PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ASYNCHRONOUS CIRCUITS: INNOVATIONS IN COMPONENTS, CELL LIBRARIES AND DESIGN TEMPLATES

MATHEUS TREVISAN MOREIRA

Dissertation submitted to the Pontifícia Universidade Católica do Rio Grande do Sul in partial fullfillment of the requirements for the degree of PhD in Computer Science.

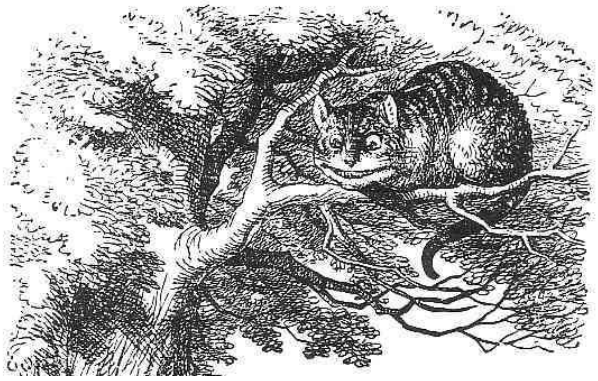
Advisor: Prof. Dr. Ney L. V. Calazans
Co-Advisor: Prof. Dr. Peter A. Beerel (University of Southern California)

Porto Alegre
2016

REPLACE THIS PAGE WITH
THE LIBRARY CATALOG
PAGE

REPLACE THIS PAGE WITH THE COMMITTEE FORMS

To Moreira.



“– Would you tell me, please, which way I ought to go from here?
– That depends a good deal on where you want to get to.
– I don’t much care where.
– Then it doesn’t matter which way you go. ”
(Lewis Carroll, Alice in Wonderland)

ACKNOWLEDGMENTS

Gostaria de agradecer aos meus pais, Alda e Edi, e à minha irmã, Manu, pelo apoio, amor e carinho recebidos em toda minha trajetória até aqui. Obrigado, mãe, por sempre tentar entender o que eram essas tais de “ligações assíncronas”. Amo vocês. À Liana, meu amor e minha companheira, obrigado por sempre estar ao meu lado. Foram muitas noites longas e finais de semana no laboratório dedicados ao desenvolvimento desse trabalho e tu sempre foste compreensiva e me apoiaste. Obrigado por ser essa pessoa que deixa meus dias mais alegres e por me ajudar a seguir crescendo e buscando meus objetivos, sempre de mãos dadas contigo. Te amo. Agradeço também aos meus amigos que sempre me ajudaram a lembrar que existe vida fora do laboratório. Obrigado por todos os momentos divertidos fazendo barulho com nossas bandas e assando uma carne.

Agradeço do fundo do meu coração ao Ney, que sempre confiou no meu trabalho e que, ao longo desses quase 8 anos de orientação, tornou-se, mais que um orientador, um mentor e um amigo. Obrigado por todas oportunidades e pelo teu tempo em todas as reuniões e discussões. Obrigado também pelos passeios por Los Angeles e Santa Monica e por me receber na tua casa junto à tua família. À Karin e às gurias, muito obrigado pelo carinho, pelas discussões filosóficas e pelas risadas. Agradeço também ao Moraes, que me acompanha desde a época do curso de engenharia e sempre esteve disponível para minhas conversas de “1 minutinho”. Muito obrigado por todas as discussões técnicas, pelas oportunidades no GAPH e por sempre motivar meu amor pela microeletrônica.

Ao Marcon, meu padrinho de formatura e mentor, muito obrigado por todas conversas e risadas. Te agradeço pela sinceridade e por me ajudar a buscar meu caminho ao longo do meu desenvolvimento acadêmico, profissional e pessoal. Ah! Obrigado por me ensinar a diferença entre o certo e o justo. Agradeço também ao Fabiano por todas discussões extremamente produtivas. Obrigado por abrir meus olhos para outros lados da pesquisa e por todo apoio e motivação. Obrigado ao Julian por guiar meus primeiros passos na pesquisa e na microeletrônica. Tu foste uma pessoa fundamental na minha vida profissional e és um querido amigo. Te agradeço por todas discussões técnicas, pela orientação e pelas cervejas que bebemos juntos. Ao Edson, cevejeiro de mão cheia, muito obrigado por todas conversas sobre os caminhos da vida e pelas risadas. Obrigado também pelas oportunidades e por tua amizade.

Aos meus amigos do GAPH e do GSE, muito obrigado pelo tempo que passamos juntos e pelos bons momentos. Obrigado em especial ao Sérgio, que tornou-se um grande

amigo, pelos cafés no bar para esquecer um pouco as questões de trabalho e pelas discussões sobre arquitetura e organização de computadores. Te agradeço por sempre tirares todas minhas dúvidas sobre o projeto de sistemas embarcados e por acreditares em mim para formarmos uma equipe de projeto. O chip funcionou! Agradeço também ao Amory, por todas as conversas, técnicas e não técnicas, e pelas cervejas no Cuca Haus. Obrigado pelas oportunidades de colaboração, aprendi muito trabalhando contigo. Ao Moratelli, que me introduziu ao hobby de projetar drones, obrigado pelos momentos divertidos colocando os brinquedos no ar. Obrigado também por todas as substituições na PUCRS. Outro amigo especial é o Felipe, que sempre esteve ao meu lado nos bons e maus momentos. Nossa amizade começou na graduação e seguiu durante nossos cursos de mestrado e doutorado. Muito obrigado por tudo, alemão. Obrigado ao Wachter, ao Madala, ao Castilhos e ao Ruaro pelas partidas de tênis e pelas risadas. Obrigado também aos meus amigos que passaram pelo GAPH e pelo GSE durante esses anos, em especial ao Jerônimo, ao Vini, ao Gnomo, ao Raupp e ao Yan.

Agradeço também ao grupo de async do GAPH por todas discussões produtivas que tivemos e por todos que apoiaram meu trabalho ou desenvolveram pesquisa em conjunto. Aos Hecks pelos trabalhos que desenvolvemos juntos e pelos bons momentos que vivemos dentro e fora do laboratório. Valeu Guilherme pelos churrascos em Los Angeles. Valeu Leandro por todos momentos de descontração e pelo trabalho com o chip, sem tua placa não conseguiríamos validar o projeto. Ao Bruno, valeu pelo apoio ao desenvolvimento do meu trabalho de TCC e de mestrado e pelo bom humor diário. Ao Gibiluka e ao Guazzelli, obrigado por todas as discussões que tivemos e pelo apoio que vocês deram ao desenvolvimento do meu trabalho. Ao Michel e ao Baiano, valeu por todo o apoio com o fluxo e pelo projeto de todos os layouts que vocês fizeram. Ao Walter, ao Jurinha e ao Bortolon, obrigado pelos cafés e pelas risadas. Obrigado ao Kuentzer por sempre me explicar os conceitos de teste e pelas partidas de tênis. Agradeço também aos alunos de Engenharia de Computação que me auxiliaram no desenvolvimento dos trabalhos práticos: Gelmar, Porto, Giovane e Trojan. Agradeço, também, a todos os meus alunos na FACIN. Mesmo sem saber, vocês foram fonte de inspiração em diversos momentos.

Agradeço à FACIN e à PUCRS por todas as oportunidades e pelo apoio que recebi. Sou muito grato pela confiança depositada em mim e, em particular, pela oportunidade de trabalhar com o que eu amo. Agradeço também o apoio financeiro que me foi dado para participar de conferências. Obrigado em especial ao Dotti e ao Bernardo. Agradeço também ao apoio da CAPES, CNPq, FAPERGS e HP, que financiaram parcialmente minha trajetória acadêmica. Em especial, agradeço pelo financiamento do meu estágio sanduíche, que me trouxe um grande desenvolvimento profissional e pessoal. Agradeço também à UNISC e o pessoal de lá, em especial ao Leonel e ao Fred. Leonel, muito obrigado por sempre acreditar em mim e por todo apoio.

Agradeço ao pessoal da UFRGS que trabalhou comigo durante esse curso de doutorado. Obrigado ao Reis e ao Adriel pelo suporte com o ASTRAN. Nossa parceria foi – e está sendo – extremamente produtiva. Ao Ribas e ao André, e seus alunos Mayler e Augusto, meu muito obrigado pelo apoio com o trabalho em síntese NCL. Obrigado por todas as discussões técnicas, em especial ao Ribas. Agradeço também ao pessoal da UFPEL pela recente parceria em trabalhos de projeto de layout. Em especial, obrigado Leomar, Felipe e Smaniotto pelas oportunidades de trabalharmos juntos.

I can't find the words to say how grateful I am to Peter, who became my co-advisor in this work. Thanks for all the technical discussions we had and thanks for the guidance in my research with asynchronous circuits. More importantly, thank you for receiving me in your house and treating me like family. Your support was crucial in my stay in the United States. Thanks for always being so supportive, for all the mentorship and for all the doors you opened in my life. I also thank Janet and the girls for all the fun times we had in Encino. Thanks also go to my friends at USC who received me with open arms. Special thanks to Fei and Dylan, who became good friends. Thank you for all the good times in LA! I also thank my colleagues and friends in the async lab, Ajay, Yang and Ramy, for the technical discussions and fun times.

Finally, thank's to IEEE and its societies, specially Computer Society and Circuits and Systems Society, for the awards and the student grants that helped me attending conferences.

CIRCUITOS ASSÍNCRONOS: INOVAÇÕES EM COMPONENTES, BIBLIOTECAS DE CÉLULAS E TEMPLATES DE PROJETO

RESUMO

O paradigma síncrono foi, por décadas, a principal escolha da indústria para o projeto de circuitos integrados. Infelizmente, com o desenvolvimento da indústria de semicondutores, restrições de projeto relativas à potência de um circuito e incertezas de atrasos aumentaram, dificultando o projeto síncrono. Alguns dos motivos para isso são o aumento na variabilidade dos processos de fabricação de dispositivo, as perdas de desempenho relativas em fios e as incertezas temporais causadas por variabilidades nas condições operacionais de dispositivos. Dessa forma, o paradigma assíncrono surge como uma alternativa, devido à sua robustez contra variações temporais e suporte ao projeto de circuitos de alto desepenho e baixo consumo. Entretanto, grande parte da indústria de ferramentas de automação de projeto eletrônico foi desenvolvida visando o projeto de circuitos síncronos e atualmente o suporte a circuitos assíncronos é consideravelmente limitado. Esta Tese propõe novas técnicas de projeto para otimizar circuitos assíncronos, desde o nível de células ao nível de sistema. Começamos analisando e otimizando componentes básicos para o projeto desses circuitos e depois apresentamos novas soluções para implementá-los no nível de transistores. As otimizações propostas permitem uma melhor exploração dos parâmetros desses circuitos, incluindo potência, atraso e área. Em um segundo momento, exploramos o uso desses componentes como células para a geração de uma biblioteca de suporte ao projeto semi-dedicado de circuitos assíncronos. Nesse contexto, propomos um fluxo completamente automatizado para projetar tais bibliotecas. O fluxo compreende ferramentas de dimensionamento de transistores e caracterização elétrica, desenvolvidas nesta Tese, e uma ferramenta de projeto de leiaute, desenvolvida por um grupo de pesquisa parceiro. Esse trabalho também apresenta uma biblioteca aberta, com centenas de componentes validados extensivamente através de simulações pós-leiaute. Além disso, usando essa bi-

blioteca desenvolvemos novos *templates* para o projeto de circuitos assíncronos no nível de sistema, propondo um fluxo automático para síntese e mapeamento tecnológico. Comparado a uma solução assíncrona no estado da arte, nosso mais novo *template* apresenta uma eficiência energética quase duas vezes maior. As contribuições desta Tese permitiram a construção de uma infraestrutura para o projeto de circuitos assíncronos, abrindo caminho para a exploração do uso de templates assíncronos para solucionar problemas modernos e futuros no projeto de circuitos integrados.

Palavras-Chave: Projeto de components assíncronos, projeto de bibliotecas de células, templates para o projeto de circuitos assíncronos, quasi-delay-insisitive, semi-custom.

ASYNCHRONOUS CIRCUITS: INNOVATIONS IN COMPONENTS, CELL LIBRARIES AND DESIGN TEMPLATES

ABSTRACT

For decades now, the synchronous paradigm has been the major choice of the industry for building integrated circuits. Unfortunately, with the development of semiconductor industry, power budgets got tighter and delay uncertainties increased, making synchronous design a complex task. Some of the reasons behind that are the increase in process variability, the losses in wire performance and the uncertainties in the operating condition of devices. These and other factors significantly impact transistor electrical characteristics, making it more complicated to meet timing closure in synchronous systems and compromising power efficiency. The asynchronous paradigm emerges as an efficient alternative to current design approaches, given its inherent high robustness against delay variations and suitability to low-power and high-performance design. However, while a major segment of the design automation industry was developed to support synchronous design, currently, design automation for asynchronous circuits is limited, to say the least. Furthermore, basic components for semi-custom design approaches, typically available in standard cell libraries were optimized to target synchronous implementations and those necessary to support asynchronous design were also left behind. This Thesis proposes new techniques to optimize asynchronous design, from cell to system level. We start by analyzing and optimizing basic components for asynchronous design and then propose new manners of implementing them at the transistor level. The proposed optimizations and novel components allow better exploring power, delay and area trade-offs, providing a guideline for asynchronous designers. We then explore how to design these components as cells for building a library to support semi-custom design. To that extent, we propose a completely automated flow for designing such libraries. This flow comprises transistors sizing and electrical characterization tools, developed in this Thesis, and a layout generation tool, developed by a fellow research group. We also provide a

freely available library, designed with the flow, with hundreds of components that were extensively validated with post-layout simulations. Using this library we devised new templates for designing asynchronous circuits at the system level, exploring an automated synthesis solution and expanding design space exploration. Compared to a similar state-of-the-art solution, our latest template provides almost twice better energy efficiency and comprises an original automated method for technology mapping and synthesis optimizations. The contributions of this Thesis allowed the construction of an infrastructure for building asynchronous designs, paving the way to explore their usage to solve contemporary and future challenges in integrated circuit design.

Keywords: Asynchronous components design, cell library design, asynchronous circuits design templates, quasi-delay-insensitive, semi-custom.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AFSM – Asynchronous Finite State Machine

ASCEND – Asynchronous Standard Cells Enabling n Designs

ASCEND-A – ASCEnD ASTRAN

BABANOC – Balsa-Based Network-on-Chip

BD – Bundled-data

BVF – Boolean Virtual Function

CDB – Cadence Data Base

CALTECH – California Institute of Technology

CES – Cell Specifier

CHP – Communicating Hardware Processes

CIF – Caltech Intermediate Form

CMOS – Complementary Metal–Oxide–Semiconductor

CSP – Communicating Sequential Processes

DE – Delay Element

DFT – Design for Testability

DI – Delay-Insensitive

DIMS – Delay-Insensitive Minterm Synthesis

DIMXS – Delay-Insensitive Maxterm Synthesis

DPC – Dual Polarity Circuit

DRC – Design Rule Check

DTA – Dynamic Transition Arc

DUV – Design Under Verification

EDA – Electronic Design Automation

EPT – Energy per Transition

FDSOI – Fully Depleted Silicon on Insulator

FET – Field Effect Transistor

FO4 – Fanout-of-4

FPGA – Field Programmable Gate Array

GALS – Globally Asynchronous Locally Synchronous

GAPH – Grupo de Apoio ao Projeto de Hardware

GDSII – Graphic Database System II

GTPS – Giga Transitions per Second

HA – Half Adder

HDL – Hardware Description Language

IC – Integrated Circuit

IEEE – Institute of Electrical and Electronics Engineers

INCL – Inverted Null Convention Logic

INCL+ – Inverted Null Convention Logic Plus

IP – Intellectual Property

ITA – Internal Transition Arc

ITRS – International Technology Roadmap for Semiconductors

LEC – Logic Equivalence Checking

LEF – Library Exchange Format

LICHEN – Library Characterization Environment

LKP – Leakage Power

LVS – Layout Versus Schematic

MF – Metastability Filter

MOPS – Millions of Operations per Second

MOS – Metal–Oxide–Semiconductor

MUTEX – Mutual Exclusion Element

MPSOC – Multi-Processor System-on-Chip

NCL – Null Convention Logic

NCL+ – Null Convention Logic Plus

NOC – Network-on-Chip

NRE – Non-Recurring Engineering

NUG – Negative Unate Gate

PCHB – Precharged Half Buffer

PDK – Process Design Kit

PR – Production Rule

PUG – Positive Unate Gate

PVT – Process Voltage and Temperature

QDI – Quasi-delay-insensitive

ROGEN – Ring Oscillator Generator

RPDFT – Robustly Path Delay Fault Testable

RSA – Rivest-Shamir-Adleman

RTL – Register Transfer Level

RTO – Return-to-One

RTZ – Return-to-Zero

SDC – Synopsys Design Constraints

SDDS – Spatially Distributed Dual Spacer

SDF – Standard Delay Format

SEE – Single Event Effect

SET – Single Event Transient

SEU – Single Event Upset

SOC – System-on-Chip

SOI – Silicon on Insulator

SS – Static State

STA – Static Timing Analysis

STFB – Single-Track Full Buffer

TLF – Threshold Logic Function

UDP – User Defined Primitive

USC – University of Southern California

VD – Validity Detector

VF – Virtual Function

VHDL – Very High Speed Integrated Circuits Hardware Description Language

VL – Virtual Library

VLSI – Very Large Scale Integration

WCHB – Weak-Conditioned Half Buffer

CONTENTS

1. INTRODUCTION

Asynchronous circuits have been the research subject of several research groups since the late 1950s, when they were first proposed. However, these circuits never made it to consumer products in a large scale because semiconductors market evolved to well accommodate synchronous design, but not asynchronous. The reason for that is the reduced complexity that the former allowed in early technology nodes. As technologies evolved and transistors shrunk, though, problems that could be easily waived in synchronous design complicated and new design challenges started to emerge. In this new scenario, interest on asynchronous circuits design regained relevance both in academia and industry. The problem is that, because the market evolved to support the synchronous paradigm, design methodologies, tools and intellectual property (IP) blocks also evolved in a specialized way to optimize synchronous design. Hence, a gap currently exists between the levels of automation for asynchronous circuits and for synchronous ones. The work conducted in the context of this Thesis provides a step towards higher levels of design automation and better design space exploration for asynchronous circuits. This chapter explores the motivation, goals and contributions of this Thesis.

1.1 Preliminaries

Digital circuits are widely spread in our society since their take off in the early 1970s, after the invention of the metal-oxide-semiconductor (MOS) transistor [RCN03]. Evolving from mainframe and minicomputers to smartphones, tablets, wearable devices and medical applications, digital circuit-based appliances are a part of our daily lives. Initially, digital integrated circuits (ICs) were all full-custom, *i.e.* truly handcrafted, where every transistor was manually placed, routed and optimized [RCN03]. However, as silicon technologies evolved, integration capabilities rocketed from thousands to millions, and then to billions, of transistors in a single chip, as predicted by Gordon Moore in the 1960s [Moo65, Moo03]. This was the welcoming the era of very large scale integration (VLSI) design. With this evolution, full-custom approaches proved to be unsustainable and digital designers had to adhere to strategies that were more amenable to automation. As a result, hardware description languages (HDLs) [Vah10] and semi-custom design approaches like the standard-cell methodology were born and quickly spread among VLSI designers [WH10, RCN03, Mic94]. Using HDLs, designers could specify the behavior of a digital circuit in a precise and formal manner using a textual description consisting of expressions, statements and control structures [Vah10]. Moreover, HDLs enabled a technology independent specification flow that allowed reusing code and performing verification steps in the early stages of the design process. The standard-cell methodology provided complementary benefits, as the process

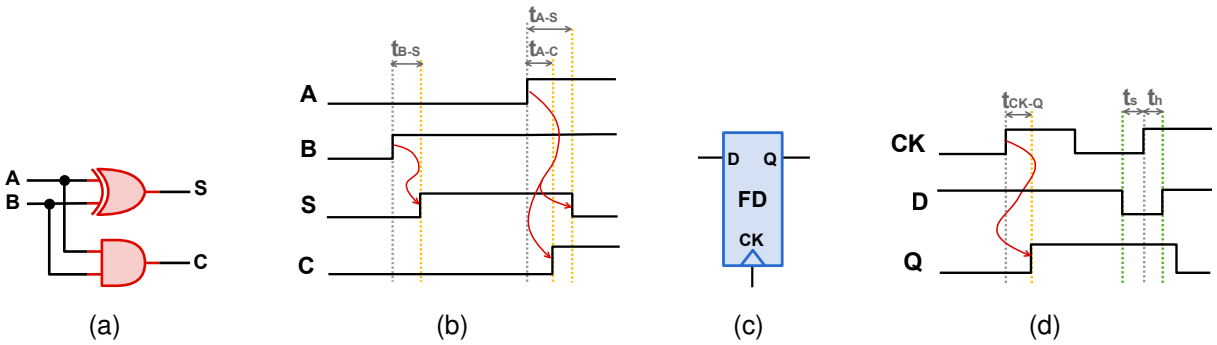
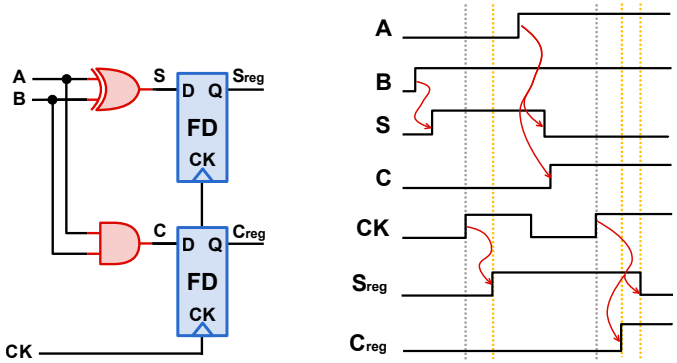


Figure 1.1 – Combinational and sequential circuits: (a) Schematic of a combinational half adder circuit; (b) example waveform for circuit (a) operation; (c) Basic symbol of a flip-flop; (d) example waveform showing the flip-flop behavior.

of implementing a circuit in a given silicon technology was facilitated. Accordingly, it provided designers with libraries of basic circuits already laid out and verified, called cells, which were used to compose complex ICs [Mic94]. In fact, the standard-cell methodology is a natural match for HDLs and is extensively used in contemporary VLSI design as an answer to design automation needs. Moreover, it is often referred as the key success factor for the rapid growth of the VLSI market [ELEHS03, HFO03, JNA00].

Digital circuits can be divided into two main classes, combinational and sequential [WH10, RCN03, Mic94]. Combinational circuits transform data by relating output values to input values using specific Boolean functions. In fact, the outputs of a combinational circuit depend only on the values of its inputs and the function that it implements [WH10, RCN03]. Combinational circuits are found at system and cell level. For example, Figure 1.1(a) shows the schematic of a combinational circuit that implements a half adder using two combinational cells: a two-input XOR and a two-input AND. In this circuit, whenever nodes A and B are at different logic values, node S will be at logic 1. If A and B are at the same logic value, S will be at logic 0. Also, whenever A and B are at logic 1, C will be at logic 1, otherwise it will be at logic 0. Figure 1.1(b) shows a waveform that presents temporal changes in nodes A and B of the example circuit being propagated to nodes S and C, according to the described functionality. Note that these circuits are not ideal: cells and wires that interconnect them will present delays for propagating a transition in the logic value of a given node to another, as Figure 1.1(b) shows.

A sequential circuit, in turn, has memory and its output is not only a function of its inputs, as in combinational logic, but also of its previous states. Consequently, the circuit has the ability to "remember" past events and tracks part of its events history [RCN03]. One of the most common building blocks of sequential circuits is the flip-flop (or 1-bit register), which symbol appears in Figure 1.1(c). As Figure 1.1(d) shows, a flip-flop samples the value on their D input only during transitions of the CK input (in this specific case, a rising transition). It then propagates the value sampled in D to output Q. Note that variations in the value of D will not be captured except after rising transitions in CK. Similarly to combinational circuits, real



(a) (b)

Figure 1.2 – Example sequential logic block: (a) circuit schematic and (b) waveform depicting its behavior.

flip-flops are not ideal and display propagation delays. Moreover, as Figure 1.1(d) shows, these circuits also present two timing constraints: setup (t_s) and hold (t_h) times. The former is the amount of time during which data needs to be stable before the flip-flop can sample its input and the latter is the amount of time it needs to be kept stable after the flip-flop sampling edge occurs. Respecting these constraints enables the circuit to safely capture information in D and propagate it to Q.

The operation of basic sequential cells like flip-flops may look simple, but it enables a powerful assumption in digital design: output Q can only have its value changed at specific transitions on CK. Such characteristic enables adding combinational logic to sequential circuits and allows the design of complex digital circuits, like finite state machines and pipelines, capable of performing a wide variety of tasks [RCN03]. In fact, according to Rabaey et al. [RCN03], most practical circuits require combinational and sequential components for their design. For instance, Figure 1.2(a) shows the schematic of an example sequential circuit using flip-flops and the combinational logic showed in Figure 1.1(a). As the associated waveform in Figure 1.2(b) shows, variations in the values of inputs A and B will immediately affect the internal nodes S and C. However, the outputs S_{reg} and C_{reg} will only have their values switched at rising transitions in CK.

The next natural question is how to generate signal CK, to correctly control the sequencing of events in a sequential circuit. This is an important question, especially because there are often hundreds, or even thousands, of flip-flops in large digital circuits. It is difficult to safely determine when a combinational circuit has ended its computation, because the validity of data in the outputs of such a circuit depends on individual delays of combinational paths, those between every two nodes of a circuit. For instance, in the tiny example circuit of Figure 1.1(a) there are 4 distinct combinational paths: (i) A-S (passing through the XOR gate); (ii) A-C (passing through the AND gate); (iii) B-S (passing through the XOR gate); and (iv) B-C (passing through the AND gate). In this way, a control block for the registers in Figure 1.2(a) would have to be aware of what is the current state of the nodes of the circuit,

and what nodes are transitioning, as well as what is the delay of each combinational path, isolated and in combination, to be able to indicate validity of output data and to control when new data can be injected at the inputs. A concept that grew with semiconductors industry, including IP and electronic design automation (EDA) tools vendors, was to use a single signal to simultaneously control all registers of a circuit. This concept is also known as the *synchronous paradigm* and it serves as basis to enable the definition of HDLs, the construction of EDA tools and the proposal of standard cell libraries, to cite a few of the main characteristics of the available electronic digital design environments. In fact, most commercial VLSI design flows available to date target synchronous design.

In synchronous circuits, a global signal called clock controls the whole sequencing of events, based on the simultaneous sampling of data in all memory elements [RCN03]. This signal triggers memory elements at a fixed frequency, defined at design-time. This conveys to the designer a discrete notion of time. The only requirement is that the time between two sampling actions must be at least equal to (but typically greater than) the worst case delay of all combinational logic blocks between any pair of flip-flops separated only by combinational logic blocks. In fact, in this relies the beauty of synchronous design. Since all memory elements will sample their inputs at the same instant of time and it as long as the design ensures that the clock period covers the worst case delay defined above, the delay of combinational blocks, cells and wires, can be ignored altogether. Figure 1.3 shows a general representation of a synchronous circuit segment. In this circuit, assume the minimum and maximum delays of the combinational logic, *i.e.* the minimum and maximum delays from Q0 to D1, are t_{CLmin} and t_{CLmax}, respectively. Assume also the delays t_{R0min} and t_{R0max} are the minimum and maximum propagation delays of register R0, i.e. the minimum and maximum delays from D0 to Q0. Assume also that the setup and hold time constraints of register R1 are $t_{R1setup}$ and t_{R1hold}, respectively, and that the clock CLK period is t_{CLK}. According to Rabaey et al. [RCN03], under ideal conditions, where the clock edges occur in both registers at the same time, i.e. CLK0=CLK1, the correct functionality of such a circuit is guaranteed provided that:

$$t_{CLK} - t_{R0max} - t_{CLmax} \geq t_{R1setup} \tag{1.1}$$

and

$$t_{R0min} + t_{CLmin} \geq t_{R1hold} \tag{1.2}$$

In other words, Equations 1.1 and 1.2 ensure that, respectively, the setup and the hold constraints of registers are respected.

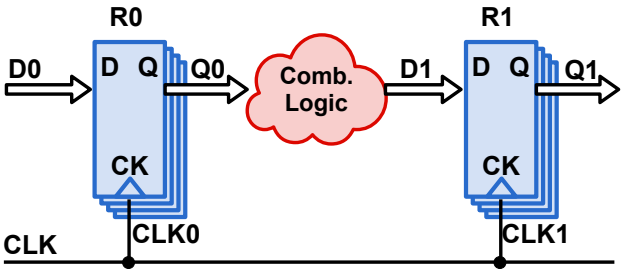


Figure 1.3 – Example of a typical segment of a synchronous datapath.

1.2 Motivation

Unfortunately, clock signals are never ideal. In fact, in a real circuit, transitions at the points CLK0 and CLK1 in Figure 1.3 would usually lag in different proportions with regard to the reference clock signal (CLK). This is due to spatial variations in the arrival time of the clock signals, caused by static mismatches in the clock paths and differences in their loads. This phenomenon is classically called *clock skew*, and is often constant from cycle to cycle [RCN03]. In other words, if the active edge of CLK1 (the edge that triggers the register) is delayed by a time δ, then on the next cycle, it will be delayed by the same amount of time. Note that clock skew modifies the analysis of the example circuit of Figure 1.3 because registers R0 and R1 are no longer activated at the same instant of time. This requires adjustments in Equations 1.1 and 1.2, as discussed in detail in [RCN03]. To guarantee the correct operation of the circuit, one must take into account the value of δ and modify equations to account for it, which gives:

$$t_{CLK} - t_{R0max} - t_{CLmax} + \delta \geq t_{R1setup} \tag{1.3}$$

and

$$t_{R0min} + t_{CLmin} - \delta \geq t_{R1hold} \tag{1.4}$$

Equation 1.3 indicates that bigger δ values have the potential to improve the performance of the circuit, since larger maximum propagation delays t_{R0max} and t_{CLmax} can be tolerated for a same clock period t_{CLK}. In fact, this is correct, but increasing the skew has the side effect of making it more difficult to meet hold constraints. As Equation 1.4 shows, the bigger the value of δ, the larger the delays required for the combinational logic path to meet the equation. Also, δ can be a negative value, which relaxes hold constraints but tightens setup constraints. Dealing with those problems was classically a low price to pay for the advantages of the synchronous paradigm. However, as reported by several authors (check [SGY+09, SRG+01, CPR10, TVC10, EKD+03, KKFK13]), this price is considerably

higher in modern technologies. Accordingly, the correct and efficient distribution of a global clock signal is becoming an increasingly expensive task, and the margins required for the clock signal to deal with delay uncertainties are getting prohibitively constraining. Some of the reasons behind that are the increase in process variability [KGB+11], the losses in wire performance [HMH01, HGD04, Cit04] and the uncertainties in the operating condition of devices [EKD+03, KKFK13]. These and other factors significantly impact transistor electrical characteristics, and make it more complicated to meet timing closure in synchronous systems. In fact, according to the International Technology Roadmap for Semiconductors (ITRS) in its 2011 edition [ITR], a shift in IC design paradigms seems to be inevitable. This is why asynchronous design techniques are receiving increasing attention of the VLSI research community.

A digital circuit is called asynchronous when no clock signal is used to control any sequencing of events in its registers. Instead, asynchronous circuits rely on a handshaking protocol for each pair of registers to locally communicate for operation and synchronization purposes [Mye01, SF01a, BOF10]. This means that each pair of registers communicates by explicitly signaling, sending and receiving data. As the resulting behavior, registers are active only when and where needed, and combinational blocks need not to be constrained by a single global maximum delay value. Such characteristics present several advantages over the synchronous paradigm and can alleviate problems faced by designers in modern technologies, as discussed in different works, such as [SRG+01, MN06, BNS+07, BOF10, CPR10, NS11, CCGC13, NS15a, NS15b]. Accordingly, asynchronous circuits have been mainly used for achieving:

High speed operation: They can operate with average case delays, while their synchronous counterparts rely on worst case delays. This is because in asynchronous design, registers synchronize locally and logic blocks can be individually optimized. In synchronous systems, all registers are controlled by a single clock signal and combinational blocks are all constrained to a common maximum delay constraint. Also, delay uncertainty is a major problem in modern silicon technologies and are caused by three main sources: process, voltage and temperature (PVT) variations [Nas00]. Delay variations caused by process are typically constant within a chip, but vary within a wafer lot. Voltage and temperature, on the other hand, depend on environmental and operating conditions and cause dynamic delay variations during the operation of the circuit. To account for PVT variations, synchronous designs need safety delay margins on the clock signal that may result in substantial performance losses [EKD+03, DTP09, KKFK13]. This is particularly problematic because such margins affect the global maximum delay constraint, constraining all combinational blocks. Asynchronous circuits, on the other hand can accommodate delay variations more easily [MN06, CPR10], leading to better performance in the presence of such variations. Examples of commercial high speed

asynchronous solutions are the Ethernet switch designed by Intel [Int], presented in [DLD+14], and the FPGAs designed by Achronix [Ach, RRP+09].

Low power operation: In order to account for PVT variations, synchronous designers need safety margins not only on the period of the clock signal, but also in the supply voltage source [EKD+03, DTP09, KKFK13]. Because static power is directly related to the supply voltage and switching power increases quadractically with supply voltage [RCN03, WH10], these margins can have a substantial impact on the total power of a synchronous system. Furthermore, when not computing, an asynchronous circuit is naturally quiescent, consuming only leakage current. This is different from synchronous circuits, which periodically have their registers activated by the clock signal, even when expecting no computation. It is true that the idle power in a synchronous circuit can be reduced by using clock gating techniques. However, even in that case, such techniques can complicate clock distribution and compromise power figures by incurring in extra static power [Wim14]. In fact, clock distribution circuitry typically accounts for 45% of the total power of a modern processor [AFE+05]. Another important point is that, due to their nature, asynchronous circuits cope better with delay uncertainties and are better candidates to voltage scaling techniques [MN06], which can provide substantial power savings, as discussed by Chang et al. in [CPR10].

The above items are the two main reasons why the VLSI research community is regaining interest on asynchronous design. They demonstrate the potentials of this technology to cope with contemporary and future design challenges and are the main reason why we decided to work with asynchronous circuits design. Another advantage of using the asynchronous paradigm is its high modularity [Mye01, SF01a, BOF10]. In fact, because asynchronous circuits do not rely on a discrete notion of time, and rather synchronize and communicate using local handshakes, they have a modular interface and can be more easily used to compose a system-on-chip (SoC) design. In other words, because an asynchronous IP block is self-timed, one can more easily place multiple communicating asynchronous IP blocks in a single design. This is in contrast to synchronous solutions, which require intricate clock distribution and synchronization strategies, like the one discussed in [RAB+12]. Such characteristic is particularly appealing for contemporary Multi-Processor SoC (MPSoC) applications, as it enables building such systems in a more “plug-and-play” manner. Another beneficial aspect of asynchronous design is that asynchronous circuits can cope better with delay variations and continue to operate correctly along a larger range of PVT variations [MN06]. For instance, in [BNS+07], Bouesse et al. discuss how to use asynchronous design to increase robustness to power supply disturbances. Furthermore, as discussed in [PCV12, SCM+11], asynchronous design can also be used for hardening other circuit aspects, like robustness to single event effects and robustness to side channel attacks in cryptographic applications. However, despite the listed potential benefits, the asynchronous paradigm has an Achilles heel, which prevented its wider adoption: an acute

lack of EDA support. Such lack of EDA support provides a large research field and is the fundamental motivation for the work presented in this document.

1.3 Problem Description

As previously explained, a major segment of the EDA industry was developed to support standard cell synchronous design. In this way, methodologies for clock generation, control, distribution and closure matured while the development of asynchronous design methodologies lagged behind [BOF10]. For instance, major EDA vendors like Cadence [Cad] and Synopsys [Syn] have automated support for clock tree and clock gating circuitry generation, test mechanisms insertion and even complicated optimizations like retiming, time borrowing and useful skew [Fri01, XD96]. All of these target synchronous design, and have no specific support for asynchronous circuits. Furthermore, basic components for semi-custom approaches, designed at the cell level and available in standard cell libraries were optimized to target synchronous implementations and those necessary for asynchronous design were also left behind [BOF10]. For example, IP vendors like ARM [ARM] provide standard cell libraries with combinational and sequential cells for synchronous design exclusively, together with libraries of cells creayed specifically for clock distribution trees. However, there is no IP vendor providing cells required for asynchronous design, which are often different from those employed in synchronous circuits [BOF10]. Even HDLs were devised and revised targeting synchronous circuits design. Note that these problems are aggravated by two facts:

1. Differently from synchronous, asynchronous circuits can be designed using multiple different templates, each with its own advantages and drawbacks. Such characteristic allow better design space for asynchronous circuits but complicates the design of EDA tools and cell libraries to support the different templates and benefit from their advantages.

2. Standard-cell libraries and tools for synchronous design are available for decades and were employed in a vast set of designs by the VLSI industry. This means that all this environment was extensively verified and validated in commercial applications. In this way, shifting to new tools and libraries to target asynchronous design is risky, because these were not extensively validated and engineers are not used to them.

In fact, different authors claim that the main reason why asynchronous design is not widely employed in VLSI design is the lack of EDA support for its design [CCKT09, PAAS14, JN08, KL02, BDL11, SSJ13, BLDK06, SRG+01, KOM13, AFE+05, GBN13, SGY+09]. In this way, there are multiple gaps to fulfill to provide EDA support for asynchronous design, and

multiple problems to be solved. Our target in this Thesis is to provide better EDA support to asynchronous designers in cell libraries and templates design.

1.4 Goals

Given the exposed motivation and problem description, the strategic goal is to propose a set of contributions that enhance the available infrastructure for the design and implementation of asynchronous circuits.

To accomplish this strategic goal, the following set of specific objectives were defined:

1. To systematize the process of composing cell libraries for asynchronous design;
2. To define a cell design flow that supports the development of cell libraries for asynchronous design;
3. To enable semi-custom asynchronous design using both conventional cell libraries and libraries of cells specifically designed for asynchronous circuits;
4. To expand the asynchronous circuit design space, providing new techniques and resources at the logical and circuit levels that complement, enhance or optimize existing semi-custom asynchronous design techniques.

1.5 Originality of this Thesis

This work has its roots in the proposition of a basic library of cells for asynchronous design called Asynchronous Standard Cells Enabling n Designs (ASCEnD). The library was initially designed as a final year undergraduate project carried out by the Author. The initial targeting technology was STMicroelectronics 65nm bulk CMOS technology. This library, called ASCEnD-ST65 [Mor10], then served the purpose of semi-custom design, where it was coupled to academic tools, during the Master of Science developments of the Author [Mor12]. Since its proposition, the library was employed in the design of different application circuits, such as Network-on-Chip (NoC) routers, as discussed in [PMMC10, PMMC11, MMG+13, GMMC15]. In the context of this Thesis, ASCEnD-ST65 was initially employed to explore semi-custom asynchronous design approaches and, during this process, a need for new components was detected. Hence, a first novel contribution of this work is the enrichment of the library with new components. To do so, a set of guidelines were specified to ensure that the library was compatible with conventional cell libraries. This avoids the need

for designing cells that are already available in such libraries and helps meeting specific objectives 1 and 3 from Section 1.4.

The design of a cell library is a laborious task and requires EDA support. Fortunately, a basic design flow for the ASCEnD-ST65 library was also available since its proposition in [Mor10], as described in [MOPC11]. The problem is that the original ASCEnD flow had low degrees of automation. Albeit it had the capability of automatically dimensioning transistors, it had no support for automatic layout generation or cell characterization. In this way, cell design still required a large design effort. The second original contribution of this Thesis, which helped overcoming that limitation, was the design of a new tool for automatic electrical characterization of cells. The tool was called Library Characterization Environment (LiChEn) [MOCO13]. All cells of the library have passed through LiChEn.

The next step was to provide a solution for automatic cell layout generation. To do so, the author integrated the ASTRAN [ZR14] tool in the ASCEnD flow [ZRM+14a, ZRM+14b]. Thus another contribution of this Thesis was the design of a set of scripts that enabled a smooth integration of LiChEn and ASTRAN in the ASCEnD flow, automating the process. This new version of the flow was called ASCEnD-A. Its use allowed to produces larger set of cells, as reference [MAZ+14b] discusses. In fact, all cells of the library developed here were designed (in some cases re-designed) using this flow, which enabled meeting the specific objective 2 from Section 1.4.

The framework for automatic cell generation above enables a multitude of experiments in producing cells for asynchronous design. As a result, a variety of original contributions were produced. Significant advances regard transistor level optimizations of asynchronous cells (cells used for asynchronous circuits design), as references [MMC14, MOMC12, MAMC15, MHBC15, HHS+15, SMT+15, ZHM+15] discuss. Disturbances in the electrical behavior of the cells caused by phenomena like charge sharing, PVT variations and radiation effects were also studied, as discussed in [MOMC13, GHMC14, MAGC14, MC13], which enabled defining guidelines for cell libraries design considering such effects. Moreover, guidelines for designing cells for voltage scaling applications were also defined, as presented in [MC13, MAGC14]. Most of these works propose optimizations to existing cells for asynchronous design, as well as suggest new cells, and helped meeting specific objectives 1 and 4 from Section 1.4.

As ASCEnD gets enriched with new components, specific objective 4 can be further pursued, by exploring optimizations in asynchronous logic at the architectural level. This Thesis brought several contributions in this context as well. These include the proposition of new asynchronous templates, as presented in [MGC12b, MGC12a, MGHC14, MOPC13b, MPC14, MTMC14, HMH+15, HHC+15], and a new technique for automatically mapping asynchronous designs to cells available in the ASCEnD library [MNM+14].

Given this set of contributions, this Thesis provides a next step towards better EDA support for asynchronous design. Its originality can be divided in the following items:

- *A set of novel components for asynchronous circuits design:* As we explored asynchronous components with the ASCEnD flow, we optimized some of them and proposed new ones. This is a step forward in the support of asynchronous circuits design because it provides more options to the designer. Furthermore, as this Thesis explore, the proposed optimizations and novel components allow better exploring power, delay and area trade offs.

- *A completely automated design flow for devising cells for asynchronous design and a case study cell library:* The ASCEnD-A flow is one of the most important contributions of this Thesis, because it allows a completely automated approach for designing components for building asynchronous circuits. As this document describes, we validated the flow with extensive components design as we explored different manners of implementing asynchronous circuits. We also provide a freely available library designed with the flow, with the only restriction being that the requester has access to the target technology. This library supports the implementation of asynchronous circuits targeting different templates, as this Thesis explores.

- *New templates that allow better design space exploration and higher degrees of automation for logic optimizations and technology mapping:* The proposition of new templates for asynchronous design brought new possibilities for asynchronous designers enabling better design space exploration. Moreover, the development of an automated method for mapping asynchronous cells in our target template and performing logic optimizations is an unforeseen possibility for this type of circuits.

1.6 Document Structure

The rest of this Thesis is organized in 5 Chapters. Chapter 2 presents basic concepts on semi-custom design approaches and asynchronous circuits. This chapter also defines a basic terminology that is used throughout this document. Next, Chapter 3 explores our innovations in asynchronous components design. Chapter 4 explores the design of asynchronous cell libraries, presenting our automated design flow and a case study library. Chapter 5 presents a set of new design templates proposed in this work and explores their usage in a set of case study. Lastly, Chapter 6 presents the final considerations of this work and discusses future work. Note that each chapter of this Thesis presents a revision of the state-of-the-art relative to its content.

2. CELL-BASED AND ASYNCHRONOUS CIRCUITS DESIGN

This Chapter explores concepts that serve as basis for the work developed in this Thesis. It starts bringing an overview of design styles for digital ICs in Section 2.1. The focus here is on cell-based design, exploring well defined concepts that are widely employed in contemporary VLSI circuits. Next, it discusses the asynchronous paradigm in Section 2.2, focusing on asynchronous digital IC design, and scrutinizes different approaches to design an asynchronous circuit. Alongside the discussion, the Chapter establishes a precise terminology employed throughout the rest of the document.

2.1 Cell-Based Design

VLSI applications have different requirements, mainly high performance, low power and high density. Applications such as supercomputing prioritize high performance, while battery-based applications have tighter power and area constraints and will trade off performance, area and power, which typically conflict in IC design [RCN03]. These applications are also usually constrained by two major factors: (i) design and fabrication cost; and (ii) time-to-market issues. Unfortunately optimizing ICs for performance, power and area increases these factors and designers need to trade off the optimization process with design time and cost. To alleviate the problem, and to meet different requirements, through the development of the semiconductor market, different design methods, or design styles, appeared to devise an IC. This section explores some of these methods, focusing on the cell-based semi-custom design style.

2.1.1 IC Design Styles

Contemporary IC design styles trade off time to market, non-recurring engineering (NRE) costs, production cost per part and performance, area and energy efficiency [WH10, RCN03, Mic94]. As Figure 2.1 shows, these design styles are often classified into full-custom and semi-custom approaches [RCN03]. In the former, all the aspects of an IC are handcrafted and the application is literally manually designed, from functional to physical level. This means that every transistor is individually sized and laid out, involving detailed manipulation of physical and electrical characteristics of these devices. A good example of a full-custom IC is the Intel 4004 microprocessor, which layout is showed in Figure 2.2. This IC was designed in the early 70s and counted with roughly 2,300 transistors, all manually placed. As a result, the layout is compact and optimized through all levels, from transis-

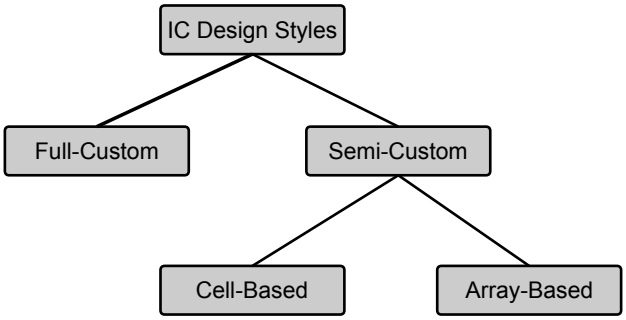


Figure 2.1 – A taxonomy for IC design styles, adapted from [RCN03].

tors placement to nets routing. Such approach leads to designs optimized to excellence. However, the drawback is that doing so implies in a labor-intensive large engineering effort, incurring in larger NRE costs and design time. According to Rabaey et al. [RCN03], a full-custom approach is only justified under the following reasons:

- The (full-)custom block can be reused many times, as in IP designs such as cells of a cell library;
- The design cost can be amortized over a large volume of fabricated parts, as is the case with microprocessors and memories;
- Cost and design time are not the prime design criteria, as in supercomputers.

The increasing complexity of contemporary ICs, added to strict time-to-market constraints has confined full-custom design techniques to specific cases, where the extremely costly effort pays off. To overcome full-custom limitations, a wide variety of semi-custom approaches have been proposed in the last decades, aiming to reduce design time by partially automating the process of designing an IC. However, such automation comes at the price of reduced density and power efficiency, as well as losses in performance figures. Generically speaking, the shorter the design time, the larger the penalty in area, power and performance. As Figure 2.1 shows, semi-custom design is subdivided in two main families: cell-based and array-based. Array-based approaches encompass prediffused and prewired arrays that can even completely avoid the designer to get involved in the fabrication process, as in the case of field programmable gate arrays (FPGAs) [WH10, RCN03, Mic94]. Accordingly, these approaches have the advantage of having reduced NRE costs and are attractive for applications that will require a relatively small volume of parts sold. Obviously, a trade off exists and array-based approaches may incur in severe penalties in performance, density and power efficiency. Furthermore, they mostly support synchronous design, and vendors of array-based solutions do not support asynchronous circuits design easily. For example, the only vendor that does implement asynchronous logic for FPGAs currently is Achronix [Ach]. However, their solution starts from a synchronous description of the circuit. Also, it is not

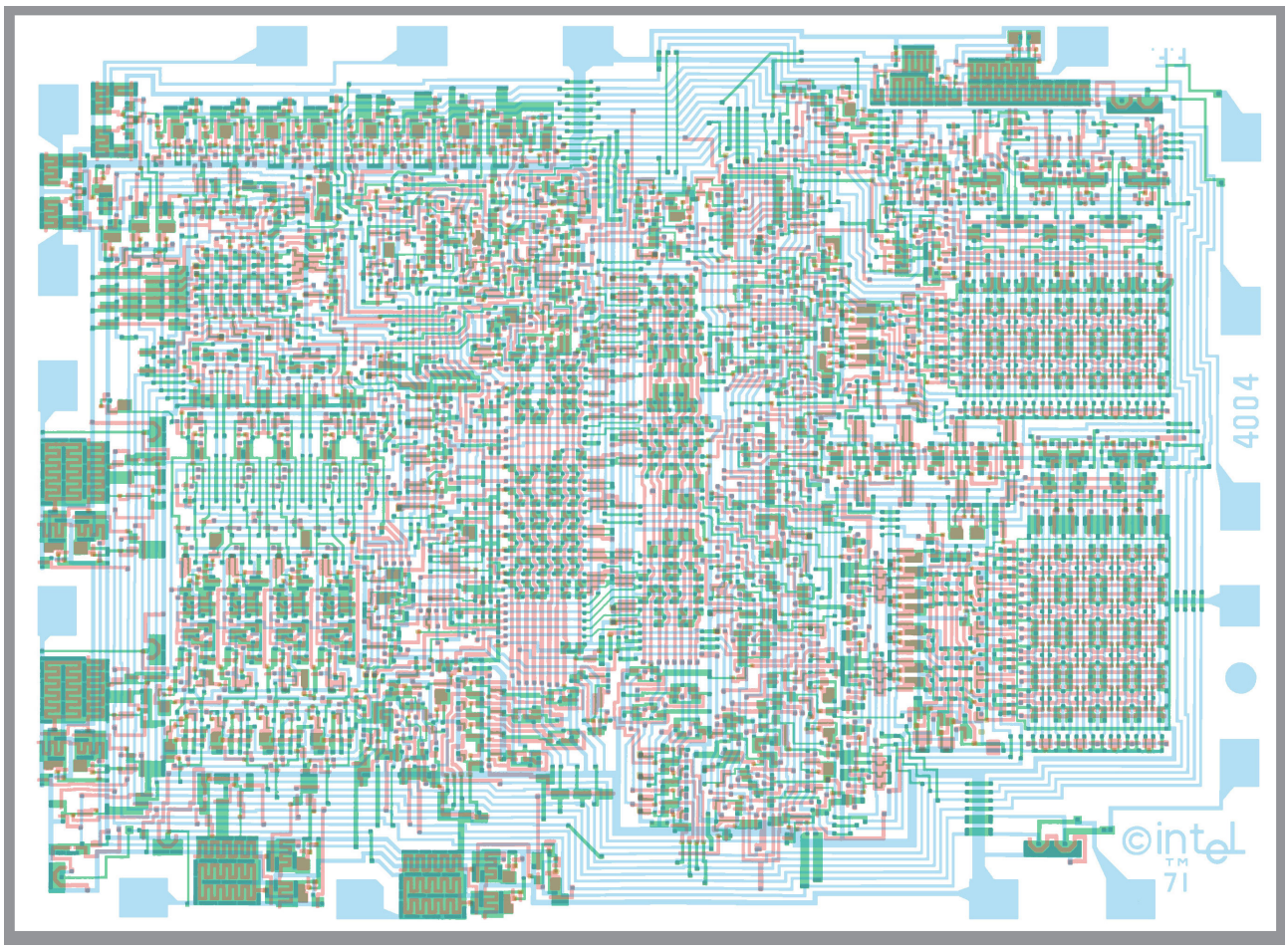


Figure 2.2 – Layout of the Intel 4004 microprocessor [Int].

clear how much the asynchronous logic can be optimized using array-based approaches. Given all these restrictions, array-based design is kept out of the scope of this Thesis.

Cell-based design assumes the use of pre-designed and pre-verified basic blocks, called *cells*, to compose an IC. Cells are designed at the layout level and implement basic logic functions, including combinational and sequential logic. Note that cells are organized in libraries, which can contain hundreds or thousands of them and the quality of a semi-custom design is a direct function of the quality of the library used to compose it. The idea behind the cell-based approach is that a design is captured as a schematic composed only by the cells available in a specific library and the layout of this design is then constructed using these cells. In other words, this design style increases the grain size of design to logic gates, in contrast to designing each transistor, as in a full-custom style. For instance, Figure 2.3 shows an example of a contemporary design, the Intel Ivy Bridge 3C. With over 1 billion transistors, it is simply impossible to design such an IC using a full-custom design approach and respect time to market and cost constraints. In fact, as the regularity in the layout of this design indicates, virtually all portions are designed using semi-custom approaches. Only the the most critical modules, such as phase-locked loops, are designed manually [WH10].

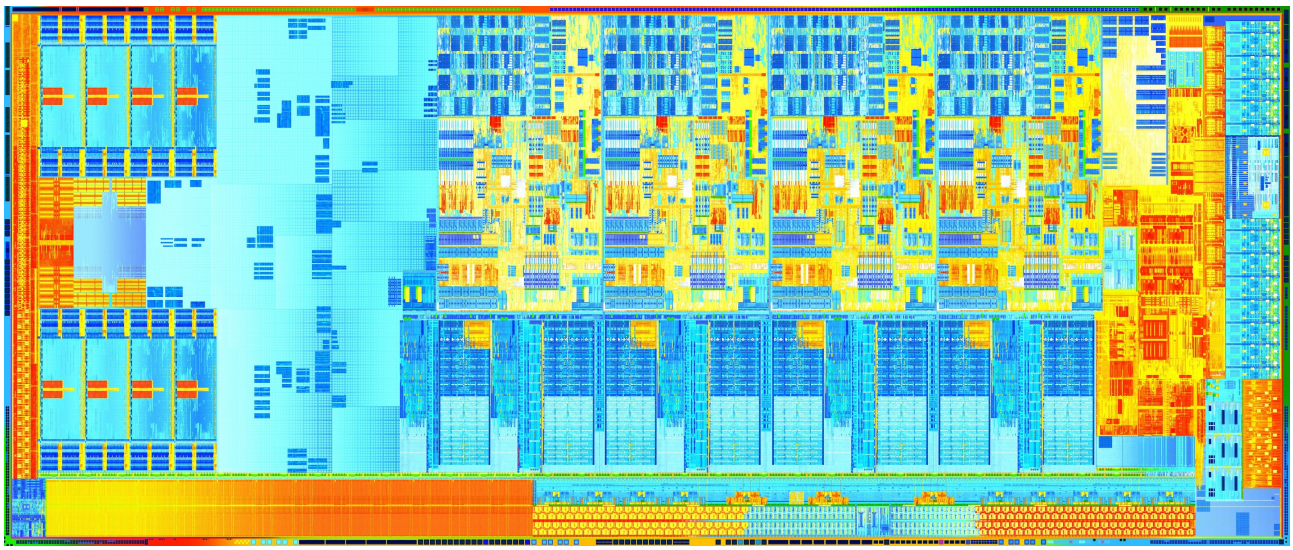


Figure 2.3 – Layout of the Intel Ivy Bridge 3C [Int].

2.1.2 Cell-Based Design Flow

A cell-based design starts with three key definitions: (i) the specification of the circuit to be designed; (ii) the target technology; and (iii) the cell libraries to use. Definition (i) entails describing the functionality and the constraints the circuit need to respect, typically in a document written in human language (which can include algorithm descriptions). This document specifies how the circuit is to behave from a functional point of view, depicting the expected output values for each set of input values and defining temporal characteristics. It also defines performance figures and physical and electrical constraints that the designer must use to guide the implementation of the circuit. Performance figures specify the speed at which the circuit will operate, measured in metrics such as throughput, clock frequency and latency. They also tell how much power the circuit should/could dissipate, which can be given as total power figures or as more specific definitions, like energy per operation.

Definition (ii) specifies the target technology and is critical. One reason for that is because it chooses the fabrication technology, which is the fundamental factor for deriving the performance figures of devices used in the design [Tay13]. Although most designs still target bulk complementary metal-oxide-semicoductor (CMOS) processes, there is a growing market for silicon on insulator (SOI) and multigate devices (like Intel's FinFETs). Moreover, some authors describe advantages of using other fabrication technologies, like GaAs [Har07]. However, these are typically not widely supported by semiconductor foundries. Another important aspect of definition (ii) is the choice for a target technology node and foundry. This is particularly important for cell-based design because it can limit definition (iii), given that IP vendors can target specific technology/library vendor combinations. In other words, definition (ii) must be carried on bearing in mind which cell libraries will be used in the design and what vendors support the fabrication technology.

While it is true that some design houses can have cell libraries designed on demand, not all companies share this privilege, as such libraries require full-custom design and can thus be very expensive to develop. In fact, in cell-based design it is more common to use third-party cell libraries that contain sets of cells optimized for different aspects, targeting a specific technology. Such aspects include high speed, low power, high density and high robustness, which the designer needs to trade off to meet the defined constraints. Furthermore, there are even libraries especially designed for projecting specific parts of a digital IC, like clock distribution circuitry or test infrastructures. Recall that defining the target technology is strictly related to what libraries will be available, that is why definitions (ii) and (iii) are typically carried out together. Also, albeit this discussion is limited to cell libraries, cell-based designers can also employ third-party IP macro blocks [RCN03, Mic94] for regular circuits like memories, register files and arithmetic units.

Once the circuit is specified and the target technology and cell libraries are defined, it is possible to start the circuit design process. One of the advantages of cell-based IC design is that it counts with extensive support of EDA tools, which can automate most parts of the design process. To employ these tools on the design of an IC, designers rely on flows like the one depicted in Figure 2.4. Note that this flow is organized to highlight issues related to cell-based design. The first step is the design of the HDL [Vah10] source code, which must describe the behavior of the circuit and fulfill its specification. One of the reasons for the popularity of HDL in IC design is the fact that it allows describing the functionality of a digital circuit using a textual description. In this way, designers can write expressions, statements and control structures in a language that is similar to human language. Furthermore, HDLs are technology independent and allow performing verification steps in early stages of the design flow. They also enable higher degrees of modularity and reuse, besides largely allowing technology independent design. In fact, because HDLs are typically employed, the target technology and cell libraries are not required in the first stages of cell-based design flows, as Figure 2.4 shows, and the decision on these can be postponed to latter design stages.

HDL design capture most often use the Verilog or/and the VHDL language [RCN03, Vah10], albeit other languages, like SystemVerilog have also been employed [BOF10]. To verify the correct functionality of an HDL description, and to verify the design before proceeding to synthesis, designers typically rely on testbenches (which are also often built using HDLs), and functional simulation of these. Testbenches reproduce the behavior of the environment where the design will be employed, providing realistic stimuli and collecting the resultant data generated by the design under verification (DUV). Functional simulation allows verifying the behavior of the designed circuit regardless of most implementation details. In other words, it assumes that the DUV is a module described using some HDL and verifies the functionality of this module as a set of input-output relations [RCN03]. Furthermore, at this stage, it is usually assumed that there is no delay in the components of the DUV, al-

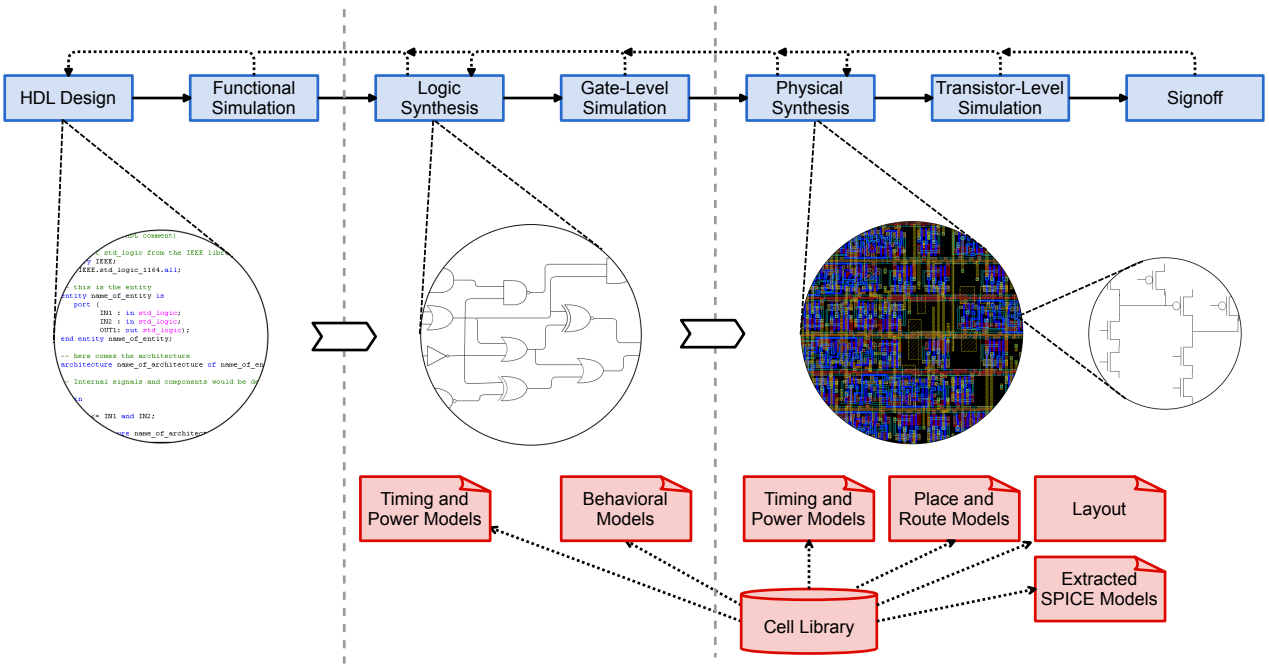


Figure 2.4 – Basic cell-based IC design flow.

beit HDLs typically allow designers to specify input-output delays. The reason for using a zero-delay model is mainly to speed-up simulation time. If during functional simulation the designer detects a misbehavior of the DUV, *i.e.* its functionality differs from that defined in the circuit specification, the HDL code needs to be revised so that the design performs as expected. Popular functional simulators are Cadence NC-Sim [Cad] and Mentor Graphics Modelsim [Men].

Once the design is validated through functional simulation, it proceeds to the logic synthesis stage. At this stage, the hardware description will be synthesized to generate a netlist, a set of cells interconnected by wires. It is important to highlight, though, that this is a very complex task, particularly for modern designs that can take up thousands or millions of cell instances. Hence, designers typically rely on the support of EDA tools, in particular logic synthesis tools that, said simply, translate HDL code to a gate netlist. The first step of a synthesis tool is usually a generic synthesis, and consists in the process of translating an HDL code to a generic set of modules, usually technology independent blocks like registers and combinational gates like ANDs, ORs and XORs. After the generic synthesis, the next step is to perform a step called technology mapping. This step generates a netlist that implements the same functionality as the HDL source code, but formed exclusively by instances of cells available in the target cell library. To perform this task, as Figure 2.4 depicts, these tools require models that describe the electrical characteristics of the target cells (commonly called timing and power models). These models describe the timing and power characteristics of each single library cell, which are used to compute the timing and power characteristics of the design using methods like static timing analysis (STA) [RCN03]. Timing and power mod-

els are typically described in the OpenSource Liberty Format [Lib] proposed by Synopsys [Syn].

STA is a method for computing the expected delay of paths in a circuit without relying on its simulation. It is widely employed in EDA Frameworks for IC design. STA tools statically analyze the delay of the logic paths, which consist of sequences of cells connected in series, using the provided timing and power models. Using STA synthesis tools are able to optimize the circuit with cells that are more *suited* to each logic path in some sense. Also, note that the tool thus requires the definition of a set of constraints that states the conditions under which the final design is to operate. These constraints guide the process of selecting cells and are provided by the designer, based on the initial circuit specification. The most common format for describing constraints that guide a synthesis tool is the Synopsys Design Constraints (SDC), widely supported by IC design frameworks.

Because different approaches can be used for optimizing a design, like employing wider or narrower logic paths or using larger or smaller cells, the process of technology mapping is iterative and can require several rounds [Mic94]. At each round, the designer will need to adjust the constraint settings to help the synthesis tool meeting the specified constraints using the defined cell libraries. It is also during logic synthesis that test circuitry is added to the design, which is usually done using automated tools like Cadence Encounter DFT Architect [Cad]. This work does not address test circuits in detail, a good reference to the interested reader is [ABF94]. Once the circuit is synthesized, mapped to the target cell library and optimized to have all its constraints met, it is verified using a logic equivalence checking (LEC) tool. This tool enables a formal verification of the circuit, ensuring that the generated netlist implements the same logic described in the HDL code. In other words it ensures that the process of mapping the circuit to a target cell library did not corrupt the functionality specified by the designer. The most used frameworks for logic synthesis nowadays are Cadence Encounter [Cad] and Synopsys Design Compiler [Syn]. Note that these frameworks encompass a set of tools to perform the various tasks described here. Also, during the logic synthesis process, issues that were not detected during functional simulation may be identified, requiring a revision of the HDL source code. These issues are typically related to the structure of the described hardware and the available components in the cell library, because this is the first step where the designer can have a glimpse of the final implementation. Another important point is that during logic synthesis, different reports can be generated, like power and timing reports, which will guide the subsequent steps in the design flow.

After logic synthesis, designers need to verify that the generated netlist implements the same functionality of its description using HDL in the same environment described in its testbench. A common approach to do so is to perform gate-level simulation of the resulting netlist with annotated gate delays. As explained before, the netlist is a schematic composed by a set of components, exclusively those available in the target library, interconnected by

wires. It is usually described in Verilog, although it can be described using other HDLs. Gate delays are annotated from the power and timing models and are usually exported to a Standard Delay Format (SDF) file, a standard defined by IEEE [IEE] and widely supported by EDA vendors. Furthermore, synthesis tools can also calculate the expected interconnect delays using wire models and capacitance tables, to approximate wire delays based on aspects like design area and cell count. However, in modern technologies the precision of these methods is questionable and synthesis tools are adopting more elaborated strategies like performing a rough physical synthesis to estimate interconnect delays [Key01].

Gate level simulation is somewhat similar to functional simulation. In fact, the same simulation tools can be employed here. The difference is that in gate level simulation the simulator relies on the cells as the primitives for evaluating the behavior of the circuit. In other words, instead of evaluating the behavior of the DUV as a set of its input-output relations only, it accounts for the input-output relations of each of the cells that compose the design. To do so, gate level simulation requires a set of behavioral models for such cells. Behavioral models are descriptions, typically in some HDL, that define the cell behavior for all states and corresponding transition arcs. These models describe the values on the cell outputs for each combination of input values as well as the transitions on outputs generated by transitions in the inputs. This is typically achieved with user defined primitives (UDPs) in Verilog, which allow to describe behavior using truth tables with special symbols that indicate static values and transitions.

Because it allows evaluating the behavior and delay characteristics of each cell of the circuit with the stimuli provided by the testbench, gate level simulation allows identifying problems that can emerge from the logic synthesis process. In fact, albeit formal verification enables ensuring that the netlist implements the specified functionality, it does not allow to explore issues related to the timing of the synthesized circuit. At this stage, issues identified in the design may require the designer to rerun logic synthesis steps and perform optimizations or even revise the HDL source code. Once the netlist has its correct functionality verified through gate level simulation, physical synthesis can take place.

Physical synthesis is, in essence, the process of transforming the synthesized netlist in a layout that can be sent to a foundry for fabrication. Its starting point is the mapped netlist generated by the logic synthesis. From there, a physical tool set generates the layout of the design using a set of models that describe technology specific parameters, and characteristics of the employed cells (physical, timing and power). A layout is a set of polygons that represent the different layers (or materials) used for the fabrication of the IC design. These layers include, but are not limited to: polysilicon, diffusion, different dopings, different levels of metal, contacts and vias. Polysilicon, diffusion and doping layers are typically employed for creating transistors (and other basic devices like diodes). Metal layers are used to route such devices and modern technologies can have as many as 13 different levels of metal. These metal layers are stacked during fabrication and interconnected using vias.

Contacts allow connecting the first level of metal to diffusion layers, *i.e.* to the terminals of transistors.

To generate the layout, the following major steps are typically performed [RCN03]:

- Floorplanning: The first step in the physical design of an IC is the floorplanning. This step is where sub-modules of the design are spatially distributed by the designer, organizing the layout. The idea is to keep modules that are connected closer to each other. Furthermore, sub-modules that are connected to the primary inputs/outputs of the design should be positioned in the periphery of the design. Note that, in designs that employ macro blocks, this is also the step where they are positioned in the layout. It is also during the floorplanning that the power distribution in the layout of the design is organized, *i.e.* power rails are created and distributed along the layout. Furthermore, in deep sub-micron technologies this step also counts with the insertion of tap cells, which connect the power rails to the body to adjust body voltage.

- Placement: With the sub-modules of the design distributed and the layout organized, the next step is to place the cells. This process defines the precise position of each cell in the layout and is guided by the floorplan. To do so, the physical synthesis tool requires a physical model that describes aspects like width and height of the cells and identifies where their input and output pins are placed. These models are known as abstract views and are extracted from the layout of the cells. A common format for abstract views is the library exchange format (LEF) from Cadence [Cad]. Abstract views are useful because they help saving computational resources, by providing only the information that the placement tool needs to use the cells, reducing memory usage during the design process. It is also during this step that the clock (and other global signals, like reset) has the circuitry for its distribution synthesized.

- Routing: The next step is to connect the input and output pins of the placed cells (and macro blocks), *i.e.* route the interconnections to reproduce the logical netlist patterns. To do so, the physical synthesis tool relies on abstract models, to locate the pins of the cells, and on technology specific parameters, such as metal width. Note that the tool may require to place extra cells, typically buffers, for ensuring signal integrity in long wires. In fact, the process of generating the interconnections in a large IC design is very complex, because the generated wires add capacitance and resistance to the nodes that connect cells.

- Place & Route Optimization: The placement and the routing of cells does affect local delay characteristics, modifying the logic synthesis output design timing. Furthermore, the delay of wires, which may have been overlooked during logic synthesis, can now cause a big impact in the IC performance [HMH01]. In this way, to ensure that the design continues to meet the defined constraints, several rounds of placement of cells

and routing of interconnects may be required. At each round, the physical synthesis tool computes the delay of logic paths using STA. To measure gate delays, timing and power models are used, and to measure wire delays the tool relies on technology specific information provided by the designer.

- Verification: Another important step, that can also be iterative is the verification of the generated layout. First, the designer can perform a LEC using tools similar to those available during the logic synthesis steps. This verification ensures that the resultant netlist, after place and route, still represents a logic equivalent to that of the input HDL code. The importance of performing such verification after the layout generation comes from the fact that, during place & route optimization, the netlist may have changed in order to accommodate wire delays in the defined constraints. Such modification can change the logic functionality of the circuit. Next, the designer needs to ensure that the layout respects design rules specified by the technology vendor, which, at this point, typically encompass only interconnecting layers, metal layers and vias, to reduce verification complexity. These include verifying and fixing antennas [RCN03] and maximum transition and capacitance values. Note that problems identified during the verification steps may require further optimization iterations.

- Fillers generation and extraction: Once the design is verified, filler cells and metal layers should be placed. They will ensure that design and minimum density rules are respected. The design can then be extracted from the IC layout, generating a netlist and the equivalent capacitance and resistance of interconnecting wires. Note that, at this point, the layout contains only abstract views in it, *i.e.*, only those layers required for place and route tasks. Using these values, EDA tools can export the delays of the circuit, now including wire delays, to a new SDF file, for performing a more precise gate-level simulation. These can also generate a transistor level netlists, to perform transistor-level analog simulation. The physical synthesis layout can be exported to a format compatible with other EDA tools and technology vendors, usually using the Graphic Database System (GDSII) format.

Once the physical synthesis is complete, the design can undergo a set of final verification steps. These can include a complete verification of the design rules with all layers of the layout, gate-level simulation, voltage drop analysis and transistor-level simulation. A complete verification of all layers of the layout is possible when the designer has full access to the cell layouts. Such verification is done importing the physical synthesis layout to a layout editor like Cadence Virtuoso [Cad], and allows ensuring that the design can be fabricated in the target technology. However, this is not the case in all semi-custom designs, as IPs and even cell libraries are usually provided by third party vendors, which may not disclose all the information of their products. In this case, designers usually have to send their physical synthesis layout, composed of abstract views only, to a third party that has

access to all views of the IPs, which will import all layers, generate the full layout and verify it. If the design complies with the defined technology rules, the layout is sent for fabrication. Otherwise, it is sent back to the designer with a detailed report for revision. This kind of intermediate verification is typically conducted by specialized enterprises, as in the case multi-project wafer services like MOSIS [MOS] or the french CMP [CMP], which serve low volume productions for private firms, research laboratories and universities.

Gate-level simulation can be performed similarly to the verification steps of logic synthesis and ensures that the post-layout design still implements the correct functionality in the implemented testbench. The difference is that, after physical synthesis, wire delays are already known and are also annotated for simulation purposes. Voltage drop analysis allows verifying that the power planning was well done and the power rails are sufficiently wide to distribute current across the chip. This analysis is particularly important for synchronous design, because given the fact that the clock signal controls all registers, at each clock edge there will be a high demand for current chip wide. In this way, the designer needs to ensure this worst case scenario is accounted for by the design.

After the layout generation phase is complete, the extracted layout can also be exported to Spice, which allows modeling the fundamental discrete components in VLSI design: transistors, capacitors, resistors, diodes and inductors [WH10]. With a Spice netlist, the designer can perform analog transistor-level simulation, which is the closest an IC designer can get to the real IC behavior in the design phase. Note that this simulation can require a lot of computational resources, as modern ICs can count with billions of transistors. In this way, it is common to simulate only specific logic paths using transistor-level simulation to speed-up verification. For instance, in synchronous designs, designers will usually simulate critical paths using analog simulators, as these are the ones that require careful handling, given the constrained nature of the synchronous paradigm. Analog simulation can take place using open source Spice simulators like Ngspice [Ngs], or tools provided by vendors, like Synopsys HSpice [Syn] or Cadence Spectre [Cad]. Also, these simulators require models for the devices used in Spice descriptions, for transistors, diodes, etc. These models are provided by the technology vendor, usually the foundry itself.

During physical synthesis and verification, issues that could not be identified in the previous design steps can now be detected. These issues can require running more optimization iterations in the physical synthesis, some times even some full custom adjustments may take place, or they can be more severe and require going back to the logic synthesis phase. In fact, in some extreme cases, even revision of the input HDL may be required. A common problem is the extra delays added by real wires, which were ignored in the first synthesis steps, which are called parasitic effects [RCN03, WH10]. All these final checks are usually referred to as signoff checks, as they are a check list before sending the design to fabrication. Once the design satisfactorily passes all these tests, the IC design can proceed to the foundry, where it is fabricated in wafers and cut into individual dies [WH10, RCN03].

Dies are then packaged and sent back to the designer, who will test them before shipping it for use in final products. This discussion is limited to the most common signoff checks. However, there are other verification steps that companies may perform before signing off, like signal integrity analysis and electromigration checks [RCN03].

2.1.3 Cell Library Design

As Section 2.1.2 described, a cell is a basic building block used for semi-custom IC design styles. They are circuits with input and output pins and a set of semiconductor devices connected by wires, and which implement basic logic functions. Cells can be combinational or sequential, which means that they can rely only in the values of their inputs to compute the value in their outputs or can also rely on internally stored values to do so. Furthermore, albeit the design of a cell can require analog design knowledge, the behavior of cells outside of their interface boundaries (input and output pins) is most often purely digital. Accordingly, cell-based design avoids the need for transistor-level design and raises the design abstraction level by enabling to consider cells as black boxes with a logic behavior. However, to do so, a set of models is required so that designers, or EDA tools can employ these black boxes to compose an IC. The models comprise information of each cell logic behavior as well as electrical and physical parameters. Figure 2.5 depicts a basic flow to generate these models. As the Figure shows, the process of designing a cell starts from a specification, which defines its basic aspects, i.e.:

- The cell logical behavior, which provides the logic function the cell will implement. For instance, as reported by Rabaey et al. [RCN03], libraries need different logic functions. Accordingly, cells from a library will implement at least the basic combinational functions, such as inversion, AND/NAND, OR/NOR, XOR/XNOR, and also some sequential logic behavors, such as the functionality of latches and flip-flops. Furthermore, libraries can also provide more complex cells, like generic sums of products, products of sums, multiplexers, demultiplexers, encoders, decoders and bit slices for arithmetic blocks, such as adders and subtractors.

- The cell electrical requirements, which can provide power and delay constraints associated to the cell. Because cells are used in semi-custom design, it is not known in advance what will be connected in their outputs; therefore, the load that they will need to drive (charge/discharge) is unknown. Bear in mind, though, that this load will have a significant impact in the performance of the cell [WH10]. Hence, libraries often present different versions of each cell, where each version has a different size (in fact, some of its transistors have different sizes) and is capable of driving a different amount of load in given period of time. Different cell sizes correspond to cells with differing driv-

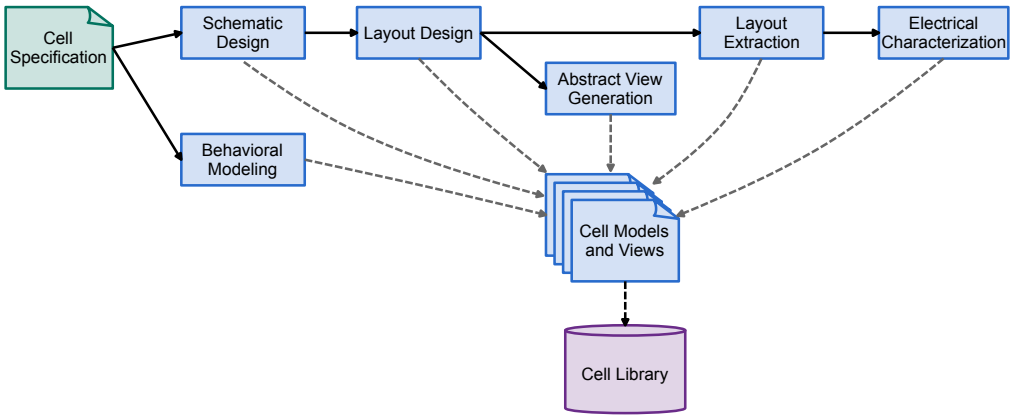


Figure 2.5 – Basic design flow for the design of a cell.

ing strengths, also called just the drive of the cell. Each of these versions is typically designed to be able drive a capacitance 4 times bigger than its input capacitance, a metric known as fanout of 4 (FO4). Note that, the bigger the driving strength, the bigger the load that the cell can drive, but also the more power hungry the cell is. In this context, electrical requirements furnish parameters to dimension the cell.

Using the specified logical behavior, behavioral models are captured. These models are usually HDL descriptions in Verilog or VHDL, and are employed in digital simulation with gate-level simulators for verification purposes. The logical behavior is also used to generate a schematic that put together semiconductor devices of a target technology and connect these with wires to implement the specified function. The devices are arranged and sized to meet the electrical requirements, according to analytic approaches or by iterative simulation approaches, which trade off design time and quality [RCN03, WH10]. After the cell has its schematic appropriately sized, it is laid out. The layout process consists of arranging polygons that represent the different layers used to fabricate an IC in a specific technology. This is typically done using a layout editor tool, like Virtuoso from Cadence [Cad].

An important aspect to keep in mind during cell-based design is the fact that EDA tools are used from the front end to back end tasks and to make it possible for automated tools to place and connect cells, rules are needed. Thinking about Lego blocks, the reason why one can build bigger structures with them is that all of these bricks can connect to each other in a standard way. Moreover, all blocks have standard sizes and standard places for the connection bumps. In order to design a standard-cell library, it is necessary that all cell respect the same set of rules [SS02]. In this way, the gates will fit together in predictable patterns and it is easier to automate the process of generating circuit layouts from abstract descriptions, such as RTL descriptions. Figure 2.6 shows a typical set of rules used to design standard-cell libraries, also called *the cell architecture*. Note that this architecture can vary depending on the technology. For instance, the one showed here does not include bulk

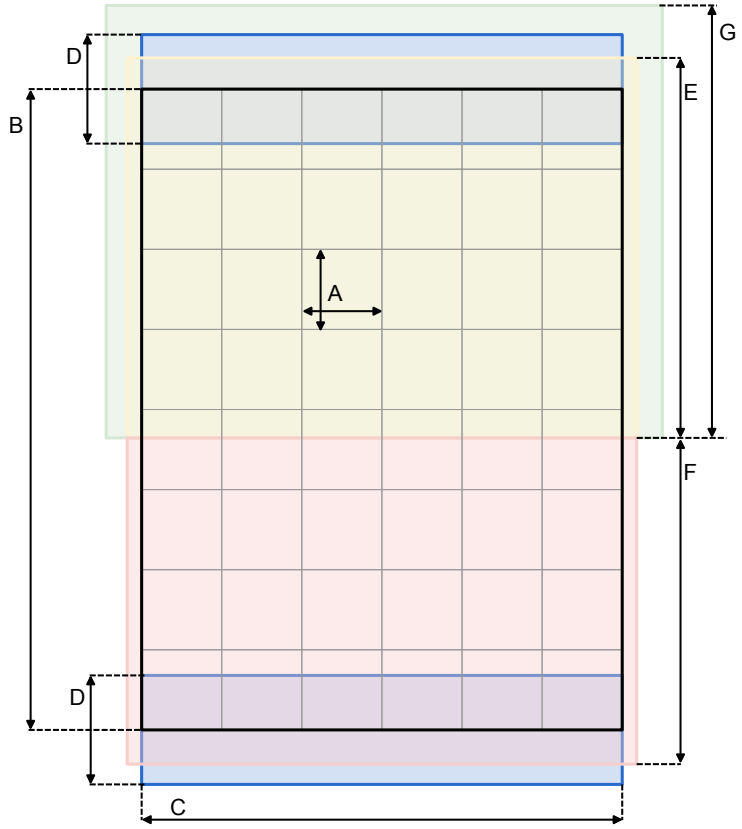


Figure 2.6 – Typical architectural rules for a cell from a standard-cell library, adapted from [Mor10]. “A” is the cell pitch (or routing grid pitch), “B” is the cell height, “C” is the cell width (which can vary), “D” is the power rails width and “E”, “F” and “G” are the height of implant layers [RCN03, SS02, SS04].

contacts. That is because these are not usual in cell libraries targeting recent technologies, which rather use special cells called tap cells for controlling bulk voltage. For older technology nodes, though, the designer should include the position of bulk connections in the cell architecture specification.

The most fundamental rule in a cell architecture is the cell pitch or routing grid pitch, “A” in Figure 2.6. This pitch is typically defined as the metal pitch of the target technology and establishes the minimum step that a cell can grow in width and height and is used by back end tools to perform power planning, place and route steps. In fact, a place and route tool is typically used to build a cell-based circuit from the cells provided by the library and does it by placing the cells side by side and then connecting the pins of these cells according to a synthesized netlist. Having a fixed cell height (“B” in Figure 2.6) reduces design complexity and helps the automation of the IC design process, as it facilitates the arrangement of power and other long or global lines/wires. Note that this height must be multiple of the cell pitch “A”. The cell width, however (depicted as “C” in the example architecture), can vary, because the transistor count varies from cell to cell and transistors are added horizontally in a cell layout. However, the width must also respect the cell pitch, *i.e.* “C” must always be a multiple of “A”.

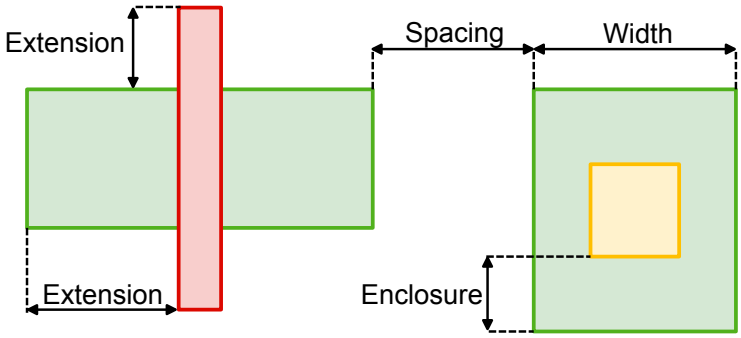


Figure 2.7 – The basic set of DRC rule classes [RCN03, SS02, SS04].

Another standardized dimension is the height of the implant layers, depicted in this example as “E”, “F” and “G”, due to the fact that the cells are usually placed side by side.

Another important aspect in the design of a full-custom design, in this case each library cell, is the manufacturing grid, which defines the metric unit to design a layout. For instance, if the manufacturing grid is 0.010 nm, every measure in the layout must be a multiple of 0.010 nm in both axes, horizontal (X) and vertical (Y), in order to have a manufacturable layout. Furthermore, to guarantee that a layout is manufacturable, the designed layers must respect some design rules stated by the semiconductor foundry. According to Rabaey et al. and Saint et al. [RCN03, SS02, SS04], 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 layers, between different structures in the same layer, or even rules to define how to draw a structure in a given layer. This makes the task of designing a layout harder, as technology complexity increases. Accordingly, sub-micron technologies from the early 2000s had a few hundreds of design rules, while the latest nanometric technologies encompass thousands of rules [ITR]. 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, like Calibre DRC from Mentor [Men]. This tool verifies if the generated layout can be properly manufactured in a specific foundry and technology process.

Ensuring that the circuit can be correctly implemented in silicon does not necessarily mean that it will operate correctly. To ensure that the layout implements the specified logic, a layout versus schematic (LVS) check must be performed through an LVS tool, which compares the schematic with the layout and checks if the circuits are equivalent, *i.e.* have the same transistor arrangement and interconnections. To do so, LVS tools extract the equivalent schematic from a layout and compare it to a golden model, typically a schematic furnished by the designer. These tools identify issues like incorrect logic behavior, missing/extra pins and nodes and incorrect pins and nodes labels. A tool commonly employed for this purpose is Calibre LVS from Mentor [Men].

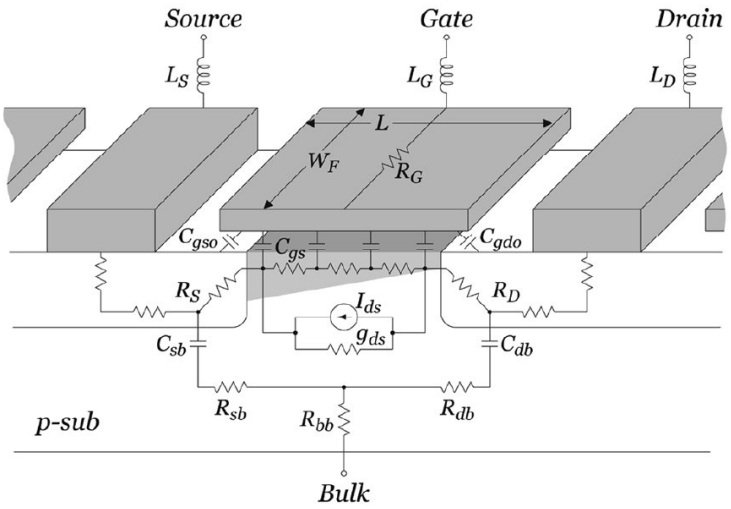


Figure 2.8 – Physical model of a planar bulk NMOS transistor, extracted from [DENB05].

