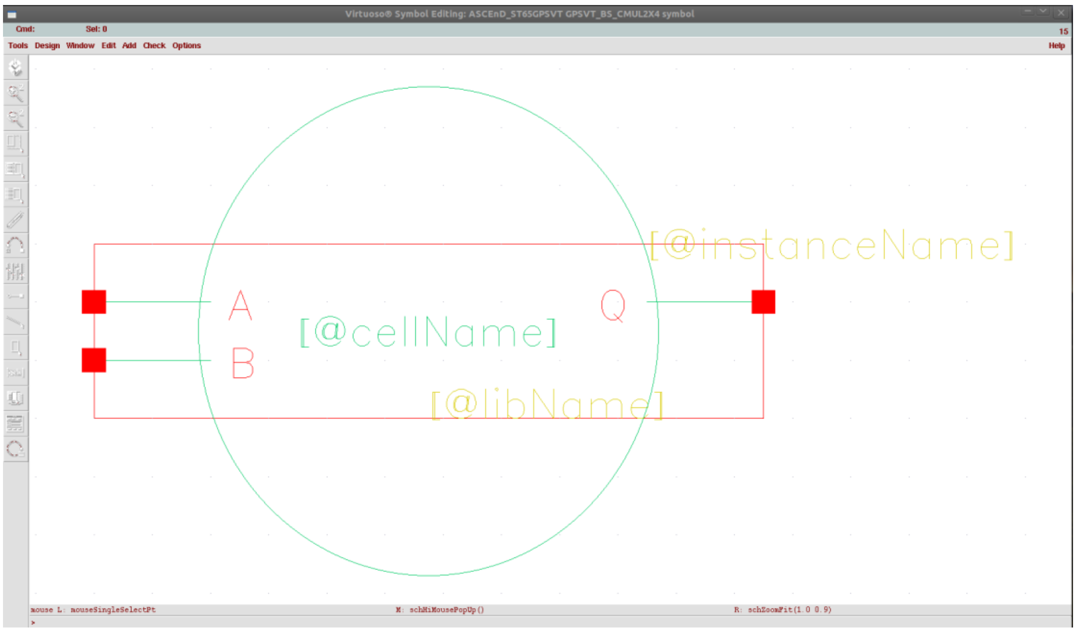


Figure 4.29 – Example of symbol generated through Virtuoso Schematic Editor.

4.2.6 Verilog

Finally, from the specifications of the gate, a behavioral view of the gate is described in Verilog. This description is simulated in a test bench and it is verified if it implements the specified logical function. If not, it must be redesigned until the specification is met. After that, a netlist containing only the described gate is placed and routed and its delay is annotated and exported to a standard delay format (SDF) file [IEE01]. The choice for this format is due to the fact that it is tool-independent and, therefore, widely accepted in EDA tools. A timing simulation is then performed and compared to the annotated delay. Figure 4.30 presents the adopted flow to generate the Verilog view.

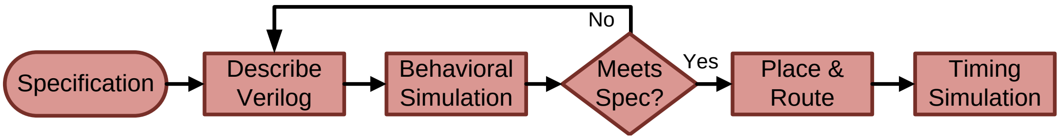


Figure 4.30 – Flow adopted for the Verilog view generation.

For instance, in Figure 4.31, lines 1-12 show the primitive generated for a 2 input C-Element. Verilog provides user defined primitives (UDP) to model complex gates, which are not built-in the language [CIL02]. The primitive is described as a truth table implementing the specified function, in this case Equation 4.2. In lines 14-25 this primitive is addressed for the BS version of the specified C-Element.

```
1  primitive u_c2 (Q, A, B);
2  output Q;
3  input A, B;
4  reg Q;
5  table
6  //A      B        : Q      :Q+1
7  0        0        : ?      : 0 ;
8  1        1        : ?      : 1 ;
9  0        1        : ?      : - ;
10 1        0        : ?      : - ;
11 endtable
12 endprimitive
13
14 `timescale 1ns/10ps
15 `celldefine
16 module GPSVT_BS_CMUL2X4 (Q, A, B);
17 input A, B;
18 output Q;
19 u_c2 i1 (Q, A, B);
20 specify
21          (A => Q) = (0.1,0.1);
22          (B => Q) = (0.1,0.1);
23 endspecify
24 endmodule
25 `endcelldefine
```

Figure 4.31 – Example of Verilog view, a 2 input C-Element.

A test bench is described in order to test every transition arch of the described behavior. This verification is automated by using a *assert* functions through a VHDL description. Figure 4.32 shows an example of a VHDL test bench for the Verilog of Figure 4.31. In order to test every possible combination of inputs transition, a clock signal is generated for each input. A base clock signal has a period of 80ns and every extra clock is 4 times faster. For instance, for a 2 input circuit, the test bench has 2 clock signals, *clock0* with a period of 80ns and *clock1* 20ns. A control signal switches the input signals every time the slowest clock completes one period. Moreover, each clock has a different transition delay. In this way, every possible transition combination is verified. The verification of the two transition arches for the described gate is represented in lines 35 and 36.

```
1  library ieee;
2  use ieee.STD_LOGIC_UNSIGNED.all;
3  use ieee.std_logic_1164.all;
4  use work.all;
5  entity tb_c2 is
6  end tb_c2;
7  architecture TB_ARCHITECTURE of tb_c2 is
8      signal clock0, clock1: std_logic := '0';
9      signal in0, in1, ctrl : std_logic := '0';
10     signal out0 : std_logic;
11     component GPSVT_BS_CMUL2X4 is
12     port(
13             A : in std_logic;
14             B : in std_logic;
15             Q : out std_logic
16     );
17     end component;
18 begin
19     in0 <= clock0 after 200 ps when ctrl = '0' else clock1 after 500 ps;
20     in1 <= clock1 after 500 ps when ctrl = '0' else clock0 after 200 ps;
21     clock0 <= not clock0 after 40 ns;
22     clock1 <= not clock1 after 10 ns;
23     bp_c_mul2 : GPSVT_BS_CMUL2X4
24       port map (
25         Q => out0,
26         A => in0,
27         B => in1
28       );
29     process(clock0)
30     begin
31         if clock0'event and clock0='1' then
32             ctrl <= not ctrl;
33         end if;
34     end process;
35     assert not (in0 = '1' and  in1 = '1' and out0 = '0') report "rise Q bp_c_mul2 not working!" severity WARNING;
36     assert not (in0 = '0' and  in1 = '0' and out0 = '1') report "fall Q bp_c_mul2 not working!" severity WARNING;
37 end TB_ARCHITECTURE;
```

Figure 4.32 – Example of a test bench, described in VHDL for automatically verifying the behavioral description from Figure 4.31.

The simulator used to verify the Verilog view is Cadence Incisive. However the simulations were also performed in Mentor Modelsim. Figure 4.33 shows the waveforms obtained by simulating the test bench described in Figure 4.32.

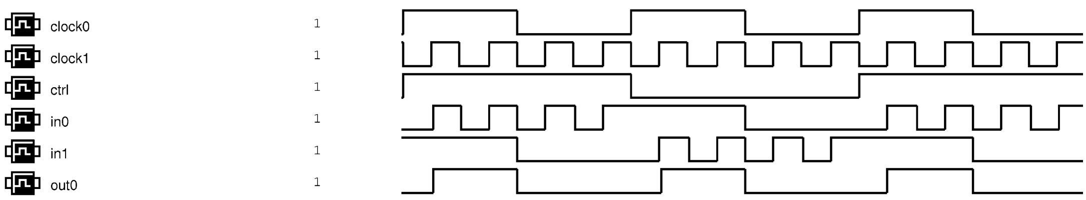


Figure 4.33 – Resultant waveform from the simulation of the VHDL test bench described in Figure 4.32.

Finally, in order to verify timing annotation, a netlist consisting of a single cell is generated and then placed and routed through Cadence Encounter, showed in Figure 4.34.

The delay of the design is annotated through Encounter from the characterized electrical behavior of the standard cell. Next, the information is exported to a SDF file. This process is fully automated by Encounter.

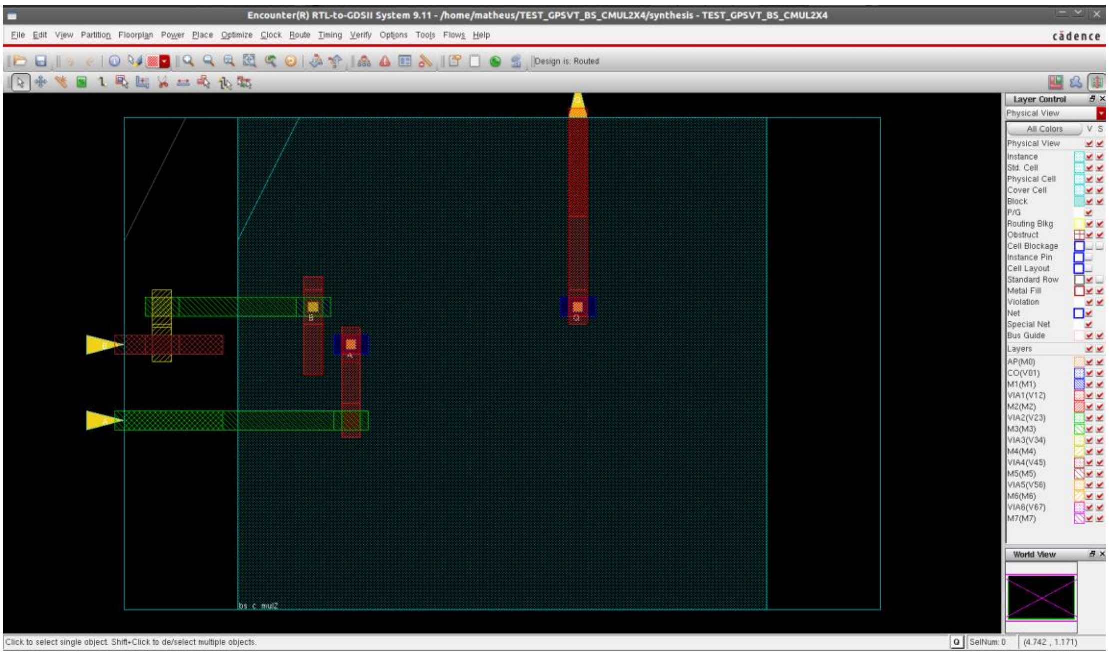


Figure 4.34 – Example of a single cell design placed and routed in Encounter. The filled blue square is the physical site of the standard cell, in the borders of the outer square are the I/O pins and the routing is through metal layers, red, green and yellow rectangles.

An example of a SDF file is given in Figure 4.35. The delays are organized according to the structure presented in lines 6 and 7. First it is defined the path, *A* to *Q* in line 6 and *B* to *Q* in line 7. The rise and fall propagation delays are defined in the first and second parenthesis, respectively. Each delay definition has three values, for best, typical and worst cases, separated by a colon. The time scale for this example is defined to 1ns.

```
 1 (CELL
 2   (CELLTYPE "GPSVT_BS_CMUL2X4")
 3   (INSTANCE bs_c_mul2)
 4   (DELAY
 5     (ABSOLUTE
 6       (IOPATH A Q (0.0120:0.0140:0.0160) (0.0210:0.0220:0.0230))
 7       (IOPATH B Q (0.0120:0.0140:0.0160) (0.0200:0.0210:0.0220))
 8     )
 9   )
10 )
11 )
```

Figure 4.35 – Example of SDF file, generated from Encounter.

Next, a timing simulation at gate level is performed through Incisive. The results of this simulation are compared to the ones obtained in the SDF file. Figure 4.36 presents the resultant waveforms for the timing simulation of the example design for a typical process corner. Recalling Figure 4.32, the signals *in0*, *in1* and *out0* are pins *A*, *B* and *Q*, respectively.

The rise propagation of input *A* to input *Q* is represented in Figure 4.36 (a), the propagation delay is the difference between the two signal rises, in this case 14ps. The same value can be verified in Figure 4.35 line 18 for a typical process rise propagation delay. Similar verifications can be done for each transition arch. If they are all met, the timing annotation is correctly working.

Figure 4.36 – Timing simulation results for a gate level design of a single 2 input C-Element standard cell. (a) and (b) presents, respectively, rise and fall propagation delays from pin *A* to *Q, while* (c) and (d) presents the propagation delays from pin *B* to *Q*.

4.2.7 Validation

From the information generated in the characterization process, it can be verified if the extracted cell electrical behavior is equivalent to the one obtained in the specification steps. If the cell does not meet the requirements, it must be redesigned. If it does, the cell follows to the library.

4.3 Library Validation under a Standard Cell Flow

After a set of cells are specified, designed and validated, a library is obtained. From this library, different ICs can be designed. Two different design flows were adopted to generate test ICs from the designed standard cells. Firstly, a typical design flow consisting of a high level description of the circuit at register transfer level (RTL), in this case VHDL, and the Cadence Framework, showed in Figure 4.37(a), was used to design the router of an asynchronous network on chip (NoC). Another flow, consisting of a Balsa source, synthesized by Teak and placed and routed by Encounter, Figure 4.37(b), was also used to implement an asynchronous RSA based cryptography circuit on the designed standard cells.

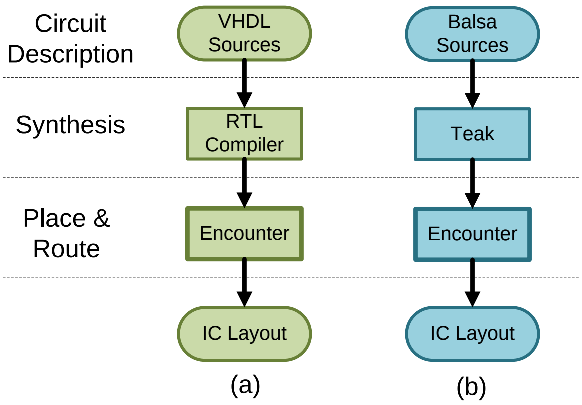


Figure 4.37 – IC design flows used to implement designs on the designed library gates.

5 ASCEnD ST65 v0.1

“Things must be done decently and in order.”

Sir Arthur Conan Doyle

The method described in Chapter 4 was adopted to create a set of gates for the ST Microelectronics 65nm bulk CMOS process (ST CMOS65) [STM10]. A total of 251 gates were successfully designed and characterized. This cell set counts with numerous implementations of C-Elements and different versions of metastability filters. The standard cell library is called Asynchronous Standard Cells Enabling *n* Designs (ASCEnD), currently at version 0.1. Together with a standard cell library with typical cells (like ANDs, ORs and buffers), provided in the design kit, ASCEnD ST65 can be used to implement asynchronous ASICs in several current design styles, including QDI and others.

5.1 The ST CMOS65 Technology

The ST CMOS65 technology provides low power and general purpose devices for a 65nm CMOS gate length fabrication process. Moreover, it gives access to standard threshold voltage (Vt) transistors (SVT), and high Vt transistors (HVT). It allows a back-end with 6 or 7 metal levels. This work makes use only of general purpose standard Vt (GPSVT) transistors. However future work includes implementing the library with low power standard Vt (LPSVT) devices as well.

5.2 Name Convention

The adopted convention to name each cell is organized as: *“DEVICE-TYPE_(BP/BS)_CELL-NAME_DRIVE”*. So far, the ASCEnD ST65 library counts with GPSVT devices only, therefore every gate name starts with GPSVT.

5.3 Designed Gates

The C-Element is a fundamental gate in most asynchronous designs [SPA01] [MYE01]. Therefore, it is essential to have available different implementations of a C-Element in a standard cell library intended to support asynchronous circuits design. Moreover, due to its continuous notion of time, the project of asynchronous circuits must deal with events like metastability. Therefore, the implementation of efficient metastability filters is indispensable.

This work presents the design and implementation of a standard cell library containing 251 gates, from which 247 are C-Elements and 4 are metastability filters. The flow implemented to design each gate is presented in Section 4.2 and the logical specifications are in Appendix B. For each specification, different cell variations were implemented, concerning to output drive strength and, in the case of C-Elements, architectural variations. Examples of layout and schematic views appear in Appendix C. Finally, one can consult the homepage of the library [ASC10] for a full documentation suite on the library.

5.3.1 C-Elements

The circuits designed in [PON10], [PON10b] and in this work showed that out of all asynchronous standard cells in an asynchronous ASIC design, 90% are C-Elements a gate originally proposed by Muller in [MUL57]. Therefore, a large set of C-Element variations provides adequate resources to trade off power consumption, area and performance in asynchronous circuits. This work proposes a total of 21 variations of the basic C-Element, regarding number and symmetry of inputs, . Three different CMOS implementations of each variation were designed: Conventional, Symmetric and Weak Feedback (Muller) [SHA96] [SHA97] [SHA98]. Moreover, each implementation of the C-Elements counts with a low power driven (BP) and a high performance (BS) version, as Figure 5.1 shows. Initially, each gate was designed for an X1 output drive. However, more output drives are provided for most of these gates (typically 4 choices, from X1 to X4).

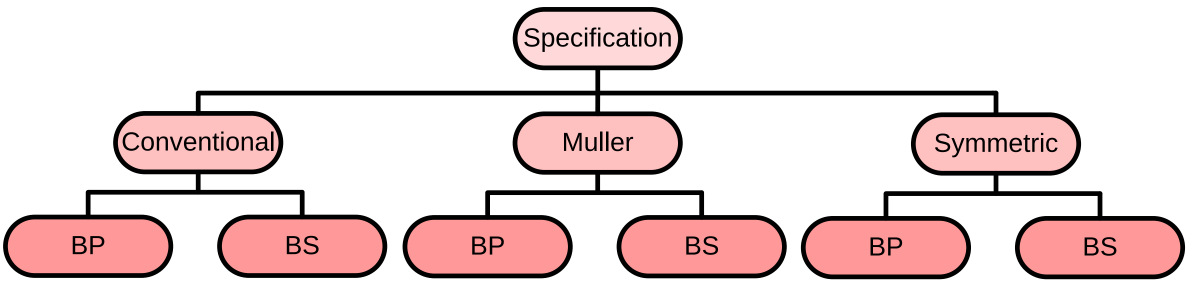


Figure 5.1 – Versions of each C-Element specification.

The designed two and three-input C-Elements are summarized in Table 5.1 and Table 5.2, respectively.

Table 5.1 – Designed 2 input versions of C-Elements.

Cell	Drives	Description
C1D2	X1	Two-input C-Element with one normal input and one falling[1] input.
C1D_RST2	X1	Two-input C-Element with one normal input, one falling input and an active-low reset input.
C1D_SET2	X1	Two-input C-Element with one normal input, one falling input and an active-high set input.
C1U2	X1, X2, X3, X4	Two-input C-Element with one normal input and one rising[2] input.
C1U_RST2	X1	Two-input C-Element with one normal input, one rising input and an active-low reset input.
C1U_SET2	X1	Two-input C-Element with one normal input, one rising input and an active-high reset input.
C2	X1, X2, X3, X4	Two-input C-Element with two normal inputs.
C_RST2	X1, X2, X3, X4	Two-input C-Element with two normal inputs and an active-low reset input.
C_SET2	X1, X2, X3, X4	Two-input C-Element with two normal inputs and an active-high set input.

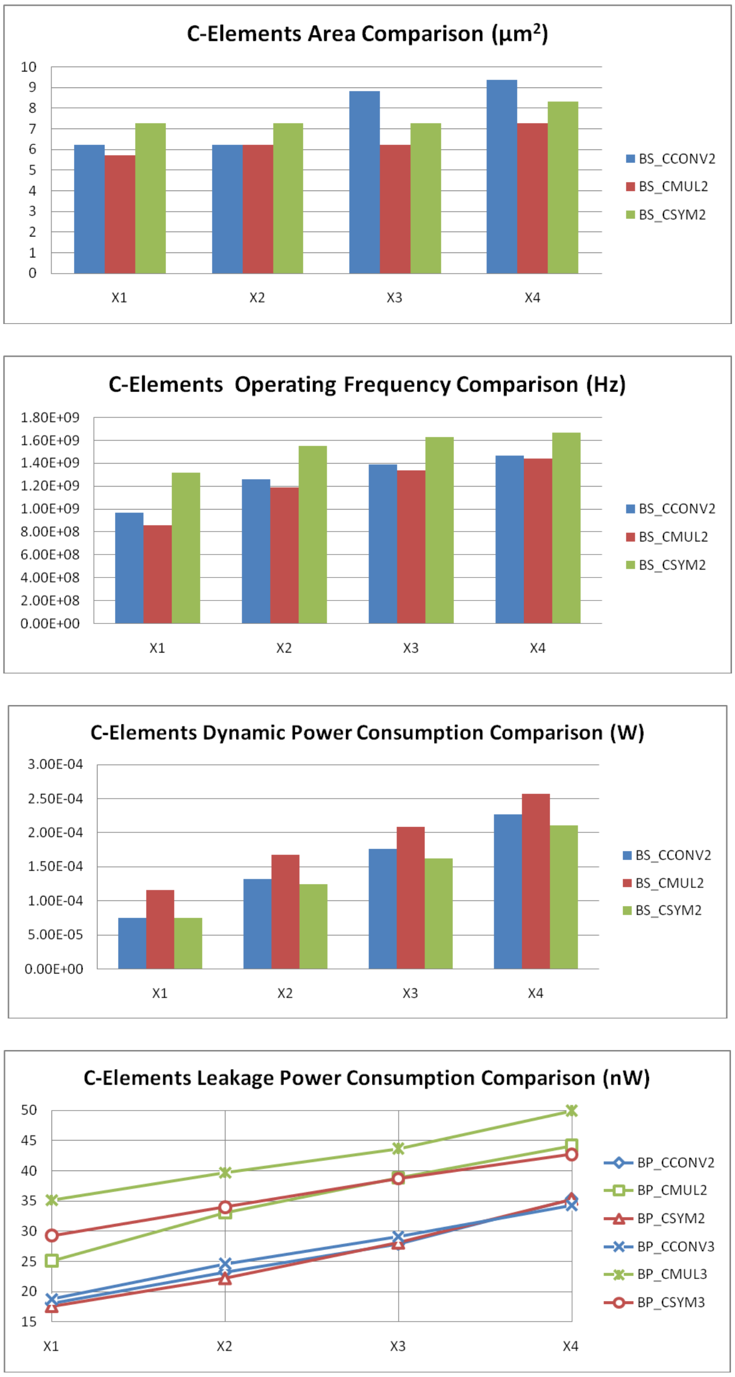
[1] A falling input is an input that has influence only in the high to low transition of the output.

[2] A rising input is an input that has influence only in the low to high transition of the output.

Table 5.2 – Designed 3 input versions of C-Elements.

Cell	Drives	Description
C1D1U3	X1	Three-input C-Element with one normal input, one rising input and one falling input.
C1D1U_RST3	X1	Three-input C-Element with one normal input, one rising input, one falling input and an active-low reset input.
C1D1U_SET3	X1, X2, X3, X4	Three-input C-Element with one normal input, one rising input, one falling input and an active-high set input..
C1D3	X1	Three-input C-Element with two normal inputs and one falling input.
C1D_RST3	X1	Three-input C-Element with two normal inputs, one falling input and an active-low reset input.
C1D_SET3	X1	Three-input C-Element with two normal inputs, one falling input and an active-high set input.
C1U3	X1	Three-input C-Element with two normal inputs and one rising input.
C1U_RST3	X1, X2, X3, X4	Three-input C-Element with two normal inputs, one rising input and an active-low reset input.
C1U_SET3	X1	Three-input C-Element with two normal inputs, one rising input and an active-high set input.
C3	X1, X2, X3, X4	Three-input C-Element with three normal inputs.
C_RST3	X1	Three-input C-Element with three normal inputs and an active-low reset input.
C_SET3	X1	Three-input C-Element with three normal inputs and an active-high set input.

A comparison of the three CMOS implementations classes (conventional, Muller and symmetrical) for a 2-input C-Element appears in Figure 5.2. The total area of each cell is compared in (a) for different output drives, showing that the Muller version presents the smallest area. The operating frequency of a ring of 10 C-Elements and a NAND gate is compared in (b) and its dynamic power consumption in (c). Results show that the symmetric C-Element presents higher operating frequencies and lower power consumption. The conventional version presents low leakage power consumption for different output drives as Figure 5.2 (d) shows. For a 2-input cell, the symmetric and the conventional static (leakage) power consumption is equivalent. However for 3-input cells, the latter dissipates in average 30% less power when the circuit is quiescent (static). The schematic and layout views of these gates are showed in Appendix C.



(a)

(b)

(c)

(d)

Figure 5.2 – Several comparisons between different C-Elements CMOS implementations with varying drive strengths, from X1 to X4. The compared C-element classes are conventional (CCONV), weak feedback or Muller (CMUL) and symmetric (CSYM). In (a) the area of the cells is compared. In (b) and (c), the operating frequency and the dynamic power consumption of a ring of C-Elements are compared. In (d), the leakage power consumption of the gates is compared for 2 and 3-input cells. For instance, BP_CMUL2 is a 2-input BP Muller (weak feedback) C-Element, while BP_CMUL3 is its 3-input version.

5.3.2 Metastability Filter

Metastability Filters are crucial elements to implement asynchronous circuits. This is due to the fact that they guarantee a robust implementation of control elements, like arbiters. These elements are essential to compute data and take decisions for the correct operation of the circuit [BER99b] [MYE01] [SPA01].

The designed library counts with 4 implementations of a metastability filter. Each of which has a different output driving strength, according to Section 4.1.3 . The specification of a metastability filter behavior is described in Appendix B and an example of schematic and layout views for it are available in Appendix C.

6 ASYNCHRONOUS CIRCUITS DESIGN WITH ASCEnD ST65

"We can but try."

Sir Arthur Conan Doyle

An asynchronous RSA based cryptography circuit was described in Balsa language and synthesized with Teak using a typical standard cell library, provided by the foundry, and the ASCEnD ST65 library. The generated netlist was placed and routed through Encounter and the obtained netlist was validated through timing simulation. Another IC flow was also employed to implement an asynchronous NoC router ASIC. This was done by describing the circuit in VHDL and synthesizing it with Cadence RTL Compiler. The generated netlist was placed and routed through Encounter and the circuit was validated through timing simulation.

6.1 Asynchronous RSA Based Cryptography Circuit

The design flow adopted to describe and implement the asynchronous RSA based cryptography circuit is showed in Figure 6.1. This flow counts with a novel language for describing asynchronous circuit, a novel tool to synthesize such circuits and the ASCEnD ST65 library.

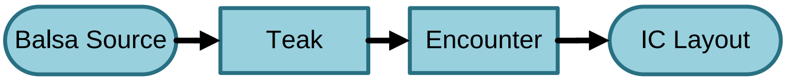


Figure 6.1 – IC design flow adopted to describe and implement an asynchronous RSA based cryptography circuit.

RSA [SCH96] is a public key cryptography algorithm. It is widely used in current electronic protocols and can be easily implemented in hardware. The designed circuit is able to encrypt and decrypt information using a public and a private key, denoted by Equation 6.1 (a) and (b) respectively. Equation 6.1 (c) and (d) shows the encryption and decryption operations, where *m* is the original message and *c* is the encrypted message.

Equation 6.1 – RSA algorithm equations and key definitions [SCH96].

$$pub\_key = (n, e) \quad \text{(a)}$$

$$priv\_key = (n, d) \quad \text{(b)}$$

$$c = m^e \bmod(n) \quad \text{(c)}$$

$$m = c^d \bmod(n) \quad \text{(d)}$$

The circuit was described in Balsa language. The sources that implement the algorithm can be found in Appendix D. For simplicity sake, the syntax of these descriptions will not be explored in this document. A user guide for the Balsa language and system can be found in [APT10].

The inputs of the circuit are: *control*, *data_in*, *data_out_ack* and *reset*. The outputs are: *data_out*, *control_ack* and *data_in_ack*. Each input, but for the *reset* input are dual rail and have a true and a false channel. Moreover, the size of the data inputs is parameterizable. The *control* signal defines the operation of the circuit. The possible operations are: load message (0x0), load e (0x1), load n (0x2), load d (0x3), encrypt (0x4) and decrypt (0x5). By issuing the identifier of a function in the *control* channel, the operation is performed and the circuit issues an acknowledge signal in the *control_ack* output. For the load operations, the data must be issued in the *data_in* channel. After reading the data, the circuit issues an acknowledge signal in the *data_in_ack* output. For the encrypt/decrypt operations, the circuit performs the algorithm with the values in its registers and put the result in the *data_out* output. An acknowledgement must be signaled in the *data_out* input and another operation may take place.

In order to verify the functionality of the circuit, a set of public and private keys were generated according to [SCH96]. The obtained public key was (3233_D, 17_D) and the private key (2753_D, 17_D). A test bench for a 32 bit asynchronous RSA circuit was described, also in Balsa language, and simulated using the Teak System. The test bench consisted in a looping procedure that, firstly, encrypted the message "123_D" and then decrypted the obtained encrypted message. Figure 6.2 shows the obtained results. By encrypting "123_D" with the public key (3233_D, 17_D), the message "855_D" was obtained. This message was decrypted using the private key (2753_D, 17_D) and the original message, "123_D", was recovered.

```
matheus@GAPHL07:~/work/balsa_test/rsa/teak/teak_mul$ teak --simulate tb_balsa.balsa
*** Simulating BEGIN: tb_balsa
+++ Simulating tb_balsa:
Channel capacity: 3
--- encryption of 123 with (3233, 17): 855
--- decryption of 855 with (3233, 2753): 123
--- encryption of 123 with (3233, 17): 855
```

Figure 6.2 – Simulation of a test bench for the Balsa description, using Teak System.

Next, this circuit was synthesized with Teak System into handshaking components and a Teak netlist of the circuit was obtained. So far, the data encoding and the handshaking protocol, were not defined. However, after the Teak synthesis, the generated circuit implements a four phase dual rail codification. This circuit is optimized by Teak by replacing data-less activation channels with separate control channels. The diagrams of the obtained circuits, before and after Teak optimization, are showed in Appendix D.

The Teak netlist is automatically exported to a Verilog netlist composed of a set of components of a target library. This netlist was simulated in Incisive in a test bench described in VHDL that implemented the same function of the Balsa test bench described before. The message "123_D" is encrypted with the key (3233_D, 17_D) and its result decrypted with the key (2753_D, 17_D). The results of this simulation are presented in Figure 6.3. Each ellipsis denotes the beginning of a new encryption/decryption. In the black square it is shown the request for a encryption operation. Initially, a message is loaded by signaling "0x0"[1] in the *control* channel and loading the value "123_D" in the *data_in* channel. Next, the two values of the public key are loaded by signaling "0x1" and "0x2" in the *control* channel and loading the values "17_D" and "3233_D" in the *data_in* channel, respectively. After that, an encryption request is issued by signaling "0x4" in the *control* channel. After computation, as it is showed in the red rectangle, the circuit responds with the encrypted message "855_D" in the *data_out* channel and a new operation may start. The time taken to cryptograph the message was 9.702μs.

---

[1] For simplicity sake, the handshaking protocol and data codification have been abstracted in the example. Each channel consists of a true and a false line. The original value is represented in the true line and its complement in the false line.

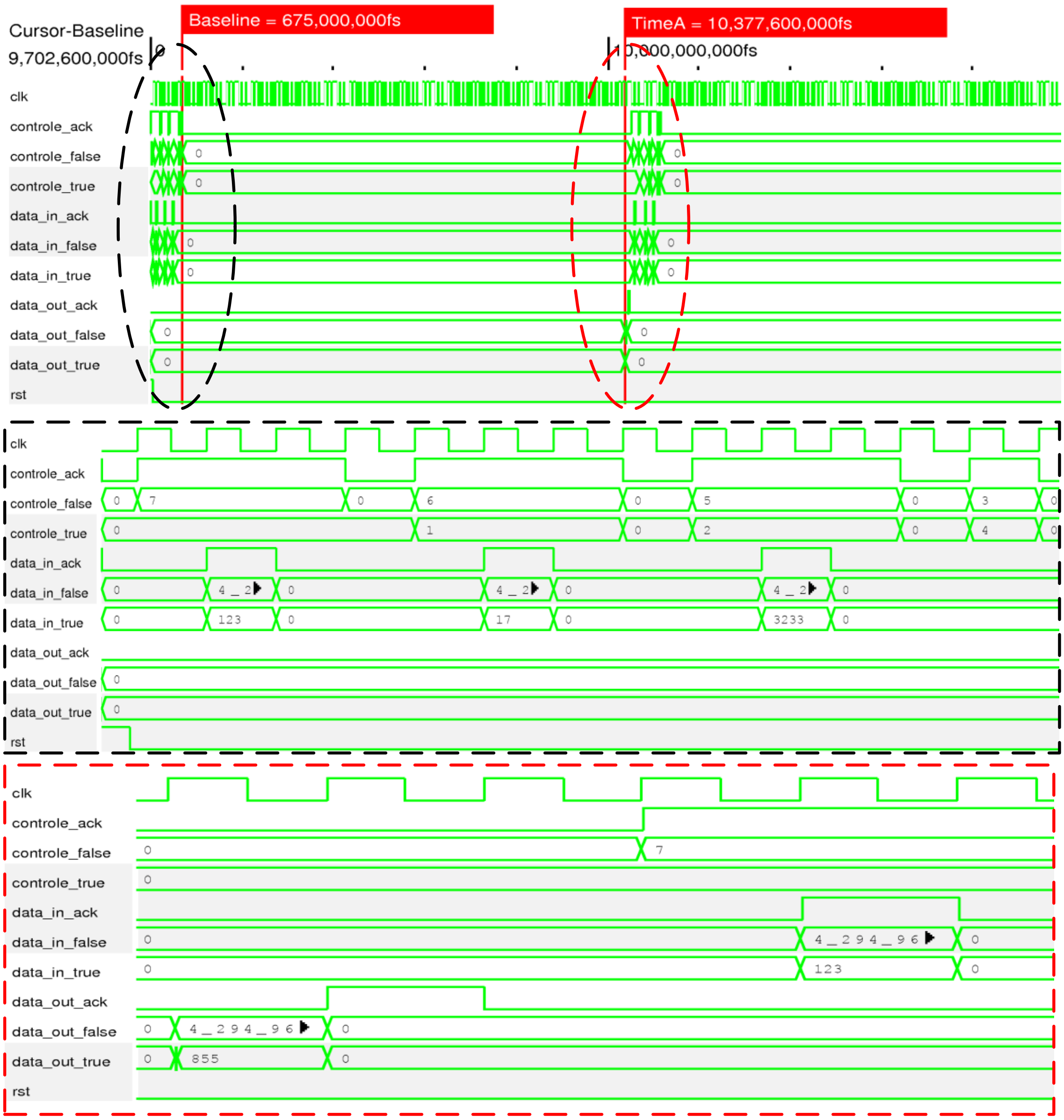


Figure 6.3 – Behavioral simulation of a test bench of the netlist generated from the Teak synthesis, using Incisive.

After the behavioral simulation, the circuit netlist was placed and routed through Encounter by using the LEF files provided by STM and ASCEnD ST65. Figure 6.4 shows the obtained layout. A total of 744095 standard cells were used in the design, which presented a total area of 2.314mm^2. Seven metal layers were employed to route the circuit with a total wire length of 9,673,661.965µm. After the place and route, the delay of each path of the circuit was annotated, by using the LIB files provided by STM and ASCEnD ST65, and exported to a SDF file. Without taking into account filler cells, 38% of the standard cells used in this design were C-Elements, proving the importance of the C-Element in an asynchronous circuit.