Once a cell is laid out and fully verified, its abstract view can be generated. This 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]. An abstract view contains only the metal layers of a layout with its terminals and boundaries, which is crucial for IC assembly automation. Specifically, this view is used in place and route tasks, as it provides the necessary information to place each gate and connect its terminals. There are two basic types of metal layers in an abstract view, blockage and net layers. The former defines metal layers employed for the internal routing of a cell and identify the boundaries for the route tool to place the interconnects during physical synthesis. The latter defines metal layers that are connected to input and output pins of the cell and help the place and route tool to identify where to connect metal layers for the routing of the IC during physical synthesis. Also, because it contains only metal layers it avoids the overhead of all the layers that are required for fabrication but not for place and route, reducing the usage of computational resources during physical synthesis. The most adopted format for abstract views is the library exchange format (LEF), by Cadence [Cad].

Following the abstract view generation step, a parasitics extraction 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 terminals of a transistor and the resistance of these terminals. This allows a more accurate model of the devices that compose an IC, enabling more precise simulation and analysis of the designed circuit. Figure 2.8 shows parasitic elements for a physical implementation of an NMOS transistor. Note that a bulk CMOS transistor is showed here, albeit there are similar models for newer technologies like SOI and multi-gate devices. A widely employed tool for prasitics extraction is Calibre PEX from Mentor [Men].

As Figure 2.5 shows, the last model required by cell-based design comprises timing and power characteristics of each cell of a library. The generation of these models is known as electrical characterization and the quality of a cell-based design depends strongly on the quality of the characterization of the cells employed. This is because the models generated in this process guide the tools employed in the design flow, from logic to physical synthesis. Moreover, due to variations in the fabrication process, cells must be characterized at different manufacturing corners, which will reflect their real behavior for different fabrication process conditions. In this way, it is possible to verify that a design will operate correctly and perform to its required timing specification, even under worst case conditions. In addition, there is also the influence of operating conditions, like temperature and voltage, which can make the delay of the cells vary as well. Hence, a cell library needs to be characterized for the several PVT variations it is expected to face, to ensure a robust design flow. Fortunately, there are tools that automate the task of exhaustively simulating all necessary conditions to ensure the reliability of a characterization process. The most adopted ones are Cadence ELC [Cad] and Synopsys NCX [Syn].

2.2 Asynchronous Design

The synchronous paradigm is very attractive for designing digital circuits mostly due to its simplicity. The clock signal provides a temporal reference for event sequencing and synchronization, which simplifies the design process. The asynchronous paradigm, on the other hand, assumes no global or, in some cases, not even regional clock signals for control and sequencing of events. Instead, asynchronous modules use handshake protocols between their sequential components to synchronize, communicate and operate [Mye01, SF01a, BOF10]. This means that each pair of registers communicates by explicitly signaling sending and receiving data. In “synchronous terms”, the resulting behavior corresponds to registers clocked only when and where needed. Such characteristic presents several advantages over the use of a global clock signal in modern technologies, as already discussed in Section 1.2. This section explores basic concepts behind asynchronous design, which are part of the foundation of this Thesis.

2.2.1 Channels and Handshake Protocols

The most basic and intuitive manner of implementing asynchronous communication consists in using two control signals in opposite directions, request and acknowledge (usually shortened to *req* and *ack*). As Figure 2.9 shows, one protocol consists in an active element sending a request to synchronize with a passive element, which issues an acknowl-

edgement when it is ready to communicate. As the Figure shows, this is done through an asynchronous channel (or just channel), which can be formed by connecting the ports of an active and a passive element. Moreover, according to Beerel et al. [BOF10], a channel also requires the definition of a protocol for synchronizing computation and communicating data between elements. Such protocols are typically based on 2- or 4-phase handshaking, or transition signaling and level signaling protocols [SF01a]. Hence, a channel is defined as a communication link, a set of wires, and a protocol for synchronizing computation and communication [BOF10].

For instance, Figure 2.10(a) shows an example of a 4-phase handshake communication. In this example, the active element starts with a requisition to communicate, rising the req signal. The passive element reads, processes the request and asserts the ack signal. When the active element reads the acknowledgement, it sets the req signal to low, which the passive element acknowledges by lowering the ack signal as well. After that, a new communication can take place at any subsequent moment. The transition signaling protocol reduces transitions to half of the former. In this protocol, illustrates in Figure 2.10(b), the active element starts with a request, switching the logic value of the req signal. When the passive element reads the request, it sends an acknowledgement by switching the logic value of the ack signal.

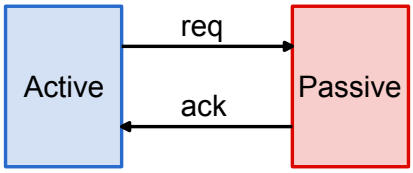


Figure 2.9 – Example of pure control communication through handshaking.

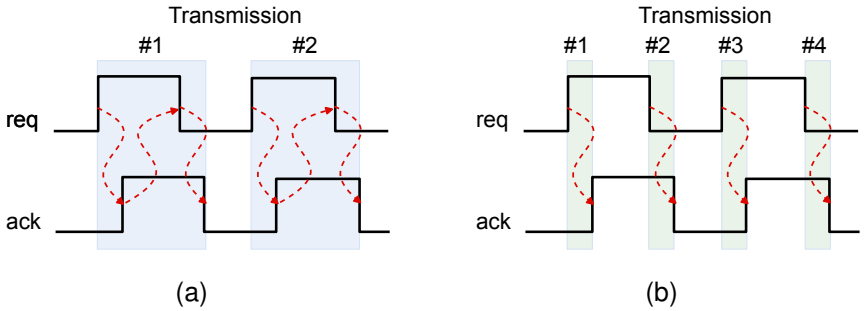


Figure 2.10 – Operation of handshake protocols: (a) 4-phase and (b) 2-phase.

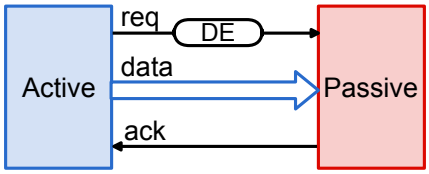


Figure 2.11 – Example of a BD channel.

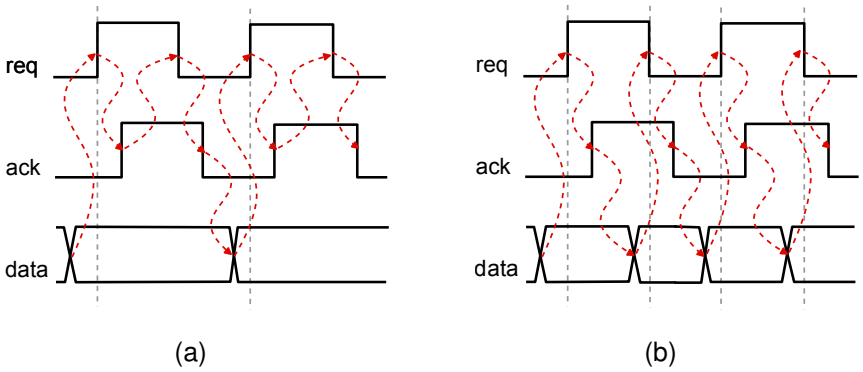


Figure 2.12 – Example of (a) 4-phase and (b) 2-phase bundled-data communication.

2.2.2 Bundled-Data Channels

Channels can serve for control purposes only, like in the example showed in Figure 2.9. However, if data is to be transferred asynchronously, designers require data ports to be included in communicating elements and channels. The most intuitive manner to do so is to add a single rail data bus to the scheme showed in Figure 2.9, which is somewhat similar to synchronous design. In fact, as in synchronous design, data validity is signaled by an explicit control signal (typically called req), equivalent to the clock signal. The difference is that data capturing must also be signalled by an explicit control signal (typically ack, as handshake protocols mandate. Asynchronous channels relying on single rail representation of data and explicit control signals for request and acknowledge are typically called single-rail, or bundled-data (BD), channels [BOF10].

Figure 2.11 shows an example where the active element sends data to the passive element through a channel, composed of the data bus data and the control wires req and ack. Beside the channel structure, it is necessary the definition of a handshake protocol to follow. Figure 2.12(a) shows an example of a 4-phase BD channel. First, information is inserted in the data channel, filling the data bus. After data stabilizes, the req wire is set to 1, indicating a communication request. This request is acknowledged by the ack signal going to 1. Next, the req signal must switch back to 0, which must be followed by the ack signal also switching back to 0. From this point on, a new communication can take place.

As Figure 2.12 shows, in BD channels there is an assumption that the control signal that identifies the validity of data will always be issued after the data in such bus is valid and stable. In this way, an explicit relationship between the delay of data and control signals exists. To ensure that this assumption holds true, a delay element (DE) [BOF10], see Figure 2.11, is typically employed and must match the worst case delay of the data bus, to ensure that the receiver obtains always correct data values from the channel. Delays are typically adjusted using a pair number of inverters or any number of non-inverting buffers. The drawback of the approach is that the assumption of bundled control signals requires extra care with the computation of timing constraints between data and control signals and implementations that respect these.

Figure 2.12(b), in turn, shows an example of a 2-phase BD channel. After data is stable in the data bus, a request is signaled by switching the logic value of req. When the acknowledgement is issued, i.e. after the ack signal has its logic value switched, a new communication can begin. Note that in this example there is no data transformation between the elements. However in logic circuits, there may be a combinational block computing the data and the DE on the request signal must match the worst case delay of this block. Furthermore, the channels depicted in the examples, where two handshaking elements transfer data from the active to the passive, are called *push channels*. Another configuration is to use *pull channels* [BOF10], where control signals flow in the same senses as before, but data flows instead from the passive to the active element. A discussion similar to that of push channels applies to pull channels, *mutatis mutandis*.

2.2.3 Delay-Insensitive Channels

Another possibility for organizing asynchronous computations is the codification of the request/acknowledge signal within the data channel. This is the strategy adopted by delay-insensitive (DI) channels. In these channels data can also be transmitted through either push or pull channels. For the former, the request signal is encoded within the data, while for the later the encoded signal is the acknowledge. To encode data, a DI channel requires the choice for a handshaking protocol and a DI code to represent data. An excellent approach of the principles behind delay insensitivity and an exploration of several different DI codes was provided by Verhoeff in [Ver88].

According to Verhoeff, a code is a pair *(I,C)*, where *I* is a finite set and *C* is a set of subsets of *I*. The length of a code is defined as the size of *I* and the size of the code is defined as the size of *C*. In other words, the length of the code is the number of wires that will be used to represent data (the size of *I*) and the size of the code is the number of different messages that can be sent using that code. Note that because Boolean representations are assumed, the size of a code can be at most two to the power of the length of the code. A

codeword is an element of *C*, and its cardinality is also called its weight. A code *(I,C)* is DI when

$$(\forall x, y : x \in C \wedge y \in C \wedge x \subset y : x = y) \tag{2.1}$$

In other words, for a code to be DI, no codeword can be contained in another codeword. Such characteristic enables a receiver to perceive unambiguously the transmission of a codeword. For instance, if *C* had two distinct codewords *a* and *b*, such that $a \in b$, the receiver could temporarily perceive *a* when either *a* or *b* was sent. However, in that case, the receiver cannot correctly detect the codeword whenever the sender transmits *a*.

Using the Verhoeff definition a wide variety of DI codes can be devised for use in DI channels. In fact, different works proposed a variety of DI channels in the past, as reported in [PCV12, BTEF03, AN12, PVT15]. Yet, according to Martin and Nyström, 4-phase handshaking coupled to 1-of-n DI codes is the most practical and employed scheme, because in general it allows reducing design complexity [MN06]. Hence, the discussion on DI channels presented here will be limited to these choices. In 1-of-n DI codes (also called one-hot codes), valid data is represented using *n* wires and data validity is signaled by setting exactly one wire to 1, leading to codes with exactly *n* distinct values. This is equivalent to rising the request/acknowledge (for push/pull channels) signal in 4-phase BD circuits. In addition, absence of data (also called spacer) is signaled by setting all wires to 0, equivalent to lowering the request/acknowledge signal in 4-phase BD. By doing so, it is ensured that Equation 2.1 holds true, because each valid data codeword sets a different wire to 1 and the spacer is the only codeword where all wires are set to 0.

For instance, in a 4-phase 1-of-2 quasi-delay-insensitive (QDI, for a definition, see Section 2.2.4) template using push channels, the request signal is encoded into the data signals making use of two wires per data bit (*d.1* and *d.0* are example names for the two distinct wires). These channels are also called dual-rail, because they employ two wires (rails) per bit of data. As Table 2.1 shows, in dual-rail channels, a valid 0 is represented by a low *d.1* and a high *d.0*, while valid 1 uses an opposite encoding. Hence, data validity corresponds to *d.1* and *d.0* having different logical values. To signal a spacer, the absence of data, both *d.0* and *d.1* are set to 0. Figure 2.13 shows an example of a 4-phase dual-rail data transmission through a push channel. Communication starts with all data wires at 0, indicating absence of data (a spacer). Next, the sender puts valid data, in this case a valid 0, in the channel. The receiver then computes the request (by observing a valid combination in *d.1* and *d.0*) and acknowledges the data, setting the *ack* signal high. When the sender receives the acknowledgement, it transmits a spacer to finish the transmission. The receiver computes the spacer and sets *ack* low. The sender can then start a new transmission. In this example it sends a valid 1. Note that because between each transmission all wires must be at 0, this protocol is also known as return-to-zero (RTZ) protocol. A similar analysis holds true for the variety of 1-of-n codes available for push and pull DI channels.

Table 2.1 – 4-phase 1-of-2 RTZ encoding for one data bit.

Value	d.1	d.0
Spacer	0	0
Valid 0	0	1
Valid 1	1	0
Invalid	1	1

2.2.4 Asynchronous Circuits Templates

Asynchronous channels are useful means for expressing asynchronous communication. Coupled to synchronizers and/or converter interfaces, these channels can alleviate IP based design complexity using approaches like globally-asynchronous locally-synchronous (GALS) design [Cha84]. However, channels alone are not sufficient for designing asynchronous logic, which requires another definition. To do so, the definition of an asynchronous template is required. As Figure 2.14 shows, an asynchronous template is characterized by the choice of an asynchronous channel type and a specific design style. A design style is defined as a specific set of components and an architecture. The former is an important definition because it typically comprehend components different from those available in conventional libraries, as they are not required for implementing synchronous logic. Moreover, these components are typically designed at transistor level as cells of cell libraries, which defines an asynchronous template as a semi-custom design approach. The architecture defines the structure of the circuit, specifying how control and logic blocks are designed and arranged. Note that architectures are usually specified by defining a pipeline structure, as pipelining is a fundamental technique to increase concurrency, widely used in digital design. Hence, the definition of an asynchronous template enables a precise spec-

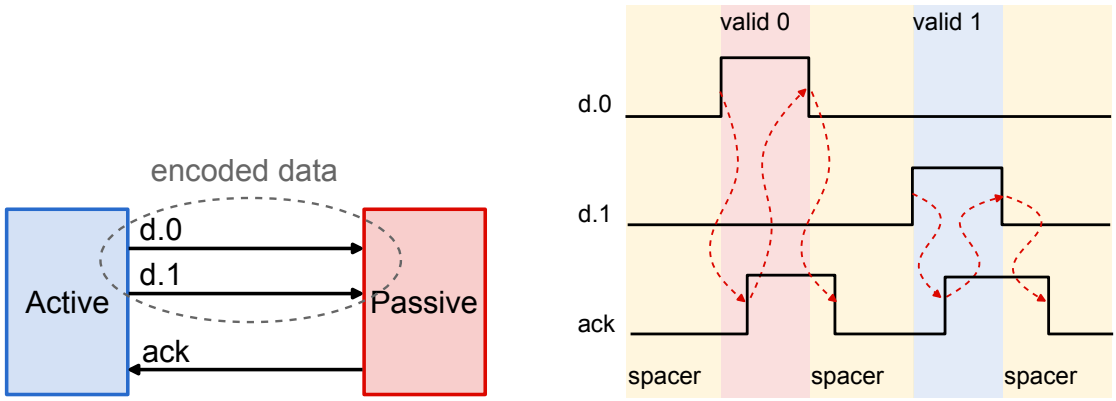


Figure 2.13 – Example of 4-phase 1-of-2 RTZ DI channel for 1 bit: (a) block diagram of two communicating elements and (b) example waveform for the transmission of two values.

ification of how data is encoded and transmitted across the circuit and how logic blocks, sequential and combinational, are designed.

Templates are an important definition in asynchronous circuits design because they make clear one of the major differences from the traditional synchronous paradigm. Accordingly, synchronous designers rely on a single template for implementing ICs, where a clock controls all sequential components [RCN03]. Furthermore, consolidated libraries are used for synthesizing a circuit from an HDL description and clock distribution circuitry and cells are used for ensuring that timing constraints are respected. For asynchronous circuits design, there are no such libraries easily accessible and no clear definitions on how to build combinational and sequential logic, because circuits can rely on different types of channels for communication. Hence, the definition of an asynchronous template enables a precise specification of how data is encoded and transmitted across the circuit and how sequential and combinational logic blocks are designed.

Asynchronous circuit templates that use BD channels are also called BD templates. Sutherland developed a seminal work where he proposes a BD template called *micropipelines* [Sut89]. According to Nowick and Singh [NS11], this template was the starting point for the development of several more advanced templates like Mousetrap [SN07] and GasP [SF01b], both devised for high performance digital asynchronous design. One of the major advantages of BD templates is that they can be enable to allow the usage of conventional EDA tools and libraries. As described in [Gib13, GMC15], commercial synthesis tools originally devised for synchronous design can be employed to design BD circuits using a set of scripts and a specifically engineered design flow. Furthermore, design styles for BD templates make wide usage of conventional cell libraries, partly because BD channels rely on single-rail data representation, as in synchronous design. In fact, several works propose design flows that start from a synchronous specification of an IC and generate its BD version using conventional EDA tools and some scripts, as in [CKLS06].

Figure 2.15 shows a generic example of circuit architecture for BD templates. These architectures are typically specified as pipeline stages that detail the design of sequential and combinational logic as well as control circuitry. The major differences between

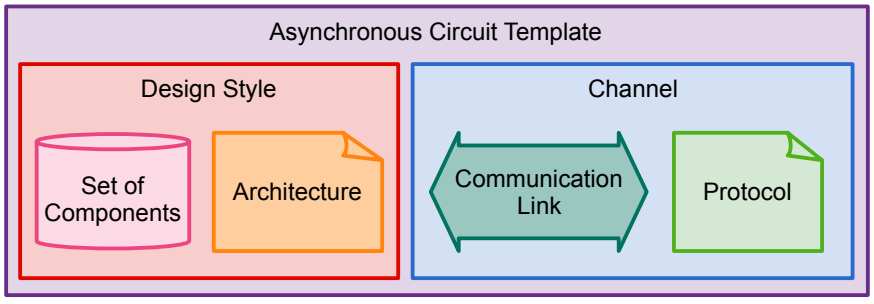


Figure 2.14 – Asynchronous circuits template definition.

different BD architectures are typically on control blocks (*Cont.*) and in the employed registers (*Reg*). Control blocks can be specified as asynchronous finite state machines (AFSMs), like in Blade [HMH+15], or as explicit circuit blocks, as in Mousetrap [SN07]. Registers can be typical latches or flip-flops [NS11] or more sofisticated circuits, as the capture-pass latches proposed by Sutherland [Sut89]. In asynchronous templates, logic blocks between registers must be compatible with the channels they are connected to. In BD templates, this is not usually a concern, because these blocks are generally realized as conventional single-rail Boolean logic, given the nature of BD channels. An important aspect of the architectures of BD templates, though, is the specification of where DEs are placed and how they relate to control and logic circuitry. Accordingly, they must always respect the definitions of Section 2.2.2, and data validity can only be signalled after computation is completed and information in the data bus is stable and valid.

One of the major drawbacks of BD templates is related to the constrained nature of BD channels. The problem is that the timing constraints imposed by these channels can be strict, depending on the design and on the target technology, as in synchronous circuits. This is problematic because it makes more challenging for designers to ensure timing closure during synthesis steps. Moreover, because these circuits are sensitive to delay variations, they usually suffer performance and power-efficiency penalties from the required addition of timing and voltage margins to take variations into account. As in synchronous design, these margins need to be added to ensure that the request signal is always slower than the worst case delay of logic paths. Furthermore, according to Martin and Nyström in [MN06], the constrained nature of BD templates makes them as challenging as synchronous design and the required margins can wave off the advantages of asynchronous circuits.

A more robust alternative is to employ DI channels, which do not rely on timing constraints for signalling data validity as BD channels do. This is because DI channels encode data validity information within the data itself, as explained in Section 2.2.3. The problem though is how to implement logic blocks that do not compromise the characteristics of these channels. In fact, in 1990 Martin proved that it is not practical to build purely DI circuit templates [Mar90]. Martin proved that in order to build useful circuits using DI channels,

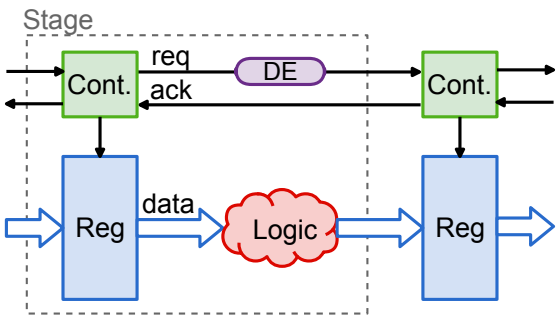


Figure 2.15 – Generic example of circuit architecture for BD templates.

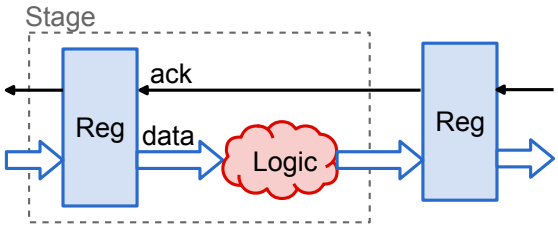


Figure 2.16 – Generic example of circuit architecture for QDI templates.

designers need to compromise the delay-insensitivity of circuits and proposes another class of circuits (a new design style) called quasi-delay-insensitive (QDI). This design style enables designers to ignore the delay of all gates and wires, due to the use of DI codes, except for a set of specific wire forks called *isochronic forks*. Practically speaking, the timing assumption in isochronic forks is that the delay from the input to one of the fork ends is smaller than the delay from the input to the other end (the constraining path). The purpose of this assumption is to ensure the ordering of transitions in the circuits, typically control paths, and it can be usually easily fulfilled because the constraining path is typically long, as explored in detail in [Mar90, MN06, BOF10].

Templates relying on DI channels and QDI design styles are called QDI templates. Classically, their architecture is specified similarly to those of BD templates, as Figure 2.16 shows. The main difference is that the control block is merged with the register. This is due to the fact that these components are usually design style-specific and are not conventional sequential components like latches of flip-flops. In these design styles, the components and methods required to build logic blocks must also be precisely specified, as they must compute data that is encoded using DI codes. Furthermore, these logic blocks cannot compromise the protocol employed in the choice of a DI channel. The components required for designing circuits targeting QDI templates are often specially specified for the target template and are rarely available in conventional cell libraries. In fact, QDI architectures usually have a set of components specifically designed for supporting them.

One of the most basic QDI templates is based on delay-insensitive minterm synthesis (DIMS) [SF01a]. From this basic template, several more sophisticated approaches were proposed in the last decades, like null convention logic (NCL) [FB96]. Furthermore, some QDI templates evolved to even avoid the separate definition of registers and combinational blocks. In these templates pipeline stages are reduced to communicating logic blocks, as in precharged half buffer (PCHB) design [Lin98]. This is possible because these blocks have sequential and control circuitry implemented within them. Unfortunately, the design of circuits targeting such templates requires deep knowledge of asynchronous design and microelectronics. This is because these components are usually full-custom, *i.e.* designers need to build their own cell libraries for each different QDI template. In fact, this is one of

the major drawbacks of these templates and one of the reasons why they do not look as appealing as BD templates.

Contemporary literature has the availability of different flavors of QDI and BD templates. The works presented in [NS11, NS15a, NS15b, BOF10] provide a solid overview of these template options. Due to their orthogonal advantages, BD and QDI templates divide asynchronous design in two clearly distinct fields. As a consequence, there is also a division in the interests of the asynchronous circuits research community. A good survey of QDI templates can be found in [MN06] and a nice summary of the advantages of BD design is available in [NS15a]. To summarize, BD templates have the advantage of relying on conventional cell libraries and tools, while QDI templates typically require special components in their design. The problem is that these special components are not compatible with conventional tools [SGY+09], which can compromise design automation. Furthermore, in comparison to QDI templates, BD templates can provide better power efficiency, due to the use of single rail channels with explicit control channels. This is because this approach can reduce switching activity, which is directly related to power overheads. QDI templates on the other hand can provide high speed design, as discussed in [DLD+14], while BD design can get over-constrained by margins, as argued in [MN06]. In fact, QDI templates can easily accomodate delay discrepancies, which make them good candidates for IC design in environments susceptible to a high degree of variability, as in modern technologies or when using near- or sub-threshold operation to obtain extrem savings in power [LOM07]. In this way, in essence, BD and QDI templates trade off design complexity and robustness.

3. INNOVATIONS IN ASYNCHRONOUS COMPONENTS DESIGN

An asynchronous template, as defined in Section 2.2.4 relies on a set of basic components for its specification. These components are basic circuits that implement specific functionalities to allow the use of semi-custom approaches to reduce complexity, costs and risk associated to full-custom design approaches. In this way, the quality of an asynchronous circuit designed for a specific template is a function of the quality of the components that compose this template. The quality of such components, as basic circuits, is typically measured by their electrical and physical characteristics such as static and dynamic power, propagation and transition delay and area. Note that at this level one cannot consider architectural optimizations, because, recalling Figure 2.14, these are provided by the architecture definition of the asynchronous template rather than the set of components. Hence, the optimization of the set of cells that compose a given target asynchronous template is an important step in the design of an asynchronous circuit.

With this in mind, the optimization of asynchronous components was one of the first tasks of this work. In fact, the first set of contributions of this Thesis is described as a set of innovations in asynchronous components design. We firstly selected a basic set of components that support different asynchronous templates and explored optimization in these circuits. Then, as the work advanced, we proposed new components for existing templates and for templates that we devised, as explored in the next Sections. This chapter is divided in four major sections, starting with a revision in the state-of-the-art in asynchronous components. The next two sections explore the novelty brought by this work on C-elements and NCL gates. Finally we discuss the impact of our innovations in asynchronous components design.

3.1 State-of-the-Art in Asynchronous Components Design

This section addresses the state-of-the-art in asynchronous components design. Note that our focus here is on the components used in the templates explored in this Thesis.

3.1.1 The Mutual Exclusion Element

The Mutual Exclusion Element (MUTEX) is a basic component for asynchronous design and is a common requirement in the set of components of asynchronous templates. It is typically employed to construct arbiters in control blocks of asynchronous circuits, although it is not an usual component in synchronous design. In fact, the discrete notion of time in

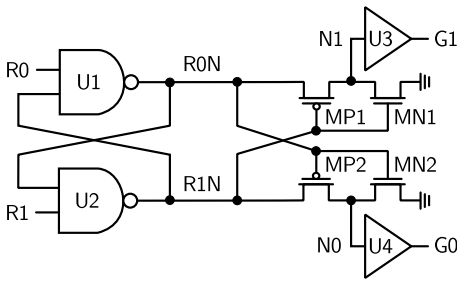


Figure 3.1 – General scheme for a simple MUTEX. It is composed by two cross-coupled NANDs and a metastability filter (MF), [ZHM+15].

synchronous systems enables avoiding the need for such a component. Having a global clock signal that dictates how time advances makes arbitration between competing requests to shared resources a relatively simple task. If two requests arrive during the same clock cycle, one can be given priority as defined by a given arbitration policy implemented using standard combinational logic.

In asynchronous circuits, on the other hand, arbitration works quite differently. This is because they employ a continuous time model. Determining who arrives first in the continuous time domain can be challenging when requests are close in time. This is where special components like the MUTEX are required. Resolving which request arrives first is so difficult that the time for any MUTEX implementation to make this decision is unbounded [Kin08, YG08, BOF10]. In particular, decisions rely on MUTEX internal non-digital behavior to determine which request to grant first. Moreover, before the decision, one or more internal signals can remain in a metastable state [BOF10]. For this reason, a MUTEX includes specialized metastability filters (MFs) that ensure outputs do not switch until the component takes a decision. These filters are expected to guarantee that metastable states do not propagate to downstream logic, which could lead to unpredictable and irreversible effects.

As Figure 3.1 shows, the simplest MUTEX has two request inputs (*R0* and *R1*) and two grant outputs (*G0* and *G1*). Its internal organization usually comprises two cross-coupled NANDs (*U1* and *U2*) that receive the input requests, and its outputs are filtered by MFs. Figure 3.2 shows the discrete behavior of the MUTEX using a state diagram. In this diagram, state representation involves primary inputs and outputs in the order *R0R1:G0G1*. Also, solid edges represent input transitions while dashed edges represent the MUTEX behavior due to the corresponding input changes.

Assuming a start state where *R0* and *R1* are at 0, *U1* and *U2* write a 1 on their outputs (*R0N* and *R1N*). When the MFs have 1s on their inputs *R0N* and *R1N*, they set their outputs *G0* and *G1* to 0, represented by the state *00:00* in Figure 3.2. For a transition on a single input *R0/R1*, with the other input remaining at 0, one of gates *U1/U2* respectively writes 0 in *R0N/R1N*. Whenever one of the inputs of an MF (*R0N/R1N*) goes to 0, the MF writes 1 in the corresponding output (*G0/G1*). This appears, for instance, in the sequence of transitions *00:00*$\rightarrow$*10:00*$\rightarrow$*10:10*. Once one output is at 1, a new low-to-high transition

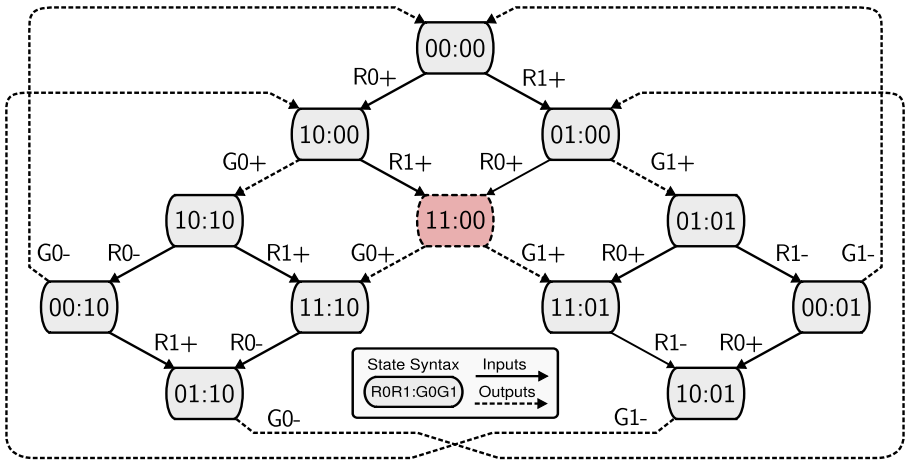


Figure 3.2 – State diagram of a MUTEX, using the state order R0R1:G0G1. Solid lines correspond to primary input transitions, while dotted lines stand for primary output transitions [ZHM+15].

on the other input of the MUTEX will not affect any of the outputs. This occurs because at these states *R0N/R1N* is at 0, which masks low-to-high transitions on *R1/R0*. This happens for example, in states *11:10* and *11:01*, where a state transition will only occur if either *R0* or *R1* switch to 0, in which case the MUTEX releases the asserted output, switching it back to 0.

Separate events on *R0* and *R1* are easily treated by the cross-coupled NAND gates. However, if both inputs switch within a time window sufficiently small, this creates a condition where both *U1* and *U2* are trying to switch their outputs to 0. Under such condition, the NANDs can be viewed as a loop of two inverters where both *R0N* and *R1N* will be lowered to a voltage close to $V_{DD}/2$ until one of them overpowers the other completely, switching *R0N/R1N* to 0, and *R1N/R0N* to 1. The problem is that *R0N* and *R1N* can stay at a metastable voltage for an unbounded period of time, until the NANDs resolve their outputs. If this metastability propagates to other circuits it can have unpredictable effects. This is where the MF plays a fundamental role. This component ensures that the metastable states do not propagate to the outputs of the MUTEX. In other words, it keeps *G0* and *G1* low until *R0N* and *R1N* settle to valid logic level voltages. The dashed state *11:00* marked in red in Figure 3.2 shows this last scenario. After *R0N* and *R1N* settle, either *G0* or *G1* will switch to 1. Note thus that the state diagram represents the non-determinism of a MUTEX behavior.

3.1.2 The C-element

Another popular component in asynchronous templates is the Muller C-element [MB57]. In fact, it is a fundamental component in asynchronous circuits design and different asynchronous templates report requiring only C-elements other than MUTEXes as

Table 3.1 – Basic 2-input C-element truth table.

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

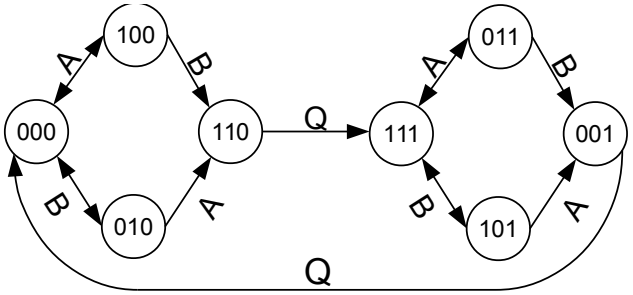


Figure 3.3 – Basic 2-input C-element state diagram.

specialized components [SF01a]. Among these templates, we highlight the BD template Mousetrap [SN07] and the QDI template dual-rail RTZ WCHB/DIMS [MN06], the latter is explored in detail in Section 5.1.1. Its properties make the C-element crucial in the implementation of asynchronous circuits control and it is typically employed for synchronization of events in asynchronous design. Table 3.1 shows the simplest possible truth table of a 2-input C-element, while Figure 3.3 depicts the associated behavioral diagram. When inputs A and B have the same value, the C-element output Q assumes this same value. However, when the inputs are different, the output keeps its previous logic value.

C-elements are typically handcrafted by asynchronous designers and can assume different transistors topologies. Figure 3.4 shows the basic symbol usually associated to this component and three different topologies for implementing it. One simple implementation is the semi-static topology, or Martin's weak feedback C-element, as addressed in some works available in the literature, Figure 3.4 (c). This implementation consists of pull-down and pull-up networks that generate the conditions where the inputs drive the output and a memory scheme for maintaining the output value when the input combination should not drive the output. When both inputs are 1, the pull-down network is activated, writing a 0 in the internal node and, consequently, a 1 in the output. Similarly, when both inputs are 0, the pull-up network is the one activated, and writes 1 in the internal node and a 0 in the output. Whenever the inputs have different values, no value is driven in the internal node, and the previous output value is kept by the loop of inverters.

The other two implementations are: the static topology, or Sutherland's pull-up pull-down, in Figure 3.4 (a); and the symmetric topology, or van Berkel's C-element, in Figure 3.4 (b). The former was proposed by Sutherland in [Sut89] and employed in his Micropipeline template. The latter was proposed by van Berkel in [Ber92]. These topologies have a similar

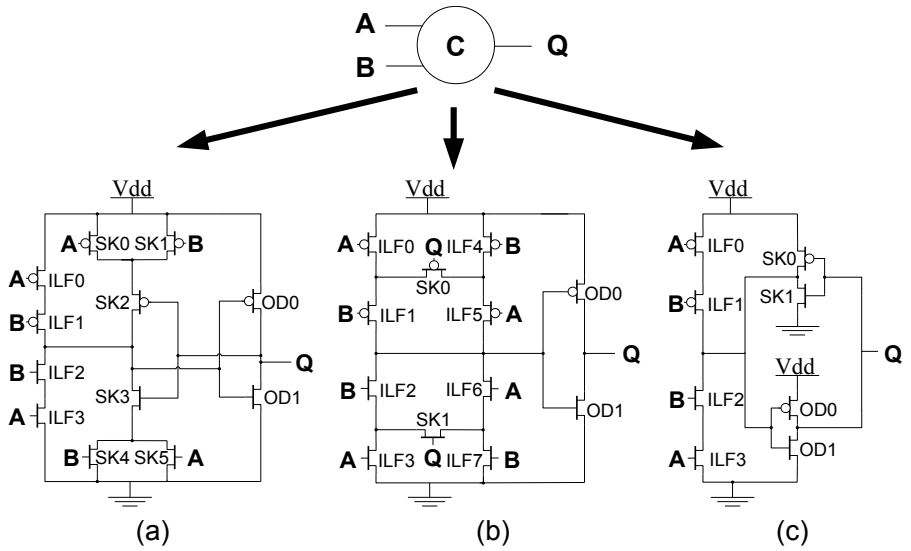


Figure 3.4 – Three classic implementations of the C-element: (a) static (Sutherland), (b) symmetric (van Berkel), (c) semi-static (Martin). Adapted from [MOMC12].

property of switching off their memory scheme while the C-element is switching its output. For example assume that both inputs of the C-element are at 0, the internal node is at 1 and the output is also at 0. In this case, both topologies will have their memory schemes keeping the internal node at 1, through transistors *SK0*, *SK1* and *SK2* in the static C-element and through transistors *ILF0*, *ILF1*, *ILF4*, *ILF5* and *SK0* in the symmetric C-element. As soon as the inputs switch to 1, the C-elements will start to write a 0 in the internal node, which will push the output to 1, at the same time the memory schemes are disabled. In the static C-element, this happens because when both inputs are at 1, transistors *SK0* and *SK1* are turned off, avoiding the memory scheme to keep writing a 1 in the internal node. In the symmetric C-element, this happens because when both inputs are at 1, transistors *ILF0*, *ILF1*, *ILF2* and *ILF3* are turned off, avoiding the memory scheme to keep writing a 1 in the internal node.

Using these topologies, different arrangements of the basic C-element can be devised, including multiple functional inputs and control set and reset inputs. A reset input sets the output to low regardless the value of the other inputs, while the set input sets it to high. Moreover, it can have some variations concerning to the symmetry of its functional inputs. Such properties can be used for constructing logic blocks of specific templates, as discussed in [YR06]. However, they are not explored here, because these templates are out of the context of this Thesis.

3.1.3 NCL Gates

NCL gates are the building blocks of the NCL template, proposed by Fant and Brandt in [FB96], as described in Section 5.1.2. These components are often called threshold gates, but this is imprecise because they do not exactly implement threshold logic functions (TLFs) [Hur69], as in threshold gates. Rather, they implement modifications of TLFs coupled to specific mechanisms to ensure completeness of input criteria [FB96]. Before defining TLFs and NCL gates, we need to be precise about the notion of unateness [BSVMH84], as all TLFs are by definition unate functions [Hur69].

Definition 1. *A logic function* $f(x_0, x_1, ..., x_{n-1})$ *is said to be* positive unate *in* x_i*, for some* i *such that* $0 \leq i \leq n-1$*, if* $f(x_0, ..., x_{i-1}, 1, x_{i+1}, ..., x_{n-1}) \geq f(x_0, ..., x_{i-1}, 0, x_{i+1}, ..., x_{n-1})$*, for all possible combinations of values of* x_j *with* $j \neq i$*. Similarly, a logic function is said to be* negative unate *in* x_i *if* $f(x_0, ..., x_{i-1}, 0, x_{i+1}, ..., x_{n-1}) \geq f(x_0, ..., x_{i-1}, 1, x_{i+1}, ..., x_{n-1})$*, for all* x_j*, with* $i \neq j$*. If a logic function is neither positive nor negative unate in* x_i*, it must necessarily be binate in* x_i*. Moreover, if a function is positive or negative unate in all its variables it is called a* unate function*. A unate function that is positive (resp. negative) unate in all its variables is simply called a* positive (resp. negative) unate function*. Also, throughout this Thesis those gates that implement positive (negative) unate functions will be called positive (resp. negative) unate gates.*

It is now possible to define a TLF.

Definition 2. *A* threshold logic function *or* TLF $t(x_0, x_1, ..., x_{n-1})$ *is an n-variable unate function defined by a threshold value* T *and specific integer weights* w_i *assigned to each variable* x_i *such that:*

$$t = \begin{cases} 1, & \sum_{i=0}^{n-1} w_i x_i \geq T \\ 0, & \sum_{i=0}^{n-1} w_i x_i < T \end{cases} \tag{3.1}$$

Also, according to [Hur69, Mur71], all TLFs are unate functions, but not all unate functions are TLFs. In this way, TLFs cannot be binate functions.

Based on these definitions, an NCL gate can now be defined as:

Definition 3. *An* NCL gate *is an n-input logic gate with a threshold* $T \in N^*$*, a specific weight* $w_i \in N^*$ *assigned to each Boolean input* x_i*, and a hysteresis mechanism to ensure a*

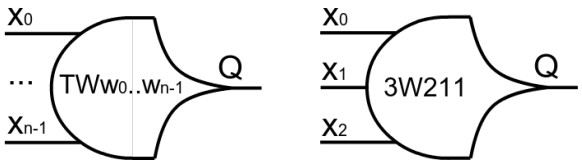


Figure 3.5 – Symbology for NCL gates: (a) generic NCL symbol; (b) symbol of an example NCL gate with 3 inputs, threshold T=3, and weights w0=2, w1=1 and w2=1. Note that the implicit ordering of inputs in the symbol is from top to bottom, associated with the list of weights from left to right.

Table 3.2 – Truth table of an example NCL gate with 3 inputs with threshold 3 and weights 2, 1 and 1. Note that *i* denotes the current instant of time.

X_0	X_1	X_2	Q_i
0	0	0	0
0	0	1	Q_{i-1}
0	1	0	Q_{i-1}
0	1	1	Q_{i-1}
1	0	0	Q_{i-1}
1	0	1	1
1	1	0	1
1	1	1	1

sequential behavior, such that its output Q_i at each instant of time i is given by:

$$Q_i = \begin{cases} 1, & \sum_{j=0}^{n-1} w_j x_j \geq T \\ 0, & \sum_{j=0}^{n-1} w_j x_j = 0 \\ Q_{i-1}, & 0 < \sum_{j=0}^{n-1} w_j x_j < T \end{cases} \tag{3.2}$$

Note that the portions of Equation 3.2 that set the output to 0 and 1 are usually known as reset and set functions, respectively.

Figure 3.5(b) shows the basic NCL gate symbol, where n is the number of inputs of a gate, T is the threshold of the underlying TLF of the gate and each input has a weight w_i. If a weight is omitted, w_i=1 is assumed. Weights always come after the *W* specifier. For example, Figure 3.5(b) shows the symbol of a 3-input NCL gate with threshold 3 and respective weights 2, 1 and 1. Table 3.2 shows the truth table for the gate in question, computed from Equation (3.2). Accordingly, the output of the gate will only switch to 0 when all inputs are at 0. Also, because x_0 has weight 2, x_1 and x_2 have weight 1 and the threshold is 3, the gate will only switch to 1 when x_0 is at 1 and at least one of the other inputs is at 1. In all other cases the output remains unchanged.

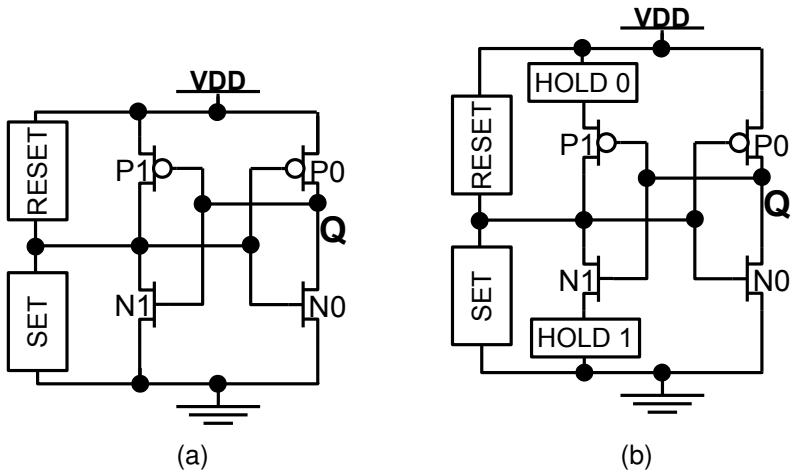


Figure 3.6 – NCL topologies: (a) semi-static and (b) static. Adapted from [MAGC14].

For implementing NCL gates at circuit level most works rely on the static and the semi-static C-element topologies presented in Section 3.1.2, because they can be adapted to the design of such gates. The difference is that the topologies are typically called static and semi-static, respectively. In the classic semi-static topology, presented in Figure 3.6(a), the reset and set functions correspond to pull-up (*RESET*) and pull-down (*SET*) networks, followed by an output inverter. *RESET* detects when all *n* inputs are 0, corresponding to a series of *n* PMOS transistors. Note that the output *Q* is then inverted by output inverter composed by P0 and N0. *SET* depends on the gate threshold *T*. Also, to ensure delay insensitivity, the gate keeps its output value when neither *RESET* nor *SET* functions are true. Since these are not complementary, the static gate requires a feedback inverter, composed by *P1* and *N1*.

The static topology, presented in Figure 3.6(b) is similar to the semi-static one. However, its feedback inverter is controlled by pull-up (*HOLD0*) and pull-down (*HOLD1*) networks. The former is the complement network of *SET* and the latter is the complement network of *RESET*. This allows avoiding that the feedback inverter interferes during output switching.

Note that C-elements are actually a special case of NCL gates where the threshold is the same number of inputs and all inputs have weight 1. Furthermore, as described by different authors, sometimes it is useful to implement special functions in NCL gates, other than threshold functions. In these cases, the ON-set of the gate is substituted by such special function. A case for special functions is the ANDOR function, which is basically a product of the sums of two pairs of inputs, *i.e.* the sum of A and B with C and D.

3.1.4 Discussion

There is a vast diversity of templates for asynchronous design in the state-of-the-art. This Thesis focuses mainly on a specific set of QDI templates, as it will be explored in Chapter 5. In this way, we limit our exploration of components to those that are required by the addressed templates. Among the other sets of components available in contemporary literature, we highlight the following for their relevance in asynchronous design: (i) PCHB gates [Lin98], used in PCHB designs in industry [DLD+14] and academia [BDL11]; (ii) single-track full buffer (STFB) gates [Fer04], used in practical applications and validated on silicon; (iii) Toggle [Sut89], used in the classic Micropipelines, which was the inspiration for many of the recent BD templates [NS11]; (iv) and Q-modules [RMCF88], a set of components proposed in the late 1980s, with the potential of allowing metastable free operation and testability in asynchronous design, recently used in a new BD template.

Another C-element topology that we do not consider in this work is the dynamic C-element [BOF10]. The reason why we decided to discard it is due to its dynamic behavior that imposes timing constraints that are not suitable to the templates we address. These constraints are due to the fact that it does not contain static storage and when inputs have different logic values it cannot keep its state for unbounded periods of time. As for NCL gates design, besides the classic static and semi-static topologies, a recent work, presented in [PS12b], proposes adapting the symmetric C-element topology to be used in NCL gates design. However, just a small portion of NCL gates can be implemented using this topology and cannot be generalized, requiring gate specific schematic implementation. In this way, we discard this topology in this work, as it does not present a general schematic and requires gate specific designs. We also discarded the multi-threshold topology proposed in [BDSM08] because they target a QDI template that is out of the scope of this Thesis.

Albeit there are different manners of constructing the components that we address here, there is no consensus on contemporary literature on what is the best way to build them. In our exploration on asynchronous design, the first step was to build such components, so that we can use them in more complex systems. In this way, before adopting a specific template for constructing these components, we performed a comprehensive analysis of their performance metrics. From these analyses we derive a set of guidelines for building sets of components that support the design of asynchronous cell libraries, as explored in Chapter 4. Note that we included an explanation of MUTEX components here because they are in the composition of the library presented in Section 4.3. However we did not explore optimizations for these components in the context of this Thesis.

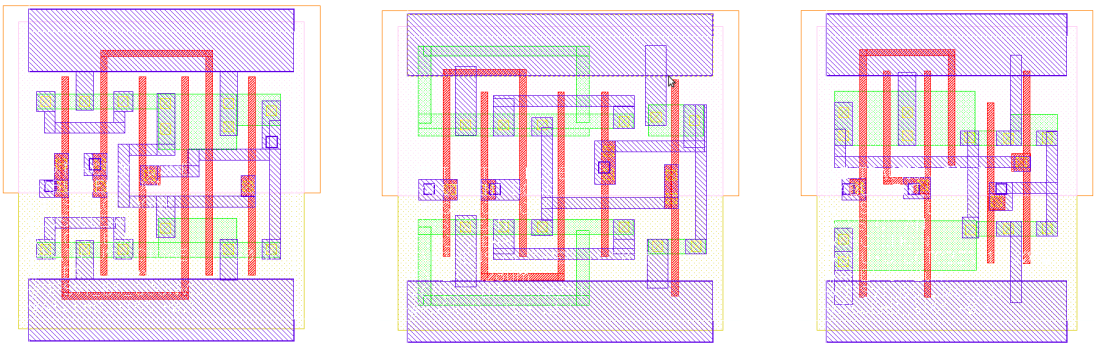


Figure 3.7 – Example physical layout at the cell level of the designed C-elements: (a) static, (b) symmetric, (c) semi-static. Adapted from [MOMC12].

3.2 C-elements Design

We start exploring the design of C-elements, classic components that are required in the majority of asynchronous circuits design templates. This section explores area, power and delay trade offs between different topologies for building these components and their suitability to low voltage operation.

3.2.1 Area, Power and Delay Trade Offs

To efficiently implement C-elements, the first step is to understand their electrical characteristics. To do so, each of the three classic implementations of the C-element, showed in Figure 3.4 was designed at transistor level. We employed general purpose standard threshold transistors for the 65nm STMicroelectronics CMOS technology. The generated designs were automatically implemented with an in-house design flow that will be explored in Chapter 4. To ensure a fair analysis, the designs are able to drive the same load with the same speed, a maximum of 270 fF in 1 ns. In other words, the driving strength of each implementation is normalized, in order to precisely compare performance and area efficiency in a fair manner. Figure 3.7 shows an example layout for each of the designed standard cells.

The silicon areas required by each C-element design appear in Table 3.3. The design that requires less area is the semi-static C-element. Table 3.3 also shows the resultant internal parasitic after RC extraction. As expected, the symmetric design, implementation that requires more silicon area, presented highest parasitic capacitance, over two times the parasitic of the semi-static C-element.

After electrical extraction, each cell was characterized for a typical fabrication process corner, with typical delay for the NMOS and PMOS transistors, for an operational con-

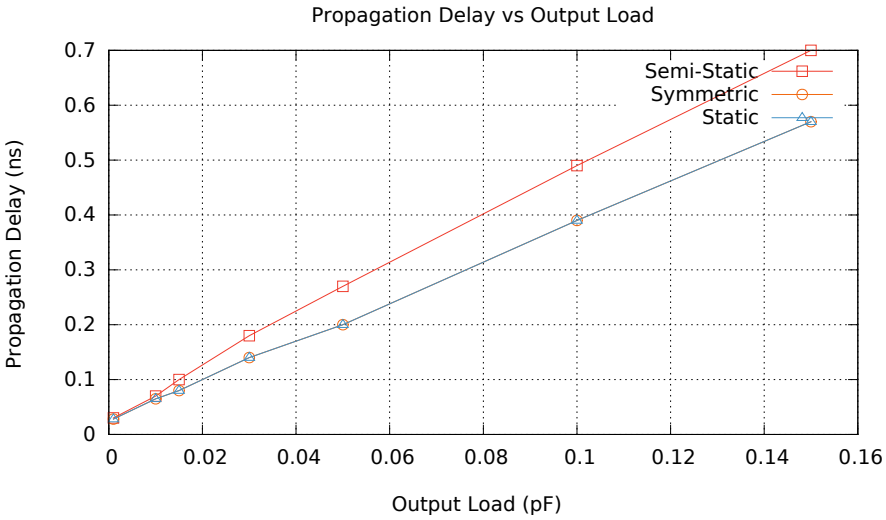
Table 3.3 – Area, parasitic capacitance, input capacitance and average dynamic and leakage power of the C-element implementations. Adapted from [MOMC12].

Topology	Cell Area (μm^2)	Par. Cap. (fF)	In. Cap. (fF)		Avg. Dyn. Pwr. (fW)		Avg. Leak. Pwr. (nW)
			A	B	Rise	Fall	
Static	6.24	6.066	4.565	4.410	1.316	12.454	20.447
Symmetric	7.28	9.383	3.113	3.081	1.190	12.871	17.628
Semi-static	5.72	4.469	5.633	5.849	4.889	21.930	30.591

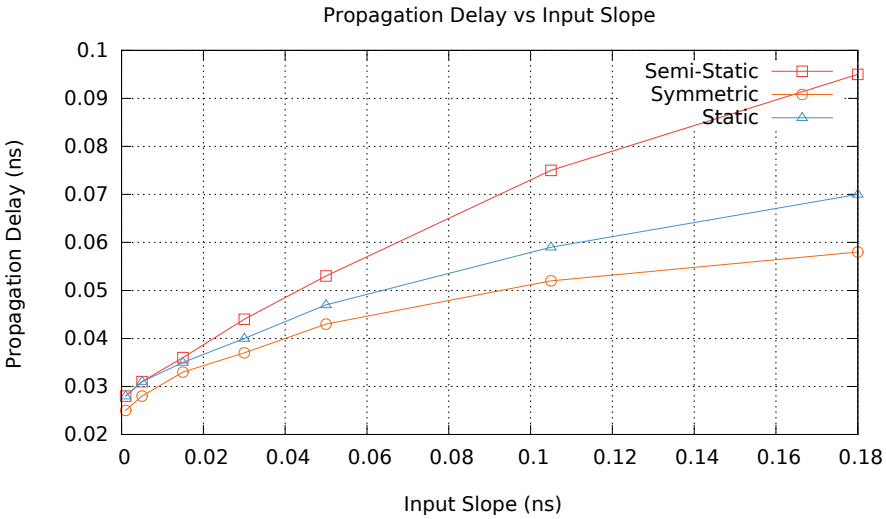
dition of 25 degrees Celsius and 1 V power supply. Table 3.3 also shows electrical results obtained through the electrical characterization. The semi-static design presents the highest capacitance on its inputs. In fact, in comparison with the symmetric implementation, which presents lower input capacitance, it shows an overhead of 85%. That is due to the fact that, albeit the symmetric implementation is the most area consuming, due to the elevated number of required transistors, the semi-static C-element is the one that employs larger transistors, as Figure 3.7 shows. As for the dynamic power, the static and the symmetric implementations are equivalent for rise and fall transitions. The former consumes slightly less, roughly 2% in average. Moreover, the semi-static design presents an overhead of roughly 300% for the dynamic power for rise transitions and almost 100% for fall transitions. As expected, semi-static implementations have the highest leakage power, when compared to the others. This is also a consequence of transistor size.

The average propagation delay of each design, was also obtained through electrical characterization. Figure 3.8 shows the obtained results for two scenarios. In Figure 3.8(a), the input slope was fixed in 1.2 ps and the output load varied from 0.001 pF to 0.15 pF. The time required for the static and the symmetric implementations to switch their respective outputs is equivalent, as illustrated by the overlapping values. Moreover, the semi-static design presents equivalent propagation delay for small output loads (from 0.001 pF to 0.015 pF). However, the higher the load gets, the worse its propagation delay is, in comparison to the other implementations. This behavior shows that the semi-static is also the implementation most sensitive to output load variations.

Figure 3.8(b) shows the electrical behavior of each implementation when the output load is fixed in 1 fF and the input slope varied from 0.0012 ns to 0.180 ns. In this scenario, the fastest implementation is the symmetric, followed by the static and next the semi-static. The obtained results show that for small input slopes (0.0012 ns to 0.0132 ns), the speed of static and semi-static C-elements is equivalent. However, as the input slope gets more significant, the propagation delay of the semi-static C-element gets worse. In this way, the semi-static implementation is, also, the most sensitive to input slope variations. As Figure 3.8(b) shows, the delay of this implementation grows at a much higher rate as the input slope grows, while the other two implementations see their delay grow more linearly. The symmetric implementation displays the smallest propagation delay, regardless of input slope variations. Therefore, we can consider the symmetric C-element the most robust implementation for



(a)



(b)

Figure 3.8 – Propagation delay of the designed C-elements after electrical extraction, as a function of the (a) output load capacitance and (b) input slope. In (a), results were obtained by fixing the input slope in 1.2 ps and varying the output load from 0.001 pF to 0.15 pF. In (b), results were obtained by fixing the output load in 1 fF and varying the input slope load from 0.0012 ns to 0.180 ns. Adapted from [MOMC12].

both input slope and output load variations. The drawback, though, is that this topology is not scalable to multi-output cells, where the static C-element is the most suited topology.

As an initial comparison of the impact of each C-element implementation on asynchronous circuits, a low complexity circuit was described in the Spice language, an oscillator ring. As Figure 3.9 shows, the circuit is composed by a NAND and 10 C-elements. Extensive simulation defined the number of C-elements in the ring (10) as an amount sufficient to normalize the effect of the NAND and allow correct evaluation of the C-elements. The NAND is required to keep the circuit static, when the *IN* pin is set to 0, and to make the circuit oscillate, when the *IN* pin is switched to 1. In this way, static and dynamic power and

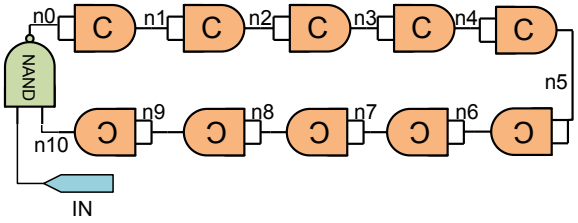


Figure 3.9 – Oscillator ring employed for an initial comparison of the impact of each C-element implementation in an asynchronous circuit. Adapted from [MOMC12].

Table 3.4 – Performance figures of the oscillator rings. Adapted from [MOMC12].

Topology	**Frequency (GHz)**	**Leak. Power (μW)**	**Dyn. Pwr (μW)**
Static	0.865	0.17	74.41
Symmetric	1.148	0.13	74.33
Semi-static	0.808	0.28	119.20

operational frequency can be precisely measured. Three oscillator rings were generated, one for each implementation of the C-element after RC extraction.

After simulating each oscillator ring, power and operational frequency, were obtained through the Spice *.measure* function, as Table 3.4 shows. Leakage power was measured as the voltage multiplied by the average current in the power source while the circuit was quiescent. The dynamic power was measured as the operating voltage multiplied by the average current in the source when the ring is oscillating. The frequency is measured as the inverse of the period between two similar edges in any node of the ring (in this case *n5*).

As Table 3.4 shows, the conducted experiment confirms the results obtained through the electrical characterization of the C-elements. The symmetric implementation presents the lowest leakage power, while its dynamic power is equivalent to the static C-element. The semi-static implementation presents higher dynamic and leakage power, as expected. Power figures enforce the statement that the semi-static is the most power consuming and the static and symmetric present similar power figures.

It would be expected that at least two of the rings, the ones composed by the static and the symmetric implementation, presented equivalent operating frequency. However, the one generated with symmetric C-elements operates roughly 32% and 42% faster than the one composed by static and semi-static C-elements, respectively. As Table 3.3 shows, the sum of the pin capacitances of the static C-element is 8.975 fF, while for the symmetric implementation it is only 6.194 fF. In other words, each cell of the ring in static implementations, except the one that drives the NAND, must drive a load roughly 44% bigger than that of the symmetric C-elements ring. Both implementations showed to be equivalently sensitive to output load variations. However, these variations interfere in the transition time of their output, which feeds the next cell in the ring. Thus, the slope in the input of the next cell increases. In this case, the resulting input slope generated in the inputs of each static

C-element, after the circuit stabilizes its oscillating frequency, is roughly 0.06ns. Considering that the static C-element is more sensitive to input slope variations, it is clear why the rings operate at different frequencies ranges.

For the semi-static C-element, the operational frequency range is even worse. That is due to the fact that this implementation presents not only higher input capacitance, which contributes for an elevated slope in the input of each cell of the ring, but it is also much more sensitive to input slope variations than the static C-element. Its performance would be yet worsened if, for instance, each C-element was required to drive a load bigger than 0.015 pF. See Figure 3.8(a), where the delay of the semi-static implementation starts to get worse than the other two. In this case, the sum of its input pins, load that each cell (except the one that drives the NAND) must drive, is 11.482 fF.

A more realistic comparison of the impact of the C-elements implementation was conducted through the design of a 32 bit RSA cryptographic core. The circuit was described in the Balsa language [Bar98] and synthesized through the Teak System [BTE09]. Teak automatically maps the Balsa description into a specific set of components, C-elements and conventional OR gates, generating asynchronous QDI circuits. Three versions of the asynchronous RSA cryptographic core were generated using Teak, each one employing exclusively one distinct C-element type implementation on its schematic. The OR gates were extracted from the core library available in the target design kit of STMicroelectronics. The choice for the RSA function was due to the fact that its algorithm employs arithmetic operations as well as control functions.

The total number of cells employed in all designs, without taking into account physical cells, was 57,168. Physical cells are the cells used to connect the power lines to the substrate, tap cells, and the filler cells employed in the core. The circuits had the same number of logic cells due to the fact that the only difference between them is the choice of C-element implementation and Teak does not optimize cells dimensions and employs a direct translation mapping approach. From the total cells, 22,063 were C-elements. This means that, for our case studies, roughly 40% of the cells were C-elements.

Table 3.5 shows the physical characteristics of the generated RSA cryptographic cores, obtained after place and route. The design implemented with semi-static C-elements requires less silicon area, while the ones implemented with symmetric and static C-elements were the largest. The area overhead imposed by both in comparison with the semi-static is roughly 12% and 7%, respectively. Therefore, the semi-static C-element is the most efficient implementation for high density designs and the symmetric C-element is the most area and wire consuming implementation. These results are in agreement with the information obtained at layout level.

The RSA netlists' delay of the paths generated after place and route were annotated and served as input to a set of simulations. These comprised multiple cryptographic operations for each netlist, collecting performance results. Employed operational conditions

Table 3.5 – Area and power results for the three asynchronous RSA cryptographic core implementations after place and route. Adapted from [MOMC12].

Metric	**Static**	**Symmetric**	**Semi-static**
Number of Cells	92,922	94,015	91,276
Cell Area (mm²)	0.295	0.311	0.276
Cell Area - Phys. Cells (mm²)	0.244	0.258	0.228
C-elements Cell Area (mm²)	0.161	0.175	0.145
Internal Power (mW)	1.878	2.161	4.361
Switching Power (mW)	1.581	1.433	1.342
Leakage Power (mW)	1.729	1.639	2.162
Total Power (mW)	5.188	5.233	7.865

were 25ºC, 1V supply for a typical fabrication process corner. The average delay to perform a cryptographic operation for the semi-static, the static and the symmetric C-element based implementations were 104.311 μs, 83.96 μs and 73.241 μs, respectively. These results are in agreement with the information obtained in the simulation of an oscillator ring. The symmetric implementation presented higher operating speed, due to the fact that it is the less sensitive implementation to input slope and output load variations and Teak synthesis is not able to optimize dimensioning of the selected cells. In other words, every cell employed in the circuit has the same output driving strength and input capacitance, some of these ending up overloaded. This is a limitation of the tool, which leads to slower designs mostly when employing static or semi-static C-elements, since these present higher delay for high output loads and input slopes.

From the simulations, the switching activity in the nets of each circuit was annotated for a period of 3 ms and served as input to evaluate power. Table 3.5 shows the information obtained for the three netlists. Employing static or symmetric C-elements generated circuits with similar power. Comparing these implementations, the latter consumed less leakage power, roughly 5%, while the former presented lower dynamic power, roughly 4%. The circuit generated with static C-elements presents slightly less total power. This is due to the fact that the power consumed while the circuit is quiescent represents a smaller portion of the total power than the dynamic power. Notably, semi-static was the less power efficient implementation.

These results are in agreement with those obtained at layout level and in the simulation of an oscillator ring, except for the dynamic power of the static C-element. In the first case study, this presented worse dynamic power consumption than the symmetric C-element. However, in that case, each cell was driving a single two-input cell, while in the circuit generated by Teak, the cells were required to drive multiple nets and, consequently, higher loads. In this scenery, the symmetric dynamic power efficiency was compromised.

Figure 3.10 details the power of the C-elements from the total power of the placed and routed netlists. The total power for the static, the semi-static and the symmetric C-

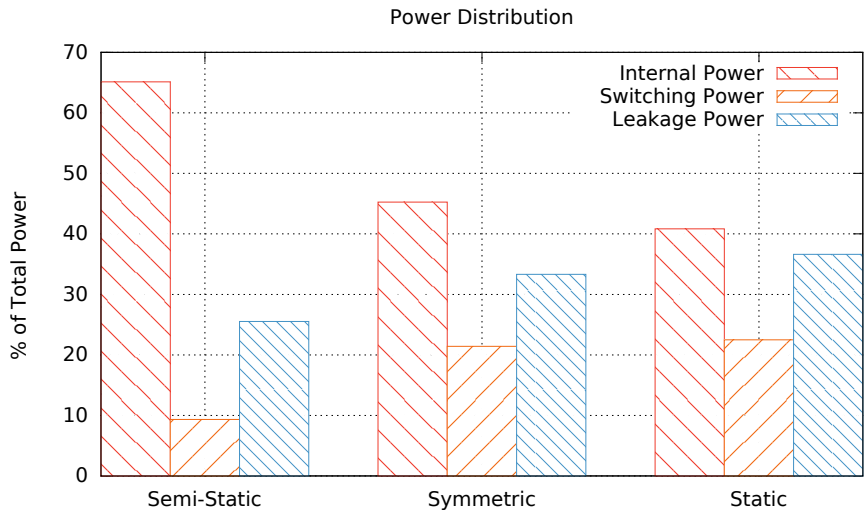


Figure 3.10 – C-elements power consumption for each asynchronous RSA cryptographic core implementation. Adapted from [MOMC12].

elements, in their respective netlists, was 2.803 mW (54%), 5.722 mW (73%) and 2.815 mW (54%), respectively. These results show that, in a realistic application, the reason for the static C-element to consume less power than the symmetric is because the internal power in the latter is more significant. One aspect that worsens internal power is the amount of transistors in short circuit when switching the symmetric C-element output logical value. Moreover, the bigger the input slope is, the bigger is the period of time that the transistors are in short circuit when switching the output of the C-element.

These results show that, in summary, albeit the symmetric C-element appears as the lowest static power implementation, it presents more dynamic power than the static C-element for bigger input slopes. In this way, there is a trade off between these two topologies in terms of static and dynamic power. The semi-static presented the worst propagation delay, regardless output load or input slope variations. Moreover, the symmetric and the static C-elements proved to be equally robust to output load variations. However, the static implementation presented higher propagation delay for high input slopes. Therefore, the symmetric C-element appears as the most speed efficient implementation. The results on required area for each C-element, showed that the symmetric implementation is the most silicon area consuming, while the semi-static is the most area efficient. Hence, the latter is the most suitable for high density designs.

3.2.2 Low Voltage Operation

Given the relevance of C-elements in asynchronous design, a next step was to evaluate their behavior under low voltage operation, as explored in [MC13]. To do so, we relied

on the three topologies evaluated in Section 3.2.1. We designed them targeting the same technology with 5 different driving strengths: X2, X4, X7, X9 and X13. All the experiments reported here are based on RC layout extracted views for a typical process.

Table 3.6 – Minimum voltage for maintaining correct functionality of the semi-static C-element for different driving strengths at varying temperatures . Adapted from [MC13].

	125 C	**100 C**	**75 C**	**50 C**	**25 C**	**0 C**	**-25 C**	**-50 C**
X2	0.15	0.15	0.15	0.2	0.5	0.6	0.65	0.65
X4	0.15	0.15	0.15	0.2	0.5	0.6	0.65	0.7
X7	0.15	0.15	0.15	0.2	0.45	0.55	0.6	0.65
X9	0.15	0.15	0.15	0.2	0.2	0.35	0.5	0.6
X13	0.15	0.15	0.15	0.2	0.2	0.25	0.45	0.5

Table 3.7 – Minimum voltage for maintaining correct functionality of the static C-element for different driving strengths at varying temperatures . Adapted from [MC13].

	125 C	**100 C**	**75 C**	**50 C**	**25 C**	**0 C**	**-25 C**	**-50 C**
X2	0.15	0.15	0.15	0.15	0.2	0.2	0.25	0.25
X4	0.15	0.15	0.15	0.15	0.2	0.2	0.25	0.25
X7	0.15	0.15	0.15	0.15	0.2	0.2	0.25	0.25
X9	0.15	0.15	0.15	0.15	0.2	0.2	0.25	0.25
X13	0.15	0.15	0.15	0.15	0.2	0.2	0.25	0.25

Table 3.8 – Minimum voltage for maintaining correct functionality of the symmetric C-element for different driving strengths at varying temperatures . Adapted from [MC13].

	125 C	**100 C**	**75 C**	**50 C**	**25 C**	**0 C**	**-25 C**	**-50 C**
X2	0.15	0.15	0.15	0.15	0.2	0.2	0.25	0.25
X4	0.15	0.15	0.15	0.15	0.2	0.2	0.25	0.25
X7	0.15	0.15	0.15	0.15	0.2	0.2	0.25	0.25
X9	0.15	0.15	0.15	0.15	0.2	0.2	0.25	0.25
X13	0.2	0.15	0.15	0.15	0.2	0.2	0.25	0.25

The first experiment detected the minimum voltages that can be applied to each C-element without interfering in their correct behavior. The experiment investigated scenarios for varying temperatures and a fixed FO4 output load. Minimum voltages were estimated by simulating all transition arcs of each C-element for each temperature/voltage scenario. When at least one arc does not generate the correct output or a static state is not able to maintain correct functionality, the scenario is defined as not functional. Also, the generated signals must have voltages respecting noise margins, for logic 1 (from 90% to 100% of the power supply) or for logic 0 (from 0% to 10% the power supply). If a signal presents a voltage level in the undefined region (from 10% to 90%), the scenario is also defined as not functional. In summary, the minimum voltage is defined as the lowest voltage at which the C-elements can operate without jeopardizing their correct logical/electrical behavior. The obtained results are summarized in Tables 3.6, 3.7 and 3.8, where six drives are analyzed.

Clearly, the higher the temperature is, the lower is the minimum operating voltage. Results suggest that the static and the symmetric C-elements are typically preferable for operating with low voltage supply, as they tolerate lower voltages than the semi-static. These results can be explained analyzing the transistors arrangement of each C-element implementation. Recalling Figure 3.4, in the semi-static C-element, there is a conflict-solving situation for every output transition. For instance, suppose that inputs *A* and *B* are at 0. In this case, transistors *ILF0* and *ILF1* are conducting and transistors *ILF2* and *ILF3* are turned off, generating a direct path the connects internal nodes nd0 and Vdd. Thus, the output inverter (*OD0* and *OD1*) is writing 0 to the output *Q* and the feedback inverter (*SK0* and *SK1*) is maintaining the output value stable, writing 1 to the internal node through the direct path to Vdd created by *SK0*.

Now, assume that input *B* switches to 1. At this point, there is no connection from the internal node to Vdd or Gnd, because *ILF1* and *ILF3* are both cut off. Then, the value is kept by the feedback inverter. Next, assume that input *A* also switches to 1, making *ILF3* conduct and creating a direct path from the internal node to Gnd. However, there is also a direct path from the internal node to Vdd that is still active (through *SK0*). There is thus an instantaneous short circuit, caused by the path from Vdd to Gnd through conducting transistors *ILF2*, *ILF3* and *SK0*. In this case, the C-element operates correctly when the resistivity of the path composed by the NMOS transistors, *ILF2* and *ILF3*, is smaller than the path crossing the PMOS, *SK0*. Similarly, to ensure correct behavior when the output switches to 0, the path formed by *ILF0* and *ILF1* must have smaller resistivity than the path crossing *SK1*.

Typically, transistors of the feedback inverter are designed with minimum size, to reduce their interference in the functionality of the C-element, while transistors *ILF0-3* are larger, to drive the output inverter. The bigger the driving strength is the bigger these transistors need to be. The transistors of the semi-static C-elements in this case study were designed to guarantee the correct behavior at typical voltages. Given the strict relationship between the dimensions of the transistors of this topology, operating parameters, like voltage and temperature, have a strong influence on the behavior of the design. As Table 3.6 shows, at room temperature (25 C) the minimum voltage for X2 and X4 drives is 0.5 V, for the X7 it is 0.45 V and for the X9 and X13 it is 0.2 V. Note that as the driving strength is increased, the design can operate at lower voltages. This is because larger driving strengths employ larger *ILF0-3* transistors, while maintaining the feedback inverter minimum sized.

The static and symmetric C-elements are not susceptible to this strict sizing relationship that arises in the semi-static C-element. This is because when these C-elements are switching their respective outputs, the feedback inverter is cut-off, see Section 3.1.2. In fact, as Tables 3.7 and 3.8 show, these topologies can typically tolerate lower voltages and wider ranges of temperature variation. Furthermore, the drive does not have the same effect

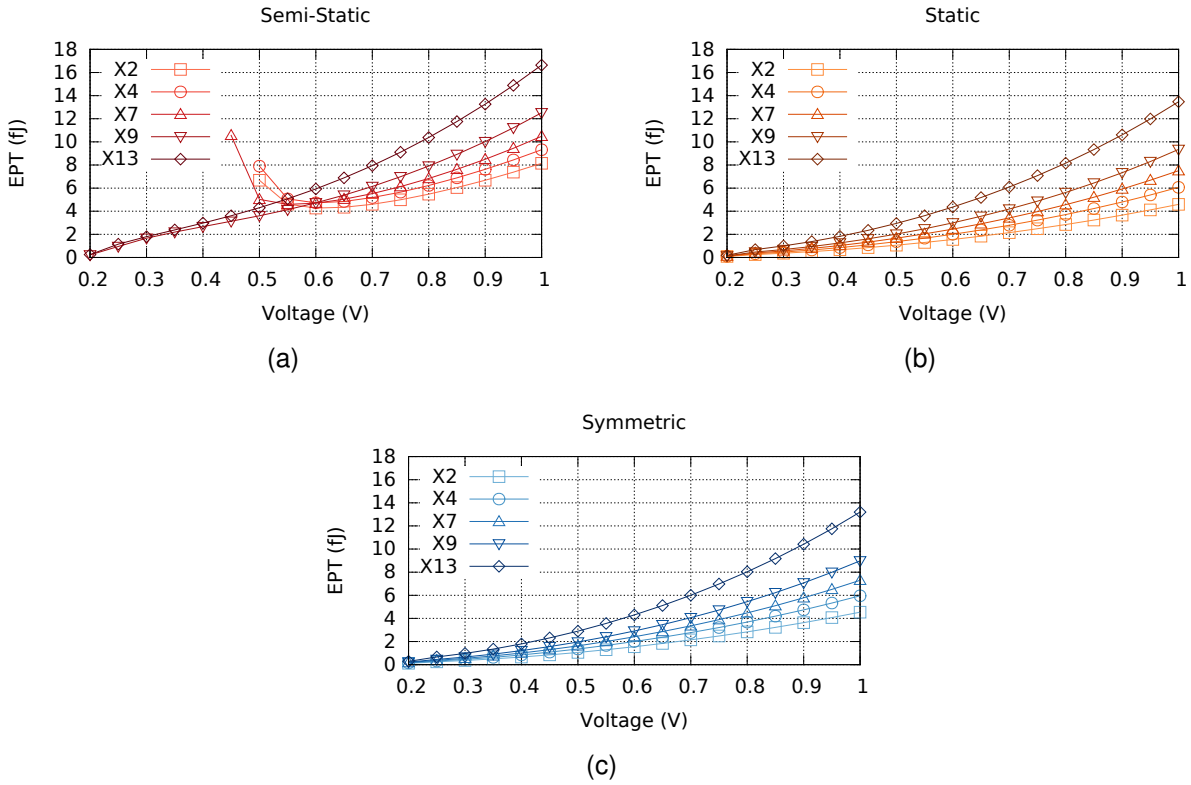


Figure 3.11 – Average EPT for varying voltage supplies for each drive of the three C-elements: (a) semi-static, (b) static and (c) symmetric. Adapted from [MC13].

in static and symmetric as in semi-static. Thus, the former are better for semi-custom low voltage design.

Another experiment measured the energy consumption and the propagation delay for each transition arc of the C-elements. Leakage power was also measured considering for all static states. In this case we assumed a fixed temperature of 25 C, as other temperatures do not change the results qualitatively, only quantitatively. Figure 3.11 shows the measured energy per transition (EPT) for each drive of each C-element implementation varying the supply voltage. We present EPT as the average of the energy consumed by all arcs of each implementation. For the semi-static C-element, in drives X2, X4 and X7, the lowest operational voltage presents very high EPT. This is due to the conflict condition. Albeit the resistivity of the paths is balanced well enough to provide correct functionality, it keeps the conflict for a relatively long period, which leads to excessive energy consumption. Fine grain optimizations in transistor dimensions could improve the obtained results. However, this complicates cell design as it compromises its behavior at higher voltages.

By analyzing the charts of Figure 3.11, we observe that the static and symmetric C-elements present lower EPT than the semi-static for a same drive in all cases, roughly 20%. Again this is dues to their mechanism for cutting off the feedback during output transition arcs. Also, the symmetric designs present EPT values slightly lower than the static ones. Similarly, the measured average leakage power for all static states appears in Figure

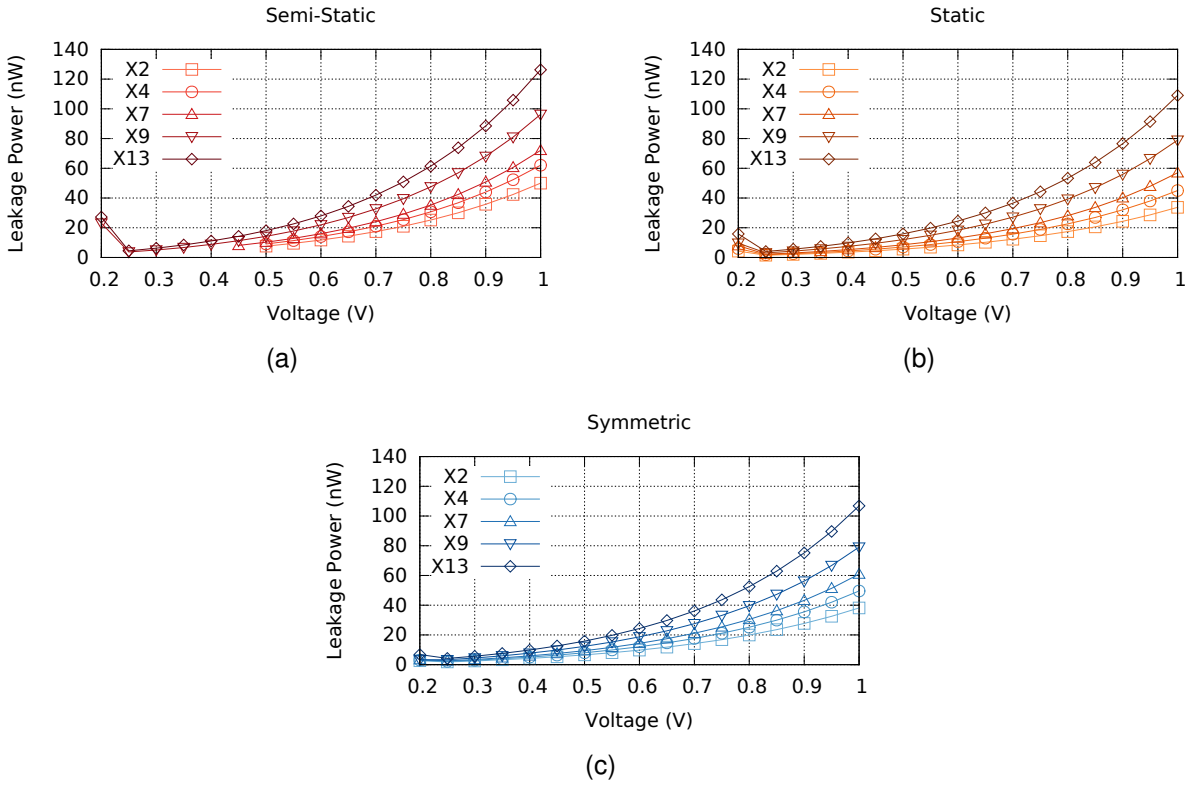


Figure 3.12 – Average leakage power for varying voltage supplies for each drive of the three C-elements: (a) semi-static, (b) static and (c) symmetric. Adapted from [MC13].

3.12, for each C-element. As the charts show, the semi-static implementation is the one that presents larger leakage power values, while static and symmetric C-elements present equivalent results. These results are a reflection of the size of transistors in each implementation. Because of the conflict during transition arcs, the semi-static C-element requires larger transistors, which leads to its excessive leakage power. In this way, the obtained results suggest that the static and symmetric C-elements can lead to better energy and power characteristics, regardless the operating voltage.

The average number of Giga transitions per second (GTPS) serves here to define the relative speed of the explored C-element designs. The measurement of these values depends on the average propagation delay of all C-elements transition arcs. Figure 3.13 presents the results obtained for the three topologies. As the charts show, the measured GTPS for the semi-static and the static C-elements are similar, while for symmetric it is roughly 20% larger. This is due to the arrangement of transistors in the latter. When a symmetric C-element switches the output, two paths connect the internal node to Vdd or Gnd in parallel, see Figure 3.4. In semi-static and static topologies, this connection occurs through a single path.

Another perspective of the obtained results shows a fairer comparison of C-elements. Three cost-benefit functions were defined to evaluate speed, leakage, energy and area trade offs: speed-energy, speed-leakage and speed-area. The ratio between the measured GTPS

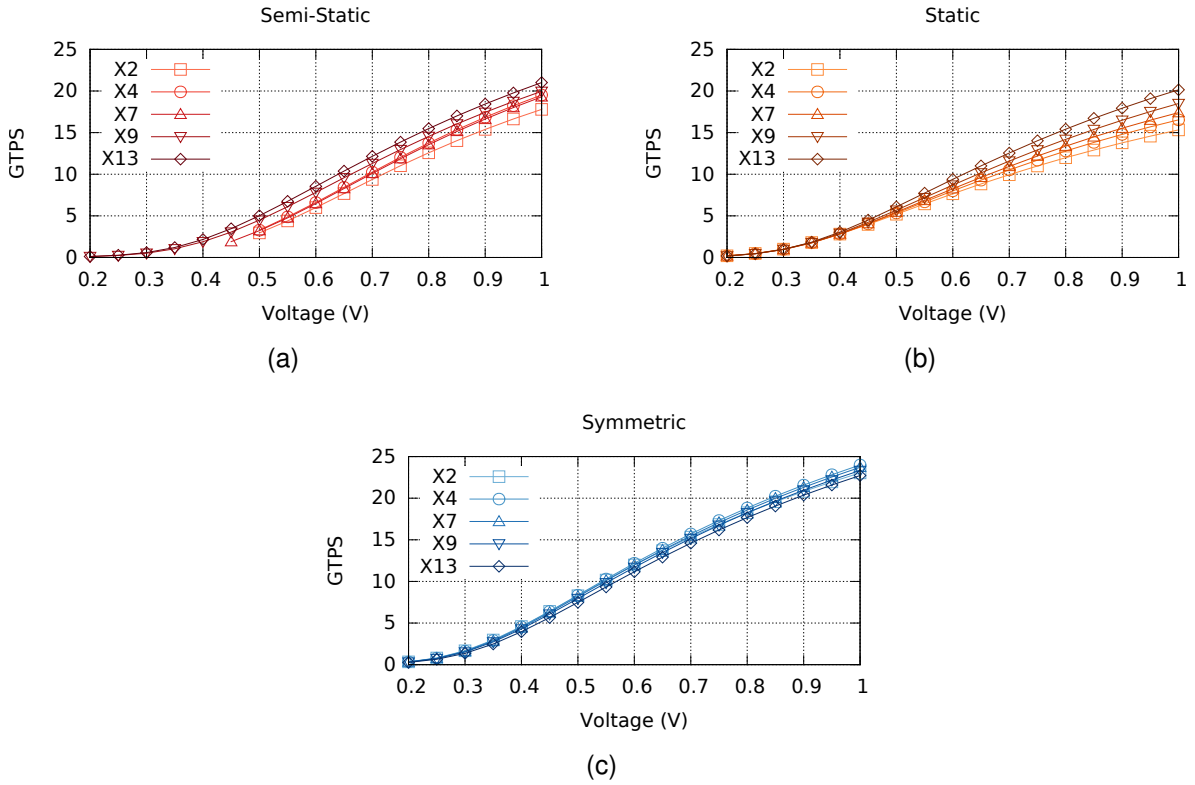


Figure 3.13 – Average GTPS for varying voltage supplies for each drive of the three C-elements: (a) semi-static, (b) static and (c) symmetric. Adapted from [MC13].

and EPT defines the speed-energy efficiency function. Using this, it is possible to evaluate the speed of the C-elements without overlooking the associated energy consumption. Figure 3.14 shows the speed-energy efficiency values, in GTPS/EPT, measured for all C-elements. As the charts show, the symmetric C-element is the one that presents highest GTPS/EPT values, followed by the static. The semi-static topology presented the worst speed-energy cost-benefit. The symmetric achieves optimizations of roughly 82%, in the best case, 46% in average and 11% in the worst case, when compared to static. In comparison to semi-static, these values are 700%, 240% and 75%, respectively.

Also, semi-static C-elements, in lower driving strengths (X2-X7), reach the measured optimum GTPS/EPT when operating at roughly 0.85 V. For higher driving strengths (X9 and X13), the optimum occurs at roughly 0.75 V. However, static and symmetric C-elements reach optimum power efficiency when supplied with 0.55 V, for all driving strengths. Moreover, as the charts in Figure 3.14 show, the lower the drive of the C-element is the best is its speed-energy efficiency. In this way, results suggest that energy optimizations can be achieved by employing low drive C-elements whenever feasible.