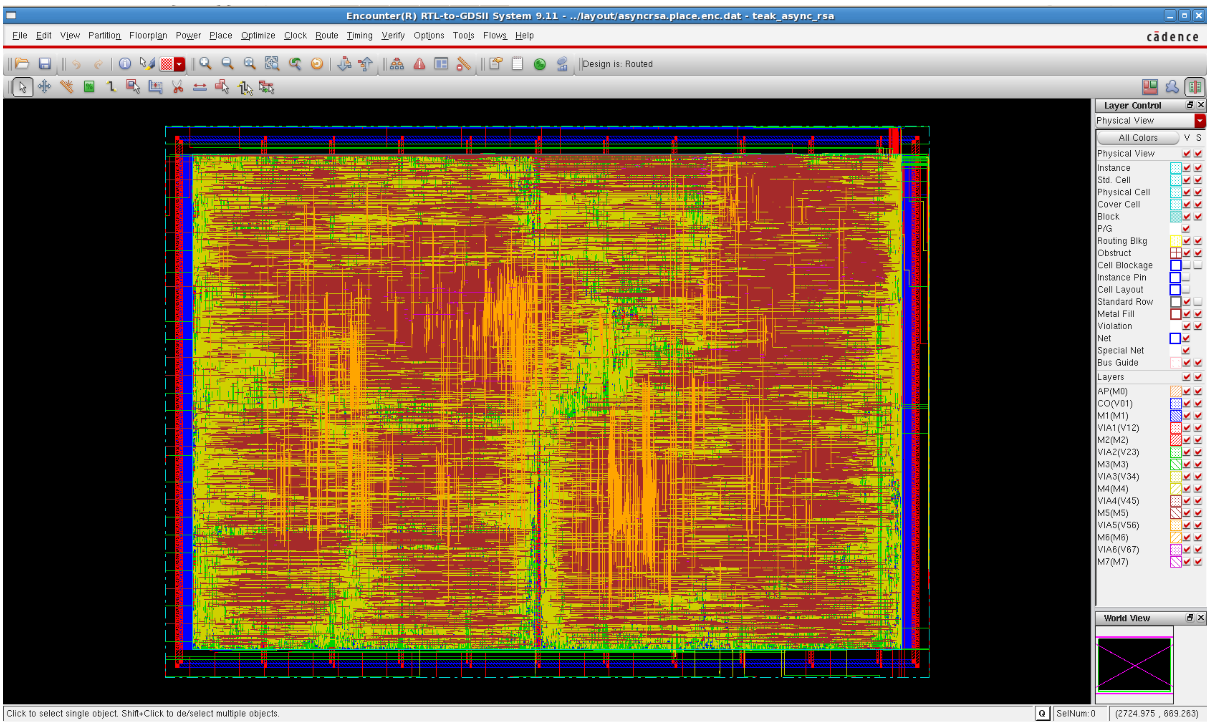


Figure 6.4 – Asynchronous RSA design routed through Encounter.

The generated circuit was exported to a Verilog netlist. A timing simulation was performed through Incisive with the same test bench used in the behavioral simulation and the delays annotated in the SDF file. Figure 6.5 shows the results of this run. This time, the circuit took 12.280467µs to encrypt a message. When compared with the unitary delay simulation (behavioral) the timing simulation presented 2.578467µs extra delay. This is due to the fact that this simulation is much more precise and considers the information generated in the electrical characterization of the cells.

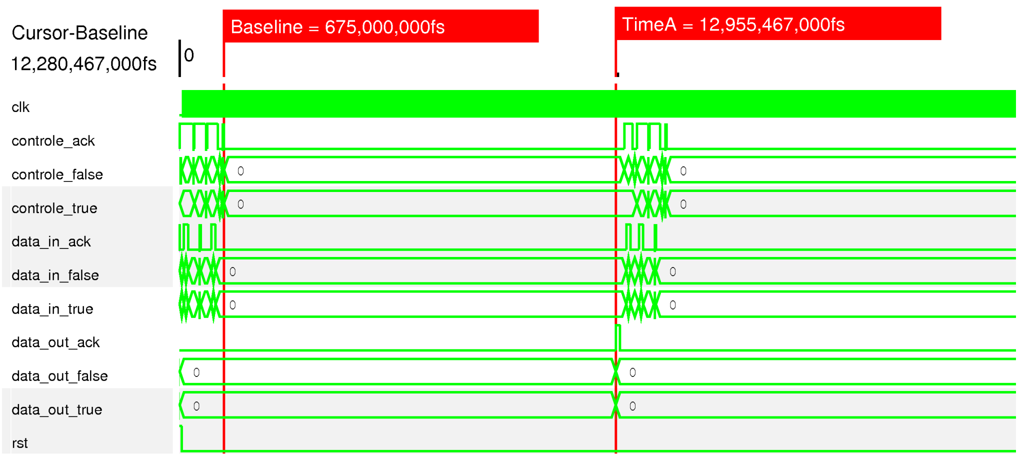


Figure 6.5 – Timing simulation of a test bench of the netlist generated from the circuit placed and routed by Encounter.

6.2 Asynchronous NoC Router

In previous works, an asynchronous NoC router was described and implemented in two different CMOS technology processes. The Hermes-A [PON10b] and the Hermes-AA [PON10] routers were implemented in an 180nm and a 65nm gate length CMOS processes, respectively. In the latter, an abridge version of ASCEnD ST65 was designed, this version counted with only a few C-Elements, required to implement the router and a metastability filter. With the development of the new library, ASCEnD ST65, Hermes-AA was redesigned through the same flow proposed in [PON10], showed in Figure 6.6.

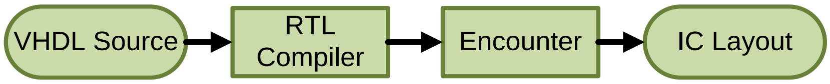


Figure 6.6 – IC design flow adopted to describe and implement an asynchronous NoC router.

Hermes-AA is described in VHDL. A netlist was generated through RTL compiler, implementing the circuit into the cells of ASCEnD ST65 and placed and routed through Encounter. The obtained circuit was simulated in order to verify its functionality. This work will not present neither Hermes-AA architecture nor its design flow in detail due to the constrained period of time available for its development. However, more details can be found in [PON10] and [PON10b].

7 CONCLUSIONS

The work described in this document has produced a standard cell library for designing asynchronous ASICs for a STM 65nm gate length CMOS process. This library was called ASCEnD ST65 (currently at version 0.1) and counts with 251 standard cells, where 247 are different implementations of C-Elements and 4 are metastability filters. Each cell has the following views: schematic, symbol, layout, abstract and Verilog. Moreover, physical characteristics are given in a LEF file, which is used in the automation of the design of standard cell based ICs and is widely accepted among CAD tools. Each standard cell has its electrical characterization for three different process variations: worst, typical and best cases. This information is given in a LIB file, which is used by synthesis tools to generate the netlist of circuits.

An early version of ASCEnD ST65, which contains 8 standard cells, was previously designed and used to implement an asynchronous NoC router, described in [PON10]. However, the library was totally redesigned here by adopting a new flow to design each cell. Currently, in ASCEnD ST65, a large set of C-Elements is available for three different CMOS implementations (Conventional, Symmetric and Weak Feedback), 21 different functional specifications and different output drive strengths. Simulation results showed that the symmetric C-Element proved to consume less dynamic power and presented better performance. The conventional C-Element on the other hand, proved to consume less leakage power, a major problem in current ICs. The weak feedback C-Element presented poor performance but requires less area. In this way, a tradeoff between performance, power consumption and density can be achieved for circuits with different requirements.

Another major contribution of this work is a novel flow to specify the transistors size of C-Elements. This is fully automated by two specially designed tools, ROGen and CeS, both implemented in C++. The result of the flow is the specification of two versions of each C-Element cell, a high performance and a low power driven CMOS implementations. These can be used to provide fine grain tuning of IC designs. Another tool, which was not mentioned in this document, was developed to aid in the design of layouts. The tool is called Sticky and was implemented in C++ language. Sticky is responsible for automatically drawing stick diagrams from Spice descriptions. It was used to ease the design of the layout for ASCEnD ST65 cells. However, some functionalities are not yet automated and were manually implemented. Future work includes upgrading this tool.

To validate the views generated for ASCEND ST65 standard cells, two ASIC design flows were used. Initially, a conventional flow, consisting of VHDL circuit descriptions synthesized, placed and routed with the Cadence Framework, was used to design an asynchronous NoC router. Next, a novel IC design flow consisting of Balsa sources synthesized through the Teak System and placed and routed by Cadence Encounter, was used to design an asynchronous RSA based cryptography circuit. The validation consisted of behavioral and timing simulations. Future work includes prototyping test chips for these and other circuits.

Finally, the objective of the proposed work was achieved through the design and validation of ASCEnD ST65. The integration of the library with the Teak System was achieved by the generation of a layout for an asynchronous RSA-based cryptographic circuit and the simulation of its netlist.

7.1 Future Work

The work described in this document presents numerous topics for further research. Firstly, work will be carried out in order to allow the library to support the Balsa System native synthesis from Balsa descriptions. This can be easily done by designing another set of gates and generating the required technology mapping files required by Balsa System.

Another important topic is the extension of ASCEnD to other technology processes. A previous version of the library was designed for a 180nm process. This version could be upgraded by designing the standard cells present in ASCEnD ST65. Moreover, a macro cell library to support different FPGA families can also be designed, as showed in [PON08]. In this way, asynchronous circuits design could be also experienced for array based approaches, which cost much less design time.

The ASCEnD ST65 is implemented for GPSVT devices. Other versions of this library can be designed in order to support a wider range of devices. Moreover, effort may also be employed to generate standard cells that support low power asynchronous circuits design. In order to do so, a set of cells implementing techniques like power shut off must be designed. However, this can easily achieved once the architecture of typical asynchronous standard cells is already defined by the present library.

Another possibility is the creation of macro cell generator based on ASCEnD cells. Such generators may be used to produce complex modules like multipliers, adders and parameterizable handshake components. The circuits generated by such tools may use different C-Element implementations guided by the tradeoffs presented in this work. Finally, a possible future work is upgrading the Sticky tool, to enable it automate more extensively the generation of stick diagrams from Spice descriptions.

REFERENCES

[AMD05] Amde, M.; Felicijan, T.; Efthymiou A.; Edwards, D.; Lavagno, L.: *"Asynchronous on-chip Networks".* Computers and Digital Techniques, IEEE Proceedings, 152(2), March 2005, pp. 273-283.

[AMI10] *"American Megatrends Inc."* Available at http://www.ami.com

[APT10] *"Advanced Processor Technologies Group"* Available at http://intranet.cs.man.ac.uk/apt.

[ASC10] *"ASCEnD: Asynchronous Standard Cells Enabling n-Designs"* Available at https://corfu.pucrs.br/redmine/embedded/asynclib

[BAR00] Bardsley, A. *"Implementing Balsa Handshake Circuits"*. PhD. Thesis, Faculty of Science & Engineering, University of Manchester, 2000, 187p.

[BAR09] Bardsley, A.; Tarazona, L.; Edwards, D.: *"Teak: A Token-Flow Implementation for the Balsa Language"* In: Application of Concurrency to System Design (ACSD), 2009, pp. 23-31.

[BAR98] Bardsley, A. *"Balsa: An Asynchronous Circuit Synthesis System"*. MSc Dissertation, Faculty of Science & Engineering, University of Manchester, 1998, 162p.

[BER91] van Berkel, K.; Kessels, J.; Roncken, M.; Saejis, R.; Schalij, F.: *"The VLSI-programming language Tangram and its translation into handshake circuits"* In: Proceedings of the European Conference on Design Automation (EDAC), 1991, pp. 384-389.

[BER99] van Berkel, C. H.; Josephs, M. B., Nowick, S. M.: *"Scanning the Technology: Applications of Asynchronous Circuits".* Proceedings of the IEEE, 1999, pp. 223-233.

[BER99b] van Berkel, C. H.; Molnar, C. E.: *"Beware the Three-Way Arbiter"*, Solid State Circuits, IEEE Journal of, 34(6), June 1999, pp.840-848.

[BOU07] Bouesse, F.; Ninon, N.; Sicard, G.; Renaudin, M.; Boyer, A.; Sicard, E.: *"Asynchronous Logic Vs Synchronous Logic: Concrete Results on Electromagnetic Emissions and Conducted Susceptibility".* In: 6th International Workshop on Electromagnetic Compatibility of Integrated Circuits (EMC), 2007, 5p.

[BRO05] Brown, S.; Vranesic, Z.: *"Fundamentals of Digital Logic With VHDL"* New York: McGraw Hill, 2005, 939 p.

[CAD07] Cadence Design Systems *"Cadence Abstract Generator User Guide"*. San Jose, 2007, 434p.

[CAD09] Cadence Design Systems "LEF/DEF Language Reference". San Jose, 2009, 413p.

[CAD10] *"Cadence Design Systems Inc."* Available at http://www.cadence.com.

[CAD10b] Cadence Design Systems *"Encounter Library Characterizer User Guide"*. San Jose, 2010, 174p.

[CHO07] Chong, K. S.; Gwee, B. H., Chang, J. S.: *"Design of Several Asynchronous-Logic Macrocells for a Low-Voltage Micropower Cell Library"*. Circuits, Devices & Systems, IET, 1(2), April 2007, pp. 161-169

[CHO92] Chong, J. W. W.; Forbes, R. G.: *"Design of Basic CMOS Cell Library"* Circuits, Devices and Systems, IEEE Proceedings G, 139(2), April 1992, pp 256-260.

[CIL02] Ciletti, M. D.: *"Advanced Digital Design with the Verilog HDL"* Prentice Hall, 2002, 1008p.

[CLE00] Clein, D.: *"CMOS IC Layout: Concepts, Methodologies, and Tools"* Woburn: Butterworth-Heinemann. 2000, 261 p.

[COR06] Cortadella, J.; Kondratyev, A.; Lavagno, L.; Sotiriou, C. S.: *"Desynchronization: Synthesis of Asynchronous Circuits from Synchronous Specifications".* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 25(10), October 2006, pp. 1904-1921

[CRI10] Cristófoli, L. F.; Henglez, A.; Benfica, J.; Bolzani, L.; Vargas, F.; Atienza, A.; Silva, F.: *"On the Comparison of Synchronous Versus Asynchronous Circuits Under the Scope of Conducted Power-Supply Noise".* In Electromagnetic Compatibility (APEMC), 2010 Asia Pacific Symposium on, 2010, pp. 1047-1050.

[DJI07] Djigbenou, J. D.; Ha D., S.: *"Development and Distribution of TSMC 0.25 µm Standard CMOS Library Cells".* In: Microelectronic Systems Education, 2007. MSE '07. IEEE

International Conference on, 2007, pp. 27-28.

[DOA05] Doan, C. H.; Emami, S.; Niknejad, A. M.; Brodersen, R. W.: *"Milimeter-wave CMOS Design"* Solid-State Circuits, IEEE Journal of, 2005, pp. 144-155.

[EDW06] Edwards, D.; Bardsley, A; Janin, L.; Plana, L.; Toms, W. *"Balsa: A Tutorial Guide"*. 2006, 157p.

[ERI03] Eriksson, H.; Larsson-Edefors, P.; Henriksson, T.; Svensson, C. *"Full-Custom vs. Standard-Cell Design Flow – an Adder Case Study"*. In: Asia and South Pacific Design Automation Conference (ASP-DAC), 2003, pp. 507-510.

[FER04] Ferretti, M.: *"Single-Track Asynchronous Pipeline Template"*. Doctor Thesis, Faculty of The Graduate School, Electrical Engineering, University of Southern California, 2004, 170p.

[FER06] Ferretti, M.; Beerel, P. A.: *"High Performance Asynchronous Design Using Single-Track Full-Buffer Standard Cells"*. Solid-State Circuits, IEEE Journal of, 41(6), June 2006, pp. 1444-1454.

[GRA03] Grad, J.; Stine, J. E. *"A Standard Cell Library for Student Projects".* In: Microelectronic Systems Education, IEEE International Conference on, 2003, pp. 98-99.

[HAS03] Hashimoto, M.; Fujimori, K.; Onodera, H.: *"Standard Cell Libraries with Various Driving Strength Cells for 0.13, 0.18 and 0.35µm Technologies"* In: Asia and South Pacific Design Automation Conference (ASP-DAC), 2003, pp. 589-590

[HEN02] Hennessy, J. L.; Patterson, D.: *"Computer Architecture: A Quantitative Approach".* San Mateo, CA: Morgan Kaufmann, 2002, 1336p.

[IEE01] IEEE Computer Society *"IEEE Standard for Standard Delay Format (SDF) for the Electronic Design Process"*. 2001, 88p.

[INT10] *"Intel Corporation"* Available at http://www.intel.com

[ITR01] International Technology Roadmap for Semiconductors. *"ITRS 2001 Edition"* http://www.itrs.net/, 2001.

[ITR08] International Technology Roadmap for Semiconductors. *"ITRS 2008 Edition"* http://www.itrs.net/, 2008.

[ITR09] International Technology Roadmap for Semiconductors. *"ITRS 2009 Edition"* http://www.itrs.net/, 2009.

[JAM99] Jambek, A. B.; NoorBeg, A. R.; Ahmad, M. R. *"Standard Cell Library Development"*. In: 11th International Conference on Microelectronics (ICM), 1999, pp. 161-163.

[KES01] Kessels, J.; Peeters, A.: *"The Tangram Framework: asynchronous circuits for low power"* In: Asia and South Pacific Design Automation Conference (ASP-DAC), 2001, pp. 255-260.

[KSC07] Kwen-Siong, C.; Bah-Hwee G.; Chang, J. S.: *"A Simple Methodology of Designing Asynchronous Circuits Using Commercial IC Design Tools and Standard Library Cells".* In: International Symposium on Integrated Circuits (ISIC), 2007, pp. 176-179.

[KWE07] Kwen-Siong, C.; Bah-Hwee G.; Chang, J. S.: *"Energy-Efficient Synchronous-Logic and Asynchronous-Logic FFT/IFFT Processors".* IEEE Journal of Solid-State Circuits, 42(9), September 2007, pp. 2034-2045.

[MAR06] Martin, A. J.; Nyström, M. *"Asynchronous Techniques for System-on-Chip Design".* Proceedings of the IEEE, 94(6), 2006, pp. 1089-1120.

[MAU03] Maurine, P.; Rigaud, J. B.; Bouesse, F.; Sicard, G.; Renaudin M.: *"Static Implementation of QDI Asynchronous Primitives"* In: International Workshop on Power and Timing Modeling, Optimization and Simulation, 2003, pp. 181-191.

[MIC94] De Micheli, G. *"Synthesis and Optimization of Digital Circuits"*. New York: McGraw-Hill, 1994, 579p.

[MOS10] *"MOSIS Integrated Circuit Fabrication Service"* Available at http://www.mosis.com.

[MUL57] Muller, D. E.; Bartky, W. S. *"A Theory of Asynchronous Circuits"* In: Proceedings of an International Symposium on the Theory of Switching, Cambridge, 1957, pp. 204-243.

[MYE01] Myers, C. J.: *"Asynchronous Circuit Design"*. New York: John Wiley & Sons, Inc. 2001, 422p.

[OZD04] Ozdag, R. O.; Beerel, P. A.: *"A Channel Based Asynchronous Low Power High Performance Standard-Cell Based Sequential Decoder Implemented with QDI Templates"* In: International Symposium on Asynchronous Circuits and Systems (ASYNC), 2004, pp. 187-197

[OZD06] Ozdag, R. O.; Beerel, P. A.: *"An Asynchronous Low-Power High-Performance Sequential Decoder Implemented with QDI Templates"* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 14(9), September 2006, pp. 975-985

[PON08] Pontes, J. *"Design and Prototyping of Non-synchronous Interfaces and Intrachip Networks in FPGAs."* MSc. Dissertation, Faculty of Science Informatics, Pontifícia

Universidade Católica do Rio Grande do Sul, 2008, 116p. In Portuguese.

[PON10] Pontes, J. J. H.; Moreira, M. T.; Moraes, F. G.; Calazans, N. L. V.: *"Hermes-AA: A 65nm Asynchronous NoC Router with Adaptive Routing"* In: IEEE International SoC Conference (SOCC), 2010, pp. 493-498.

[PON10b] Pontes, J. J. H.; Moreira, M. T.; Moraes, F. G.; Calazans, N. L. V.: *"Hermes-A – An Asynchronous NoC Router with Distributed Routing"* In: International Workshop on Power and Timing Modeling, Optimization and Simulation, 2010, 10p.

[RAB03] Rabaey, J. M.; Chandrakasan A.; Nikolic, B. *"Digital Integrated Circuits a Design Perspective"*. Upper Saddle River: Pearson Education, 2003, 761p.

[REI00] Reis, R. *"Concepção de Circuitos Integrados"*. Porto Alegre: Sagra Luzzato, 2000, 252p. In Portuguese.

[SAI02] Saint, C.; Saint, J. *"IC Layout Basics a Pratical Guide"*. New York: McGraw-Hill, 2002, 300p.

[SAI04] Saint, C.; Saint, J. "*IC Mask Design Essential Layout Techniques*". New York: McGraw-Hill, 2004, 394p.

[SCH96] Schneier, B.: *"Applied Cryptography, Second Edition"* New York: John Wiley & Sons, INC. 1996, 758 p.

[SHA96] Shams, M.; Ebergen, J. C.; Elmasry, M. I.: *"A Comparison of CMOS Implementations of an Asynchronous Circuits Primitive: the C-Element"* In: International Symposium on Low Power Electronics and Design (ISLPED), 1996, pp. 93-96.

[SHA97] Shams, M.; Ebergen, J. C.; Elmasry, M. I.: *"Optimizing CMOS Implementations of the C-Element"* In: IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD), 1997, pp. 700-705.

[SHA98] Shams, M.; Ebergen, J. C.; Elmasry, M. I.: *"Modeling and Comparing CMOS Implementations of the C-Element"* IEEE Transactions on Very Large Scale Integration Systems, 6(4), December 1998, pp. 563-567.

[SMI05] Smirnov A.; Taubin A.; Su M.; Karpovsky M.: *"An Automated Fine-Grain Pipelining Using Domino Style Asynchronous Library".* In: Application of Concurrency to System Design, 2005, pp. 68-76

[SMI97] Smith, M. J. S.: *"Application-Specific Integrated Circuits".* Addison-Wesley Professional, 1997, 1040p.

[SPA01] Sparso, J.; Furber, S. *"Principles of Asynchronous Circuit Design – A Systems Perspective"*. London: Kluwer Academic Publishers, 2001, 337p.

[STM10] *"STMicroelectronics"* Available at http://www.st.com.

[SYN07] Synopsys Inc. *"Liberty Reference Manual"* 2007, 305p.

[SYN10] *"Synopsys Inc."* Available at http://www.synopsys.com

[TOM06] Toms, W. B. *"Synthesis of Quasi-Delay-Insensitive Datapath Circuits"*. Phd. Thesis, Faculty of Science & Engineering, University of Manchester, 2006, 237p.

[TSM10] *"Taiwan Semiconductor Manufacturing Company Limited"* Available at http://www.tsmc.com

[VAH10] Vahid, F. *"Digital Design with RTL Design, VHDL and Verilog"* Riverside: John Wiley & Sons, Inc. 2010, 592 p.

[WON05] Wong, B. P.; Mittal, A.; Cao, Y.; Starr, G. *"Nano-CMOS Circuit and Physical Design"* New Jersey: John Willey & Sons, INC. 2005, 383 p.

[WON09] Wong, B. P.; Zach, F.; Moroz, V.; Mittal, A.; Starr, G.; Kahng, A.: *"Nano-CMOS Design for Manufacturability: Robust Circuit and Physical Design for Sub-65nm Technology Nodes"* New Jersey: John Willey & Sons, Inc. 2009, 385 p.

[XFA10] *"XFab Mixed Signal Foundry"* Available at http://www.xfab.com

[XIL10] *"Xilinx Inc."* Available at http://www.xilinx.com.

[YAH06] Yahya E.; Renaudin M.: *"QDI Latches Characteristics and Asynchronous Linear-Pipeline Performance Analysis"* In: International Workshop on Power and Timing Modeling, Optimization and Simulation, 2006, 2p.

[YEO09] Yeoh, K. L.; Lim, J. S.; Goh, K. L.; Tee, S. M.: *"Custom Digital Cell Generation Flow for 65nm Processes"* In: International SoC Design Conference, 2009, pp. 177-181

A APPENDIX A

This appendix present the results of the simulations of different sizes of inverters used to define the PMOS-to-NMOS ratio and the output driving strength of different cell implementations.

PMOS-to-NMOS ratio:

The PMOS-to-NMOS ratio is a constant that defines the size of the PMOS as a function of the size of the NMOS. This constant was defined by simulating an inverter for a fixed output load and input slope with varying NMOS and PMOS sizes. A total of 244 values for the PMOS transistor size were simulated for each size of the NMOS. The results are demonstrated in Figure A.**1-**Figure A.8 and summarized in Table A.1. From the obtained results, an average ratio of 2.11 was found and adopted by this work.

Table A.1 – Summarized results for different PMOS-to-NMOS ratio simulations.

NMOS size (μm)	PMOS size (μm)	PMOS-to-NMOS ratio
0.135	0.295	**2.19**
0.2	0.43	**2.15**
0.3	0.645	**2.15**
0.4	0.855	**2.14**
0.5	1.03	**2.06**
0.6	1.24	**2.07**
0.7	1.45	**2.07**
0.8	1.67	**2.09**

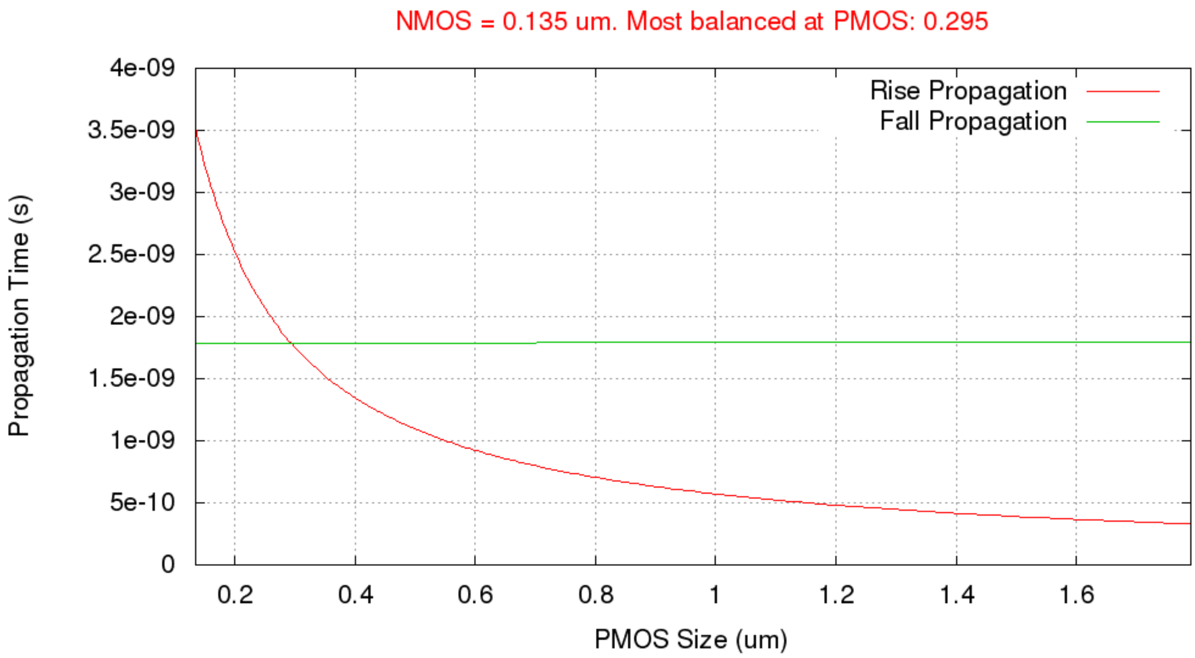


Figure A.1 – Results for the simulation of an inverter with varying PMOS sizing and an NMOS transistor size of 0.135µm.

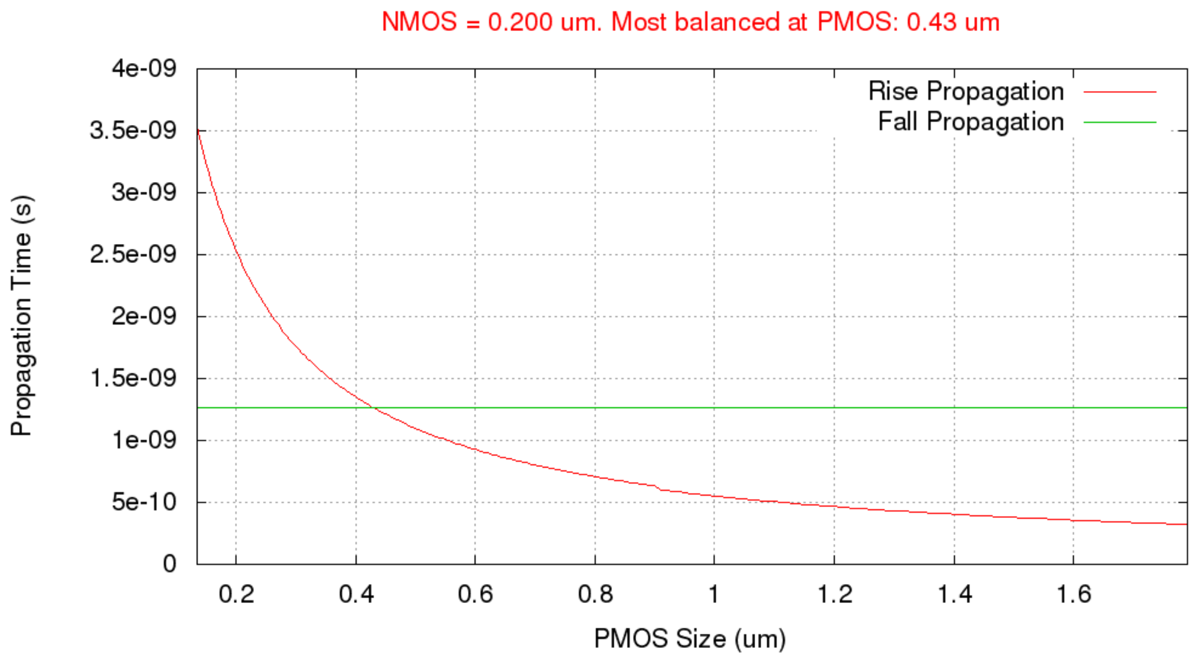


Figure A.2 – Results for the simulation of an inverter with varying PMOS sizing and an NMOS transistor size of 0.200µm.

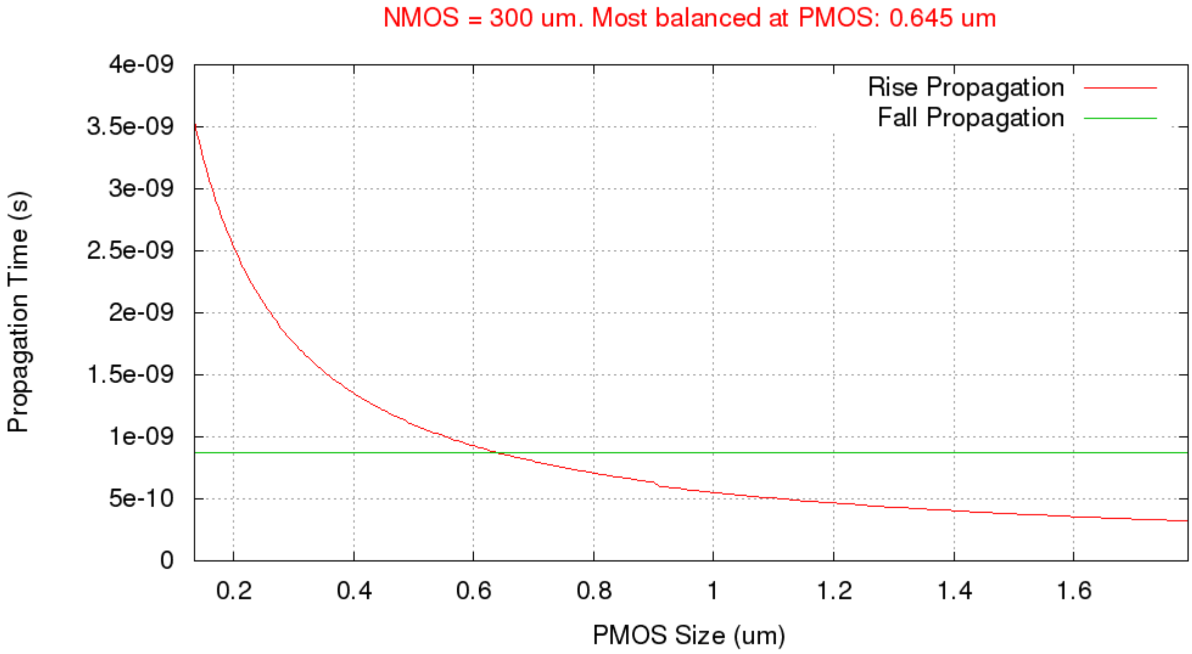


Figure A.3 – Results for the simulation of an inverter with varying PMOS sizing and an NMOS transistor size of 0.300µm.

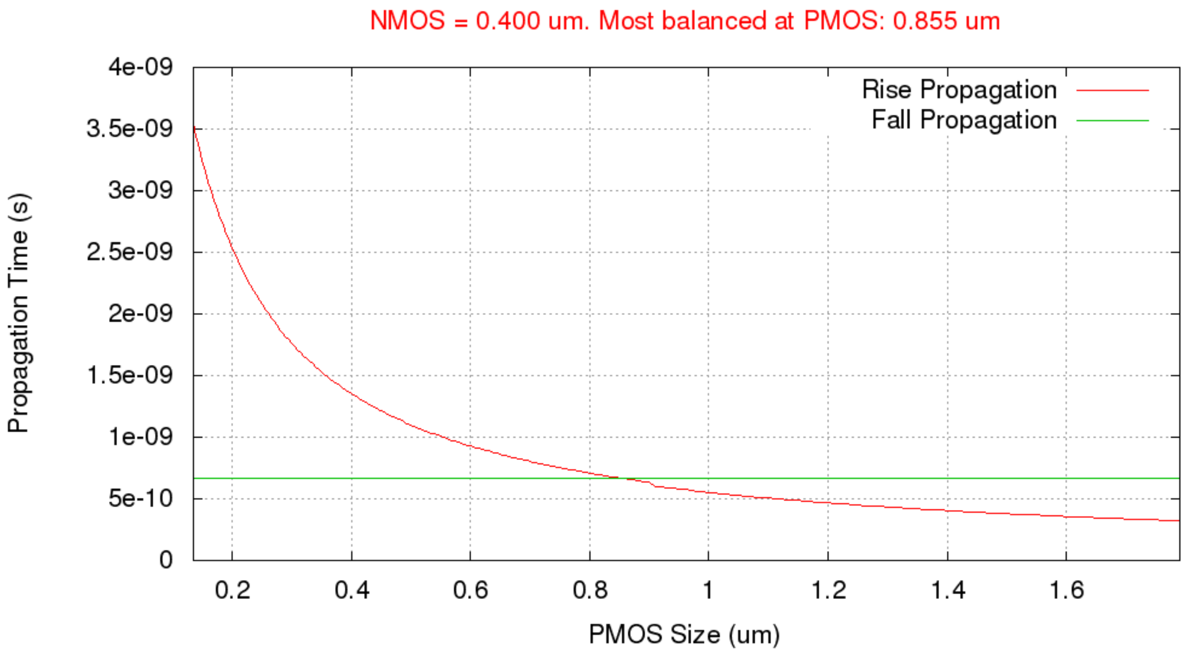


Figure A.4 – Results for the simulation of an inverter with varying PMOS sizing and an NMOS transistor size of 0.400µm.

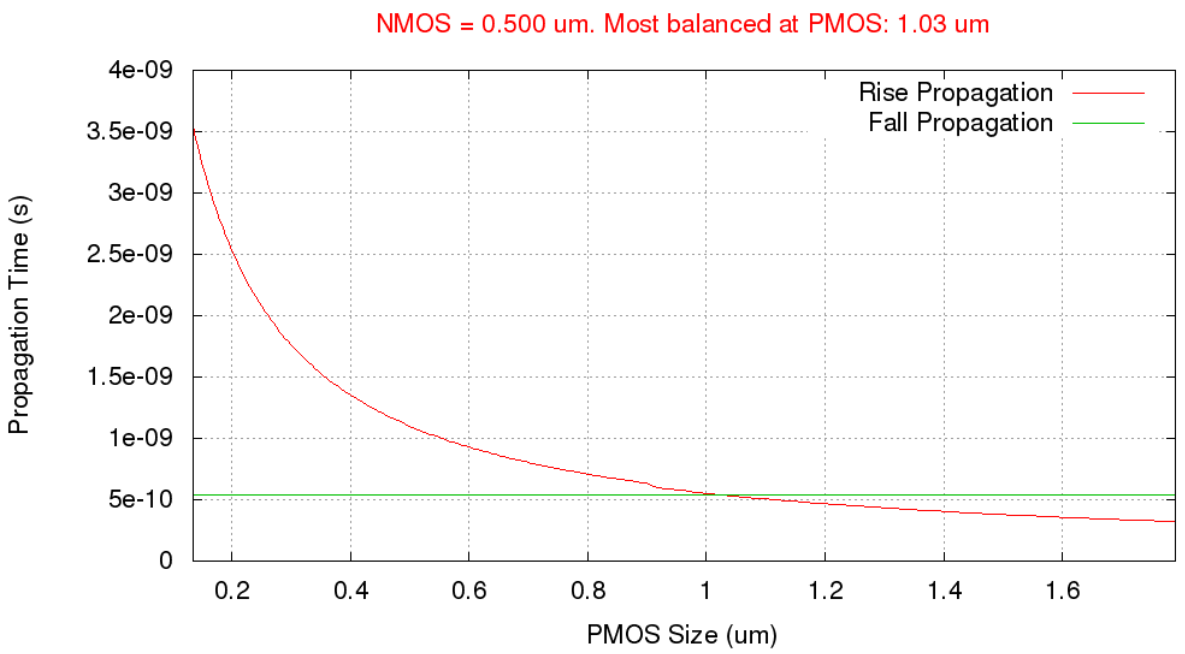


Figure A.5 – Results for the simulation of an inverter with varying PMOS sizing and an NMOS transistor size of 0.500µm.