The ratio between the measured GTPS and leakage power (LKP) produces the speed-leakage efficiency function definition. With this function it is possible to evaluate the speed of C-elements without overlooking the associated leakage power. Figure 3.15 shows the speed-power efficiency values, in GTPS/LKP, measured for all C-elements considered

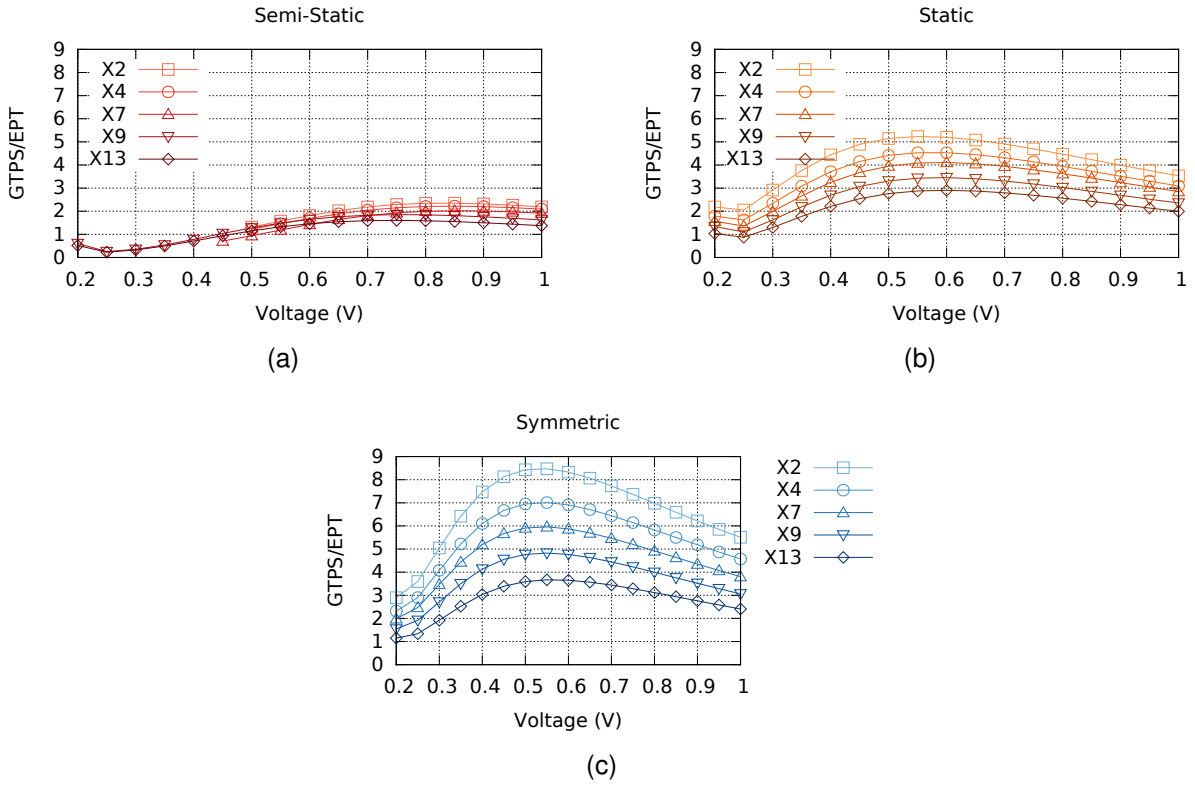


Figure 3.14 – Speed-energy efficiency for varying voltage supplies for each drive of the three C-elements: (a) semi-static, (b) static and (c) symmetric. Adapted from [MC13].

here. As the charts show, the symmetric C-element is again the best, since it presents highest GTPS/LKP, followed by the static. The semi-static implementation presents the worst speed-leakage cost-benefit. The symmetric design displays optimizations of roughly 51%, in the best case, 32% in average and 15% in the worst case, when compared to the static. In comparison to the semi-static, these values are 256%, 92% and 28%, respectively.

As Figure 3.15(a) shows, optimum efficiency for the semi-static C-element can be obtained at 0.7 V for lower driving strengths and at 0.6 V for higher driving strengths. Also, as Figure 3.15(b) and Figure 3.15(c) show, for static and symmetric, optimum efficiency is obtained at 0.55 V. In addition, similarly to the speed-energy efficiency, the lower is the drive of the C-element, the best is its speed-leakage efficiency, suggesting that improvements in the static power of circuits can again be obtained by employing low drive C-elements. The speed-area efficiency function allows evaluating the obtained speed for each implementation without ignoring silicon area. The ratio between GTPS and the total silicon area of each C-element defines this last evaluated function. Figure 3.16 shows the area of each implementation. As the chart shows, the higher the drive is the larger is the required area. The speed-area function was defined as the ratio between the GTPS and these area results for each C-element.

Figure 3.17 shows the obtained results. As the charts show, small drive semi-static C-elements are the ones that present best GTPS/Area values. This is expected, given their

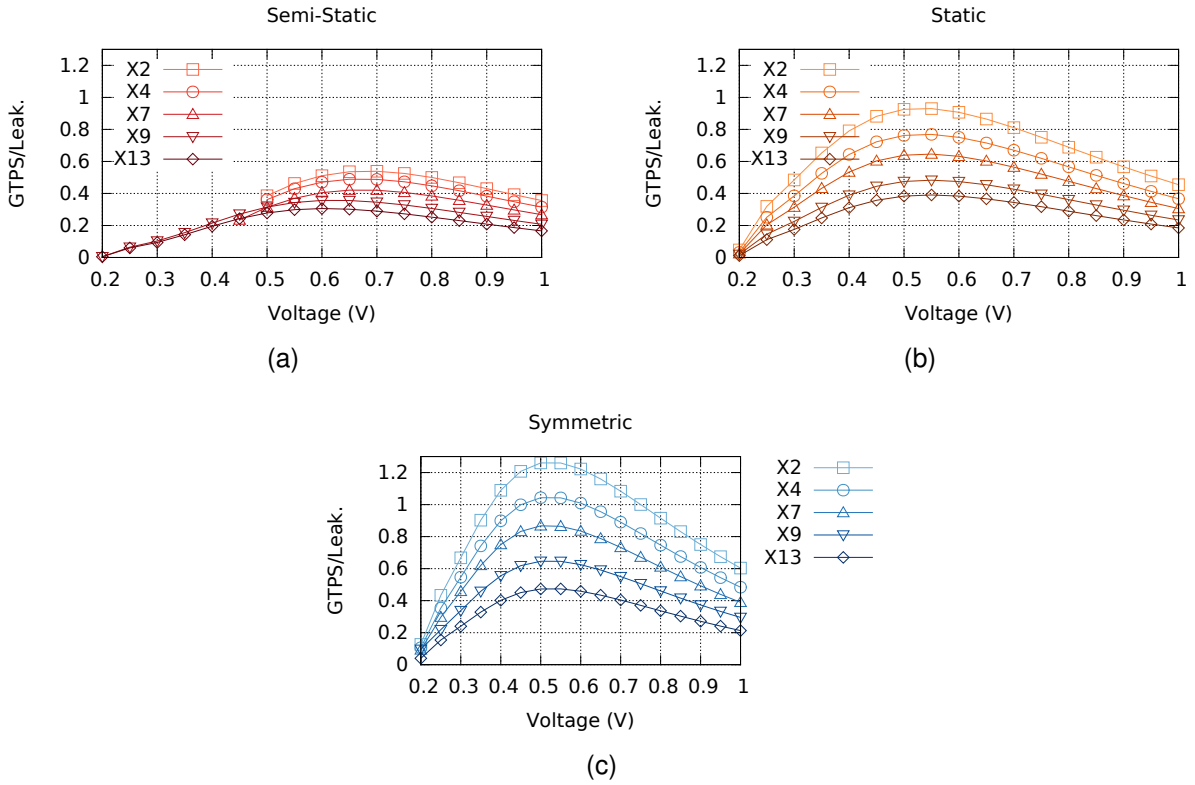


Figure 3.15 – Speed-leakage efficiency for varying voltage supplies for each drive of the three C-elements: (a) semi-static, (b) static and (c) symmetric. Adapted from [MC13].

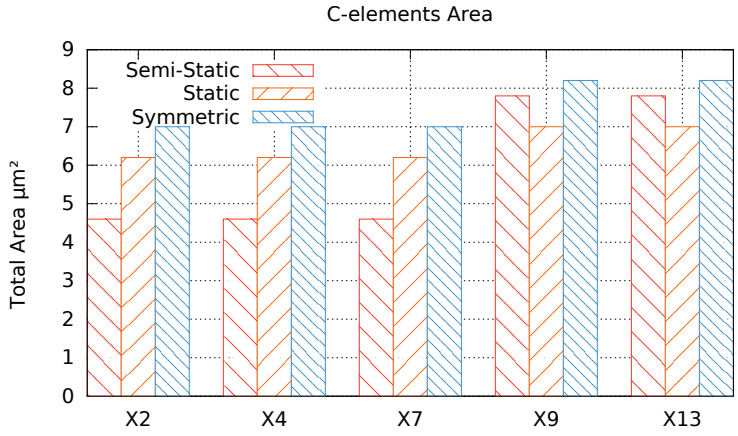


Figure 3.16 – C-elements area for each drive. Adapted from [MC13].

low area, as Figure 3.16 makes clear. However, albeit the symmetric C-element requires more silicon area in all cases compared to the static C-element, it still presents the best speed-area efficiency. This is due to the higher GTPS provided by the symmetric design, as Figure 3.13 shows, meaning that although the static topology provides area reduction, these are not as substantial as the GTPS improvements provided by the symmetric C-element.

The main suggestion here is that the symmetric topology is the most adequate one for low voltage applications. Although it has slightly worse speed-area efficiency (yet, note

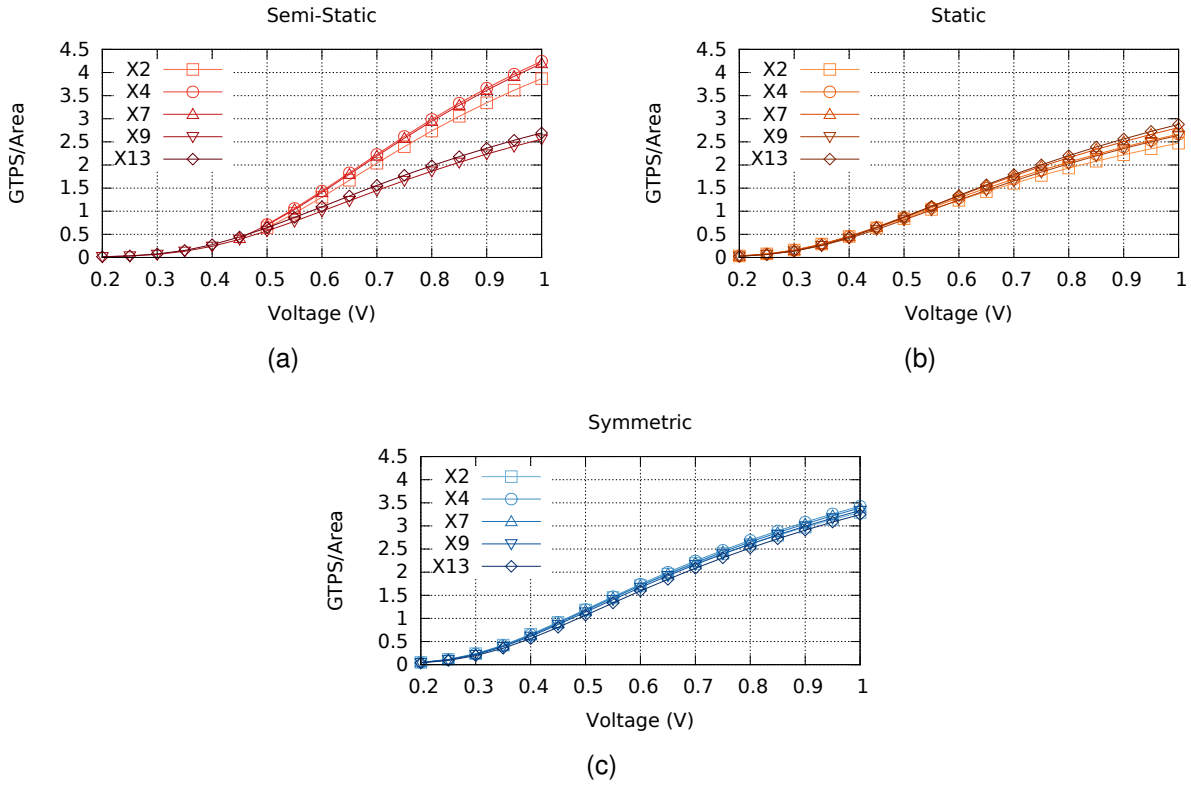


Figure 3.17 – Speed-area efficiency for varying voltage supplies for each drive of the three C-elements: (a) semi-static, (b) static and (c) symmetric. Adapted from [MC13].

that for high drives, it is actually better), it provides the highest speed-energy and speed-leakage efficiency in all cases. Thus, its use for low power applications using voltage scaling techniques is strongly recommended. Besides, results suggest that the best speed-energy and speed-leakage efficiency are obtained when operating at voltages near the threshold. These results help designers using C-elements, as they enable another operation mode: best cost benefit considering speed, energy and leakage. This can be used for coping with many contemporary problems such as battery-based systems, low power budgets and green computing challenges. Finally, as Figure 3.13 and Figure 3.15 show, further optimizations can be achieved by employing low drive C-elements, enabling a better design space exploration for low power application. This occurs because these C-elements present better efficiency than higher drives ones.

3.3 NCL Gates Design

As explored in Section 3.1.3, NCL gates are typically designed using semi-static or static topologies. These topologies are a generalization of the same semi-static and static topologies employed for C-elements design explored in Section 3.2. Hence, the same results obtained for the C-element topologies analyses apply to NCL gates. In this way, we

limit our experiments with NCL gates to the static topology only, as it is the one that presents best overall trade offs, including area, power, energy and delay. In our first experiments with NCL gates design we designed a basic set of gates, as explored in [MOPC13a], employing the classic static topology. During the design of these gates we noticed that they were susceptible to charge sharing effects [RCN03, CH87] and proposed a methodology to overcome this issue. We then proposed a new topology for NCL components, that is also useful for the design of C-elements. This Section explores these innovations.

3.3.1 Charge Sharing Aware Design

When two electric equipotential regions at different initial voltages are connected together [RCN03, CH87], they share charges to reach a new global potential value, causing voltage variations on both regions due to charge redistribution. Such voltage changes in, *e.g.*, nodes of a transistor circuit are effects of charge sharing. Also, it is possible to discern two kinds of charge sharing: pure charge sharing and charge sharing with a driven path [CH87]. In NCL, the latter is not problematic, as it will only interfere in the output of a gate during output switching, which is the expected behavior. Pure charge sharing, on the other hand, can be hazardous. This phenomenon occurs when, *e. g.*, two non-driven RC subnets are connected through a switch that is closed. Classically, in dynamic synchronous circuits, like domino logic, charge sharing arises when either charge flows from a gate output to internal nodes capacitances previously discharged, or when charge flows from initially charged internal nodes parasitic capacitances to the output. These phenomena cause glitches on the outputs that, depending on the scenario, may trigger adjacent gates erroneously, producing SETs that can be latched and cause SEUs.

In asynchronous circuits, glitches caused by charge sharing effects can have irreversible consequences [BOF10]. These circuits rely on complex handshake communication protocols that can stall in the presence of a soft error, causing the whole circuit to halt. Also, in asynchronous circuits, SETs are hardly masked [KH04]. This is due to the fact that there is no temporal assumption and, consequently, no temporal masking. Additionally, logic and electrical masking are very restricted. This is because the majority of asynchronous templates make extensive use of sequential components, which limit the depth of logic paths and of signal regeneration effects and increase the possibility of SETs to be latched and generate SEUs. Hence, an important characteristic in asynchronous components design is robustness against phenomena that can cause SETs and SEUs.

With this in mind, we conducted an extensive analysis on NCL gates design to understand their analog behavior and susceptibility to charge sharing effects [MOMC13]. Recalling Section 3.1.3 static NCL gates employ, as usual in static logic, series and parallel transistor connections. Depending on the threshold and on the number of inputs, more tran-

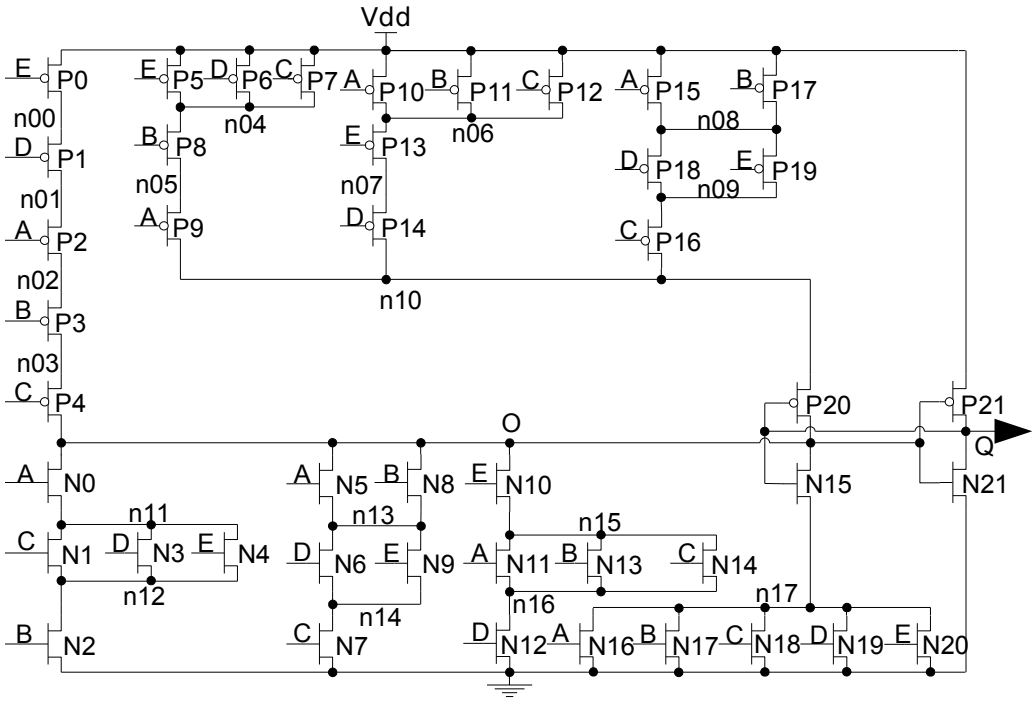


Figure 3.18 – A0 CMOS schematic for the 3-of-5 example NCL gate. Adapted from [MOMC13].

sistors can be required. Complexity may lead to routing congestion due to the existence of many internal nodes when implemented on silicon, which in turn increases parasitic capacitances of the internal nodes of the gate. Also, transistors of the logic stack are usually big, because they are required to drive the output inverter. In this way, their drain and source areas are made larger, which also increases parasitic capacitances. With larger internal parasitic capacitances, the gate is more susceptible to problems caused by charge sharing, as bigger glitches can be generated, increasing the possibility of these glitches to be latched and to SEUs.

As an example, and without loss of generality, we will use a static 3-of-5 NCL threshold gate as a case study for our analysis. Figure 3.18 shows a first schematic of this gate, called Arrangement 0 (*A0*). The choice of this gate as case study is justified for its extensive use in fundamental blocks usually found in QDI circuits, like adders and for being a critical case for the analysis herein. All transistors in the experiments are assumed to be general purpose standard threshold devices.

As Figure 3.18 shows, the PMOS transistors in the logic stack of the 3-of-5 gate (*P0-P4*) can only be arranged in series, to guarantee that all inputs at 0 drive the output to 0. The capacitances of nodes *n00-n03*, that connect these transistors, are usually not increased by parasitics inherent to metal wiring after layout design. This is because such transistors are likely to be connected by abutting the diffusion of their respective drains and sources. In this way, it is not expected that charge sharing effects caused by these capacitances generate relevant hazardous glitches.

Table 3.9 – Input sequence for observing charge sharing effects in a 3-of-5 gate. Adapted from [MOMC13].

Inputs	**Time instant**								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
A	0	0	0	0	0	0	0	0	1
B	0	1	1	1	0	0	0	0	0
C	0	0	1	1	1	0	0	0	0
D	0	0	0	1	1	1	0	0	0
E	0	0	0	0	0	0	0	1	1

The NMOS transistors of the logic stack, on the other hand, can be arranged in many different forms. Each form can contribute differently to increase glitches caused by charge sharing in the pull-down region. Arrangement *A0*, specifically, is a worst case. For instance, assume the scenario presented in Table 3.9. Initially, all inputs are at 0. In this way, nodes *n00-n03* and *O* are charged and the output is set to 0 through the output inverter. This is the typical scenario of the beginning of data transmission in QDI circuits. Next, say that inputs *B*, *C* and *D* are set to 1, one at a time (time instants 1-3). In this way, nodes *O* and *n11-n16* are discharged and the output is switched to 1. This represents data being propagated through the circuit.

Now say that all inputs are set back to 0, one at a time, starting with *B*, then *C* and then *D* (time instants 4-6), to avoid capacitance coupling of the initially charged nodes *n00-n03* and the discharged nodes *n11-n16*. In this stage, node *O* is charged and all nodes *n11-n16* are kept discharged. This is an initial setup for the analysis of the charge sharing problem. Finally, say that inputs *E* and *A* switch to 1, in this order (time instants 7-8). The effect is that when input *E* switches, capacitance of the following pairs of nodes are coupled: *n11* and *n12*, *n13* and *n14* and *O* and *n15*. At this point, a glitch occurs in node *O* as part of its charge is absorbed by node *n15*. However, a larger glitch occurs when input *A* switches, because, in this case, the charge stored in *O* is partially absorbed by capacitances of nodes *n11-n14* and *n16*, which represent a much bigger value. In such a scenario, this glitch can be more easily latched, generating an SEU. This scenario is very common in NCL applications. From our own experience, in adders this exact situation or an equivalent one always happens during the computation of each single bit. Additionally, even worse scenarios may occur, if more inputs are assumed to switch, which is also realistic in applications implemented in NCL.

Given the input sequence of Table 3.9, we propose another arrangement for the 3-of-5 NCL threshold gate, namely Arrangement 1 (*A1*). Figure 3.19 shows its CMOS schematic. The difference from *A0* is that the NMOS transistors of the logic stack were redistributed. Note the transistor parallel is now closer to node *O*. In this way, at each input switch, the charge stored in node *O* is slightly absorbed. This helps avoiding that bigger capacitances accumulate before coupling with node *O*, like in *A0*. For instance, assume that

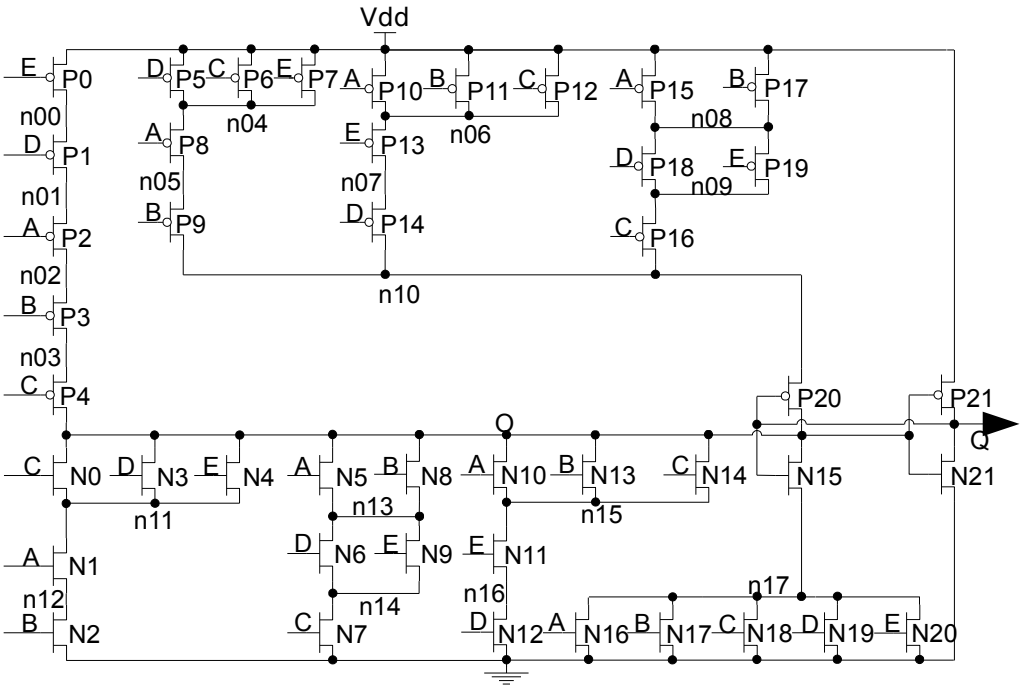


Figure 3.19 – A1 CMOS schematic for the 3-of-5 example NCL gate. Adapted from [MOMC13].

A1 is submitted to the same initial setup as *A0* (instants of time 0-6). At this point, if any input switches to 1, less parasitic capacitances will be accumulating, since critical nodes of parallel transistors will have their capacitances coupled to node *O*. For the example scenario, in the next instant of time (7), when input *E* switches to logical 1, capacitance of node *n11* is coupled to *O*, absorbing part of its charge. In last instant of time, when input *A* switches, less capacitance accumulates and smaller glitches are expected.

This implementation alleviates glitches generated by charge sharing effects. A bigger capacitance in node *O* would provide further improvements. However, a bigger capacitance in this node can drastically impact the gate performance. In other words, a bigger load would take longer to be charged/ discharged or bigger transistors, which would display bigger power figures. In this context, we propose Arrangement 2 (*A2*), showed in Figure 3.20. In *A2* the capacitance of node *O* is only increased when the cell is not switching. This is done by placing parallel PMOS transistors of the feedback group closer to *P20*. Here, when the output is at 0 and is not switching, capacitance of node *O* is coupled with capacitance of node n10, which has an increased parasitic capacitance in *A2*, due to the larger number of transistors connected to it. Also, depending on the combination of inputs set to 0, bigger loads are coupled, as nodes *n04-n09* can be connected to node *n10*. When the cell is to switch its output to 1, the connection of nodes *n04-n10* to power is cut off, and their charge is drained through the direct connection to ground provided by the NMOS transistors of the logic stack.

For instance, say that *A2* is submitted to the initial setup of Table 3.9. When input *E* switches to 1, at the time instant 7, there is a bigger capacitance coupled to node *O*,

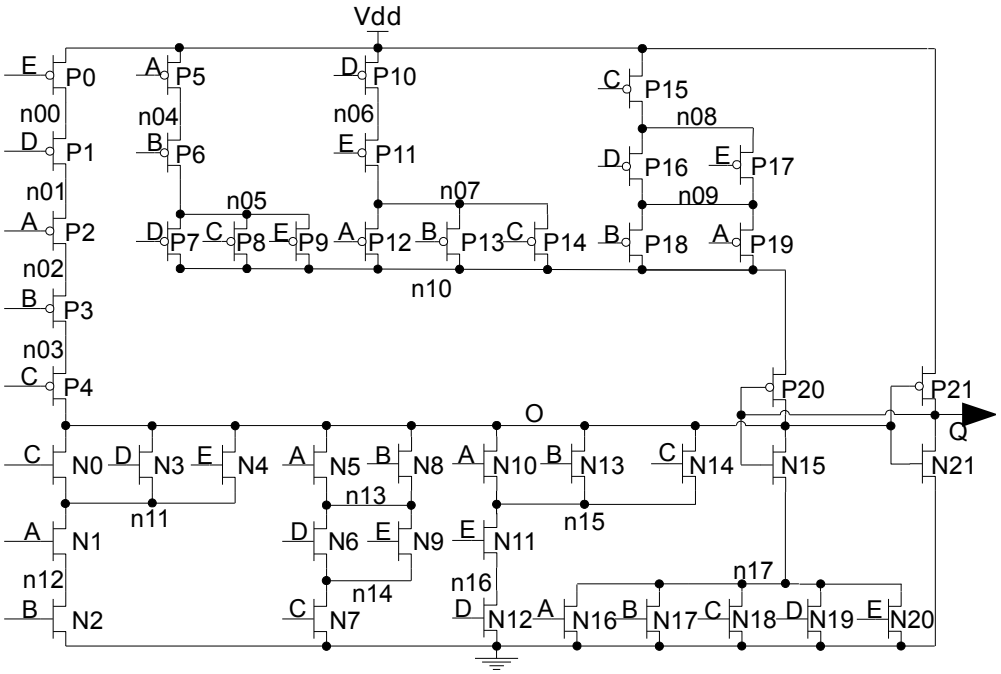


Figure 3.20 – A2 CMOS schematic for the 3-of-5 example NCL gate. Adapted from [MOMC13].

and the generated glitch is alleviated. When input *A* switches, at the next time instant, the same effect is observed. However, as soon as another input switches to 1, the connection of feedback group nodes to power source is cut off, lowering their interference in node *O* switching.

To analyze the analog behavior of arrangements *A0*, *A1* and *A2*, each arrangement was described in Spice using the 65nm STMicroelectronics general purpose standard threshold models. Each of the four arrangements was designed for six different driving strengths, X2, X4, X7, X9, X13 and X18, for a total of 3*6=18 different designs. The transistors of the output inverter had the same size of those of an equivalent drive inverter of the core library of the target technology. Transistors of the feedback group are minimum size, a classical approach for designing static components for asynchronous circuits and the transistors of the logic stack had their size calculated according to the flow presented in Chapter 4.

As explained before, SEUs in NCL gates caused by charge sharing are directly related to internal parasitic capacitances. In this way, an experiment was performed to define hazardous parasitic capacitance values. Note that no layout information is available in this phase. The schematic of all designed gates was simulated assuming no wire parasitics, only drain and source diffusion and gate capacitances, to isolate and evaluate the impact of the parasitics on nodes *n11-n16* that could be extracted after layout design. Simulations were performed inserting varying values of capacitances on these nodes and on node *O*, to define safe parasitic combinations limits. Capacitances were varied from 0 fF to 4 fF in 0.05 fF steps. In this way, a total of 80*80=6400 scenarios of parasitic capacitances

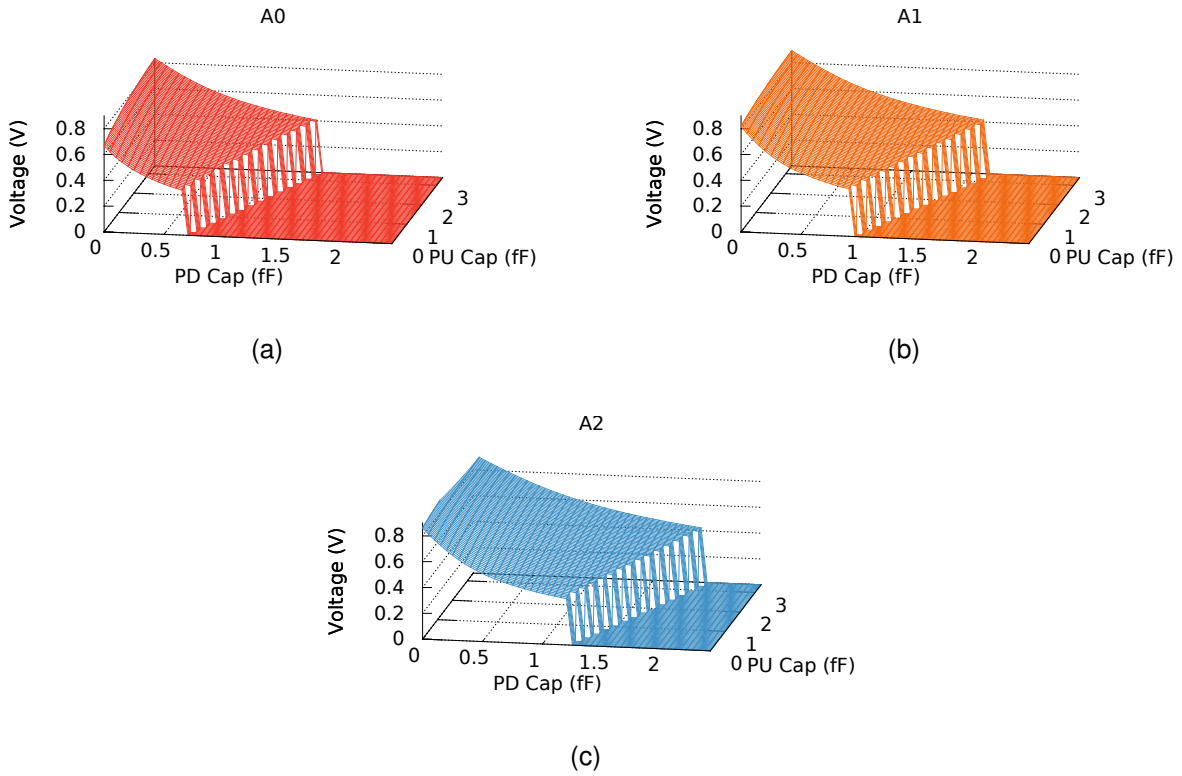


Figure 3.21 – Maximum negative peak values (in Volts) of glitches caused in node *O* due to charge sharing effects for (a) A0 , (b) A1 and (c) A2. Adapted from [MOMC13].

combinations were simulated. All designs had 50 ps input slopes, equivalent to an average size inverter, and output loads equivalent to four inverters of the same driving strength, FO4. The input sequence scenario is that in Table 3.9. During simulations, the peak of the glitch, lowest voltage value, in node *O* was measured after time instant 8 (remember glitches here are from high to low voltage, i.e. negative peaks). Results used typical process models operating at typical conditions (1 V and 25º C). The choice simulator was Cadence Spectre.

Figure 3.21 summarizes the results obtained for the X2 drive version of each arrangement. In the charts, PU Cap is the parasitic capacitance in node *O* and PD Cap is the parasitic capacitance in each node of the pull down network of the logic stack (*n11-n16*). As Figure 3.21(a) shows, for *A0*, considering no parasitic capacitance in node O (PU Cap=0), PD Cap values bigger than 0.6 fF, generate SEUs. The abrupt fall towards 0 V in the chart represents the critical points, where if bigger PD Caps are present, the glitch generated in node *O* is sufficiently big to cause it to be latched in the memory scheme, switching the output value to 1 and the value of node *O* to 0.

In other words, for the bottom of the chart (0 V), the glitch generated by charge sharing effects was sufficiently large to be latched and generate an SEU. PD Cap values that are lower than those in the critical point also generate glitches. However such glitches are not high enough to cause an SEU. For instance, for a PU and PD Cap of 0, the peak of the glitch measured in node O is of 0.67 V, a glitch of 33% of VDD. However, maintaining the

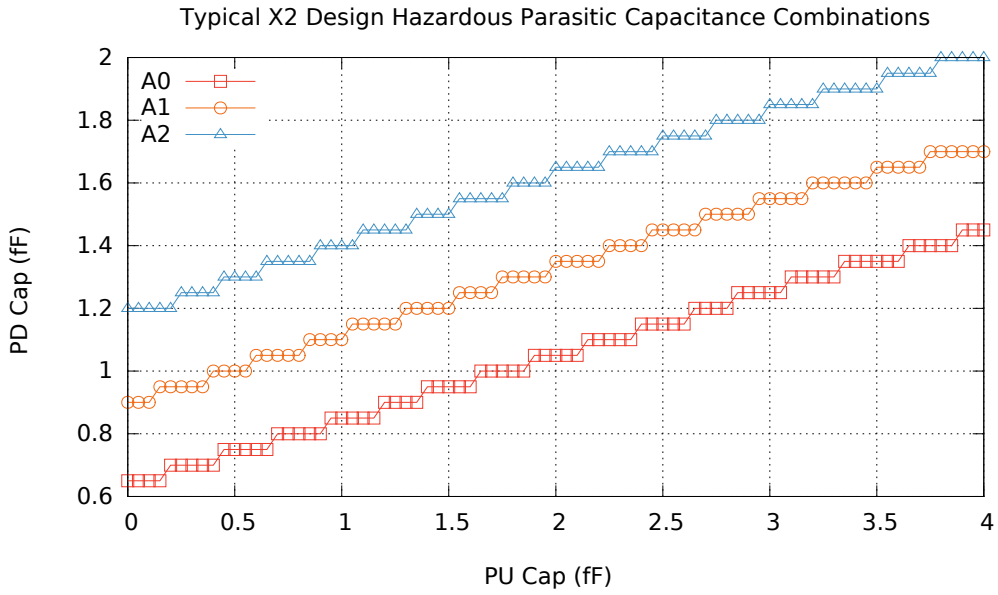


Figure 3.22 – Critical PU and PD Cap for the X2 drive cells for a typical process and typical operating conditions. Adapted from [MOMC13].

PU Cap of 0, for a PD Cap of exactly 0.6 fF, just before the critical point, the glitch peak is 0.36 V, which is much more critical (64% of VDD) but not enough to be latched. Note that PD Caps of 0 are impossible in practice, especially because node *O* is used for interconnecting many transistor branches. The problem is that, even if PU Cap is raised to 3 fF, or even 4 fF, values of PD Cap close to 1 fF can generate SEUs, which is 3 or 4 times lower than PU Cap.

The values measured for *A1*, showed in Figure 3.21(b), move the step that represents the critical point to larger values of PD Cap. In this way, more PD Cap parasitics are required to cause SEUs. In fact, in average, for the same PU Cap values, PD Cap parasitics in *A1* must be 45% bigger than those in *A0*. Also, glitches for similar PD Caps, and lower than those of the critical point, are roughly 22% smaller in *A1* when compared to *A0*. As Figure 3.21(c) shows, for *A2*, these values are improved even more. For the same PU Cap values, PD Cap parasitics in *A2* must be in average 92% bigger than those in *A0* to cause SEUs and glitches for similar PD Caps are roughly 36% smaller. Another analysis of the same simulation results appears in Figure 3.22. In the chart, critical points (the highest value of PD Cap for each PU Cap before causing an SEU) were isolated for each arrangement. In this way, *A2* is almost twice as robust as *A0*.

After analyzing the results obtained for all driving strengths, we noticed that bigger drive cells are more susceptible to charge sharing problems, depending on the arrangement. The difference on the results for different drives is just quantitative, not qualitative. Thus, only the worst case, X18 is analyzed here. Figure 3.23 shows critical PU and PD Cap combinations for this drive for the same scenario above. It is clear that the improvements are much bigger. For instance, in worst case, PD Caps of 0.25 fF are required to generate

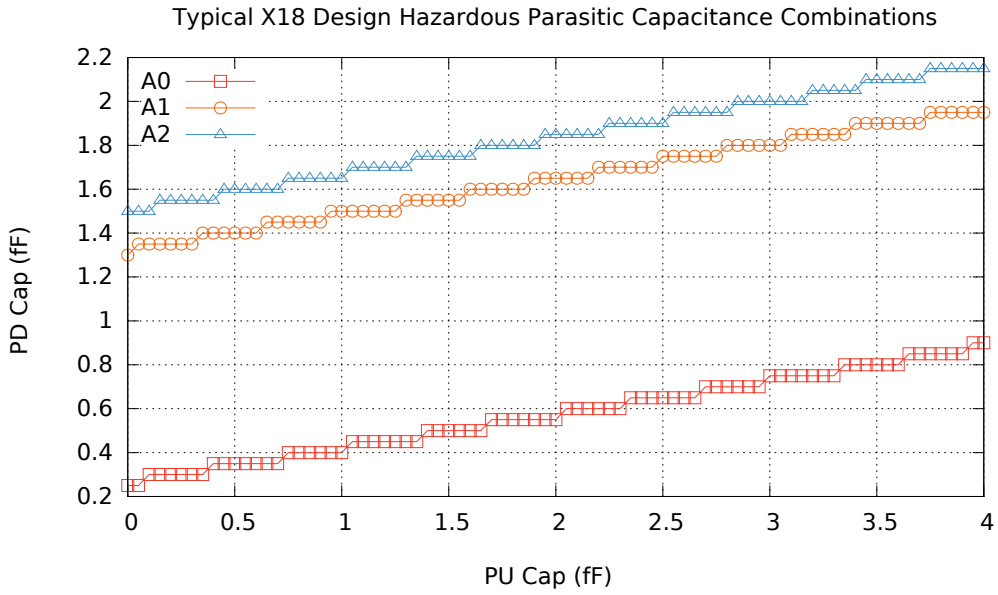


Figure 3.23 – Critical PU and PD Cap for the X18 drive cells for a typical process and typical operating conditions. Adapted from [MOMC13].

SEUs for *A0*, while for *A1* and *A2*, this value is of 1.3 fF, 1.5 fF and 1.45 fF, respectively. This means arrangements 5.2 and 6 times more robust than *A0*, respectively.

The reason why bigger drives are more susceptible to charge sharing effects is the elevated diffusion capacitances of NMOS transistors of the logic stack that contribute to internal parasitics, making lower interconnection parasitics required for causing SEUs. Also these transistors are bigger and discharge node *O* at faster rates. The presented results are all based on typical fabrication processes. However, as variations get critical in current technology nodes, this can be an overoptimistic approach. Accordingly, we simulated all the circuits for the same scenario for a worst case fabrication process variation: slow PMOS transistors and fast NMOS transistors. Thus, bigger glitches are generated in node *O* and less glitches are filtered by the output inverter.

Figure 3.24 presents the results obtained for the most critical case, namely the X18 drive cells. As observable in the Figure, for *A0* no interconnection parasitics are needed to cause SEUs. However, the proposed optimizations push critical PD Cap values to over 0.65 fF. In this way, it is expected that 3-of-5 NCL gates designed according to *A0* for an X18 driving strength generate SEUs when under the scenario shown in Table 3.9. In fact, we detected that this problem starts at driving strengths larger than X7. The degradation in robustness of NCL gates as their driving strength get bigger observed for *A0* is undesirable. In fact, the quality of cell-based logic synthesis relies on the availability of gates with different functionalities and driving capabilities. From a functional point of view, the more gates available to implement logic functions, the more optimizations are possible. However optimizations in power, area and speed require different driving strengths for each gate dur-

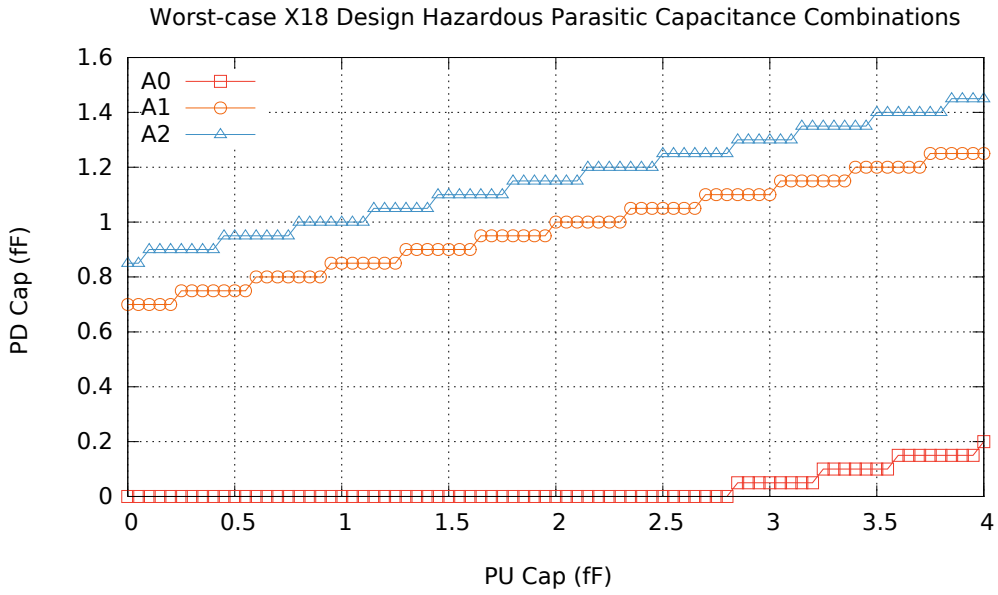


Figure 3.24 – Critical PU and PD Cap for the X18 drive for a slow PMOS and fast NMOS process and typical operating conditions. Adapted from [MOMC13].

ing physical synthesis. Thus, it is essential that NCL gates maintain their robustness for increased driving strengths, in order to provide quality standard-cell-based design.

Another perspective on the obtained results shows the observed degradation. Figure 3.25(a) shows the critical PU and PD Cap combinations for each driving strength. The chart is a steep ramp where results get worse as the driving strength increases. There is a small deviation, in drive X13, which presents slightly better robustness than X9. However this effect is due to transistor sizing effects. Transistors of the logic stack in these drives were set to similar dimensions, while transistors of their output inverter are different; in X13 they are bigger. A bigger output inverter leads to a bigger capacitance in *O* for X13 while maintaining the same logic stack nodes capacitance, generating a slightly more robust cell. More interestingly, though, the steep ramp problem is not observed in the proposed optimizations. As charts of Figure 3.25(b) and 3.25(c) show, *A1* and *A2*, present very similar robustness no matter the driving strength. For these arrangements, the ramp observed in *A0* is turned into flatter surfaces, which indicates that these arrangements are suited for increasing the robustness of NCL-based logic, while maintaining the advantages of using a cell-based approach for circuit design.

3.3.2 New Topology

The exploration of the existing topologies for designing NCL gates allowed a better understanding of the trade offs of different transistor arrangements. This in turn enabled us to propose a new topology targeting low voltage operation, as explored in [MAGC14]. Fig-

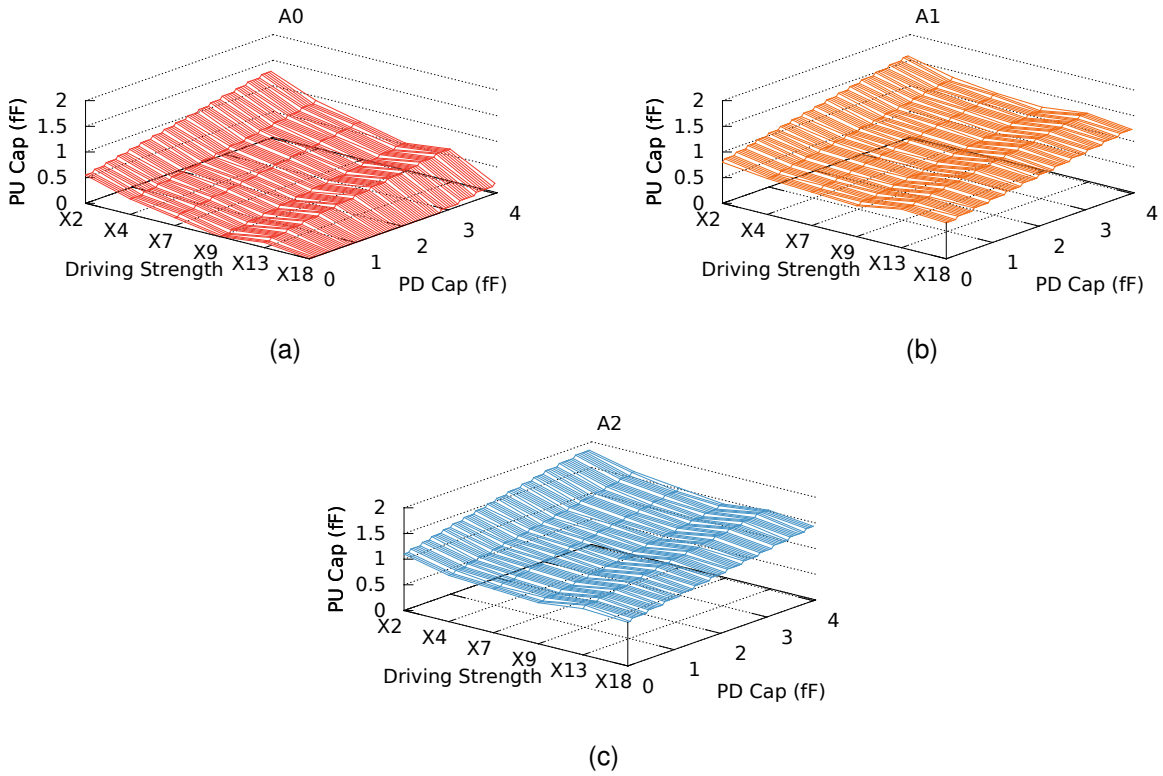


Figure 3.25 – Robustness of arrangements (a) A0, (b) A1 and (c) A2. Adapted from [MOMC13].

ure 3.26 presents the basic schematic for this topology. As the Figure shows, the output *Q* switches to 1 or 0, according to the set and reset transistors *P0* and *N0*, respectively. Dimensioning of these follows a process similar to that of the output inverter for the previously explored topologies. The same *RESET* and *SET* networks control these transistors. However, such transistors are also controled by the complements of these networks, *HOLD1* and *HOLD0*. Accordingly, the *P0* and *N0* driving transistors are both turned off by a hold state input combination. This state uses transistors *P1-P3* and *N1-N3* to maintain the output stable. All these are minimum-sized and the mechanism employs a loop of two inverters, where the driving inverter (*P3-N3*) is controlled by *HOLD1/RESET* or by *HOLD0/SET* (through *P2* and *N2*). Note that *HOLD0* and *HOLD1* transistors are also minimum size and transistors used for *SET* and *RESET* are dimensioned for driving *P0* and *N0* input capacitance loads. Also, since C-elements are special cases of NCL gates, the new scheme is also a new topology for C-elements.

The new scheme guarantees that *P0* and *N0* will never be ON simultaneously, since the *RESET* network is ON only when all inputs are 0, while *SET* requires that at least one input be 1. Practical *SET* functions for this topology require at least 2 inputs to be 1. Albeit it is true that some *SET* functions can require only 1 input to be 1 in NCL design, these functions do not require a memory mechanism and avoid the need for the proposed topology or the classic static and semi-static topologies. This renders the proposed topology

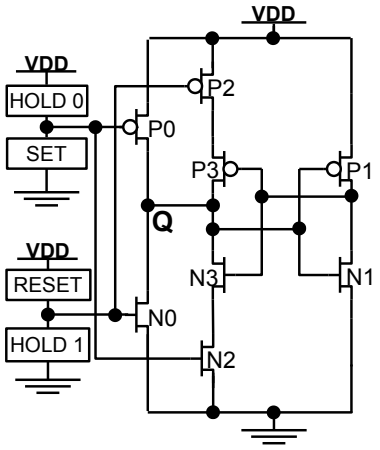


Figure 3.26 – Proposed topology for designing NCL gates. Adapted from [MAGC14].

even more interesting, because momentary shorts present in the output inverter of the static topology are avoided. In fact, before making either *P0* or *N0* ON, both transistors are always turned off, because as soon as *SET* and *RESET* are OFF, their complements *HOLD0* or *HOLD1* will be ON, and a minimum of two inputs are required to switch from an ON *SET* network to an ON *RESET* network and vice-versa. Also, because input capacitances of *P0* and *N0* are decoupled, contrary to what happens in the classical static and semi-static topologies, *SET* and *RESET* networks need to drive smaller capacitances to switch the output. In this way, better power and delay trade offs are expected as characteristics of the new topology.

In order to compare the proposed topology with the static one, 3 different NCL functionalities were implemented to the layout level for 3 different driving strengths (X2, X7 and X13). This produced a total of 2*3*3=18 case study gates. The target technology was the STMicroelectronics 65nm bulk CMOS and gate design followed the procedure presented in Chapter 4. The selected functionalities were: 2-of-2 (M=2, N=2 and weights=1 1); ANDOR2-of-4 (special AND-OR function); and 3W22-of-4 (M=3, N=4 and weights=2 2 1 1). This choice allows evaluating trade offs of topologies for low (2-of-2), medium (ANDOR2-of-4) and high (3W22-of-4) complexity gates. Also, in our experience these functionalities are commonly used in NCL designs. After layout extraction, analog simulation allowed evaluating the case study gates. All results presented herein are based on worst case RC parasitics extraction. Tables 3.10, 3.11 and 3.12 present a general comparison of five relevant parameters for all designs: area, input capacitance, parasitic capacitance, speed-to-energy efficiency and speed-to-leakage efficiency. Regarding area, it is clear that for small driving strengths, the proposed topology displays substantial area overhead. In fact, for a 2-of-2 gate of drive X2 and X7, it requires 1.4 times the area required by a static topology. This area overhead for small driving strengths is a consequence of the fixed overhead of four transistors. In fact, as the driving strength increases, the overhead decreases. Therefore, the bigger the driving strength is, the lower is the area overhead. Besides, as gate complex-

Table 3.10 – Area, input and parasitic capacitances, speed-energy and speed-leakage trade-offs for the 2-of-2 case study gates. Adapted from [MAGC14].

Drive	Topology	Area (um²)	Inp. Cap. (fF)	Para. Cap. (fF)	GTPS/EPT	GTPS/LP
X2	*Static*	5.20	0.60	3.084	5.87	0.13
	Proposed	7.28	0.61	3.760	5.74	0.12
X7	*Static*	5.20	0.66	3.400	2.86	0.06
	Proposed	7.28	0.62	3.662	3.82	0.08
X13	*Static*	6.76	0.69	3.821	1.52	0.03
	Proposed	8.32	0.78	4.980	2.19	0.05

Table 3.11 – Area, input and parasitic capacitances, speed-energy and speed-leakage trade-offs for the 3W22-of-4 case study gates. Adapted from [MAGC14].

Drive	Topology	Area (um²)	Inp. Cap. (fF)	Para. Cap. (fF)	GTPS/EPT	GTPS/LP
X2	*Static*	9.36	1.34	7.986	3.33	0.47
	Proposed	11.44	1.20	7.190	3.36	0.31
X7	*Static*	9.36	1.53	8.440	1.80	0.28
	Proposed	11.44	1.18	7.723	2.30	0.25
X13	*Static*	10.92	1.20	8.700	0.99	0.16
	Proposed	13.00	0.93	8.657	1.36	0.17

Table 3.12 – Area, input and parasitic capacitances, speed-energy and speed-leakage trade-offs for the AO2-of-4 case study gates. Adapted from [MAGC14].

Drive	Topology	Area (um²)	Inp. Cap. (fF)	Para. Cap. (fF)	GTPS/EPT	GTPS/LP
X2	*Static*	8.32	0.94	6.344	3.21	0.49
	Proposed	8.84	0.77	5.754	3.43	0.35
X7	*Static*	8.84	0.91	6.240	1.68	0.28
	Proposed	9.36	0.77	5.938	2.56	0.30
X13	*Static*	9.36	0.97	6.938	0.94	0.16
	Proposed	10.40	0.76	6.730	1.61	0.20

ity grows, the cost is also amortized. This can be noticed by analyzing the area results for 3W22-of-4 and ANDOR2-of-4 gates. For the latter, the area required by the static topology is never less than 90% the area of the new topology.

Regarding the input capacitance data, the tables present results obtained after layout extraction. Values correspond to the worst case among capacitances for all inputs in each cell. The static topology presents an input capacitance which is always bigger than the new topology for the more complex gates (3W22-of-4 and ANDOR2-of-4). In the worst case, it presents an overhead of almost 30%. For the 2-of-2 case study, the input capacitance is quite similar to that obtained for the proposed topology. The justification for this is because the new topology typically requires smaller transistors in the *SET* and *RESET* networks, leading to reduced gate capacitance. This indicates that the new topology is suited for cell-based design, as input capacitance is crucial in technology mapping and optimization steps. In fact, smaller capacitances enable better speed, energy and power trade offs at

the system level. As for parasitics, compared to the static topology the proposed topology presents lower parasitics in all cases, except for the lowest complexity gate (2-of-2). This indicates the suitability of the new topology to produce complex NCL gates.

For speed, energy and leakage power figures simulations employed typical fabrication process corners and operating conditions (1 V and 25 C) and all gates had a fixed, FO4, output load. For each case study gate, all transition arcs for each input/output pair were simulated and, for each arc, propagation delay was measured as the time a transition in an input takes to cause an output switching. The energy consumed for switching the output was also measured for each arc. The energy was measured as the integral of the current in the power supply during the time the cell is switching, multiplied by the operating voltage. Also, we simulated all static states of each case study gate and we measured leakage power as the average current in the power source during each state, multiplied by the operating voltage. From these results we obtained the number of times the cell is capable of switching its output per second, measured in GTPS. The GTPS value considers the average between all obtained propagation delays, for each case study. EPT was measured as the average energy consumed for switching the output of each design.

Since each topology can present varying propagation delay, energy and leakage power figures, even for a same driving strength, a fair comparison is not possible by analyzing just these figures. Accordingly, the expression of results employs cost-benefit functions. The sixth column of each table presents the GTPS/EPT results. The ratio between the measured GTPS and EPT values defines the speed-energy efficiency function. This enables evaluating the speed of the gates without overlooking the associated energy consumption. The proposed topology presents bigger GTPS/EPT figures in all cases but for the 2-of-2 gate with a driving strength of X2, where its results are very close to the ones obtained for the static topology. This means that in general the new topology is capable of making more transitions than the static and semi-static topologies for a given amount of consumed energy.

Note that the bigger the driving strength is the bigger is the gain in delay-energy efficiency of the proposed topology. In the best case, an improvement of over 150% is observed when compared to the static topology. *E.g.*, for the ANDOR2-of-4 gate, the static topology has a value of GTPS/EPT that is just 39.9% of the new topology value. This is justified by the fact that the new topology has typically lower capacitances to drive when switching the output. Note that in this topology the gate capacitances of the PMOS and NMOS transistors of the output inverter (see Figure 3.26, *P0* and *N0*) are decoupled. In this way, the proposed topology is suited for energy-efficient applications. Also, the proposed topology is well suited for cell-based design, as its speed-energy efficiency does not decay as in the static topology, as driving strength increases. This allows design optimizations to take place without compromising low power operation.

The Tables also show speed-leakage efficiency results. GTPS/Average Leakage was measured as the obtained GTPS for each gate, divided by the average leakage power.

This creates a cost function to evaluate speed-leakage trade offs and enables analyzing the speed of the gates without overlooking leakage power. In this case, improvements were not as substantial as in delay-energy trade offs. In fact, for X2 gates, our topology presented worse GTPS/Average Leakage trade offs in all cases. However, as the driving strength increases, our topology proves to be more efficient in terms of the speed-leakage power trade off. In the best case, improvements are above 70%. This corroborates the previous results, indicating the suitability of the topology for cell-based design.

Another desirable feature for contemporary integrated circuits is increased tolerance to PVT variations. In fact, process variations are a critical problem in current technologies. A second set of experiments enabled to measure the variability in the observed performance figures for the case study gates while varying operating voltage, temperature and process fabrication parameters. Note that leakage variations were not significant and there were little discrepancies among the analyzed gates. Hence, they were omitted in this analysis.

Firstly, we submitted the gates to variations in operating voltage and temperature. We simulated these for a range from 90% to 110% of the nominal values (1V and 25C) in steps of 1%, combining all possible values. This means 21 distinct voltages and temperatures that combined lead to 21*21=441 simulation scenarios for each design. Figure 3.27 shows the variations observed in measured energy per transition (EPT). In the Figure, the case study gates are distributed along the horizontal axis. A PR prefix indicates the gate employs the proposed topology, while ST and SS prefixes indicate the cell employs the static and semi-static topologies, respectively. Also, 22 stands for a 2-of-2 gate, 3W22 is a 3W22-of-4 gate and AO24 is an ANDOR2-of-4 gate. Gate names were abbreviated to allow a more compact representation in the charts. In these, variations can be either positive or negative and they are measured from the base value obtained for nominal voltage and temperature, represented by the 0 value in the vertical axis in the charts. Note that in most cases the proposed topology presents smaller amplitude in energy per transition variations, when compared to the static topology. Also, as driving strength increases, susceptibility to voltage and temperature variations is worsened in all topologies. However, for the proposed topology, the increase is not as substantial as in for the static and semi-static topologies.

Concerning variations in propagation delay, Figure 3.28 shows that the topologies present similar susceptibility to voltage and temperature variations. However, for the biggest simulated driving strength, the proposed topology always presents smaller variations. In this way, charts of Figure 3.27 and Figure 3.28 confirm the suitability of the proposed topology for cell-based design.

A second set of simulations allowed evaluating the effect of process variations in gate performance figures. To do this, we proceeded to a mismatch Monte Carlo analysis with 5000 samples and measured energy per transition and average propagation delay for each simulated scenario. The charts in Figure 3.29 and Figure 3.30 summarize the results. As

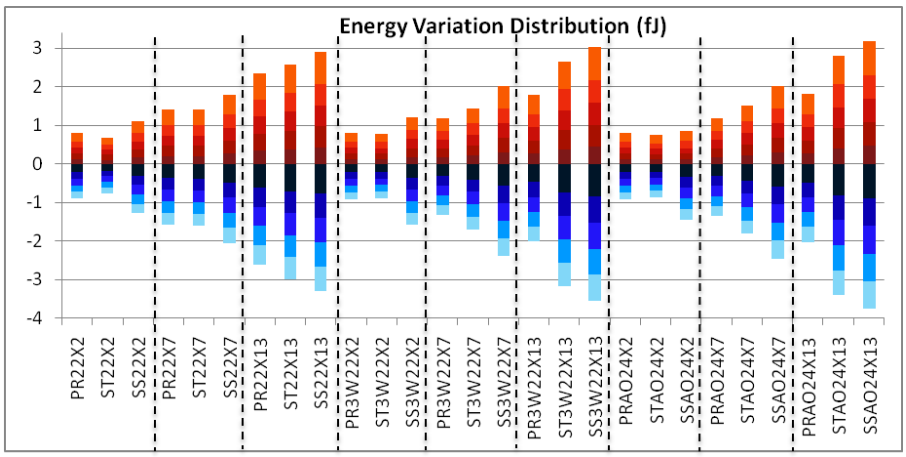


Figure 3.27 – Energy variation distribution varying operating voltage and temperature. Data were divided into 10 sets (5 positive and 5 negative). Each color represents a quintile of positive sets (in red) and negative sets (in blue), where darker colors represent lower quintiles and brighter colors represent upper quintiles. Adapted from [MAGC14].

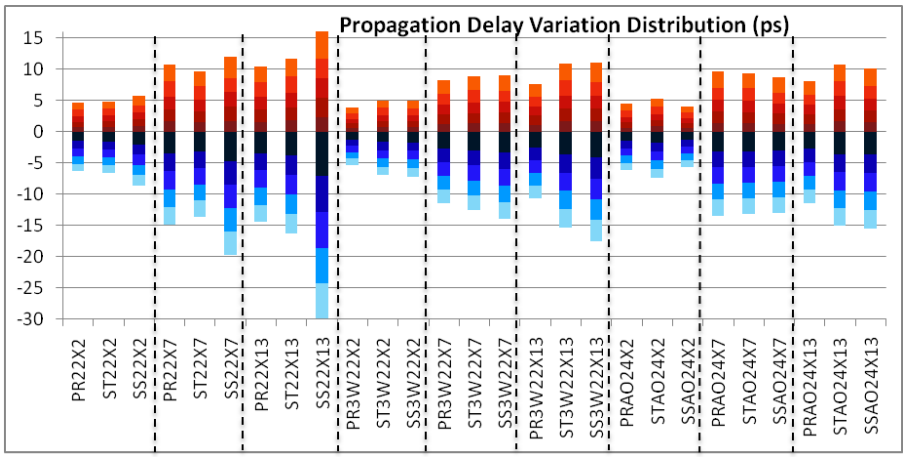


Figure 3.28 – Propagation delay variation distribution for varying operating voltage and temperature. Data were divided into 10 sets (5 positive and 5 negative). Each color represents a quintile of positive sets (in red) and negative sets (in blue), where darker colors represent lower quintiles and brighter colors represent upper quintiles. Adapted from [MAGC14].

the charts show the proposed topology presents smaller variations in energy per transition for X7 and X13 driving strengths. For X2, the results observed are comparable to those of the static topology. As for the observed propagation delay variations, the proposed topology presents values comparable to the ones observed for the static topology in most cases. Hence, in general, the proposed topology is the most robust against PVT variations.

Another very important aspect for contemporary technologies is the ability to operate at voltage levels lower than the nominal. In fact, according to Hanson et al. [HZB+06], voltage scaling is the most effective solution to cope with increasing power constraints. Accordingly, we performed a set of experiments to evaluate the impact of voltage scaling in the case study gates. The first experiment detected the minimum voltages that can be applied to each gate without interfering in their correct behavior. The experiment investigated sce-

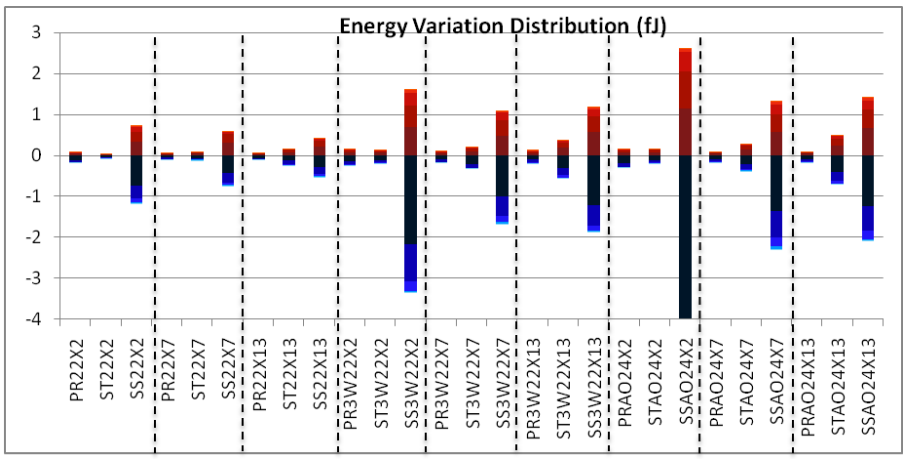


Figure 3.29 – Energy variation distribution observed from Monte Carlo analysis. Data were divided into 10 sets (5 positive and 5 negative). Each color represents a quintile of positive sets (in red) and negative sets (in blue), where darker colors represent lower quintiles and brighter colors represent upper quintiles. Adapted from [MAGC14].

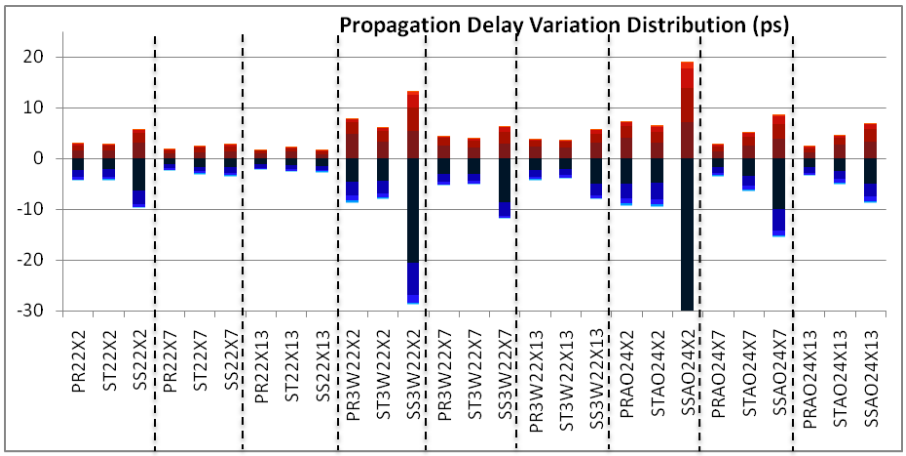


Figure 3.30 – Propagation delay variation distribution observed from Monte Carlo analysis. Data were divided into 10 sets (5 positive and 5 negative). Each color represents a quintile of positive sets (in red) and negative sets (in blue), where darker colors represent lower quintiles and brighter colors represent upper quintiles. Adapted from [MAGC14].

narios for varying temperatures and an FO4 output load. Minimum voltages were estimated similarly to what we did in a previous Section, by simulating all transition arcs and static states of each gate for each temperature/voltage scenario. When at least one arc does not generate the correct output or a static state is not able to maintain correct functionality, the scenario is defined as not functional. Tables 3.13, 3.14 and 3.15 summarize the obtained results. As the Tables show, the proposed and the static topologies tolerate voltages as low as 0.15V and 0.1V, respectively. In fact, the obtained results for these topologies are quite similar and, therefore, they are both suited for semi-custom low voltage design.

A second experiment allowed us to compare speed-energy and speed-leakage trade offs for the static and the proposed topologies under varying supply voltages. Figures 3.31 and 3.32 show the speed-energy efficiency values, in GTPS/EPT. As the charts

Table 3.13 – Observed minimum operating voltage for different versions of the 2-of-2 case study. Darker case values are worse than light case values. Adapted from [MAGC14].

Topology	Drive	125 C	100 C	75 C	50 C	25 C	0 C	-25 C	-50 C
Proposed	*X2*	0.20	0.15	0.15	0.15	0.15	0.15	0.15	0.20
	X7	0.20	0.20	0.15	0.15	0.15	0.15	0.20	0.20
	X13	0.25	0.20	0.20	0.20	0.15	0.15	0.20	0.20
Static	*X2*	0.15	0.15	0.15	0.15	0.20	0.20	0.25	0.25
	X7	0.15	0.15	0.15	0.15	0.20	0.20	0.25	0.25
	X13	0.15	0.15	0.15	0.15	0.20	0.20	0.25	0.25

Table 3.14 – Observed minimum operating voltage for different versions of the 3W22-of-4 case study. Darker case values are worse than light case values. Adapted from [MAGC14].

Topology	Drive	125 C	100 C	75 C	50 C	25 C	0 C	-25 C	-50 C
Proposed	*X2*	0.20	0.15	0.15	0.15	0.15	0.15	0.20	0.20
	X7	0.20	0.20	0.15	0.15	0.15	0.15	0.20	0.20
	X13	0.25	0.20	0.20	0.20	0.15	0.15	0.20	0.25
Static	*X2*	0.15	0.15	0.15	0.15	0.15	0.15	0.20	0.25
	X7	0.15	0.15	0.15	0.15	0.15	0.15	0.20	0.25
	X13	0.15	0.15	0.15	0.15	0.15	0.20	0.20	0.25

show, the proposed topology is the one that presents highest GTPS/EPT values for all case study gates. In fact, the proposed topology achieves optimizations of roughly 50%, in average when compared to the static one. Also, both topologies reach optimum power efficiency when supplied with near-threshold voltages (between 0.5 V and 0.6 V), for all driving strengths. Note that for minimum operating voltages the proposed topology presents GTPS/EPT values that are in average 37% better than the same values for the static topology.

Figures 3.33 and 3.34 show the speed-leakage efficiency values, in GTPS/Avg. Leak., measured for all gates considered here. As the charts show, in this case, the static topology is the best for lower driving strengths. However, as the driving strength is increased, the proposed topology becomes advantageous. This confirms previous results. As the charts show, optimum speed-leakage efficiency is also obtained in the near-threshold voltages. Moreover, for minimum operating voltages, the differences in the observed GPS/Avg. Leak. between the proposed and the static topologies were negligible. In view of the obtained results, we understand that in general the proposed topology presents better speed, energy and leakage power trade offs for different voltage levels. Therefore, we consider that the topology is suited for low voltage operation.

Single event effects (SEEs) can cause the output of an NCL gate to incorrectly flip. Depending of the state of the gate, this can have irreversible consequences. For instance, if the gate is in a state of memorization, i.e. if its output is not being driven by the SET or RESET networks, a glitch on the output may generate a single event upset (SEU). This is discussed for example in [BSK+10] and can compromise the correct functionality of QDI

Table 3.15 – Observed minimum operating voltage for different versions of the AO2-of-4 case study. Darker case values are worse than light case values. Adapted from [MAGC14].

Topology	Drive	125 C	100 C	75 C	50 C	25 C	0 C	-25 C	-50 C
Proposed	*X2*	0.20	0.15	0.15	0.15	0.15	0.15	0.20	0.20
	X7	0.20	0.20	0.15	0.15	0.15	0.15	0.20	0.25
	X13	0.25	0.20	0.20	0.20	0.15	0.15	0.20	0.25
Static	*X2*	0.15	0.15	0.15	0.10	0.15	0.15	0.20	0.25
	X7	0.15	0.15	0.15	0.15	0.15	0.20	0.20	0.25
	X13	0.15	0.15	0.15	0.15	0.15	0.20	0.20	0.25

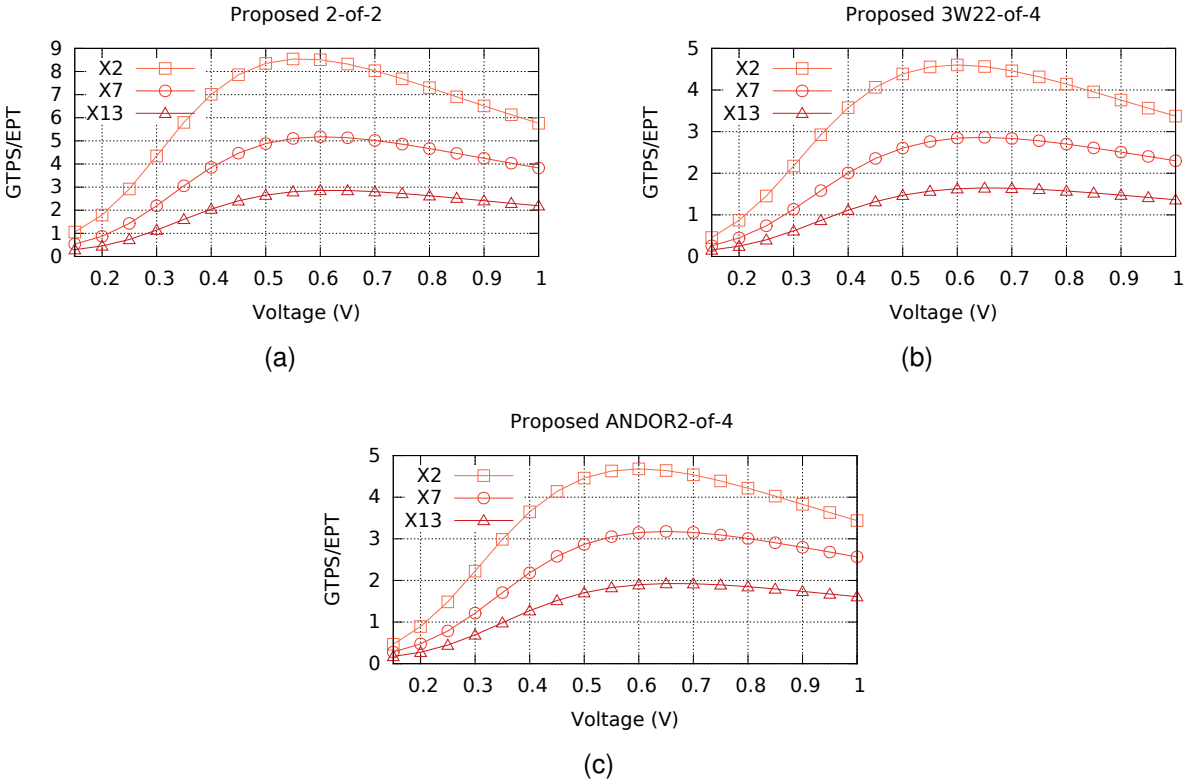


Figure 3.31 – Speed-energy efficiency for 2-of-2, 3W22-of-4 and ANDOR2-of-4 gates using the proposed topology. Adapted from [MAGC14].

circuits. A final experiment allowed us to evaluate the robustness of the implemented gates against SEEs.

To compute robustness data we simulated the behavior of NCL gates under SEEs using the particle strike model proposed in [GK08]. In this model, the current injected in the gate nodes is a consequence of the charge collected, expressed by the following equation:

$$I(t) = \frac{Q}{(\tau_\alpha - \tau_\beta)} \left(e^{\frac{-t}{\tau_\alpha}} - e^{\frac{-t}{\tau_\beta}} \right), \tag{3.3}$$

where Q is the charge collected at the junction, τ_α is the junction collection time constant and τ_β is the ion-track establishment time constant. Note that τ_α and τ_β are technology specific constants. The resulting model entered in MATLAB was converted to a transient

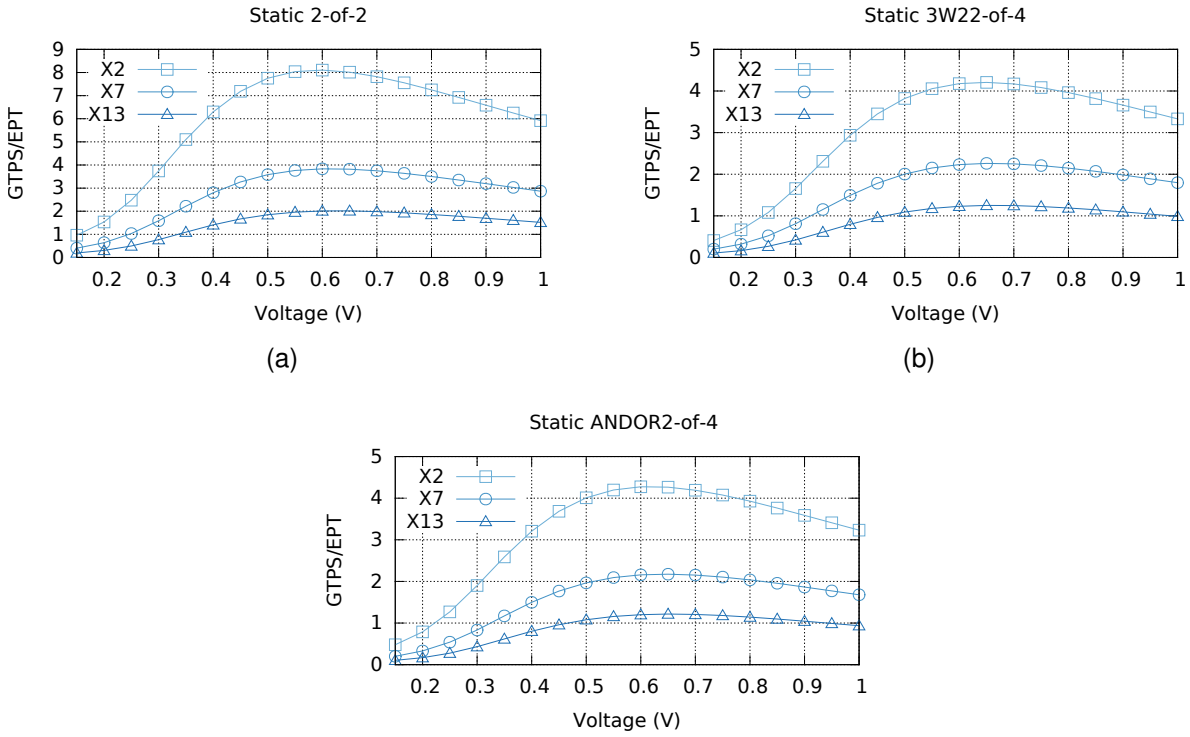


Figure 3.32 – Speed-energy efficiency for 2-of-2, 3W22-of-4 and ANDOR2-of-4 gates using the static topology. Adapted from [MAGC14].

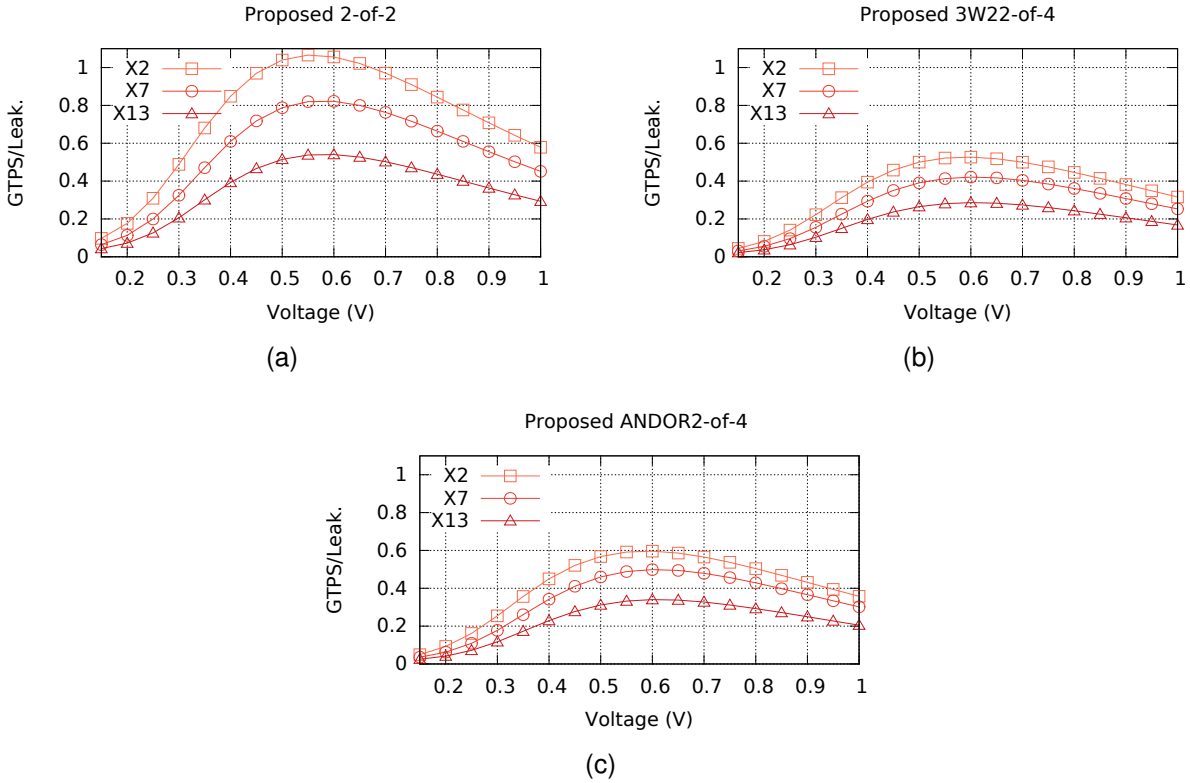


Figure 3.33 – Speed-leakage efficiency for 2-of-2, 3W22-of-4 and ANDOR2-of-4 gates using the proposed topology. Adapted from [MAGC14].

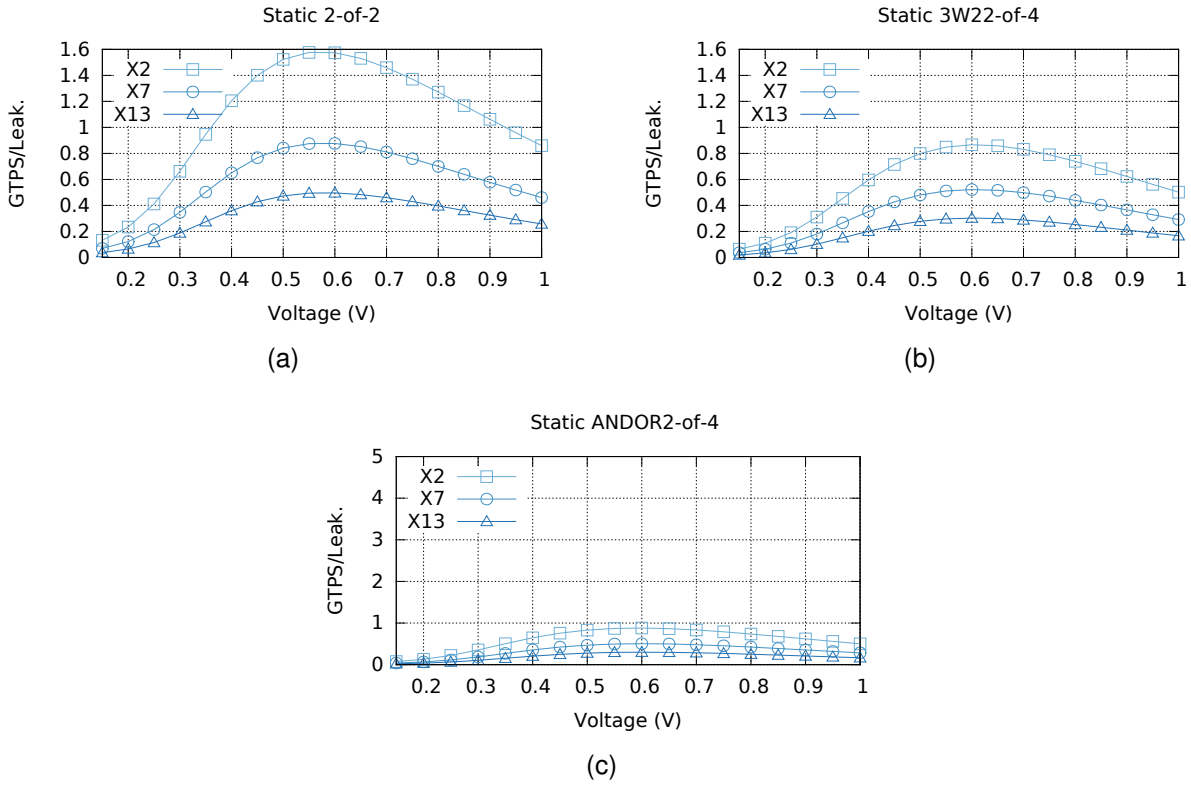


Figure 3.34 – Speed-leakage efficiency for 2-of-2, 3W22-of-4 and ANDOR2-of-4 gates using the static topology. Adapted from [MAGC14].

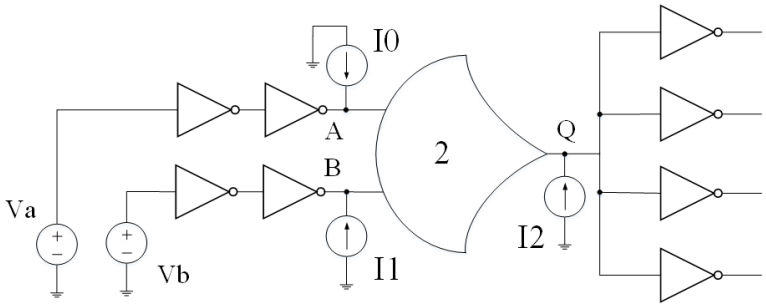


Figure 3.35 – Example of simulation environment for evaluating SEEs in a 2-of-2 NCL gate. Adapted from [MAGC14].

current source described in SPICE. This source was used to simulate the effect of particle strikes in all input and output nodes of the NCL gates while these are kept in memorization states. It is important to clarify that a pair of inverters was inserted in the inputs of the simulated gates, to allow injecting current without interference of fixed input sources, which were employed for feeding the input inverters. Furthermore, four parallel inverters with driving strength identical to that of the gate under simulation were added at the output, to respect the FO4 output load principle. Figure 3.35 shows an example simulation environment, as described for the 2-of-2 NCL gate, where the effect of charge collection for each scenario is generated by two current sources at the inputs (I0 and I1) and one at the output (I2).

Table 3.16 – Input (I.) and output (O.) critical charge for the 2-of-2 case study gates. Adapted from [MAGC14].

Drive	Topology	1 V		0.6 V		0.2 V	
		I. (fC)	O. (fC)	I. (fC)	O. (fC)	I. (fC)	O. (fC)
X2	*Proposed*	22.5	9.3	8.9	4.4	0.1	0.5
	Static	21.7	20	8.6	8.5	0.1	0.5
X7	*Proposed*	-	12.6	21.7	5.7	0.1	1.1
	Static	-	-	22.1	23.6	0.1	1.1
X13	*Proposed*	-	17.2	-	8.3	0.1	1.7
	Static	-	-	-	-	0.1	2.3

Table 3.17 – Input (I.) and output (O.) critical charge for the 3W22-of-4 case study gates. Adapted from [MAGC14].

Drive	Topology	1 V		0.6 V		0.2 V	
		I. (fC)	O. (fC)	I. (fC)	O. (fC)	I. (fC)	O. (fC)
X2	*Proposed*	22.2	9.8	9	4.1	0.1	0.4
	Static	23.1	20.3	9.4	8.8	0.1	0.2
X7	*Proposed*	-	12.6	21.9	5.8	0.1	0.5
	Static	-	-	22.8	23.8	0.1	0.5
X13	*Proposed*	-	17.3	-	8.4	0.1	1
	Static	-	-	-	-	0.1	1

Table 3.18 – Input (I.) and output (O.) critical charge for the AO2-of4 case study gates. Adapted from [MAGC14].

Drive	Topology	1 V		0.6 V		0.2 V	
		I. (fC)	O. (fC)	I. (fC)	O. (fC)	I. (fC)	O. (fC)
X2	*Proposed*	22.6	9.8	9.2	4.1	0.1	0.2
	Static	22.3	20.0	9.0	8.6	0.1	0.2
X7	*Proposed*	-	12.6	22.3	5.7	0.1	0.5
	Static	-	-	21.6	23.9	0.1	0.6
X13	*Proposed*	-	17.2	-	8.4	0.1	1.0
	Static	-	-	-	-	0.1	1.1

Using this model, different scenarios were simulated, where the collected charge Q varies from 0.1 fC to 30 fC, in 0.1 fC steps. These values are realistic for the target technology, according to the information in the manual of the process design kit (PDK) and private communications with the foundry. All simulation scenarios employed typical fabrication process and operating temperature. During simulation, we measured the minimum collected charge that caused the output of the gate to flip incorrectly, i.e. the minimum collected charge that generated an SEU. This was done for both, injections in the inputs and in the outputs. A first set of simulations were conducted assuming an operating voltage of 1V, as Tables 3.16, 3.17 and 3.18 show.

As the tables show, for this gate set, the static topology displayed superior results in the majority of the scenarios. More precisely, for injections at the inputs, the proposed topology presented results similar to those obtained for the static topology. However, for injections at the outputs, the new topology showed to be much more sensitive to SEEs. In fact, it typically required less than 50% of the charge required by the static topology to produce an SEU. Note that as the driving strength is increased, the minimum charge for generating SEUs can be over the boundary of the performed experiments (30 fC) and is marked with a dash. This is due to increases in input and parasitics capacitances of the gates, which filter transients. Considering the proposed topology, for input injections, bigger driving strengths allow improving robustness, as input and internal capacitances are increased. However, for injections at the output, robustness is only minimally affected by driving strength. This is justified because during memorization states, the integrity of the new topology output relies on minimum sized transistors in a loop of inverters. In this way, transients in the output node easily corrupt its value. For the static topology, the larger the driving strength, the less sensitive it is to radiation effects. In fact, for this topology, the X7 and X13 drive versions did not generate SEUs in any of the simulated scenarios.

Another two sets of experiments were conducted, employing the same simulation environment, but using operating voltages of 0.6 V and 0.2 V. It was then possible to evaluate the robustness of the topologies when operating at low voltages. The obtained results are also summarized in the following columns of Tables 3.16, 3.17 and 3.18. As these Tables show, when the gates operate at 0.6V, results similar to those observed for 1 V appear, and the static topology tolerates bigger charges than the proposed topology. However, when operating at minimum voltage levels (0.2 V), the proposed topology presents a robustness similar to that observed for the static topology. This confirms the suitability of the proposed topology for low voltage operation.

3.4 Discussion

Asynchronous components are the building blocks of asynchronous circuits. In this way, their characteristics have a substantial impact in the quality of such circuits. Here we explored the trade offs in C-elements design, providing a set of guidelines for designers using these components. In essence, there is a trade off between speed, area, energy and power when selecting a topology for building C-elements. The semi-static topology should be used whenever high density is required, the static one is more suited for low power applications, as it presents lower energy consumption and static power in average. The symmetric topology is more suited for high speed applications where the area is not a concern, given its overhead in transistors count. Our experiments were all based on post-

layout simulations and relied on rudimentary synthesis environments that force the use of C-elements in case study circuits.

As for low voltage operation, the static and symmetric topologies are the ones that are more suited, while the semi-static complicates the design, due to its constrained nature. In fact, the correct behavior of the latter relies on strict relationships between the driving capability of its transistors. Measurements point that the most efficient topology for low voltage applications is the symmetric, as it presents the overall best speed-energy and speed-leakage efficiency with acceptable losses in speed-area efficiency. In this way, we advise low voltage asynchronous circuits designers to choose the symmetric topology in general. Also, our results suggest that C-elements present best speed, energy and leakage cost benefits when operating near the threshold voltage.

Our experiments evaluating the C-element topologies indicate that having the three topologies available in cell libraries provides more flexibility to designers. Interestingly, these experiments extend to NCL gates design, as they rely on the same topologies for their implementation at transistor level. In this way, our analyses also provide guidelines for building these gates. Another point that we explored for NCL gates design is their susceptibility to charge sharing effects. In fact, albeit there is a wide variety of works available in the literature that deal with soft errors problems, most of these focus on problems other than charge sharing. In special some works approach soft errors caused by particle strikes. Examples of these works are presented in [BSK+10, PM05, LM04, KZYD10].

Among these, the work presented in [KZYD10] is particularly interesting. There, the authors detect weak conditions for NCL gates and propose optimizations by using Schmitt-Triggers and transistor resizing. The presented results suggest that for deep submicron technology nodes, NCL gates are very sensitive to particle strikes, even with their optimizations. Yet, they also propose an efficient method to detect and correct soft errors in computational blocks, assuming that registers are error free. However, the use of Schmitt-Triggers and bigger transistors considerably interferes in the performance of the circuit at system level. Also, there is no reference to charge sharing problems that, as it will be demonstrated throughout this work, can also jeopardize the correct functionality of NCL circuits in current technologies. Additionally, differently from soft errors caused by particle strikes, those caused by charge sharing require no external interference and are inevitable in CMOS design [KH04].

Here, we presented an analysis of the analog behavior for NCL gates in the presence of hazards caused by charge sharing effects, providing more guidelines to building these components. Three different transistors arrangements were proposed and better robustness in absolute terms was verified for arrangements A1 and A2. In this way, and given that the only difference between the arrangements is the transistor topology (there is no area overhead associated), we strongly recommend the use of A2 or A3. In fact, while building the library presented in Section 4.3 we observed the importance of our guidelines. Before defining our charge-sharing-aware design approach we had some NCL gates available in

our in-house cell library. These gates were being laid out and when we characterized them we noticed that the characterization tool could not simulate some scenarios. A deeper analysis of the cause of that showed that these scenarios were being affected by charge-sharing effects. After adopting our set of guidelines, and redesigning the cell, this problem was solved.

We also proposed a new topology for designing NCL gates. Experimental results indicate the suitability of the topology to low voltage applications. Accordingly, when operating at minimum voltages, it provides improvements in speed, energy and leakage trade-offs while maintaining robustness against single event effects at levels similar to the classic static topology. Unfortunately, this topology was proposed in the last years of this work and could not be further explored, as a cell library was already being devised using the static topology. Having that said, the availability of this new topology allows better design space exploration for asynchronous circuits in general. That is because this topology is not only useful for building NCL gates, but also C-elements, which are, in fact, a special case of NCL.

Finally, all results presented here assume the use of the design flow explored in Chapter 4. In this way, some of our analyses are based on assumptions of transistors size, as defined by the flow. However, this does not compromise the generality of this evaluation, because the flow ensures the best dimensioning of transistors, through simulation, for a specific cost function. Furthermore, the cost function we used in all cases was the best energy and delay product, which is a typical metric used for the dimensioning of transistors in contemporary technologies.

4. INNOVATIONS IN ASYNCHRONOUS CELL LIBRARIES DESIGN

This Chapter presents another set of contributions of this thesis, in the form of an automated design flow to devise cell libraries for asynchronous circuits design and a complete library designed using this flow. Section 4.1 explores the state-of-the art in designing cell libraries for asynchronous circuits. The remaining of the Chapter proposes and details the different parts of the proposed flow, highlighting novel solutions for coping with issues specific to asynchronous cell design. We then present a library designed using the flow.

4.1 State-of-the-Art in Cell Libraries for Asynchronous Design

A variety of templates that can be employed for designing asynchronous circuits exists. Each of these templates can rely on a different design style, which, in turn, can require different cells for their implementation. Therefore, a library that is sufficiently generic to support different templates must usually comprise a large set of specific cells. It is implicit in this discussion that templates often require cells different from those available in conventional cell libraries. In this way, building a cell library for asynchronous design is a challenging task. In fact, such cells are typically designed on demand for the target template and sometimes even built on demand for a specific circuit design process [DLD+14]. The drawback is that the tools used for dealing with conventional standard cells usually fail to recognize the functionality of several asynchronous cells, which makes the design of a single cell challenging. For example, asynchronous sequential cells like C-elements are not easily (if ever) manipulated with cell characterization environments like ELC from Cadence. In view of these issues, two major categories of work are addressed in this Section that propose: (i) libraries of asynchronous cells; and (ii) tools for automating the process of designing such cells.

4.1.1 TIMA Asynchronous Library

The only fair way to allow widespread experimentation with asynchronous application specific integrated circuits (ASICs) is to have available an asynchronous standard cell library. 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 [MRB+03]. 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). The strategy these authors adopted for sizing the transistors

of the cells was to balance rise and fall propagation delays using first order models. The decision was made to balance the latency to propagate spacers and valid data through a circuit that employs these cells. Their sizing strategy also targets reducing the input capacitance of the gates, to optimize energy-efficiency and performance. Furthermore, the authors define guidelines for transistors arrangements that are specific for the target cells. All the work in transistor sizing is manual. For the cell layout, on the other hand, the authors relied on an in-house tool for automatic layout generation. An information missing in this work is about the availability of information on whether timing and power models are available for these cells. The approach used to design circuits using this library was to build schematic VHDLs, instantiating cells of the library. Such choice eliminates the need for high level models of the cells' characteristics. Of course, this has a substantial impact on design efficiency, as it reduces the support of EDA tools.

A total of 30 cells comprise this library that was employed in asynchronous designs using flows like the one described in [FBFR07]. Furthermore, at present, a 65 nm transistor gate length version of this library has also been designed. This library is called TAL-65, with 150 cells, and is fully validated through simulation and silicon implementations of different QDI designs. Two cell sets compose the library, a set of C-elements and a set of latches. Several variations of C-elements are available, which are typically required for building control blocks. Among these are two-, three- and four-input cells, including settable and resettable gates.

One major problem is that the TAL libraries are not freely available. The information obtained about it was a courtesy of TIMA and LETI to the author. Furthermore, there are little degrees of automation in the flow employed for the design of these libraries. In this way, porting them to other technologies can require laborious manual work. Another weakness is that there are no power and timing models to be used in more modern synthesis approaches.

4.1.2 University of Utah Asynchronous Cell Library

In the work presented in [GB05], Gulati and Brunvand explore a solution to cope with the reduced support for semi-custom asynchronous design. According to the authors, asynchronous cells can be implemented using conventional cell libraries. However, using conventional cells for that purpose not only increases design time but also severely impacts design performance. For example, a full-custom C-element cell is typically 40% more area-efficient and up to 30% faster than a C-element designed using conventional cells.

To overcome this limitation, they identify the cell set required for designing the blocks that compose the architecture of asynchronous microengines – a specific template. Then, they design these cells targeting a 0.5 μm technology and use it to fabricate a test-chip that implements the control path of a differential equation solver. The authors also design the

same circuit using conventional cells and compare it to the one that uses their asynchronous cells. As expected, they report a high degree of optimization in the blocks that compose the circuit, enabled by the use of the designed cells, which includes improvements of up to 74% in area and 51% in speed.

The authors also report a specific design flow adopted for designing the cell library. They claim that the design of asynchronous standard cells can involve different challenges, when compared to the design of conventional logic cells, and that special care must be taken to the design flow. Still according to these authors, the main differences between conventional and asynchronous cells design arise due to the special constraints imposed on the latter by the asynchronous design paradigm. Such constraints increase design complexity, causing overheads in the library design time. They dimensioned the transistors of the cells using logical effort [RCN03], ensuring that all constraints were respected. Another challenge mentioned was the cells' layout. According to the authors, many of these cells perform complex functions that require many transistors, complicating the cell internal routing. To cope with that, they employed double height cell templates in cases where internal cell routing was considerably constrained.

Albeit a nice analysis of the challenges on how to design asynchronous cells is presented in this work, the authors target the support of a single asynchronous circuit template. In this way, the usage of this library is considerably constrained, and compromises the design space for asynchronous logic. Furthermore, the adopted flow involves a gret deal of manual work, increasing design time for other asynchronous cells.

4.1.3 USC Asynchronous Cell Libraries

The University of Southern California (USC) Asynchronous CAD and VLSI group has an extensive work on asynchronous design. It has reported the design of at least two cell libraries for asynchronous templates. The first library was designed in the context of Ferretti's thesis [Fer04] and includes a set of basic cells for implementing circuits using a specific QDI template called *single-track full buffer* (STFB). In his thesis, Ferretti describes all techniques employed for transistor sizing, noise and performance analysis during the development of the cell library. For the sake of validation, a test chip was fabricated using the cells in a design with 260,000 transistors [Fer04].

The library is freely available through MOSIS and targets the Taiwan Semiconductor Manufacturing Company (TSMC) 25o nm bulk CMOS process. A drawback is that there are no power or timing models available for the cells, as reported in [Fer04]; only layout, schematic and symbol views are available. This limitation compromises the levels of automation that can be adopted for designing circuits targeting the library. Also, the design of the library was completely handcrafted and adding cells or extending the library to other

technologies can be very laborious. Another drawback is the lack of EDA tools for the STFB template. In other words, the usage of this template is practically limited to full custom design approaches based on a set of available cell layouts.

A second library, designed by the same group, is reported in [BDL11]. In this work, the authors describe an automated design flow that targets semi-custom PCHB design [Lin98], and a library of basic components that supports it, implemented at the cell level. The interesting fact about this library is that PCHB design is usually associated to high-speed circuits, providing definite advantages to asynchronous design. Besides, cells of this library served to design several test chips in the Fulcrum Microsystems enterprise and several blocks from a high speed commercial switch from Intel, the current proprietary of the flow to design PCHB circuits. In fact, a recent work shows that the company keeps using PCHB cells for building some commercial designs, as reported in [DLD+14].

The proprietary library was designed for the TSMC 65nm bulk CMOS process, targets 1-of-2 4-phase QDI and, according to Beerel et al. in [BDL11], implements all 2-bit and 3-bit logic functions. The library also includes control circuit cells and C-elements with up to 4 inputs. Dedicated cells for implementing buffers are also included, together with specialized scan cells for enabling the testability of circuits built with the library. An advantage is that there is a tool for characterizing PCHB basic cells, as reported in [Pra07]. Unfortunately, characterization does not include dynamic and leakage power metrics. Other drawbacks are that the library is proprietary and there is no reported support for automatic transistor sizing and cell layout generation. In this way, access to this library is very limited and extending it to other technologies can be very laborious.

4.1.4 CellTK

Due to the limited availability of cell libraries for asynchronous design, Karmazin et al. proposed cellTK in [KOM13]. The idea is to avoid the need of a cell library by having a flow that automatically generates the layout of asynchronous cells on demand. In this flow, the proposed tool, cellTK, automatically implements the physical layout of asynchronous netlists. A drawback is that this generator is not compatible with semi-custom techniques and tools for asynchronous design automation proposed to date. This is because it employs a non-standard flow, rather than a cell-based semi-custom flow. Unfortunately, this limits the design space for asynchronous logic design.

Another drawback is that the proposed flow does not provide any support for automatic transistor sizing. Also, there is no discussion on the available reference about the optimizations at transistor level for the proposed flow, which may limit the design space and quality of the generated layouts. Moreover, there is no mention about how the generated layouts can be characterized to obtain power and timing models, which can limit its usability

to the proposed flow only. In other words, a designer cannot employ this flow for designing a cell library without relying on extra tools for transistor sizing, optimization and electrical characterization. Another drawback is the area, energy and delay penalties imposed by cellTK, as reported by the authors, 51%, 12% and 9% in average, respectively, when compared to a manual design.

4.1.5 Characterization Flow by Prakash

The work by Prakash, presented in [Pra07], also tries to alleviate the problems of the lack of support for semi-custom asynchronous circuits design. In her work, she presents a tool for the electrical characterization of asynchronous cells and a flow to the generated models in synthesis frameworks. The generated tool is employed to characterize a set of cells, used in a semi-custom asynchronous circuits design flow using the Synopsys framework. The reported results estimate an error during the characterization process of 7.1%. Unfortunately, the tool was originally devised for a STFBs [Pra07], and has many features specifically designed targeting cells for that template. In this way, the generality in using the tool to characterize other cell types is not clear. Furthermore, the tool still lacks precise characterization of power metrics.

4.1.6 Discussion

It is widely accepted that cell-based design was one of the key factors for the success of the semiconductor industry [RCN03]. This design style enables companies to focus on the design of complex ICs at the system level, rather than focusing on the specifics of how to design each transistor or gate in an IC design. Furthermore, standard cell libraries have been validated along several decades as they are used in synchronous designs. Unfortunately, asynchronous designers do not have the same privileges, as they need to design their own cell libraries and synthesis flows to implement asynchronous logic. For example, in the case discussed above, TIMA and LETI developed the TAL library [MRB+03], for working with asynchronous designs. Intel uses PCHB libraries, originally designed at USC, for commercial asynchronous circuits [DLD+14]. Furthermore, different companies rely today on in-house libraries for exploring asynchronous circuits design. Examples of such companies are Tiempo [Tie, RF12], Achronix [Ach] and Wave [Wav].

The result of this lack of support for semi-custom asynchronous circuits design is that designers can spend too much time building the infrastructure for designing the circuit, before they can even start working on the circuit itself. Moreover, because different designers can decide to adopt different asynchronous templates, each of them needs to rebuild a

whole new environment for designing a circuit for a specific template. By doing so, the time available for exploring optimizations at the IC at system level, and the quality of the resultant IC is reduced. In this way, there is a clear need for making asynchronous cell libraries more widely available. However, as explained before, the problem is that each asynchronous template can require a different set of cells. Hence, the design of a single encompassing library is a very challenging and laborious task. An alternative to cope with the problem is to have an automated flow for generating asynchronous cell libraries. This was the starting reasoning and motivation for the contributions described in this Chapter.

Before the GAPH research group could experiment with asynchronous design, it detected the need for a cell library to avoid the complexity of full custom approaches. To ease the generation of cell libraries for asynchronous circuits design, so that different templates could be explored, the ASCEnD flow was first developed. The original version of this flow was proposed by the Author in [Mor10]. In its first version, the flow had little degrees of automation and designing cells was a time consuming task. In fact, it defined some guidelines and the only automation it counted with was for transistor sizing. The evolved flow described here adds new tools so that all steps are now automated. The new flow, ASCEnD-A, is discussed in the remaining of this Chapter.

4.2 The ASCEnD-A Flow

Designing the special cells required by asynchronous circuits is not an easy task. These cells require considerable manual effort. The ASCEnD-A flow was devised to automate this design process. As Figure 4.1 shows in colored rounded corners rectangles, the flow comprises four major action blocks:

1. *Cell Library Templates* corresponds to the process of generating template files that will be used throughout the design flow. This block executes a single time for a whole library, as it generates the template files of all cells that in that library. As inputs, it requires a template library and a set of library and technology specific configurations. The block generates behavioral models for digital simulation, since these are generic enough to be derived from cell template files.

2. *Cell Sizing* is the process of dimensioning the cells of a library. Each cell undergoes the actions in block separately, as individual cells present different transistor arrangements and varying electrical specifications. Cell sizing starts from the templates generated in the previous action block and its output is a transistor level schematic described in Spice, for analog simulation. The process requires the definition of a cost function, which will guide the decision process about specific transistor sizes. This block re-

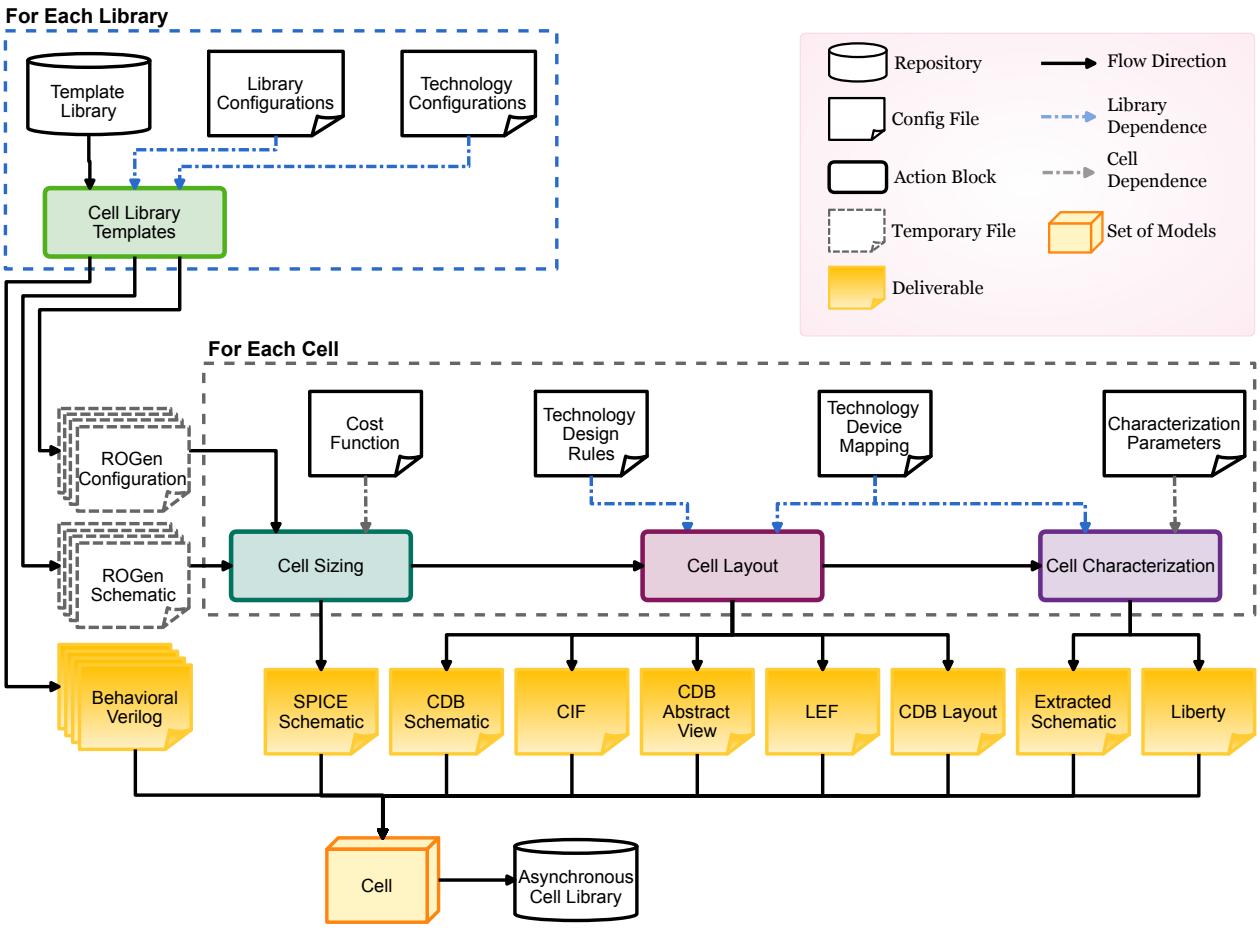


Figure 4.1 – Overview of the ASCEnD-A flow.

lies on two in-house tools, one called Ring Oscilator Generator (ROGen) and another called Cell Specifier (CeS) described in more detail in Section 4.2.2.

3. *Cell Layout* is the process of generating the physical views of cells. Because each cell has distinct transistor arrangement and sizing, each cell undergoes actions in this block individually. It starts from the Spice schematic generated in the Cell Sizing block and generates a set of models for physical synthesis, using a set of technology-dependent configurations. Accordingly, the block generates the layout, the abstract view and the schematic in the Cadence Data Base (CDB) format for use by physical synthesis flows. The block also exports the layout and the abstract view to text based files, respectively to the Caltech Intermediate Form (CIF) and to LEF. This block relies on a tool called AS-TRAN [Zie14, ZR14], developed in a fellow research group and adapted to ASCEnD-A in the context of this thesis.

4. *Cell Characterization*, is the last action block, and consists in obtaining timing and power figures for the cells, which are necessary for by logic and physical synthesis tools. This step is also performed individually for each cell, because these need to be separately characterized and have its figures made available for the synthesis flows.

The starting point of the cell characterization block is the layout generated in the previous action block. From this layout, the equivalent cell is extracted and exported to a Spice annotated schematic. This schematic contains all the devices in the layout, including parasitics, and is the source for electrical characterization tasks. After characterizing the extracted schematic, the block exports the obtained figures to a Liberty file, which is employed in synthesis flows. Note that this block relies on technology specific information and on a set of characterization parameters, which are library-dependent and can vary among distinct cells. This block employs an in-house tool called LiChEn, developed in the context of this thesis.

After a cell goes through all the last three action blocks of the ASCEnD-A flow, it has all its deliverable files available to form the library. Such deliverables are: behavioral verilog views for digital simulation spice, CDB and extracted schematics for analog simulation, CIF and CDB layout views for masks specification, CDB abstract view and LEF for place and route and Liberty for STA and synthesis tasks. The remaining of this Section details each of the action block. Appendix A shows an example of the generation of a cell using the flow. The example provides a step by step demonstration of the generated views at each stage of the design flow.

4.2.1 Cell Library Templates

As Figure 4.1 shows, the first action block in the design of a cell library using the ASCEnD-A flow is the generation of a set of cell templates. Figure 4.2 details this block. These templates are the source for conducting all the tasks in the design flow. They can be automatically generated using a set of scripts available in ASCEnD-A coupled to a technology independent template library and a set of technology and library-dependent configurations. The strategy enables high modularity and reuse in cell libraries design, easing the task of porting libraries from one technology to another.

A Template Library is a library composed of a set of schematics, or cell templates, described in Spice, which defines the functionality of the cells to design, using the following conventions:

- NMOS transistors use “NFET” as their model;
- PMOS transistors use “PFET” as their model;
- The width of driving transistors is defined as “ninv” for NMOS transistors and “pinv” for PMOS transistors;
- The width of feedback transistors is defined as “nfback” for NMOS transistors and “pfback” for PMOS transistors;

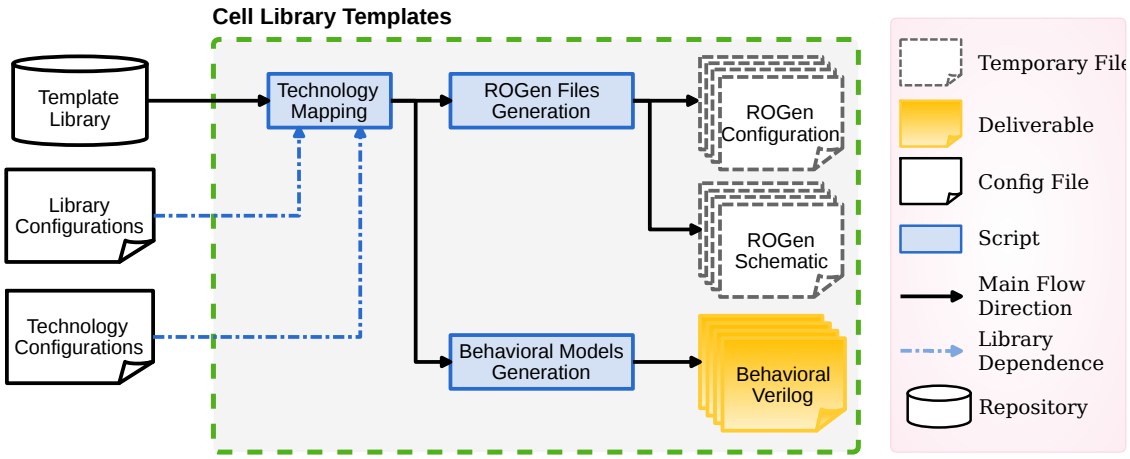


Figure 4.2 – Details of the Cell Library Templates action block, responsible for the process of library template generation in the ASCEnD-A flow.

- The width of transistors to be dimensioned is defined as “nparam” for NMOS transistors and “pparam” for PMOS transistors;
- The length of all transistors is defined as “length”.

A set of comments in the end of the Spice description are also required to allow the generation of configuration files for the remaining tools of the flow. These comments are:

- *ROGen:T *X*, where *X* defines the threshold of a logic function, used for NCL cells;
- *ROGen:I *X*, where *X* defines the number of inputs of a cell;
- *ROGen:P *list*, where the list defines the direction/configuration of each pin of the cell, which can be *in* (for inputs), *out* (for outputs), *gnd* (for ground reference), *gnds* (for body ground reference), *vdd* (for supply voltage reference) and *vdds* (for body voltage reference);

Back to Figure 4.2, from a template library three scripts automate the process of generating ROGen configuration files and schematics, which combination is called ROGen templates. Note that these scripts also generate behavioral Verilog models. The ROGen templates are the source for conducting the following steps of the flow. To generate them, the scripts also need library and technology configurations. The required technology configurations are:

- Models for Spice simulation;
- Operating Voltage.

Also, the following library specific configurations must be provided:

- An inverter for using as driving cells and fan-out loads, which can be obtained from a core library;
- The fan-out of the cells under design;
- A control NAND gate, which can be obtained from a core library, that will be used for generating a ring oscilator;
- Minimum and maximum PMOS and NMOS transistors dimensions that fit inside a cell;
- Minimum and maximum dimensions and increase step for dimensioning PMOS and NMOS transistors, which defines how the W of transistors is increased during the sizing of transistors;
- Time windows for measuring leakage and dynamic power;
- Voltage levels for valid logic 0 and 1;
- Voltage level for identifying rising and falling transitions;
- Reference driving strengths;

4.2.2 Cell Sizing

The next action block in the flow (Cell Sizing) consists in dimensioning the transistors of each library cell. Figure 4.3 details this block. In the block, a simulation environment is generated using ROGen, an in-house tool, based on the ROGen templates mentioned in Section 4.2.1. ROGen reads the provided configuration file and uses the schematic to construct a ring oscillator and a simulation deck based on the cell to size [Mor10, MOPC11].

Figure 4.4 shows the generic schematic of the ring oscillators produced by ROGen. As the Figure shows, the circuit consists of a ring composed by instances of the cell which transistors are to be dimensioned and a control NAND gate. The NAND gate is responsible for controlling when the ring oscillates and when it remains stationery. When the *IN* pin of the NAND is at 0, the ring stays in a quiescent state. However, when *IN* is at 1 the ring oscillates. Albeit the number of cells in the ring can be configured, simulations showed that a ring of 4 cells is sufficient for collecting the required data without compromising generality. Also, as the Figure shows, the inputs of a cell are usually tied to a single node, the output of the previous cell. However, depending on the number of inputs of the cell, this strategy can lead to a fan-out larger or smaller than that defined in the configuration file. If a larger fan-out is required, ROGen adds inverters to the outputs of the cells to ensure the fan-out of the cell is as specified. If a smaller fan-out is required, ROGen builds a buffer tree between each two contiguous cells.

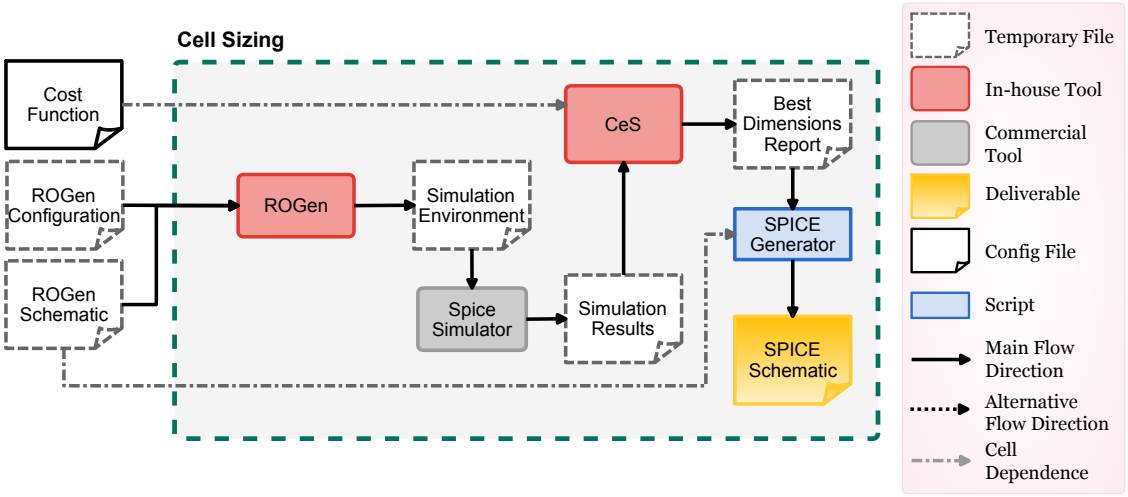


Figure 4.3 – Details of the Cell Sizing action block of the ASCEnD-A flow.

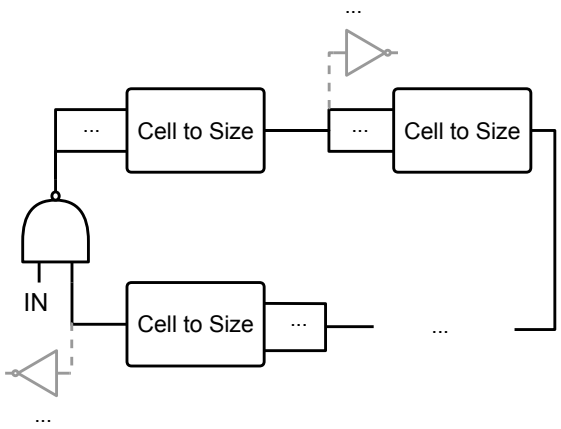


Figure 4.4 – Controlled ring oscillator produced by ROGen to perform transistor sizing simulations.

ROGen can also deal with NCL cells [MOPC13a] and it can read the function of these cells to guide the connection of their input and output pins to compose an operational ring oscillator. To do so, it computes the threshold of the function and ensures that only m input pins, where m is the threshold, are connected. The remaining pins are tied to 0. By doing so, it avoids optimistic sizing of the transistors, avoiding that branches in the schematic be activated together in pull-down and pull-up networks.

Once the ring is ready, ROGen creates the simulation deck and adds it to the simulation environment. This deck uses the configurations provided by the user to vary transistor sizes and collect performance figures during simulation. Transistor sizes are exhaustively varied according to the specified minimum and maximum dimensions for PMOS and NMOS transistors in steps defined by the designer, all specified in the configuration file. For these variations, the tool combines all NMOS and PMOS dimensions available.

For collecting performance figures, it relies on *.measure* Spice constructs that allow measuring the following parameters:

- Dynamic power: measured as the average current on the power source while the ring is oscillating, multiplied by the operating voltage;
- Leakage power: measured as the average current on the power source while the ring is quiescent, multiplied by the operating voltage;
- Rise transition delay: measured as the time it takes for the output of a specific cell on the ring to perform a full swing from logic 0 to 1 (a rising transition);
- Fall transition delay: measured as the time it takes for the output of a specific cell on the ring to perform a full swing from logic 1 to 0 (a falling transition);
- Rise propagation delay: measured as the time it takes for a rising transition in the input of a specific cell on the ring to propagate to the output of the cell;
- Fall propagation delay: measured as the time it takes for a falling transition in the input of a specific cell on the ring to propagate to the output of the cell;
- Ring propagation delay: measured as the time it takes for a rising transition to propagate through the whole ring.

Note that for measuring power figures, ROGen isolates the power supply of the cell to size and the one from the fan-out fixing inverters and the control NAND gate. This allows more precise measurement without the impact of these extra cells. Furthermore, it controls how long the ring will be quiescent using the parameters provided by the designer in the configuration file. For the delays, the tool generates measurements using the cell in a specific level of the ring. This is also defined by the designer in the configuration file. Ring propagation delay can be a useful measurement if the designer is not interested in specific rise and fall delay relationships in the cell to size. However, this parameter is affected by the delay of the control NAND gate. To amortize this effect, the ring can employ a larger number of stages. Simulation results indicate that 4 cells are sufficient to ensure precise measurement of this figure.

The environment generated by ROGen is simulated using Spice. In its current version, ASCEnD-A is scripted to employ the Cadence Spectre simulator. However, other flavors can be employed with some adjustments. During the simulation, Spice produces a text file containing results. This file is the source for a next in-house tool called Cell Specifier (or CeS) [MOP+11], which selects the best transistor dimensions based on the collected simulation results. To select these dimensions, the tool requires a cost function, also specified by the user. This function can include the 7 different variables measured by the simulation of the environment generated by ROGen and the following arithmetic operators: +, -, / and *.

The designer can also use parenthesis and vertical bar delimiters. The former allow ensuring the order of operations and the latter return absolute values only. From this cost function, CeS will generate the best NMOS-PMOS size matches in descending order, according to the cost function.

This allows exploring trade offs and cell sizing design space. For instance, a cost function F_1 can focus only in high speed e.g.:

$$F_1 = \frac{1}{prop\_rise + prop\_fall} \tag{4.1}$$

where *prop_rise* and *prop_fall* correspond to rising and falling propagation delays, respectively. Another possibility is trying to achieve the best trade off between delay and power. The example cost function F_2 below illustrates this.

$$F_2 = \frac{1}{(prop\_rise + prop\_fall) * dynpwr} \tag{4.2}$$

Here *dynpwr* corresponds to the measured dynamic power. Also, the tool allows the use of incremental analysis. To do so, a first cost function can be defined to select only the best NMOS-PMOS size matches. Next, another cost function is applied to that set. For instance, assume that the designer defines an incremental analysis, providing the following initial cost function F_{3i}:

$$F_{3i} = \frac{1}{|prop\_rise - prop\_fall|} \tag{4.3}$$

This will select only NMOS-PMOS size matches that allow balanced rising and falling propagation delays. Next, if the designer provides F_3:

$$F_3 = \frac{1}{(prop\_rise + prop\_fall) * leakpwr} \tag{4.4}$$

where *leakpwr* is the measured leakage power, CeS will select the best NMOS-PMOS size match, from the previously selected set, which presents the best delay and leakage power trade offs. In other words, the selected match will present the best delay and leakage power trade off among the most balanced matches, in terms of rising and falling propagation delays. Using cost functions, the designer can specify different versions of each gate. This enables enriching the library.

Note that the approach of exhaustively simulating all possibilities of NMOS and PMOS combinations and then collecting the simulation results ensures that the designer will have the best transistors sizes for the specified set of variations. This set is composed using the minimum and maximum dimensions and step sizes for transistors. Albeit the approach leads to computing intensive tasks, this step is done just once for each cell of the library and is fully automated. Furthermore, the set of variations can be significantly reduced

with preliminary simulations and analyses that allow define tight bounds on minimum and maximum transistors sizes.

From the dimensions identified by CeS, a script automatically generates a Spice schematic of the properly sized cell. This schematic can be used for electrical simulation and netlists composition for circuits that employ the designed cell. Also, this schematic is the basic input for the next step of the flow.

4.2.3 Cell Layout

After dimensioning the cell transistors, the next action block generates the cell layout [MAZ+14b], which is detailed in Figure 4.5. To do so, the ASCEnD flow integrates [ZRM+14a] the netlist-to-layout tool ASTRAN [Zie14, ZR14]. The flow is in this way capable of synthesizing cell layouts from the transistor level schematic, using the dimensions defined by the CeS tool. ASTRAN can produce the layout of any transistor network as described in [ZR14]. The tool supports unrestricted circuit structures, continuous transistor sizing, folding, poly and over-the-cell metal 1 routing, redundant contacts insertion, minimum distance relaxation for DFM and conditional design rules. It features a transistor placement algorithm for width reduction and an intra-cell router. ASTRAN employs mixed-integer linear programming for two-dimensional compaction. It produces the final layout according to the results obtained from the placement/routing steps and respects the technology design rules. As Figure 4.5 shows, after ASTRAN generates a layout, the Cadence Framework allows to edit layouts when necessary and perform verification and extraction tasks. Other commercial frameworks can be adapted with some adjustments to run tasks conducted today within the Cadence framework.

The input to ASTRAN is a transistor level description of a circuit in Spice format, which is automatically generated by a script that reads the dimensioned Spice schematic, output of the Cell Sizing action block. Technology rules are set according to the values specified by the foundry. ASTRAN supports design rule sets from most processes down to 45*nm* [ZR14]. The cell topology (height, routing grid, wells/power rails position and other library specific aspects) usually follows the templates defined in the foundry-furnished library. Figure 4.6 illustrates the flow employed by ASTRAN to generate layouts.

The ASTRAN flow makes use of the one-dimension (1D) layout style, which consists of two transistor rows (one for PMOS and one for NMOS transistors) with vertical gates as Figure 4.7 shows. To create the layout ASTRAN first estimates the maximum transistor width and the routing resources to create a graph that can be compacted to the layout level afterwards. Because the tool can under- or over-estimate the number of horizontal tracks that fits in the cell height, the designer is advised to start with an optimistic approach (maximizing both the PMOS/NMOS diffusion regions and number of tracks) and then be-

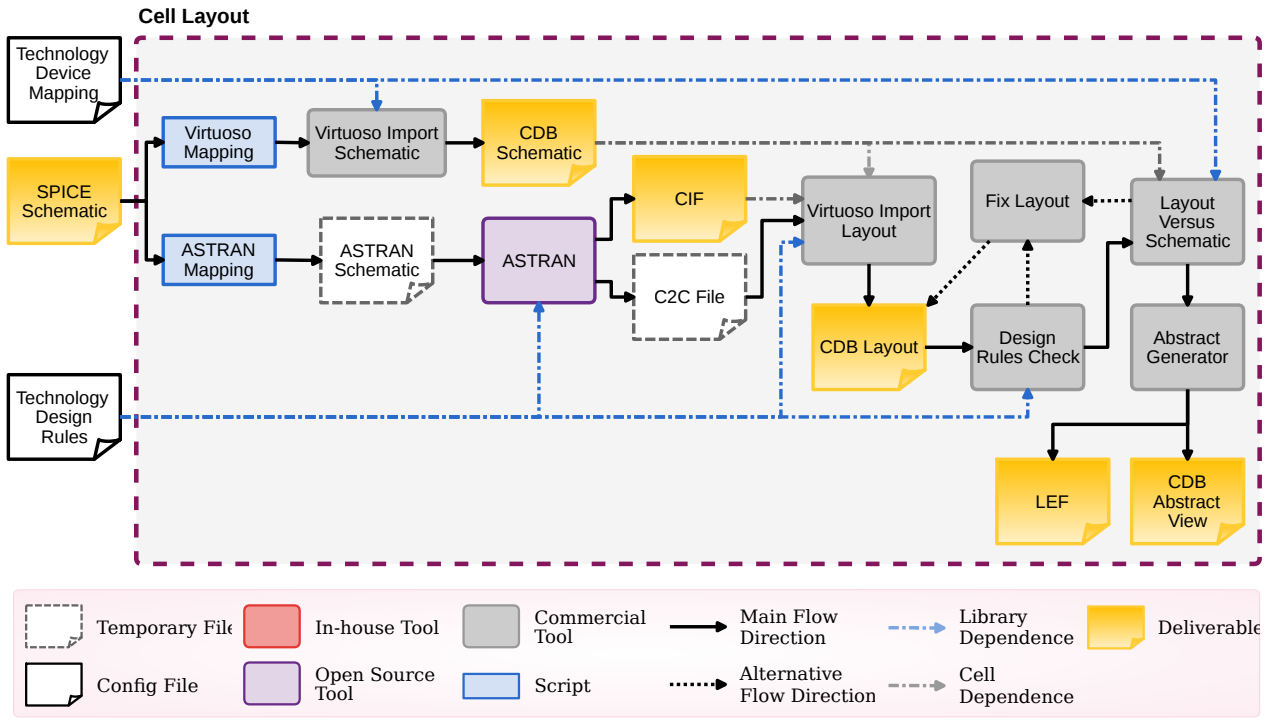


Figure 4.5 – Details of the Cell Layout action block, used for layout generation in the ASCEnD-A flow.

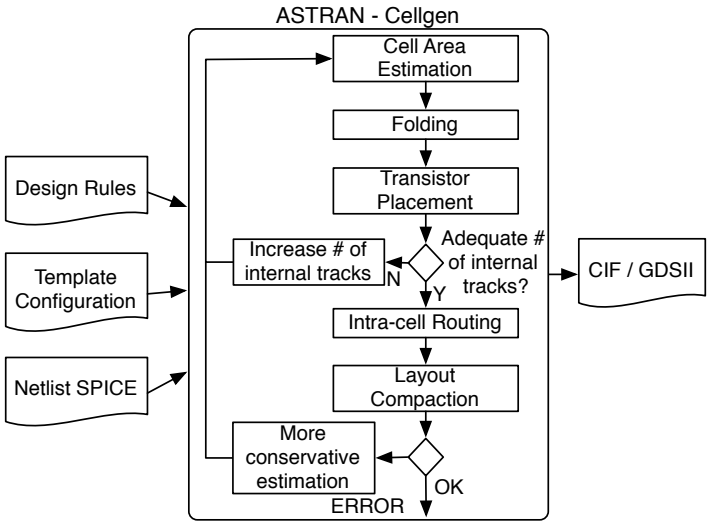


Figure 4.6 – The cell generation flow employed by ASTRAN [ZRM+14b].

come more conservative (reducing the number of tracks and the size of the PMOS/NMOS regions) if the tool is not able to compact the layout. Such choices are parameterizable for each layout.

Given the diffusion row height limits, computed during the previous step, transistors can require folding. ASTRAN folds transistors by modifying the cell netlist, creating new transistors in parallel, before the execution of the placement step. Next, transistors can be placed; to do so, the tool needs to find out a transistor ordering for the PMOS and NMOS

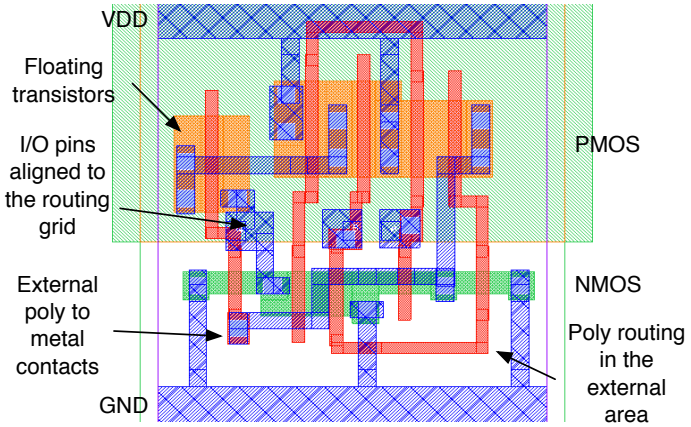


Figure 4.7 – ASTRAN layout style [MAZ+14b].

networks that leads to optimal design. The ordering is obtained through a multidimensional cost function specified to evaluate the placement quality, as detailed in [ZRM+14a]. The abstract cell layout representation obtained in the placement step is translated into a routing graph.

The graph is generated according to the number and position of the tracks calculated in the cell area estimation step, the placement result and the width of the transistors. Layout compaction is the process of translating the abstract cell representation produced by the previous steps into the cell layout. ASTRAN compacts the layout in 2-D simultaneously using mixed-integer linear programming.

Recently [ZRM+14b] a new feature that allows ASTRAN to automatically iterate with different parameters in case it fails to complete the cell generation process was added to the flow, as Figure 4.6 shows. This incremental generation flow starts with an optimistic approach, attributing the maximum height for the PMOS and NMOS regions of the cell in order to minimize the number of transistor folding. After the placement step completes, ASTRAN evaluates the channel density of the current solution and increases the number or internal tracks if it exceeds the capacity of the cell, re-executing the flow from the beginning. At the end, the layout compaction step is called to generate the actual cell layout. Models for which the mixed-integer linear programming solver proves infeasible are aborted and re-started using a more conservative approach: reducing the height of the diffusion areas of the cell and the number of horizontal tracks.

The resulting layouts are exported to CIF or GDSII formats, which are then imported into Cadence where DRC, LVS and extraction can be performed with conventional tools. In order to automate the cell layout generation and integrate it in the ASCEnD-A library, a set of scripts imports the input schematic and the layout generated by ASTRAN into the Cadence Framework. With the imported schematic and layout, the designer can perform verification and extraction tasks, and adjust the design, when necessary, as explained next.

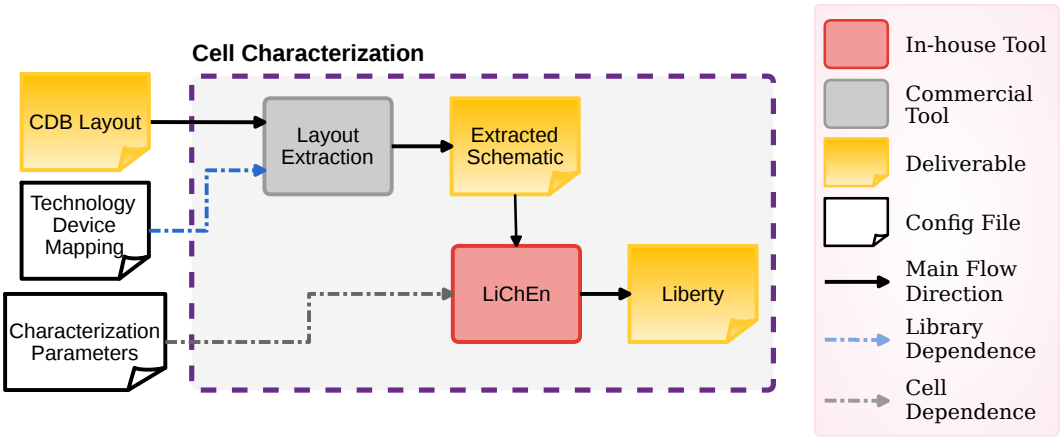


Figure 4.8 – Details of the Cell Characterization action block, required for the characterization tasks in the ASCEnD-A flow.

In fact, layouts generated by ASTRAN can sometimes require small corrections to meet DRC tests [MAZ+14b]. Such corrections are typically the trimming of small corners left exposed during the placement of polygons for the layout composition.

After the layout is fully verified, a physical view is generated with the Cadence Abstract Editor (albeit any other conventional tool could be used) and exported to the Library Exchange Format (LEF), widely accepted by most EDA vendors.

4.2.4 Cell Characterization

From the generated layout, the next tasks in the ASCEnD-A flow are the extraction of the circuit from the layout and the electrical characterization of the cell. Figure 4.8 details this, which is the last action block in the ASCEnD-A flow. For the extraction, the flow relies on commercial tools that read the layout imported to the Cadence framework and generate a Spice description of the extracted schematic. This extracted schematic must comprise the drawn transistors as well as the RC parasitics. Using this latter schematic, the cell is then characterized, i.e. has its power and delay figures measured and exported to a model compatible with synthesis frameworks.

For the electrical characterization of asynchronous cells, a new tool called Library Characterization Environment (LiChEn) [MOCO13] was proposed in the context of this work and is another original contribution of this thesis. The need for LiChEn arises from the fact that available commercial tools fail to recognize the logic of these cells, which complicates or even precludes the conduction of characterization tasks. Before designing LiChEn, two major vendor tools were explored: Cadence ELC [Cad] and Synopsys Liberty NCX [Syn]. For example, consider the use of ELC. Figure 4.9 shows what happens when trying to characterize a 2-input C-element. The tool identifies all transistors, but cannot compute the cell

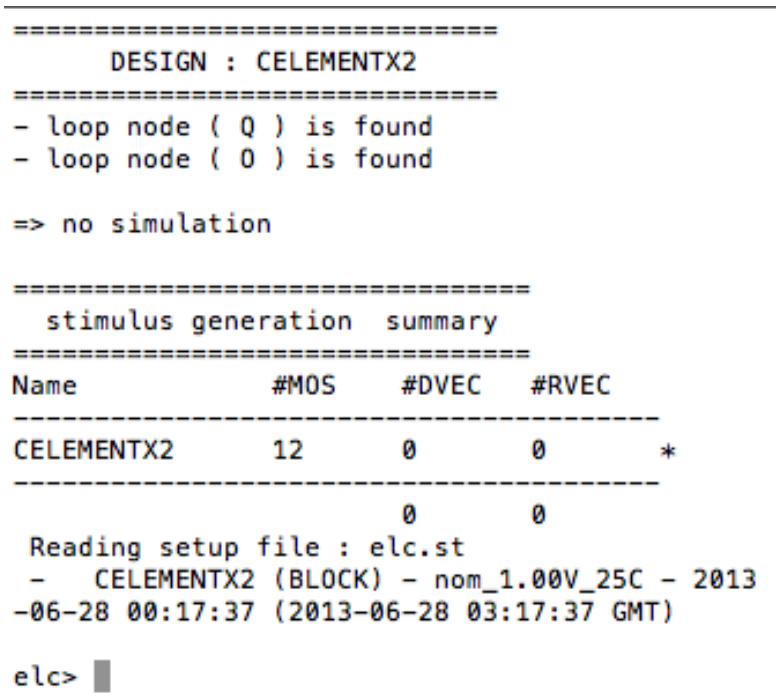


Figure 4.9 – Example of interaction during a tentative characterization of a specific C-element using the Cadence ELC tool [MOCO13].

logic function. When it detects the loop that forms the memory mechanism of a C-element, it stops the characterization process and prompts the user to solve the problem.

The problem is that when the logic behavior of the standard cell is not automatically identified by ELC, a designer needs to manually specify it inside the tool database. This means that the characterization process must stop while the modification is done. It requires the generation of some text files by the tool, which are then modified by the designer, according to a specific syntax, defined within ELC. Once the correct logic behavior is specified, the designer must update the database and the tool can continue the flow. Albeit some scripts could help automating the corrections required to use ELC, these are usually dependent on internal node labels that are only available after layout extraction and vary from cell to cell. Thus, even using such an enhanced approach would lead to extra manual labor, where scripts need to be configured for each cell that ELC fails to recognize. Furthermore, for cells more complex than C-elements, like large NCL gates or MUTEXes, the specification of their functionality in a manner that the tool can compute it is a complicated task. A similar problem was observed when using NCX. Considering an asynchronous library where the problem arises for hundreds of cells, using available commercial tools is not a scalable option, and the need for enhanced tools is clear.

LiChEn is implemented using the C language and is an open source code tool. It is based on the generation of a Spice simulation environment, where each standard cell has all of its arcs and states exercised and its power and timing figures characterized and exported to the Liberty format [Lib]. All commands to LiChEn are given through a com-

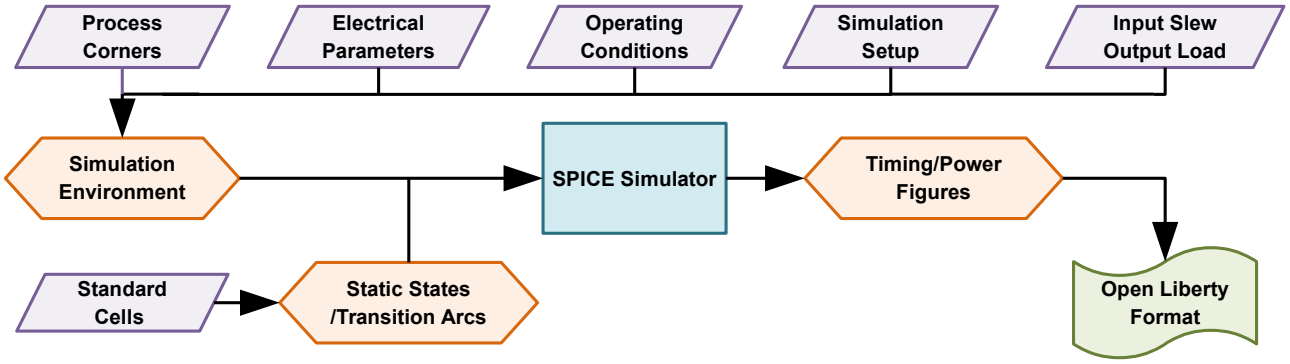


Figure 4.10 – The LiChEn electrical characterization flow [MOCO13].

mand line interface, and text-based scripts can help the automation of standard cell libraries characterization. The electrical characterization flow employed by LiChEn is represented in Figure 4.10. It comprises three main steps, represented by diamonds: (i) producing a cell Simulation Environment, (ii) finding Static States and Transition Arcs, and (iii) generating Timing/Power Figures through Spice simulation.

Simulation Environment

In order to characterize asynchronous standard cells, a simulation environment must be configured. This requires the use of technology-specific data, providing corner selection, operating conditions, electrical specifications, and simulation parameters. First, technology models are furnished to LiChEn, together with the specific corner to use during simulation. Next, electrical parameters for collecting information must be provided, such as the minimum logical 1 and maximum logical 0 voltages, and low-to-high and high-to-low transition thresholds. The precision of the results generated by LiChEn for the provided technology depends on the quality of choice for these parameters. The tool also requires information about operating conditions, nominal voltage and temperature, and global power nets like *Vdd* and *Gnd*, which feed the standard cells. Afterwards, parameters to control the simulation must be furnished, including maximum simulation time, the specification of the moment when to start measuring the information and the simulation minimum step.

Pin-to-pin propagation, transition delays and dynamic power depend on the input slew rate, as well as on the output capacitance load. These delay and power figures are modeled in non-linear tables, where each value depends on a combination of an input slew rate and an output load capacitance. However, to do so, input slew rate and output load capacitance vectors are required, to generate the characterization simulation environment. These vectors can then be independently used to generate the models for each standard cell. Moreover, since characterization relies on non-linear table models, the quality of the results is a function that strongly depends on the precision of the provided input slew rate and output load capacitance vectors.

Static States/Transition Arcs Search

Once the simulation environment setup is done, the tool can deal with specific standard cells. These are available as Spice netlists containing the transistors schematic along with all parasitics. Also, the standard cell logic function and output and input pins names must be defined, to guide the tool in its search for transition arcs and static states, which will be used to conduct the simulations required for the standard cell characterization. In LiChEn, logic functions can employ the following logic operations: conjunction (*), disjunction (+) and complement (~). Moreover, parenthesis can be used to make precedence rules explicit.

For instance, a basic C-element with inputs *A* and *B* and output *Q* has the following logic function:

$$Q = (A * B) + (Q * A) + (Q * B) \tag{4.5}$$

In this function, it is clear the expression of a feedback loop, since the value of *Q* participates in the *Q* value generation. As explained before, characterizing a standard cell that executes such a logic function with conventional tools requires laborious manual work. Also, as a standard cell library typically contains hundreds of cells, it is advantageous to have available a highly automated characterization process for this job. LiChEn employs a search tree algorithm to find all transition arcs and static states (SSs). This algorithm works for functions that count with feedback loops. Basically, the tool initially sets all input and output pins to logical 0 and computes the resulting value of the output according to the logic function, obtaining the first static state. Next, it switches the logical value of one input at a time and evaluates the next output value through a recursive function. Each new input/output value after a transition is accounted for as a new SS. To avoid exponential computation time, as soon as LiChEn detects that a previously computed SS was already computed previously, it stops the recursion, pruning the search at this point. During switching of the input logical values, LiChEn evaluates in which cases the output value changes when an input switches. These cases are called Dynamic Transition Arcs (DTAs). The cases where input switching does not cause an output switching are called Internal Transition Arcs (ITAs).

Figure 4.11 presents the computation of transition arcs and SSs for the logic function of the 2-input C-element described by Equation 4.5. In the Figure, graph edges are numerically labeled according to the order in which they are computed. Initially, inputs *A* and *B* and the output *Q* are set to logical 0 (*E0*). The resulting value is *Q = 0*. This characterizes an SS (*A=0, B=0, Q=0*). Next, through a recursive function, input A switches to logical 1 (*E1*). The result is still *Q = 0*. However, a new SS (*A=1, B=0, Q=0*) and an ITA (*A=R, B=0, Q=0*) are found. Then, the value of *A* is switched once more (*E2*), resulting in the static state (*A=0, B=0, Q=0*). Because this state was already computed previously, this branch of the search graph is pruned. The function go back one node (to *A=1, B=0, Q=0*), and switches the other input (*B*) logical value (*E3*). This time, the input switching event causes the output

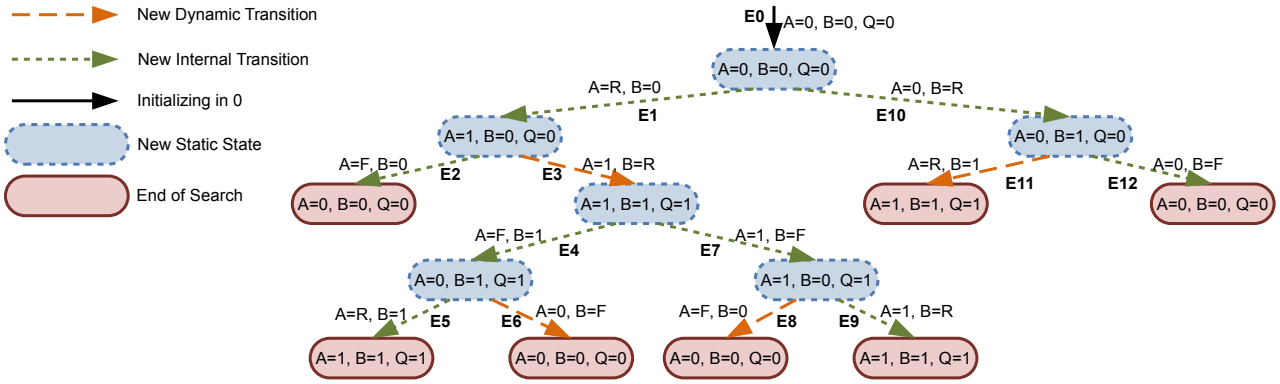


Figure 4.11 – Computation of transition arcs and static states of a two input C-element. “R” stands for low-to-high transitions (rise) and “F” for high-to-low transitions (fall). Graph edges are labeled numerically as En, according to the order in which they occur during the transition arcs search [MOCO13].

Table 4.1 – Resulting SSs, DTAs and ITAs after the complete search for the logic function of a 2-input C-element. Logical values assume the order A, B, Q for inputs/outputs [MOCO13].

SSs	*DTAs*	*ITAs*
0,0,0	1,R,R	R,0,0
1,0,0	0,F,F	F,0,0
1,1,1	F,0,F	F,1,1
0,1,1	R,1,R	R,1,1
1,0,1		1,F,1
0,1,0		1,R,1
		0,R,0
		0,F,0

to switch, generating a new SS (*A=1, B=1, Q=1*) and a new DTA (*A=1, B=R, Q=R*). From this node, the next step is to switch the logical value of input *A* (*E4*), generating a new branch in the search graph as *E5* and *E6* take place. Once this branch evolves and is pruned, input *B* has its logical value switched again (*E7*). This search continues until all branches are covered (*E12*) and pruned. Table 4.1 shows the complete set of SSs, DTAs and ITAs, in the order they were computed, for the logic function of the 2-input C-element.

This simple algorithm is capable of detecting all SSs, DTAs and ITAs, even those that depend on feedback loops. In this way, it is possible to efficiently characterize the electrical behavior of asynchronous standard cells. Yet, currently the tool supports only single output standard cell, and cannot characterize typical synchronous constraints for sequential cells, such as setup and hold times. Other than asynchronous cells, LiChEn can also be used to characterize typical combinational logic cells, such as ANDs and ORs. However, it is not recommended its use to characterize synchronous sequential standard cells like flip-flops or latches, as the tool does not support timing constraints characterization.

Timing/Power Figures Generation

LiChEn may start the characterization process once all SSs, DTAs and ITAs have been computed. Based on these results, the tool generates Spice files that implement each transition arc and each SS. Measurements are conducted during Spice simulation, based on the configuration given in the first stage of the characterization flow. During characterization, LiChEn measures input gate capacitance for low-to-high and high-to-low transitions, static and dynamic power and timing figures.

Timing figures are measured as pin-to-pin propagation delays and output transition delays. The static power is measured for each SS. Dynamic power, in turn is divided in two parts: switching and internal power, measured as the power observed for DTAs and ITAs, respectively.

The input gate capacitance is computed from the average current for low-to-high and high-to-low input transitions and exported as rising and falling input capacitances, respectively. Static power, on the other hand, is measured for each SS according to the average current drawn from the power source. In addition, the static power of each SS is measured for each power source independently.

LiChEn models dynamic power using non-linear tables to achieve comprehensive coverage of the electrical behavior of standard cell combinations. In this way, a vector of input slew rates and output capacitance loads is required by the simulation environment. The tool measures switching and internal power for each combination of input slew and output load, generating a two-dimensional non-linear table. The switching power is measured as the power for DTAs, and the internal power as the power in ITAs, without considering static power. The internal power P_I is computed according to the following equations:

$$P_I = V_{(vdd)} * \int_0^T I_{(t)} dt - Static_{tot} \tag{4.6}$$

$$Static_{tot} = V_{(vdd)} * (I_{before} * t_{init} + I_{after} * (t - t_{init})) \tag{4.7}$$

Here, $Static_{tot}$ stands for the total measured static power. Computation starts by calculating the total charge drawn from the power source while in the SS that occurred before the transition arc ($I_{before} * t_{init}$). This charge is added to the charge drawn from the power source for the SS that results from the transition arc in the remaining simulation time. This sum is then multiplied by the supply voltage, generating $Static_{tot}$. Simulation time definitions are furnished prior to characterization. In this way, internal power considers only the charge drawn from the power source, which is caused by the switching of an input. Switching power is calculated in a similar way. However, the current required to charge the output capacitance

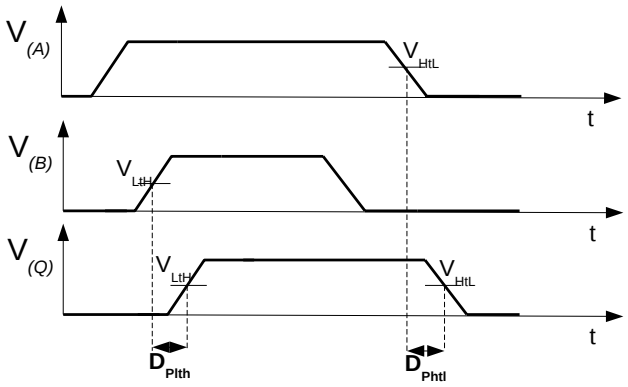


Figure 4.12 – Example of propagation delays for a 2-input C-element [MOCO13].

is also subtracted. In this way, the switching power is measured as P_s, according to:

$$P_s = V_{(vdd)} * \int_0^T I_{(t)}dt - Static_{tot} - (\frac{1}{2} * C_{out} * V^2_{(vdd)}) \tag{4.8}$$

As for the cell timing characteristics, LiChEn models propagation and transition delays. Propagation delay is measured for DTAs as the total time it takes for the switching at an input to cause switching at some output. This depends on high-to-low and low-to-high transition thresholds, defined by the designer. Figure 4.12 shows an example of such measurement, the propagation delays D_{Plth} and D_{Phtl} for a 2-input C-element.

These are based on DTAs (*A=1, B=R, Q=R*) and (*A=F, B=0, Q=F*), respectively. First, both inputs and the output are at logical 0. Next, input *A* switches to logical 1, without modifying the value of the output. However as soon as input *B* reaches the low-to-high transition threshold starts the measurement of the time it takes for *Q* to reach the high-to-low transition threshold. This is modeled as a low-to-high (rising) propagation delay from pin *B* to *Q*. Similarly, a high-to-low transition in input *B* followed by a high-to-low transition in input *A* is modeled as the high-to-low (falling) propagation delay from pin *A* to *Q*. In this manner, the tool models all possible propagation delays of a given standard cell from all inputs to the output(s). Transition delays are modeled as the time it takes for an output to switch from a logical value to another. These values are based on minimum logical 1 and maximum logical 0 voltages. Figure 4.13 shows an example of the transition delays for an output pin *Q*. The low-to-high (rising) transition delay (D_{Tlth}) is measured as the time an output takes to achieve the minimum logical 1 voltage, after it reached the maximum logical 0 voltage. Similarly, the high-to-low (falling) transition delay (D_{Thtl}) is defined in an opposite way.

Currently, the tool does not characterize setup time and minimum pulse width. This is due to the fact that these are unnecessary values in most asynchronous design templates, because circuit monotonicity is guaranteed. Also, all simulations are currently performed through Cadence Spectre Spice simulator. However, LiChEn is easily adaptable to support

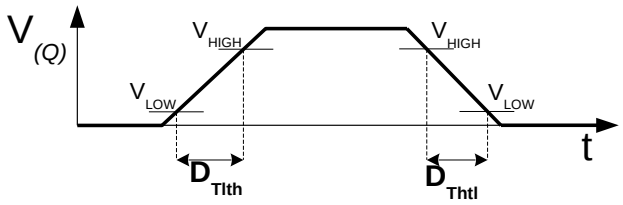


Figure 4.13 – Transition delays example for an output pin Q [MOCO13].

other simulators. After modeling all the electrical figures of the library standard cells, LiChEn exports the generated results to a text based file in the Liberty Format, which is compatible with most EDA tools.

4.3 The ASCEnD-ST65 v2 Library

Using the ASCEnD-A flow, and its earlier versions, some libraries were designed. For example, [Mor10] reports the design of a library of 251 cells, including different C-element variations, called ASCEnD-ST65 that targets the STMicroelectronics 65nm bulk process. The work also mentions a small library designed targeting the XFAB 180nm technology. Through the development of the design flow, the ASCEnD-ST65 library got bigger, as we explored new components and templates, until reaching over 900 components. After that, a more compact version of the library designed in the 65nm process was implemented and called ASCEnD-ST65-Lite. This library does not have any related publications, but it is already being employed by a research group in the Technical University of Denmark. It contains only basic C-elements, 2 and 3 inputs and 2 inputs with set and reset.

Once the ASCEnD-A flow was complete, and the project of cells had all its steps automated, we redesigned the library targeting the STMicroelectronics 65nm process, including only the cells that were useful for our target design templates. We called this new library ASCEnD-ST65 v2. It is not as massive as the original library, but that is because we filtered out many cells. This Section explores the design of its library and brings to light its composition.

4.3.1 Library Architecture

The ASCEnD-v2 library assumes an architecture similar to that employed in the original ASCEnD-ST65 library, in order to maintain compatibility. As Figure 4.14 shows, 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 cell. 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 cell library and the core library provided by the foundry. Furthermore, being tapless enables better usage of the layout area. Tap cells are available in the core library for system level layout design.

Note that whenever a transistor is too big to fit in the specified architecture, it is folded in more transistors. In other words it is divided in a sequence of parallel transistors which sum of widths is the width of the original transistor. The process of folding a large transistor is transparent to the designer and done automatically by ASTRAN.

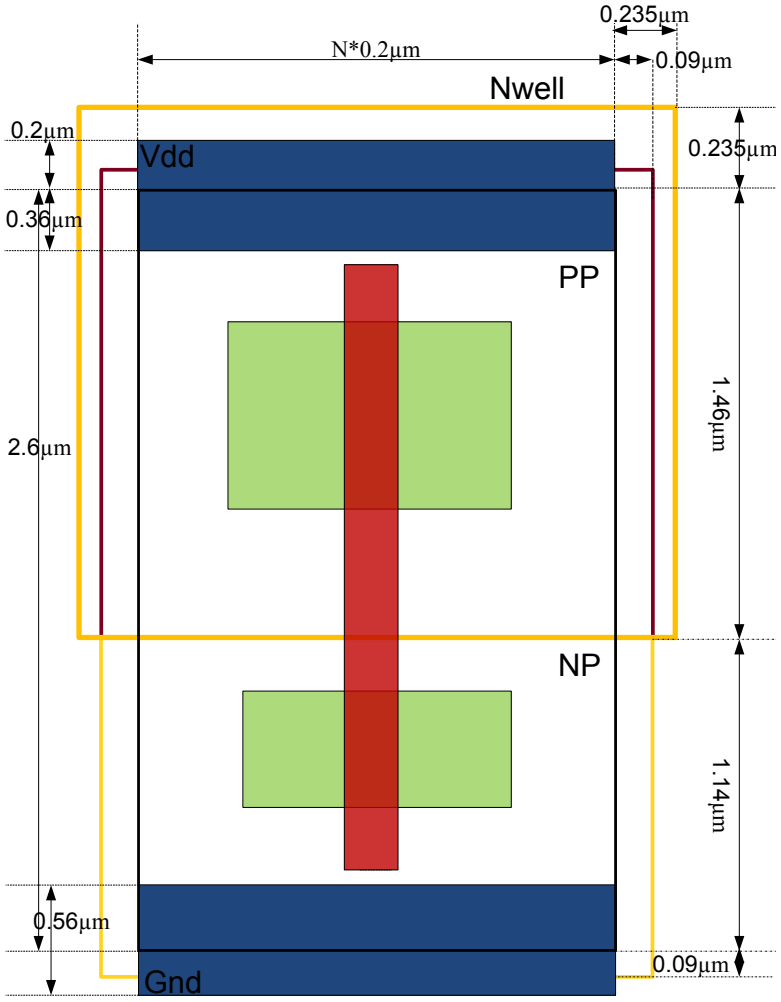


Figure 4.14 – Standard layout architecture. The red polygon is polysilicon and the green rectangles are diffusion. NP and Nwell are the negative doping layers and PP is the positive doping layer. Adapted from [Mor10].

4.3.2 Library Composition

For the design of the library we selected a large set of NCL and NCL+ gates, which includes some C-elements, and 4 MUTEX gates. The gates have driving strengths that vary from X2 to X31, providing a good range for design space exploration during synthesis tasks. Initially, we intended to design the library such that all positive unate gates had the following driving strengths: X2, X4, X7, X9, X13 and X18; and the negative unate gates had: X2, X4, X7, X9, X13, X18, X22 and X31. These ranges were defined according to the available driving strengths in the cells of the core library, available with the target PDK, where a similar criteria was observed. In fact, our driving strength is defined such that an X2 gate of the ASCEnD-ST65 v2 library has a similar driving capability of a gate of the same driving strength in the core library.

Unfortunately, during the process of laying out the cells, some driving strengths of specific cells were not generated by the layout generation tool. In this way, we ommited those from our cell library set. Having that said, all the cell functions we specified have at least one version implemented in the library. These functions were the same that will be defined in Section 5.5.4 for NCL and NCL+ gates and the functionality defined in Section 3.1.1 for MUTEX gates. We also added settable and resettable options for those C-elements, special cases within NCL gates. In this way, we implemented a total of 121 functions in a total of 614 cells. All these cells ara available at the layout level and have all models available for logic and physical synthesis and analog and digital simulation. We detail each functionality available at the library in Appendix B.

4.3.3 Library Distribution and Organization

The cell library is freely available, provided that the requester have access to the target PDK. Note that we request a copy of the NDAs required by STMicroelectronics for accessing the 65nm bulk CMOS PDK and core library. ASCEnD-ST65 v2 is stored in a subversion repository at the GAPH research at PUCRS, available at: *https://corfu.pucrs.br/svn/ascend/st65nm-cmos065/tags/ASCEnD-ST65-v2* The library is organized in folders as follows:

- doc
  This folder contains the documentation of the library.
- extracted
  This folder contains the extracted views of each cell of the library. It counts with RC extraction and is available in Spice language.

- layout
  This folder contains all the layouts of the cells of the library. It is based on CDB and compatible with Cadence IC 5.1. Abstract views, symbols and schematics are also available in this folder in CDB.

- lef
  This folder contains the abstract views for place and route, exported to the LEF format.

- lib
  This folder contains the power and timing models for STA and synthesis tasks. The models are all available in the Liberty format.

- schematic
  This folder contains the transistor level schematics of all cells described in SPICE.

- verilog
  This folder contains behavioral views for back-annotated delay simulation. All views are described using UDPs in Verilog. These views also assume a unitary delay of 100 ps for all arcs, which can be overwritten in timing simulation with more precise values extracted from the Liberty models.

4.4 Discussion

The design flow detailed in this Section is the evolution of a flow that started a few years ago, as described in [Mor10]. As original contributions in this Thesis, we added more automation capabilities to the flow. These include a new tool for electrical characterization, LiChEn, and the integration of ASTRAN for automatic layout generation. Also, we added a set of scripts that automate the connection of these tools, facilitating the design of each cell. In this way, instead of a designer being able to devise in average a cell per day, as it is our own experience with the earlier versions of the flow, we are able to generate the complete views of a single cell in less than an hour.

Unfortunately there are still some limitations in the design flow, which are work in progress. One of these limitations is the restriction to single output cells. The tools employed for transistors sizing currently assume the implementation of an oscillator ring, which makes it very complex to dimension a multi-output cell. In fact, we usually employed a less automated approach for such cells, which require manual description of Spice decks and collection and analysis of results.

The characterization tool, LiChen, also has a limitation for multi-output cells. We started to optimize it to support such cells, but it is still ongoing work and we could not finish and verify a new version of this tool. That said, it is our experience that the algorithms

employed for the current version, which support only single output cells, are compatible with multi-output cells as well. Another feature that is still not automated is the support for charge sharing effects verification. We currently do it manually using a set of scripts that must be specifically tuned for each cell. We intend to incorporate an automated version of this analysis in LiChEn, so that it enables designers to assess charge sharing effects in early stages of the design process. For example, the tool could be used in the schematic view of the cell, before proceeding to layout.

ASTRAN is also currently limited to single row cells, which compromises its usability for complex asynchronous components like PCHB gates. Having that said, it was sufficient for the implementation of all NCL, NCL+ and MUTEX gates we designed in ASCEnD-ST65 v2. Another limitation of the tool is that it has a set of design rules that support only technologies as recent as 45 nm. In fact, ASTRAN was recently successfully used for designing NCL cells in the Free PDK 45nm predictive technology [Fre]. However, that limitation prevented us to designing a cell library in more recent technologies like the STMicroelectronics 28 nm FDSOI PDK, as the layout generation step is too timing consuming.

Besides these limitations, we employed the flow to build a large cell library that supports asynchronous circuits design. It contains a large set of NCL and NCL+ cells, as well as some MUTEXes. Moreover, within this set are C-elements, a special case of NCL cells. In this way, this library enables building circuits based on different QDI and BD templates, as Chapter 5 explores in more detail. Using this cell library, several case study circuits were designed to the layout level and validated with extensive post-layout timing digital and analog simulation. Among these, we highlight the following NoC designs: BaT-Hermes [GMMC15], BaBaNoC [MMG+13] and Hermes-AA [PMMC10]. Furthermore, we recently collaborated in the design of a new template, called Blade [HMH+15], where all the cells used in it were designed, at least partially, using the ASCEnD-A flow.

Through the development of the design flow and associated libraries, we could better understand the challenges of designing a cell library. After that, though, we saw the benefits of having such a library available for exploring architectural optimization in asynchronous circuits, as Chapter 5 explores. In fact, we believe that the lack of easy access and generic cell libraries for asynchronous circuits design, is one of the major barriers that impair wider usage of such circuits. In this way, the ASCEnD-A flow is a step forward better acceptance and wider usage of asynchronous design techniques, as it enables rapid and automated design of cell libraries. Moreover, the ASCEnD-ST65 v2 library provides an easy access option for designers that want to experiment with this technology, as it is freely available. The only limitation is that its design targets a proprietary PDK and the designer must have access to it to use the library.

5. INNOVATIONS IN ASYNCHRONOUS DESIGN TEMPLATES

An asynchronous template, as defined in Section 2.2.4 structures how an asynchronous circuit is designed and operates. In this way, templates allow exploring architectural optimizations of these circuits. With this in mind, the optimization of asynchronous templates is a major step in this work, once a design flow for rapidly devising any component required by such templates is available. Accordingly, the next set of contributions of this Thesis is described in this Chapter as a set of innovations in asynchronous circuits design templates.

First we explore templates available in the state-of-the-art and discuss the design of case study circuits to validate the usage of components and cell libraries, in Section 5.1. Then, Section 5.2 explores the usage of return-to-one QDI channels and the optimizations that they allow in a classic case study template. Sections 5.3, 5.4 and 5.5 propose new templates and discuss their trade offs. Finally, we discuss the impact of the proposed innovations in asynchronous components design in Section 5.6.

5.1 State-of-the-Art in QDI Templates for Asynchronous Design

This Section addresses the state-of-the-art in QDI templates for asynchronous design. Note that the focus here are the templates explored in this Thesis only. For a survey of other templates, QDI or others, please address works like [NS11, NS15a, NS15b].

5.1.1 The Dual-rail RTZ WCHB/DIMS Template

The dual-rail RTZ WCHB/DIMS template is one of the most adopted for implementing QDI combinational logic [MN06]. As its name suggests, this template relies on a dual-rail (or a 1-of-2 DI code) for constructing its channels (see Figure 2.14). For its design style, it requires only one specific asynchronous component type other than conventional gates like ANDs and ORs, the C-element. This characteristic facilitates the adoption of conventional EDA tools and the use of semi-custom cell-based design flows, and is one of the reasons why this template is so commonly employed in QDI design. As its architecture, it relies on weak-conditioned half-buffers (WCHBs) for building registers and control circuitry. For combinational blocks, it employ DIMS logic.

Sequential QDI circuits are often designed using WCHBs. Among other possibilities, discussed in detail for example in [YR06], WCHBs are advantageous because they enable a reduced design complexity. This is due to the enforcement of a design methodol-

ogy, similar to the one employed in conventional sequential circuits. Furthermore, the WCHB can be employed for different QDI templates, which can rely in different DI channel flavors. For example, Figure 5.1 shows the schematic of a 1-of-2 4-phase WCHB.

The *reset* signal ensures that after its activation the output will have a spacer. In this condition, the *INack* signal will have a 1 issued by the validity detector, a NOR gate, signaling to the previous WCHB that new data can be transmitted. Valid data will only be written on the output when it is available on the inputs (*INt* and *INf*) and the next WCHB can receive new data, i.e. *OUTack*=1. As soon as new data propagates to the output, the *INack* signal switches to 0, signaling to the previous WCHB that a spacer may be sent. Accordingly, spacers will only be written to the output of a WCHB when a spacer is available on the inputs and the next WCHB can receive a spacer (*OUTack*=0).

To compose WCHBs with the capacity to store more than one bit of data, the *OUTack* control signal may be shared by all 1-bit WCHBs. To do so, the *INack* signals of a subsequent stage can be merged using C-elements, forming a multibit validity detector, as discussed in [MN06]. Note that two successively connected WCHBs in a stable condition always store a valid data and a spacer at any moment, respectively, or vice-versa. This is why this latch is called a half-buffer. The consequence is that a QDI pipeline with n stages based on WCHBs can contain at most n/2 valid data items at any time, as explored in [NS11].

In DIMS blocks, used here for combinational logic, all minterms of a logic function are first computed using C-elements. Next, the needed sum of the calculated minterms is computed using an OR gate similar to traditional two-level logic. This guarantees that the circuit will operate correctly and will respect the principles of delay insensitivity. In fact, DIMS can be used for any circuit that employs 1-of-n DI codes and 4-phase handshaking for its channels. As an example, Figure 5.2 (a) shows the DIMS implementation of a half adder circuit using a 1-of-2 (dual-rail) 4-phase DI channel.

In this example, the dual-rail sum output is 1 when exactly one of the inputs is 1, otherwise it is 0, and the carry output is 1 only when both inputs are 1, otherwise it is 0. This is computed by first producing all combinations of *A* and *B* lines (true and false wires *At*, *Bt* and *Af*, *Bf*), producing the minterms. A 1 in the sum output ($S_{out}t$=1, $S_{out}f$=0) corresponds to exactly one input at 1 (*At*=1, *Bf*=1 or *Af*=1, *Bt*=1). In a similar manner, a 0 in the sum output ($S_{out}t$=0, $S_{out}f$=1) is generated when the inputs are the same, i.e. *At*=1, *Bt*=1 or *Af*=1, *Bf*=1.

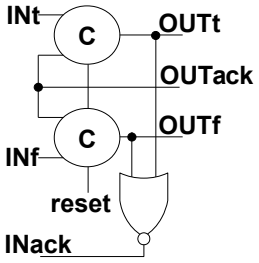


Figure 5.1 – Schematic of a 1-of-2 4-phase QDI 1-bit WCHB.

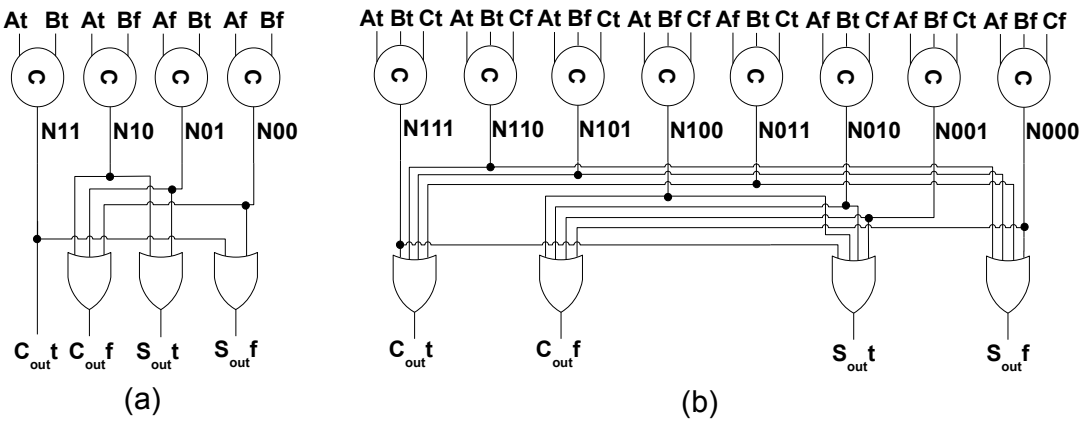


Figure 5.2 – Example 1-of-2 DIMS-based blocks: (a) a one-bit half adder; and (b) a one-bit full adder.