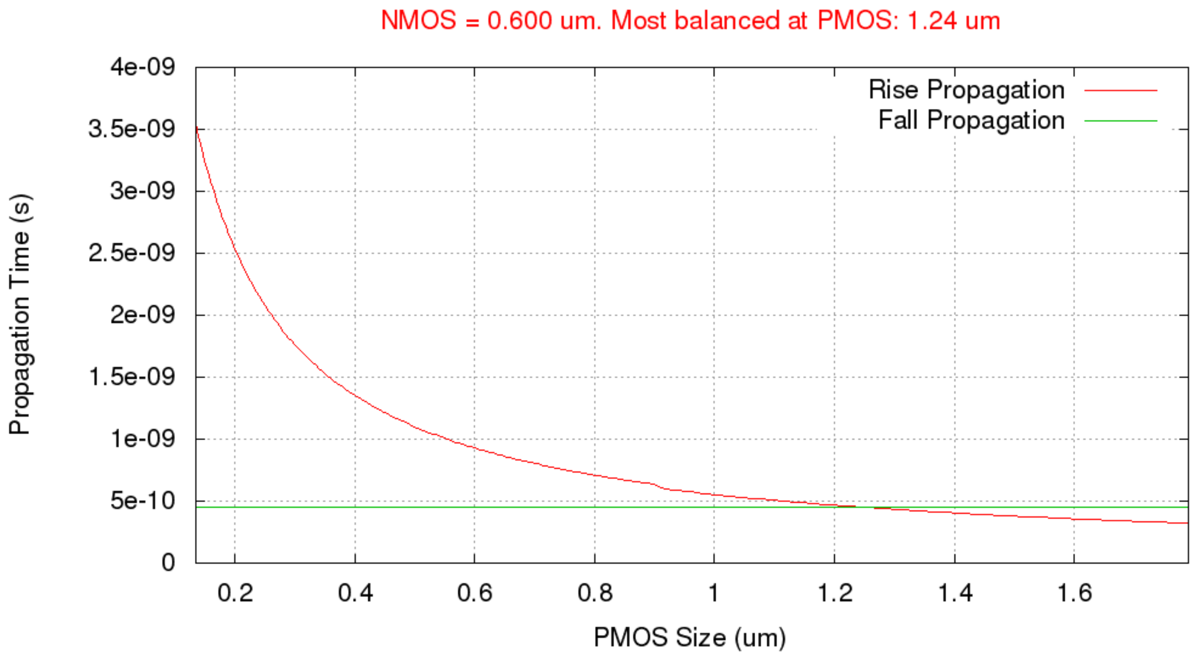


Figure A.6 – Results for the simulation of an inverter with varying PMOS sizing and an NMOS transistor size of 0.600µm.

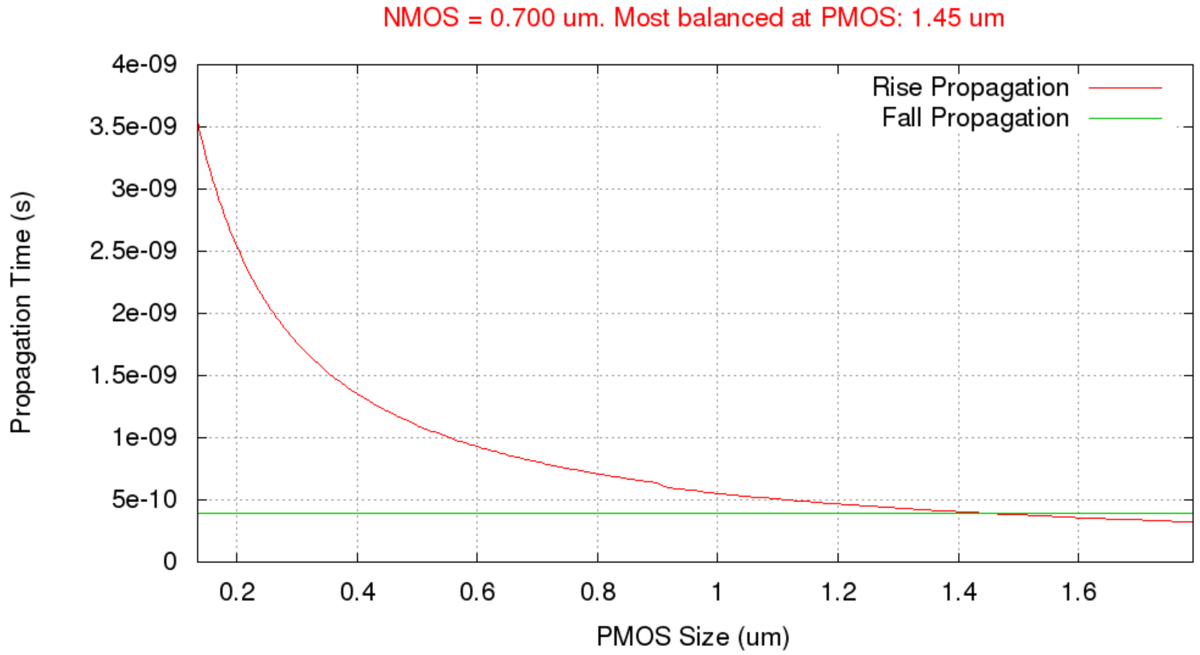


Figure A.7 – Results for the simulation of an inverter with varying PMOS sizing and an NMOS transistor size of 0.700µm.

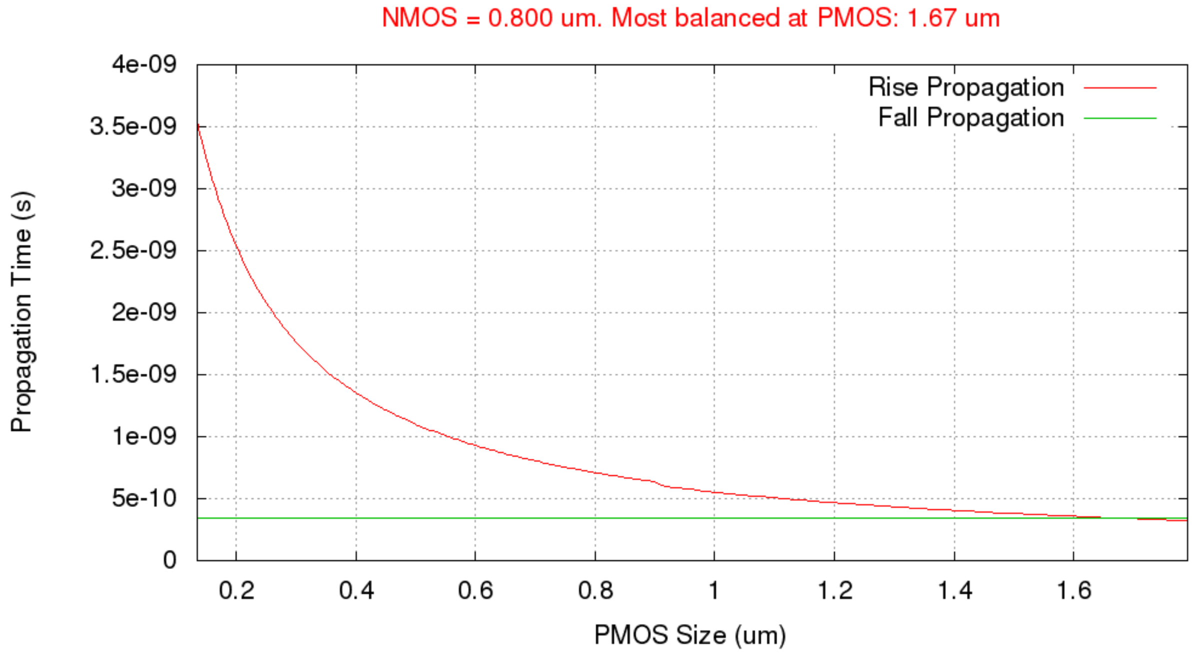


Figure A.8 – Results for the simulation of an inverter with varying PMOS sizing and an NMOS transistor size of 0.800µm.

Driving Strength:

The availability of different driving strengths for each cell is strongly related to the performance of a standard cell based design. Therefore, the definition of the load that each version of a standard cell is able to drive is essential in the design of a standard cell library. The definition adopted by the present work is that a X1 gate is able to drive an equivalent load of the one drove by an inverter at minimum size in a period of time of 1ns. A X2 drives two times this load, a X3 three times and so on.

Through simulation, it was verified that an inverter at minimum size was capable of driving 0.27pF in 1ns. From there, simulations were performed to determine what transistors size an inverter needed for every drive from X2 to X8. The results of the simulations are showed in Figure A.9 to Figure A.15. The red lines represent the time taken for each transistors dimension to drive the output load, while green lines correspond to the balance for rise and fall propagation delays. The point where the red lines cross the 1ns line is the best transistors size for that output drive. Moreover, the closer the green lines are to 1, the more balanced is the cell.

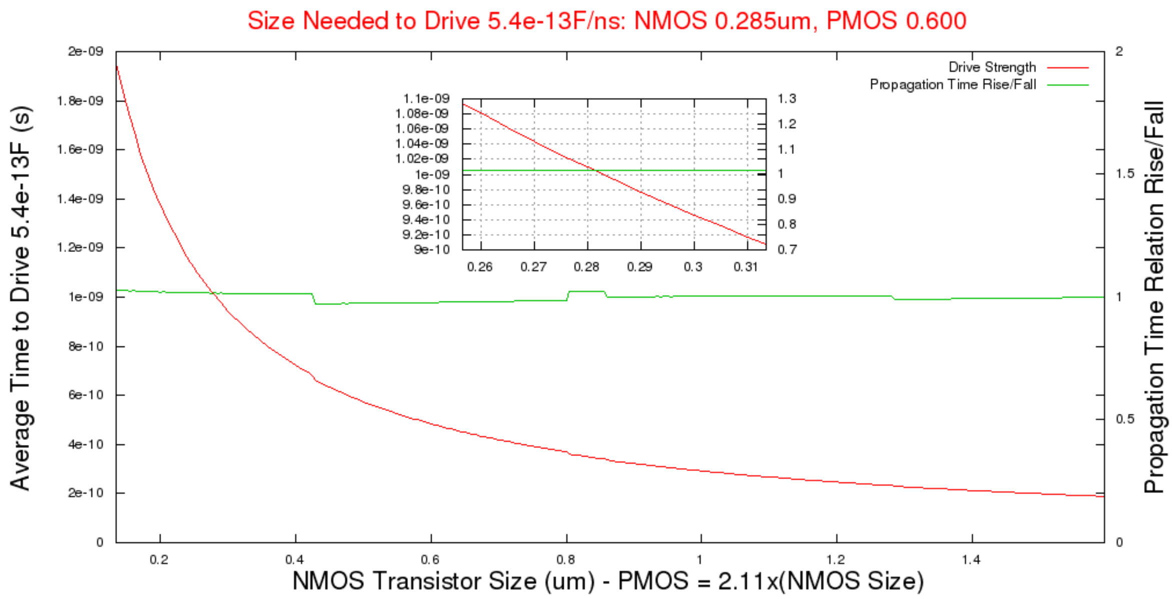


Figure A.9 – Results of simulations for an inverter with variable transistors size and an output load of 0.54pF (X2).

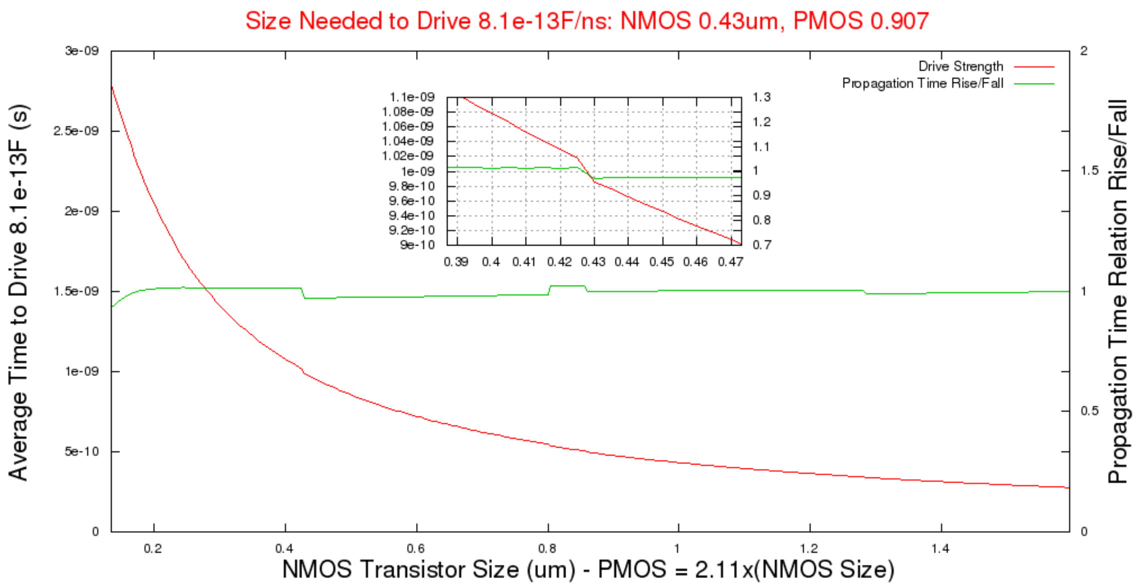


Figure A.10 – Results of simulations for an inverter with variable transistors size and an output load of 0.81pF (X3).

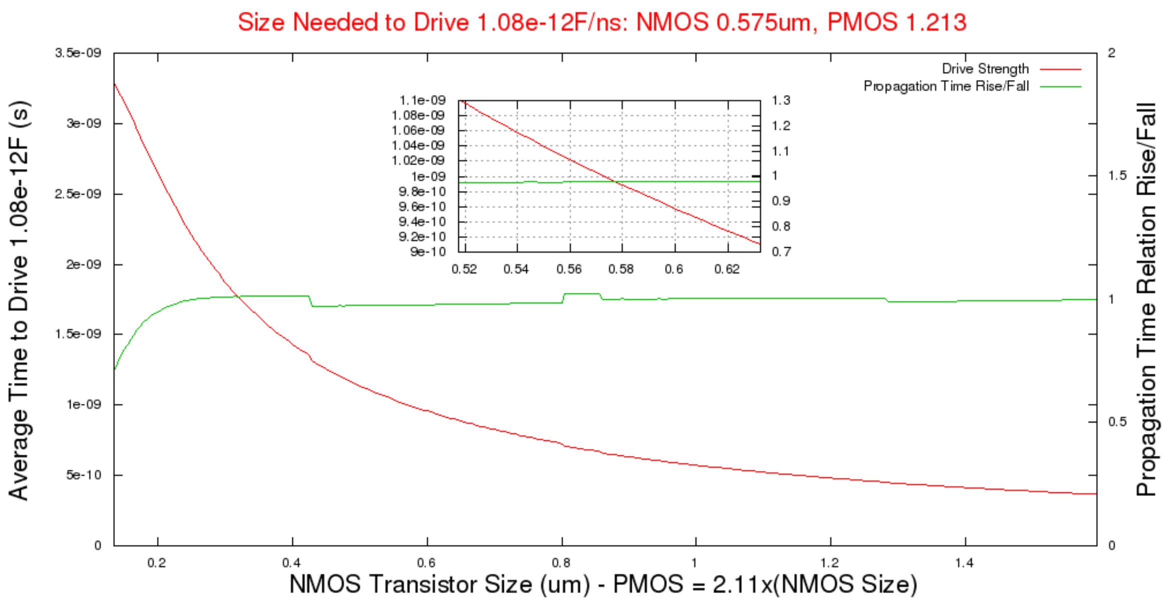


Figure A.11 – Results of simulations for an inverter with variable transistors size and an output load of 1.08pF (X4).

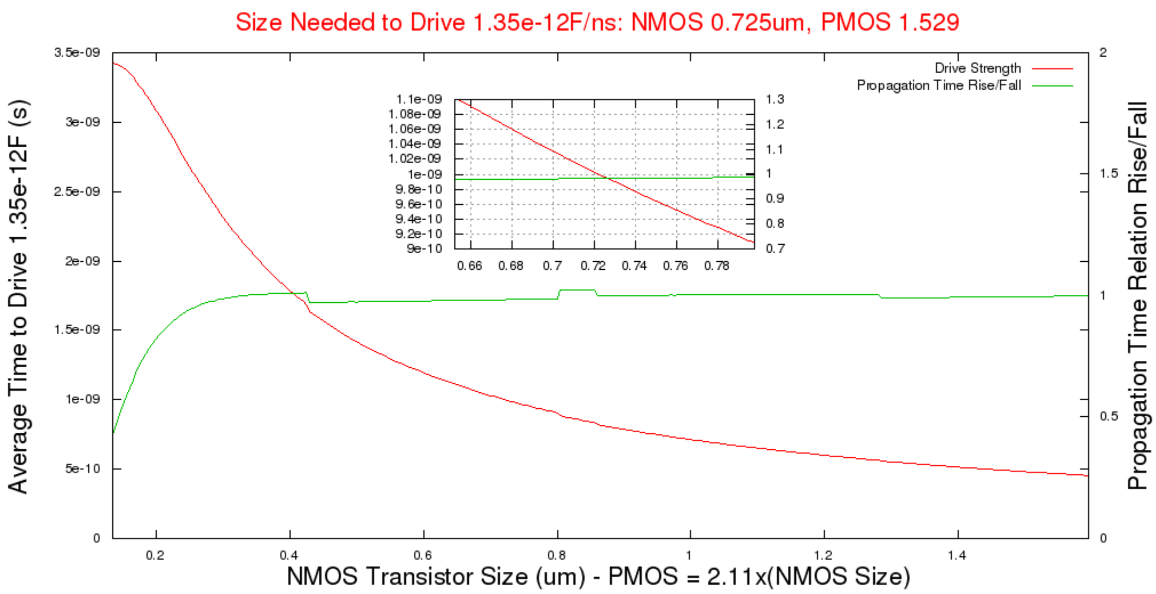


Figure A.12 – Results of simulations for an inverter with variable transistors size and an output load of 1.35pF (X5).