The remaining outputs are computed in a similar way. Note that the usage of a C-element for producing the minterms guarantees that a spacer will only be signaled in the outputs ($S_{out}t$=0 and $S_{out}f$=0 / $C_{out}t$=0, $C_{out}f$=0) when both inputs have a spacer (At=0, Af=0, Bt=0, Bf=0). This is due to the functionality of this block, as Figure 5.2 (b) shows, coupled to the functionality of OR gates, which will only signal a 0 when all their inputs are at 0.

Currently, many works on QDI design automation that target the dual-rail RTZ WCHB/DIMS template are available. An example of a framework for synthesizing QDI circuits that supports this template is Balsa [Bar98, Bar00], proposed in the University of Manchester. It uses an extension of the Philips Tangram language [vBKR+91]. Balsa includes a language for describing asynchronous hardware, also called Balsa. The language allows describing several classes of circuits in high levels of abstraction. The synthesis tool included in the Framework adopts a syntax-directed translation approach, which maps language constructions to handshake components. Syntax-directed translation is quite interesting, since modifications in the description lead to predictable modifications in the generated hardware, contrary to conventional hardware description languages, such as VHDL. However, it limits the quality of the generated circuits, because design space exploration is limited.

Another alternative to synthesize circuits targeting this template is the flow proposed by Thonnart et al. in [TBV12], which employs conventional tools for synthesis and power/timing analysis. The flow starts with a structural VHDL description of a circuit using WCHB for sequential logic and DIMS for combinational logic. The flow treats WCHBs as flip-flops and a virtual clock signal exists for guiding the synthesis tool. One of the drawbacks is the manual labor required by the flow, as it mandates the modification of timing and power model files for tricking the synchronous design tools. Also, several iterations are needed during the synthesis to achieve the desired results. Besides, in some cases the generated circuits violate the constraints specified by the designer, which is a consequence of using a framework deemed for synchronous design while implementing asynchronous cir-

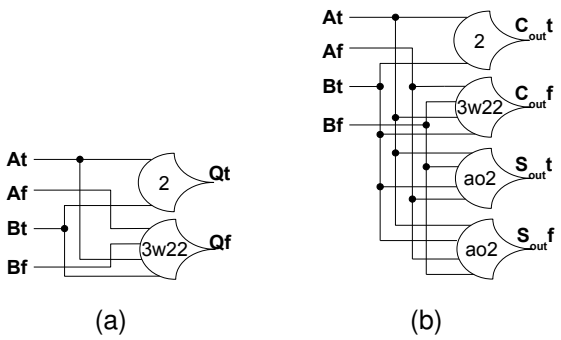


Figure 5.3 – Examples of 1-of-2 4-phase QDI circuits built with NCL gates: (a) 2-input AND gate; (b) half adder combinational block.

cuits. Another consequence of using a synchronous approach is the fact that the used tools always try to optimize the worst-case delay, which again considerably limits design space exploration.

5.1.2 The NCL Template

Another QDI template, called NCL, was proposed by Fant and Brandt [FB96]. In 1990, Fant and Brandt founded Theseus Logic, an enterprise that successfully prototyped many test chips based on NCL logic and had a synthesis flow for automating part of this process. Since then, NCL has been explored by different researchers, as reported in several works e.g. [LFS+00, KL02, BS07, BDSM08, JN08, RST12]. In its original proposition, the NCL template targets 1-of-2 RTZ DI channels. For building its logic blocks it relies on special cells called NCL gates, explored in Section 3.1.3. Using these cells, combinational and sequential blocks can be built. The architecture of this template is similar to the one presented in Section 5.1.1. In fact, registers are built using WCHBs. The difference is that logic blocks are built using NCL gates and are not restricted to C-elements and conventional gates.

For instance, a 1-of-2 4-phase 2-input AND can be built using 2 NCL gates as Figure 5.3(a) shows. Note that the true output *Qt* is computed using a 2-of-2 NCL gate with inputs *At* and *Bt*, i.e. the output will only be 1 when both inputs are 1. The false output, in turn, employs a 3-of-4 gate with input weights (2, 2, 1, 1). Note that the false inputs *Af* and *Bf* are the ones connected to the top two inputs of the NCL gates (those with weight=2, according to Figure 3.5), while the true inputs *At* and *Bt* have weight 1 in the gate. This means that the gate will only switch to 1 when both inputs have valid values and at least one of them is 0 (*Af*=1, *Bf*=1 or *Af*=1, *Bt*=1 or *At*=1, *Bf*=1). In addition, spacers will only be generated in the outputs when both inputs have spacers. This is guaranteed by the fact that the output of NCL gates will only switch to 0 when all inputs are at 0.

A half adder can be implemented in a similar manner. As Figure 5.3(b) shows, the AND block of Figure 5.3(a) generates the carry signal ($C_{out}t$ and $C_{out}f$). The sum employs two special NCL gates that use an AND-OR function. Consider generating the true sum output, where $S_{out}t$ switches to 1. This occurs either when At=1 and Bf=1 or when Af=1 and Bt=1. The false sum output ($S_{out}f$), on the other hand, switches to 1 when either At=1 and Bt=1 or when Af=1 and Bf=1. Note that, in comparison to the DIMS approach, showed in Figure 5.2, complexity to design a combinational block using NCL is considerably reduced and requires a smaller number of logic gates [MOPC13a].

Several authors propose automated flows and tools for designing and optimizing QDI circuits based on NCL, e.g. [JN08, KL02, LFS+00, RST12, PAAS14]. In fact, even Balsa can be configured to target this template during synthesis. The drawback is that most of these flows and tools perform logic optimizations before technology mapping only. However, the latter step is precisely the first one in the synthesis of a circuit that allows optimizations with realistic cost parameters of the target technology. Besides, these flows all rely on a template-based approach for technology mapping. This means that, they synthesize the circuit as if it was a single-rail version, then replace single rail logic gates by equivalent NCL templates (combination of NCL gates). This procedure often prevents the exploration of optimizations enabled by logic sharing and specific optimizations allowed by NCL logic.

For example, consider the synthesis of an 8-bit NCL Kogge-Stone adder [KS73]. Its single-rail implementation can be devised as showed in Figure 5.4. Note that the general architecture of the adder is showed in Figure 5.4(a) and its components in Figures 5.4(b), 5.4(c) and 5.4(d). In a template-based approach, single rail gates that compose the *YBs* and *RBs* are replaced by the equivalent set of NCL gates that implement the same function in a given template, a dual-rail RTZ NCL in this example. Albeit this is simple, the resultant circuit is unoptimized. Moreover, conventional synthesis tools cannot optimize it, because they fail to recognize the logic of NCL gates.

The only work the author could find that somewhat allows post-mapping optimizations is the work of Cheljoo and Nowick, described in [JN08]. However, this work provides only basic optimizations again based on predefined logic templates. It allows performing logic optimizations replacing predefined sets of templates when these appear in the netlist.

5.1.3 The PCHB Template

The templates from Sections 5.1.1 and 5.1.2 are all based on static CMOS design. They enable straightforward implementations, but usually lead to large overheads in latency, given the need to compute data validity for each pair of registers at each stage. Another alternative is the use of the PCHB template, based on dynamic CMOS logic. In fact, several works demonstrate the advantages of this model to obtain high speed and low power circuits,

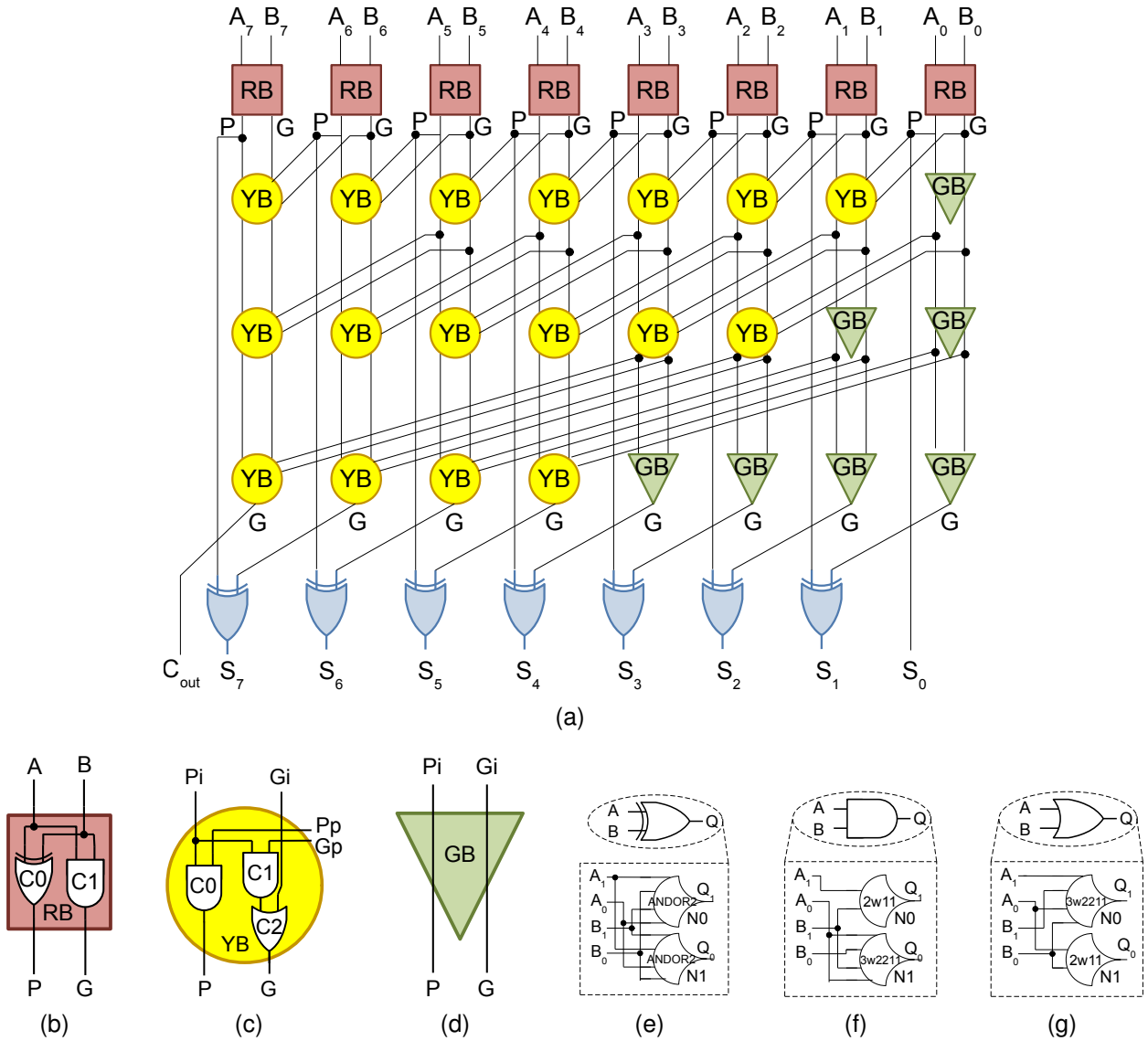


Figure 5.4 – Details of the Kogge-Stone adder design: (a) block diagram (b, c, d) red (RB), yellow (YB) and green (GB) boxes internal structure; (e, f, g) NCL implementation of 2-input XOR, AND and OR gates. Adapted from [MTMC14].

as discussed in [BOF10]. Also, PCHB-based circuits have been extensively explored by Fulcrum Microsystems (recently acquired by Intel), which developed commercial products that employ QDI PCHB-based logic [DLD+14].

The PCHB template relies on the usage of 1-of-2 RTZ DI channels. For its desgn style, it employs a specific set of domino logic cells. These cells implement registered functions, avoiding the need of explicit registers in the definition of the architecture. In fact, as Figure 5.5 shows, this template has the definition of each stage of its architecture as a single block. The block is responsible for implementing a specific functionality, *e.g.* an adder, and registering the computed value. Communication with neighbour stages is done through a DI channel and controlled using a set of two completion detectors. These completion detectors can be implemented in a similar manner to those used in the templates explored in Section 5.1.1.

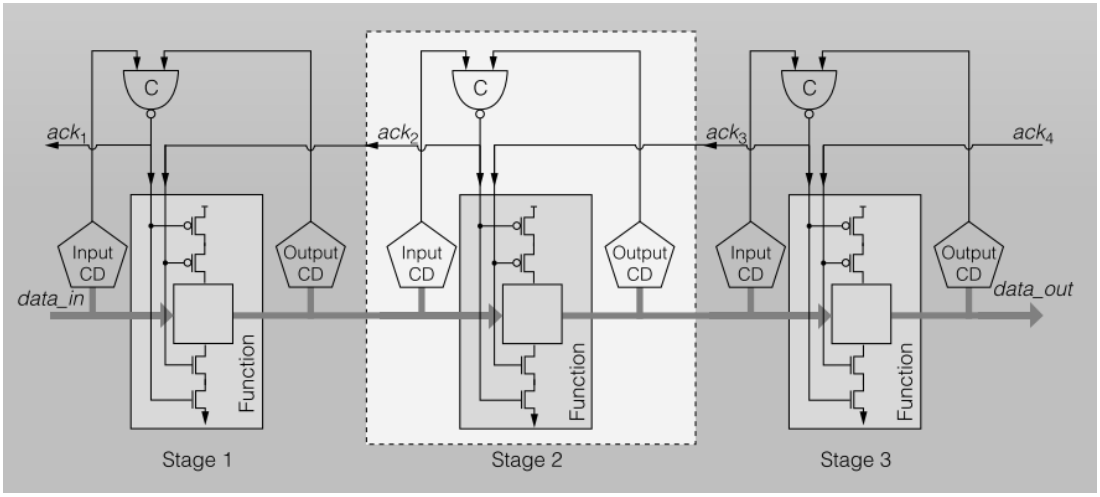


Figure 5.5 – Architecture of the PCHB template, adapted from [NS11]. CDs are completion detection blocks.

An advantage of this template is the availability of Proteus, a design flow proposed by Beerel et al. in [BDL11] that automates the synthesis of circuits targeting PCHB. The flow relies on the usage of a proprietary cell library [BDL11], and uses conventional synthesis tools. The flow input can be either RTL legacy code or code in a Fulcrum proprietary language based on communicating sequential processes (CSP). From the specification, the circuit is synthesized focusing only on the true wire of the 1-of-2 design, similar to a conventional single-rail synthesis, and is then mapped into components of an image library. At this stage, the synthesis tools are capable of performing logic optimizations, which allows the flow to take advantage of mature algorithms available to these tools.

Next, the netlist is converted to a circuit that manipulates 1-of-2 encoded information and the image cells are replaced by physical PCHB cells for the physical design. Note that in this process there is a series of optimizations that are performed in the asynchronous logic, as discussed in [BDL11], using a set of proprietary tools. Such optimizations allow improvements that conventional tools would not be able to do, because the latter focus on worst-case delays, which is not adapted to asynchronous design. Next, place and route of the optimized design can occur, using conventional backend tools and the models of the PCHB library.

The PCHB template is no further detailed here because this work focuses on static CMOS logic only. More information on implementations of control blocks for generating the enable signals can be found in [BOF10, NS11].

5.1.4 Discussion

QDI templates have a rich set of options for defining channels and design styles, as explored in Section 2.2.4. This is mostly because their communication links can be built using any DI code, and combined with different handshake protocols. Furthermore, once a channel is defined, different sets of cells can be employed for constructing logic blocks that compose the architecture of a template. In this way, QDI templates provide a large design space to explore.

The PCHB template has been used to devise high speed QDI circuits [DLD+14]. This is mostly due to the dynamic CMOS logic employed in the design of PCHB cells. However, such high speed comes with high costs in power, which makes this template less interesting for low-power applications. Furthermore, the design of PCHB cells is complicated. These cells require precise sizing of their transistors because they rely on semi-static memory elements, using weak inverter-based staticizers. In fact, the design of such cells requires careful noise margin analysis, due to these sensitive memory elements. This sensitivity also compromises the robustness of the template to PVT variations and makes it a not well-suited candidate to voltage scaling applications. The layout and electrical characterization of PCHB cells are also challenging. For the former, these cells are usually composed by some dozens of transistors and require multi-row layout design, which is not compatible with the ASCEnD-A flow, because it is not supported by ASTRAN. For their electrical characterization, their dynamic behavior requires special care and their multi-output nature cannot be naturally handled by LiChEn, which limits their compatibility with the ASCEnD-A flow. Furthermore, access to synthesis tools for state of the art PCHB synthesis is limited, due to their current proprietary nature. These are the reasons why this template is not further explored in this Thesis.

Circuits based on DIMS logic are straightforward to design and require a small set of specialized cells. However, they are known to present inherent very high penalties in terms of performance, area and power [BOF10]. Hence, there is a clear trade-off between performance figures and design complexity. This does not mean that the dual-rail RTZ WCHB/DIMS template is not an interesting option for QDI design. In fact, it is commonly used for applications that do not have complex logic blocks, such as networks-on-chip (NoCs) [PMMC11, PMMC10, PCV12, TVC10], which mitigates the high costs related to DIMS logic. Furthermore, the C-elements required by DIMS blocks and WCHBs are easily obtained with the ASCEnD-A flow, which reduces the complexity of designing a cell library for this template.

The NCL template provides another alternative for QDI design and several works associate it to high speed [CCGC10, SCH+11], low power [ZSD10, GLY10] and robust implementations [JSL+10]. In fact, this template is gaining relevance in the asynchronous

research community and in the industry, as there are several recent works on NCL design and at least one new company, called Wave, that is targeting to commercialize low power designs based on it [Wav]. NCL also allows easy development of a semi-custom standard cell-based design flow and the cells required for its design are similar to C-elements. In fact, these cells are fully supported by the ASCEnD-A flow, easing the design of a cell-library for NCL circuits.

This Section was limited to three templates for the following reasons: (i) they are the most accepted templates by the asynchronous research community, being explored in books [BOF10, SF01a, Mye01] and survey articles [NS11, NS15a, NS15b]; (ii) they were employed for fabricating test-chips and used in real applications [NS15a]; and (iii) they ease timing closure, allowing timing robust implementations [JN08]. Furthermore, the templates assumed a 1-of-2 RTZ channel, albeit the design styles they adopt support other channel flavors, as explored in [BTEF03, PCV13]. This is due to the fact that according to Martin and Nyström [MN06], such channel choice enables reducing design complexity and is the most practical.

The dual-rail RTZ WCHB/DIMS and the NCL templates were the starting points for the work conducted in this Thesis in the synthesis of QDI circuits. Many works on QDI design automation are available in contemporary literature and were initially explored. These works either propose dedicated sets of synthesis tools or design flows using conventional synthesis tools for implementing QDI circuits. Such tools and flows rely on one or on a combination of different asynchronous templates for QDI design. However, due to the lack of interest in asynchronous circuits in the 80s and 90s, these tools and flows are still in their early stages of development and are usually not mature enough for adoption in the design of large-scale commercial ICs.

Balsa, for example, allows synthesizing circuits for different templates. Unfortunately, it is the Author's own experience that circuits generated by Balsa usually impose very big area and power overheads. Moreover, the tool does not allow power and timing analysis, which prevents proper dimensioning of the cells that compose the generated circuit. Other flows enable optimizing QDI circuits using conventional synthesis tools. However they are all limited to pre-technology mapping optimizations, which limits the quality of the resultant circuits. This is a fundamental problem in the synthesis of QDI circuits based on the dual-rail RTZ WCHB/DIMS and the NCL templates. The remaining of this Chapter explores the work conducted in the context of this Thesis to improve the quality of QDI synthesis. It starts by proposing a new manner for constructing QDI channels and design styles compatible with these new channels. Next, it explores new templates that are born combining these novel design styles and channels.

Table 5.1 – Static power consumption in C-elements when inputs are at the same logical level. Adapted from [MGC12b].

C-element	*Static Power (nW)*		*Savings*
	Both inputs at 0	*Both inputs at 1*	
Static	61.703	30.894	50%
Semi-static	66.858	39.107	42%
Symmetric	57.926	28.425	51%

5.2 Return-to-One QDI Channels

In this Section we propose a new manner of constructing QDI channels. This relies in a different way of generating *m-of-n* DI codes, called return-to-one (RTO, in contrast with RTZ), as first discussed in [MGC12b]. The approach consists of representing spacers with all wires at 1, rather than all wires at 0, and data validity is signaled by setting all *m* data wires of a channel to 0. In fact, the RTO protocol is analogous to RTZ. The only difference is that wire values are inverted compared to RTZ.

A first motivation behind the proposal of RTO was the fact that the C-elements designed in ASCEnD-ST65 v2 presented leakage power lower when they have all-1s in their inputs than when inputs have the all-0s value [MGC12b]. This observation indicated that employing an all-1s spacer could lead to more power efficient QDI designs, because while not computing, these circuits contain spacers only. Consequently, their components will have 1s in all outputs. To illustrate the fact quantitatively Table 5.1 shows the leakage power of three different 2-input C-element implementations (static, semi-static and symmetric) for two different states: both inputs tied to 0 and both inputs tied to 1. As the table shows, having both inputs at 1 leads to a reduction of static power to half of its value, compared to keeping both inputs at 0. Considering that C-elements are fundamental gates in the architecture of some QDI templates, RTO looked promising from this first analysis.

Table 5.2 shows conventions for a 1-of-2 example DI code based on RTO. N wires at 1 (all-1s) represent spacers. A valid 1 is denoted by *d.t* at 0 and a valid 0 by *d.f* at 0. As Figure 5.6 shows, differently from RTZ, RTO data transmission starts after the all-1s value is in the data channel. Accordingly, in the example, communication must start with all wires at 1, indicating absence of data. Next, the sender puts a valid 0 in the data channel by lowering *d.0*. The receiver computes the request from this value and acknowledges setting the *ack* signal low. When the sender reads the acknowledgment, it transmits a new spacer to finish the handshaking. The receiver computes the spacer and sets *ack* high. The sender can then start a new transmission, sending a valid 1 this time.

Recalling Figure 2.13(b), the process is very similar to transmissions in an equivalent RTZ channel. The difference is that all bits are inverted. In this way, RTO-RTZ domain interfaces for a same code require only inverters for all channel wires. As a generalization,

Table 5.2 – 4-phase 1-of-2 RTO encoding for one data bit.

Value	d.1	d.0
Invalid	0	0
Valid 1	0	1
Valid 0	1	0
Spacer	1	1

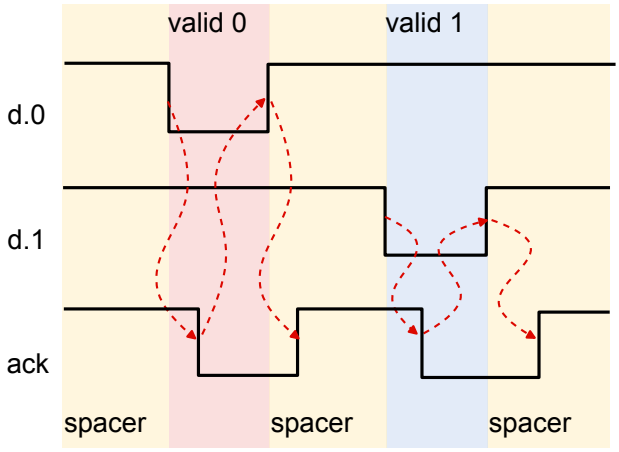


Figure 5.6 – Example of a waveform for the transmission of two 1-bit values in a 4-phase 1-of-2 RTZ DI channel.

an RTO D.x wire logical value is translatable from RTZ by:

$$\forall x, 0 \leq x \leq n-1 : RTO(D.x) = \neg RTZ(D.x) \tag{5.1}$$

Here, expressions *RTO(D.x)* and *RTZ(D.x)* correspond to wire logic values in the RTO and RTZ domains, respectively. In this way, according to Martin [Mar90], the conversion of data from one domain to another is DI. Also, albeit this Thesis focuses only on 1-of-2 DI codes, RTO design techniques can be easily adjusted to any 1-of-n code [MGC12b].

5.3 The Dual-rail RTO WCHB/DIMxS Template

A natural development after proposing RTO channels is to use it for constructing a new QDI template. Am intuitive template emerges adapting Dual-rail RTZ WCHB/DIMS to produce a new template. DIMS and WCHB logic blocks require modifications to be compatible with RTO channels. The proposition and development of such modifications gave birth to a new manner of designing QDI logic blocks based on RTO, called delay insensitive maxterm synthesis (DIMxS), which were used in a new QDI template called Dual-rail RTO WCHB/DIMxS.

5.3.1 Proposed Architecture

Before describing the Dual-rail RTO WCHB/DIMxS, it is useful to analyze the behavior of the blocks that compose the architecture of the Dual-rail RTZ WCHB/DIMS template. Assume that D represents the data inputs, Q represents the data outputs, *req* is the active-high request signal and *rst* is the active-low reset signal of a j-bit 1-of-2 RTZ WCHB (see Figure 5.7 (a)). Hence, a valid set of PRs describing such circuit is:

$$\begin{aligned} \forall i, 0 \leq i \leq j-1 : & \\ rst \wedge D_i.0 \wedge req \rightarrow Q_i.0 \uparrow & \\ rst \wedge D_i.1 \wedge req \rightarrow Q_i.1 \uparrow & \\ \neg rst \vee (\neg D_i.0 \wedge \neg req) \rightarrow Q_i.0 \downarrow & \\ \neg rst \vee (\neg D_i.1 \wedge \neg req) \rightarrow Q_i.1 \downarrow & \end{aligned} \tag{5.2}$$

Similarly, assume that the j-bit V signal is the vector of individual validity bits, and *valid* is the active-low global validity output signal. Hence, the validity of a j-bit 1-of-2 RTZ data channel, output of the example WCHB, is then given by:

$$\begin{aligned} \forall i, 0 \leq i \leq j-1 : & \\ D_i.0 \vee D_i.1 \rightarrow V_i \downarrow & \\ \neg D_i.0 \wedge \neg D_i.1 \rightarrow V_i \uparrow & \\ All0(V) \rightarrow valid \uparrow & \\ All1(V) \rightarrow valid \downarrow & \end{aligned} \tag{5.3}$$

where predicates *ALL1(x)* and *ALL0(x)* are used to determine whether all wires of x are respectively at 1 or 0.

These PRs implement the first and third phases of the protocol when driving the previous WCHB *req* input in a pipeline. Also, each data wire pair requires a 2-input NOR gate to compute its validity (the $V_i s$). Note that the generation of the intermediate $V_i s$ uses NOR gates rather than ORs. If all intermediate values are at 0, data D is valid. This can be computed with a j-input NOR gate. If all intermediate values are at 1, a spacer is detected. This function can be computed with a j-input NAND gate. Conditions where some of the intermediate values are at 1 while others are at 0 cannot determine a new value for valid. Hence, outputs of the j-input NAND and NOR gates need to be synchronized with a 2-input C-element.

The resulting validity detector (VD) circuit appears in Figure 5.7 (b). Another approach to synchronize all intermediate signals is to use a tree of C-elements. However, experience showed that these are usually more expensive in terms of silicon area, power and

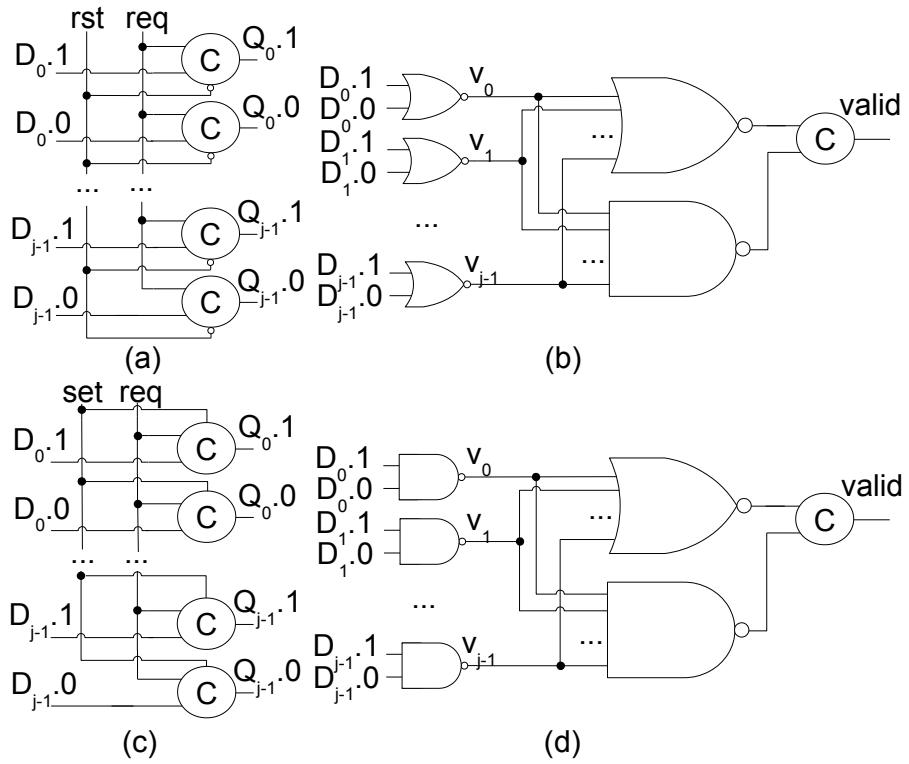


Figure 5.7 – WCHB latch and validity detector implementations: (a) RTZ buffer using resettable, active-low C-elements (lower input); (b) RTZ validity detector; (c) RTO latch using settable, active-high C-elements (upper input); and (d) RTO validity detector. Adapted from [MPC14].

delay than using *j*-input NORs and NANDs. The latter gates can be built as trees of cells with less inputs, due to the associative nature of the non-inverted equivalent functions. Also, partial VDs can be implemented with 2-input NOR gates inside each single value WCHB and synchronized at higher hierarchical levels. This can be useful, depending on the circuit data dependencies.

The RTO implementation of a WCHB is similar to the RTZ one. The main difference is the all-1s spacer. Because its initial state must be a spacer, there is a slight modification in the PRs for this WCHB, active-high set signal replaces the active-low rst signal of the RTZ WCHB PRs. The PRs for a j-bit 1-of-2 RTO WCHB are then given by:

$$\begin{aligned} \forall i, 0 \leq i \leq j-1 : \\ \neg set \wedge \neg D_i.0 \wedge \neg req \rightarrow Q_i.0 \downarrow \\ \neg set \wedge \neg D_i.1 \wedge \neg req \rightarrow Q_i.1 \downarrow \\ set \vee (D_i.0 \wedge req) \rightarrow Q_i.0 \uparrow \\ set \vee (D_i.1 \wedge req) \rightarrow Q_i.1 \uparrow \end{aligned} \tag{5.4}$$

As Figure 5.7 (c) shows, in this implementation, the resettable C-elements of Figure 5.7 (a) are replaced by settable ones, enabling RTO spacers for initialization.

Table 5.3 – Truth table for an RTZ 1-of-2 half-adder. Adapted from [MPC14].

$A_{in}.1$	$A_{in}.0$	$B_{in}.1$	$B_{in}.0$	$C_{out}.1$	$C_{out}.0$	$S_{out}.1$	$S_{out}.0$
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	0
1	0	0	1	0	1	1	0
1	0	1	0	1	0	0	1

The RTO VD block is also similar to the RTZ one. The difference is that it must detect 0s rather than 1s for data validity. Resulting PRs are:

$$\begin{aligned} \forall i, 0 \leq i \leq j-1 : \\ \neg D_i.0 \vee \neg D_i.1 \rightarrow V_i \downarrow \\ D_i.0 \wedge D_i.1 \rightarrow V_i \uparrow \\ All0(V) \rightarrow valid \uparrow \\ All1(V) \rightarrow valid \downarrow \end{aligned} \tag{5.5}$$

When mapped to logic gates, the result is that instead of employing a NOR gate for each pair of wires to detect whether one of them is at 0, a NAND gate is used to verify if data is valid (if one of the wires is at 0). The resulting circuit appears in Figure 5.7(d)). Similarly to what happens in RTZ, each intermediate signal is synchronized with j-input NAND and NOR gates and a 2-input C-element.

As for DIMS and DIMxS blocks, they require only C-elements other than conventional standard cells. For example, Table 5.3 shows the truth table of an RTZ 1-of-2 half-adder with inputs A_{in} and B_{in} and outputs C_{out} and S_{out}. The first line of the Table occurs when both inputs are spacers, which sets the outputs to spacers as well. Whenever one of the inputs is a spacer the outputs do not change. For that reason, such states are omitted. From Table 5.3 it is possible to specify the following PRs for $C_{out}.0$, $C_{out}.1$, $S_{out}.0$ and $S_{out}.1$:

$$\begin{aligned} (A_{in}.1 \wedge B_{in}.1) \rightarrow C_{out}.1 \uparrow \\ \neg A_{in}.1 \wedge \neg B_{in}.1 \rightarrow C_{out}.1 \downarrow \\ (A_{in}.0 \wedge B_{in}.0) \vee (A_{in}.0 \wedge B_{in}.1) \vee (A_{in}.1 \wedge B_{in}.0) \rightarrow C_{out}.0 \uparrow \\ \neg A_{in}.0 \wedge \neg B_{in}.0 \wedge \neg A_{in}.1 \wedge \neg B_{in}.1 \rightarrow C_{out}.0 \downarrow \\ (A_{in}.0 \wedge B_{in}.1) \vee (A_{in}.1 \wedge B_{in}.0) \rightarrow S_{out}.1 \uparrow \\ \neg A_{in}.0 \wedge \neg B_{in}.0 \wedge \neg A_{in}.1 \wedge \neg B_{in}.1 \rightarrow S_{out}.1 \downarrow \\ (A_{in}.1 \wedge B_{in}.1) \vee (A_{in}.0 \wedge B_{in}.0) \rightarrow S_{out}.0 \uparrow \\ \neg A_{in}.0 \wedge \neg B_{in}.0 \wedge \neg A_{in}.1 \wedge \neg B_{in}.1 \rightarrow S_{out}.0 \downarrow \end{aligned} \tag{5.6}$$

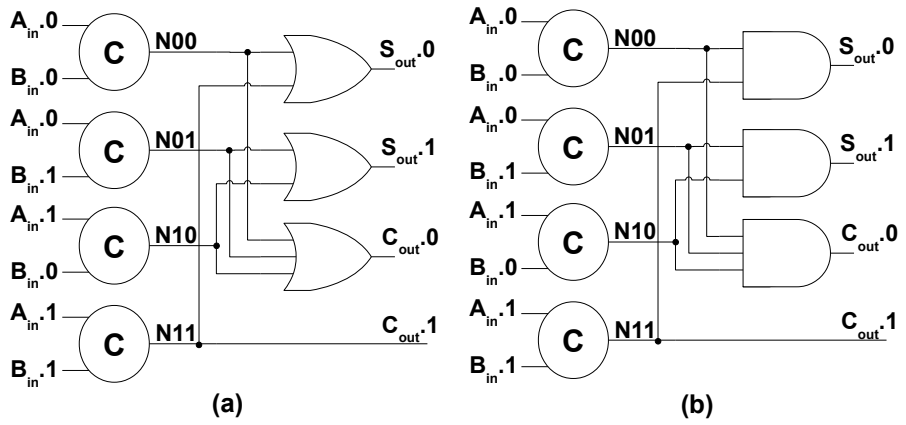


Figure 5.8 – DIMS/DIMxS implementation of (a) RTZ and (b) RTO half-adders. Adapted from [MPC14].

When mapping these to DIMS-based components, conjunctions of the pairs inside parenthesis are mapped to 2-input C-elements, to guarantee synchronization of both valid data and spacers. The disjunctions outside parenthesis in the PRs filter the C-elements that have their outputs set to 1. This models rising output transitions. The conjunctions for falling-transition PRs synchronize spacers in all C-elements. These functions are mapped directly to OR gates. The equivalent gate level schematic of these PRs appears in Figure 5.8.

An RTO 1-of-2 half-adder implementation can be obtained in a similar manner [MGC12a]. For example, Table 5.4 presents the truth table of an equivalent RTO version of this half-adder, which derives from Table 5.3 by Equation 5.1. The PRs of this adder are similar to those of the RTZ protocol, presented in Equation 5.6. Indeed, the disjunctions inside parenthesis to compute minterms are the same. The difference is that in RTO valid data is given by 0s. Therefore, instead of computing minterms, the maxterms are the ones that need to be computed. Thus, disjunctions are used for falling transitions in PRs and conjunctions indicating when all maxterms are at 1 detect spacers. Accordigly, the PRs of this adder are specified by:

$$\begin{aligned}
A_{in}.1 \wedge B_{in}.1 &\rightarrow C_{out}.1 \uparrow \\
(\neg A_{in}.1 \wedge \neg B_{in}.1) &\rightarrow C_{out}.1 \downarrow \\
A_{in}.0 \wedge B_{in}.0 \wedge A_{in}.1 \wedge B_{in}.1 &\rightarrow C_{out}.0 \uparrow \\
(\neg A_{in}.0 \wedge \neg B_{in}.0) \vee (\neg A_{in}.0 \wedge \neg B_{in}.1) \vee (\neg A_{in}.1 \wedge \neg B_{in}.0) &\rightarrow C_{out}.0 \downarrow \\
A_{in}.0 \wedge B_{in}.0 \wedge A_{in}.1 \wedge B_{in}.1 &\rightarrow S_{out}.1 \uparrow \\
(\neg A_{in}.0 \wedge \neg B_{in}.1) \vee (\neg A_{in}.1 \wedge \neg B_{in}.0) &\rightarrow S_{out}.1 \downarrow \\
A_{in}.0 \wedge B_{in}.0 \wedge A_{in}.1 \wedge B_{in}.1 &\rightarrow S_{out}.0 \uparrow \\
(\neg A_{in}.1 \wedge \neg B_{in}.1) \vee (\neg A_{in}.0 \wedge \neg B_{in}.0) &\rightarrow S_{out}.0 \downarrow
\end{aligned} \tag{5.7}$$

Table 5.4 – Truth table for an RTO 1-of-2 half-adder. Adapted from [MPC14].

$A_{in}.1$	$A_{in}.0$	$B_{in}.1$	$B_{in}.0$	$C_{out}.1$	$C_{out}.0$	$S_{out}.1$	$S_{out}.0$
1	1	1	1	1	1	1	1
1	0	1	0	1	0	1	0
1	0	0	1	1	0	0	1
0	1	1	0	1	0	0	1
0	1	0	1	0	1	1	0

Therefore, ORs used in the RTZ classic DIMS are replaced by ANDs in the RTO version. Figure 5.8 (b) shows the associated RTO gate level schematic. Using this approach, any RTO DIMxS logic block can be implemented. Classically, AND/NAND gates are preferred over OR and NOR gates in VLSI design. A stack of NMOS transistors is present in these gates, while ORs and NORs employ a stack of PMOS transistors. Due to the fact that electron mobility is normally three times that of holes, NAND/AND gates are expected to present better power and delay trade-offs than NOR/OR gates for the same silicon area. As a consequence, RTO DIMxS circuits are expected to present a better power-delay trade-off than equivalent RTZ circuits. Additionally, the use of mixed systems combining RTO and RTZ for communicating blocks can be implemented by just inverting outputs, which can lead to further optimization degrees.

5.3.2 Power and Performance Trade Offs

To analyze the proposed template and its trade-offs with the original Dual-rail RTZ WCHB/DIMS template, consider a pipelined, natural numbers multiplier using 1-of-2 4-phase data encoding. This circuit multiplies two natural n-bit values, *A* and *B*, using a shift-and-add algorithm. Figure 5.9 shows the block diagram that maps the algorithm to handshake components. All dotted lines represent single wire control signals and full lines represent 1-of-2 data channels. The algorithm was pipelined and the shift and add iteration is unrolled into *n+2* stages.

The choice for such a case study is because it enables the separate analysis of RTO and RTZ protocols effects in registers, validity detectors, control circuits, arithmetic blocks, and yet allows overall circuit analysis. Registers are called REG_i, where *i* is its number in the pipeline and all registers are WCHB components. Inputs are initially registered in REG_0 , with size *2n*, and a zero generator is employed to initialize the value of variable *x*. This generator is represented by the box labeled 0, before REG_1. The validity of the registered data is given by the first validity detector (VD0). Also, before the next register (REG_1), the registered input B_0 has its size *n* extended to *2n* through the EXT operator, to allow shift left operations without any bit loss. Each of the next *n* pipeline stages comprises a register REG_i, with size *5n*, a validity detector VD_i and a shift and add step $STEP_i$, where

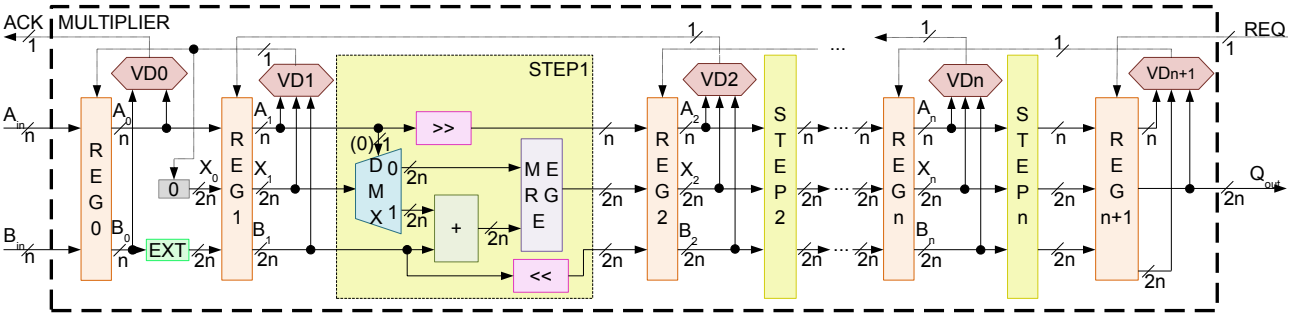


Figure 5.9 – Block diagram of the case study pipelined multiplier. Dotted lines are single wire control signals and full lines are 1-of-2 encoded data channels. Adapted from [MPC14].

$1 \leq i \leq n$. Finally, stage *n+1* has its output registered in REG_{n+1}, which produces the circuit output Q_{out} and has also size *5n*.

For comparison sake, RTZ- and RTO-based versions of this circuit were described in structural VHDL. To do so, the first step was to create handshake components as macro blocks. These blocks implement the functionalities showed in Figure 5.9 and were initially modeled using PRs. Then, the required C-elements were manually mapped to the description. Boolean logic, such as the AND/OR operations required by the blocks were extracted from PRs and mapped in the description, but the synthesis tool was able to optimize their logic using basic logic gates. Also, the macros were implemented in two flavors, assuming RTZ and RTO, where all registers were WCHBs and all logic blocks were based on DIMS logic.

The target technology for this set of experiments was STMicroelectronics CMOS 65nm. Typical gates (such as ANDs, ORs and buffers) were mapped in the Corelib provided by the vendor and C-elements were mapped to the ASCEnD-ST65 v2 library. Design optimization of the circuit was enabled by the adoption of the flow proposed in [TBV12], coupled to industrial tools [MPC14]. The specific tools employed were the RTL Compiler and Encounter from Cadence [Cad]. Thirteen distinct implementations of each RTO and RTZ designs were produced, each with a different maximum operating speed. Different speeds are obtained in the classical way, using gates with distinct drive strengths. The generated designs had their internal nets and gate delays annotated in an SDF file, which was the source to simulate the mapped netlist. Also, similar speed RTO and RTZ circuits presented equivalent silicon area, which suggests that none of the protocols can be classified as more area efficient. This is expected, because the set of cells of these templates is mostly equivalent.

The simulation scenario employed a producer of random data and a consumer, both described in SystemC. These were used to verify correct operation of the circuits and to measure delay values. Delay measurements considered the throughput of the designs, measured in millions of operations per second (MOPS), and average acknowledge delays, measured for each design by simulating it with random data on the inputs, during 1 ms. Internal activity of the nets was annotated and exported for power analysis.

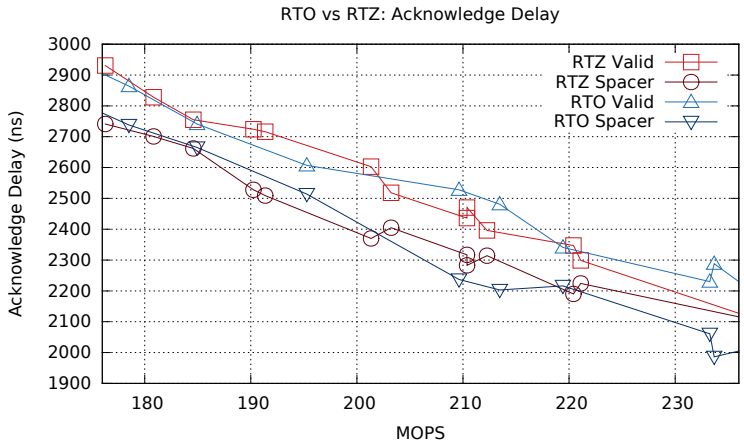


Figure 5.10 – Acknowledge delay of valid data and spacer for RTO and RTZ. Adapted from [MPC14].

Simulation results show that RTO fastest versions were able to compute 236 MOPS and the slowest versions 148, while for RTZ these values were 237 and 176, respectively. Figure 5.10 presents the average acknowledge delays for valid data and spacers for each design (note that only the intersection between these intervals is shown in the graphs). The measured values are very similar. Yet, for slow designs, under 200 MOPS, the RTO protocol presents slightly faster responses for valid data and slower responses for spacers. For the other designs, this scenario is reversed. Still, the speed-up in responses for valid data and spacers is always under 3%. Thus, neither protocol can be said to be more efficient in absolute terms for acknowledge speed.

A similar simulation scenario was employed to measure the forward propagation delay with the same producer and consumer. However, the producer generates a single pair of random data and the consumer measures how long it takes to compute data and to generate a valid output, without considering the propagation of spacers that would eventually follow it. Then, the circuit is reset and another pair of data is sent. In this way, inter-register acknowledge does not interfere while measuring the forward propagation delay. This process simulates for 1 ms. Figure 5.11 presents the measured average values.

Results show that the RTO designs present lower forward propagation delay in most cases. In the best case, this difference was around 12%. In the fastest designs, RTZ forward propagation delay equals that of RTO. In fact, for the fastest design, the RTZ version presents slightly lower forward propagation delay. Still, this result displays a clear advantage of using RTO, as forward propagation delay is a very important characteristic of handshake-based circuits. Typically, given a sender/receiver scenario, the smaller the percentage of forward propagation delay of the sender in total communication time, the faster the receiver will acknowledge. This leads to scenarios where the sender is released faster and may start a new communication earlier, which leads to overall speed up.

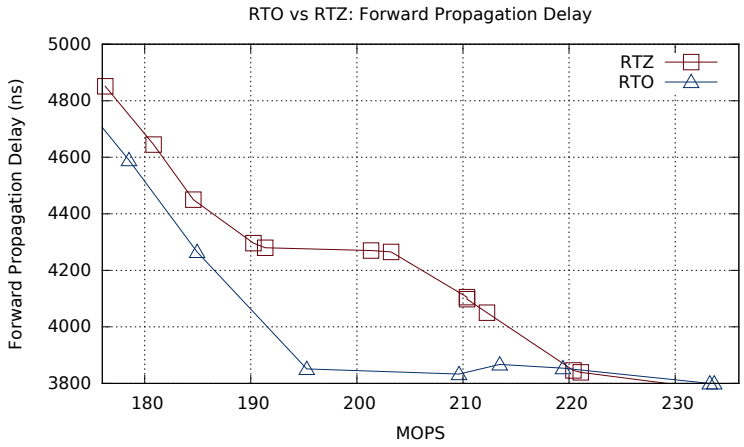


Figure 5.11 – Forward propagation delay results for the simulated designs. Adapted from [MPC14].

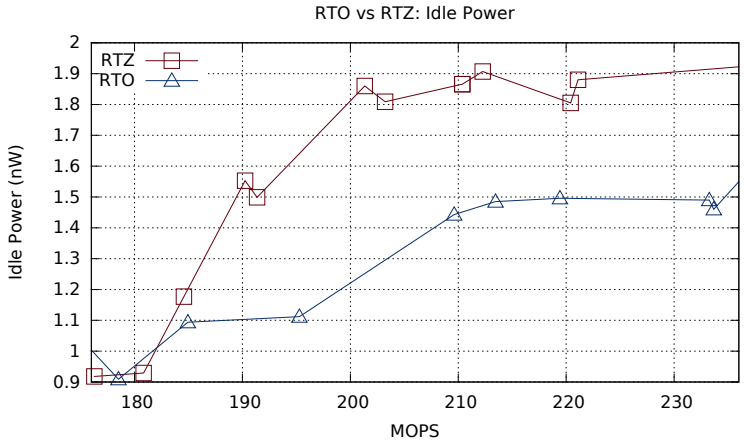


Figure 5.12 – Total power of the designs while at reset/set and idle states. Adapted from [MPC14].

The activity of all internal nets was annotated and used as the source for five power analysis scenarios: an idle state, where the circuit was reset and was not fed with any data, and four dynamic scenarios, where the circuit was fed with data during 25%, 50%, 75% and 100% of the simulation time. Figure 5.12 presents the measured total power for idle states. The obtained results display a clear advantage of the RTO protocol concerning idle power, 25% in average and 35% in the best case. This is important for asynchronous circuits, since there is no global synchronization and while some parts of the system are operating, others may be quiescent. Albeit it was expected a close to linear growth of power in the presented charts, these are slightly distorted. This is because the used synthesis method employs tools that are designed for synchronous systems that end up optimizing the critical path delay of the circuit rather than its average delay. This explains the peaks and valleys in the power charts. Nevertheless, the qualitative aspect of the results is not jeopardized. If tools adequate for asynchronous synthesis were available, these peaks and valleys would not appear in the charts, but the correlation between the latter would remain.

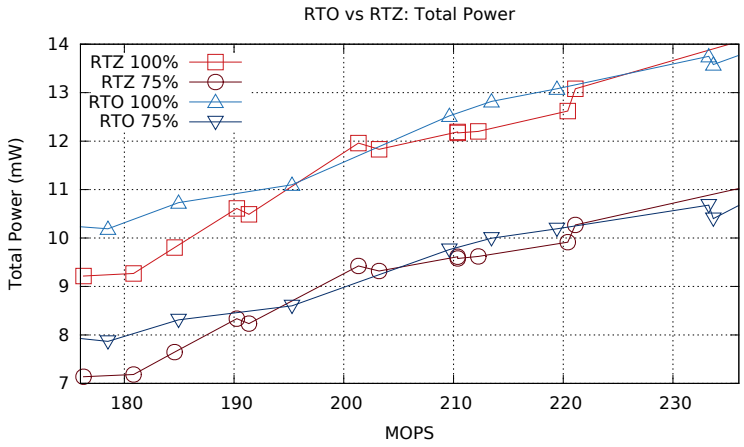


Figure 5.13 – Total power of the RTO and RTZ designs when computing 100% and 75% of the time. Adapted from [MPC14].

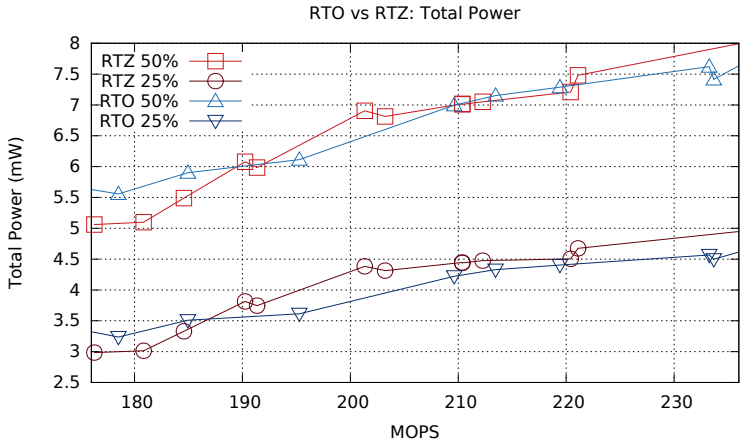


Figure 5.14 – Total power of the RTO and RTZ designs when computing 50% and 25% of the time. Adapted from [MPC14].

As for the dynamic scenarios, RTO- and RTZ-based designs present similar total power. Figures 5.13 and 5.14 present the measured total power of the circuits for such scenarios. Figure 5.13 presents the charts for 100% and 75% operating times and Figure 5.14 does the same for 50% and 25% values. Typically, RTO and RTZ charts of a same scenario are intertwined with one another. In general, RTO presents higher total power for slower designs, (8% higher in worst case). For faster designs the situation is reversed, and in the best case RTO presents reductions of over 10%. In this way, both protocols are assumed to have similar power efficiency for dynamic scenarios. In this context, at circuit level, the advantage of RTO-based systems is the reduced power in idle states. In other words, a reduction on leakage power is observed.

For the case study, effects of the lower leakage characteristic appear in the charts of Figure 5.13 and 5.14. The lower the operating time percentage is, the lower the RTO chart values become, compared to RTZ values. This characteristic of RTO can be understood by analyzing the power distribution of the simulated circuits as Figure 5.15 shows. Figures

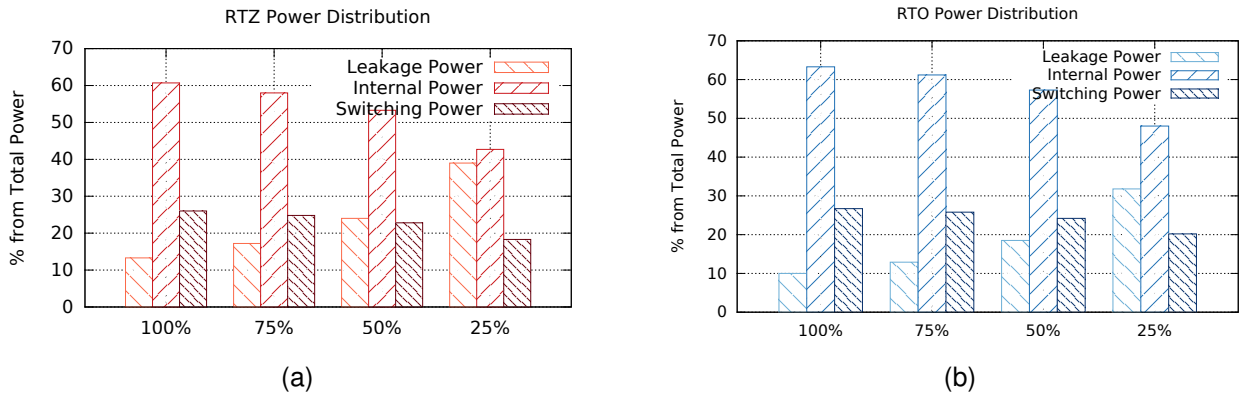


Figure 5.15 – Average power distribution of designs: (a) RTZ and (b) RTO. Adapted from [MPC14].

5.15(a) and 5.15(b) present the average power distribution of the simulated RTZ- and RTO-based circuits, respectively, for the four dynamic scenarios. In the presented charts, power is distributed in leakage, internal and dynamic power. The first is due to subthreshold leakage currents, which are a major concern for recent technology nodes [Eke10]. Internal power is due to parasitics and short-circuit currents when switching standard-cells inputs. Switching power comes from charge and discharge of capacitances when switching standard-cells outputs. The charts show leakage power represents a smaller portion of the total power in RTO, when compared to RTZ. These results point to good potential for RTO to cope with leakage power constraints in future technologies. Measured power for dynamic scenarios did not meet expectations at the system level. With the use of AND/NAND gates, rather than OR/NOR gates, a better power efficiency was expected for RTO in the dynamic scenarios as well, i.e. RTO-based designs were expected to provide the same throughput as RTZ designs with lower power.

A more precise circuit power analysis reveals that the total power of the pipeline is the sum of the power from three block types (Figure 5.9): VDs, REGs and STEPs power. It is possible to measure the isolated effects of RTO and RTZ in logic arithmetic blocks (steps), sequential blocks and validity detectors measuring the percentage of total power accountable for each block. Figure 5.16 presents the obtained results of this power analysis for the idle scenario (parts (a), (c) (e)) and for dynamic scenarios with 100% operation, in (b), (d), (f). The 100% operation scenario represents the worst case dynamic scenario for RTO justifying its choice. Note that the presented values are not absolute and, rather, charts show the percentage of total power for each block of each circuit for both simulation scenarios. Also, note that REGs and VDs portion of power decreases as designs get faster, while the STEPs portion increases. This is mostly due to critical path optimizations in synthesis.

As charts in Figures 5.16(a), 5.16(c) and 5.16(e) show, the distribution of RTO and RTZ total power is similar in idle state. This demonstrates that reduction in leakage power, when using RTO, reflects at the circuit level. However, for dynamic state results,

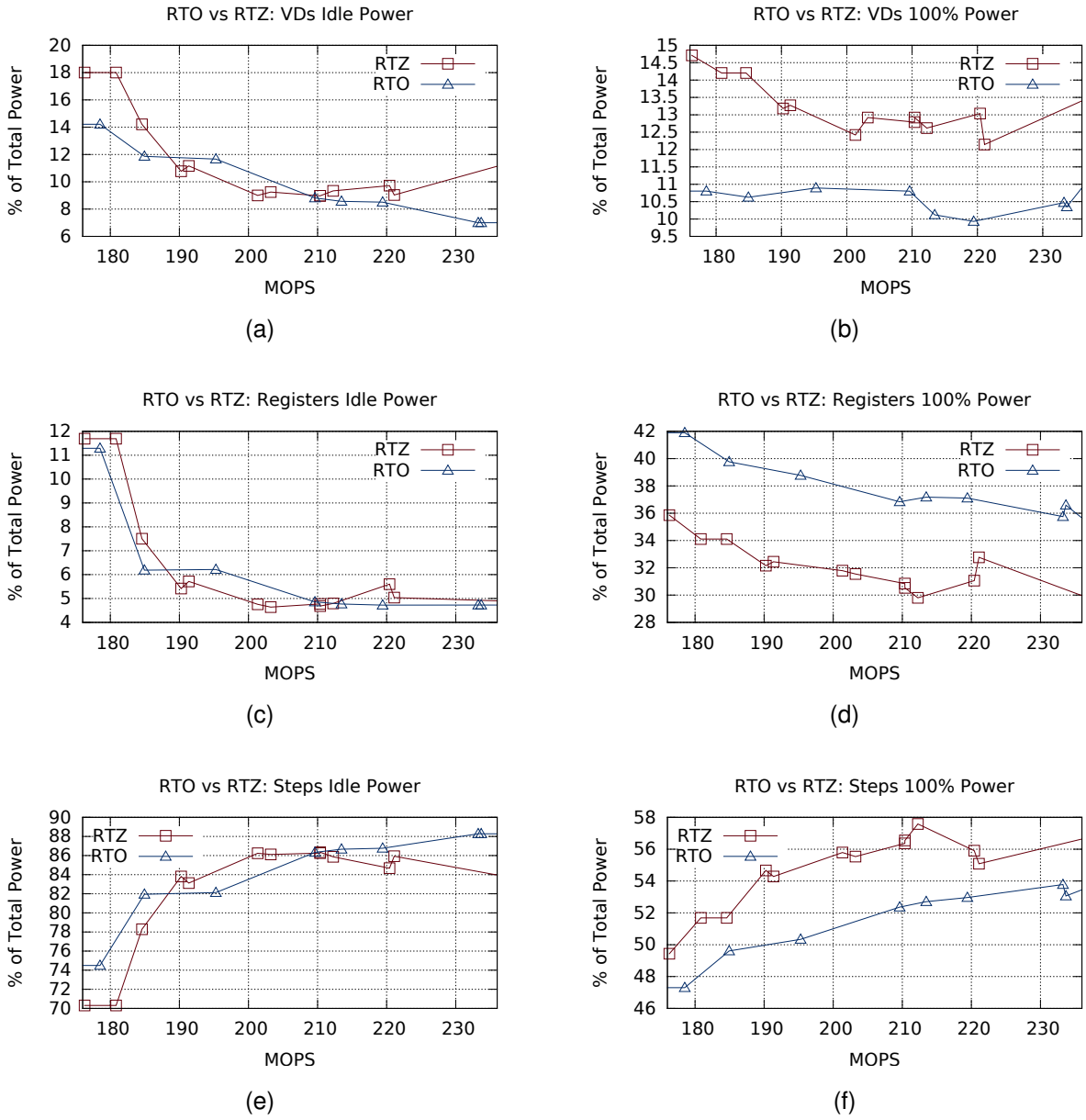


Figure 5.16 – Power distribution in the multiplier components for a scenario where the circuit is computing 100% of the time. VD stands for validity detector. Adapted from [MPC14].

RTO and RTZ display significant discrepancies. For VDs and STEPs power, significant savings occur when employing RTO. This agrees with expected results, since these blocks employ AND/NAND gates rather than OR/NOR gates, which is classically preferable in VLSI synthesis. Thus, the obtained results point to design optimizations when using RTO for arithmetic DIMxS-based circuits and validity detectors. On the other hand, as Figure 5.16(d) shows, the RTZ REGs blocks present a significantly lower part of the total power, when compared to RTO.

These blocks are composed exclusively of resettable or settable C-elements. The former are used in RTZ and the latter in RTO. Results suggest that, albeit settable C-elements of RTO present lower static power than resettable C-elements of RTZ, their dy-

namic power is bigger. In fact, an accurate analysis of the characterized timing and power models of the ASCEnD-ST65 library confirmed these conclusions. One explanation is the fact that the settable C-element employs a stack of three PMOS transistors, while in the resettable C-elements this stack uses NMOS transistors. This discussion leads to the use of smaller transistors in resettable C-elements. The issue is independent of C-element topology. Thus, results point to power reductions when using RTZ-based WCHB registers. Case study data suggest the RTO protocol is more power efficient in dynamic scenarios for validity detectors and logic arithmetic blocks, while RTZ is more suited for buffers.

Therefore, a system that employs both protocols is expected to provide further optimizations. Differently from the works presented in [Sok06, MY06, MAM+03a, CLP+13], which propose the use of two distinct spacers that are *temporally distributed*, mixing RTZ and RTO as described here implies the use of two spacers *spatially distributed*. In this way, avoidance of the hardware complexity and area overheads observed in [Sok06] is possible. Also, translation of RTZ signals into RTO and vice-versa is very simple (See Equation 5.1). In the worst case, it requires one inverter per wire. However, resettable C-elements of RTZ registers already have inverters in their outputs, required by their memory scheme. Then, the output conversion of these blocks to RTO can come for free. Also, AND gates in arithmetic RTO blocks right before an RTZ block can be changed to NAND gates, implying further optimization.

The combination of RTZ and RTO in a single system is expected to provide much lower total power, given that, as Figure 5.16(d) shows, REGs are responsible for a minimum of 35% (in the worst case, over 40%) of the total power of RTO-only circuits. Because the portion of idle power of REGs for both protocols is usually lower than 6.5% (over 10% in a few cases), as Figure 5.16(c) shows, the use of RTZ REGs is not expected to cause significant increase in circuits' idle power. Moreover, mixed RTO/RTZ circuits can take advantage of the benefits of each protocol for each different block, leading to an enlarged design optimization space. For instance, according to Figure 5.16(b), faster RTO VDs can be obtained for a same power budget of their RTZ version. These circuits acknowledge inputs faster, enabling better communication at system level, and potentially leading to overall circuit speed-up. Finally, as mentioned before, Equation 5.1 satisfies the conditions stated in [Mar90] for maintaining the circuit DI property. In this way, mixing RTO and RTZ does not jeopardize the functionality of a QDI circuit.

5.3.3 Template Robustness

Although QDI logic allows achieving delay insensitivity, as any digital circuit it is not insensitive to single event transients (SETs) affecting its signals. These faults are unavoidable in CMOS technologies and can be caused by effects such as glitches produced

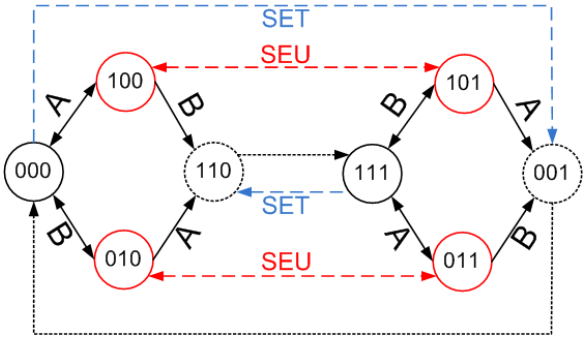


Figure 5.17 – State transition graph (ABQ) of a C-element considering SET and SEU effects [PCV12]. Full-line nodes are static states. Dotted-line nodes are transition (unstable) states. Dashed lines are transitions that generate SETs or SEUs in the output. Dotted lines are transitions from unstable to stable states. Adapted from [MGHC14].

by crosstalk noise, radiation or charge sharing [RCN03]. Specially when using DIMS or DIMxS design styles, the correct functionality of a QDI circuit can be easily jeopardized if a glitch propagates to inputs that directly feed C-elements, because these components are quite vulnerable to transient faults [Ber92, OMAF10]. The same problem arises if a glitch is generated in an internal node or in the output of the C-element. Depending on the C-element internal state, glitches large enough can be latched, which produces a single event upset (SEU). This is particularly problematic because SEUs can corrupt their stored data, and the correct functionality of the circuit. Hence, is is useful to evaluate the robustness of the proposed template and compare it to the dual-rail RTZ WCHB/DIMS template.

Pontes et al. [PCV12] and Bastos et al. [BSK+10] demonstrated that C-elements can be rather vulnerable to transient faults. When operating as a buffer (when the inputs have the same value), the worst consequence is the generation of a SET in the output, which propagates as a glitch. However, when operating as a memory (when the inputs have different values) this SET can be latched and generate an SEU. For instance, Figure 5.17 shows the state transition graph of a 2-input C-element (with inputs A and B and output Q) considering SEU and SET generation, as reported in [PCV12]. Accordingly, the graph shows that when both inputs are at the same logic value, the output assumes this logic value and enters in a stable state where Q=A=B. During this state, any effect that causes the output to switch do not generate an SEU, because the inputs ensure the output value. In fact, the result is the generation of an SET, where the output incorrectly switches for a moment and then switches back, given that $A = B$, generating a glitch. But when $A \neq B$, the C-element relies only on its internal memory scheme to keep the output. If the output switches incorrectly, this transition can be latched and generate an SEU. This fault will not be forced back to the correct value, because $A \neq B$. Hence, we refer to the memorizing states of a C-element as vulnerable states.

Some previous works explore the susceptibility of C-elements to transient faults. Mohammadi et al. [MFG03] investigate the effects of charge sharing on C-elements internal

nodes. They demonstrate that glitches caused by such effects are a significant source of potential failure and propose modifications in the C-element topology to alleviate the problem. Vaidyanathan et al. [VXVZ06] evaluate the sensitivity of C-elements to radiation effects and propose a new transistor topology to increase robustness. None of these works presents a solution to cope with glitches that are propagated to the output, which will end up in the input of another cell, potentially a C-element in a logic block. Bastos et al. [BSK+10] identify all the vulnerable logic states of C-elements, including possible propagation of transient faults in the inputs and demonstrate that the correct choice of transistor topology can mitigate, but not eliminate the problem. These authors propose a technique for further improvements, which consists in increasing internal nodes capacitance for filtering transient faults, but this incurs in additional delay/power/area penalties and is susceptible to PVT variation effects.

All the revised works propose modifications in the C-element at the circuit level to harden QDI circuits against transient faults, but all of these degrade performance and increase area. Also, most do not explore techniques for coping with possible transient faults that can propagate to the inputs of C-elements. In a first experiment, we simulate the three classic C-element topologies, all available in the ASCEnD-ST65 v2, for vulnerable states (100, 010, 101 and 011 from Figure 5.17). For each state, we simulate the injection of a trapezoidal glitch in the input that can cause the output to switch. For example, for states 100 and 010, 0 to 1 glitches were injected in B and A, respectively, and for states 101 and 011, 1 to 0 glitches were injected in A and B, respectively. Such glitches had a varying height from 5 mV to 1 V in steps of 5 mV and a varying width, from 1 ps to 200 ps, in steps of 1 ps. Note that for 1 to 0 glitches the height was given as a negative variation. According to the work presented in [ORM04], such setup provides grounds for a realistic analysis. Performing such investigation required a total of 200*200*4=160,000 simulation scenarios for each C-element, totalizing 480,000 simulations. Experiments use the Spectre simulator, from Cadence, employing post-layout extraction C-elements from ASCEnD-ST65. Extractions occur for typical fabrication process and typical operating conditions (1 V and 25 C). For all scenarios, C-elements have an output load equivalent to four inverters of the same driving strength input capacitances (FO4).

During simulation, we measure the variation in the voltage of the output for each scenario. The obtained results are summarized in Figure 5.18. Results for the semi-static C-element are showed in Figures 5.18(a) and 5.18(b), for the static C-element in Figures 5.18(c) and 5.18(d) and for the symmetric C-element in Figures 5.18(e) and 5.18(f). The first column of charts presents the results collected for 1 to 0 glitches (Glitch Down). In these charts, the worst case between 011 and 101 was collected. Similarly, the second column of charts present worst case for 0 to 1 glitches (Glitch Up), collected as the worst case between 010 and 100 states. Note that the points where the charts of the first column reach the bottom indicate scenarios (combination of a height and width for a glitch) that generate a SEU, because the output switched its logic value. The same applies to the points in the

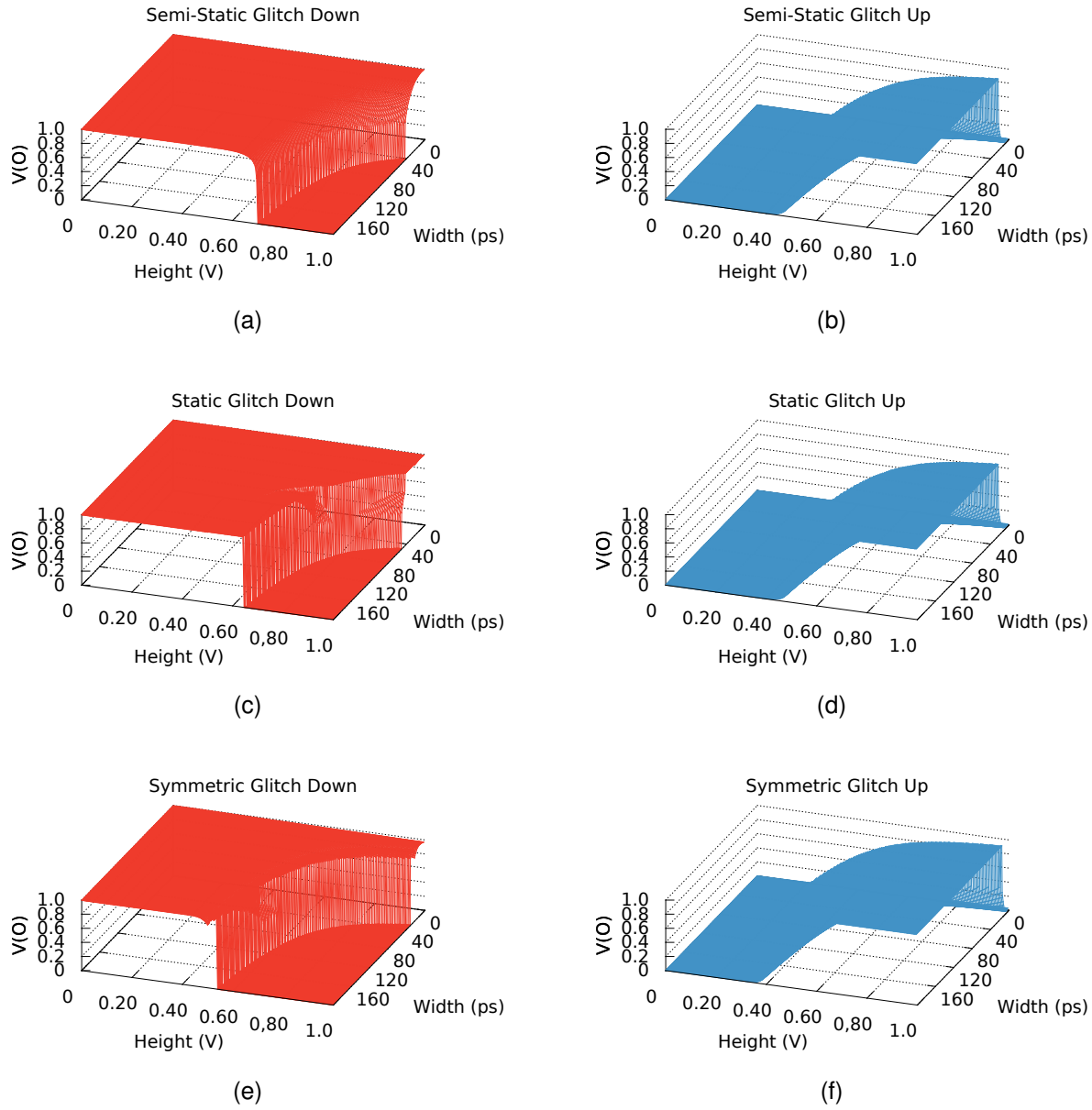


Figure 5.18 – Output voltage variation in the semi-static (a) and (b), static (c) and (d) and symmetric (e) and (f) topologies. Glitch Up stands for 010 and 100 scenarios and Glitch Down stands for 011 and 101 scenarios, where the worst case result was collected. Adapted from [MPC14].

charts of the second column that reach the top. We refer to points in the frontier of those cases that generate SEUs as critical points, since they represent the minimum width for each glitch height (or vice-versa) that produces SEUs.

Note that the region beyond the critical points for the semi-static topology is smaller than those for the static and symmetric topologies, displaying the increased robustness of this topology and confirming results pointed in [BSK+10]. However, one interesting characteristic of C-elements is the big discrepancy between critical points for 0 to 1 and 1 to 0 glitches. Analyzing the region beyond the critical points of the charts in the second column of Figure 5.18, it is clear that they imply a much bigger region than those of the charts of the

first column. What is more alarming is that for the semi-static C-element, for instance, 0 to 1 glitches with heights of roughly 0.5 V can already generate SEUs, while for 1 to 0 glitches the minimum height is of over 0.7 V. Also, for the same topology, the minimum width for 0 to 1 glitches to produce an SEU is 22 ps, while for 1 to 0 glitches it is 42 ps. A similar analysis applies to the other topologies. In this way, it is clear that vulnerable states where the output of the C-element is fixed at 0 are much more sensitive to glitches in the inputs than those where the output is at 1.

Another perspective of the results allows quantifying this sensitivity more clearly. Figure 5.19 presents the isolated critical points for the semi-static 5.19(a), static 5.19(b) and symmetric 5.19(c) topologies, obtained from those of Figure 5.18. Accordingly, the minimum heights for generating SEUs in the outputs of these C-elements for 0 to 1 glitches are roughly 0.5 V, 0.5 V and 0.4 V, respectively. For 1 to 0 glitches these values change to 0.7 V, 0.65 V and 0.55 V, respectively. Also, for the same heights, 1 to 0 glitches require much bigger widths than 0 to 1 glitches to produce SEUs. The right Y axis of the charts of Figure 5.19 quantifies how bigger these widths need to be. For a worst case, these values are of 91%, 127% and 135% for semi-static, static and symmetric, respectively. However, the worst case is the one where glitches have bigger heights. If we evaluate the effects of glitches with small heights, the obtained improvements are much more significant, reaching up to 311%, 237% and 215% for semi-static, static and symmetric, respectively. This means that for the semi-static topology, which is preferable for robust applications, 1 to 0 glitches in the inputs require almost twice the width of 0 to 1 glitches to generate SEUs in worst case and over four times in the best case.

These results indicate that submitting C-elements to vulnerable states with 1 in the output rather than those with 0 in the output allows mitigating problems caused by glitches in the inputs during such states, which enables hardening QDI circuits against transient faults. An analysis of the behavior of DIMS logic blocks reveals that the C-elements in this design style are often submitted to vulnerable states with 0s in the output. Meanwhile, in DIMxS blocks subject to vulnerable states, C-elements components have 1s in the outputs.

Figure 5.20 presents an example of a 2-input dual-rail DIMS AND logic block. Given that this design style is used with DI channels where a spacer precedes each valid data transmission, the reset (start) state for this circuit is all inputs at 0, which sets all internal nodes M00, M01, M10 and M11 to 0 as well. As soon as there are valid data in inputs A and B (represented by their 1-of-2 wires A.0, A.1, B.0 and B.1), one of the internal nodes will be set to 1, switching the output signal directly or triggering the OR gate, which will switch the output. For instance, if both inputs are 1 (A.1=1 and B.1=1), M11 will be set to 1, writing logic 1 in Q.1. The remaining internal nodes will still be at logic 0, keeping Q.0 in 0. This produces a logic 1 in the output. On the other hand, if at least one input is 0, as soon as both inputs have their values available in the C-elements inputs, either M00, M01 or M10 will

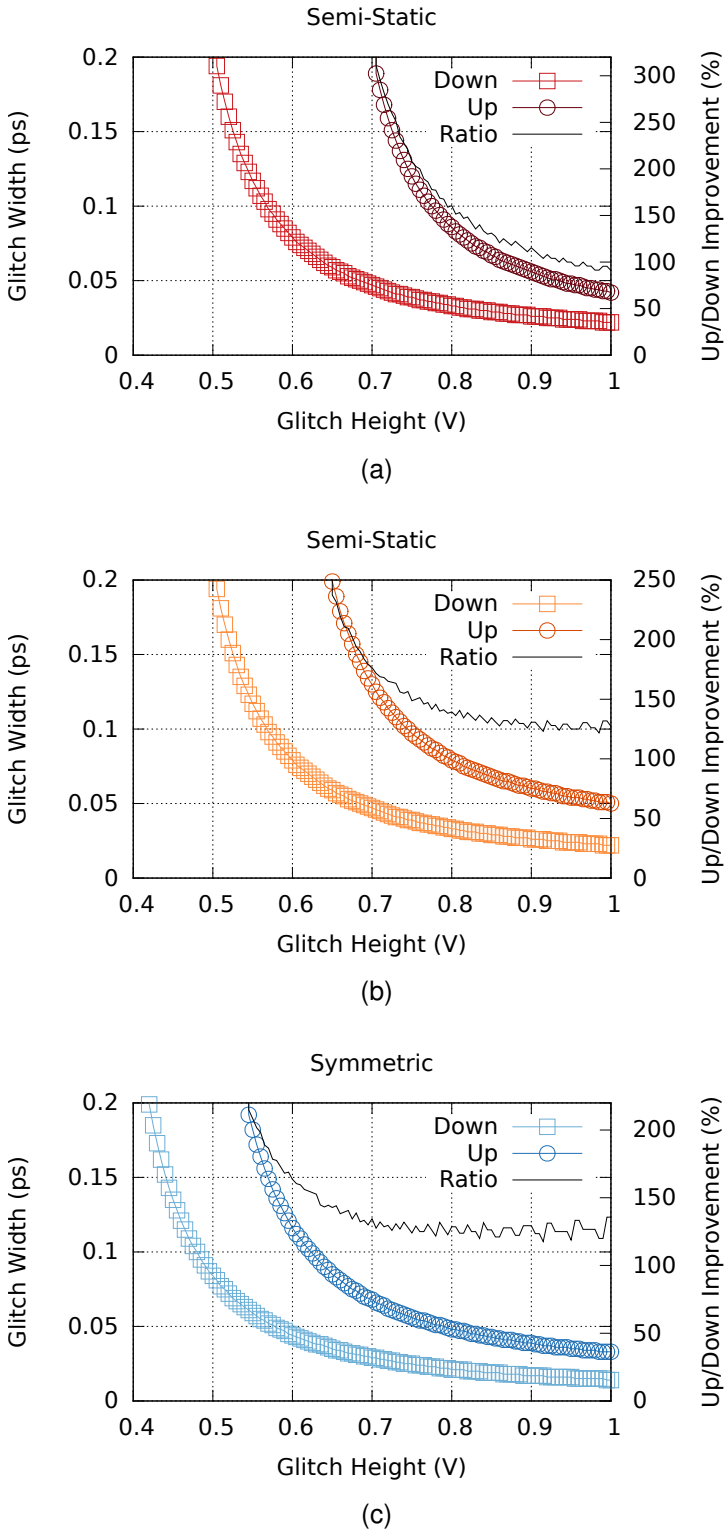


Figure 5.19 – Isolated critical points for semi-static (a), static (b) and symmetric (c) topologies. Adapted from [MPC14].

be set to 1, writing 1 in Q.0, through the OR gate. Q.1 will still hold a 0, due to the previous spacer.

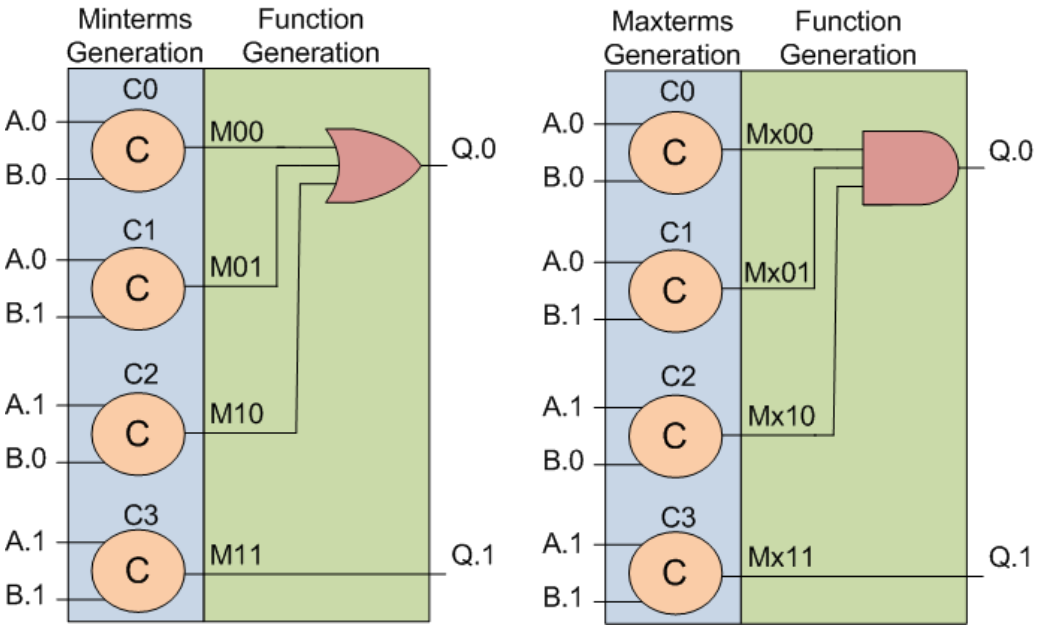


Figure 5.20 – Schematics for 2-input, 1-of-2 AND in DIMS and DIMxS. Adapted from [MGHC14].

Recalling Section 5.3.1, the construction of DIMxS circuits is similar to DIMS. Figure 5.20 also shows the schematic of a 2-input dual-rail DIMxS AND logic block. In this case, every data transmission is preceded by an all-1s spacer and C-elements generate the maxterms for the set of inputs, which are combined through an AND that implements the function output. In this way, the value of each signal of a DIMxS block is exactly the inverse of those of a DIMS block during operation.

Figure 5.21 shows the transition diagram for the 2-input DIMS and DIMxS AND blocks. Communications always start with two spacers. Next, valid data is inserted in both inputs, producing an output. Then, inputs must return to spacers so that new values can follow. The state of each C-element for the DIMS AND, for each state of Figure 5.21 appears in Table 5.5. The only state where no C-element is at a vulnerable state is with two input spacers. For all other 8 states there are always two C-elements vulnerable. This shows that the problem cannot be ignored, as most of the time components are vulnerable. Given that in RTZ spacers are represented by the all-0s codeword, vulnerable states of C-elements in DIMS blocks are always with a 0 in the output.

Table 5.6 presents the same analysis for a similar DIMxS logic block. In this case, there is also a single state where C-elements are not vulnerable. However, in all remaining vulnerable states, components have 1s in the output, because spacers in RTO are represented by the all-1s codeword. From Figure 5.20 it is clear that any fault generated by C-elements C0, C1, C2 or C3 corrupt the values of M00, M01, M10 or M11, causing the circuit to operate incorrectly. The same is valid for the equivalent DIMxS circuit. Therefore, we believe that employing DIMxS rather than DIMS logic blocks mitigates problems caused by transient effects, as these effects are more expressive when the output of C-elements

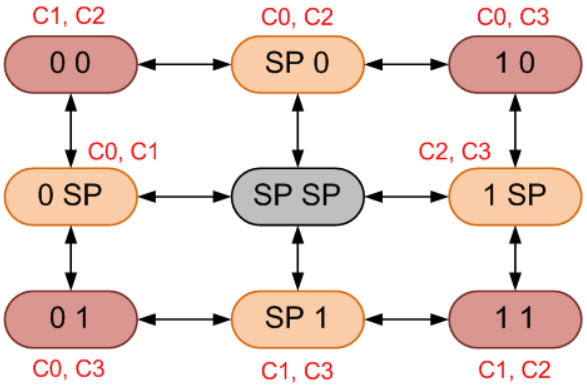


Figure 5.21 – Transition diagram for 2-input DIMS and DIMxS AND blocks. Inputs are assumed to use the 1-of-2 code. SP stands for spacer. Adapted from [MGHC14].

is at logic 0. To do so, RTO must be adopted in the design rather than RTZ. This can be done globally or locally in QDI circuits, as RTZ and RTO are compatible. This approach does not require modifications in the C-elements and does not increase area or power. The only modification is in the protocol assumption, which requires the usage of DIMxS rather than DIMS. In other words, OR gates are replaced by AND gates.

5.4 The NCL+ Template

Another development proposed with basis on RTO channels is a new template combining RTO and NCL. To do so, modifications were required in the structure of the basic cells. Yet, the overall architecture is still similar to the original NCL. This new template was called NCL+ [MOPC13b].

Table 5.5 – States of the C-elements for a 2-input DIMS AND. Inputs are assumed to be dual-rail and SP stands for spacer. Adapted from [MGHC14].

Logic Values		**Wires' Logic Values**				**C-elements' states**			
A	**B**	**A.1**	**A.0**	**B.1**	**B.0**	**C0**	**C1**	**C2**	**C3**
SP	SP	0	0	0	0	000	000	000	000
SP	0	0	0	0	1	010	000	010	000
SP	1	0	0	1	0	000	010	000	010
0	SP	0	1	0	0	100	100	000	000
0	0	0	1	0	1	111	100	010	000
0	1	0	1	1	0	100	111	000	010
1	SP	1	0	0	0	000	000	100	100
1	0	1	0	0	1	010	000	111	100
1	1	1	0	1	0	000	010	100	111

Table 5.6 – States of the C-elements for a 2-input DIMxS AND. Inputs are assumed to be dual-rail and SP stands for spacer. Adapted from [MGHC14].

Logic Values		Wires' Logic Values				C-elements' states			
A	**B**	**A.1**	**A.0**	**B.1**	**B.0**	**C0**	**C1**	**C2**	**C3**
SP	SP	1	1	1	1	111	111	111	111
SP	0	1	1	1	0	101	111	101	111
SP	1	1	1	0	1	111	101	111	101
0	SP	1	0	1	1	011	011	111	111
0	0	1	0	1	0	000	011	101	111
0	1	1	0	0	1	011	000	111	101
1	SP	0	1	1	1	111	111	011	011
1	0	0	1	1	0	101	111	000	011
1	1	0	1	0	1	111	101	011	

5.4.1 Proposed Design Style

Recalling Section 5.1.2, in NCL gates the output switches to logic '0' when all N inputs are at logic '0' and to logic '1' when the weights of inputs at logic '1' sum at least the value of the threshold T. Otherwise, it keeps its output state. There are different ways to design NCL gates in CMOS, such as the six approaches that Parsan and Smith discuss in [PS12a]. Among these, classical static implementations are of particular interest, as they are simpler and present a good area-power compromise. Accordingly, we assume here the use of static implementations only, albeit the proposed techniques apply for other topologies as well.

In NCL+, gates also implement a threshold function, a subset of which appears in Figure 5.22, where T is the defined gate threshold (written inside each gate). However, the assumption of the RTO protocol mandates the switching function of an NCL+ gate to be the reverse of its NCL counterpart: the output will only switch to logic '1' when all inputs are at logic '1' and will only switch to logic '0' when the weights of inputs at logic '0' sum at least the value of the threshold T. For other combinations of inputs, the output keeps the previous value. In this way, an NCL+ gate is defined as:

Definition 4. *An* NCL+ gate *is an n-input logic gate with a threshold* $T \in N^*$*, a specific weight* $w_i \in N^*$ *assigned to each Boolean input* x_i*, and a hysteresis mechanism to ensure a sequential behavior, such that its output* Q_i *at each instant of time i is given by:*

$$Q_i = \begin{cases} 1, & \sum_{j=0}^{n-1} w_j \overline{x_j} = 0 \\ 0, & \sum_{j=0}^{n-1} w_j \overline{x_j} \geq T \\ Q_{i-1}, & 0 < \sum_{j=0}^{n-1} w_j \overline{x_j} < T \end{cases} \tag{5.8}$$

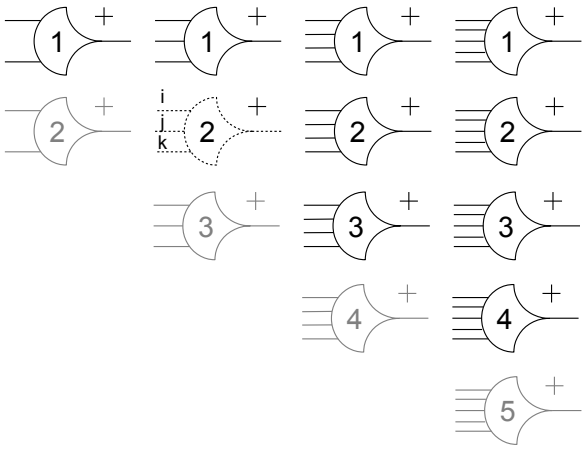


Figure 5.22 – A basic set of 14 NCL+ gates. Adapted from [MOPC13b].

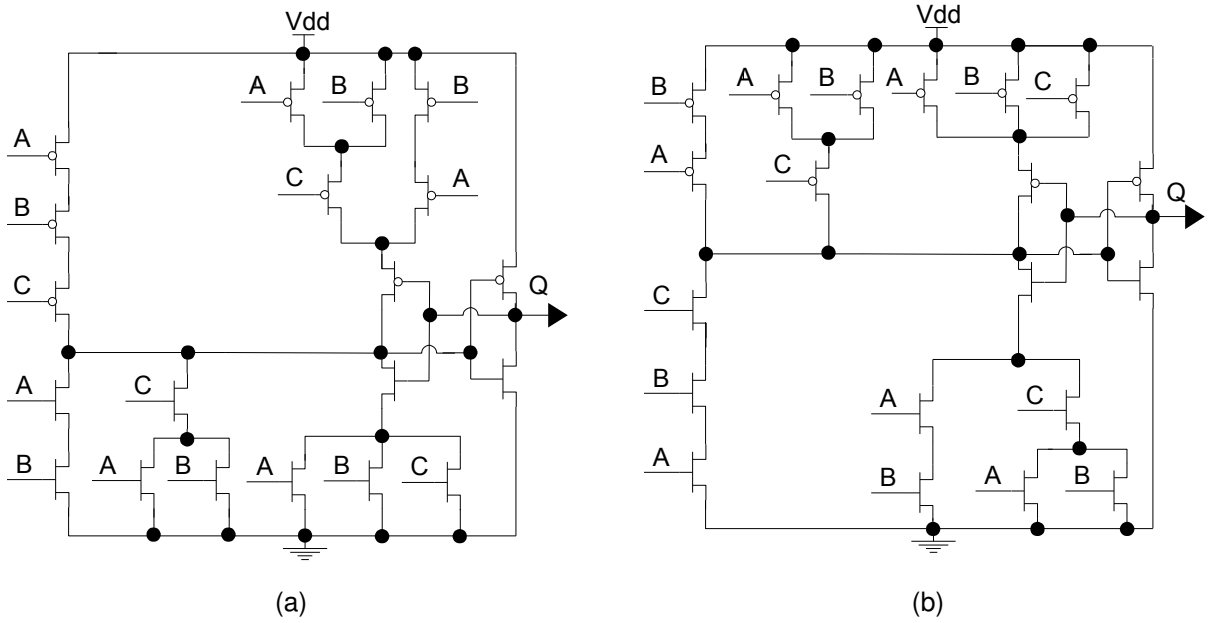


Figure 5.23 – Example transistor topologies for NCL and NCL+ gates (2-of-3 gates): (a) topology for the 3-input T=2 NCL gate; (b) an NCL+ gate with the same number of inputs and threshold. Adapted from [MOPC13b].

Figure 5.23(a) shows the CMOS schematic of a 3-input, T=2 NCL gate, and Figure 5.23(b) shows the schematic of an equivalent NCL+ gate. Note that the gate drawn with dotted lines in Figure 5.22 is the 2-of-3 gate described in Figure 5.23.

5.4.2 Experiments

To compare NCL+ to NCL, we designed a set of 70 NCL+ threshold gates at the layout level as standard-cells in the ASCEnD-ST65 library. The fourteen distinct threshold functions described in Figure 5.22 were designed, each being implemented in five different driving strengths: X2, X4, X7, X9 and X18. This set effectively supports a wide variety

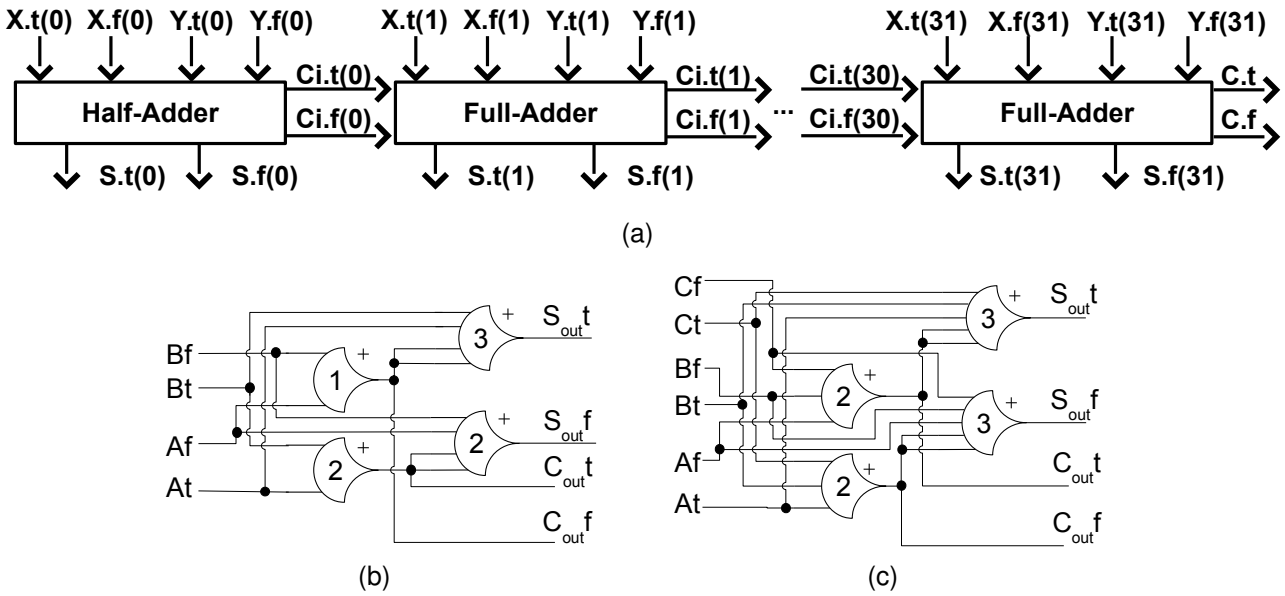


Figure 5.24 – A 32-bit ripple carry adder: (a) the block diagram; gate level schematics of the NCL+ (b) half adder and (c) full adder. Adapted from [MOPC13b].

of functionalities. Note that the shadowed gates, where the threshold is the same as the number of inputs. These have exactly the same implementation in either NCL or NCL+, as the threshold for the output to switch to logic '1' or to logic '0' is the same. Using these cells, a 32-bit ripple carry adder was described in SPICE, corresponding to the block diagram of Figure 5.24(a). The circuit writes the sum of X and Y to the S output and produces carry signal C. The internal carry signals are represented by Ci. Five versions of the adder exist, one for each of the driving strengths.

For comparison, the adder was also implemented using the NCL template for the same five driving strengths. Previously designed NCL gates, available in the ASCEnD-ST65 library, were employed. The gate level schematics for the NCL implementations are the same as those of NCL+, but NCL+ gates are replaced by NCL gates of the same threshold. The choice for this case study circuit is due to the fact that it employs a series connection of a half adder and a series of 31 full adders, resulting in long gate level combinational paths. This is a good case study for evaluating the efficiency of threshold gates in each template in terms of delay and power. Also, it provides a fixed fanout of 4 (FO4) for gates that generate internal carry signals, avoiding unrealistic power and timing analysis.

To scrutinize the effects of each design style, a mixed-signal simulation environment was set up for each of the 10 adders. It consists of a VHDL testbench that instantiates an adder described in Spice and a verification block, described in SystemC. A VHDL-AMS description was used to convert analog to digital signals and vice-versa. All Spice descriptions employed post-layout extracted gates, in order to account for parasitics. The employed simulators for digital and analog parts were Cadence Incisive and Spectre, respectively. Simulation scenarios assumed typical fabrication process and typical operating conditions (25C

Table 5.7 – Simulation results for NCL adder. Adapted from [MOPC13b].

	NCL				
	X2	**X4**	**X7**	**X9**	**X18**
MOPS	536	622	691	694	835
Forward Delay (ps)	564	485	444	488	424
Total Delay (ps)	1866	1608	1446	1440	1197
Dynamic Power (mW)	0.370	0.530	0.845	1.164	1.748
Idle Power (μW)	2.785	3.898	6.032	8.002	10.343

Table 5.8 – Simulation results for NCL+ adder. Adapted from [MOPC13b].

	NCL+				
	X2	**X4**	**X7**	**X9**	**X18**
MOPS	600	657	673	744	820
Forward Delay (ps)	692	673	602	549	473
Total Delay (ps)	1668	1521	1484	1344	1219
Dynamic Power (mW)	0.409	0.549	0.711	0.901	1.745
Idle Power (μW)	2.245	3.131	4.069	5.159	9.587

and 1V). Also, the adder input slope was fixed at 50ps and output loads varied depending on the cells' driving strength, to obtain a load equivalent to FO4 in the last full-adder and in the S outputs as well.

The SystemC verification block consists of a random data generator that feeds the adder inputs, and a data checker that verifies its correct behaviour. Also, this block measures average, best and worst case delays to perform the sum of two values for forward delay and transmission delay. It also measures the module throughput, in millions of operations per second (MOPS). The analog simulator enabled the measurement of precise dynamic and idle power values. The former is given as the average power of the adder when operating, while the latter is the average power when idle (quiescent and filled with spacer data, i.e. all-0s for NCL and all-1s for NCL+). As dual-rail data values are always balanced for both logic values '1' and '0', random data can be used without compromising the generality of the obtained results. Tables 5.7 and 5.8 summarize the measured values for MOPS, forward propagation delay, total computation delay, dynamic power and idle power. Area results are omitted, because the observed areas for the same driving strength circuits are always practically equivalent. Furthermore, in terms of standard-cell area, NCL and NCL+ are typically equivalent, because they require the same number of transistors and have similar topology.

As the Tables show, circuits with similar driving strength have similar MOPS. However some of them present considerable deviations, like X2, for instance, which presents almost 12% higher MOPS for NCL+. These deviations are due to parasitic and gate capacitance effects, given that the delay of the standard-cells depends not only on its driving strength but also on factors such as input slope and output load, which are directly related to these effects. Also, deviations were expected, given the differences in NCL and NCL+

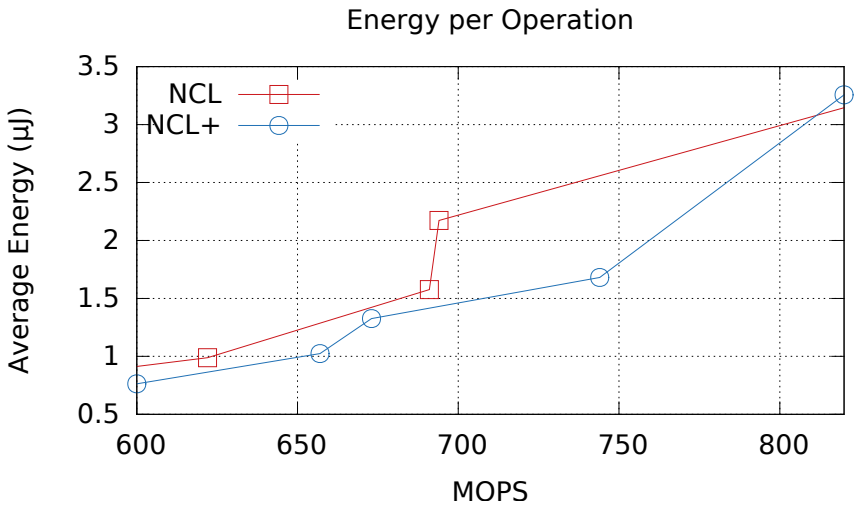


Figure 5.25 – Energy per operation for the NCL and NCL+ adders. Adapted from [MOPC13b].

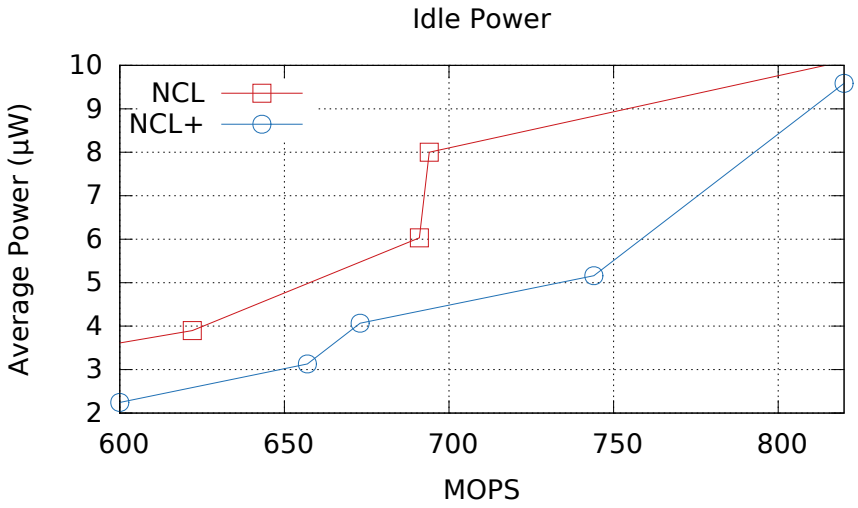


Figure 5.26 – Idle power for the NCL and NCL+ adders. Adapted from [MOPC13b].

gate design processes. Figures 5.25, 5.26 and 5.27 give another perspective on the results. First, Figure 5.25 presents the average energy consumption per operation for each NCL and NCL+ design. These results were obtained from the measured dynamic power and MOPS of each design. Similarly, Figure 5.26 presents the results for idle power and Figure 5.27 the percentage of forward propagation delay in total delay. Note that only the intersection between similar MOPS intervals appears in the graphs.

The obtained results suggest that NCL+ is more energy efficient when compared to NCL. As Figure 5.25 shows, its energy consumption per operation is typically lower for equivalent MOPS designs. Also, as Figure 5.26 shows, the total idle power is lower in all cases, presenting significant savings and suggesting lower leakage currents at the standard-cell level for NCL+. This is particularly interesting for asynchronous designs, as some parts

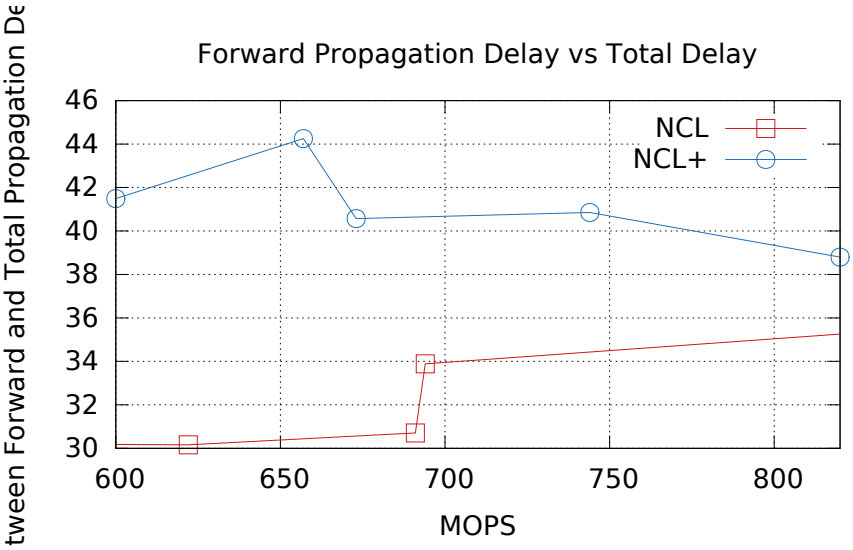


Figure 5.27 – Percentage of the total delay represented by forward propagation delay for the NCL and NCL+ adders. Adapted from [MOPC13b].

of these circuits are usually quiescent while others are operating. In this way, NCL+ potentially leads to systematic total power savings. However, for forward propagation delay, results show that NCL+ presents worse results in all cases. In this way, the obtained results point to a power and forward propagation delay trade-off for NCL and NCL+.

5.5 The SDDS-NCL Template

A natural next step from the NCL+ template is to mix it with the classic NCL template, given that mixing RTO- and RTZ-based templates enables better design space exploration, as explained in Section 5.3. To do so, there is no significant hardware overhead because the conversion between RTO and RTZ domains requires only a signal inversion. This Section proposes a new template that is based in both NCL and NCL+ gates and a design flow that allows automating the process of technology mapping in the synthesis of circuits based on this template.

5.5.1 Proposed Architecture

As discussed in Sections 5.1.2 and 5.4, NCL and NCL+ gates can be used for constructing QDI logic blocks. For instance, recalling the Kogge-Stone adder showed in Figure 5.4, the generate path of one of its *YBs* can be constructed with a series of equivalent AND and OR gates, see Figure 5.4(c). In this way, according to the schematics showed in Figures 5.4(f) and 5.4(g), the resultant NCL circuit is equivalent to the schematic showed

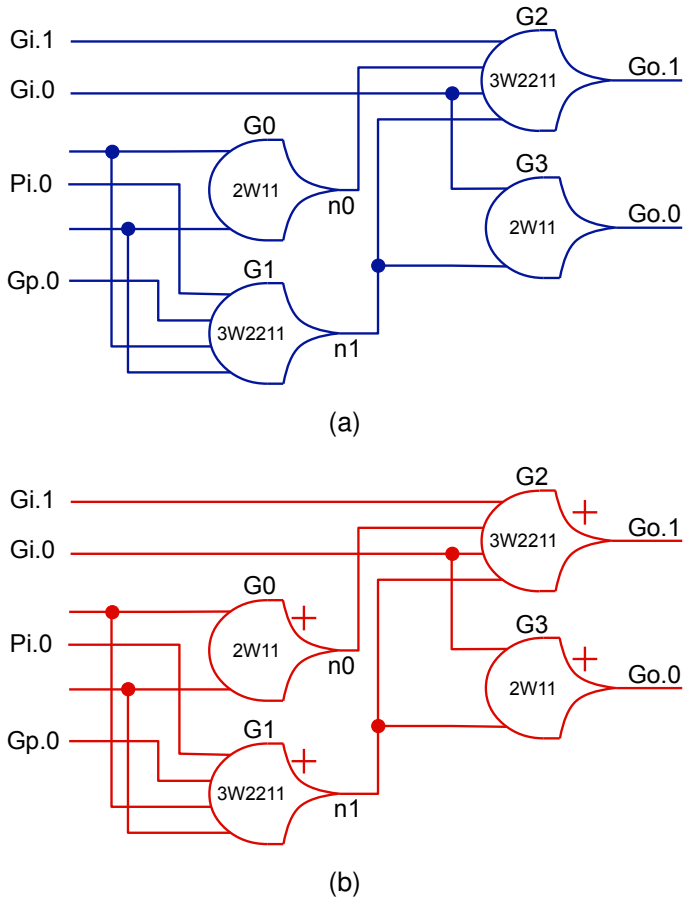


Figure 5.28 – NCl and NCL+ example implementations of the generate path of block YB from the Kogge0Stone adder of Figure 5.4: (a) NCL implementation; (b) NCL+ implementation. Adapted from [MTMC14].

in Figure 5.28(a). Recalling Section 5.4, according to Equation 5.1, the equivalent NCL+ implementation of the same circuit can be extracted. The schematics in Figure 5.28 show the NCL and the NCL+ versions of the same generate path of the yellow box of a Kogge-Stone adder. The NCL+ appears in Figure 5.28(b). Note that the structure of the circuit is exactly the same of that using NCL gates. In fact, even the function of the gates is the same. The only difference is that NCL gates are replaced by equivalent NCL+ gates and all internal nodes and primary inputs and outputs follow the RTO protocol, instead of the RTZ one.

The advent of NCL+ brought the possibility of mixing NCL and NCL+ in a single circuit, as RTO and RTZ are compatible [MPC14]. Recalling Section 5.2, the conversion of a signal from RTO to RTZ, or the other way around, requires only inverters, which implement negative unate functions. In this way, as discussed in [MTMC14], the mix of NCL and NCL+ enables the use of negative unate functions, which allows further optimization opportunities and expands the design space of QDI circuits. For each positive unate NCL (NCL+) gate, a negative unate NCL (NCL+) gate exists, where the latter implements the complement of the function implemented by the former. To differentiate these gates from positive unate NCL

and NCL+ gates from Definitions 3 and 4, we call them inverted NCL and NCL+ gates (or INCL and INCL+, respectively). We can now define INCL and INCL+ gates.

Definition 5. *An* INCL gate *is an n-input logic gate with a threshold $T \in N^*$, a specific weight $w_i \in N^*$ assigned to each Boolean input x_i, and a hysteresis mechanism to ensure a sequential behavior, such that its output Q_i at each instant of time i is given by:*

$$Q_i = \begin{cases} 1, & \sum_{j=0}^{n-1} w_j x_j = 0 \\ 0, & \sum_{j=0}^{n-1} w_j x_j \geq T \\ Q_{i-1}, & 0 < \sum_{j=0}^{n-1} w_j x_j < T \end{cases} \tag{5.9}$$

Definition 6. *An INCL+ gate is an n-input logic gate with a threshold $T \in N^*$, a specific weight $w_i \in N^*$ assigned to each Boolean input x_i, and a hysteresis mechanism to ensure a sequential behavior, such that its output Q_i at each instant of time i is given by:*

$$Q_i = \begin{cases} 1, & \sum_{j=0}^{n-1} w_j \overline{x_j} \geq T \\ 0, & \sum_{j=0}^{n-1} w_j \overline{x_j} = 0 \\ Q_{i-1}, & 0 < \sum_{j=0}^{n-1} w_j \overline{x_j} < T \end{cases} \tag{5.10}$$

Note that, from a functional point of view, the only difference between an NCL (NCL+) and an INCL (INCL+) gate is that the ON- and OFF-sets are swapped. However they all still rely on a sequential behavior to ensure that QDI properties are respected. Furthermore, because INCL gates are basically negative unate versions of NCL gates, respectively, we will refer to them as NCL or INCL interchangeably. The same is valid for INCL+ and NCL+ gates. For the representation of these gates, as in conventional logic gates, the only difference in their symbol is the addition of a circle in the output of inverted gates. Using these inverted gates it is possible to convert signals from RTZ to RTO, and vice-versa. Because each time an inverted gate is used the protocol change and the code word for representing the spacer also change, circuits using (I)NCL and (I)NCL+ gates are called spatially-distributed dual-spacer NCL (or SDDS-NCL) [MTMC14].

Figure 5.29 shows an example of the SDDS-NCL construction of the NCL and NCL+ circuits showed in Figures 5.28(a) and 5.28(b), respectively. The same color codes were respected in the design. As the Figure shows, when the inputs of this circuit have spacers it will issue a spacer in its output. Accordingly, all the INCL gates will have a 1 in their outputs, which means that the INCL+ gates will have 1 in all their inputs and a 0 in their outputs. In other words, all blue wires will be at 0 and all red wires at 1. From this state,

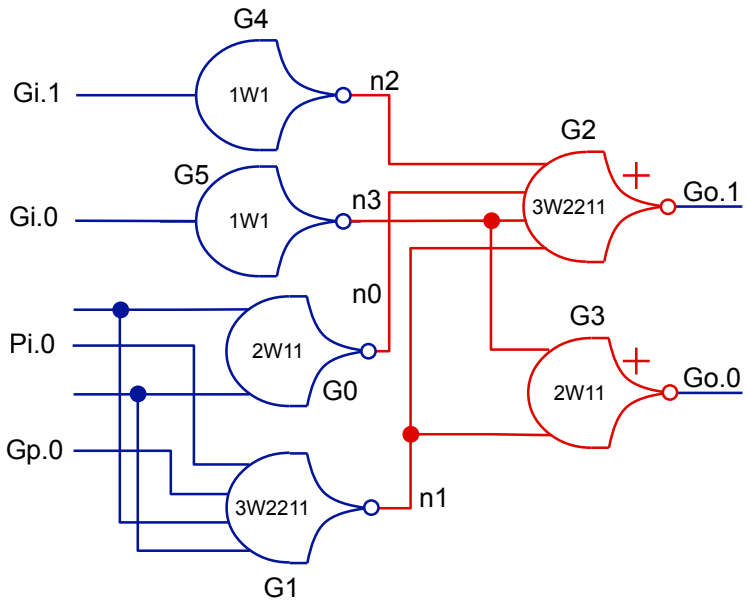


Figure 5.29 – Example of an SDDS-NCL implementation of the generate path for the YB yellow box of a Kogge-Stone adder. Adapted from [MTMC14].

whenever the inputs become valid a pair of INCL gates will fire, setting their outputs to 0, which will cause one of the INCL+ gates to fire, setting its output to 1.

Note that the pair of inverters *G4* and *G5* (equivalent to INCL 1W1 gates) are required to ensure that all the inputs of gates *G2* and *G3* are in the same domain (RTO). Also, because there are only two levels of logic and all gates are negative unate, the inputs and the outputs of this circuit are RTZ. If the output was to be RTO, inverters could be added after gates *G2* and *G3*. Moreover, different combinations of NCL, INCL, NCL+ and INCL+ gates can be explored, depending on the requirements for the circcuit input and output channels.

5.5.2 Proposed Design Flow

Having the possibility of using negative and positive unate cells for composing a circuit enables the use of commercial tools to perform logic and physical synthesis steps. These tools allow exploring new automation degrees in technology mapping and further logic optimizations. *Technology mapping* is the process of translating a generic logic network into a technology specific logic network. This is one of the most relevant steps performed during logic synthesis. Said otherwise, technology mapping is the task of transforming a netlist composed by generic logic functions into a netlist composed only by gates of a given technology library. However, conventional CAD tools cannot explicitly handle NCL and NCL+ gates, as these present a sequential behavior distinct of those expected by these tools. To overcome this problem and leverage the optimization and automation degrees allowed by

EDA tools, we propose a design flow for synthesizing circuits based on the SDDS-NCL template. Note that, albeit we employ the Cadence Framework in our work, the flow is generic and tools from other vendors can be used with adjustments in out scripts. Also, the flow currently supports combinational logic blocks only and the designer needs to split sequential from combinational blocks and synthesize the latter in isolation.

The flow is divided in two steps: logic and physical synthesis. Both steps are based in the usage of virtual libraries. Before defining a virtual library (VL), we first define a virtual function (VF), as they are the basis for the construction of VLs.

Definition 7. *A* virtual function VF *can be positive or negative:*

(a) Let θ be a function of some NCL gate, Θ be the set of functions for all possible NCL gates, ϕ be a function of some NCL+ gate, and Φ be the set of functions of all possible NCL+ gates. Let also ON(x) and OFF(x) be operators that return the ON-set and OFF-set of a function, respectively. A positive VF *is a Boolean function f such that:*

$$\exists\theta \in \Theta : ON(f) = ON(\theta) \tag{5.11}$$

and

$$\exists\phi \in \Phi : OFF(f) = OFF(\phi). \tag{5.12}$$

(b) Let ψ be a function of an INCL gate, Ψ be the set of functions for all possible INCL gates, υ be a function of some INCL+ gate and Υ be the set of functions of all possible INCL+ gates. A negative VF *is a Boolean function g such that:*

$$\exists\psi \in \Psi : OFF(g) = OFF(\psi) \tag{5.13}$$

and

$$\exists\upsilon \in \Upsilon : ON(g) = ON(\upsilon). \tag{5.14}$$

In this way, all positive VFs are positive unate functions, but not all positive unate functions are positive VFs. Furthermore, all negative VFs are negative unate functions, but not all negative unate functions are negative VFs. Hence, all VFs are unate functions, but not all unate functions are VFs.

Definition 8. *A* virtual library VL *is a library of cells such that their functions are modeled exclusively using VFs. In this way, it is guaranteed that synthesis tools will be able to handle a VL, as all VFs are unate functions. Two types of VLs exist, NCL VLs and NCL+ VLs.*

(a) An NCL VL *is a VL composed exclusively by NCL and INCL gates modeled using VFs.*

(b) An NCL+ VL *is a VL composed exclusively by NCL+ and INCL+ gates modeled using VFs.*

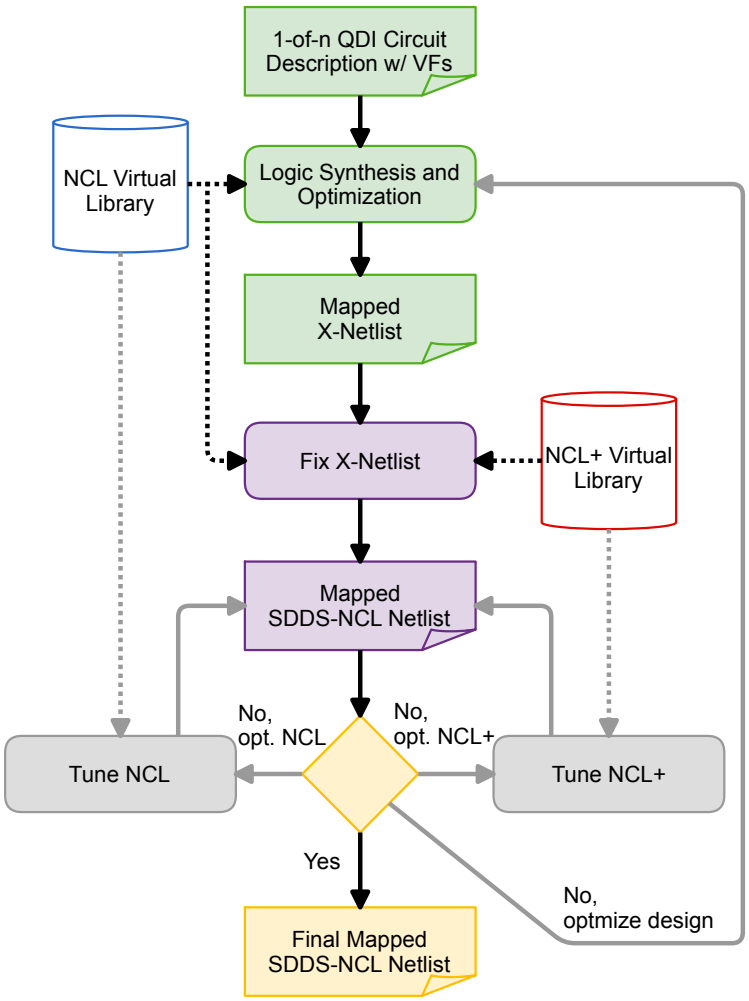


Figure 5.30 – Design flow for technology mapping and logic optimization of 1-of-n QDI circuits using SDDS-NCL.

Section 5.5.4 will detail the process of creating a VL. Figure 5.30 presents a diagram that details the logic synthesis flow for the SDDS-NCL template. The input of the flow is a description of an 1-of-n QDI circuit using VFs only. This description can be manually generated or be the output of an asynchronous synthesis tool, like the ones presented in [Bar98, Bar00, BDL11], with little modifications, like replacing NCL gates by their respective VFs. These VFs can be constructed using the *and* and *or* operators. Figure 5.31 for instance shows the Verilog description of a dual-rail half-adder (HA) using only VFs. This description is handled to a logic synthesis tool, in our case RTL Compiler, together with an NCL VL. Because the cells of such library are all modeled using VFs compatible with conventional libraries, the tool can recognize and use them. Note that at this point the designer must also define the constraints for guiding the tool in the process of mapping the circuit to the virtual NCL gates and optimizing it accordingly. These constraints can be defined using conventional *max_delay* SDC constraints.

During this step the synthesis tool performs elaboration and generic and mapped synthesis processes. Hence, at this point, the designer can already provide design con-

```
module HA (A.1, A.0, B.1, B.0, C.1, C.0, S.1, S.0);
  input  A.1, A.0, B.1, B.0;
  output C.1, C.0, S.1, S.0;
  wire   C.1, C.0, S.1, S.0;
  assign S.1=(A.1 & B.0)|(A.0 & B.1);
  assign S.0=(A.1 & B.1)|(A.0 & B.0);
  assign C.1=(A.1 & B.1);
  assign C.0=(A.1 & B.0)|(A.0 & B.0)|(A.0 & B.1);
endmodule
```

Figure 5.31 – Example Verilog description of a dual-rail half-adder using only VFs.

straints to the tool so it can optimize the design accordingly. The result of this step is a mapped netlist that employs the VFs defined in the NCL VL. Unfortunately, this netlist may have corrupted the correct functionality of the design, or even the principles of the employed DI channel flavor (as defined in Section 2.2.1). This is because at this point only NCL gates were available and, as discussed in Section 5.5.1, negative unate gates compromise the correct operation of a QDI circuit composed by NCL gates only. That is the reason why we call the resultant netlist an *X-Netlist*.

Figure 5.32 shows an example of an *X-Netlist*, resultant of the description showed in Figure 5.31. Also, the representation of this netlist using VFs only is showed in Figure 5.33. Note that for the sake of simplicity we assume the use of a minimum library, with only NAND and inverters as VFs, for the synthesis process in this example. To demonstrate how a correct functionality can be corrupted, assume that all primary inputs of the circuit showed in the Figure (*A.0*, *A.1*, *B.0* and *B.1*) are at 0 (spacer), making all its outputs (*C.0*, *C.1*, *S.0* and *S.1*) also go to 0. Now, assume that inputs *A.1* and *B.1* switch to 1, signaling a 1 value in inputs *A* and *B*. Accordingly, the NAND *G3* will switch the node *n2* to 0 and the inverter *G6* will switch *C.1* to 1, signaling a 1 value in the carry out signal (*C*). However *S.0* will not switch to 1, which denotes that the circuit functionality has been corrupted.

This behavior takes place because the VF of the gate that generates this output, a NAND, has an OFF-set that is not covered by its corresponding NCL gate, the actually mapped gate. This problem arises when inverted logic is used. In fact, if the designer provides inverters to the tool, the functionality of the netlist can be compromised. Classically, there was no way to guarantee the coverage of VFs ON-set of inverted functions using NCL gates, because such functions alter the domain from RTZ to RTO as defined in Equation 5.1. This is one of the major barriers that prevented designers from using conventional CAD tools to map QDI circuits. The method proposed here overcomes the problem.

The next step is to fix this netlist using an NCL+ VL. To do so, we rely on the definition of *inversion polarity*.

Definition 9. *The* inversion polarity *of a net is the number of inversions along the path that connects this net to the primary inputs. If all inputs of a gate (or cell) are even, it is said that*

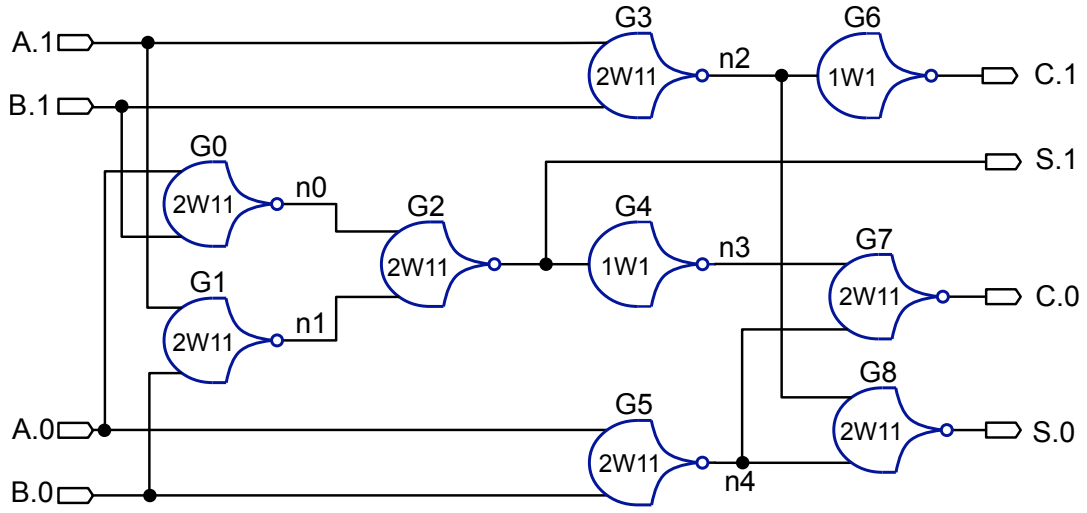


Figure 5.32 – X-Netlist resultant from the synthesis of the circuit described in Figure 5.31.

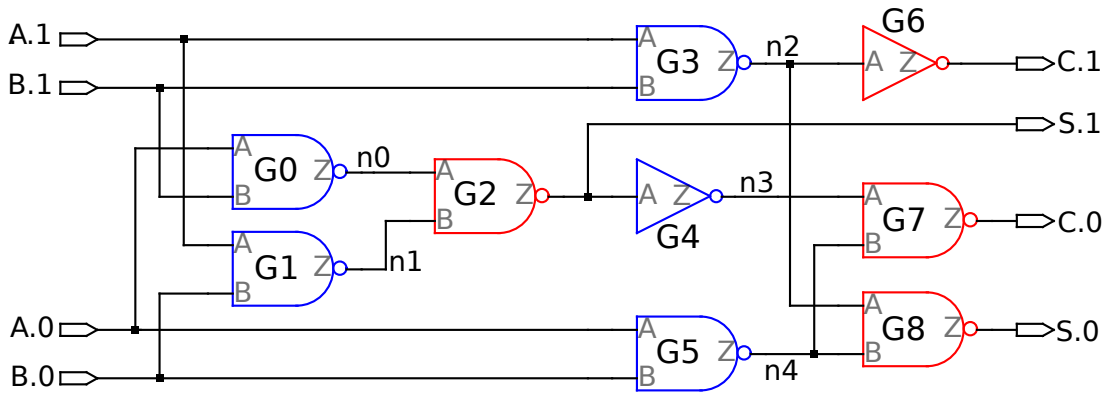


Figure 5.33 – VF representation of an X-Netlist resultant of the synthesis of the circuit described in Figure 5.31.

its inversion polarity is even. *Similarly, if all inputs of a gate (or cell) are even, it is said that its inversion polarity is* even.

Note that the NCL+ VL must contain exactly the same VFs and driving strengths of those available in the employed virtual NCL library. Using it, the simple algorithm depicted in Figure 5.34 can fix the *X-Netlist* and produce a mapped SDDS-NCL netlist that uses gates from both VLs. This algorithm ensures that the resultant netlist complies with QDI principles, while enabling the full potential of technology mapping and optimization capabilities of consolidated EDA tools. As the figure shows, the algorithm calculates if each cell of an *X-Netlist* must be NCL or NCL+. Note that whenever it detects that a cell should be an NCL cell, and the current cell is not NCL, it replaces the cell by an equivalent one from the NCL VL. This equivalent cell must have the same VF and the same driving strength of the replaced cell. The same is done whenever it detects that a cell should be an NCL+ cell, where it replaces the cell by an equivalent one in the NCL+ VL.

```
for each cell do
    if primary inputs are RTZ then
        if inversion polarity is even then
          cell is NCL
        else
          cell is NCL+
    else if primary inputs are RTO then
        if inversion polarity is even then
          cell is NCL+
        else
          cell is NCL
```

Figure 5.34 – Algorithm for fixing an X-Netlist.

In practice, this algorithm replaces NCL by NCL+ gates and INCL by INCL+ gates, or vice-versa, whenever it is necessary, in order to ensure that the SDDS-NCL arquitecture definition is respected, see Section 5.5.1. For example, in the circuit showed in Figure 5.32, gates *G0* and *G1* are connected to the primary inputs, which, according to the description showed in Figure 5.31, are RTZ. Hence, these gates must be NCL or INCL gates, in this case both must be INCL gates and the tool selects equivalent gates from the NCL VL. Gate *G2*, on the other hand, has an odd inversion polarity, and the primary inputs of the circuit are RTZ. In this way, this gate should be an NCL+ or an INCL+ gate. Hence, the tool must select an equivalent cell in the NCL+ VL. Using this algorithm, all cells of the example *X-Netlist* can be corrected. For instance, in Figure 5.33, all VFs that must be mapped using cells from the NCL VL are marked in blue and all the ones that must be replaced by cells from the NCL+ VL are marked in red.

As Figure 5.30 shows, after the algorithm replaces all the cells we have a Mapped SDDS-NCL Netlist. Figure 5.35 shows the fixed SDDS-NCL netlist for the example showed in Figures 5.33 and 5.32. As the Figure shows, all gates with an even inversion polarity belong to the NCL VL, because the primary inputs are RTZ encoded. Similarly, all gates with an odd inversion polarity are from the NCL+ VL. More specifically, *G0*, *G1*, *G3*, *G4* and *G5* have an even inversion polarity and, therefore, are mapped to the NCL VL. *G2*, *G6*, *G7* and *G8*, on the other hand, have an odd inversion polarity and are mapped to the NCL+ VL.

If we consider the same scenario used to show how correct functionality can be corrupted in an *X-Netlist*, we can observe that functionality is fixed now in the mapped SDDS-NCL netlist. For example, let us assume that all primary inputs of the circuit showed in Figure 5.35 (*A.0*, *A.1*, *B.0* and *B.1*) are at 0 (spacer), making all its outputs (*C.0*, *C.1*, *S.0* and *S.1*) also go to 0. Now, assume that inputs *A.1* and *B.1* switch to 1, signaling a 1 value in inputs *A* and *B*. Accordingly, *G3* will switch the node *n2* to 0 and *G6* will switch *C.1* to 1, signaling a 1 value in the carry out signal (*C*). Furthermore, *S.0* will now switch to 1,

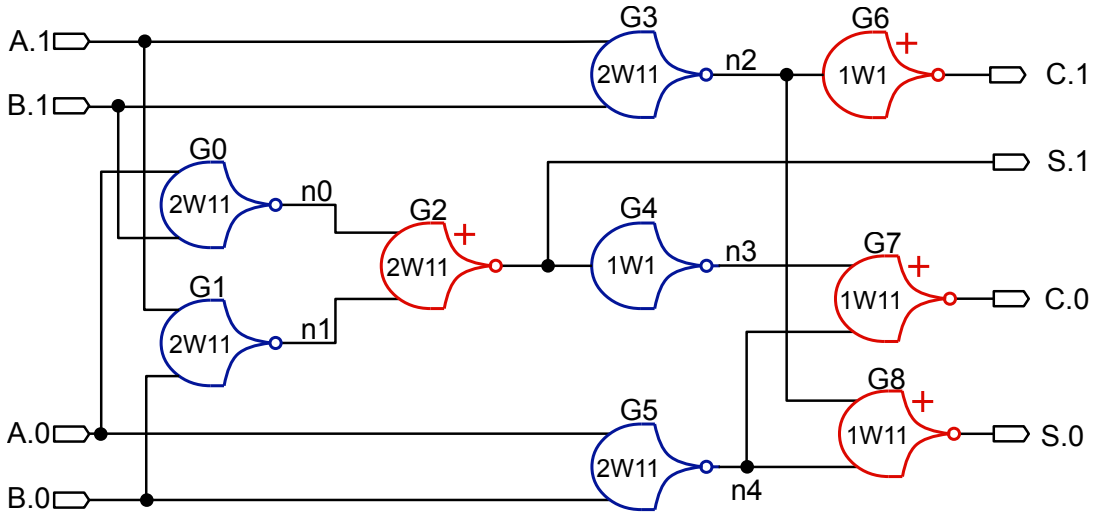


Figure 5.35 – Mapped SDDS-NCL Netlist equivalent to Figure 5.33.

given the functionality of *G8*. A similar scenario can be observed for all other possible input vectors.

The idea behind the fixing of an *X-Netlist* relies on the fact that valid data is always computed based on the number of inputs at 1 in NCL gates and based on the number of inputs at 0 in NCL+ gates. Hence, whenever the synthesis tool uses negative VFs, it inverts how valid data is represented (see Equation 5.1) and to fix that we can replace an NCL gate by its equivalent NCL+ or vice-versa. Furthermore, spacers are always identified by all inputs at 0 in RTZ and all inputs at 1 in RTO and synchronization of all-0s and all-1s is guaranteed by NCL and NCL+ gates, respectively. Hence, this fix in the *X-Netlist* ensures the correct propagation of spacers as well. Also, note that, as Figure 5.30 shows, we assume that the flow starts using the NCL VL and then the netlist is fixed using both NCL and NCL+ VLs. Depending on design specific parameters, a designer can chose to start the synthesis process using an NCL+ VL. This choice is useful for instance when the input description assumes RTO inputs.

As Figure 5.30 shows, once the netlist is fixed, the design characteristics are verified and if they meet the requirements, the designer can proceed to the next step, physical synthesis. If the design does not meet the specified requirements, for example if it does not meet the defined *max_delay* constraints, it can be optimized, by reiterating the logic synthesis flow, or at least parts of it. To do such verification, an STA tool is employed. In fact, this is one of the major advantages of the proposed flow, as these tools allow a precise, fast and automated assessment of the characteristics of a design. At this stage, if the design needs to be optimized the designer can perform three different tasks: (i) optimize the NCL cells; (ii) optimize the NCL+ cells; or (iii) optimize the whole design.

For option (i), the synthesis tool is configured not to touch cells belonging to the NCL+ VL and executes resynthesis and optimizations in the design. Because the NCL+

cells are defined as "*don't touch*" the tool has less freedom and will mainly resize the NCL cells. In rare cases it can optimize and resynthesize logic arrangements, due to the tightly constrained scenario, where it can only modify NCL cells. For option (ii) a similar optimization strategy is employed. However the cells set as "*don't touch*" are those belonging to the NCL VL and the tool will optimize the design changing only NCL+ cells. Option (iii) consists of a complete resynthesis of the design using the whole flow. In this option, the synthesis tool has more freedom to optimize the design, because it can modify all cells, NCL and NCL+ ones. However, in this case, the designer needs to apply the fix netlist algorithm at the end of each resynthesis. The designer can iterate the synthesis flow as many times as required until constraints are met. At each iteration, traditional synthesis parameters can be modified, *e.g.* timing constraints can be adjusted or new cells can be added to the libraries. Once the constraints are met, the designer can proceed to the next stage of physical synthesis.

For the physical synthesis flow, as Figure 5.36 shows, the first step is to generate the layout without modifying the SDDS-NCL Netlist. To do so, the designer performs floor and power planning steps and place and route tasks, as usual. The difference here is that the physical NCL and NCL+ libraries are used, instead of VLs. These libraries have the same cells available in the virtual libraries but described at the layout level. Note that at this stage specific constraints like matching delays in isochronic forks must be dealt with using conventional clock libraries, which provide delay cells. The same libraries can be used for constructing clock trees to distribute control signals in the registers of the circuit. Also, at this stage, the designer can explore optimizations at the layout level, like clustering registers and rearranging the spatial distribution of logic blocks. However, such optimizations are out of the scope of this Thesis.

The physical synthesis tasks so far accounted only for layout generation and optimization. At this stage, the NCL and NCL+ cells of the circuit are the same of the output circuit from the logic synthesis. In fact, the only cells that can be added during layout generation and optimization are delay cells for timing closure. As Figure 5.36, this gives us an SDDS-NCL circuit layout. If this layout meets the specified constraints, it can proceed to layout verification and sign-off tasks. If it needs optimization, though, there are three possible paths to follow. As in logic synthesis, the designer can chose to: (i) optimize the NCL cells (*Tune NCL*); (ii) optimize the NCL+ cells (*Tune NCL+*); or (iii) optimize the whole design (*Design Optimization*).

For options (i) and (ii), the only possibilities are to isolatedly optimize NCL and NCL+ cells, respectively. This can be done by defining NCL+ or NCL cells, respectively, as "*don't touch*" and allowing the tool to perform design optimizations. This is very similar to optimizations (i) and (ii) in the logic synthesis and the tool will mostly resize NCL and NCL+ cells, respectively. However, note that after such optimizations, the tool will perform new place and route tasks, as cells can have their dimensions altered. In this way, new timing closure verifications must be performed. Option (iii) allows higher degrees of freedom in the

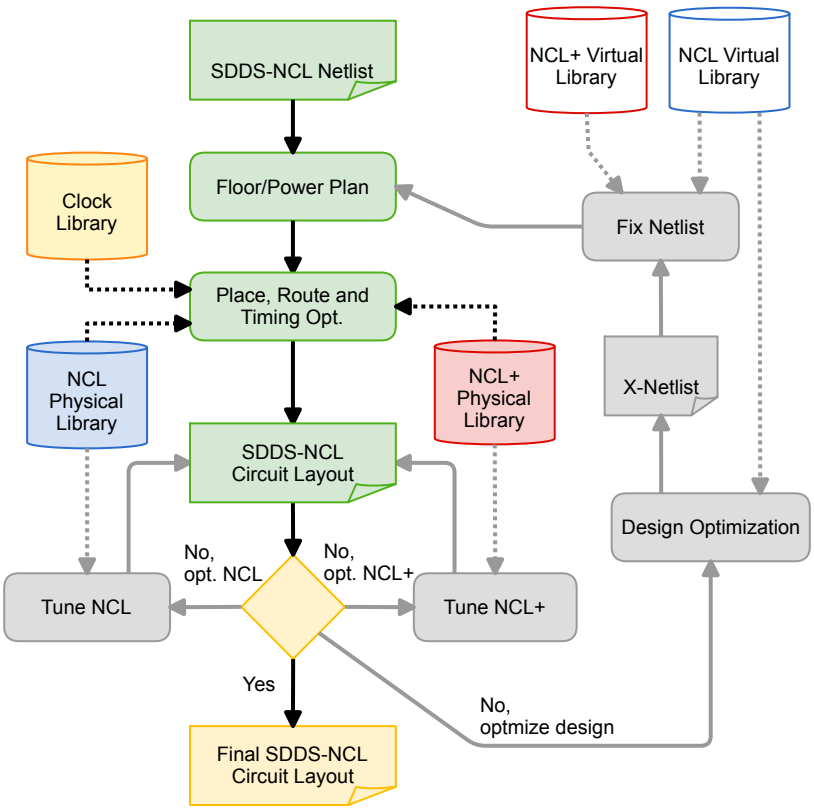


Figure 5.36 – Design flow for physical synthesis of 1-of-n QDI circuits using SDDS-NCL.

optimization of the circuit. Here, the designer can modify timing constraints and trigger the synthesis tool to optimize the whole design. In fact, this option allows the tool to completely resynthesize the design, modifying any cell. To do so, the synthesis tool employs an NCL VL for performing synthesis and optimizations, according to the newly specified constraints. As in the logic synthesis flow, the outcome of synthesizing such a design using an NCL VL only is an *X-Netlist*. This netlist can then be fixed using the same algorithm described in Figure 5.34.

After fixing the *X-Netlist*, the design may have been significantly changed, because at the design optimization phase the tool had all the liberty to modify the cells of the design. In this way, the next step is to redo the layout generation steps. Note that in some cases, a new floor and power plan can be required. Once the new SDDS-NCL circuit layout is generated, the designer can verify if it meets the specified constraints. If it does not, the optimization must be repeated. This process can iterate until the design meets the specifications and can proceed to layout verification and sign-off tasks.

5.5.3 Analysis of the Algorithm for Fixing *X-Netlists*

From the physical and electrical points of view, replacing an NCL cell for an NCL+ one with an equivalent VF can change timing and power properties. That is why we constrain this substitution to a cell with similar driving strength. By doing so, discrepancies on power and delay are mitigated because the transistors of cells with the same driving strength were designed to charge/discharge the same load at the same speed. From the functional point of view, however, this algorithm is only consistent if all the inputs of a same cell have the same inversion polarity. Because in our design flow the input description always implements unate functions all internal nodes will be unate in the primary inputs. Hence, in the synthesized netlist, the inversion polarity of all inputs of a given node will be the same, provided that no binate logic is used by the tool for composing the netlist.

For example, consider gate *G7* in Figure 5.35. Its inputs *n3* and *n4* have different numbers of logic levels from the primary inputs of the circuit. In this way, one can argue that *n3* and *n4* could have a different inversion polarity and compromise correct functionality of the circuit. However, because all NCL and NCL+ gates are unate, the only way for this assumption to be true is if the synthesis tool performs a binate realization of the unate input description using unate gates, which is possible according to Das et al., as described in the work presented in [DCB93]. If it does happen, we have a method that can safely guarantee the same inversion polarity for all the inputs of all gates in a netlist produced by our flow. To do so, we rely on design for testability (DFT) techniques.

Before proving the correctness of this method we need some definitions.

Definition 10. *A* positive unate gate (PUG) *is a circuit designed at the cell level that implements a positive unate function.*

Definition 11. *A* negative unate gate (NUG) *is a circuit designed at the cell level that implements a positive unate function.*

Definition 12. *A* dual polarity circuit (DPC) *is a combinational logic circuit composed by PUGs and NUGs only. In this way, all SDDS-NCL circuits are DPCs, because all NCL and NCL+ cells are either PUGs or NUGs.*

We also rely on a set of definitions proposed by Devadas and Keutzer [DK92b].

Definition 13. *A* path *in a combinational circuit is defined as an alternating sequence of vertices and edges* $(g_0, e_0, ..., g_n, e_n, g_{n+1})$*, where edge* e_i*,* $1 \leq i \leq n$*, connects the output of vertex gi to an input of vertex* g_{i+1}*. For all i such that* $1 \leq i \leq n$*,* g_i *is a gate. Vertex* g_0 *is a primary input,* g_{n+1} *is a primary output and each* e_i *is a net.*

Definition 14. *An* event *is a transition (low-to-high or high-to-low) at the output of a gate. Consider a sequence of events* $\{r_0, r_1, ..., r_n\}$ *occurring at the outputs of gates* $\{g_0, g_1, ..., g_n\}$

along a path, such that, for all i such that $1 \leq i \leq n$, r_i *occurs as a result of event* r_{i-1}*. Then, event* r_0 *is said to* propagate *along the path. If there exists a vector pair such that under appropriate delay assumptions in the circuit an event could propagate along a path, then the path is said to be* event sensitizable*. If there exists an input vector pair such that under arbitrary delays in the circuit an event propagates along a path, then the path is said to be* single event sensitizable*.*

Definition 15. *A* controlling value *of a gate type is the value that determines the output state of the gate independent of the other inputs. A* non-controlling value *of a gate input is the value which is not a controlling value for the gate.*

Considering an AND gate, 0 is a controlling value, as it sets the output of the gate to 0 regardless of the other input values. For the AND example, 1 is a non-controlling value. Hence, we say a gate *g* has a controlled value if one of its inputs has a controlling value. Otherwise, we say *g* has a non-controlled value.

Definition 16. *Let* $\pi = \{g_0, e_0, ..., g_n, e_n, g_{n+1}\}$ *be a path in a circuit C. The inputs of* g_i *other than* e_{i-1} *are referred to as the* side inputs *to* π*. An input vector w* sensitizes *path* π *in C to 1 iff the value of* g_{n+1} *is 1 and, for all* g_i*, with* $1 \leq i \leq n+1$*, if* g_i *has a controlled value then the edge* e_{i-1} *presents a controlling value. Similarly, an input vector w* sensitizes *path* π *in C to 0 iff the value of* g_{n+1} *is 0 and, for all* g_i*, with* $1 \leq i \leq n+1$*, if* g_i *has a controlled value then the edge* e_{i-1} *presents a controlling value. Hence, for a path* π *in a circuit C to be single event sensitizable (to 1) it is necessary that there exists a vector pair* $< w_1, w_2 >$ *such that* w_1 *sensitizes to 0 path* π *and* w_2 *sensitizes to 1 path* π*. Furthermore, all side-inputs along* π *must have (the same) non-controlling value for both* w_1 *and* w_2*, and at each gate* g_i *along* π *if the value of* g_i *on* w_1 *is the non-controlled value then the value of* g_i *on* w_2 *is the controlled value. Changes on the side-inputs imply that the path is not single event sensitized, i.e. event sensitized under arbitrary delays.*

Definition 17. *A circuit has a* path-delay-fault *iff there exists a path from a primary input to a primary output via a set of gates and nets such that a primary input event is slow to propagate along the path to the primary output. A* robust test *for a path-delay-fault on a path* π *is a test which single event sensitizes* π*. Therefore, such test is valid under arbitrary delays and is not invalidated by glitches, hazards or races.*

Similar definitions have been proposed by Pramanick and Reddy in [PR90] for restricted delay test pairs and by Savir and McAnney in [SM88] for single-path propagation hazard-free robust tests.

With these definitions, it is possible now to define a robustly path delay fault testable (RPDFT) DPC:

Definition 18. *An* RPDFT DPC *is a DPC such that all its paths have a robust test for path-delay-faults.*

Albeit robustly path delay fault testability is a stringent requirement for a DPC, it is only applied to circuits where the proposed algorithm for fixing *X-Netlists* fails, which in our practical experience is not usual. In fact, out of 14 benchmark circuits, all of them were successfully synthesized using the proposed design flow. Furthermore, there are techniques in the state-of-the-art that allows the implementation of RPDFT DPCs. For example, Devadas and Keutzer show in [DK92a] a set of combinational synthesis for testability approaches that lead to implementations of ICs that are completely RPDFT, *i.e.* all its paths are RPDFT. The authors demonstrate the proposed approach in complex designs that include a microprocessor and a data encryption core. Similar works are presented in [KLSV95, KM95, PK93, Fuc95, EMR95], where the authors demonstrate techniques that guarantee completely RPDFT IC implementation. Moreover, according to these articles, ensuring RPDFT implementation does not compromise the performance of an IC and incurs in slight overheads in area. More recently, Mitra *et al.* [MDB14] revisited completely RPDFT implementation issues based on new functional properties. The authors report experimental results indicating that not only their designs can achieve completely RPDFT, but that the overheads for ensuring so are smaller for contemporary designs.

We now provide two theorems that define sufficient conditions for fulfilling the assumption of the proposed algorithm for fixing *X-Netlists*, that all inputs of the gates that compose such a netlist have the same inversion polarity. Note that, since our SDDS-NCL circuits can be either negative or positive unate, as discussed in Section 5.5.1, we rely on two theorems.

Theorem 1. *Let g be the output gate of a positive unate RPDFT DPC, which primary inputs are* $x_0, x_1, ..., x_{n-1}$*, that implements the function* $f(x_0, x_1, ..., x_{n-1})$*, like the example showed in Figure 5.37. According to Definition 12, gate g can be either a PUG or a NUG:*

(a) If g is a PUG, then all its inputs are positive unate in the primary inputs, i.e. have an even inversion polarity.

(b) If g is a NUG, then all its inputs are negative unate in the primary inputs, i.e. have an odd inversion polarity.

Proof. (a) Without loss of generality let g be a positive unate gate which inputs are the nodes $y_0, y_1, ..., y_m$, and all of them, but y_0, are positive unate in the primary inputs. This means that it exists at least one combination of primary inputs values such that: $y_0(x_0, x_1, ..., 1, ..., x_{n-1}) < y_0(x_0, x_1, ..., 0, ..., x_{n-1})$. In other words, there is a set of values such that a high-to-low transition in a given primary input will cause a low-to-high transition in y_0. According to Definition 17, because the DPC is RPDFT, such transition cannot be masked by a controlling value in one of the remaining inputs of g ($y_1, ..., y_m$) and will cause a low-to-high transition in its output f. If such a transition takes place, it means that there exists at least one combination of primary input values such that

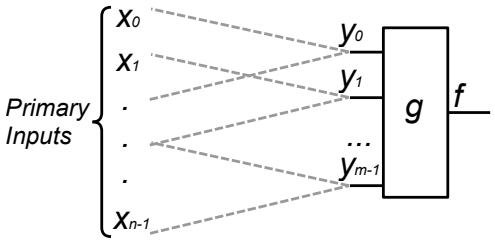


Figure 5.37 – Example combinational logic circuit that implements $f(x_0, x_1, ..., x_n) = g(y_0, y_1, ..., y_m)$.

$f(x_0, x_1, ..., 1, ..., x_{n-1}) < f(x_0, x_1, ..., 0, ..., x_{n-1})$. In other words, there is a set of values such that a high-to-low transition in a given primary input will cause a low-to-high transition in f, which cannot happen because we know that f is positive unate in the primary inputs. In this way we have proved by contradiction that $y_0, ..., y_m$ must be positive unate in the primary inputs.

(b) Without loss of generality let g be a negative unate gate which inputs are the nodes $y_0, y_1, ..., y_m$, and all of them, but y_0, are negative unate in the primary inputs. This means that it exists at least one combination of primary inputs values such that: $y_0(x_0, x_1, ..., 0, ..., x_{n-1}) < y_0(x_0, x_1, ..., 1, ..., x_{n-1})$. In other words, there is a set of values such that a low-to-high transition in a given primary input will cause a low-to-high transition in y_0. As defined by Definition 17, because the DPC is RPDFT, such transition cannot be masked by a controlling value in one of the remaining inputs of g ($y_1, ..., y_m$) and will cause a high-to-low transition in its output f. If such a transition takes place, it means that there exists at least one combination of primary input values such that $f(x_0, x_1, ..., 1, ..., x_{n-1}) < f(x_0, x_1, ..., 0, ..., x_{n-1})$. In other words, there is a set of values such that a low-to-high transition in a given primary input will cause a high-to-low transition in f, which cannot happen because we know that f is positive unate in the primary inputs. In this way we have proved by contradiction that $y_0, ..., y_m$ must be positive unate in the primary inputs.

□

Theorem 2. *Let g be the output gate of a negative unate RPDFT DPC, which primary inputs are $x_0, x_1, ..., x_{n-1}$, that implements the function $f(x_0, x_1, ..., x_{n-1})$, like the example showed in Figure 5.37. According to Definition 12, gate g can be either a PUG or a NUG:*

(a) If g is a PUG, then all its inputs are negative unate in the primary inputs, i.e. have an odd inversion polarity.

(b) If g is a NUG, then all its inputs are positive unate in the primary inputs, i.e. have an even inversion polarity.

Proof. (a) Without loss of generality let g be a positive unate gate which inputs are the nodes $y_0, y_1, ..., y_m$, and all of them, but y_0, are negative unate in the primary inputs. This means that it exists at least one combination of primary inputs values such that $y_0(x_0, x_1, ..., 0, ..., x_{n-1}) < y_0(x_0, x_1, ..., 1, ..., x_{n-1})$. In other words, there is a set of values such that a low-to-high transition in a given primary input will cause a low-to-high transition in y_0. As defined by Definition 17, because the DPC is RPDFT, such transition cannot be masked by a controlling value in one of the remaining inputs of g ($y_1, ..., y_m$) and will cause a low-to-high transition in its output f. If such a transition takes place, it means that there exists at least one combination of primary input values such that $f(x_0, x_1, ..., 0, ..., x_{n-1}) < f(x_0, x_1, ..., 1, ..., x_{n-1})$. In other words, there is a set of values such that a low-to-high transition in a given primary input will cause a low-to-high transition in f, which cannot happen because we know that f is negative unate in the primary inputs. In this way we have proved by contradiction that $y_0, ..., y_m$ must be negative unate in the primary inputs.

(b) Without loss of generality let g be a negative unate gate which inputs are the nodes $y_0, y_1, ..., y_m$, and all of them, but y_0, are positive unate in the primary inputs. This means that it exists at least one combination of primary inputs values such that: $y_0(x_0, x_1, ..., 1, ..., x_{n-1}) < y_0(x_0, x_1, ..., 0, ..., x_{n-1})$. In other words, there is a set of values such that a high-to-low transition in a given primary input will cause a low-to-high transition in y_0. As defined by Definition 17, because the DPC is RPDFT, such transition cannot be masked by a controlling value in one of the remaining inputs of g ($y_1, ..., y_m$) and will cause a high-to-low transition in its output f. If such a transition takes place, it means that there exists at least one combination of primary input values such that $f(x_0, x_1, ..., 0, ..., x_{n-1}) < f(x_0, x_1, ..., 1, ..., x_{n-1})$. In other words, there is a set of values such that a high-to-low transition in a given primary input will cause a high-to-low transition in f, which cannot happen because we know that f is negative unate in the primary inputs. In this way we have proved by contradiction that $y_0, ..., y_m$ must be positive unate in the primary inputs.

□

5.5.4 Virtual Libraries Design

The flow presented in Section 5.5.2 assumes the availability of NCL and NCL+ VLs. However, the physical components of such libraries are not available off the shelf in conventional cell libraries. In this way, we present here a design flow for generating such libraries using the ASCEnD-A flow. Figure 5.38 shows the proposed flow. As the Figure shows, it starts with a positive unate function and branches in two directions: one for generating the

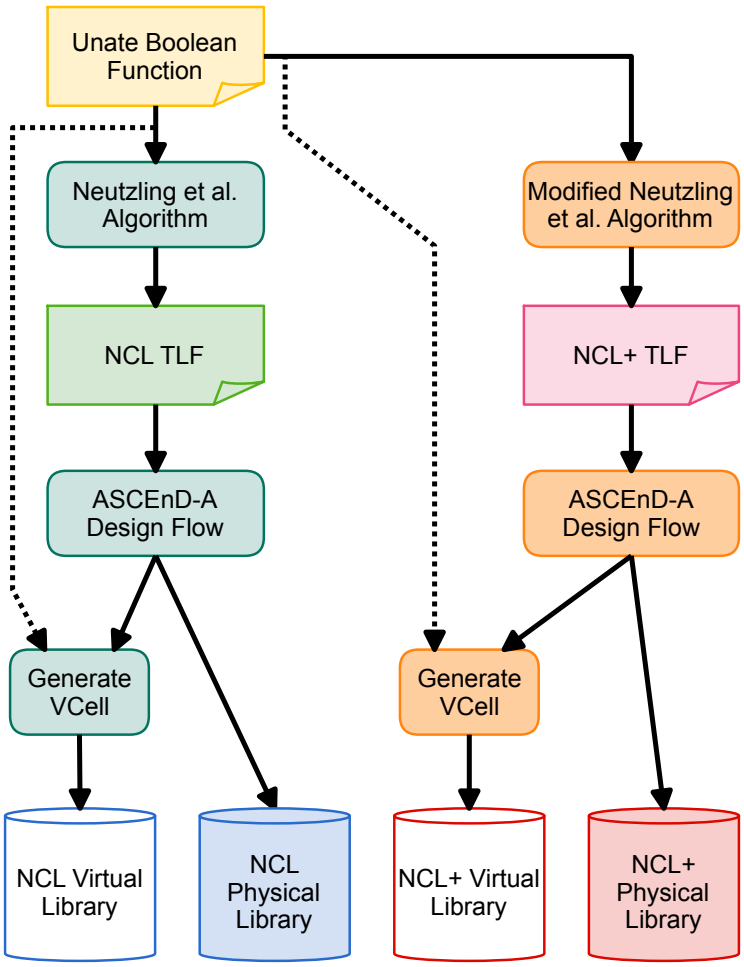


Figure 5.38 – Design flow for generating virtual libraries.

NCL virtual and physical libraries and another for generating the NCL+ virtual and physical libraries.

Before detailing the library generation process, it is necessary to differentiate TLFs from the Boolean functions that implement it.

Definition 19. *A* Boolean virtual function (BVF) *is a completely specified function f that implements a TLF with:*

$$f : \{0,1\}^n \rightarrow \{0,1\} \tag{5.15}$$

Now, for the NCL libraries generation we use the defined positive unate function (a BVF) to feed the method proposed by Neutzling *et al.* in [NMRR13], which allows identifying an equivalent TLF, recall Section 3.1.3. The algorithm is based on the generation of inequalities and Chow parameters from the truth table of an input unate function to assign the threshold and the weight of the variables of a corresponding TLF. To explain how this works, assume the example BVF used in [MNM+14]: $Q = A(B + C)$. Table 5.9 shows the associated truth table. As the Table shows, inequalities are computed from the relationship between input weights and the gate threshold to a TLF, as given by Equation (3.1) (from Definition 2).

Table 5.9 – Input weights and threshold relationship to Q=A(B+C) [MNM+14].

A	**B**	**C**	**Q**	
0	0	0	0	$0 < T$
0	0	1	0	$w_C < T$
0	1	0	0	$w_B < T$
0	1	1	0	$w_B + w_C < T$
1	0	0	0	$w_A < T$
1	0	1	1	$w_A + w_C \geq T$
1	1	0	1	$w_A + w_B \geq T$
1	1	1	1	$w_A + w_B + w_C \geq T$

Next, input weights and the threshold value can be computed from these inequalities. The relationship between input weights and the gate threshold of a TLF for BVF $Q = A(B + C)$ appears in the rightmost column of Table 5.9. Each sum of input weights greater than the threshold belongs to the *greater side set* of inequalities and each sum of weights which is less than the threshold belongs to the *lower side set*. Table 5.10 shows the sets for the function in Table 5.9.

Since greater side set elements are greater than the threshold value, and lower side set elements are smaller than such threshold, each greater side element is greater than each lower side element. Accordingly, the inequalities system is generated by performing the Cartesian product of the greater side set and the lower side set, as showed in Table 5.11. Some relationships in Table 5.10 are not useful. For instance, since we have the relation $w_A + w_B \geq T$ and the input weights are always positive, the relation $w_A + w_B + w_C \geq T$ is redundant, since it is contained in the former. In fact, for the example $Q = A(B + C)$ the algorithm creates only the inequalities 4, 5, 9 and 10 from Table 5.11. Moreover, besides generating only irredundant inequalities, the algorithm simplifies each of them when possible. The simplification occurs when the variable weight appears on both sides of the same inequality. When this happens, the variable is removed from such inequality. Consider for instance inequality 4. It is simplified removing C, resulting on a new inequality $w_A > w_B$. Since all input weights are positive, the algorithm also discards the inequalities having weight 0 in the smaller side as these inequalities are trivially satisfied.

After simplification, input weights are computed by selecting inequalities with only one input in the greater side. A single input in the greater side denotes that this input must have larger weight than all inputs in the lower side. From the example, $Q = A(B + C)$, the selected inequalities are only $w_A > w_B$ and $w_A > w_C$. Therefore, the weight from the greater side (input A in both cases) is defined as the key of the inequality. Next, a temporary variable is created, which controls the value assigned to weights. Such temporary variable is initialized with a minimum value (being initially set to 1) and can be incremented (+1)

Table 5.10 – Greater and smaller sets of inequalities from Table 5.9.

greater side		**lower side**
$w_A + w_C$	$> T >$	0
$w_A + w_B$	$> T >$	w_C
$w_A + w_B + w_C$	$> T >$	w_B
	$T >$	$w_B + w_C$
	$T >$	w_A

Table 5.11 – Generated inequalities from Table 5.10 [MNM+14].

Cartesian product: greater side $\times$ lower side			
1	$w_A + w_C > 0$	**9**	$w_A + w_B > w_B + w_C$
2	$w_A + w_C > w_C$	**10**	$w_A + w_B > w_A$
3	$w_A + w_C > w_B$	**11**	$w_A + w_B + w_C > 0$
4	$w_A + w_C > w_B + w_C$	**12**	$w_A + w_B + w_C > w_C$
5	$w_A + w_C > w_A$	**13**	$w_A + w_B + w_C > w_B$
6	$w_A + w_B > 0$	**14**	$w_A + w_B + w_C > w_B + w_C$
7	$w_A + w_B > w_C$	**15**	$w_A + w_B + w_C > w_A$
8	$w_A + w_B > w_B$		

during iterations. The order of weight assignment starts from variables that have the lowest weight, and continues in ascending order. Each time a weight is assigned, the consistency of the corresponding inequalities is checked. If the current value of the temporary variable satisfies the inequalities, this value is indeed the weight of the variable. Otherwise, the value is increased to satisfy the inequalities or until an upper limit is reached. This procedure is repeated for each variable.

In the example $Q = A(B + C)$, the assignment is first done to the weights of inputs B and C. Since there are no inequalities constrains for these keys, value 1 is assigned. The temporary variable increases to 2. Then, the weight of A must be assigned. For this key there are two identical inequalities: $w_A > 1$. The method assigns the temporary value 2 and checks the inequalities. The inequalities are true since $2 > 1$. As all weights are already assigned, this step finishes. Therefore, the weights assigned for the BVF $Q = A(B + C)$ are $w_A = 2$, $w_B = 1$ and $w_C = 1$. According to the authors of [NMRR13], this bottom-up approach ensures that the weights assigned by the algorithm are always the minimum possible, allowing a gate implementation synthesizable with minimum circuit area. The threshold value is defined as the sum of weights of the smallest value of the greater side. For $Q = A(B+C)$, the calculated threshold is 3, from $w_A + w_B$ or $w_A + w_C$ (refer to Table 5.11). Accordingly, this defines a TLF such that $T = 3$ and $w_A = 2$, $w_B = 1$ and $w_C = 1$, which (see Definition 3) corresponds to an *NCL3W211* gate.

Table 5.12 – Equivalent NCL and NCL+ gates for VF Q=A(B+C). Adapted from [MNM+14].

A	B	C	Q_{VF}	Q_{NCL}	Q_{NCL+}
0	0	0	0	0	0
0	0	1	0	-	0
0	1	0	0	-	0
0	1	1	0	-	0
1	0	0	0	-	0
1	0	1	1	1	-
1	1	0	1	1	-
1	1	1	1	1	1

As Figure 5.38 shows, this TLF proceeds to the ASCEnD-A flow for creating an NCL cell at the layout level. After all models of the NCL cell are generated, they proceed to the physical library. Their Liberty models, though, go through the process of generating a VL. In this process, the cell has its functionality not represented by its TLF, but by the original unate function used for generating it. To do so, the original Liberty is modified.

As Table 5.12 shows, the selected NCL gate for the example function has the same ON-set (in red) of its equivalent VF. On the other hand, the OFF-set, in blue for the VF, is not the same. In fact, in NCL gates, the output will switch to 0 only when all inputs are at 0, as the table shows in green. As explained before, the ON-set of NCL gates signals valid data, which generates varying logic values in the inputs of such gates. Moreover, their OFF-set is used for synchronizing spacers, guaranteeing that there is no early spacer generation. This is in accordance to the proposed design flow to build an *X-Netlist*. However for fixing such netlist the flow still requires equivalent NCL+ virtual and physical libraries.

To cope with that, as Figure 5.38 shows, the flow also has steps for generating NCL+ physical and virtual libraries. The idea is similar to that followed for generating the NCL libraries, but it relies on a modified version of the algorithm proposed by Neutzling *et al.* The basic idea is that because the TLF in an NCL+ gate is computed for its OFF-set (see Definition 4), the inequalities are constructed searching for inputs that are at 0 when the output is set to 0. This means that for the example VF ($Q = A(B + C)$) we can obtain the greater and lower side sets of inequalities showed in Table 5.13.

Employing the algorithm leads to a TLF with $T = 2$ and weights w_A=2, w_B=1 and w_C=1. The equivalent NCL+ is the NCL+2W21. Accordingly, the truth table of this gate appears in Table 5.12. As it can be seen, this gate presents the same OFF-set of the VF. Moreover, it guarantees that the output will only switch to 1 when all inputs are at 1, allowing synchronization of RTO spacers.

VFs can also be negative unate logic functions and can be mapped to an equivalent INCL or INCL+ gate. However, in this case, the algorithm proposed by Neutzling *et al.* must

Table 5.13 – NCL+ greater and smaller sets of inequalities from Table 5.9 [MNM+14].

greater side		**lower side**
A+B+C	$> T >$	B
A+B	$> T >$	C
A+C	$> T >$	0
A	$> T$	
B+C	$> T$	

Table 5.14 – Equivalent NCL and NCL+ gates for a NAND VF. Adapted from [MNM+14].

A	B	Q_{VF}	Q_{NCL}	Q_{NCL+}
0	0	1	1	1
0	1	1	-	1
1	0	1	-	1
1	1	0	0	0

be applied searching for the OFF-set of the VF. Take for example a 2-input NAND ($Q = \overline{AB}$). By applying the algorithm, we find the threshold $T = 2$ and the weight of the inputs $w_A = 1$ and $w_B = 1$, which corresponds to an INCL2W11 gate. The truth table of the BVF and the NCL gate are shown in Table 5.14. As the Table shows, the gate also respects QDI definitions and will only switch its output to 1 when a spacer is detected (all inputs at 0) and the OFF-set is the same as that of the VF, ensuring that valid data will propagate correctly. Similarly, an INCL+ gate can be obtained for this function. The corresponding gate is an inverted NCL+1W11 ($T = 1$, weights $w_A = 1$ and $w_B = 1$). This covers the ON-set of the BVF, as Table 5.14 shows.

Using this flow, a complex library of NCL and NCL+ gates can be generated targeting positive and negative unate VFs. Moreover, with the ASCEnD-A flow, a designer can rapidly devise several different driving strength versions for each cell. This is important in optimization stages, as discussed in Section 5.5.2. It is important though that the designer must ensure that the same driving strength cells are available in NCL and NCL+ libraries. Buffers and inverters are also part of NCL and NCL+ libraries and can be reused from core libraries. During our work with the generation of virtual libraries, we devised a set of scripts that enabled us searching for Boolean functions that can be implemented as TLFs and, assuming these as VFs, look for their equivalent NCL and NCL+ gates. To do so we relied on the algorithms described in [NMRR13] and the flow presented in Figure 5.38. Accordingly, we list all VFs that can be constructed with up to 4 inputs in Tables 5.15 and 5.16.

Table 5.15 – Positive unate VFs and respective NCL/NCL+ gates.

#in	VF	TLF	
		NCL	NCL+
2	$A + B$	1W11	2W11
	$A \cdot B$	2W11	1W11
3	$(A \cdot (B + C)) + (B \cdot C)$	2W111	2W111
	$A \cdot (B + C)$	3W211	2W211
	$A + B + C$	1W111	3W111
	$A + (B \cdot C)$	2W211	3W211
	$A \cdot B \cdot C$	3W111	1W111
4	$(A \cdot (B + C + D)) + (B \cdot (C + D)) + (C \cdot D)$	2W1111	3W1111
	$(A \cdot (B + C + D)) + (B \cdot (C + D))$	3W2211	4W2211
	$(A \cdot (B + C + D)) + (B \cdot C \cdot D)$	3W2111	3W2111
	$(A \cdot (B + C + D)) + (B \cdot C)$	4W3221	5W3221
	$A \cdot (B + C + D)$	4W3111	3W3111
	$(A \cdot (B + C)) + (B \cdot (C + D))$	4W2321	5W2321
	$(A \cdot (B + C)) + (B \cdot C \cdot D)$	5W3221	4W3221
	$(A \cdot B) + (A \cdot C \cdot D) + (B \cdot C \cdot D)$	4W2211	3W2211
	$(A \cdot B) + (A \cdot C \cdot D)$	5W3211	3W3211
	$(A \cdot B \cdot (C + D)) + (A \cdot C \cdot D) + (B \cdot C \cdot D)$	3W1111	2W1111
	$(A \cdot B \cdot C) + (A \cdot B \cdot D) + (A \cdot C \cdot D)$	4W2111	2W2111
	$A \cdot B \cdot (C + D)$	5W2211	2W2211
	$A + (B \cdot (C + D)) + (C \cdot D)$	2W2111	4W2111
	$A + (B \cdot (C + D))$	3W3211	5W3211
	$A + B + C + D$	1W1111	4W1111
	$A + B + (C \cdot D)$	2W2211	5W2211
	$A + (B \cdot C \cdot D)$	3W3111	4W3111
	$A \cdot B \cdot C \cdot D$	4W1111	1W1111
	$(A \cdot B) + (A \cdot C) + (B \cdot D)$	Special0	Special0
	$(A \cdot B) + (C \cdot D)$	Special1	Special1
	$(A \cdot C) + (A \cdot D) + (B \cdot C) + (B \cdot D)$	Special2	Special2

5.5.5 Experiments

As discussed in the work published in [MNM+14], the first experiments with the proposed design flow for SDDS-NCL circuits comprised the following set of case study circuits:

Table 5.16 – Negative unate VFs and respective INCL/INCL+ gates.

#in	VF	TLF	
		INCL	INCL+
2	$\overline{A+B}$	1W11	2W11
	$\overline{A \cdot B}$	2W11	1W11
3	$\overline{(A \cdot (B+C)) + (B \cdot C)}$	2W111	2W111
	$\overline{A \cdot (B+C)}$	3W211	2W211
	$\overline{A+B+C}$	1W111	3W111
	$\overline{A+(B \cdot C)}$	2W211	3W211
	$\overline{A \cdot B \cdot C}$	3W111	1W111
4	$\overline{(A \cdot (B+C+D)) + (B \cdot (C+D)) + (C \cdot D)}$	2W1111	3W1111
	$\overline{(A \cdot (B+C+D)) + (B \cdot (C+D))}$	3W2211	4W2211
	$\overline{(A \cdot (B+C+D)) + (B \cdot C \cdot D)}$	3W2111	3W2111
	$\overline{(A \cdot (B+C+D)) + (B \cdot C)}$	4W3221	5W3221
	$\overline{A \cdot (B+C+D)}$	4W3111	3W3111
	$\overline{(A \cdot (B+C)) + (B \cdot (C+D))}$	4W2321	5W2321
	$\overline{(A \cdot (B+C)) + (B \cdot C \cdot D)}$	5W3221	4W3221
	$\overline{(A \cdot B) + (A \cdot C \cdot D) + (B \cdot C \cdot D)}$	4W2211	3W2211
	$\overline{(A \cdot B) + (A \cdot C \cdot D)}$	5W3211	3W3211
	$\overline{(A \cdot B \cdot (C+D)) + (A \cdot C \cdot D) + (B \cdot C \cdot D)}$	3W1111	2W1111
	$\overline{(A \cdot B \cdot C) + (A \cdot B \cdot D) + (A \cdot C \cdot D)}$	4W2111	2W2111
	$\overline{A \cdot B \cdot (C+D)}$	5W2211	2W2211
	$\overline{A+(B \cdot (C+D)) + (C \cdot D)}$	2W2111	4W2111
	$\overline{A+(B \cdot (C+D))}$	3W3211	5W3211
	$\overline{A+B+C+D}$	1W1111	4W1111
	$\overline{A+B+(C \cdot D)}$	2W2211	5W2211
	$\overline{A+(B \cdot C \cdot D)}$	3W3111	4W3111
	$\overline{A \cdot B \cdot C \cdot D}$	4W1111	1W1111
	$\overline{(A \cdot B) + (A \cdot C) + (B \cdot D)}$	Special0	Special0
	$\overline{(A \cdot B) + (C \cdot D)}$	Special1	Special1
	$\overline{(A \cdot C) + (A \cdot D) + (B \cdot C) + (B \cdot D)}$	Special2	Special2

(i) an 8-bit ripple carry adder (8bRC); (ii) an 8-bit Kogge Stone adder (8bKS); (iii) a 32-bit Kogge Stone adder (32bKS); (iv) a 32-bit Arithmetic Logic Unit (32bALU); and (v) a 16-bit shift and add multiplier (16bMUL). The choice for such case studies was due to the fact that they present different gate counts and, more importantly, different numbers of gates in series in the critical path, which allows displaying the functionality of the proposed design flow for

different logic depths. Synthesis targeted the STMicroelectronics 65nm Bulk CMOS technology and all models were based on typical corners. For simplicity, apart from inverters with tens of different driving strengths, we employed a minimum VL, containing only the NAND VF. NCL and NCL+ VLs contain gates with six different driving strengths, to allow for timing optimizations, and are a subset of the ASCEnD-ST65 library. A library containing only the NAND VF is useful for a first set of experiments as it will generate long logic paths, stressing the proposed design flow.

After synthesis, we exported the netlist and the annotated delay to perform timing simulation. Simulation allowed us to evaluate the correctness of the design and to conduct performance measurements in terms of forward propagation delay, the time it takes for valid data to propagate from the inputs to the outputs, and transmission delay, the time it takes for both data and spacer to propagate through the circuit. Simulations were performed for 2 scenarios, both simulated during 1 ms: (i) the circuit was filled of spacers and there was no data injection; and (ii) the circuit was operating with random data being injected 100% of the time. During simulation, internal signals activity was annotated and used as the source to conduct static power analysis. This (i) allowed us to measure idle, or quiescent, power and (ii) provided results for the dynamic, or fully active, power consumption by the case studies.