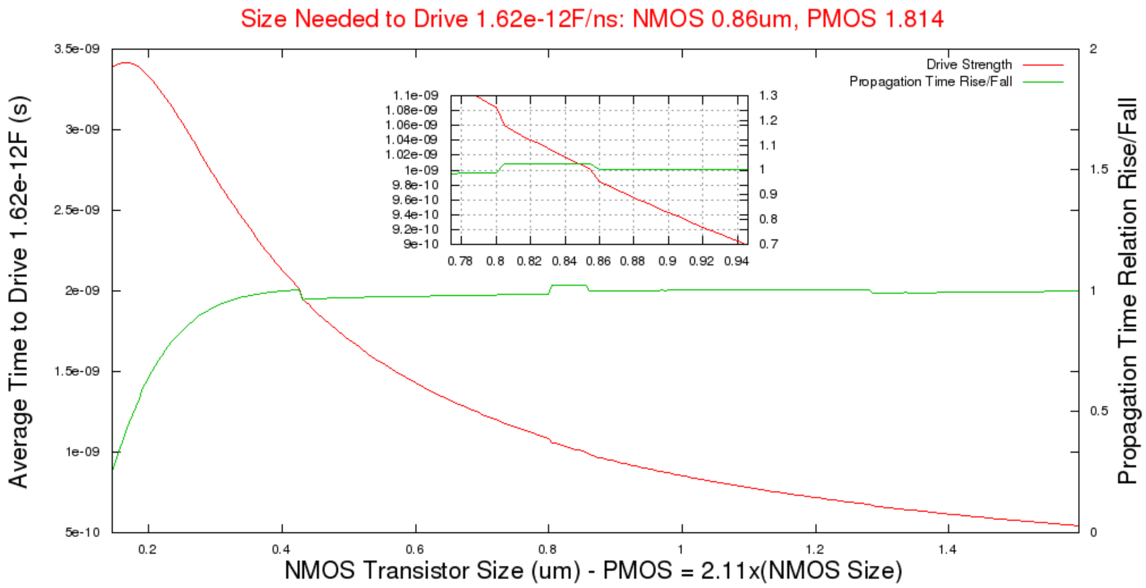


Figure A.13 – Results of simulations for an inverter with variable transistors size and an output load of 1.62pF (X6).

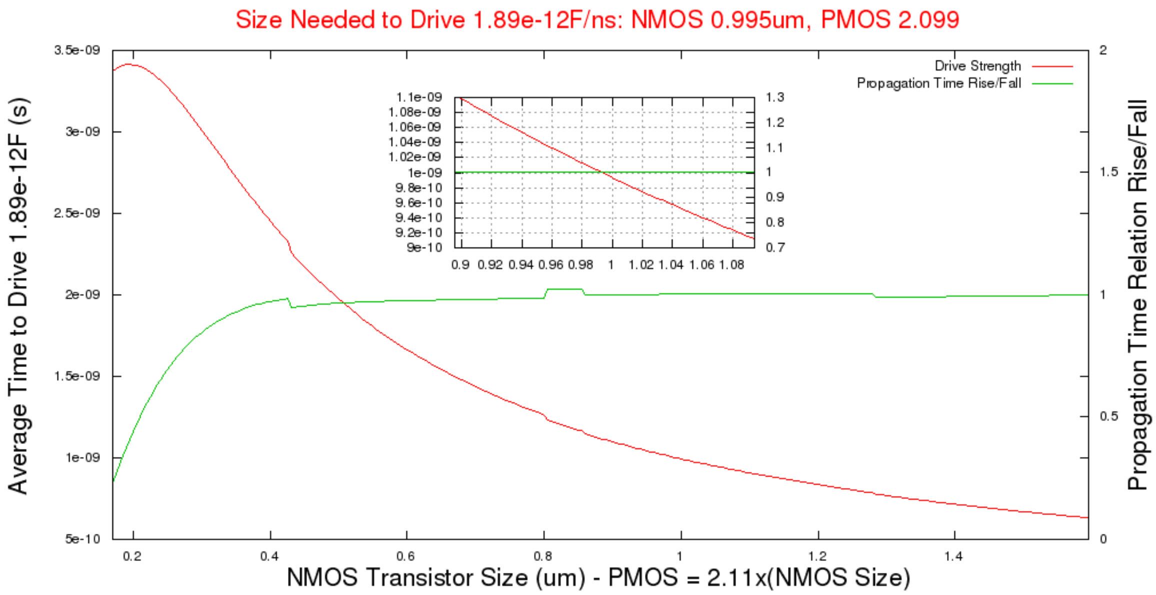


Figure A.14 – Results of simulations for an inverter with variable transistors size and an output load of 1.89pF (X7).

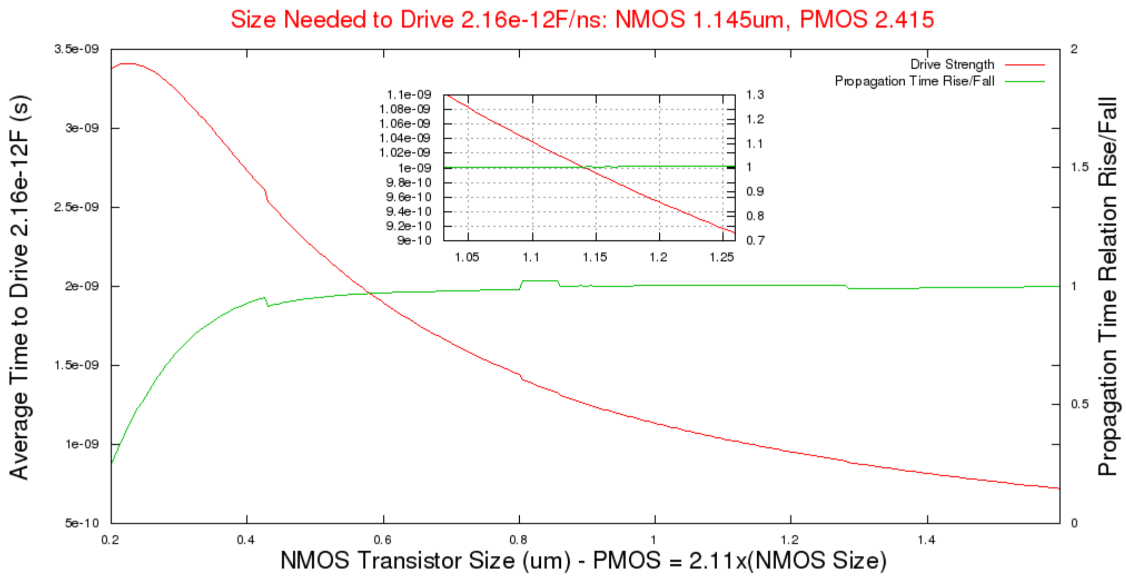


Figure A.15 – Results of simulations for an inverter with variable transistors size and an output load of 2.16pF (X8).

B Appendix B

This appendix presents the functionality of each designed cell from ASCEnD ST65, which is described through a logical function. The function that each cell must implement is described in Table B.2.

Table B.2 – Logical specification of each gate designed for ASCEnD ST65.

Gate	Logical Specification
C2	$Q_{i+1} = (A \wedge Q_i) \vee (A \wedge B) \vee (Q_i \wedge B)$
C_RST2	$Q_{i+1} = RST \wedge ((A \wedge Q_i) \vee (A \wedge B) \vee (Q_i \wedge B))$
C_SET2	$Q_{i+1} = SET \vee (A \wedge Q_i) \vee (A \wedge B) \vee (Q_i \wedge B)$
C1D2	$Q_{i+1} = A \vee (A \wedge B) \vee (Q_i \wedge B)$
C1D_RST2	$Q_{i+1} = RST \wedge (A \vee (A \wedge B) \vee (Q_i \wedge B))$
C1D_SET2	$Q_{i+1} = SET \vee A \vee (A \wedge B) \vee (Q_i \wedge B)$
C1U2	$Q_{i+1} = B \wedge ((A \wedge B) \vee (Q_i \wedge B))$
C1U_RST2	$Q_{i+1} = RST \wedge B \wedge ((A \wedge B) \vee (Q_i \wedge B))$
C1U_SET2	$Q_{i+1} = SET \vee (B \wedge ((A \wedge B) \vee (Q_i \wedge B)))$
C3	$Q_{i+1} = (A \wedge B \wedge C) \vee (A \wedge Q_i) \vee (B \wedge Q_i) \vee (C \wedge Q_i)$
C_RST3	$Q_{i+1} = RST \wedge ((A \wedge B \wedge C) \vee (A \wedge Q_i) \vee (B \wedge Q_i) \vee (C \wedge Q_i))$
C_SET3	$Q_{i+1} = SET \vee (A \wedge B \wedge C) \vee (A \wedge Q_i) \vee (B \wedge Q_i) \vee (C \wedge Q_i)$
C1D3	$Q_{i+1} = (A \wedge B \wedge C) \vee (A \wedge B) \vee (A \wedge Q_i) \vee (B \wedge Q_i) \vee (C \wedge Q_i)$
C1D_RST3	$Q_{i+1} = RST \wedge ((A \wedge B \wedge C) \vee (A \wedge B) \vee (A \wedge Q_i) \vee (B \wedge Q_i) \vee (C \wedge Q_i))$
C1D_SET3	$Q_{i+1} = SET \vee (A \wedge B \wedge C) \vee (A \wedge B) \vee (A \wedge Q_i) \vee (B \wedge Q_i) \vee (C \wedge Q_i)$
C1U3	$Q_{i+1} = (B \vee C) \wedge ((A \wedge B \wedge C) \vee (A \wedge Q_i) \vee (B \wedge Q_i) \vee (C \wedge Q_i))$
C1U_RST3	$Q_{i+1} = RST \wedge (B \vee C) \wedge ((A \wedge B \wedge C) \vee (A \wedge Q_i) \vee (B \wedge Q_i) \vee (C \wedge Q_i))$
C1U_SET3	$Q_{i+1} = SET \vee ((B \vee C) \wedge ((A \wedge B \wedge C) \vee (A \wedge Q_i) \vee (B \wedge Q_i) \vee (C \wedge Q_i)))$
C1D1U3	$Q_{i+1} = (B \vee C) \wedge ((A \wedge B \wedge C) \vee (A \wedge B) \vee (A \wedge Q_i) \vee (B \wedge Q_i) \vee (C \wedge Q_i))$
C1D1U_RST3	$Q_{i+1} = RST \wedge (B \vee C) \wedge ((A \wedge B \wedge C) \vee (A \wedge B) \vee (A \wedge Q_i) \vee (B \wedge Q_i) \vee (C \wedge Q_i))$
C1D1U_SET3	$Q_{i+1} = SET \vee ((B \vee C) \wedge ((A \wedge B \wedge C) \vee (A \wedge B) \vee (A \wedge Q_i) \vee (B \wedge Q_i) \vee (C \wedge Q_i)))$

FILTER2	$AA = \overline{RB} \wedge RA$ $AB = \overline{RA} \wedge RB$

C APPENDIX C

This appendix presents example of schematic and layout for the three variations of a C-Element implemented: Conventional, Muller and Symmetric. Moreover, examples of a Metastability Filter schematic and layout view are also presented. The complete ASCEnD ST65 set of views can be obtained at [ASC10].

Conventional C-Element:

The schematic of a two input Conventional C-Element is showed in Figure C.16 and its respective layout in Figure C.17.

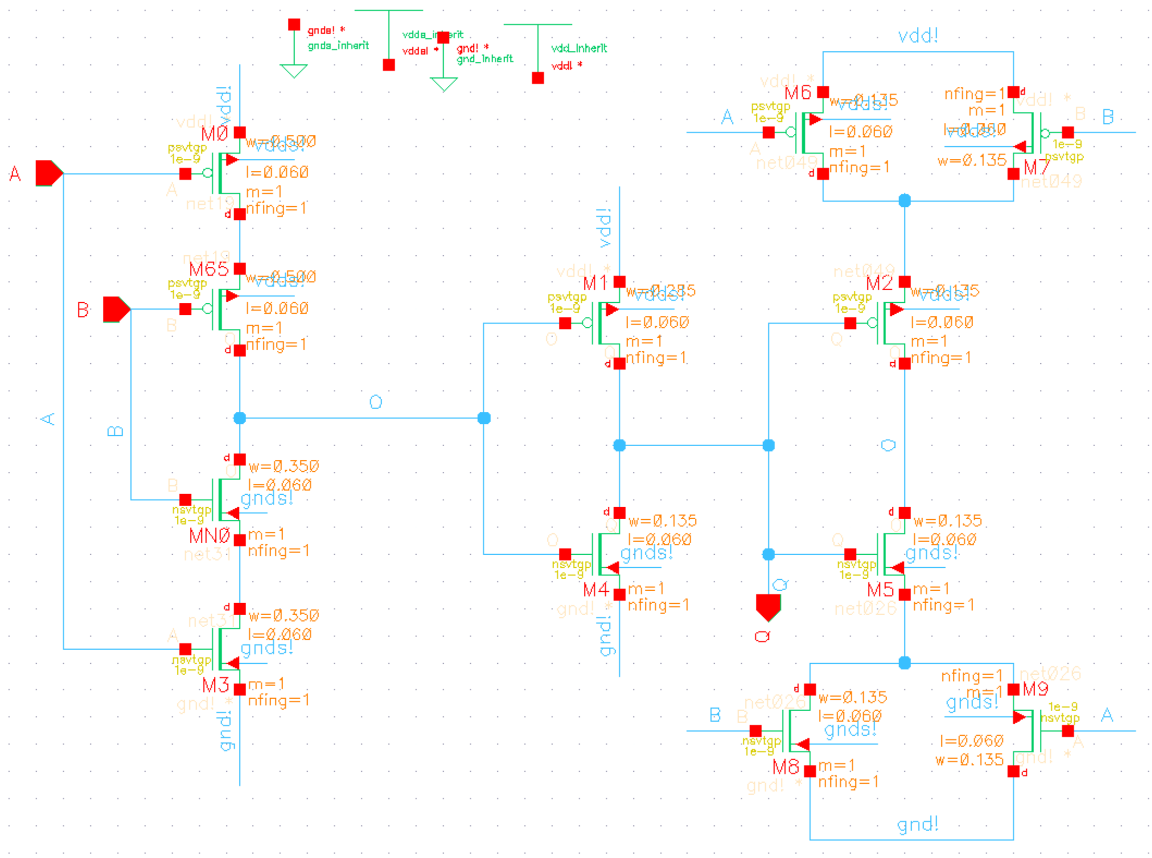


Figure C.16 – Schematic of a two input Conventional C-Element.

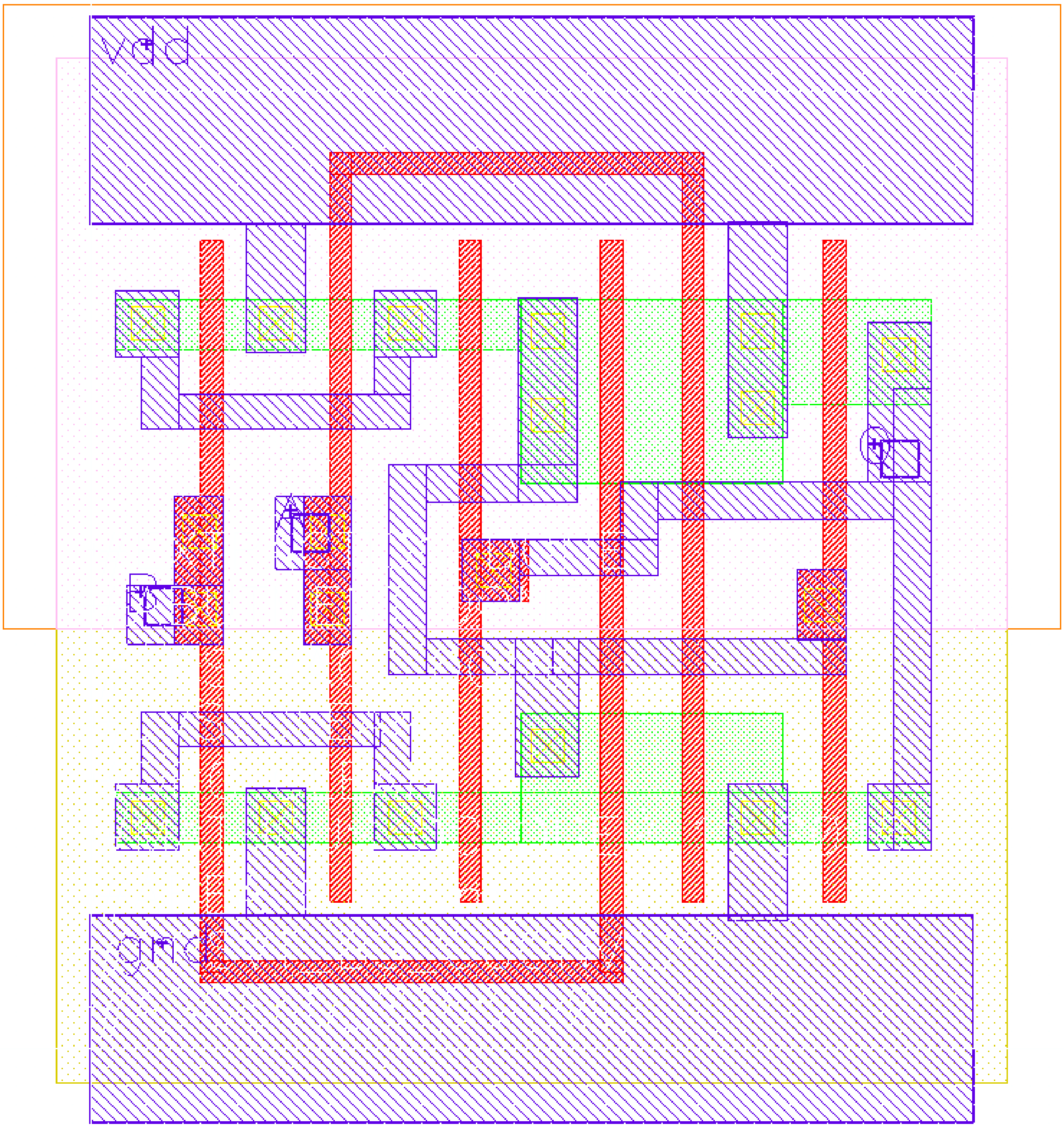


Figure C.17 – Layout of a two input Conventional C-Element.