Table 5.17 provides an overview of the obtained results for the synthesis of the case studies considering relaxed and strict timing constraints. These constraints were defined using a standard constraint file in the design flow defining the maximum latency of circuits using *max_delay* attributes. Note that strict constraints were defined iteratively, to achieve the maximum speed possible with the provided VLs. Relaxed constraints were defined as the maximum latency that enabled achieving 0 slack during synthesis. In other words, it guarantees that the employed cells are not underutilized.

As the Table shows, case studies presented different gate counts: from 247 to 18,371 for relaxed constraints. In all cases, more than 40% were NCL gates and less than 30% NCL+ gates. The remaining gates are inverters required by the logic. Note that the increased percentage of inverters in the design is a consequence of providing only NAND-based VFs. As constraints are made stricter, gate counts increased up to 23% and the percentage of NCL and NCL+ gates decayed, a consequence of the logical optimizations that the tool performed and the possible insertion of extra inverters. The Table also shows the number of gates in the critical path (*Crit. Path*), which varied from 24 to 432 for the relaxed timing designs. Note that as timing constraints are made stricter, substantial reductions on the critical path occur, up to 50% in the case of *16bMUL*. This demonstrates that the design flow enables the synthesis tool to perform logic optimizations in asynchronous designs according to specified constraints.

A direct consequence of that is observed in the transmission delay of the case studies. Accordingly, as we make timing constraints strict, this figure reduces substantially, up to 65%. Another interesting result was that the forward delay is always around 50% of the

Table 5.17 – Case studies synthesis, simulation and power analysis results. Adapted from [MNM+14].

		8bRC		8bKS		32bKS		32bALU		16bMUL	
		Relaxed	*Fast*	*Relaxed*	*Fast*	*Relaxed*	*Fast*	*Relaxed*	*Fast*	*Relaxed*	*Fast*
Gates	*Total*	247	282	422	521	2942	3446	4924	5141	18371	21041
	% NCL	41	35	49	41	48	42	45	40	42	39
	% NCL+	27	28	27	24	26	22	23	22	25	22
	% INV	32	37	24	35	26	36	32	38	33	39
	Crit. Path	60	48	24	18	36	26	44	34	432	216
Latency (ns)	*Forward*	3.90	1.41	2.33	1.55	2.74	1.96	1.23	0.84	27.02	20.53
	Transmission	7.71	2.72	4.41	3.07	5.13	3.84	2.27	1.56	53.68	40.73
Power (mW)	*Idle*	0.014	0.021	0.030	0.041	0.124	0.249	0.334	0.371	1.188	1.529
	Fully Act.	11.649	16.964	21.631	43.840	85.104	237.009	145.143	250.870	37.791	68.569

transmission delay for all designs. This is a very important aspect for asynchronous circuits, as a short foward delay allows starting communication with interconnected modules earlier. It also corroborates the potential of delay optimizations achieved with SDDS-NCL. If one desires to reduce forward propagation delay, is just a matter of tuning the NCL and NCL+ gates that compose the employed libraries, unbalancing low to high and high to low delays.

Power results show the overheads incurred when constrains are tightened, which forces the tool to use more cells and broader logic paths, to reduce latency. For example, considering the 8bRC, the simplest design, its fast version has 50% and 46% larger idle and active power than its relaxed version. Note that, at the same time, transmission latency is reduced by 65%. These results indicate the suitability of the proposed flow to explore power efficiency, where the designer can trade off power and latency, for example. For verification sake, we also picked the 16bMUL circuit and introduced all combinations of valid data and spacer in the inputs. This allowed us to validate that the circuit would only generate valid data/spacers in the output after all inputs had valid data/spacers, respecting QDI principles.

Another interesting experiment explores the improvements provided by SDDS-NCL and the proposed template over traditional NCL design using a template based approach. To do so, we relied on the ISCAS85 benchmark circuits [BF85] as a representative case study set. The original circuit netlists were modified in order to use dual-rail RTZ channels. These modified netlists were then employed in the proposed flow, generating SDDS-NCL netlists. We used the ISCAS85 netlists to generate NCL circuits replacing the logic gates by equivalent NCL templates, as described in literature. In this case, all channels were dual-rail RTZ. To generate all netlists in this experiment we limited our cell libraries to two input cells only, for the sake of simplicity. Note that for SDDS-NCL, libraries also had inverters.

Table 5.18 compares the area results (in gate count only) between the SDDS-NCL and the traditional NCL template, for each of the ISCAS85 benchmark circuits. Other figures are omitted from this table because its purpose is to demonstrate how logic synthesis can be further explored when using the proposed SDDS-NCL template and design flow. As the Table shows, in all cases the SDDS-NCL version of the netlists allowed reducing the circuit gate count significantly. This is mainly due to the freedom of the synthesis tool to explore logic simplification and the sharing allowed for internal nets. Such freedom is not

Table 5.18 – Gates count comparison for ISCAS85 benchmarks considering the proposed SDDS-NCL template and design flow and traditional NCL design.

Circuit	Total Gates		Optimization
	Template-Based NCL	SDDS-NCL	
c432	1290	754	42%
c1355	1676	1040	38%
c499	1704	1040	39%
c1908	2452	1399	43%
c2670	4184	2435	42%
c3540	8024	4736	41%
c6288	9642	6230	35%
c7552	9644	5530	43%
c5315	10138	5982	41%

feasible in traditional NCL design, due to the constrained nature of template-based mapping. Moreover, NCL assumes that all channels of the circuit are based on RTZ, while SDDS-NCL allows mixing RTZ and RTO. In this way, negative unate cells can be employed in synthesis, allowing further logic optimizations to take place. The result, as the Table shows, are optimizations of 35% in the worst case and 43% in the best case, in terms of the number of gates employed in the design. This reduction impacts in power and area, as the next experiment explores.

After these first experiments, we designed two case study 16-bit multipliers until the layout level: (i) using a template-based approach for mapping NCL gates; and (ii) using the proposed logic and synthesis design flow for targeting maximum performance. For case study (i) we synthesized a single rail version of the multiplier targeting the STMicroelectronics 65nm technology, using the core library available with the PDK. This design targets the maximum possible performance by iteratively setting a *max_delay* constraint in the synthesis tool. We then translated the generated netlist to RTZ dual-rail and replaced the gates mapped by the synthesis tool to NCL templates, as defined in several works [FB96, LFS+00, KL02, BS07, BDSM08, JN08, RST12]. In this case we used the complete ASCEnD-ST65 v2 library, with its 614 NCL cells, matching the mapped driving strengths using a set of shell scripts. Note that this is a fair assumption because the cells in the ASCEnD-ST65 v2 library have driving strengths matching those available in the core library. Moreover, no further performance optimizations were allowed, a restriction imposed by template-based approaches.

For the SDDS-NCL design, case study (ii), we started from the same dual-rail input, but instead of mapping it to gates we mapped it to VFs and synthesized it through the proposed design flow. To do so, we split our library in two VLs, as described in Section 5.5.4, in a total of 56 VFs. For achieving maximum performance in this case we executed several iterations of the proposed design flow, tightening the *max_delay* constraint to maximum

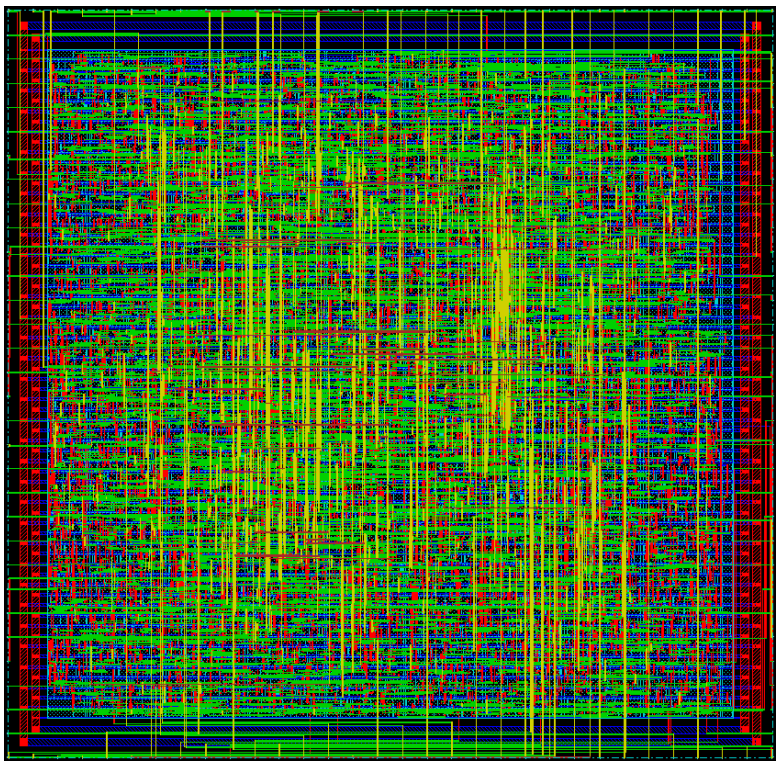


Figure 5.39 – Layout of a case study 16-bit multiplier designed using the proposed SDDS-NCL flow.

stress during timing closure. Figure 5.39 shows a snapshot of one of the generated layouts for case study (ii).

We collected throughput, area and leakage, dynamic and total power figures for each of the designed case studies. Table 5.19 summarizes the results, where case studies (i) and (ii) are called NCL and SDDS-NCL, respectively. As the Table shows, our SDDS-NCL version presented a throughput around 170% larger than that measured for the NCL version. However its power figures are larger, an overhead of roughly 10%, 55% and 45% in terms of leakage, dynamic and total power respectively. Also, as expected, gate count was larger in the SDDS-NCL design. In fact, about 19% more gates were used in this case. This is due to the tight timing constraints specified for achieving maximum performance, which forces the synthesis tool to broaden combinational logic levels. Interestingly, though, the area of the design was reduced by roughly 32%. These is because, albeit the synthesis tool relied on more gates, it employed a large portion of negative unate NCL and NCL+ gates during synthesis, which are typically cheaper in terms of area. Moreover, our flow enabled the use of a larger variety of gates during synthesis, which are quite limited by template-based approaches. In fact, our flow employed 32 of the 56 available VFs for this case study.

These results mean that by allowing overheads in area and power, substantial improvements can be obtained in performance by using the proposed SDDS-NCL design flow, when compared to a template-based approach. Yet, the analysis of these figures combined provides better intuition about the trade-offs enabled by our flow. To this extent, the last sec-

Table 5.19 – Results for the 16-bit multiplier case study synthesized using an NCL template-based approach and the proposed design flow.

	NCL	**SDDS-NCL**
Throughput (MOPS)	103.50	277.78
Leakage Power (mW)	0.124	0.137
Dynamic Power (mW)	0.478	0.739
Total Power (mW)	0.602	0.876
Gate Count	3336	3963
Area mm²	0.0290	0.0197
Normalized MOPS / Leak. Power	1	2.428
Normalized MOPS / Dyn. Power	1	1.737
Normalized MOPS / Total Power	1	1.845
Normalized MOPS / Area	1	3.940

tion of Table 5.19 also presents several efficiency results. These results were obtained by dividing the speed values measured in MOPS by the leakage power, dynamic power, total power and area, respectively. Also, we normalized the results to those of the NCL design, to allow an easier understanding of these metrics. As the Table shows, the SDDS-NCL design provides better efficiency in all cases. Among these, we highlight 294% better MOPS-area efficiency and 143% better MOPS-leakage power efficiency. Another interesting observation is that the SDDS-NCL version of this case study presents roughly 85% better MOPS-total power efficiency, which means that it enables a better compromise between low power and high performance designs, when compared to NCL.

The break down of gates, area and power figures of the design allows a better understanding of the obtained results and the capabilities of the proposed design flow. Accordingly, we divided the gates that composed our designs in 5 types: NCL, NCL+, INCL, INCL+ and inverters (INV). The distribution of the employed gate types in the SDDS-NCL design appears in Figure 5.40(a). Note that for the NCL design, all gates are obviously of the NCL type, limited by the template-based approach. As the chart shows, the majority of the gates used in the SDDS-NCL design were negative unate, and were of types INV, INCL and INCL+. Moreover, there is a balanced division between NCL and NCL+ gates, which indicates that the synthesis tool stressed the SDDS-NCL template, leveraging its advantages. This justifies the obtained results, as interleaving negative unate gates for decomposing unate logic blocks is a classic design optimization performed by these tools.

Figure 5.40(b) shows similar results for area distribution. However, as the chart shows, NCL+ and INCL+ gates present a larger proportion than expected by the analysis of Figure 5.40(a). This means that, broadly speaking, the NCL+ and INCL+ gates used in this design are larger than the employed NCL, INCL and INV gates. On the other hand, leakage and dynamic power distributions, as Figures 5.40(c) and 5.40(d) show, follow the gate type distribution observed in Figure 5.40(a). The reason for this discrepancy in area and power distributions can be explained by the fact that NCL+ logic relies on increased use of PMOS

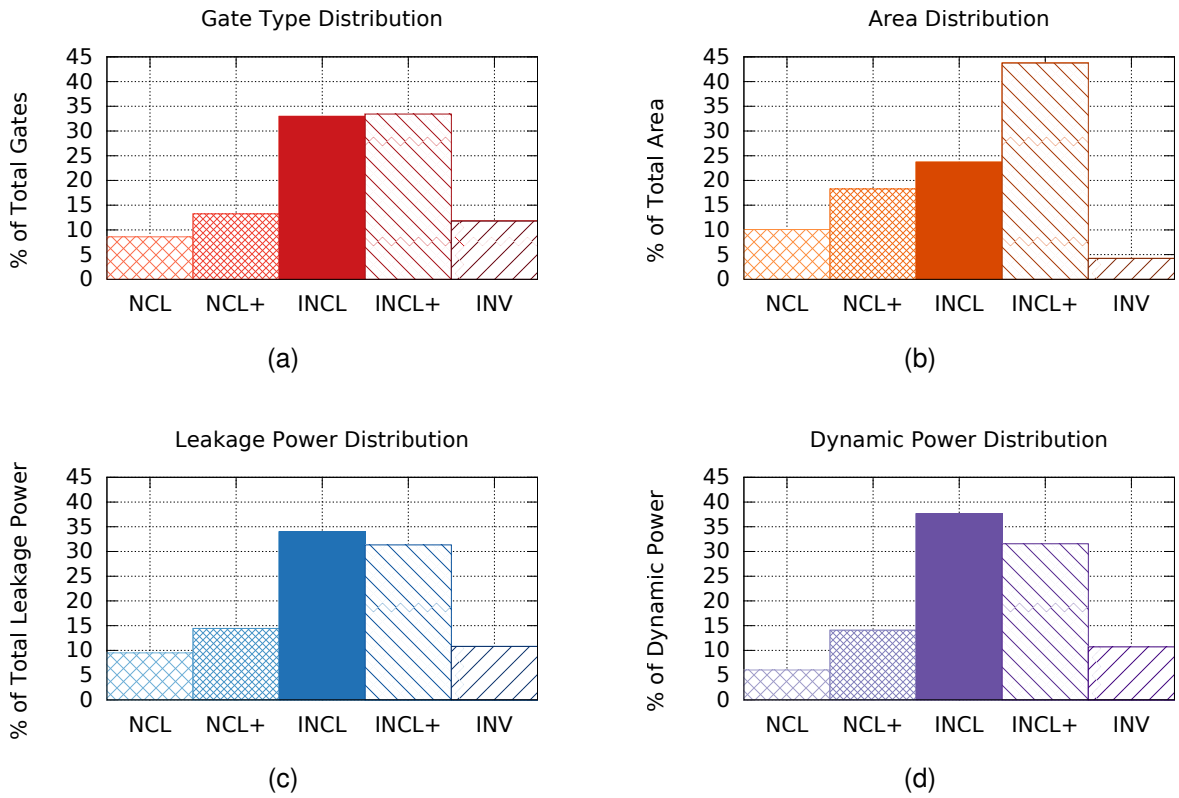


Figure 5.40 – Distribution of (a) gate count, (b) area, (c) leakage power, and (d) dynamic power among different gate types in the fast SDDS-NCL version of the synthesized 16-bit multiplier. Gate types considered here are NCL, NCL+, INCL, INCL+ and INV (inverters).

devices, which are typically larger than NMOS transistors in this technology. In this way, it is expected that the layouts of these cells typically require larger area when compared to NCL, specially as driving strength is increased. Hence, these results confirm our expectations that the usage of SDDS-NCL logic provides better design automation support, performance, power and area trade-offs, when compared to classic NCL approaches.

5.6 Discussion

QDI channels are classically built using RTZ conventions. Our work showed that this is not the most efficient manner of building such channels when considering the electrical behavior of asynchronous cells. Accordingly, our findings suggest that using RTO conventions can lead to substantial savings in static power in these basic cells. Moreover, the combination of RTZ and RTO allows further design exploration in QDI templates, and can lead to further design optimization opportunities, allowing a better trade-off between performance and static and dynamic power. We believe that one of the main reasons that allowed us to propose this simple, but efficient, modification in the way QDI channels are built was the availability of a cell library. Accordingly, this library allowed to explore the physi-

cal implementation of our QDI designs and better understand the electrical behavior of these circuits. Previous works were mainly focused on the architectural level and, probably due to the lack of physical cells, ignored such behavior.

We highlight that using alternative representations for spacers is not a novelty itself, but the proposed templates are fundamentally different from other available works in contemporary literature. In fact, related works that mention the use of all-1s spacers do not employ these to define an RTO-based channel. Neither the design template and associated hardware are built to target such channels. For example, Sokolov *et al.* [Sok06, SMBY05] proposed alternating spacers, a technique that uses all-0s and all-1s spacers successively in computations. They showed their technique is adequate to build secure cryptographic processors, but it incurs in large area overhead when compared to the usual RTZ. This comes from the fact that spacers are temporally distributed in the circuit. Thus, every logic block has to deal with both types of spacers and alternate codes, which increases complexity and limits usage. Cilio *et al.* [CLP+10] also proposed a similar protocol, where each data value is between two different spacers: all-0s and all-1s. They claim the single spacer scheme falls short in balancing switching activity between rails, increasing vulnerability to side-channel attacks (based on e.g. power and electromagnetic leaked information). Albeit the scheme improves robustness, the drawback is again increased area and power. Murphy and Yakovlev [MY06] presented measurements in a prototype AES cryptographic core, which employs the Sokolov *et al.* alternating spacers, and Moore *et al.* [MAM+03b] used the all-1s encoding as an alarm state to reach balanced implementations. For both works, results are high robustness to attacks at high area costs.

There is no specific work proposing the exploration of a protocol based only in the all-1s spacer and its delay and power trade-offs. Hence, the proposed dual-rail RTO WCHB/DIMxS, NCL+ and SDDS-NCL templates allow new degrees of optimizations in QDI circuits. In fact, we demonstrated that RTO DIMxS circuits present a better power-delay trade-off than equivalent RTZ DIMS circuits, and the same is valid for WCHB circuits. Also, C-elements of DIMS circuits are often subject to states that are more prone to generate SEUs and DIMxS alleviates the problem, by keeping C-elements for more time in robust states. Hence, the usage of DIMxS can not only explore energy efficiency, but also mitigate problems caused by transient faults in QDI circuits. One can argue that the modification of transistors sizes in the cells employed in our experiment could alter the obtained results. However, this is not a fair assessment, because all the cells used in this work were dimensioned targeting the best energy efficiency and automatically designed using the ASCEnD-A flow. Note that this sizing strategy is similar to that used in conventional cell libraries for contemporary technologies, where transistors are typically dimensioned targeting the best trade-off between delay and other aspects and balanced delays are ignored.

The proposed NCL+ template provides a new trade-off between power and forward propagation delay. It allows reducing idle power and increasing energy efficiency, which

is of great importance for contemporary challenges like battery-based systems, low power budgets and green computing challenges. Thus, it is a promising way for coping with power requirements in current and future technologies. In this way, NCL+ enables a new approach for asynchronous design, providing better design space exploration capability. SDDS-NCL, can be seen as a further optimization that relies on the usage of NCL and NCL+ and enables conventional EDA tools to seamlessly optimize QDI circuits implementation. Note that related works either employ custom made tools that map NCL designs using template-based approaches or impose severe restrictions to the synthesis tool, which avoids taking advantage of available logic optimization algorithms. A common problem with most of these works e.g. [FBFR07, RST12, PAAS14, LFS+00] is the fact that they focus on pre-mapping synthesis optimizations and then translate the generic netlist to a mapped netlist using a template-based approach that relies on the usage of NCL gates, which is not effective.

Another flow, presented by Kondratyev and Lwin in [KL02] presents an optimization of template-based methods that allow some post-mapping optimization. In fact, the approach is somewhat similar to the one proposed here. For the synthesis steps, NCL gates are represented by their set Boolean functions so that the synthesis tool can use them. The drawback is that because the authors employ only NCL gates, they are able to optimize only the set phase of functions. Thus, they assume that all functions are non-inverted and use only the ON-sets for performing logic optimizations. This clearly avoids taking advantage of more efficient optimizations, which are supported by our design flow through the use of NCL+ gates. In addition, it is not clear how the authors cope with inverted logic that can be inserted in the design, which as we demonstrated here can jeopardize correct functionality. The possibility of having the complement of an NCL gate, defined as an NCL+ gate as we showed in Section 5.5.4, is the core of the proposed design flow. This explains why it was not possible to use commercial tools for mapping NCL gates until the advent of NCL+.

We believe it is a fair assessment to say that the proposed design flow for SDDS-NCL circuits allows extensive usage of IC design commercial frameworks features. In fact, the flow allows taking advantage of well-defined optimization algorithms rather than relying in template-based approaches and optimizations. Salient features of the method include to enable the exploration of STA and pre- and post-mapping optimizations. Furthermore, the flow is capable of mapping QDI circuits into inverted and/or non-inverted NCL and NCL+ gates, which allows generating shorter logic paths than the ones produced when using only non-inverted NCL gates. Another important aspect is that it allows exploring timing/power/area optimizations with conventional constraint specifications and allows post-synthesis optimizations. It also allows handling arbitrary networks with high fanin and complex gate types. Moreover, the proposed flow allows the automatic choice of different NCL components when these are available in the VL. In fact, the synthesis tool decides the best trade-offs based on its own power/timing/area models and constraints. Another major contribution is the definition of a systematic approach for implementing NCL and NCL+ VLs to use in the

proposed design flow. Finally, a very important aspect of the proposed flow is that it can be used together with previously proposed design flows, allowing post mapping optimizations for QDI circuits generated with them. Hence, the flow can be a complement to existing synthesis environments like the ones proposed in [Bar00, BDL11].

6. CONCLUSIONS

There is a large set of works in contemporary literature claiming that asynchronous circuits can help solving problems related to modern IC fabrication processes and application requirements. Yet, these works typically face a same unique problem: immature design automation support. This immature support is majorly a consequence of the lack of basic cell libraries and synthesis tools and flows. Moreover, differently from conventional cell libraries, the ones required for asynchronous design rely on components that have not been extensively validated on silicon applications. In this way, there is no clear definition on how to design such components and their trade offs.

The problem of these limitations is that adhering to asynchronous solutions is too risky for the majority of IC application vendors. This hinders the usage of asynchronous techniques for coping with contemporary IC design problems. This Thesis addressed important issues in this scenario, going from basic components transistor level design evaluation and guidelines definition, to cell libraries implementation and system level asynchronous design and optimizations. In this way, we consider that this work is a step forward in asynchronous circuits design. This Chapter summarizes the original contributions of this Thesis, putting them into context with the state-of-the-art in asynchronous design, and discusses final remarks and directions for future work.

6.1 Contributions of this Work

During this Thesis, the Author collaborated with different research groups. Some of these collaborations were directly related to the subjects addressed here and others were a consequence of the rapid acceptance of our work. Therefore, this Section explores the contributions of the Thesis dividing them in two groups, those directly related to this Thesis and other contributions. The publications and awards granted to the work and somehow related to this Thesis are presented Appendix C.

6.1.1 Thesis Contributions

It is possible to list six major contributions of this Thesis:

1. *Proposition of novel asynchronous components:* In this Thesis, we proposed a novel set of components for QDI design that comprises the set of components required by the NCL+ template. These components allow an alternative to conventional NCL design and, more importantly, were the foundation for the development of the SDDS-NCL

template. Another contribution that we highlight here is the new topology for NCL and NCL+ gates, which allow better design space exploration for building a cell library composed by these components. Furthermore, the new topology serves to build C-elements, as these are special cases of NCL gates. Note that these gates were developed inspired in the challenges we faced while optimizing cells of the first version of the ASCEnD-ST65 library [Mor10]. Moreover, some of the ideas were conceived as we explored new asynchronous templates, after our initial works with RTO channels.

2. *Development of a completely automated flow for asynchronous cell libraries design:* One of the most important contributions of this Thesis is the development of the ASCEnD-A flow. The flow enables generating all models required by semi-custom approaches automatically from a set of specifications. In this way, the ASCEnD-A flow can support designers experimenting with asynchronous techniques, as they can rapidly devise the components required by specific templates or applications. Moreover, the flow eases the task of building novel asynchronous solutions, as it reduces the workload for proposing new templates, for instance by enabling automatic generation of required cells. In fact, this was the case for the development of the NCL+ and the SDDS-NCL templates. With the flow available, we could focus on architectural aspects of an asynchronous design, rather than doing the manual and laborious work of building libraries to support such design. For the development of this flow we worked on tools for transistor sizing and electrical characterization. Transistor sizing tools differ from those available in the original ASCEnD flow [Mor10] as they enable more flexibility in specifying the cost function for transistors. The electrical characterization tool was built from scratch and is the only available tool supporting timing and power characterization of asynchronous cells that we are aware of to this date.

3. *Definitions of guidelines for the design of asynchronous components and cell libraries:* Another important original contribution of this Thesis is the definition of guidelines for designing asynchronous components for building cell libraries. Accordingly, we explored trade offs between different C-element topologies, defining their suitability for energy efficiency, low power, high performance and voltage scaling applications. Moreover, we defined a strategy for building NCL gates that is aware of charge sharing effects. We highlight that this strategy is important, because during the design of the cells that compose the ASCEnD-ST65 v2 library we observed the sensitivity of NCL gates to such effects. In fact, in a first moment, some of the cells of the library presented anomalous behavior during electrical characterization steps. A detailed analysis of the cells revealed that some of them were not respecting the defined guidelines, due to a bug in one of the scripts used for composing their schematics. After fixing the bug, we verified the mitigation of these effects to a level that avoided compromising the cell correct functionality.

4. *Design of the ASCEnD-ST65 v2 library:* Another important contribution of this work is the development of a cell library for asynchronous design that comprises 614 cells and is freely available, provided that the requester has access to the target PDK. This library enables the usage of templates that require C-elements, MUTEXes, NCL and NCL+ gates, including the new templates we proposed here. Moreover, the library has a flexibility in terms of the driving strength of the cells, with a variety of options, that enables exploring power, area and performance trade offs, as discussed in Section 5.4.2. In fact, it is currently being employed in three research groups other than our own.

5. *Proposition of novel QDI templates:* We started working with a different flavor for QDI channels, based on RTO, once we noticed that it reduced the leakage on the cells of DIMS circuits. From this observation we managed to devise three novel QDI templates, namely the Dual-rail RTO WCHB/DIMxS, the NCL+ and the SDDS-NCL. As Chapter 5 explored, these templates allow better design space exploration for QDI circuits, enabling achieving aspects like better energy efficiency, lower absolute power and larger robustness against faults that can be generated by phenomena like radiation, charge sharing and crosstalk.

6. *Development of an automated flow for SDDS-NCL circuits design:* The last original contribution of this Thesis, and also one of the most important, is the proposition of an automated flow for the design of SDDS-NCL circuits. We highlight that one of the major advantages of this flow is the usage of conventional synthesis tools, leveraging their maturity and consolidated optimization algorithms. Moreover, the flow enables for the first time exploring such optimizations during technology mapping, which is precisely where it is possible to obtain insight on the characteristics of the designed circuit. In fact, the obtained results were superior to those achievable with state-of-the-art synthesis techniques for the baseline NCL template, as explored in Section 5.4.2.

6.1.2 Other Contributions

We divide other contributions in two subgroups, component design and optimization and asynchronous templates design. For the former, during our initial studies with C-element topologies, we explored the dynamic transistor topology, scrutinizing its analog behavior and understanding its limitations, as published in [MMC14]. By doing so we realized that the topology was not suited for QDI design because it was too constrained. However, we defined guidelines for using it in applications that assume timing constraints, like BD design or clock managing circuitry. Exploring these options is left as a suggestion of future work.

We then started to explore a differential NCL design template proposed in [YS10]. During our exploration of this template we proposed a new manner of building the components required by it [MAMC15], motivated by the initial results we achieved with the template, which indicated its suitability for low power design. Our component design approach allowed savings of roughly 61.9%, 67.3% and 67.2% in leakage power, dynamic power and energy per operation, respectively. However, despite our substantial improvements, as we further detailed its trade offs, we discarded its usage due to the high complexity for building differential gates and composing circuits using them. In fact, we faced complications when designing differential cells with the ASCEnD-A flow because they have multiple outputs that make sizing of their transistors with our current approach very hard. Moreover, our characterization tool, LiChEn does not yet support multiple output cells, which impairs its usage and support for such cells. Therefore, we didn't explore differential design any further and decided to discard it from the work presented here.

In a recent collaboration with the University of Southern California we worked in the development of Blade, a resilient BD asynchronous design template [HMH+15, HHC+15]. Accordingly, the template allows a performance boost of up to 56% when compared to traditional synchronous solutions. Blade benefited from our mature design flow for cell generation and we devised the required cells for its implementations using ASCEnD-A. Moreover, in [MHBC15] we explored design metrics for error detecting latches, basic components required by resilient architectures. This work evaluates the sensitivity of these components to glitches and proposes a set of optimizations to ensure that their sensitivity can be adjusted according to specific design requirements. We decided to leave this work out of the discussion in this document because it is not a typical requirement for asynchronous design, rather being used on resilient architectures.

In the context of the Blade template design, we also explored delay elements (DEs) design. In [HHS+15], we addressed the issue of designing DEs for BD circuits targeting voltage scaling. The work presents a set of techniques for designing these components and ensuring that their delay scales harmonically with the delay of a logic path as voltage scales. We then devised a new DE using these techniques, as published in [SMT+15]. These works were left out of the scope of the discussion presented in this Thesis because they target BD design specifically and, in terms of design templates, our focus here is QDI.

Also in the context of our collaboration with the University of Southern California, we worked on the proposition of a testable MUTEX component, as explored in [ZHM+15]. MUTEXes are useful in a variety of asynchronous designs including register files [FM04], crossbars, clock domain converters [Lin04], and NoC routers [DGC07, GBN13, PMMC11]. However, despite the importance of MUTEX components in asynchronous design, research literature has overlooked the problem of testing such components. In fact, most works rely on functional test approaches only [AFE+05]. The specialized nature of MUTEXes makes testing them challenging – not only its digital behavior needs to be testable but it is also

important to ensure that they metastability filters (MFs) work and that downstream logic tolerates arbitrarily long metastability resolution times. In this way, we proposed a testable MUTEX that ensures full coverage for stuck-at faults and also enables the testing of its internal MFs. However, we decided to leave it out of this Thesis because it do not address test issues.

6.2 Discussion and Future Work

Asynchronous circuits can avoid the tight constraints imposed by synchronous design. However they require specialized knowledge for their implementation and do not count with the large degrees of design automation support that synchronous circuits do. One of the fundamental needs for IC design is to have available a cell library, because full-custom approaches are too time consuming and impractical, but for specific cases. Unfortunately, this is precisely one of the greatest barriers for asynchronous designers, as there is no such library easily available. As a consequence, different authors devise their own components on demand, so that they can experiment with asynchronous techniques. The drawback is that designing such libraries is a very laborious and error-prone task, if done manually and, to aggravate this, EDA tools for cell libraries design do not always support such components directly.

In this context, ASCEnD-A is a step forward in the direction of better design space exploration and acceptance of asynchronous circuits, as it allows the rapid design of cell libraries in an automated manner. In fact, the flow was well accepted by the research community, and several publications related to the flow and its optimizations exist. Furthermore, we received 2 awards for works related to the flow, another indication of its well acceptance. Moreover, we were invited to present the flow in a demo session at the International Symposium on Asynchronous Circuits and Systems, the flagship conference worldwide in asynchronous design, in its 2014 edition [MAZ+14a]. We also presented the flow in the University Booth at the Design Automation Conference in its 2014 edition [MOC+14]. This indicates the relevance that such flow and how it is perceived by the research community.

From our perspective, the flow was of great importance, because it enabled us to explore alternatives and optimizations in the design of components required for the design of asynchronous circuits. This, in turn, enabled us to address existing design templates and, more importantly, propose new ones. In fact, the major relevance of the flow in this process comes from the fact that having the possibility of rapidly generating cells allowed us to collect precise measurements from circuit simulation. Also, we could focus more on the architectural design of asynchronous circuits. Moreover, with the automated design of components we could understand new manners of designing and optimizing them, defining a set of guidelines for the ones we used in our experiments. Such guidelines proved to

be important as we observed the negative effects of not following them when using the generated components, like the problems with charge sharing effects on NCL gates.

Another drawback in asynchronous design that we observed is that the large variety of design templates for building asynchronous design makes the definition of what components are to be designed for an asynchronous circuit a hard task. In this way, another advantage of having an automated flow like ASCEnD-A is that it enables an exploration similar to the one we did with the selected templates, but targeting other flavors of asynchronous design templates. This provides designers with better flexibility, allowing them to experiment with different templates before defining one, and more agility, as they can quickly collect power, performance and area figures. Hence, generally speaking there is now a broader road to be paved in the design of asynchronous components.

Using the components we addressed here, we built a library that is freely available, provided that the requester has the NDAs required by the supplier of the target PDK. This library was of great relevance in our work, because it allowed the design of different IPs, like the two NoC routers presented in [GMMC15, MMG+13], which helped to guide our way through the available templates. The library supports different BD and QDI templates by having a generic basic cell set, besides the specialized cells for SDDS-NCL design. In fact, having such a library available motivated different work fronts within our research group and triggered worldwide collaborations. In our group the library is now being used for devising BD circuits in a flow called ACDC, as presented in [GMC15], and for subthreshold design, as discussed in [GMCM15]. Also, during the development of this work, in its different versions, the library was used in research conducted by four different research groups, from Universities in Brazil (*Universidade Federal do Rio Grande do Sul* and *Universidade Federal de Pelotas*), United States (*University of Southern California*) and, recently, Denmark (*Technical University of Denmark*).

Having this library and design flow available was one of the major factors that allowed the proposition of Blade, in a cooperation between our group and the University of Southern California, and our new QDI templates, NCL+ and SDDS-NCL, detailed in this Thesis. The SDDS-NCL template is a good example of how asynchronous designers can benefit from the availability of our design flow for cell libraries. In fact, it was conceived due to the availability of NCL and NCL+ gates in a cell library, The inspiration for the ideia of template came from an article by Ligthart et al. [LFS+00], where the authors claim that for NCL synthesis they could assume circuit operation as being two-phased, with a set and a reset phase. The set phase computes valid data, setting some cells of the circuit to 1 and the reset phase computes spacers, setting all cells to 0. In this way, the set phase can fully express the functionality of the circuit, as the reset phase is performed uniformly for all cells, which is guaranteed by their OFF-set. Using this concept, we managed to overcome the limitation of using only positive unate cells, now that NCL+ gates are available. In this way, whenever a negative unate gate was used in the circuit, it changed the spacer representa-

tion and we could fix that by changing the family of the gates in the series, NCL whenever inputs had all-0 spacers and NCL+ whenever they had all-1 spacers.

One of the major advantages of this new template is that we can now use commercial tools to perform all the steps of technology mapping and synthesis optimizations using an automated flow. The obtained results indicate the quality and relevance of the approach, leading to substantial improvements in area, performance and power figures. Note that in the comparisons we presented for the SDDS-NCL template we did not include synchronous or BD designs because they are out of the scope of the work we presented here, limited to some QDI templates. We believe this is a fair assessment, because our optimizations are all based on well-accepted templates that have triggered movement in both academia and industry.

This Thesis explored different aspects of asynchronous design, from transistor- to system-level. Moreover, it allowed the integration of the work developed by our groups with the work developed at different fellow research groups. Therefore, it paves the way for the a large spectrum of future works, some of which are actually ongoing work. Among the envisaged possibilities we can cite:

1. *Improvements in the ASCEnD-A flow*: In its current version the flow has some limitations. Firstly, it supports only single-output cells, which limits its application to templates like differential and PCHB-based ones. To overcome this, we are already working on the development of a new tool for transistor sizing called Tweak, which will be an alternative to ROGen. Its output is compatible with CeS and the remainder of the ASCEnD-A flow. It is also part of future work adding to Tweak the capability of fine grain sizing transistors, allowing it to optimize the size of individual transistors one by one. Another ongoing work is the generation of the version 2 of LiChEn, an optimization of the tool that will allow the electrical characterization of multi-output components. In this implementation we are also adding the automated analysis of charge sharing effects on the cells under characterization. Once these tools are designed, the flow can be used to generate multi-output cells. Secondly, ASTRAN currently supports only PDKs with geometries as fine as 45 nm. Moreover it does not support multi-row cells, which limits the automation of cell libraries for templates like PCHB-based ones. In this way, as future work we will work on ASTRAN for providing better support to newer technologies and layout arrangements.

2. *Synthesis of QDI descriptions compatible with our SDDS-NCL design flow*: We currently assume that the input of our flow is a QDI description based on an 1-of-n 4-phase channel. However this assumption is too constraining for real applications, because designers are not used to describing circuits in this way. The description is not easily mastered and requires specific knowledge on asynchronous design. Therefore, it is part of future work the integration of a tool that automates the process of generating

this description from a higher level abstraction, such as an RTL or a communicating hardware processes (CHP) description [BOF10]. In fact, we are already exploring possible solutions, such as the framework provided by Wave Semiconductor [Wav] and Proteus [BDL11].

3. *Library optimizations for SDDS-NCL designs*: A point that was not addressed in this Thesis is the composition of a cell library for supporting SDDS-NCL design. Accordingly, it is part of future work defining minimum and optimal sets of cells and exploring their trade offs. Such exploration can help guiding designers to implement SDDS-NCL designs faster by reducing cell library development time.

4. *Automated support of sequential SDDS-NCL design*: A current limitation of the SDDS-NCL flow is its automated support for combinational circuits only. In fact, for sequential designs there is no automation and the designer must synthesize and optimize each combinational block separately. Ongoing work comprises the addition of a set of scripts for overcoming this limitation.

5. *Exploration of Sleep Convention Logic to optimize SDDS-NCL design*: A recently proposed QDI template called sleep convention logic [PS13] relies on fine grain power gating for improving energy efficiency in QDI design. We believe this design template can benefit from a spatial spacer distribution as in SDDS-NCL. In this way, we are currently working on the optimization of the components required by this template and also in architectural optimizations for it.

6. *Exploration of near- and sub-threshold design with QDI*: Given their robustness against delay variations, we believe QDI circuits are good candidates for near- and sub-threshold applications, as they face delay variations that are substantially more aggressive than those in nominal voltage operation. In fact, it has been demonstrated [Fan15] that these circuits gracefully scale their power and delay characteristics as voltage is scaled. Moreover, they do not require complex circuitry for managing voltage scaling as synchronous or BD solutions do. As ongoing work, we are starting to explore the usage of SDDS-NCL for such applications, as published in [GMCM15].

7. *Design of a set of test chips*: We have recently sent to fabrication and validated on silicon a synchronous microcontroller, in a multi-project wafer program that allows us free runs to prototype academic designs. The target technology was the TSMC 180 nm Bulk CMOS. The whole process allowed us to master the flow of fabricating ICs and the next step is the prototyping of a test chip to validate our experiments with asynchronous design. To that extent we are currently working on the development of a new asynchronous cell library targeting this technology.

REFERENCES

[ABF94] Abramovici, M.; Breuer, M. A.; Friedman, A. D. “Digital Systems Testing and Testable Design”. Wiley Interscience-IEEE Press, 1994.

[Ach] “Achronix Semiconductor Corporation”. Source: http://www.achronix.com/.

[AFE+05] Amde, M.; Felicijan, T.; Efthymiou, A.; Edwards, D.; Lavagno, L. “Asynchronous On-chip Networks”, *IEE Proceedings - Computers and Digital Techniques*, vol. 152–2, Mar 2005, pp. 273–283.

[AN12] Agyekum, M.; Nowick, S. “Error-Correcting Unordered Codes and Hardware Support for Robust Asynchronous Global Communication”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31–1, Jan 2012, pp. 75–88.

[ARM] “Advanced RISC Machines”. Source: http://www.arm.com/.

[Bar98] Bardsley, A. “Balsa: An Asynchronous Circuit Synthesis System”, Master’s Thesis, Faculty of Science & Engineering, University of Manchester, 1998.

[Bar00] Bardsley, A. “Implementing Balsa Handshake Circuits”, Ph.D. Thesis, Faculty of Science & Engineering, University of Manchester, 2000.

[BDL11] Beerel, P.; Dimou, G.; Lines, A. “Proteus: An ASIC Flow for GHz Asynchronous Designs”, *IEEE Design & Test of Computers*, vol. 28–5, Sep 2011, pp. 36–51.

[BDSM08] Bailey, A.; Di, J.; Smith, S.; Mantooth, H. “Ultra-low power delay-insensitive circuit design”. In: 51st Midwest Symposium on Circuits and Systems (MWSCAS), 2008, pp. 503–506.

[Ber92] van Berkel, K. “Beware the Isochronic Fork”, *Integration the VLSI Journal*, vol. 13–2, Jun 1992, pp. 103–128.

[BF85] Brglez, E.; Fujiwara, H. “A Neutral Netlist of 10 Combinational Benchmark Designs and a Special Translator in Fortran”. In: International Symposium on Circuits and Systemsn (ISCAS), 1985.

[BLDK06] Beerel, P.; Lines, A.; Davies, M.; Kim, N.-H. “Slack matching asynchronous designs”. In: 12th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2006, pp. 184–194.

[BNS+07] 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 the Electromagnetic Compatibility of Integrated Circuits (EMC Compo), 2007.

[BOF10] Beerel, P.; Ozdag, R.; Ferretti, M. "A Designer's Guide to Asynchronous VLSI". Cambridge University Press, 2010.

[BS07] Bandapati, S. K.; Smith, S. C. "Design and characterization of {NULL} convention arithmetic logic units", *Microelectronic Engineering*, vol. 84–2, 2007, pp. 280–287.

[BSK+10] Bastos, R.; Sicard, G.; Kastensmidt, F.; Renaudin, M.; Reis, R. "Evaluating transient-fault effects on traditional C-element's implementations". In: IEEE 16th International On-Line Testing Symposium (IOLTS), 2010, pp. 35–40.

[BSVMH84] Brayton, R.; Sangiovanni-Vincentelli, A.; McMullen, C.; Hachtel, G. "Logic Minimization Algorithms for VLSI Synthesis". Norwell, MA, USA: Kluwer Academic Publishers, 1984.

[BTE09] Bardsley, A.; Tarazona, L.; Edwards, D. "Teak: A Token-Flow Implementation for the Balsa Language". In: Ninth International Conference on Application of Concurrency to System Design (ACSD), 2009, pp. 23–31.

[BTEF03] Bainbridge, W.; Toms, W.; Edwards, D.; Furber, S. "Delay-insensitive, point-to-point interconnect using m-of-n codes". In: 9th International Symposium on Asynchronous Circuits and Systems (ASYNC), 2003, pp. 132–140.

[Cad] "Cadence Design Systems". Source: http://www.cadence.com/.

[Cad07] Cadence Design Systems. "Cadence Abstract Generator User Guide", Technical Report, San Jose, CA, USA, 2007, 434p.

[CCGC10] Chen, J.; Chong, K.-S.; Gwee, B.-H.; Chang, J. S. "An ultra-low power asynchronous quasi-delay-insensitive (QDI) sub-threshold memory with bit-interleaving and completion detection". In: 8th IEEE International NEWCAS Conference (NEWCAS), 2010, pp. 117–120.

[CCGC13] Chang, K.-L.; Chang, J.; Gwee, B.-H.; Chong, K.-S. "Synchronous-Logic and Asynchronous-Logic 8051 Microcontroller Cores for Realizing the Internet of Things: A Comparative Study on Dynamic Voltage Scaling and Variation Effects", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 3–1, Mar 2013, pp. 23–34.

[CCKT09] Carmona, J.; Cortadella, J.; Kishinevsky, M.; Taubin, A. "Elastic Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28–10, Oct 2009, pp. 1437–1455.

[CH87] Chu, C.-Y.; Horowitz, M. “Charge-Sharing Models for Switch-Level Simulation”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6–6, Nov 1987, pp. 1053–1061.

[Cha84] Chapiro, D. M. “Globally-asynchronous locally-synchronous systems”, Ph.D. Thesis, Stanford University, 1984.

[Cit04] Citron, D. “Exploiting Low Entropy to Reduce Wire Delay”, *Computer Architecture Letters*, vol. 3–1, Jan 2004, pp. 4.

[CKLS06] Cortadella, J.; Kondratyev, A.; Lavagno, L.; Sotiriou, C. “Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25–10, Oct 2006, pp. 1904–1921.

[CLP+10] Cilio, W.; Linder, M.; Porter, C.; Di, J.; Smith, S.; Thompson, D. “Side-channel attack mitigation using dual-spacer Dual-rail Delay-insensitive Logic (D3L)”. In: IEEE SoutheastCon (SoutheastCon), 2010, pp. 471–474.

[CLP+13] Cilio, W.; Linder, M.; Porter, C.; Di, J.; Thompson, D. R.; Smith, S. C. “Mitigating Power- and Timing-based Side-channel Attacks Using Dual-spacer Dual-rail Delay-insensitive Asynchronous Logic”, *Microelectronics Journal*, vol. 44–3, Mar 2013, pp. 258–269.

[CMP] “Circuits Multi-Projects”. Source: http://cmp.imag.fr/.

[CPR10] Chang, I. J.; Park, S. P.; Roy, K. “Exploring Asynchronous Design Techniques for Process-Tolerant and Energy-Efficient Subthreshold Operation”, *IEEE Journal of Solid-State Circuits*, vol. 45–2, Feb 2010, pp. 401–410.

[DCB93] Das, D. K.; Chakraborty, S.; Bhattacharya, B. “Irredundant binate realizations of unate functions”, *International Journal of Electronics*, vol. 75–1, Jul 1993, pp. 65–73.

[DENB05] Doan, C.; Emami, S.; Niknejad, A.; Brodersen, R. “Millimeter-wave CMOS design”, *IEEE Journal of Solid-State Circuits*, vol. 40–1, Jan 2005, pp. 144–155.

[DGC07] Dobkin, R.; Ginosar, R.; Cidon, I. “QNoC Asynchronous Router with Dynamic Virtual Channel Allocation”. In: First International Symposium on Networks-on-Chip (NoCs), 2007, pp. 218–218.

[DK92a] Devadas, S.; Keutzer, K. “Synthesis of robust delay-fault-testable circuits: practice”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11–3, Mar 1992, pp. 277–300.

[DK92b] Devadas, S.; Keutzer, K. “Synthesis of robust delay-fault-testable circuits: theory”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11–1, Jan 1992, pp. 87–101.

[DLD+14] Davies, M.; Lines, A.; Dama, J.; Gravel, A.; Southworth, R.; Dimou, G.; Beerel, P. “A 72-Port 10G Ethernet Switch/Router Using Quasi-Delay-Insensitive Asynchronous Design”. In: 20th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2014, pp. 103–104.

[DTP09] Das, S.; Tokunaga, C.; Pant, S. “RazorII: In situ error detection and correction for PVT and SER tolerance”, *IEEE Journal of Solid-State Circuits*, vol. 44–1, Jan 2009, pp. 32–48.

[EKD+03] Ernst, D.; Kim, N. S.; Das, S.; Pant, S.; Rao, R.; Pham, T.; Ziesler, C.; Blaauw, D.; Austin, T.; Flautner, K.; Mudge, T. “Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation”. In: 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36), 2003, pp. 7–18.

[Eke10] Ekekwe, N. “Power dissipation and interconnect noise challenges in nanometer CMOS technologies”, *IEEE Potentials*, vol. 29–3, May 2010, pp. 26–31.

[ELEHS03] 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.

[EMR95] El-Maleh, A.; Rajski, J. “Delay-fault testability preservation of the concurrent decomposition and factorization transformations”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14–5, May 1995, pp. 582–590.

[Fan15] Fant, K., “Private communication”, May, 2015.

[FB96] Fant, K.; Brandt, S. “NULL Convention Logic^{TM}: a complete and consistent logic for asynchronous digital circuit synthesis”. In: International Conference on Application Specific Systems, Architectures and Processors (ASAP’96), 1996, pp. 261–273.

[FBFR07] Folco, B.; Bregier, V.; Fesquet, L.; Renaudin, M. “Technology Mapping for Area Optimized Quasi Delay Insensitive Circuits”. In: IEEE/IFIP 15th International Conference on VLSI and System-on-Chip (VLSI-SoC), 2007, pp. 55–69.

[Fer04] Ferretti, M. “Single-Track Asynchronous Pipeline Template”, Ph.D. Thesis, University of Southern California, 2004.

[FM04] Fang, D.; Manohar, R. “Non-Uniform Access Asynchronous Register Files”. In: 10th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2004, pp. 78–85.

[Fre] “FreePDK45”. Source: http://www.eda.ncsu.edu/wiki/FreePDK45:Contents.

[Fri01] Friedman, E. “Clock distribution networks in synchronous digital integrated circuits”, *Proceedings of the IEEE*, vol. 89–5, May 2001, pp. 665–692.

[Fuc95] Fuchs, K. “Synthesis for path delay fault testability via tautology-based untestability identification and factorization”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14–12, Dec 1995, pp. 1470–1479.

[GB05] Gulati, G.; Brunvand, E. “Design of a Cell Library for Asynchronous Microengines”. In: 15th ACM Great Lakes Symposium on VLSI (GLSVLSI), 2005, pp. 385–389.

[GBN13] Ghiribaldi, A.; Bertozzi, D.; Nowick, S. M. “A transition-signaling bundled data NoC switch architecture for cost-effective GALS multicore systems”. In: Design, Automation & Test in Europe Conference Exhibition (DATE), 2013, 2013, pp. 332–337.

[GHMC14] Guazzelli, R.; Heck, G.; Moreira, M.; Calazans, N. “Schmitt trigger on output inverters of NCL gates for soft error hardening: Is it enough?” In: 15th Latin American Test Workshop (LATW), 2014, pp. 1–5.

[Gib13] Gibiluka, M. “Design and Implementation of an Asynchronous NoC Router Using a Transition-Signaling Bundled-Data Protocol”, Technical Report, Pontifícia Universidade Católica do Rio Grande do Sul. End of term work, 2013.

[GK08] Garg, R.; Khatri, S. “A novel, highly SEU tolerant digital circuit design approach”. In: IEEE International Conference on Computer Design (ICCD), 2008, pp. 14–20.

[GLY10] Guan, X.; Liu, Y.; Yang, Y. “Performance Analysis of Low Power Null Convention Logic Units with Power Cutoff”. In: Asia-Pacific Conference on Wearable Computing Systems (APWCS), 2010, pp. 55–58.

[GMC15] Gibiluka, M.; Moreira, M.; Calazans, N. “A Bundled-Data Asynchronous Circuit Synthesis Flow Using a Commercial EDA Framework”. In: EUROMICRO Conference on Digital System Design (DSD), 2015, pp. 79–86.

[GMCM15] Guazzelli, R. A.; Moraes, F. G.; Calazans, N. L. V.; Moreira, M. T. “SDDS-NCL Design: Analysis of Supply Voltage Scaling”. In: 28th Symposium on Integrated Circuits and Systems Design (SBCCI), 2015, pp. 2:1–2:7.

[GMMC15] Gibiluka, M.; Moreira, M. T.; Moraes, F. G.; Calazans, N. L. V. “BAT-Hermes: A Transition-Signaling Bundled-Data NoC Router”. In: IEEE 6th Latin American Symposium on Circuits and Systems (LASCAS), 2015.

[Har07] Hartmann, J. “Towards a New Nanoelectronic Cosmology”. In: IEEE International Solid-State Circuits Conference (ISSCC), 2007, pp. 31–37.

[HFO03] Hashimoto, M.; Fujimori, K.; Onodera, H. “Standard cell libraries with various driving strength cells for 0.13, 0.18 and 0.35 μm technologies”. In: Asia and South Pacific Design Automation Conference (ASP-DAC), 2003, pp. 589–590.

[HGD04] Ho, R.; Gainsley, J.; Drost, R. “Long wires and asynchronous control”. In: 10th International Symposium on Asynchronous Circuits and Systems, 2004, pp. 240–249.

[HHC+15] Hand, D.; H., H.; Chang, B.; Zhang, Y.; Moreira, M. T.; Breuer, M.; Calazans, N. L. V.; Beerel, P. A. “Performance Optimization and Analysis of Blade Designs Under Delay Variability”. In: 21st IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2015, pp. 61–68.

[HHS+15] Heck, G.; Heck, L.; Singhvi, A.; Moreira, M. T.; Beerel, P.; Calazans, N. L. V. “Analysis and Optimization of Programmable Delay Elements for 2-Phase Bundled-Data Circuits”. In: 28th International Conference on VLSI Design (VLSID), 2015, pp. 321–326.

[HMH01] Ho, R.; Mai, K.; Horowitz, M. “The future of wires”, *Proceedings of the IEEE*, vol. 89–4, 2001, pp. 490–504.

[HMH+15] Hand, D.; Moreira, M. T.; H., H.; Chen, D.; Butzke, F.; Gibiluka, M.; Breuer, M.; Calazans, N. L. V.; Beerel, P. A. “Blade - A Timing Violation Resilient Asynchronous Template”. In: 21st IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2015, pp. 21–28.

[Hur69] Hurst, S. “An Introduction to Threshold Logic: A Survey of Present Theory and Practice”, *The Radio and Electronic Engineer*, 1969, pp. 339–351.

[HZB+06] Hanson, S.; Zhai, B.; Bernstein, K.; Blaauw, D.; Bryant, A.; Chang, L.; Das, K.; Haensch, W.; Nowak, E.; Sylvester, D. “Ultralow-voltage, minimum-energy CMOS”, *IBM Journal of Research and Development*, vol. 50–4.5, Jul 2006, pp. 469–490.

[IEE] “Institute of Electrical and Electronics Engineers”. Source: https://www.ieee.org/.

[Int] “Intel Corporation, url=http://www.intel.com, urldate=2015-06-06”.

[ITR] “The International Technology Roadmap for Semiconductors (ITRS)”. Source: http://www.itrs.net/.

[JN08] Jeong, C.; Nowick, S. “Technology Mapping and Cell Merger for Asynchronous Threshold Networks”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27–4, Apr 2008, pp. 659–672.

[JNA00] Jambek, A.; NoorBeg, A.; Ahmad, M. “Standard cell library development”. In: The 11th International Conference on Microelectronics (ICM), 2000, pp. 161–163.

[JSL+10] Jorgenson, R.; Sorensen, L.; Leet, D.; Hagedorn, M.; Lamb, D.; Friddell, T.; Snapp, W. “Ultralow-Power Operation in Subthreshold Regimes Applying Clockless Logic”, *Proceedings of the IEEE*, vol. 98–2, Feb 2010, pp. 299–314.

[Key01] Keyes, R. “Fundamental limits of silicon technology”, *Proceedings of the IEEE*, vol. 89–3, Mar 2001, pp. 227–239.

[KGB+11] Kuhn, K. J.; Giles, M. D.; Becher, D.; Kolar, P.; Kornfeld, A.; Kotlyar, R.; Ma, S. T.; Maheshwari, A.; Mudanai, S. “Process Technology Variation”, *IEEE Transactions on Electron Devices*, vol. 58–8, Aug 2011, pp. 2197–2208.

[KH04] Karnik, T.; Hazucha, P. “Characterization of soft errors caused by single event upsets in CMOS processes”, *IEEE Transactions on Dependable and Secure Computing*, vol. 1–2, Apr 2004, pp. 128–143.

[Kin08] Kinniment, D. “Synchronization and Arbitration in Digital Systems”. John Wiley & Sons, Ltd, 2008.

[KKFK13] Kim, S.; Kwon, I.; Fick, D.; Kim, M. “Razor-lite: A side-channel error-detection register for timing-margin recovery in 45nm SOI CMOS”. In: IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013, pp. 264–265.

[KL02] Kondratyev, A.; Lwin, K. “Design of asynchronous circuits using synchronous CAD tools”, *IEEE Design & Test of Computers*, vol. 19–4, Jul 2002, pp. 107–117.

[KLSV95] Keutzer, K.; Lavagno, L.; Sangiovanni-Vincentelli, A. “Synthesis for testability techniques for asynchronous circuits”, *IEEE Transactions on Computer-Aided*

Design of Integrated Circuits and Systems, vol. 14–12, Dec 1995, pp. 1569–1577.

[KM95] Ke, W.; Menon, P. “Path-delay-fault testable nonscan sequential circuits”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14–5, May 1995, pp. 576–582.

[KOM13] Karmazin, R.; Otero, C.; Manohar, R. “cellTK: Automated Layout for Asynchronous Circuits with Nonstandard Cells”. In: IEEE 19th International Symposium on Asynchronous Circuits and Systems (ASYNC), 2013, pp. 58–66.

[KS73] Kogge, P. M.; Stone, H. S. “A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations”, *IEEE Transactions on Computers*, vol. C-22–8, Aug 1973, pp. 786–793.

[KZYD10] Kuang, W.; Zhao, P.; Yuan, J.; DeMara, R. “Design of Asynchronous Circuits for High Soft Error Tolerance in Deep Submicrometer CMOS Circuits”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18–3, Mar 2010, pp. 410–422.

[LFS+00] Ligthart, M.; Fant, K.; Smith, R.; Taubin, A.; Kondratyev, A. “Asynchronous Design Using Commercial HDL Synthesis Tools”. In: 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC), 2000, pp. 114–125.

[Lib] “Liberty User Guides and Reference Manual Suite”. Source: https://www.opensourceliberty.org/opensourceliberty.html.

[Lin98] Lines, A. M. “Pipelined Asynchronous Circuits”, Technical Report, Department of Computer Science – California Institute of Technology, 1998, 37p.

[Lin04] Lines, A. “Asynchronous Interconnect for Synchronous SoC Design”, *IEEE Micro*, vol. 24–1, Jan/Fev 2004.

[LM04] LaFrieda, C.; Manohar, R. “Fault detection and isolation techniques for quasi delay-insensitive circuits”. In: International Conference on Dependable Systems and Networks (DSN), 2004, pp. 41–50.

[LOM07] Lotze, N.; Ortmanns, M.; Manoli, Y. “A study on self-timed asynchronous subthreshold logic”. In: 25th International Conference on Computer Design (ICCD), 2007, pp. 533–540.

[MAGC14] Moreira, M. T.; Arendt, M. E.; Guazzelli, R. A.; Calazans, N. L. V. “A New CMOS Topology for Low-Voltage Null Convention Logic Gates Design”. In: 20th IEEE

International Symposium on Asynchronous Circuits and Systems (ASYNC), 2014, pp. 93–100.

[MAM+03a] Moore, S.; Anderson, R.; Mullins, R.; Taylor, G.; Fournier, J. J. A. “Balanced Self-Checking Asynchronous Logic for Smart Card Applications”, *Journal of Microprocessors and Microsystems*, vol. 27, 2003, pp. 421–430.

[MAM+03b] Moore, S.; Anderson, R.; Mullins, R.; Taylor, G.; Fournier, J. J. A. “Balanced Self-Checking Asynchronous Logic for Smart Card Applications”, *Microprocessors and Microsystems*, vol. 27–9, 2003, pp. 421–430.

[MAMC15] Moreira, M. T.; Arendt, M.; Moraes, F. G.; Calazans, N. L. V. “Static Differential NCL Gates: Toward Low Power”, *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62–6, Jun 2015, pp. 563–567.

[Mar90] Martin, A. J. “The limitations to delay-insensitivity in asynchronous circuits”. In: 6th MIT Conference on Advanced Research in VLSI (AUSCRYPT), 1990, pp. 263–278.

[MAZ+14a] Moreira, M.; Arendt, M. E.; Ziesemer Jr., A.; Reis, R. A. L.; Calazans, N. “Automated Synthesis of Cell Libraries for Semi-Custom Asynchronous Design”. In: 20th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), Fresh Ideas Workshop, 2014.

[MAZ+14b] Moreira, M. T.; Arendt, M.; Ziesemer Jr., A.; Reis, R.; Calazans, N. L. V. “Automated Synthesis of Cell Libraries for Asynchronous Circuits”. In: 27th Symposium on Integrated Circuits and Systems Design (SBCCI), 2014, pp. 16:1–16:7.

[MB57] Muller, D. E.; Bartky, W. S. “A Theory of Asynchronous Circuits”. In: International Symposium on the Theory of Switching, 1957.

[MC13] Moreira, M. T.; Calazans, N. L. V. “Voltage scaling on C-elements: A speed, power and energy efficiency analysis”. In: IEEE 31st International Conference on Computer Design (ICCD), 2013, pp. 329–334.

[MDB14] Mitra, R.; Das, D.; Bhattacharya, B. “On Designing Robust Path-Delay Fault Testable Combinational Circuits Based on Functional Properties”. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2014, pp. 202–207.

[Men] “Mentor Graphics”. Source: http://www.mentor.com/.

[MFG03] Mohammadi, S.; Furber, S.; Garside, J. “Designing robust asynchronous circuit components”, *Circuits, Devices and Systems, IEE Proceedings* -, vol. 150–3, Jun 2003, pp. 161–166.

[MGC12a] Moreira, M. T.; Guazzelli, R. A.; Calazans, N. L. V. “Return-to-One DIMS logic on 4-phase m-of-n asynchronous circuits”. In: 19th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2012, pp. 669–672.

[MGC12b] Moreira, M. T.; Guazzelli, R. A.; Calazans, N. L. V. “Return-to-one protocol for reducing static power in C-elements of QDI circuits employing m-of-n codes”. In: 25th Symposium on Integrated Circuits and Systems Design (SBCCI), 2012, pp. 1–6.

[MGHC14] Moreira, M. T.; Guazzelli, R. A.; Heck, G.; Calazans, N. L. V. “Hardening QDI Circuits Against Transient Faults Using Delay-insensitive Maxterm Synthesis”. In: 24th Great Lakes Symposium on VLSI (GLSVLSI), 2014, pp. 3–8.

[MHBC15] Moreira, M.; Hand, D.; Beerel, P. A.; Calazans, N. “TDTB Error Detecting Latches: Timing Violation Sensitivity Analysis and Optimization”. In: 16th International Symposium on Quality Electronic Design (ISQED), 2015.

[Mic94] Micheli, G. D. “Synthesis and Optimization of Digital Circuits”. McGraw-Hill Higher Education, 1994.

[MMC14] Moreira, M. T.; Moraes, F. G.; Calazans, N. L. V. “Beware the Dynamic C-element”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22–7, Jul 2014, pp. 1644–1647.

[MMG+13] Moreira, M. T.; Magalhaes, F. G.; Gibiluka, M.; Hessel, F. P.; Calazans, N. L. V. “BaBaNoC: An asynchronous network-on-chip described in Balsa”. In: International Symposium on Rapid System Prototyping (RSP), 2013, pp. 37–43.

[MN06] Martin, A.; Nystrom, M. “Asynchronous Techniques for System-on-Chip Design”, *Proceedings of the IEEE*, vol. 94–6, Jun 2006, pp. 1089–1120.

[MNM+14] Moreira, M.; Neutzling, A.; Martins, M.; Reis, A.; Ribas, R.; Calazans, N. “Semi-custom NCL Design with Commercial EDA Frameworks: Is it Possible?” In: 20th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2014, pp. 53–60.

[MOC+14] Moreira, M.; Oliveira, B.; C., O.; Guazzelli, R.; Medeiros, G.; Arendt, M. E.; Ziesemer Jr., A.; Reis, R. A. L.; Calazans, N. “ASCEnD: Automated Asynchronous Cell Libraries Design”. In: Design Automation Conference (DAC), University Booth, 2014.

[MOCO13] Moreira, M. T.; Oliveira, C. M.; Calazans, N. L. V.; Ost, L. C. “LiChEn: Automated Electrical Characterization of Asynchronous Standard Cell Libraries”. In: Euromicro Conference on Digital System Design (DSD), 2013, pp. 933–940.

[MOMC12] Moreira, M.; Oliveira, B.; Moraes, F.; Calazans, N. “Impact of C-elements in asynchronous circuits”. In: 13th International Symposium on Quality Electronic Design (ISQED), 2012, pp. 437–343.

[MOMC13] Moreira, M. T.; Oliveira, B. S.; Moraes, F. G.; Calazans, N. L. V. “Charge sharing aware NCL gates design”. In: IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2013, pp. 212–217.

[Moo65] Moore, G. “Cramming more components onto integrated circuits”, *Electronics Magazine*, vol. 38–8, 1965.

[Moo03] Moore, G. E. “No exponential is forever: but "Forever" can be delayed! [semiconductor industry]”. In: IEEE International Solid-State Circuits Conference (ISSCC), 2003, pp. 20–23.

[MOP+11] Moreira, M.; Oliveira, B.; Pontes, J.; Moraes, F.; Calazans, N. “Adapting a C-element design flow for low power”. In: 18th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2011, pp. 45–48.

[MOPC11] Moreira, M.; Oliveira, B.; Pontes, J.; Calazans, N. “A 65nm standard cell set and flow dedicated to automated asynchronous circuits design”. In: IEEE International System-on-Chip Conference (SOCC), 2011, pp. 99–104.

[MOPC13a] Moreira, M. T.; Oliveira, C. H. M.; Porto, R. C.; Calazans, N. L. V. “Design of NCL gates with the ASCEnD flow”. In: IEEE 4th Latin American Symposium on Circuits and Systems (LASCAS), 2013, pp. 1–4.

[MOPC13b] Moreira, M. T.; Oliveira, C. H. M.; Porto, R. C.; Calazans, N. L. V. “NCL+: Return-to-one Null Convention Logic”. In: IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS), 2013, pp. 836–839.

[Mor10] Moreira, M. T. “Design and Implementation of a Standard Cell Library for Building Asynchronous ASICs”, Technical Report, Pontifícia Universidade Católica do Rio Grande do Sul. End of term work, 2010.

[Mor12] Moreira, M. T. “Contributions to the Design and Prototyping of GALS and Asynchronous Systems”, Master’s Thesis, Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, 2012.

[MOS] “The MOSIS Service”. Source: https://www.mosis.com/.

[MPC14] Moreira, M. T.; Pontes, J. J. H.; Calazans, N. L. V. “Tradeoffs between RTO and RTZ in WCHB QDI asynchronous design”. In: 15th International Symposium on Quality Electronic Design (ISQED), 2014, pp. 692–699.

[MRB+03] Maurine, P.; Rigaud, J.; Bouesse, F.; Sicard, G.; Renaudin, M. “Static Implementation of QDI Asynchronous Primitives”. In: International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2003, pp. 181–191.

[MTMC14] Moreira, M. T.; Trojan, G.; Moraes, F. G.; Calazans, N. L. V. “Spatially Distributed Dual-Spacer Null Convention Logic Design”, *Journal of Low Power Electronics*, vol. 10–3, 2014, pp. 313–320.

[Mur71] Muroga, S. “Threshold Logic and Its Applications”. New York: Wiley-Interscience, 1971.

[MY06] Murphy, J.; Yakovlev, A. “An Alternating Spacer AES Crypto-processor”. In: 32nd European Solid-State Circuits Conference (ESSCIRC), 2006, pp. 126–129.

[Mye01] Myers, C. J. “Asynchronous Circuit Design”. New York: John Wiley & Sons, Inc., 2001.

[Nas00] Nassif, S. “Delay variability: sources, impacts and trends”. In: IEEE International Solid-State Circuits Conference (ISSCC), 2000, pp. 368–369.

[Ngs] “Ngspice”. Source: http://ngspice.sourceforge.net/.

[NMRR13] Neutzling, A.; Martins, M. G. A.; Ribas, R. P.; Reis, A. I. “Synthesis of threshold logic gates to nanoelectronics”. In: Symposium on Integrated Circuits and Systems Design (SBCCI), 2013, pp. 1–6.

[NS11] Nowick, S.; Singh, M. “High-Performance Asynchronous Pipelines: An Overview”, *IEEE Design & Test of Computers*, vol. 28–5, Sep 2011, pp. 8–22.

[NS15a] Nowick, S.; Singh, M. “Asynchronous Design (Part 1): Overview and Recent Advances”, *IEEE Design & Test of Computers*, vol. 32–3, May/June 2015, pp. 5–18.

[NS15b] Nowick, S.; Singh, M. “Asynchronous Design (Part 2): Systems and Methodologies”, *IEEE Design & Test of Computers*, vol. 32–3, May/June 2015, pp. 19–28.

[OMAF10] Ouchet, F.; Morin-Allory, K.; Fesquet, L. “Delay Insensitivity Does Not Mean Slope Insensitivity!” In: IEEE Symposium on Asynchronous Circuits and Systems (ASYNC), 2010, pp. 176–184.

[ORM04] Omana, M.; Rossi, D.; Metra, C. “Model for Transient Fault Susceptibility of Combinational Circuits”, *Journal of Electronic Testing*, vol. 20–5, 2004, pp. 501–509.

[PAAS14] Parsan, F.; Al-Assadi, W.; Smith, S. “Gate Mapping Automation for Asynchronous NULL Convention Logic Circuits”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22–1, Jan 2014, pp. 99–112.

[PCV12] Pontes, J.; Calazans, N.; Vivet, P. “Adding Temporal Redundancy to Delay Insensitive Codes to Mitigate Single Event Effects”. In: 18th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2012, pp. 142–149.

[PCV13] Pontes, J.; Calazans, N.; Vivet, P. “Parity check for m-of-n delay insensitive codes”. In: IEEE 19th International On-Line Testing Symposium (IOLTS), 2013, pp. 157–162.

[PK93] Pramanick, A.; Kundu, S. “Design of scan-based path delay testable sequential circuits”. In: International Test Conference (ITC), 1993, pp. 962–971.

[PM05] Peng, S.; Manohar, R. “Efficient failure detection in pipelined asynchronous circuits”. In: 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT), 2005, pp. 484–493.

[PMMC10] Pontes, J.; Moreira, M. T.; Moraes, F. G.; Calazans, N. L. V. “Hermes-AA: A 65nm asynchronous NoC router with adaptive routing”. In: IEEE International System-on-Chip Conference (SOCC), 2010, pp. 493–498.

[PMMC11] Pontes, J.; Moreira, M.; Moraes, F.; Calazans, N. “Hermes-A – An Asynchronous NoC Router with Distributed Routing”. In: International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2011, pp. 150–159.

[PR90] Pramanick, A. K.; Reddy, S. M. “On the design of path delay fault testable combinational circuits”. In: 20th International Symposium on Fault-Tolerant Computing (FTCS), 1990, pp. 374–381.

[Pra07] Prakash, M. “Library Characterization and Static Timing Analysis of Asynchronous Circuits”, Master’s Thesis, University of Southern California, 2007.

[PS12a] Parsan, F.; Smith, S. “CMOS implementation comparison of NCL gates”. In: IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS), 2012, pp. 394–397.

[PS12b] Parsan, F. A.; Smith, S. C. “CMOS implementation of static threshold gates with hysteresis: A new approach”. In: IEEE/IFIP 20th International Conference on VLSI and System-on-Chip (VLSI-SoC), 2012, pp. 41–45.

[PS13] Palangpour, P.; Smith, S. “Sleep Convention Logic using partially slept function blocks”. In: IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS), 2013, pp. 17–20.

[PVT15] Pontes, J.; Vivet, P.; Thonnart, Y. “Two-phase protocol converters for 3D asynchronous 1-of-n data links”. In: 20th Asia and South Pacific Design Automation Conference (ASP-DAC), 2015, pp. 154–159.

[RAB+12] Riedlinger, R.; Arnold, R.; Biro, L.; Bowhill, B.; Crop, J.; Duda, K.; Fetzer, E.; Franza, O.; Grutkowski, T.; Little, C.; Morganti, C.; Moyer, G.; Munch, A.; Nagarajan, M.; Parks, C.; Poirier, C.; Repasky, B.; Roytman, E.; Singh, T.; Stefaniw, M. “A 32 nm, 3.1 Billion Transistor, 12 Wide Issue Itanium Processor for Mission-Critical Servers”, *IEEE Journal of Solid-State Circuits*, vol. 47–1, Jan 2012, pp. 177–193.

[RCN03] Rabaey, J. M.; Chandrakasan, A.; Nikolic, B. “Digital Integrated Circuits: A Design Perspective”. Prentice Hall, 2003.

[RF12] Renaudin, M.; Fonkoua, A. “Tiempo Asynchronous Circuits System Verilog Modeling Language”. In: 18th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2012, pp. 105–112.

[RMCF88] Rosenberger, F.; Molnar, C.; Chaney, T.; Fang, T.-P. “Q-modules: internally clocked delay-insensitive modules”, *IEEE Transactions on Computers*, vol. 37–9, Sep 1988, pp. 1005–1018.

[RRP+09] Ramaswamy, S.; Rockett, L.; Patel, D.; Danziger, S.; Manohar, R.; Kelly, C.; Holt, J.; Ekanayake, V.; Elftmann, D. “A radiation hardened reconfigurable FPGA”. In: IEEE Aerospace Conference, 2009, pp. 1–10.

[RST12] Reese, R.; Smith, S.; Thornton, M. “Uncle - An RTL Approach to Asynchronous Design”. In: 18th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2012, pp. 65–72.

[SCH+11] Seok, M.; Chen, G.; Hanson, S.; Wieckowski, M.; Blaauw, D.; Sylvester, D. “CAS-FEST 2010: Mitigating Variability in Near-Threshold Computing”, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1–1, Mar 2011, pp. 42–49.

[SCM+11] Soares, R. I.; Calazans, N. L. V.; Moraes, F. G.; Maurine, P.; Torres, L. “A Robust Architectural Approach for Cryptographic Algorithms Using GALS Pipelines”, *IEEE Design & Test of Computers*, vol. 28–5, Sep 2011, pp. 62–71.

[SF01a] Sparsø, J.; Furber, S. “Principles of Asynchronous Circuits Design - A systems Perspective”. London: Kluwer Academic Publishers, 2001.

[SF01b] Sutherland, I.; Fairbanks, S. “GasP: A Minimal FIFO Control”. In: 7th International Symposium on Asynchronous Circuits and Systems (ASYNC), 2001, pp. 46–53.

[SGY+09] Stevens, K. S.; Gebhardt, D.; You, J.; Xu, Y.; Vij, V.; Das, S.; Desai, K. “The Future of Formal Methods and {GALS} Design”, *Electronic Notes in Theoretical Computer Science*, vol. 245, 2009, pp. 115–134.

[SM88] Savir, J.; McAnney, W. H. “Random Pattern Testability of Delay Faults”, *IEEE Transactions on Computers*, vol. 37–3, Mar 1988, pp. 291–300.

[SMBY05] Sokolov, D.; Murphy, J.; Bystrov, A.; Yakovlev, A. “Design and analysis of dual-rail circuits for security applications”, *IEEE Transactions on Computers*, vol. 54–4, Apr 2005, pp. 449–460.

[SMT+15] Singhvi, A.; Moreira, M. T.; Tadros, R.; Calazans, N. L. V.; Beerel, P. A. “A Fine-Grained, Uniform, Energy-Efficient Delay Element for FD-SOI Technologies”. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2015, pp. 6.

[SN07] Singh, M.; Nowick, S. “MOUSETRAP: High-Speed Transition-Signaling Asynchronous Pipelines”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15–6, Jun 2007, pp. 684–=–698.

[Sok06] Sokolov, D. “Automated synthesis of asynchronous circuits using direct mapping for control and data paths”, Ph.D. Thesis, School of Electrical, Electronic & Computer Engineering, University of Newcastle Upon Tyne, 2006.

[SRG+01] Stevens, K.; Rotem, S.; Ginosar, R.; Beerel, P.; Myers, C.; Yun, K.; Koi, R.; Dike, C.; Roncken, M. “An asynchronous instruction length decoder”, *IEEE Journal of Solid-State Circuits*, vol. 36–2, Feb 2001, pp. 217–228.

[SS02] Saint, C.; Saint, J. “IC Layout Basics a Pratical Guide”. New York: McGraw-Hill, 2002.

[SS04] Saint, C.; Saint, J. “IC Mask Design Essential Layout Techniques”. New York: McGraw-Hill, 2004.

[SSJ13] Sridharan, A.; Sechen, C.; Jafari, R. “Low-voltage low-overhead asynchronous logic”. In: IEEE International Symposium on Low Power Electronics and Design (ISLPED), 2013, pp. 261–266.

[Sut89] Sutherland, I. E. “Micropipelines”, *Communications of the ACM*, vol. 32–6, Jun 1989, pp. 720–738.

[Syn] “Synopsys”. Source: http://www.synopsys.com/.

[Tay13] Taylor, M. B. “A Landscape of the New Dark Silicon Design Regime”, *IEEE Micro*, vol. 33–5, Sep 2013, pp. 8–19.

[TBV12] Thonnart, Y.; Beigné, E.; Vivet, P. “A Pseudo-Synchronous Implementation Flow for WCHB QDI Asynchronous Circuits”. In: 18th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2012, pp. 73–80.

[Tie] “Tiempo Secure”. Source: http://www.tiempo-secure.com/.

[TVC10] Thonnart, Y.; Vivet, P.; Clermidy, F. “A fully-asynchronous low-power framework for GALS NoC integration”. In: Design, Automation & Test in Europe Conference Exhibition (DATE), 2010, 2010, pp. 33–38.

[Vah10] Vahid, F. “Digital Design with RTL Design, Verilog and VHDL”. Wiley Publishing, 2010, 2nd ed..

[vBKR+91] van Berkel, K.; Kessels, J.; Roncken, M.; Saeijs, R.; Schalij, F. “The VLSI-programming language Tangram and its translation into handshake circuits”. In: European Conference on Design Automation (EDAC), 1991, pp. 384–389.

[Ver88] Verhoeff, T. “Delay-insensitive codes - an overview”, *Distributed Computing*, vol. 3–1, 1988, pp. 1–8.

[VXVZ06] Vaidyanathan, B.; Xie, Y.; Vijaykrishnan, N.; Zheng, H. “Soft Error Analysis and Optimizations of C-elements in Asynchronous Circuits”. In: 2nd Workshop on System Effects of Logic Soft Errors, 2006, pp. 1–4.

[Wav] “Wave Semiconductor”. Source: http://www.wavesemi.com/.

[WH10] Weste, N.; Harris, D. “CMOS VLSI Design: A Circuits and Systems Perspective”. Pearson, 2010.

[Wim14] Wimer, S. “Easy and difficult exact covering problems arising in VLSI power reduction by clock gating”, *Discrete Optimization*, vol. 14, 2014, pp. 104–110.

[XD96] Xi, J.; Dai, W.-M. “Useful-skew clock routing with gate sizing for low power design”. In: 33rd Design Automation Conference (DAC), 1996, pp. 383–388.

[YG08] Yan, C.; Greenstreet, M. “Verifying an Arbiter Circuit”. In: International Conference on Formal Methods in Computer-Aided Design (FMCAD), 2008, pp. 9.

[YR06] Yahya, E.; Renaudin, M. “QDI Latches Characteristics and Asynchronous Linear-Pipeline Performance Analysis”. In: International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2006, pp. 583–592.

[YS10] Yancey, S.; Smith, S. “A differential design for C-elements and NCL gates”. In: 53rd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), 2010, pp. 632–635.

[ZHM+15] Zhang, Y.; Heck, L.; Moreira, M.; Calazans, N. L. V.; Beerel, P. A. “Design and Analysis of Testable Mutual Exclusion Elements”. In: 21st IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2015, pp. 124–131.

[Zie14] Ziesemer Jr., A. M. “Automatic layout synthesis of transistor networks”, Ph.D. Thesis, Universidade Federal do Rio Grande do Sul. In Portuguese, 2014.

[ZR14] Ziesemer Jr., A.; Reis, R. “Simultaneous Two-Dimensional Cell Layout Compaction Using MILP with ASTRAN”. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2014, pp. 350–355.

[ZRM+14a] Ziesemer Jr., A.; Reis, R.; Moreira, M. T.; Arendt, M. E.; Calazans, N. L. V. “Automatic layout synthesis with ASTRAN applied to asynchronous cells”. In: IEEE 5th Latin American Symposium on Circuits and Systems (LASCAS), 2014, pp. 1–4.

[ZRM+14b] Ziesemer Jr., A.; Reis, R.; Moreira, M. T.; Arendt, M. E.; Calazans, N. L. V. “A design flow for physical synthesis of digital cells with astran”. In: 24th Great Lakes Symposium on VLSI (GLSVLSI), 2014, pp. 245–246.

[ZSD10] Zhou, L.; Smith, S.; Di, J. “Bit-Wise MTNCL: An ultra-low power bit-wise pipelined asynchronous circuit design methodology”. In: 53rd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), 2010, pp. 217–220.

APPENDIX A – EXAMPLE OF USAGE OF THE ASCEND-A FLOW

This Appendix presents a demonstration on the usage of the ASCEnD-A flow. It relies on the tools and templates available in the svn of the design flow:

https://corfu.pucrs.br/svn/ascend/design-flow/tags/ascend-a-v1.

The tools and scripts used here are in the *tools* and *scripts* folders of the repository. Also, for the case study we rely on the templates available in the *gift* folder of the repository.

A.1 Templates Generation

The generation of templates to be used in the ASCEnD-A flow relies on the availability of a template library. Such library in our example is available in the *gift/ncl-template-library/schematic* folder in the repository of ASCEnD-A. The cells in this library must follow the following conventions:

- NMOS transistors use “NFET” as their model;
- PMOS transistors use “PFET” as their model;
- The width of driving transistors is defined as “ninv” for NMOS transistors and “pinv” for PMOS transistors;
- The width of feedback transistors is defined as “nfback” for NMOS transistors and “pfback” for PMOS transistors;
- The width of transistors to be dimensioned is defined as “nparam” for NMOS transistors and “pparam” for PMOS transistors;
- The length of all transistors is defined as “length”.

A set of comments in the end of the Spice description are also required to allow the generation of configuration files for the remaining tools of the flow. These comments are:

- *ROGen:T *X*, where *X* defines the threshold of a logic function, used for NCL cells;
- *ROGen:I *X*, where *X* defines the number of inputs of a cell;
- *ROGen:P *list*, where the list defines the direction/configuration of each pin of the cell, which can be *in* (for inputs), *out* (for outputs), *gnd* (for ground reference), *gnds* (for body ground reference), *vdd* (for supply voltage reference) and *vdds* (for body voltage reference);

Note that such conventions were defined as we devised the flow. The following schematic is an example of cell of this library that will be used as a baseline in this document to explore the flow:

```
.subckt EXAMPLE_CELL A B Q GND GNDS VDD VDDS
MPS00 VDD  A    pl00 VDDS PFET W=pparam L=length
MPS01 pl00 B    PREQ VDDS PFET W=pparam L=length
MNS00 PREQ A    nl00 GNDS NFET W=nparam L=length
MNS01 nl00 B    GND  GNDS NFET W=nparam L=length
MPI00 VDD  PREQ Q    VDDS PFET W=pinv   L=length
MNI00 GND  PREQ Q    GNDS NFET W=ninv   L=length
MPF00 FBP  Q    PREQ VDDS PFET W=pfback L=length
MPF01 FBP  A    VDD  VDDS PFET W=pfback L=length
MPF02 FBP  B    VDD  VDDS PFET W=pfback L=length
MNF00 FBN  Q    PREQ GNDS NFET W=nfback L=length
MNF01 FBN  A    GND  GNDS NFET W=nfback L=length
MNF02 FBN  B    GND  GNDS NFET W=nfback L=length
.ends
*ROGen:T 2
*ROGen:I 2
*ROGen:P in in out gnd gnds vdd vdds
```

This cell is a 2-input static C-element. Next, the following variables must be set, for configuring the target technology and building the environment to size transistors, as explored in Section 4.2.1:

```
_TECH_="st65"
_MODELS_PATH_="../models/st65.scs"
_MODELS_LANG_="spectre"
_NAND_="nand in in out gnd gnds vdd vdds"
_NAND_FILE_="../models/nandx"
_INV_="inv in out gnd gnds vdd vdds"
_INV_FILE_="../models/invx"
_TEMP_="25"
_VDD_="1"
_RISE_="0.4"
_FALL_="0.6"
_LOG1_="0.9"
_LOG0_="0.1"
_LEAK_START_="0.5"
_LEAK_END_="1"
_DYN_START_="1"
_DYN_END_="3"
_DYN_SWITCH_="3"
```

```
_SIM_TIME_="10"
_CELLS_DIR_="~/ascend-a/gift/ncl-template-library/schematic"
_STEP_N_="0.05"
_STEP_P_="0.05"
_N_SINGLE_FING_="0.700"
_P_SINGLE_FING_="0.900"
_METRIC_UNIT_="none"
_MIN_N_="0.135 0.250 0.400 0.600"
_MAX_N_="0.500 1.000 1.600 2.400"
_MIN_P_="0.200 0.350 0.600 0.800"
_MAX_P_="0.800 1.400 2.200 3.000"
_DRIVING_STRENGTHS_="2 4 7 9"
_MIN_N_NI_="0.135 0.135 0.200 0.250"
_MAX_N_NI_="0.300 0.350 0.400 0.500"
_MIN_P_NI_="0.135 0.135 0.250 0.350"
_MAX_P_NI_="0.400 0.450 0.500 0.700"
_DRIVING_STRENGTHS_NI_="2 4 7 9"
_OUT_INV_N_="0.200 0.390 0.580 0.780"
_OUT_INV_P_="0.280 0.550 0.830 1.100"
```

With these variables set, the designer can execute the *generate-template-library.sh* script. The *_TECH_* variable defines the name given to the target technology, the *_MODELS_PATH_* variable defines the path to the file containing the transistor models of the target technology and *_MODELS_LANG_* defines what language was used in these models (Spice or Spectre). The *_NAND_*, *_NAND_FILE_*, *_INV_* and *_INV_FILE_* variables define the name interface of the required NAND gate, the path to its Spice description, the name and interface of the required inverter and the path to its Spice description, respectively. Variables *_TEMP_* and *_VDD_* defines the temperature and voltage to be used during transistors sizing. *_RISE_*, *_FALL_*, *_LOG1_* and *_LOG0_* define the thresholds for rising and falling transitions and for valid logic 1s and 0s, respectively. Variables *_LEAK_START_*, *_LEAK_END_*, *_DYN_START_*, *_DYN_END_*, *_DYN_SWITCH_* and *_SIM_TIME_* define the start and stop instants to measure leakage power, the start and stop instants to measure dynamic power, the transition during which dynamic power is to be measured and total simulation time, respectively. Variable *_CELLS_DIR_* defines a folder containing the template library. *_STEP_N_*, *_STEP_P_*, *_N_SINGLE_FING_*, *_P_SINGLE_FING_* and *_METRIC_UNIT_* define the step in which NMOS and PMOS transistors are to be increased, the maximum sizes for NMOS and PMOS transistors for a given cell architecture and the metric unit used in the target design kit, respectively. Variables *_MIN_N_*, *_MAX_N_*, *_MIN_P_*, *_MAX_P_* and *_DRIVING_STRENGTHS_* define the minimum and maximum sizes for NMOS transistors, the minimum and maximum sizes for PMOS transistors and the corresponding driving strengths, respectively. Variables *_MIN_N_NI_*, *_MAX_N_NI_*, *_MIN_P_NI_*,

*_MAX_P_NI_* and *_DRIVING_STRENGTHS_NI_* define the same parameters but for non-inverted (or positive unate) functions. *_OUT_INV_N_* and *_OUT_INV_P_* define the dimension of the NMOS and PMOS transistors, respectively for driving inverters, relative to those driving strengths defined in *_DRIVING_STRENGTHS_*.