Muller C-Element:

The schematic of a two input Muller C-Element is showed in Figure C.18 and its respective layout in Figure C.19.

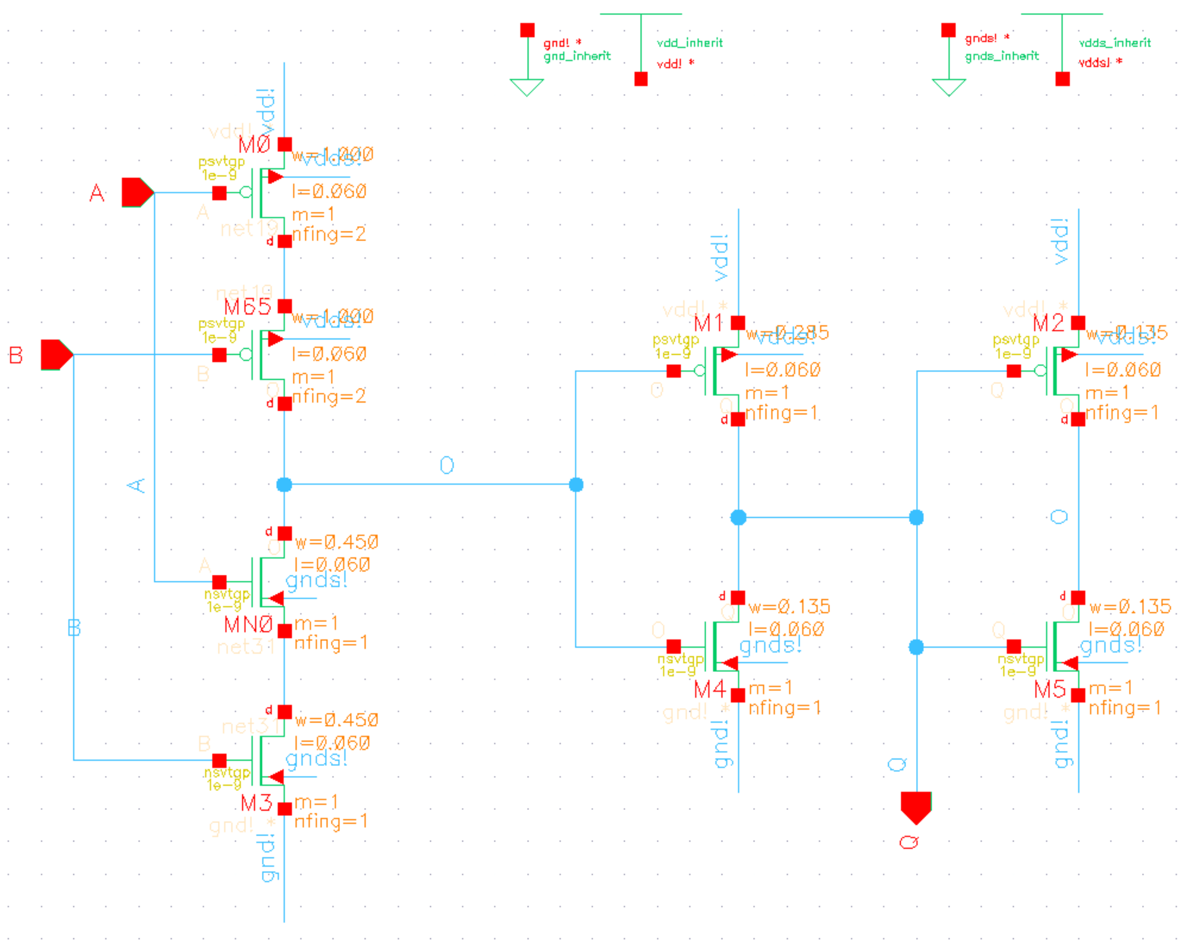


Figure C.18 – Schematic of a two input Muller C-Element.

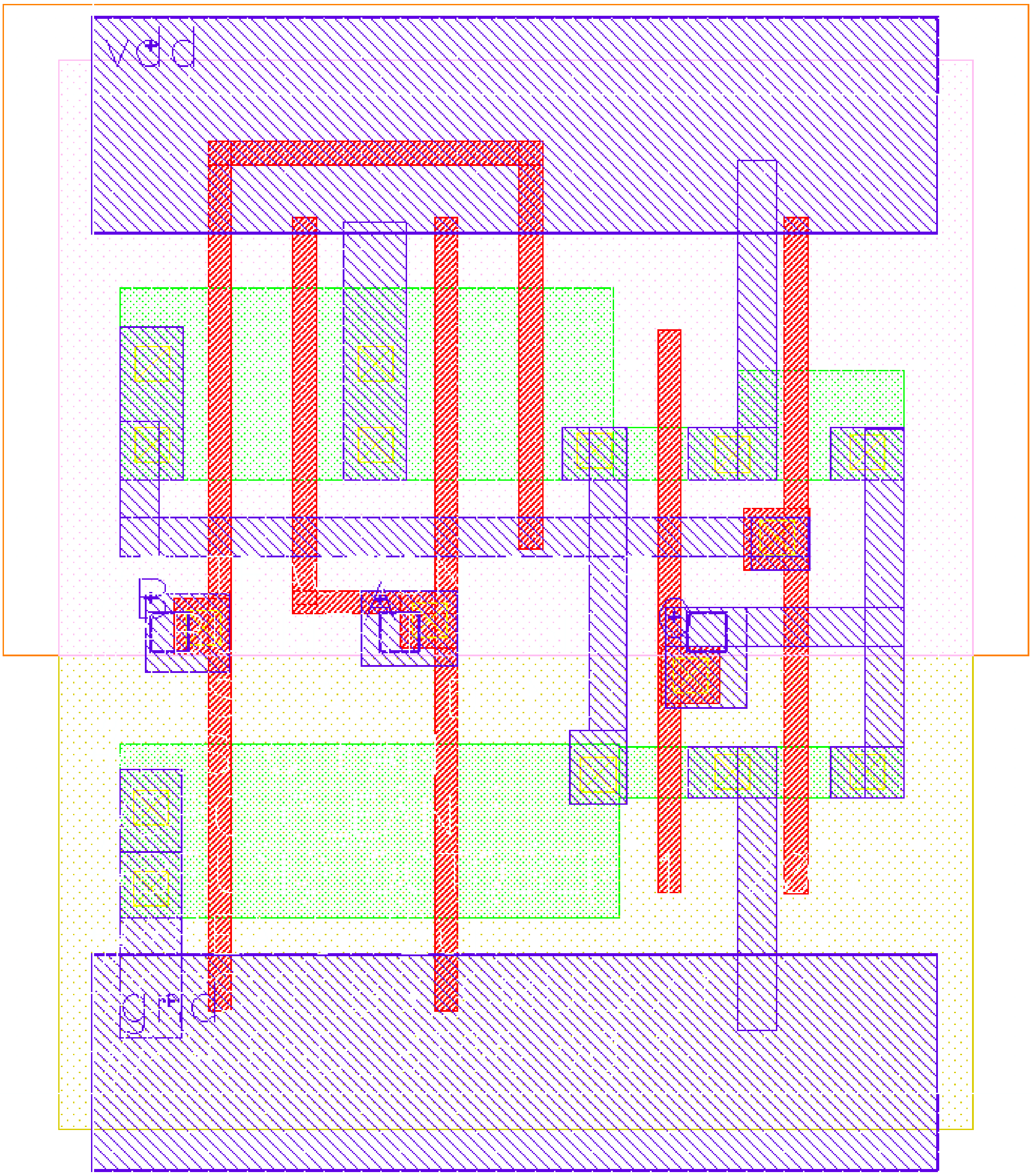


Figure C.19 – Layout of a two input Muller C-Element.

Symmetric C-Element:

The schematic of a two input Symmetric C-Element is showed in Figure C.20 and its respective layout in Figure C.21.

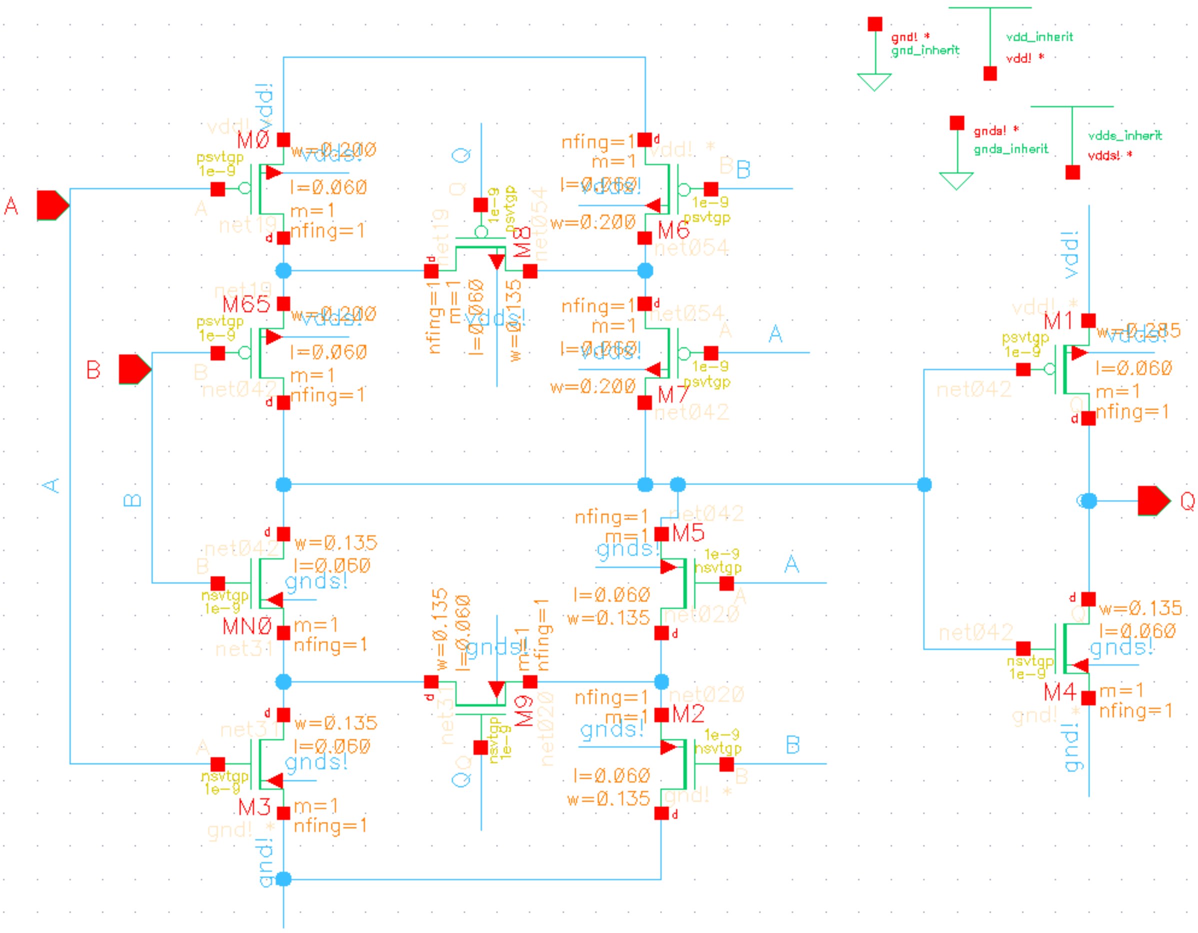


Figure C.20 – Schematic of a two input Symmetric C-Element.

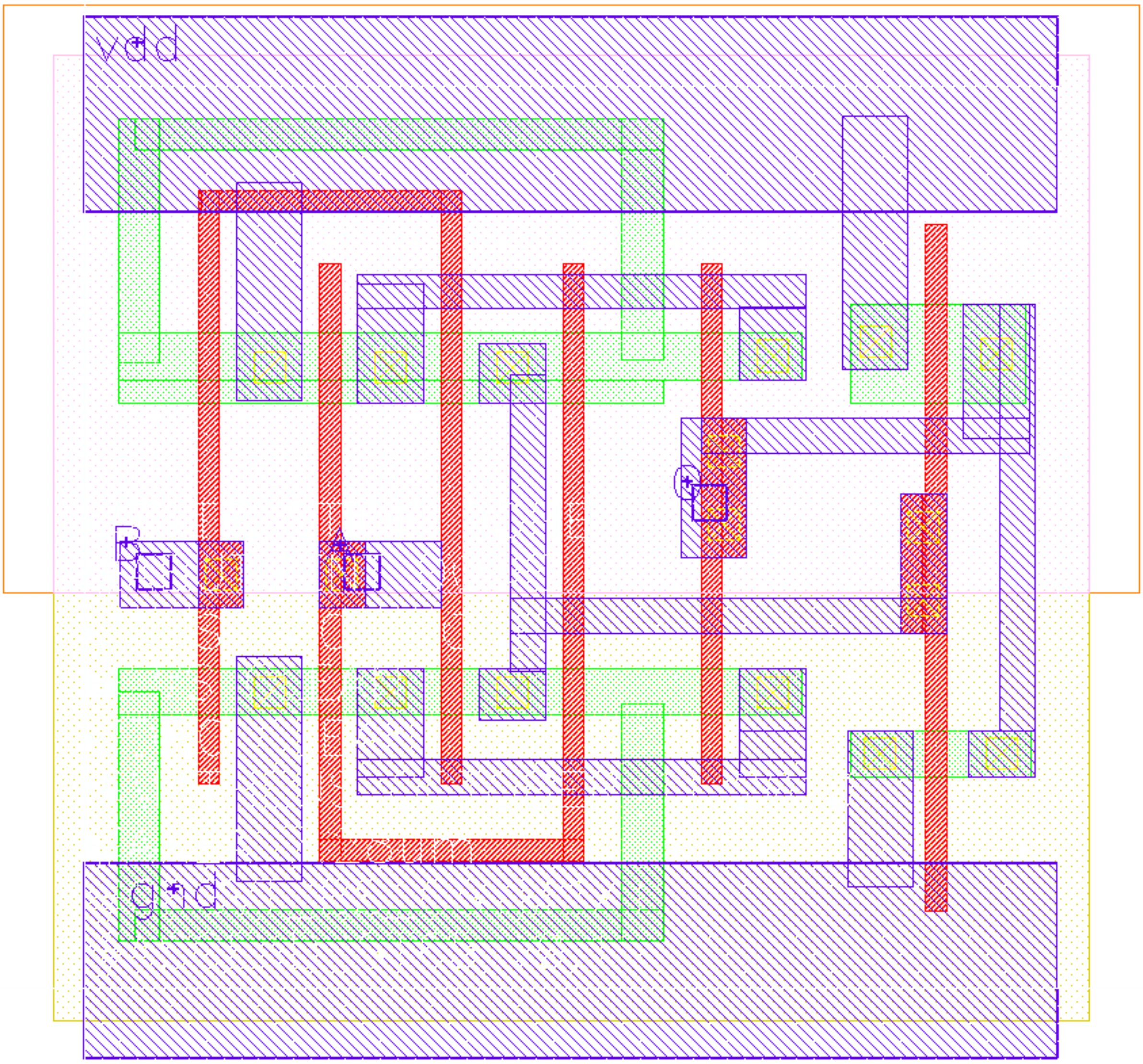


Figure C.21 – Layout of a two input Symmetric C-Element.

Metastability Filter:

The schematic of a two input Metastability Filter is showed in Figure C.22 and its respective layout in Figure C.23.

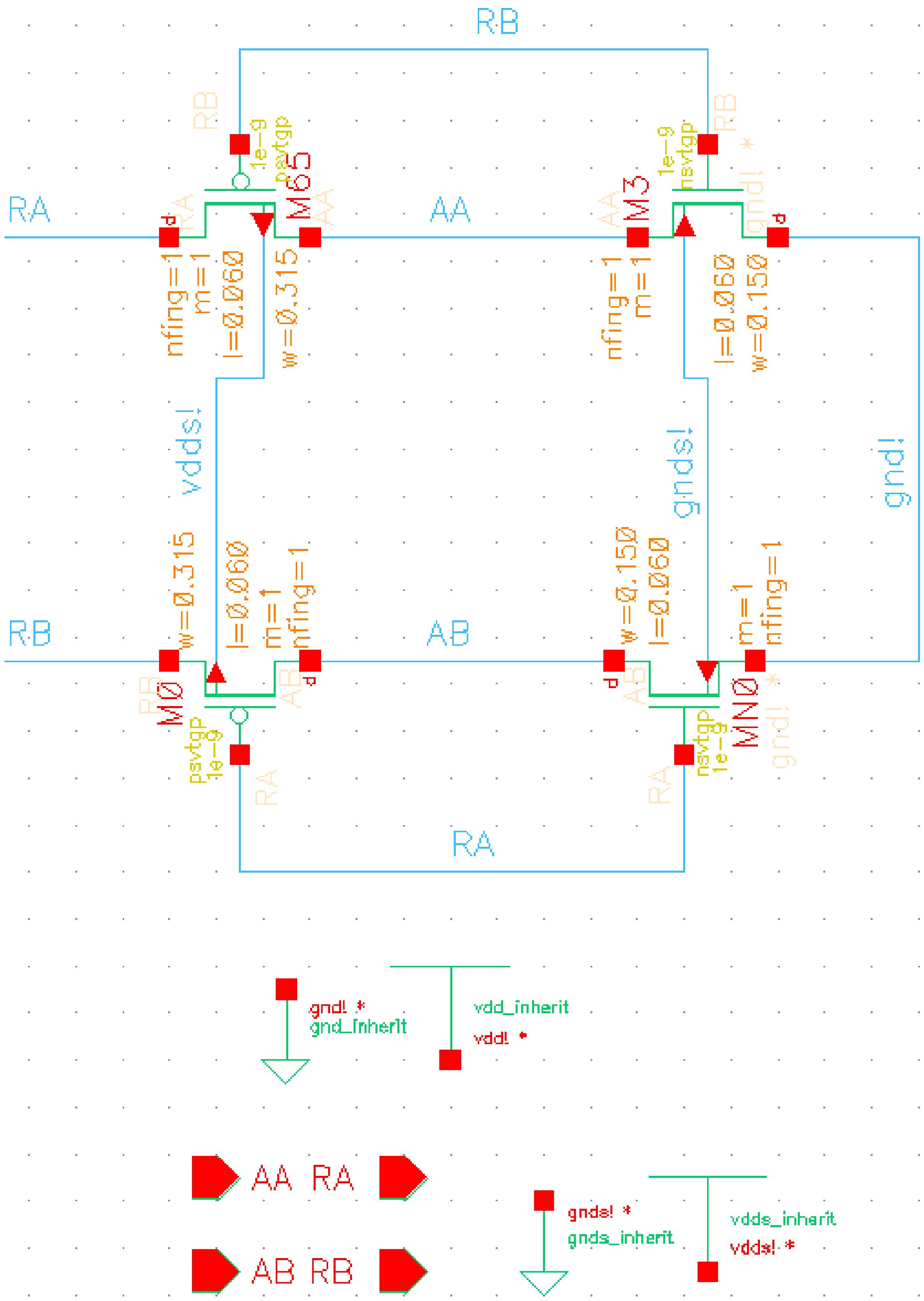


Figure C.22 – Schematic of a two input Metastability Filter.

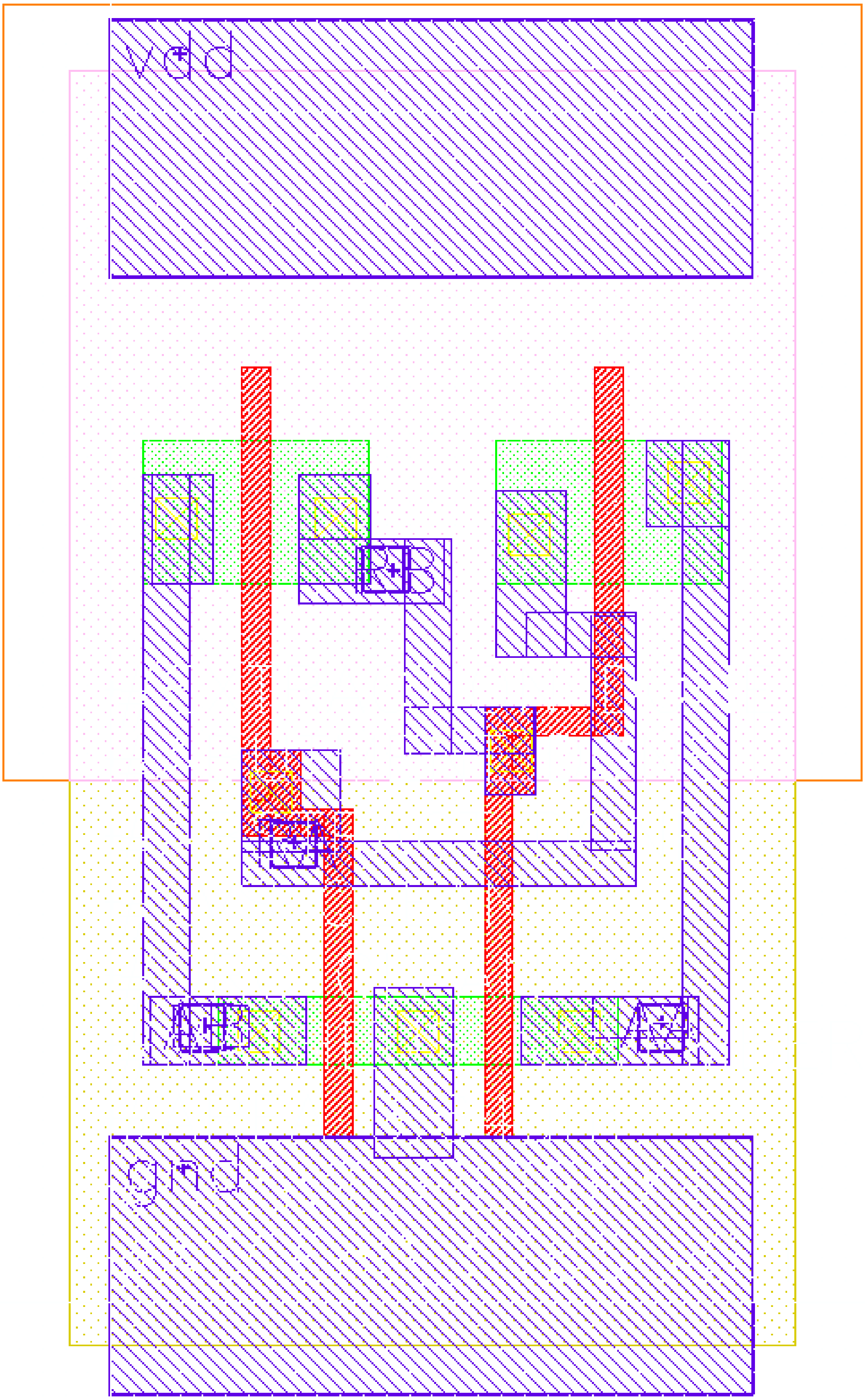


Figure C.23 – Layout of a two input Metastability Filter.

D APPENDIX D

This appendix presents the project of an asynchronous RSA based cryptography circuit. The source files of the circuit described in Balsa language are showed in Figure D.24 – Figure D.27. The synthesized Teak netlists before and after optimization are showed in Figure D.28 and Figure D.29, respectively.

```
import [balsa.types.basic]
procedure mult (
        parameter size_a : cardinal;
        parameter size_b : cardinal;
        input data_in_a : size_a bits;
        input data_in_b : size_b bits;
        output data_out : size_a + size_b bits
) is
variable result : size_a + size_b bits
variable data_a : size_a bits
variable acc : size_a + 1 bits
variable data_b : size_b bits
begin
 loop
  select data_in_a then
   data_a := data_in_a --load a
  end ||
  select data_in_b then
   data_b := data_in_b --load b
  end ;
  result := (0 as size_a + size_b bits) ;--preload result with 0
  for ; n in 1 .. size_b  then --"n" iterarions
   acc := (#result [size_a + size_b-1..size_b] as size_a+1 bits) ;
   if #data_b [n-1] = 1 then --sum
    acc := (acc+data_a as 1 + size_a bits )
   end ;--if
   result := (#result[size_b-1..1] @ #acc as size_a + size_b bits)
  end ;-- for
  data_out <- result --write result
 end
end
```

Figure D.24 – Variable size multiplier Balsa description.

```
import [balsa.types.basic]
procedure mod (
        parameter size_a : cardinal;
        parameter size_b : cardinal;
        input data_in_a : size_a bits;
        input data_in_b : size_b bits;
        output data_out : size_b bits
) is
variable acc : size_b bits
variable data_a : size_a bits
variable data_b : size_b bits
begin
 loop
  select data_in_a then
   data_a := data_in_a --load a
  end ||
  select data_in_b then
   data_b := data_in_b --load b
  end ;
  for ; n in size_a .. size_b+1 then --"n" iterarions4
   if (#data_a[size_a-1..size_b] as size_b bits) >= data_b then
    acc := (((#data_a[size_a-1..size_b] as size_b bits)-data_b) as size_b bits);
    data_a := (#0 @ #data_a[size_b-1..0] @ #acc[size_b-2..0] as size_a bits)
   else
    data_a := (#0 @ #data_a[size_a-2..0] as size_a bits)
   end
  end
  if (#data_a[size_a-1..size_b] as size_b bits) >= data_b then
   acc := (((#data_a[size_a-1..size_b] as size_b bits)-data_b) as size_b bits);
   data_a := (#data_a[size_b-1..0] @ #acc[size_b-1..0] as size_a bits)
  end ;
  data_out <- (#data_a[size_a-1..size_b] as size_b bits)
 end
end
```

Figure D.25 – Variable size division remainder Balsa description.

```
import [balsa.types.basic]
import [mult]
import [mod]
procedure funcao_rsa (
        parameter size_a : cardinal;
        parameter size_e : cardinal;
        parameter size_m : cardinal;
        input data_in_a : size_a bits;
        input data_in_e : size_e bits;
        input data_in_m : size_m bits;
        output data_out : size_m bits
) is
variable data_a : size_a bits
variable data_e : size_e bits
variable data_m, temp : size_m bits
channel mult1_a, mult1_b : size_a bits
channel mult1_o, mod1_a : 2*size_a bits
channel mod1_b, mod1_o : size_m bits
begin
  mult(size_a, size_a, mult1_a, mult1_b, mult1_o) ||--1 mult
  mod(2*size_a, size_m, mod1_a, mod1_b, mod1_o) ||--1 mod
 loop
   data_in_a -> data_a ||--load a
   data_in_e -> data_e ||--load e
   data_in_m -> data_m ;--load m
   temp := (data_a as size_m bits) ;--load temp
   loop while data_e > 1 then --powering
    mult1_a <- (temp as size_a bits); --load a in mult
    mult1_b <- (data_a as size_a bits); --load b in mult
    mult1_o -> mod1_a; --load the result in a in mod
    mod1_b <- data_m; --load b in mod
    mod1_o -> temp; --write partial result in temp
    data_e := (data_e - 1 as size_e bits) --decrement exponent
   end;
   data_out <- (temp as size_m bits)
 end
end
```

Figure D.26 – Variable size asynchronous description of the function implemented by RSA.

```
import [funcao_rsa]
procedure func is funcao_rsa(word_size,word_size,word_size)
procedure async_rsa (
        input data_in : 32;
        input controle_in : 3 bits;
        output data_out : 32
) is
channel func_a, func_e, func_m, func_o : 32 --func
variable rsa_b, rsa_e, rsa_n, rsa_d :  32 --regs
begin
 func(func_a, func_e, func_m, func_o) || --func instance
 loop
  controle_in -> then
   case controle_in of
   0 then --load rsa_b
    data_in -> rsa_b
   | 1 then--load rsa_e
    data_in -> rsa_e
   | 2 then --load rsa_n
    data_in -> rsa_n
   | 3 then --load rsa_d
    data_in -> rsa_d
   | 4 then --decrypt
    func_a <- (rsa_b as word)  ||
    func_e <- (rsa_e as word)  ||
    func_m <- (rsa_n as word) ||
    func_o ->  data_out
   | 5 then --encrypt
    func_a <- (rsa_b as word)  ||
    func_e <- (rsa_d as word)  ||
    func_m <- (rsa_n as word) ||
    func_o ->  data_out
   else
    continue
   end
  end
 end
end
```

Figure D.27 – Balsa description of the asynchronous RSA top circuit.

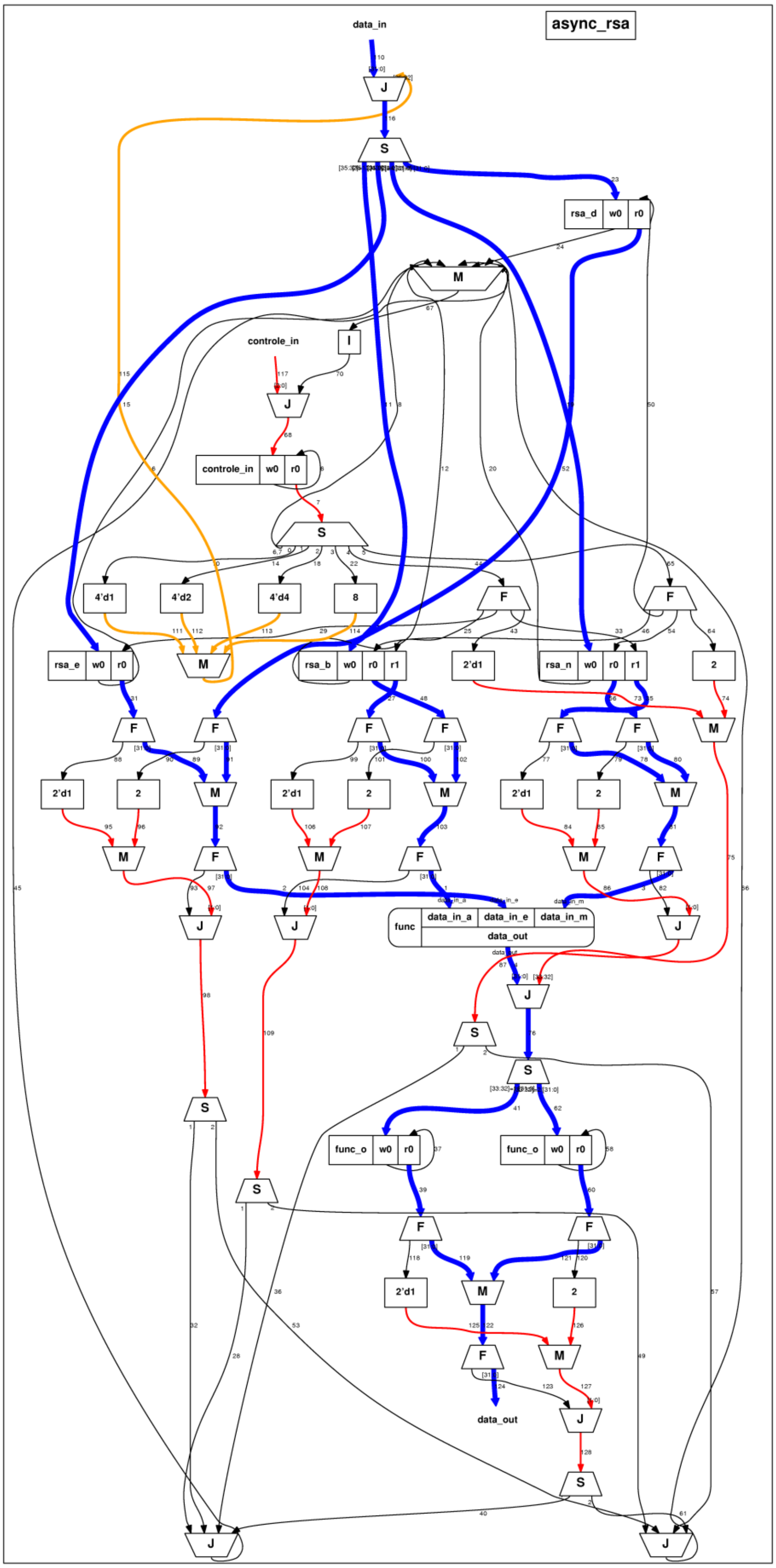


Figure D.28 – Diagram of Teak synthesis before optimization.

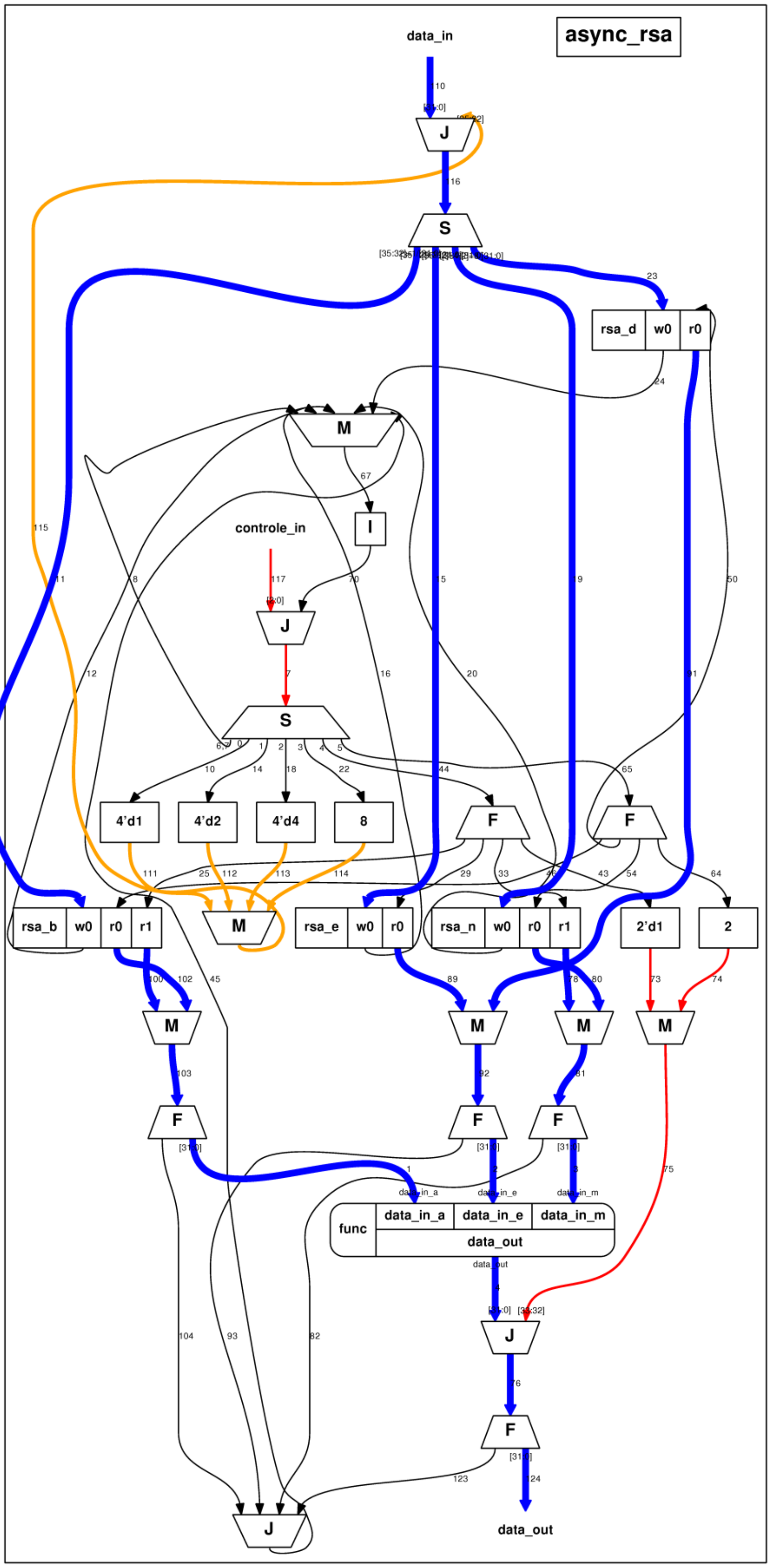


Figure D.29 – Diagram of Teak synthesis after optimization.