The output of this script is a set of configuration files for ROGen, one of the transistor sizing tools. For each cell, 2 configuration files are generated, one is used for the configuration of ROGen and one is the schematic employed by ROGen during simulation. For example, for our case study cell (now with an X4 driving strength), the resulting configuration file is:

```
$tech st65
$models ../models/st65.scs
$models_lang spectre
$nand nand in in out gnd gnds vdd vdds
$nand_file ../models/nandx4
$inv inv in out gnd gnds vdd vdds
$inv_file ../models/invx4
$differential 0
$protocol 0
$threshold 2
$cell_num_in 2
$cell_config n
$cell_rst 0
$cell_set 1
$cell_inv 0
$cell EXAMPLE_CELLX4  in in out gnd gnds vdd vdds
$cell_file rogen_EXAMPLE_CELLX4.sp
$cell_num 5
$volt 1
$leak_start_time 0.5
$leak_end_time 1
$dyn_start_time 1
$dyn_end_time 3
$dyn_switch_event 3
$time_unit n
$sim_step 0.001
$sim_time 10
$in_slew 0.01
$rise_trans 0.4
$fall_trans 0.6
$log_1 0.9
$log_0 0.1
```

```
$max_1fing_n 0.700
$max_1fing_p 0.900
$min_n 0.135
$max_n 0.350
$n_step 0.05
$min_p 0.135
$max_p 0.450
$p_step 0.05
$metric_unit none
$temp 25
$fan_out 4
```

The schematic generated for transistor sizing is:

```
.subckt EXAMPLE_CELLX4 A B Q GND GNDS VDD VDDS
MPS00 VDD  A    pl00 VDDS PSVTGP W=pparam nfing=pnum L=0.06
MPS01 pl00 B    PREQ VDDS PSVTGP W=pparam nfing=pnum L=0.06
MNS00 PREQ A    nl00 GNDS NSVTGP W=nparam nfing=nnum L=0.06
MNS01 nl00 B    GND  VDDS NSVTGP W=nparam nfing=nnum L=0.06
MPI00 VDD  PREQ Q    VDDS PSVTGP W=0.550 L=0.06
MNI00 GND  PREQ Q    VDDS NSVTGP W=0.390 L=0.06
MPF00 FBP  Q    PREQ VDDS PSVTGP W=0.135 L=0.06
MPF01 FBP  A    VDD  VDDS PSVTGP W=0.135 L=0.06
MPF02 FBP  B    VDD  VDDS PSVTGP W=0.135 L=0.06
MNF00 FBN  Q    PREQ GNDS NSVTGP W=0.135 L=0.06
MNF01 FBN  A    GND  GNDS NSVTGP W=0.135 L=0.06
MNF02 FBN  B    GND  GNDS NSVTGP W=0.135 L=0.06
*Transistors count: NMOS 6 PMOS 6 TOTAL 12
.ends
*ROGen:T 2
*ROGen:I 2
*ROGen:P in in out gnd gnds vdd vdds
```

The designer can then execute the *generate-behavioral-models.sh* script. This script will generate the behavioral Verilog views of the cells under design. For our case study cell, the result is:

```
module EXAMPLE_CELLX4 (Q, A, B);
input A, B;
output Q;
u_c2 i1 (qi, A, B);
buf i2 (Q, qi);
specify
        (A => Q) = (0.1,0.1);
        (B => Q) = (0.1,0.1);
```

```
endspecify
endmodule
```

A.2 Cell Sizing

As explored in Section 4.2.2, the next step is to use the generated configuration files to dimension the generated Spice circuit. Using the configuration file with ROGen, the following simulation environment is created:

```
* Sizing Simulation (Cell Ring)
* Cell under specification: EXAMPLE_CELLX4
* Description automatically generated with ROGen v1.0
* Dec  1 2015 19:57:56 HOSTNAME: matheus-moreira-xps

simulator lang=spectre insensitive=no
include "../models/st65.scs"
simulator lang=spice

*******************************************************************************
*CKT: inv
*******************************************************************************

.SUBCKT inv A Z gnd gnds vdd vdds
*.PININFO A:I Z:O gnd:B gnds:B vdd:B vdds:B
MM64 Z A gnd gnds nsvtgp accurateFlow=0 l=0.06 m=1 nfing=1 ngcon=1 w=0.39
MM65 Z A vdd vdds psvtgp accurateFlow=0 l=0.06 m=1 nfing=1 ngcon=1 w=0.55
.ENDS inv

*******************************************************************************
*CKT: nand
*******************************************************************************

.SUBCKT nand A B Z gnd gnds vdd vdds
*.PININFO A:I B:I Z:O gnd:B gnds:B vdd:B vdds:B
MM64 net247 A gnd gnds nsvtgp accurateFlow=0 l=0.06 m=1 nfing=1 w=0.39
MM65 Z A vdd vdds psvtgp accurateFlow=0 l=0.06 m=1 nfing=1 w=0.55
MM66 Z B vdd vdds psvtgp accurateFlow=0 l=0.06 m=1 nfing=1 w=0.55
MM67 Z B net247 gnds nsvtgp accurateFlow=0 l=0.06 m=1 nfing=1 w=0.39
.ENDS nand

*******************************************************************************
```

```
*CKT: EXAMPLE_CELLX4
********************************************************************************

********************************************************************************
* NCL Gate 2w11, threshold: 2 weights [1, 1] inputs 2
* Generated by Matheus Moreira - GAPH/PUCRS - Porto Alegre, Brazil
* 2014-04-24 18:09:15
********************************************************************************
.subckt EXAMPLE_CELLX4 A B Q GND GNDS VDD VDDS
MPS00 VDD  A    pl00 VDDS PSVTGP W=pparam nfing=pnum L=0.06
MPS01 pl00 B    PREQ VDDS PSVTGP W=pparam nfing=pnum L=0.06
MNS00 PREQ A    nl00 GNDS NSVTGP W=nparam nfing=nnum L=0.06
MNS01 nl00 B    GND  GNDS NSVTGP W=nparam nfing=nnum L=0.06
MPI00 VDD  PREQ Q    VDDS PSVTGP W=0.550 L=0.06
MNI00 GND  PREQ Q    GNDS NSVTGP W=0.390 L=0.06
MPF00 FBP  Q    PREQ VDDS PSVTGP W=0.135 L=0.06
MPF01 FBP  A    VDD  VDDS PSVTGP W=0.135 L=0.06
MPF02 FBP  B    VDD  VDDS PSVTGP W=0.135 L=0.06
MNF00 FBN  Q    PREQ GNDS NSVTGP W=0.135 L=0.06
MNF01 FBN  A    GND  GNDS NSVTGP W=0.135 L=0.06
MNF02 FBN  B    GND  GNDS NSVTGP W=0.135 L=0.06
.ends
*ROGen:T 2
*ROGen:I 2
*ROGen:P in in out gnd gnds vdd vdds

********************************************************************************
*CKT: RING(6)
********************************************************************************

X0 IN ni00 ni10 0 0 vdd_fo vdd_fo nand
XI0 ni10 ni100 0 0 vdd_fo vdd_fo inv
XI1 ni10 ni101 0 0 vdd_fo vdd_fo inv
X1 ni10 ni10 n2 0 0 vdd vdd EXAMPLE_CELLX4
XI2 n2 ni200 0 0 vdd_fo vdd_fo inv
XI3 n2 ni201 0 0 vdd_fo vdd_fo inv
X2 n2 n2 n3 0 0 vdd vdd EXAMPLE_CELLX4
XI4 n3 ni300 0 0 vdd_fo vdd_fo inv
XI5 n3 ni301 0 0 vdd_fo vdd_fo inv
X3 n3 n3 n4 0 0 vdd vdd EXAMPLE_CELLX4
XI6 n4 ni400 0 0 vdd_fo vdd_fo inv
XI7 n4 ni401 0 0 vdd_fo vdd_fo inv
```

```
X4 n4 n4 n5 0 0 vdd vdd EXAMPLE_CELLX4
XI8 n5 ni500 0 0 vdd_fo vdd_fo inv
XI9 n5 ni501 0 0 vdd_fo vdd_fo inv
X5 n5 n5 ni00 0 0 vdd vdd EXAMPLE_CELLX4
XI10 ni00 ni600 0 0 vdd_fo vdd_fo inv
XI11 ni00 ni601 0 0 vdd_fo vdd_fo inv

********************************************************************************
*CKT: RING(st1)
********************************************************************************

XL1 ln1 ln1 ln2 0 0 vdd1 vdd1 EXAMPLE_CELLX4

********************************************************************************
*CKT: RING(st2)
********************************************************************************

XXL1 ln1 ln1 ln2 0 0 vdd1 vdd1 EXAMPLE_CELLX4

********************************************************************************
*Initial Condition: ln
********************************************************************************

.ic ln1 = 1

********************************************************************************
*Initial Condition: nln
********************************************************************************

.ic nln1 = 0

********************************************************************************

.temp 25
vdd      vdd      0        dc       1
vdd_fo   vdd_fo   0        dc       1
vdd0     vdd0     0        dc       1
vdd1     vdd1     0        dc       1
vnand    IN       0        pwl      ( 0 0 1n 0 1.01n 1 )

.measure tran perfr trig v(n4) val = 0.4 rise = 3
+ targ v(n4) val = 0.4 rise = 4
```

```
.measure tran propf trig v(n3) val = 0.6 fall = 3
+ targ v(n4) val = 0.6 fall = 3

.measure tran propr trig v(n3) val = 0.4 rise = 3
+ targ v(n4) val = 0.4 rise = 3

.measure tran tranr trig v(n4) val = 0.1 rise = 3
+ targ v(n4) val = 0.9 rise = 3

.measure tran tranf trig v(n4) val = 0.9 fall = 3
+| targ v(n4) val = 0.1 fall = 3

.measure tran   fr              param = '1/(perfr)'

.measure tran   pt_rl   param = 'propr/propf'

.measure tran   leakpwrn        AVG i(vdd0) from=0.5ns to=1ns
.measure tran   leakpwr AVG i(vdd1) from=0.5ns to=1ns
.measure tran   dynpwr  AVG i(vdd) from=1ns to=3ns

.measure tran   sizep   param = 'pparam'
.measure tran   sizen   param = 'nparam'
.measure tran   nump    param = 'pnum'
.measure tran   numn    param = 'nnum'

.tran 0.001n 10n

.param nnum=1   pnum=1  nparam=0.135    pparam=0.185
.alter
.param nnum=1   pnum=1  nparam=0.135    pparam=0.235
.alter
.param nnum=1   pnum=1  nparam=0.135    pparam=0.285
.alter
.param nnum=1   pnum=1  nparam=0.135    pparam=0.335
.alter
.param nnum=1   pnum=1  nparam=0.135    pparam=0.385
.alter
.param nnum=1   pnum=1  nparam=0.135    pparam=0.435
.alter
.param nnum=1   pnum=1  nparam=0.185    pparam=0.235
.alter
```

```
.param nnum=1    pnum=1   nparam=0.185     pparam=0.285
.alter
.param nnum=1    pnum=1   nparam=0.185     pparam=0.335
.alter
.param nnum=1    pnum=1   nparam=0.185     pparam=0.385
.alter
.param nnum=1    pnum=1   nparam=0.185     pparam=0.435
.alter
.param nnum=1    pnum=1   nparam=0.235     pparam=0.285
.alter
.param nnum=1    pnum=1   nparam=0.235     pparam=0.335
.alter
.param nnum=1    pnum=1   nparam=0.235     pparam=0.385
.alter
.param nnum=1    pnum=1   nparam=0.235     pparam=0.435
.alter
.param nnum=1    pnum=1   nparam=0.285     pparam=0.335
.alter
.param nnum=1    pnum=1   nparam=0.285     pparam=0.385
.alter
.param nnum=1    pnum=1   nparam=0.285     pparam=0.435
.alter
.param nnum=1    pnum=1   nparam=0.335     pparam=0.385
.alter
.param nnum=1    pnum=1   nparam=0.335     pparam=0.435

.END
```

This environment is then simulated using Spectre from Cadence, which generates an output file containing the measurements done during simulation. This output file contains the information in blocks defined for each transistor sizing variation, as in:

```
Measurement Name    :  transient1
Analysis Type       :  tran
dynpwr              =  -6.74059e-05
fr                  =  1.48368e+09
leakpwr             =  -3.74358e-08
leakpwrn            =  0
numn                =  1
nump                =  1
perfr               =  6.74e-10
propf               =  6.83518e-11
propr               =  5.71404e-11
```

```
pt_rl           =  0.835975
sizen           =  0.135
sizep           =  0.185
tranf           =  3.75135e-11
tranr           =  4.28003e-11
```

Using CeS, the designer reads this file and defines a cost function. With this function, CeS will select the best transistor sizes, depending on the collected results. As explored in Section 4.2.2, for this cost function, a designer can use the following parameters:

- Dynamic power: measured as the average current on the power source while the ring is oscillating, multiplied by the operating voltage;
- Leakage power: measured as the average current on the power source while the ring is quiescent, multiplied by the operating voltage;
- Rise transition delay: measured as the time it takes for the output of a specific cell on the ring to perform a full swing from logic 0 to 1 (a rising transition);
- Fall transition delay: measured as the time it takes for the output of a specific cell on the ring to perform a full swing from logic 1 to 0 (a falling transition);
- Rise propagation delay: measured as the time it takes for a rising transition in the input of a specific cell on the ring to propagate to the output of the cell;
- Fall propagation delay: measured as the time it takes for a falling transition in the input of a specific cell on the ring to propagate to the output of the cell;
- Ring propagation delay: measured as the time it takes for a rising transition to propagate through the whole ring.

For example, executing CeS with a cost function that searches the best trade off between propagation delays and dynamic power, for example $(1/(((propr+propf)/2)*dynpwr))$, the result is:

```
Name =  transient1 fr =  1.48368e+09 dynpwr =  -6.74059e-05 leakpwr =
   -3.74358e-08 leakpwrn =  0 numn =  1 nump =  1 sizen =  0.135 sizep =
    0.185 result = 2.36437e+14
Name =  transient7 fr =  1.6207e+09 dynpwr =  -7.49221e-05 leakpwr =
   -3.78261e-08 leakpwrn =  0 numn =  1 nump =  1 sizen =  0.185 sizep =
    0.235 result = 2.34148e+14
Name =  transient12 fr =  1.72071e+09 dynpwr =  -8.04355e-05 leakpwr =
   -3.82426e-08 leakpwrn =  0 numn =  1 nump =  1 sizen =  0.235 sizep =
    0.285 result = 2.32915e+14
```

```
Name = transient16 fr = 1.79497e+09 dynpwr = -8.44904e-05 leakpwr =
    -3.86645e-08 leakpwrn = 0 numn = 1 nump = 1 sizen = 0.285 sizep =
     0.335 result = 2.32328e+14
Name = transient19 fr = 1.85048e+09 dynpwr = -8.94429e-05 leakpwr =
    -3.90845e-08 leakpwrn = 0 numn = 1 nump = 1 sizen = 0.335 sizep =
     0.385 result = 2.27046e+14
Name = transient20 fr = 1.87897e+09 dynpwr = -9.31776e-05 leakpwr =
    -3.94204e-08 leakpwrn = 0 numn = 1 nump = 1 sizen = 0.335 sizep =
     0.435 result = 2.21683e+14
```

The next step is to execute the Spice generator script, *post_ces.sh*. This script generates a dimensioned Spice:

```
*********************************************************************
* Dimensioned by ASCEnD
* Fri Dec 1 20:04:52 PST 2015
* '( 1 / ( ( ( propr + propf ) / 2 ) * dynpwr ) )'
*********************************************************************
.subckt ST_NCL2W11OF2X4 A B Q GND GNDS VDD VDDS
MPS00 VDD  A    pl00 VDDS PSVTGP W=0.185 nfing=1 L=0.06
MPS01 pl00 B    PREQ VDDS PSVTGP W=0.185 nfing=1 L=0.06
MNS00 PREQ A    nl00 GNDS NSVTGP W=0.135 nfing=1 L=0.06
MNS01 nl00 B    GND  GNDS NSVTGP W=0.135 nfing=1 L=0.06
MPI00 VDD  PREQ Q    VDDS PSVTGP W=0.550 L=0.06
MNI00 GND  PREQ Q    GNDS NSVTGP W=0.390 L=0.06
MPF00 FBP  Q    PREQ VDDS PSVTGP W=0.135 L=0.06
MPF01 FBP  A    VDD  VDDS PSVTGP W=0.135 L=0.06
MPF02 FBP  B    VDD  VDDS PSVTGP W=0.135 L=0.06
MNF00 FBN  Q    PREQ GNDS NSVTGP W=0.135 L=0.06
MNF01 FBN  A    GND  GNDS NSVTGP W=0.135 L=0.06
MNF02 FBN  B    GND  GNDS NSVTGP W=0.135 L=0.06
.ends
```

A.3 Cell Layout

The next step of the flow is the layout generation. To do so, we read the sized schematic with the scripts *VirtuosoMapping* and *ImpSchematic*. The results is a schematic in the CDB for the target library. Figure A.1 shows the resulting schematic of our case study cell.

Following this, the layout of the cell is generated using ASTRAN. To do so, two scripts must be executed, *AstranMapping* and *runAstran*. The former ensures that the

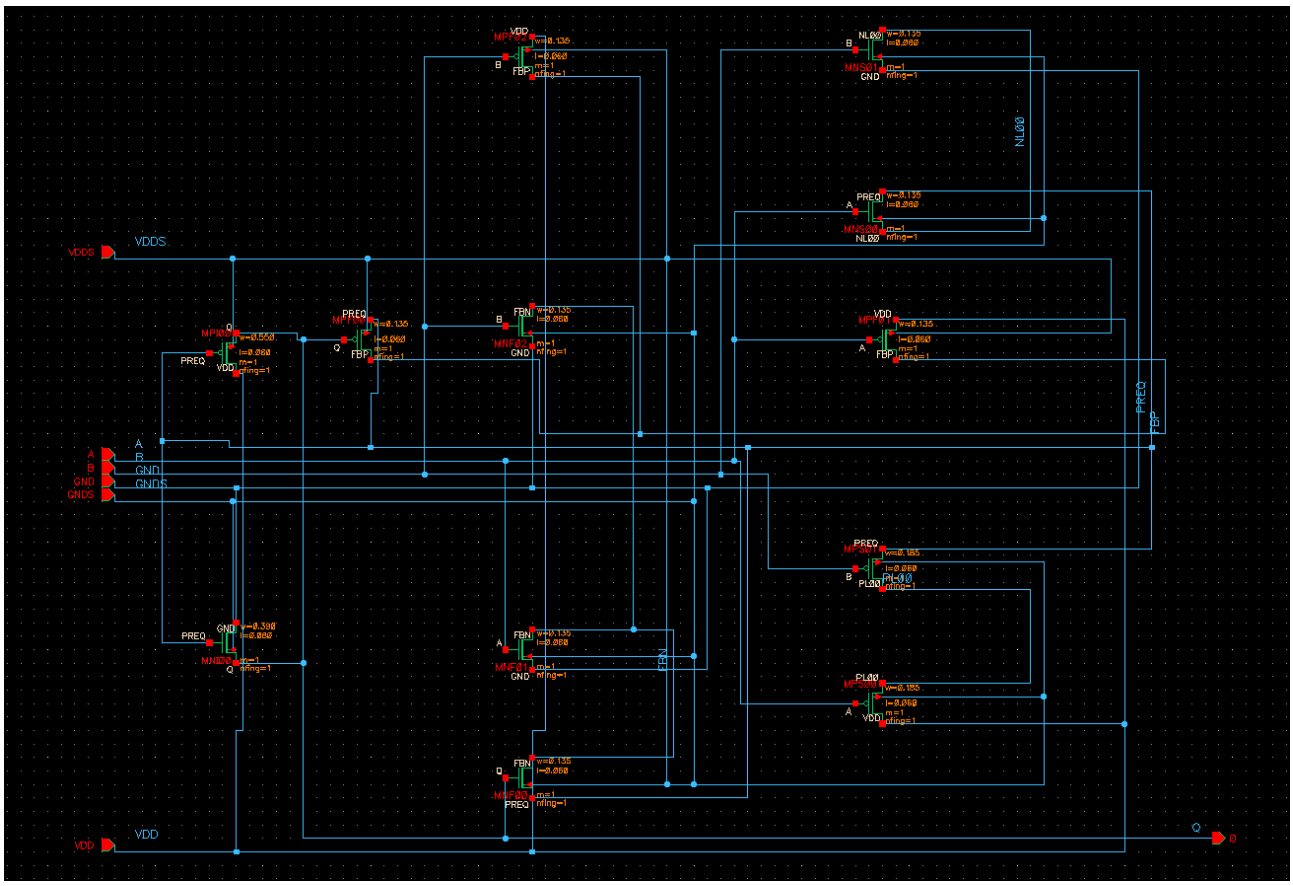


Figure A.1 – Schematic of the case study 2-input C-element imported to CDB.

schematic is compatible with the tool, which has the assumption that all transistors are modelled as PMOS or NMOS devices and requires units after the dimensions of the Ls and Ws of transistors. Moreover, it does not support any other parameters other than W and L. The following example, generated for our case study cell, is a schematic compatible with ASTRAN:

```
* ASTRAN MAPPING
.SUBCKT EXAMPLE_CELLX4 A B Q GND GNDS VDD VDDS
MPS00 VDD  A    PL00 VDDS PMOS W=185n L=60n
MPS01 PL00 B    PREQ VDDS PMOS W=185n L=60n
MNS00 PREQ A    NL00 GNDS NMOS W=135n L=60n
MNS01 NL00 B    GND  GNDS NMOS W=135n L=60n
MPI00 VDD  PREQ Q    VDDS PMOS W=550n L=60n
MNI00 GND  PREQ Q    GNDS NMOS W=390n L=60n
MPF00 FBP  Q    PREQ VDDS PMOS W=135n L=60n
MPF01 FBP  A    VDD  VDDS PMOS W=135n L=60n
MPF02 FBP  B    VDD  VDDS PMOS W=135n L=60n
MNF00 FBN  Q    PREQ GNDS NMOS W=135n L=60n
MNF01 FBN  A    GND  GNDS NMOS W=135n L=60n
MNF02 FBN  B    GND  GNDS NMOS W=135n L=60n
```

```
.ENDS
```

The *runAstran* script executes the tool, reading the generated schematic and, as output, creates a CIF file, containing the layout, and a C2C file containing information for importing the layout into the CDB. The generated C2C for our case study cell was:

```
cifInKeys = list( nil
  'runDir                 "./"
  'inFile                 "EXAMPLE_CELLX4.cif"
  'primaryCell            "EXAMPLE_CELLX4"
  'libName                "TEST"
  'dataDump               ""
  'techfileName           "tech.lib"
  'scale                  0.0025
  'units                  "micron"
  'errFile                "PIPO.LOG"
  'cellMapTable           ""
  'layerTable             "CIFLTable.txt"
  'textFontTable          ""
  'userSkillFile          ""
  'hierDepth              32
  'maxVertices            1024
  'checkPolygon           nil
  'snapToGrid             t
  'caseSensitivity        "upper"
  'zeroWireToLine         "lines"
  'wireToPathStyle        "extend"
  'roundFlashToEllipse    "ellipse"
  'skipUndefinedLPP       nil
  'reportPrecision        nil
  'runQuiet               nil
  'saveAtTheEnd           nil
  'noWriteExistCell               nil
  'NOUnmappingLayerWarning                nil
 )
```

Figure A.2 shows the generated CIF for our case study cell.

With the CIF and the C2C file, another script, *ImpLayout*, is used for importing the layout in the CDB. Figure A.3 shows the imported layout for our case study cell.

The layout is then verified using DRC and LVS tools. Some minor errors, when they occur, can be identified by the DRC tool that captures a current limitation of ASTRAN. Such errors are violations caused by the approach of placing multiple squares and rectangles for routing wires, which cause the generation of corners in internal nets. After eventual (manual)

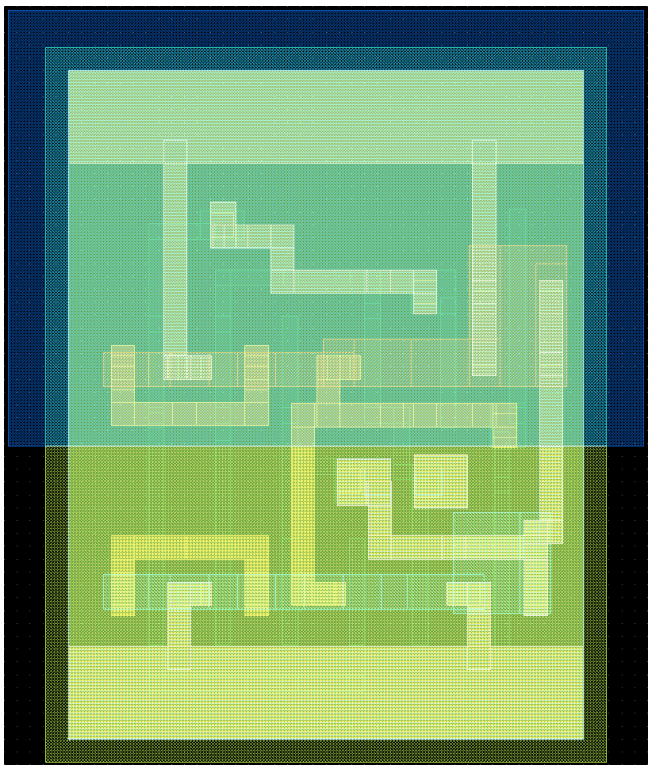


Figure A.2 – CIF of the generated layout for the case study 2-input C-element.

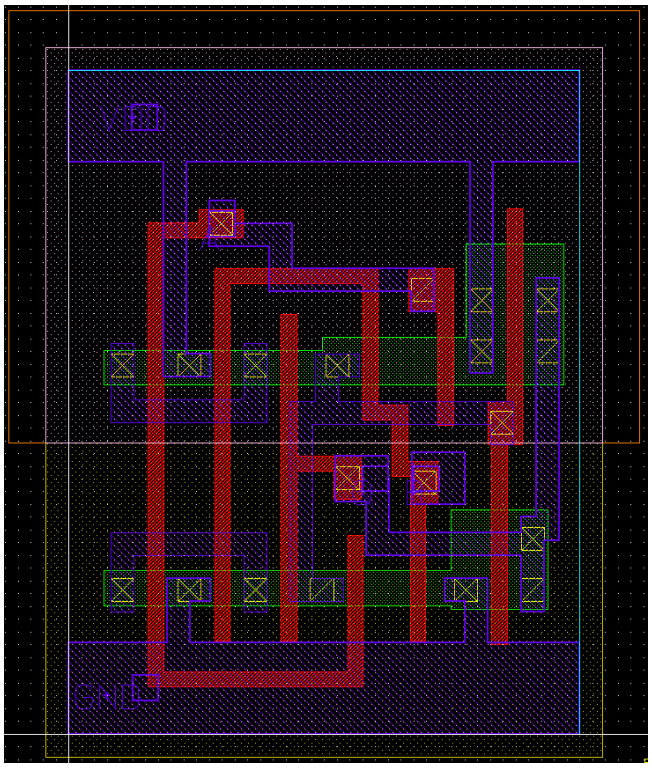


Figure A.3 – Imported layout of the generated layout for the case study 2-input C-element.

fixes in this layout, an abstract view is generated using the Cadence Abstract Generator tool. Figure A.4 shows the generated abstract view for our case study cell.

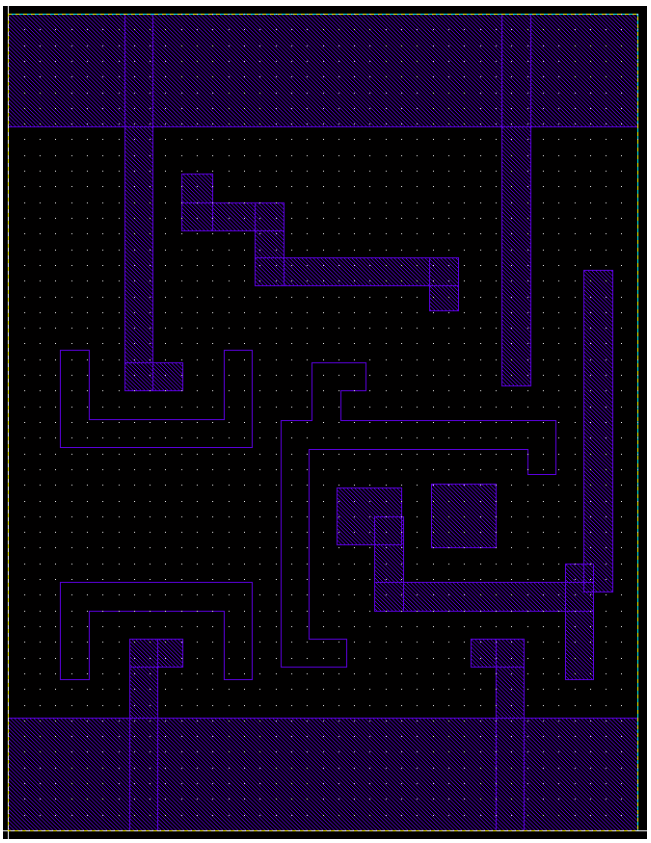


Figure A.4 – Generated abstract view for the case study 2-input C-element.

From this abstract view a LEF file can be exported for enabling place and route tasks. For our case study, we obtained the following LEF:

```
MACRO EXAMPLE_CELLX4
    CLASS CORE ;
    FOREIGN EXAMPLE_CELLX4 0 0 ;
    ORIGIN 0.000 0.000 ;
    SIZE 2.000 BY 2.600 ;
    SYMMETRY X Y ;
    SITE CoreSite ;
    PIN Q
        DIRECTION OUTPUT ;
        PORT
        LAYER M1 ;
        RECT 1.830 0.760 1.920 1.785 ;
        RECT 1.165 0.700 1.860 0.790 ;
        RECT 1.770 0.480 1.860 0.850 ;
        RECT 1.165 0.700 1.255 1.000 ;
        RECT 1.045 0.910 1.250 1.090 ;
        END
    END Q
```

```
 PIN B
      DIRECTION INPUT ;
      PORT
      LAYER M1 ;
      RECT  1.345 0.900 1.550 1.105 ;
      END
 END B
 PIN A
      DIRECTION INPUT ;
      PORT
      LAYER M1 ;
      RECT  0.785 1.735 1.430 1.825 ;
      RECT  1.340 1.655 1.430 1.825 ;
      RECT  0.550 1.910 0.875 2.000 ;
      RECT  0.785 1.735 0.875 2.000 ;
      RECT  0.550 1.910 0.650 2.090 ;
      END
 END A
 PIN GND
      DIRECTION INPUT ;
      USE GROUND ;
      SHAPE ABUTMENT ;
      PORT
      LAYER M1 ;
      RECT  0.000 0.000 2.000 0.360 ;
      RECT  1.470 0.520 1.640 0.610 ;
      RECT  1.550 0.000 1.640 0.610 ;
      RECT  0.385 0.520 0.555 0.610 ;
      RECT  0.385 0.000 0.475 0.610 ;
      END
 END GND
 PIN VDD
      DIRECTION INPUT ;
      USE POWER ;
      SHAPE ABUTMENT ;
      PORT
      LAYER M1 ;
      RECT  0.000 2.240 2.000 2.600 ;
      RECT  1.570 1.415 1.660 2.600 ;
      RECT  0.370 1.400 0.555 1.490 ;
      RECT  0.370 1.400 0.460 2.600 ;
      END
```

```
    END VDD
    OBS
        LAYER M1 ;
        RECT  0.165 1.220 0.775 1.310 ;
        RECT  0.165 1.220 0.255 1.530 ;
        RECT  0.685 1.220 0.775 1.530 ;
        RECT  0.165 0.480 0.255 0.790 ;
        RECT  0.685 0.480 0.775 0.790 ;
        RECT  0.165 0.700 0.775 0.790 ;
        RECT  0.865 0.520 1.075 0.610 ;
        RECT  0.865 0.520 0.955 1.305 ;
        RECT  1.650 1.135 1.740 1.305 ;
        RECT  0.865 1.215 1.740 1.305 ;
        RECT  0.965 1.215 1.055 1.490 ;
        RECT  0.965 1.400 1.135 1.490 ;
    END
END EXAMPLE_CELLX4
```

A.4 Cell Characterization

After layout generation, the cell must be characterized. To do so, the first step is to extract the parasitics from the layout. The *runPEX* script relies on Mentor Calibre for performing this task. Figure A.5 shows the generated abstract view for our case study cell.

No.	Lay...	Source Net	R Count	L Count	C Total (F)	CC Total (F)	C+CC Total (F)
1	A	A	18	7	1.64878E-19	7.97789E-16	7.97954E-16
2	Q	Q	17	4	5.75349E-19	7.48000E-16	7.48575E-16
3	B	B	8	2	4.99371E-19	6.47804E-16	6.48304E-16
4	4	PREQ	18	8	8.66225E-19	7.37205E-16	7.38071E-16
5	5	FBN	7	3	0.00000	2.76896E-16	2.76896E-16
6	6	FBP	7	3	0.00000	2.62950E-16	2.62950E-16
7	GND	GND	14	0	4.66510E-19	5.19087E-16	5.19553E-16
8	VDD	VDD	12	0	2.50380E-19	6.27444E-16	6.27694E-16

Figure A.5 – Extraction report for the case study 2-input C-element.

As explored in Section 4.2.4, this extracted cell is the input of LiChEn, our characterization tool. An example of script for characterizing our case study cell with LiChEn is:

```
config_devices -vdd vdd -gnd gnd
set_library_name TEST
add_schematics EXAMPLE_CELLX4.spi
```

```
add_cells -n EXAMPLE_CELLX4 -i A B -o Q -f Q=(A*B)+(A*Q)+(B*Q)
add_slope slope0 0.005 0.010 0.020 0.050 0.080
add_load load0 0.0005 0.001 0.030 0.080
add_model_table EXAMPLE_CELLX4 slope0 load0
set_voltage 1
set_temp 25
set_process typ
set_vdd vdd vdds
set_gnd gnd gnds
set_models ../models/st65.scs
set_vh 0.9
set_vl 0.1
set_vhtl 0.6
set_vlth 0.4
set_start_time 0.5
set_sim_time 2000
set_sim_step 0.01
set_load_unit pF
set_time_unit ns
set_resistance_unit Mo
set_max_tran 0.08
characterize_library
export test.lib
exit
```

After executing the tool, the generated Liberty model is the one shown below. This concludes the prrocess of library cell generation.

```
cell(EXAMPLE_CELLX4){
   area : 7.4;
   cell_leakage_power : 36483.3;
   cell_footprint : EXAMPLE_CELLX4;
   rail_connection(vdd, vdd);
   rail_connection(gnd, gnd);
   leakage_power(){
    power_level : "vdd";
    value : 24958;
    when : "!A*!B*!Q";
   }
   leakage_power(){
    power_level : "vdd";
    value : 36371.4;
    when : "A*!B*!Q";
```

```
}
leakage_power(){
 power_level : "vdd";
 value : 17575.1;
 when : "A*B*Q";
}
leakage_power(){
 power_level : "vdd";
 value : 41793.6;
 when : "!A*B*Q";
}
leakage_power(){
 power_level : "vdd";
 value : 41793.6;
 when : "A*!B*Q";
}
leakage_power(){
 power_level : "vdd";
 value : 36371.4;
 when : "!A*B*!Q";
}
pin(A){
 capacitance : 0.00126829;
 direction : input;
 fall_capacitance : 0.00127583;
 fall_capacitance_range(0.000866389 ,0.00127643);
 input_signal_level : vdd;
 max_transition : 0.3;
 rise_capacitance : 0.00126074;
 rise_capacitance_range(0.000910915, 0.00126549);
 internal_power(){
  power_level : "vdd";
  when : "!B*!Q";
  fall_power(pwr_table0){
   values("0.000730593, 0.000731427, 0.000730967, 0.00073053,
      0.00073017, 0.000730129, 0.000728993",\
          "0.000721588, 0.000722547, 0.000722425, 0.000722328,
             0.000721834, 0.000721092, 0.000720667",\
          "0.000710472, 0.000710998, 0.000710942, 0.000710985,
             0.000710914, 0.000710882, 0.000710899",\
          "0.000703229, 0.000703438, 0.000703733, 0.000703574,
             0.000703864, 0.000703844, 0.000704124",\
```

```
          "0.000698946, 0.000699226, 0.000700208, 0.000699285,
             0.000699291, 0.000699285, 0.000699614",\
          "0.000697912, 0.000698497, 0.000698711, 0.000698473,
             0.000698536, 0.000698525, 0.000698602",\
          "0.00069656, 0.00069671, 0.000696845, 0.000696677,
             0.000696768, 0.000696708, 0.000696847");
 }
 rise_power(pwr_table0){
  values("0.000433972, 0.000433773, 0.000433758, 0.000433793,
     0.000434323, 0.000434059, 0.000433953",\
          "0.000433223, 0.000433288, 0.000433308, 0.00043331,
             0.000433552, 0.000433277, 0.000433201",\
          "0.000433137, 0.000433472, 0.000433584, 0.000433567,
             0.00043356, 0.000433518, 0.00043353",\
          "0.00043376, 0.000433937, 0.000434023, 0.000434435,
             0.000434009, 0.000433998, 0.000433981",\
          "0.000434265, 0.000434619, 0.000435043, 0.000434978,
             0.000434644, 0.000434821, 0.000433868",\
          "0.000433899, 0.000434115, 0.00043441, 0.000435128,
             0.000434055, 0.000434924, 0.00043502",\
          "0.000434989, 0.000435001, 0.000435136, 0.000434941,
             0.00043522, 0.0004349, 0.000435093");
 }
}
internal_power(){
 power_level : "vdd";
 when : "B*Q";
 rise_power(pwr_table0){
  values("0.000278555, 0.000278511, 0.000278369, 0.000278256,
     0.000278009, 0.000278039, 0.000278006",\
          "0.000287117, 0.000286912, 0.000286839, 0.000286666,
             0.000286511, 0.000286348, 0.000286293",\
          "0.00030623, 0.00030625, 0.000306007, 0.000305758,
             0.000305502, 0.000305236, 0.000305156",\
          "0.000326052, 0.000326096, 0.000326197, 0.000326021,
             0.000325856, 0.00032515, 0.000325235",\
          "0.000337947, 0.000337989, 0.000338001, 0.000338113,
             0.000337822, 0.000337272, 0.000337178",\
          "0.000342536, 0.000342658, 0.000342513, 0.000342539,
             0.000342541, 0.000342413, 0.00034189",\
          "0.000339806, 0.000339503, 0.000339736, 0.000339408,
             0.00033976, 0.000339943, 0.000339966");
```

```
  }
  fall_power(pwr_table0){
   values("0.000613539, 0.000613469, 0.000613357, 0.000613284,
      0.000613194, 0.000612935, 0.000612777",\
          "0.00061297, 0.000612966, 0.000612916, 0.000612847,
             0.000612758, 0.000612607, 0.000612438",\
          "0.000612192, 0.000612169, 0.000612102, 0.000612464,
             0.000611932, 0.000611729, 0.000611523",\
          "0.000612025, 0.000612047, 0.000611955, 0.000612603,
             0.000611784, 0.000611772, 0.00061151",\
          "0.000611181, 0.000611143, 0.000611727, 0.000612117,
             0.000611165, 0.000610958, 0.000610721",\
          "0.000611258, 0.000611265, 0.000611217, 0.000611333,
             0.000611, 0.000611238, 0.000610794",\
          "0.000608839, 0.000608897, 0.000609237, 0.000609147,
             0.000609212, 0.000609003, 0.000608925");
  }
 }
}
pin(B){
 capacitance : 0.00126895;
 direction : input;
 fall_capacitance : 0.00127589;
 fall_capacitance_range(0.000866384 ,0.00127651);
 input_signal_level : vdd;
 max_transition : 0.3;
 rise_capacitance : 0.00126201;
 rise_capacitance_range(0.000911072, 0.00126473);
 internal_power(){
  power_level : "vdd";
  when : "A*Q";
  rise_power(pwr_table0){
   values("0.000278339, 0.000278364, 0.000278239, 0.000278116,
      0.000277999, 0.00027784, 0.000277811",\
          "0.000286944, 0.000286764, 0.000286687, 0.000286514,
             0.000286361, 0.000286199, 0.000286145",\
          "0.000306079, 0.000306097, 0.000305854, 0.000305606,
             0.00030535, 0.000305085, 0.000305002",\
          "0.000325896, 0.000325929, 0.000326041, 0.000325885,
             0.000325574, 0.000325046, 0.00032508",\
          "0.000337786, 0.000337829, 0.000337841, 0.000337937,
             0.000337665, 0.000337115, 0.000337021",\
```

```
            "0.000342604, 0.0003426, 0.000342345, 0.00034271,
                0.000342383, 0.000342214, 0.00034197",\
            "0.00033997, 0.000339669, 0.000339908, 0.000339574,
                0.00034017, 0.000340106, 0.000340194");
    }
    fall_power(pwr_table0){
     values("0.000613716, 0.000613653, 0.00061354, 0.000613467,
        0.000613416, 0.000613115, 0.000612957",\
            "0.000613146, 0.000613181, 0.000613097, 0.000613029,
                0.000612941, 0.000612781, 0.000612614",\
            "0.000612369, 0.000612341, 0.00061228, 0.000612654,
                0.000612107, 0.000611905, 0.000611691",\
            "0.000612091, 0.000612217, 0.000612129, 0.000612916,
                0.000611962, 0.00061195, 0.000611684",\
            "0.000611362, 0.000611381, 0.000611903, 0.000612283,
                0.000611428, 0.000611126, 0.000610715",\
            "0.000611523, 0.000611463, 0.000611515, 0.000611682,
                0.000611172, 0.000611185, 0.000611054",\
            "0.000609112, 0.000609076, 0.000609336, 0.000609115,
                0.000609221, 0.000609104, 0.00060889");
    }
}
internal_power(){
    power_level : "vdd";
    when : "!A*!Q";
    fall_power(pwr_table0){
     values("0.000730611, 0.000731452, 0.000730969, 0.00073053,
        0.000730169, 0.000730128, 0.000728992",\
            "0.000721597, 0.000722385, 0.000722425, 0.000722326,
                0.000721833, 0.000721091, 0.000720666",\
            "0.000710479, 0.000710994, 0.000710937, 0.000710981,
                0.00071091, 0.000710878, 0.000710895",\
            "0.000703223, 0.00070349, 0.000703745, 0.000703839,
                0.000703867, 0.000703846, 0.000704169",\
            "0.00069894, 0.000699219, 0.000700203, 0.000699279,
                0.000699286, 0.000699279, 0.000699629",\
            "0.000698318, 0.000698497, 0.000698695, 0.000697897,
                0.000698537, 0.0006985, 0.000698689",\
            "0.000696558, 0.000696706, 0.000696844, 0.000696676,
                0.000696766, 0.000696707, 0.000696845");
    }
    rise_power(pwr_table0){
```

```
    values("0.000434054, 0.000433797, 0.000433745, 0.000433777,
       0.000434323, 0.00043405, 0.000433941",\
           "0.0004335, 0.000433326, 0.000433291, 0.000433302,
              0.000433557, 0.000433269, 0.000433192",\
           "0.000433217, 0.000433356, 0.000433596, 0.000433555,
              0.000433578, 0.00043352, 0.000433516",\
           "0.000433605, 0.000433706, 0.000433959, 0.000434431,
              0.000433998, 0.000433992, 0.000433986",\
           "0.000434252, 0.000434405, 0.000434964, 0.000434897,
              0.000434679, 0.000434593, 0.000434196",\
           "0.0004339, 0.000434472, 0.000434462, 0.000435076,
              0.000434365, 0.000435001, 0.000434256",\
           "0.000435241, 0.000435448, 0.000435491, 0.000435629,
              0.000435348, 0.000434876, 0.000435493");
   }
  }
}
pin(Q){
 capacitance : 0;
 direction : output;
 fall_capacitance : 0;
 fall_capacitance_range(0, 0);
 function : "(A*B)+(A*Q)+(B*Q)";
 max_capacitance : 0.07;
 output_signal_level : vdd;
 rise_capacitance : 0;
 rise_capacitance_range(0, 0);
 internal_power(){
  power_level : "vdd";
  related_pin : "B";
  fall_power(pwr_table0){
   values("0.00276258, 0.0017496, 0.000748019, 0.00274613, 0.0057455,
      0.0142444, 0.0317434",\
           "0.00275186, 0.00173897, 0.000758343, 0.00275619,
              0.00575437, 0.0142526, 0.0317513",\
           "0.00272455, 0.00171385, 0.00078077, 0.00277834, 0.00577685,
               0.0142745, 0.0317735",\
           "0.00268351, 0.00166937, 0.000820755, 0.00281504,
              0.00581151, 0.0143062, 0.0318045",\
           "0.00276143, 0.00171072, 0.000798604, 0.00279385,
              0.00578976, 0.0142831, 0.0317791",\
```

```
            "0.00301218, 0.00191045, 0.000637603, 0.00264383,
               0.00564541, 0.0141426, 0.0316408",\
            "0.00367898, 0.00249422, 0.000135418, 0.00216011,
               0.00517385, 0.0136873, 0.0311941");
    }
    rise_power(pwr_table0){
     values("0.00213488, 0.0031589, 0.00567541, 0.00767298, 0.0106645,
        0.0191177, 0.0365224",\
            "0.00212943, 0.00315653, 0.00568165, 0.00768524, 0.0106849,
               0.0191657, 0.0366268",\
            "0.00210374, 0.00313153, 0.00566066, 0.0076672, 0.010668,
               0.0191594, 0.0366287",\
            "0.00206011, 0.00307579, 0.00560574, 0.0076163, 0.0106219,
               0.0191181, 0.0365897",\
            "0.00208703, 0.00307271, 0.00558059, 0.00758891, 0.0105958,
               0.0190965, 0.0365744",\
            "0.00223218, 0.00319034, 0.00566942, 0.00766751, 0.0106668,
               0.0191656, 0.0366441",\
            "0.00266581, 0.00356603, 0.00601046, 0.00799994, 0.010989,
               0.0194683, 0.0369373");
    }
  }
  internal_power(){
    power_level : "vdd";
    related_pin : "A";
    fall_power(pwr_table0){
     values("0.00276259, 0.00174964, 0.000747927, 0.002746, 0.00574535,
        0.0142442, 0.0317432",\
            "0.00275187, 0.00173902, 0.000758287, 0.00275604,
               0.00575446, 0.0142525, 0.0317512",\
            "0.00272456, 0.00171394, 0.000780718, 0.00277821,
               0.00577649, 0.0142735, 0.0317729",\
            "0.00268306, 0.00166939, 0.000820615, 0.00281488,
               0.00581133, 0.014306, 0.0318043",\
            "0.00276156, 0.00171086, 0.000798593, 0.00279367,
               0.00578957, 0.0142825, 0.0317789",\
            "0.00301235, 0.00191013, 0.000637457, 0.00264368,
               0.00564526, 0.0141425, 0.0316406",\
            "0.00367903, 0.00249428, 0.000135559, 0.00216004,
               0.00517382, 0.0136873, 0.0311939");
    }
    rise_power(pwr_table0){
```

```
  values("0.00213467, 0.00315893, 0.00567544, 0.0076726, 0.0106626,
     0.0191174, 0.0365186",\
         "0.00212896, 0.00315656, 0.00568168, 0.00768527, 0.010685,
            0.0191656, 0.0366269",\
         "0.0021038, 0.00313159, 0.00566059, 0.00766728, 0.0106685,
            0.0191589, 0.036633",\
         "0.00206013, 0.00307582, 0.00560576, 0.00761633, 0.0106219,
            0.0191181, 0.0365879",\
         "0.00208704, 0.00307273, 0.00558058, 0.00758893, 0.0105958,
            0.0190966, 0.0365744",\
         "0.00223219, 0.00319035, 0.00566938, 0.00766754, 0.0106667,
            0.0191657, 0.0366441",\
         "0.00266562, 0.00356571, 0.0060104, 0.00799969, 0.0109882,
            0.0194702, 0.0369382");
 }
}
timing(){
 related_pin : "B";
 cell_fall(table0){
  values("0.0228401, 0.0270686, 0.0354682, 0.0414158, 0.0498794,
     0.0730062, 0.120001",\
         "0.0233394, 0.0275586, 0.0359367, 0.0418711, 0.0503225,
            0.0734368, 0.120424",\
         "0.0250759, 0.0292851, 0.0376313, 0.043557, 0.0519885,
            0.0750674, 0.122028",\
         "0.0297951, 0.03399, 0.0423397, 0.0482503, 0.0566694,
            0.0797142, 0.126628",\
         "0.0389718, 0.0433367, 0.0520065, 0.0580187, 0.0664866,
            0.0895177, 0.136366",\
         "0.0476722, 0.0522916, 0.0612978, 0.0675063, 0.0761721,
            0.0994988, 0.14629",\
         "0.0605404, 0.0657656, 0.0757444, 0.0823943, 0.0914512,
            0.115062, 0.162091");
 }
 cell_rise(table0){
  values("0.0189567, 0.023867, 0.0345216, 0.0426633, 0.0547394,
     0.0887482, 0.158508",\
         "0.0194576, 0.0243626, 0.0350068, 0.0431436, 0.0552173,
            0.0892144, 0.158958",\
         "0.0212148, 0.0261008, 0.0367254, 0.0448487, 0.056909,
            0.0908814, 0.160602",\
```

```
            "0.025804, 0.0306264, 0.0412207, 0.0493333, 0.0613783,
               0.0953208, 0.165008",\
            "0.0336332, 0.038604, 0.0493368, 0.0574766, 0.0694948,
               0.103381, 0.173012",\
            "0.0399867, 0.0452182, 0.0561124, 0.0642625, 0.0763146,
               0.110235, 0.179781",\
            "0.0465761, 0.0524188, 0.064021, 0.0723023, 0.0843763,
               0.118257, 0.18786");
    }
    fall_transition(table0){
     values("0.00999774, 0.0146981, 0.0260439, 0.0351268, 0.0489681,
        0.0890689, 0.172587",\
            "0.00994599, 0.0146393, 0.0259848, 0.0350637, 0.0489108,
               0.0890205, 0.172942",\
            "0.0098485, 0.0145366, 0.0258255, 0.0349132, 0.0487812,
               0.0888756, 0.172442",\
            "0.00981317, 0.0144636, 0.0256775, 0.0347441, 0.048579,
               0.0886464, 0.172111",\
            "0.0111352, 0.0155801, 0.0263554, 0.0351406, 0.0486918,
               0.0884619, 0.172162",\
            "0.0134487, 0.0175685, 0.0280155, 0.0366032, 0.0498207,
               0.0889099, 0.171787",\
            "0.0182568, 0.0221335, 0.0319078, 0.0399817, 0.0525222,
               0.0905432, 0.172347");
    }
    rise_transition(table0){
     values("0.0115176, 0.0190882, 0.0385517, 0.0543713, 0.0782755,
        0.146005, 0.285542",\
            "0.011485, 0.019039, 0.0384956, 0.0543139, 0.078184,
               0.14593, 0.285424",\
            "0.0114105, 0.0189468, 0.0383812, 0.0542054, 0.0780696,
               0.14585, 0.28532",\
            "0.0114049, 0.0188668, 0.0382113, 0.0540126, 0.0778793,
               0.145683, 0.285187",\
            "0.0125152, 0.0197014, 0.0384706, 0.0540457, 0.0777193,
               0.14537, 0.284921",\
            "0.0143464, 0.02114, 0.0393022, 0.0546466, 0.0780606,
               0.145426, 0.28473",\
            "0.0180988, 0.0246176, 0.0416583, 0.056344, 0.0792705,
               0.146058, 0.284964");
    }
}
```

```
timing(){
 related_pin : "A";
 cell_fall(table0){
  values("0.0228399, 0.0270683, 0.0354681, 0.0414158, 0.0498793,
     0.0730063, 0.120001",\
         "0.0233392, 0.0275586, 0.0359378, 0.0418713, 0.0503242,
            0.0734369, 0.120424",\
         "0.0250756, 0.0292921, 0.0376324, 0.0435522, 0.05199,
            0.0750662, 0.122029",\
         "0.0297957, 0.0339906, 0.0423398, 0.0482513, 0.0566705,
            0.0797154, 0.126629",\
         "0.0389733, 0.0433385, 0.0520086, 0.0580205, 0.0664884,
            0.0895196, 0.136367",\
         "0.047674, 0.0522933, 0.061301, 0.0675083, 0.076172,
            0.0995005, 0.146292",\
         "0.0605416, 0.0657688, 0.0757441, 0.082396, 0.0914528,
            0.115063, 0.16209");
 }
 cell_rise(table0){
  values("0.0189569, 0.0238668, 0.0345216, 0.0426634, 0.054742,
     0.0887466, 0.158491",\
         "0.0194573, 0.0243625, 0.0350068, 0.0431435, 0.0552173,
            0.0892145, 0.158958",\
         "0.0212149, 0.026101, 0.0367252, 0.0448487, 0.0569091,
            0.0908788, 0.160609",\
         "0.0258039, 0.0306262, 0.0412206, 0.0493332, 0.0613783,
            0.0953208, 0.165015",\
         "0.0336337, 0.0386045, 0.049337, 0.0574767, 0.069495,
            0.103381, 0.173012",\
         "0.0399879, 0.0452193, 0.0561133, 0.0642633, 0.0763149,
            0.110236, 0.179782",\
         "0.0465783, 0.0524215, 0.0640234, 0.0723043, 0.0843765,
            0.118258, 0.187858");
 }
 fall_transition(table0){
  values("0.00999769, 0.014698, 0.0260437, 0.0351271, 0.0489678,
     0.0890689, 0.172587",\
         "0.00994591, 0.0146392, 0.0259845, 0.035064, 0.0489113,
            0.0890202, 0.172939",\
         "0.00984835, 0.0145392, 0.0258261, 0.0349166, 0.048781,
            0.0888755, 0.172441",\
```

```
            "0.00981279, 0.0144647, 0.0256799, 0.0347354, 0.0485789,
               0.0886463, 0.172111",\
            "0.0111348, 0.01558, 0.0263541, 0.0351405, 0.0486928,
               0.0885168, 0.17215",\
            "0.0134515, 0.0175686, 0.0280146, 0.0366031, 0.0498194,
               0.0889092, 0.171795",\
            "0.018257, 0.0221344, 0.0319102, 0.0399818, 0.0525257,
               0.0905412, 0.172353");
    }
    rise_transition(table0){
     values("0.0115177, 0.0190884, 0.0385518, 0.0543758, 0.0782548,
        0.146011, 0.28553",\
            "0.0114851, 0.0190392, 0.0384958, 0.054314, 0.0781844,
               0.14593, 0.285425",\
            "0.0114108, 0.0189469, 0.0383815, 0.054206, 0.0780709,
               0.145834, 0.28537",\
            "0.0114049, 0.0188668, 0.0382115, 0.0540128, 0.0778795,
               0.145683, 0.285202",\
            "0.0125147, 0.0197009, 0.0384704, 0.0540456, 0.0777193,
               0.145368, 0.28492",\
            "0.014346, 0.0211395, 0.0393017, 0.0546461, 0.0780603,
               0.145426, 0.28473",\
            "0.0180982, 0.0246167, 0.0416561, 0.0563435, 0.0792645,
               0.146066, 0.284963");
    }
   }
  }
}
```

APPENDIX B – LIST OF CELLS CURRENTLY AVAILABLE IN THE ASCEND-ST65 V2 LIBRARY

ST_INCL1W11OF2 {X2 X4 X7 X9 X13 X18}: *Inverted NCL gate implementing the 1W11OF2 threshold function in a static transistor topology.*

ST_INCL2W11OF2 {X4 X7 X13 X18 X22 X31}: *Inverted NCL gate implementing the 2W11OF2 threshold function, equivalent to an inverted 2 inputs C-element, in a static transistor topology.*

ST_INCLP1W11OF2 {X2 X4 X7 X9 X13 X18 X27}: *Inverted NCL+ gate implementing the 1W11OF2 threshold function in a static transistor topology.*

ST_NCL1W11OF2 {X2 X4 X7 X9 X13}: *NCL gate implementing the 1W11OF2 threshold function in a static transistor topology.*

ST_NCL2W11OF2 {X2 X4 X7 X9 X13}: *NCL gate implementing the 2W11OF2 threshold function, equivalent to a 2 inputs C-element, in a static transistor topology.*

ST_NCLP1W11OF2 {X2 X4 X7 X9 X13 X18}: *NCL+ gate implementing the 1W11OF2 threshold function in a static transistor topology.*

ST_RINCL2W11OF2 {X2 X4 X7 X9 X13 X18 X22}: *Inverted NCL gate implementing the 2W11OF2 threshold function with a reset pin, equivalent to an inverted resettable 2 inputs C-element, in a static transistor topology.*

ST_RNCL2W11OF2 {X2 X4 X7 X9 X13 X18}: *NCL gate implementing the 2W11OF2 threshold function with a reset pin, equivalent to a resettable 2 inputs C-element, in a static transistor topology.*

ST_SINCL2W11OF2 {X2 X4 X7 X9 X13 X18 X22}: *Inverted NCL gate implementing the 2W11OF2 threshold function with a set pin, equivalent to an inverted settable 2 inputs C-element, in a static transistor topology.*

ST_SNCL2W11OF2 {X2 X4 X7 X9 X13 X18}: *NCL gate implementing the 2W11OF2 threshold function with a set pin, equivalent to a settable 2 inputs C-element, in a static transistor topology.*

SY_INCL2W11OF2 {X2 X4 X7 X9 X13 X18}: *Inverted NCL gate implementing the 2W11OF2 threshold function, equivalent to an inverted 2 inputs C-element, in a symmetric transistor topology.*

SY_NCL2W11OF2 {X2 X4 X9 X13 X18}: *NCL gate implementing the 2W11OF2 threshold function, equivalent to a 2 inputs C-element, in a symmetric transistor topology.*

SY_RINCL2W11OF2 {X2 X4 X7 X9 X13 X18 X22}: *Inverted NCL gate implementing the 2W11OF2 threshold function with a reset pin, equivalent to an inverted resettable 2 inputs C-element, in a symmetric transistor topology.*

SY_RNCL2W11OF2 {X2 X4 X7 X9 X13 X18}: *NCL gate implementing the 2W11OF2 threshold function with a reset pin, equivalent to a resettable 2 inputs C-element, in a symmetric transistor topology.*

ST_INCL1W111OF3 {X2 X4 X7 X9 X13 X18 X31}: *Inverted NCL gate implementing the 1W111OF3 threshold function in a static transistor topology.*

ST_INCL2W111OF3 {X2 X4 X7 X9 X13 X18}: *Inverted NCL gate implementing the 2W111OF3 threshold function in a static transistor topology.*

ST_INCL2W211OF3 {X2 X7 X9 X13 X18 X22 X31}: *Inverted NCL gate implementing the 2W211OF3 threshold function in a static transistor topology.*

ST_INCL3W111OF3 {X2 X4 X7 X9 X13 X18 X31}: *Inverted NCL gate implementing the 3W111OF3 threshold function, equivalent to an inverted 3 inputs C-element, in a static transistor topology.*

ST_INCL3W211OF3 {X2 X4 X7 X9 X13 X18 X31}: *Inverted NCL gate implementing the 3W211OF3 threshold function in a static transistor topology.*

ST_INCLP1W111OF3 {X2 X4 X7 X9 X13 X18}: *Inverted NCL+ gate implementing the 1W111OF3 threshold function in a static transistor topology.*

ST_INCLP3W211OF3 {X2 X4 X7 X9 X13 X18 X27}: *Inverted NCL+ gate implementing the 3W2211OF3 threshold function in a static transistor topology.*

ST_NCL1W111OF3 {X2 X4 X7 X9 X13 X18}: *NCL gate implementing the 1W111OF3 threshold function in a static transistor topology.*

ST_NCL2W111OF3 {X2 X4 X7 X9 X13}: *NCL gate implementing the 2W111OF3 threshold function in a static transistor topology.*

ST_NCL2W211OF3 {X2 X4 X7 X9 X13}: *NCL gate implementing the 2W211OF3 threshold function in a static transistor topology.*

ST_NCL3W111OF3 {X2 X4 X7 X9 X13 X18}: *NCL gate implementing the 3W111OF3 threshold function, equivalent to a 3 inputs C-element, in a static transistor topology.*

ST_NCL3W211OF3 {X2 X4 X7 X9 X13 X18}: *NCL gate implementing the 3W211OF3 threshold function in a static transistor topology.*

ST_NCLP1W111OF3 {X2 X4 X7 X9 X13 X18}: *NCL+ gate implementing the 1W111OF3 threshold function in a static transistor topology.*

ST_NCLP2W111OF3 {X2 X4 X7 X9}: *NCL+ gate implementing the 2W111OF3 threshold function in a static transistor topology.*

ST_NCLP2W211OF3 {X2 X4 X7 X9 X13 X18}: *NCL+ gate implementing the 2W211OF3 threshold function in a static transistor topology.*

ST_NCLP3W211OF3 {X2 X4 X7 X9 X13 X18}: *NCL+ gate implementing the 3W211OF3 threshold function in a static transistor topology.*

ST_RINCL3W111OF3 {X13}: *Inverted NCL gate implementing the 3W111OF3 threshold function with a reset pin, equivalent to an inverted resettable 3 inputs C-element, in a static transistor topology.*

ST_RNCL3W111OF3 {X2 X4 X7 X9 X13}: *NCL gate implementing the 3W111OF3 threshold function with a reset pin, equivalent to a resettable 3 inputs C-element, in a static transistor topology.*

ST_SINCL3W111OF3 {X2 X4 X9 X13}: *Inverted NCL gate implementing the 3W111OF3 threshold function with a set pin, equivalent to an inverted settable 3 inputs C-element, in a static transistor topology.*

ST_SNCL3W111OF3 {X2 X4 X7 X9 X13}: *NCL gate implementing the 3W111OF3 threshold function with a set pin, equivalent to a settable 3 inputs C-element, in a static transistor topology.*

ST_INCL1W1111OF4 {X2 X4 X7 X9 X13 X18 X31}: *Inverted NCL gate implementing the 1W1111OF4 threshold function in a static transistor topology.*

ST_INCL2W1111OF4 {X2 X4 X7 X13}: *Inverted NCL gate implementing the 2W1111OF4 threshold function in a static transistor topology.*

ST_INCL2W2111OF4 {X2 X4 X7 X9 X13}: *Inverted NCL gate implementing the 2W2111OF4 threshold function in a static transistor topology.*

ST_INCL2W2211OF4 {X2 X4 X7 X9 X13 X18 X31}: *Inverted NCL gate implementing the 2W2211OF4 threshold function in a static transistor topology.*

ST_INCL3W1111OF4 {X2 X4 X7 X9}: *Inverted NCL gate implementing the 3W1111OF4 threshold function in a static transistor topology.*

ST_INCL3W2111OF4 {X2 X4 X7 X9 X13}: *Inverted NCL gate implementing the 3W2111OF4 threshold function in a static transistor topology.*

ST_INCL3W2211OF4 {X2 X4 X7 X9}: *Inverted NCL gate implementing the 3W2211OF4 threshold function in a static transistor topology.*

ST_INCL3W3111OF4 {X2 X4 X7 X9 X13 X18 X31}: *Inverted NCL gate implementing the 3W3111OF4 threshold function in a static transistor topology.*

ST_INCL3W3211OF4 {X2 X4 X7 X9 X18 X22 X31}: *Inverted NCL gate implementing the 3W3211OF4 threshold function in a static transistor topology.*

ST_INCL4W1111OF4 {X2 X4 X7 X9}: *Inverted NCL gate implementing the 4W1111OF4 threshold function, equivalent to an inverted 4 inputs C-element, in a static transistor topology.*

ST_INCL4W2111OF4 {X2 X4 X7 X9 X13}: *Inverted NCL gate implementing the 4W2111OF4 threshold function in a static transistor topology.*

ST_INCL4W2211OF4 {X2 X4 X7 X9}: *Inverted NCL gate implementing the 4W2211OF4 threshold function in a static transistor topology.*

ST_INCL4W2321OF4 {X2 X4 X7 X9 X13}: *Inverted NCL gate implementing the 4W2321OF4 threshold function in a static transistor topology.*

ST_INCL4W3111OF4 {X2 X4 X7 X9 X13 X18 X22}: *Inverted NCL gate implementing the 4W3111OF4 threshold function in a static transistor topology.*

ST_INCL4W3221OF4 {X2 X4 X7 X9 X13}: *Inverted NCL gate implementing the 4W3221OF4 threshold function in a static transistor topology.*

ST_INCL5W2211OF4 {X2 X4 X7 X9 X13 X18 X22}: *Inverted NCL gate implementing the 5W2211OF4 threshold function in a static transistor topology.*

ST_INCL5W3211OF4 {X2 X4 X7 X9 X13 X18 X22}: *Inverted NCL gate implementing the 5W3211OF4 threshold function in a static transistor topology.*

ST_INCL5W3221OF4 {X2 X4 X7}: *Inverted NCL gate implementing the 5W3221OF4 threshold function in a static transistor topology.*

ST_INCLAO21O2OF4 {X2 X4 X7 X9 X13}: *Inverted NCL gate implementing the special 0 function in a static transistor topology.*

ST_INCLAO22OF4 {X2 X4 X7 X9 X13 X18}: *Inverted NCL gate implementing the special 1 function in a static transistor topology.*

ST_INCLOA22OF4 {X2 X4 X7 X9 X13 X18}: *Inverted NCL gate implementing the special 2 function in a static transistor topology.*

ST_INCLP1W1111OF4 {X2 X4 X7 X9 X13 X18}: *Inverted NCL+ gate implementing the 1W1111OF4 threshold function in a static transistor topology.*

ST_INCLP2W1111OF4 {X2 X4 X7 X9 X13}: *Inverted NCL+ gate implementing the 2W1111OF4 threshold function in a static transistor topology.*

ST_INCLP2W2111OF4 {X2 X4 X7 X9}: *Inverted NCL+ gate implementing the 2W2111OF4 threshold function in a static transistor topology.*

ST_INCLP2W2211OF4 {X2 X4 X7 X9 X13 X18}: *Inverted NCL+ gate implementing the 2W2211OF4 threshold function in a static transistor topology.*

ST_INCLP3W1111OF4 {X2 X4 X9}: *Inverted NCL+ gate implementing the 3W1111OF4 threshold function in a static transistor topology.*

ST_INCLP3W2111OF4 {X2 X4 X7 X9 X13}: *Inverted NCL+ gate implementing the 3W2111OF4 threshold function in a static transistor topology.*

ST_INCLP3W2211OF4 {X2 X4 X7 X9 X13}: *Inverted NCL+ gate implementing the 3W2211OF4 threshold function in a static transistor topology.*

ST_INCLP3W3111OF4 {X2 X4 X7 X9 X13 X18}: *Inverted NCL+ gate implementing the 3W3111OF4 threshold function in a static transistor topology.*

ST_INCLP3W3211OF4 {X2 X4 X7 X9 X13 X18}: *Inverted NCL+ gate implementing the 3W3211OF4 threshold function in a static transistor topology.*

ST_INCLP4W2111OF4 {X2 X4 X9 X13}: *Inverted NCL+ gate implementing the 4W2111OF4 threshold function in a static transistor topology.*

ST_INCLP4W2211OF4 {X4 X7 X9 X13}: *Inverted NCL+ gate implementing the 4W2211OF4 threshold function in a static transistor topology.*

ST_INCLP4W3111OF4 {X2 X4 X7 X9}: *Inverted NCL+ gate implementing the 4W3111OF4 threshold function in a static transistor topology.*

ST_INCLP4W3221OF4 {X2 X4 X7 X9 X13}: *Inverted NCL+ gate implementing the 4W3221OF4 threshold function in a static transistor topology.*

ST_INCLP5W2211OF4 {X2 X4 X7 X9 X13}: *Inverted NCL+ gate implementing the 5W2211OF4 threshold function in a static transistor topology.*

ST_INCLP5W2321OF4 {X2 X4 X7 X9}: *Inverted NCL+ gate implementing the 5W2321OF4 threshold function in a static transistor topology.*

ST_INCLP5W3211OF4 {X2 X4 X7 X9}: *Inverted NCL+ gate implementing the 5W3211OF4 threshold function in a static transistor topology.*

ST_INCLP5W3221OF4 {X2 X4 X7 X9 X13}: *Inverted NCL+ gate implementing the 5W3221OF4 threshold function in a static transistor topology.*

ST_INCLPAO22OF4 {X2 X4 X7 X9 X13}: *Inverted NCL+ gate implementing the special 1 function in a static transistor topology.*

ST_INCLPAO2O21OF4 {X4 X7 X9}: *Inverted NCL+ gate implementing the special AO2O21OF4 function in a static transistor topology.*

ST_INCLPOA22OF4 {X2 X4 X7 X9 X13}: *Inverted NCL+ gate implementing the special 2 function in a static transistor topology.*

ST_NCL1W1111OF4 {X2 X4 X7 X9 X13 X18}: *NCL gate implementing the 1W1111OF4 threshold function in a static transistor topology.*

ST_NCL2W1111OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 2W1111OF4 threshold function in a static transistor topology.*

ST_NCL2W2111OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 2W2111OF4 threshold function in a static transistor topology.*

ST_NCL2W2211OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 2W2211OF4 threshold function in a static transistor topology.*

ST_NCL3W1111OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 3W1111OF4 threshold function in a static transistor topology.*

ST_NCL3W2111OF4 {X2 X4 X7 X13}: *NCL gate implementing the 3W2111OF4 threshold function in a static transistor topology.*

ST_NCL3W2211OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 3W2211OF4 threshold function in a static transistor topology.*

ST_NCL3W3111OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 3W3111OF4 threshold function in a static transistor topology.*

ST_NCL3W3211OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 3W3211OF4 threshold function in a static transistor topology.*

ST_NCL4W1111OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 4W1111OF4 threshold function, equivalent to a 4 inputs C-element, in a static transistor topology.*

ST_NCL4W2111OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 4W2111OF4 threshold function in a static transistor topology.*

ST_NCL4W2211OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 4W2211OF4 threshold function in a static transistor topology.*

ST_NCL4W2321OF4 {X4 X7 X9 X13}: *NCL gate implementing the 4W2321OF4 threshold function in a static transistor topology.*

ST_NCL4W3111OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 4W3111OF4 threshold function in a static transistor topology.*

ST_NCL4W3221OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 4W3221OF4 threshold function in a static transistor topology.*

ST_NCL5W2211OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 5W2211OF4 threshold function in a static transistor topology.*

ST_NCL5W3211OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 5W3211OF4 threshold function in a static transistor topology.*

ST_NCL5W3221OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the 5W3221OF4 threshold function in a static transistor topology.*

ST_NCLAO21O2OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the special 0 function in a static transistor topology.*

ST_NCLAO22OF4 {X2 X4 X7 X9 X13}: *NCL gate implementing the special 1 function in a static transistor topology.*

ST_NCLOA22OF4 {X2 X4 X7 X9}: *NCL gate implementing the special 2 function in a static transistor topology.*

ST_NCLP1W1111OF4 {X2 X4 X7 X13 X18}: *NCL+ gate implementing the 1W1111OF4 threshold function in a static transistor topology.*

ST_NCLP2W1111OF4 {X2 X4 X7 X9 X13}: *NCL+ gate implementing the 2W1111OF4 threshold function in a static transistor topology.*

ST_NCLP2W2111OF4 {X2 X4 X7 X9 X13}: *NCL+ gate implementing the 2W2111OF4 threshold function in a static transistor topology.*

ST_NCLP2W2211OF4 {X2 X4 X7 X9}: *NCL+ gate implementing the 2W2211OF4 threshold function in a static transistor topology.*

ST_NCLP3W1111OF4 {X2 X4 X7 X9 X13}: *NCL+ gate implementing the 3W1111OF4 threshold function in a static transistor topology.*

ST_NCLP3W2111OF4 {X2 X4 X7 X9 X13}: *NCL+ gate implementing the 3W2111OF4 threshold function in a static transistor topology.*

ST_NCLP3W2211OF4 {X2 X4 X13}: *NCL+ gate implementing the 3W2211OF4 threshold function in a static transistor topology.*

ST_NCLP3W3111OF4 {X2 X4 X7 X9}: *NCL+ gate implementing the 3W3111OF4 threshold function in a static transistor topology.*

ST_NCLP3W3211OF4 {X2 X7 X9}: *NCL+ gate implementing the 3W3211OF4 threshold function in a static transistor topology.*

ST_NCLP4W2111OF4 {X2 X4 X7 X9 X13}: *NCL+ gate implementing the 4W2111OF4 threshold function in a static transistor topology.*

ST_NCLP4W2211OF4 {X2 X4 X7 X9 X13}: *NCL+ gate implementing the 4W2211OF4 threshold function in a static transistor topology.*

ST_NCLP4W3111OF4 {X2 X4 X9 X13}: *NCL+ gate implementing the 4W3111OF4 threshold function in a static transistor topology.*

ST_NCLP4W3221OF4 {X2 X4 X7 X9 X13}: *NCL+ gate implementing the 4W3221OF4 threshold function in a static transistor topology.*

ST_NCLP5W2211OF4 {X2 X4 X7 X9 X13}: *NCL+ gate implementing the 5W2211OF4 threshold function in a static transistor topology.*

ST_NCLP5W2321OF4 {X2 X4 X7 X9 X13}: *NCL+ gate implementing the 5W2321OF4 threshold function in a static transistor topology.*

ST_NCLP5W3211OF4 {X2 X4 X7 X9 X13}: *NCL+ gate implementing the 5W3211OF4 threshold function in a static transistor topology.*

ST_NCLP5W3221OF4 {X2 X4 X7 X9 X13}: *NCL+ gate implementing the 5W3221OF4 threshold function in a static transistor topology.*

ST_NCLPAO22OF4 {X2 X4 X7 X9}: *NCL+ gate implementing the special 1 function in a static transistor topology.*

ST_NCLPAO2O21OF4 {X2 X4 X7 X9 X13}: *NCL+ gate implementing the special AO2O21OF4 function in a static transistor topology.*

ST_NCLPOA22OF4 {X2 X4 X7 X9 X13}: *NCL+ gate implementing the special 2 function in a static transistor topology.*

APPENDIX C – LIST OF AWARDS AND PUBLISHED ARTICLES

During the development of the work presented in this document, 37 scientific articles were authored or co-authored by this Author. From these articles, 3 were published on renowned scientific journals and 34 in international conferences with a high impact on VLSI research. Among these publication vehicles we highlight the three journals: IEEE Transactions on Very Large Scale Integration Systems, IEEE Transactions on Circuits and Systems II and the Journal of Low Power Electronics; as well as the International Conference on Computer Design and the International Symposium on Asynchronous Circuits and Systems. Among these publications the Author collaborated with 31 different co-authors from Brasil, United States, India and China, including 9 Professors, 2 postdoctoral researchers, 10 Ph.D. students, 5 M.Sc. students and 5 undergraduate students.

Another factor that we highlight is that the work conducted in this Thesis received the following 3 awards in the field of VLSI:

1. *ICECS 2012 PhD Competition Award, IEEE Circuits and Systems Society*: The LiChEn tool was presented at the Ph.D. competition held in Seville during the International Conference of Electronics, Circuits and Systems, in 2012, and won the award with an unanimous decision of the jury.

2. *ISVLSI 2013 PhD Forum Best Paper Award, IEEE Computer Society*: The ASCEnD flow was presented in the Ph.D. forum held in the IEEE Computer Society Annual Symposium on VLSI, in 2013, and won the best paper award.

3. *VLSI-SoC 2014 3rd Best PhD Forum Poster Award, IFIP/IEEE*: The SDDS-NCL flow was presented at the Ph.D. forum held in the IFIP International Conference on Very Large Scale Integration, in 2014, and won the third best paper award.

Another achievement that we consider of great importance was the presentation of a case study SDDS-NCL design in the ACM Student Research Competition at the IEEE International Conference on Computer Aided Design (ICCAD), in 2014. Accordingly, the presented work made it to the final top 5 works. The following papers, related to the work presented in this Thesis, have been published in scientific journals:

1. Moreira, M. T.; Arendt, M. E.; Moraes, F. G.; Calazans, N. L. V.
   Static Differential NCL Gates: Towards Low Power.
   In: IEEE Transactions on Circuits and Systems II, Express Briefs, 62(6), pp. 563-567, 2015.

2. Moreira, M. T.; Calazans, N. L. V.; Trojan, G.; Moraes, F. G.
   Spatially Distributed Dual-Spacer Null Convention Logic Design.
   In: Journal of Low Power Electronics (Print), 10(3), pp. 313-320, 2014.

3. Moreira, M. T.; Moraes, F. G.; Calazans, N. L. V.
   Beware the Dynamic C-Element. In: IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 22(7), p. 1644-1647, 2014.

The following papers, related to the work presented in this Thesis, have been published in international conferences:

1. Heck, G.; Heck, L. S.; Singhvi, A.; Moreira, M. T.; Beerel, P. A.; Calazans, N. L. V.
   Analysis and Optimization of Programmable Delay Elements for 2-Phase Bundled-Data Circuits.
   In: 28th International Conference on VLSI Design (VLSID), 2015, Bangalore, India.

2. Moreira, M. T.; Hand, D.; Beerel, P. A.; Calazans, N. L. V.
   TDTB error detecting latches: Timing violation sensitivity analysis and optimization.
   In: 16th International Symposium on Quality Electronic Design (ISQED), 2015, Santa Clara, CA, USA.

3. Gibiluka, M.; Moreira, M. T.; Moraes, F.; Calazans, N. L. V.
   BAT-Hermes: A Transition-Signaling Bundled-Data NoC Router.
   In: Latin American Symposium on Circuits & Systems (LASCAS), 2015, Montevideo, Uruguay.

4. Heck, G.; Heck, L.; Moreira, M. T.; Moraes, F.; Calazans, N. L. V.
   A Digitally Controlled Oscillator for Fine-Grained Local Clock Generators in MP-SoCs.
   In: Latin American Symposium on Circuits & Systems (LASCAS), 2015, Montevideo, Uruguay.

5. Hand, D.; Moreira, M. T.; Huang, H. H.; Chen, D.; Butzke, F.; Li, Z.; Gibiluka, M.; Breuer, M.; Calazans, N. L. V.; Beerel, P. A.
   Blade – A Timing Violation ResiLient Asynchronous Template.
   In: 21st IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2015, Mountain View, CA, USA.

6. Hand, D.; Huang, H. H.; Cheng, B.; Zhang, Y.; Moreira, M. T.; Breuer, M.; Calazans, N. L. V.; Beerel, P. A.
   Performance Optimization and Analysis of Blade Designs under Delay Variability.
   In: 21st IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2015, Mountain View, CA, USA.

7. Zhang, Y.; Heck, L.; Moreira, M. T.; Calazans, N. L. V.; Beerel, P. A.
   Design and Analysis of Testable Mutual Exclusion Elements.

In: International Symposium on Asynchronous Circuits and Systems (ASYNC), 2015, Mountain View, CA, USA.

8. Singhvi, A.; Moreira, M. T.; Ramy Tadros ; Calazans, N. L. V.; Beerel, P. A.
A Fine-Grained, Uniform, Energy-Efficient Delay Element for FD-SOI Technologies.
In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Montpellier, France, 2015.

9. R. Guazzelli ; Moraes, F. G.; Calazans, N. L. V.; Moreira, M. T.
Analysis of Supply Voltage ScaLing on SDDS-NCL Design.
In: Symposium on Integrated Circuits and Systems Design (SBCCI), 2015, Salvador, BA, Brazil.

10. Gibiluka M.; Moreira, M. T.; Calazans, N. L. V.
A Bundled-Data Asynchronous Circuit Synthesis Flow Using a Commercial EDA Framework.
In: Euromicro Conference on Digital System Design (DSD), 2015, Funchal, Portugal.

11. Moreira, M. T.; Pontes, J. J. H.; Calazans, N. L. V.
Tradeoffs between RTO and RTZ in WCHB QDI asynchronous design.
In: 15th International Symposium on Quality Electronic Design (ISQED), 2014, Santa Clara, CA, USA.

12. Ziesemer, A.; Reis, R.; Moreira, M. T.; Arendt, M. E.; Calazans, N. L. V.
Automatic layout synthesis with ASTRAN appLied to asynchronous cells.
In: IEEE 5th Latin American Symposium on Circuits and Systems (LASCAS), 2014, Santiago, Chile.

13. Moreira, M. T.; Guazzelli, R. A.; Heck, G.; Calazans, N. L. V.
Hardening QDI circuits against transient faults using delay-insensitive maxterm synthesis.
In: 24th edition of the Great Lakes Symposium on VLSI (GLSVLSI), 2014, Houston, TX, USA.

14. Ziesemer, A.; Reis, R.; Moreira, M. T.; Arendt, M. E.; Calazans, N. L.V.
A design flow for physical synthesis of digital cells with ASTRAN.
In: 24th edition of the Great Lakes Symposium on VLSI (GLSVLSI), 2014, Houston, TX, USA.

15. Moreira, M. T.; Martins, M.; Neutzler, A.; Reis, A.; Ribas, R.; Calazans, N. L. V.
Semi-custom NCL Design with Commercial EDA Frameworks: Is it Possible?.
In: International Symposium on Asynchronous Circuits and Systems (ASYNC), 2014, Potsdam, Germany.

16. Moreira, M. T.; Arendt, M. E.; Guazzelli, R. A.; Calazans, N. L. V.
A New CMOS Topology for Low-Voltage Null Convention Logic Gates Design.
In: International Symposium on Asynchronous Circuits and Systems (ASYNC), 2014, Potsdam, Germany.

17. Moreira, M. T.; Arendt, M.; Ziesemer, A.; Reis, R.; Calazans, N. L. V.
Automated Synthesis of Cell Libraries for Asynchronous Circuits.
In: 27th Symposium on Integrated Circuits and Systems Design (SBCCI), 2014, Aracaju, SE, Brazil.

18. Moreira, M. T.; Calazans, N. L. V.
Advances on the state of the art in QDI design.
In: 22nd International Conference on Very Large Scale Integration (VLSI-SoC), 2014, Playa del Carmen, Mexico.
3rd Best Paper at Ph.D. Forum

19. Moreira, M. T.; Oliveira, C. H. M.; Porto, R. C.; Calazans, N. L. V.
Design of NCL gates with the ASCEnD flow.
In: IEEE 4th Latin American Symposium on Circuits and Systems (LASCAS), 2013, Cuzco, Peru.

20. Moreira, M. T.; Oliveira, C. H. M.; Calazans, N. L. V.; Ost, L. C.
LiChen: Automated Electrical Characterization of Asynchronous Standard Cell Libraries.
In: Euromicro Conference on Digital System Design (DSD), 2013, Santander, Spain.

21. Moreira, M. T.; Calazans, N. L. V.
Design of standard-cell Libraries for asynchronous circuits with the ASCEnD flow.
In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2013, Natal, RN, Brazil.
Best Paper at Ph.D. Forum

22. Moreira, M. T.; Oliveira, C. H. M.; Porto, R. C.; Calazans, N. L. V.
NCL+: Return-to-one Null Convention Logic.
In: IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS), 2013, Columbus, OH, USA.

23. Moreira, M. T.; Magalhães, F. G.; Gibiluka, M.; Hessel, F. P.; Calazans, N. L. V. **BaBaNoC: An asynchronous network-on-chip described in Balsa.**
In: International Symposium on Rapid System Prototyping (RSP), 2013, Montreal, Canada.

24. Moreira, M. T.; Oliveira, B. S.; Moraes, F. G.; Calazans, N. L. V.
Charge sharing aware NCL gates design.
In: IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), 2013, New York, NY, USA.

25. Moreira, M. T.; L. V. Calazans, N.
Voltage scaling on C-elements: A speed, power and energy efficiency analysis.
In: IEEE 31st International Conference on Computer Design (ICCD), 2013, Asheville, NC, USA.

26. Moreira, M.; Oliveira, B.; Moraes, F.; Calazans, N.
Impact of C-elements in asynchronous circuits.
In: 13th International Symposium on Quality Electronic Design (ISQED), 2012, Santa Clara, CA, USA.

27. Moreira, M. T.; Guazzelli, R. A.; Calazans, N. L. V.
Return-to-one protocol for reducing static power in C-elements of QDI circuits employing m-of-n codes.
In: 25th Symposium on Integrated Circuits and Systems Design (SBCCI), 2012, Brasília, DF, Brazil.

28. Moreira, M. T.; Guazzelli, R. A.; Calazans, N. L. V.
Return-to-One DIMS logic on 4-phase m-of-n asynchronous circuits.
In: 19th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2012, Seville, Spain.

29. Moreira, M. T.; Calazans, N. L. V.
Electrical characterization of a C-Element with LiChen.
In: 19th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2012, Seville, Spain.
Ph.D. Competition Award

30. Moreira, M.; Oliveira, B.; Pontes, J.; Calazans, N.
A 65nm standard cell set and flow dedicated to automated asynchronous circuits design.
In: IEEE 24th International System-on-Chip Conference (SOCC), 2011, Taipei, Taiwan.

31. Moreira, M.; Oliveira, B.; Pontes, J.; Moraes, F.; Calazans, N.
Adapting a C-element design flow for low power.
In: 18th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2011, Beirut, Lebanon.