PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
FACULTY OF INFORMATICS
COMPUTER SCIENCE GRADUATE PROGRAM

QDI Asynchronous Design and Voltage Scaling

Ricardo Aquino Guazzelli

Dissertation submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Prof. Dr. Ney Laert Vilar Calazans

Porto Alegre
2017

Replace this page with the Library Catalog Page

Replace this page with the Committee Forms

"*If I have seen further than others, it is by standing upon the shoulders of giants.*"
(Isaac Newton)

PROJETO ASSÍNCRONO QDI E ESCALAMENTO DA TENSÃO DE ALIMENTAÇÃO

RESUMO

Circuitos *quasi-delay insensitive* (QDI) são uma solução promissora para lidar com variações significativas de processo em tecnologias modernas, já que suas características acomodam variações significativas de atraso de fios e portas lógicas. Além disso, com as novas tendências de dispositivos móveis e a Internet das Coisas (IoT), o projeto QDI apresenta-se como um alternativa de implementação para aplicações operando em tensão baixa ou muito baixa. Como principal desvantagem, o projeto QDI conduz a um aumento significativo no consumo de área e potência, o que pode comprometer seu emprego. Este trabalho investiga a compatibilidade de projeto QDI sob escalamento de tensão, explorando e analisando *templates* QDI disponíveis na literatura. Entre estes *templates*, seleciona-se o *template* SCL como um opção interessante para amenizar consumo de área e potência. Contudo, a proposta original deste *template*, demonstra-se aqui, apresenta problemas temporais. Devido a estes problemas, propõe-se aqui um *template* alternativo. Usando o fluxo de projeto ASCEnD, bibliotecas de *standard cells* para os *templates* em questão foram geradas a fim de avaliar os benefícios e desvantagens destes.

Palavras-Chave: Circuitos assíncronos, projeto assíncrono, quasi-delay insensitive, projeto QDI, escalamento de tensão, tensão de limiar, tensão quase-limiar, tensão sub-limiar.

QDI ASYNCHRONOUS DESIGN AND VOLTAGE SCALING

ABSTRACT

Asynchronous quasi-delay-insensitive (QDI) circuits are a promising solution to cope with aggressive process variations faced by circuit design in modern technologies, as these can gracefully accommodate a wide range of gate and wire delay variations. Moreover, with the trend for mobile devices and the Internet of Things (IoT), QDI design presents itself as an interesting circuit implementation alternative for use in conjunction with aggressive low-power techniques such as deep voltage scaling (VS). The main drawback of QDI design is often cited as the fact that it leads to significant area and power overheads, which can compromise its adoption. This work investigates the compatibility of QDI design with voltage scaling, by exploring and analyzing several available QDI templates. Among these, this work selects the SCL template as an interesting option to reduce area and power overheads. However, as this Dissertation demonstrates, SCL current implementation presents serious timing issues. Because of these, an alternative asynchronous design template is proposed. Using the ASCEnD Design Flow, standard cell libraries for the studied templates are generated, to evaluate the benefits and overheads of the newly proposed template.

Keywords: Asynchronous circuits, asynchronous design, quasi delay insensitive, QDI design, voltage scaling, threshold voltage, near-threshold voltage, sub-threshold voltage.

List of Figures

List of Tables

List of Acronyms

ALU – Arithmetic Logic Unit

ASVHB – Autonomous Signal-Validity Half-Buffer

BD – Bundled-Data

CAD – Computer Aided Design

CMOS – Complementary Metal Oxide Semiconductor

DE – Delay Element

DI – Delay Insensitive

DIMS – Delay Insensitive Minterm Synthesis

DSM – Deep Sub-Micron

E^2DP – Energy-squared-Delay Product

EDA – Electronic Design Automation

EDP – Energy-Delay Product

EMI – Electromagnetic Interference

FDSOI – Fully Depleted Silicon on Insulator

FinFET – Fin Field Effect Transistor

GAPH – Grupo de Apoio ao Projeto de Hardware

I/O – In and Out

IC – Integrated Circuit

IoT – Internet of Things

LDP – Leakage-Delay Product

LVT – Low Threshold Voltage

MG – Marked Graph

MOS – Metal Oxide Semiconductor

MTCMOS – Multi-Threshold CMOS

MTNCL – Multi-Threshold Null Convention Logic

NCL – Null Convention Logic

NMOS – N-channel Metal Oxide Semiconductor

NoC – Network-on-a-Chip

PCHB – Pre-charged Half-Buffer

PDN – Pull-Down Network

PDK – Process Design Kit

PMOS – P-channel Metal Oxide Semiconductor

PUN – Pull-Up Network

PVT – Process, Voltage and Temperature

QDI – Quasi-Delay Insensitive

RTO – Return-to-One

RTZ – Return-to-Zero

SCL – Sleep Convention Logic

SDDS-NCL – Spatially Distributed Dual Spacer Null Convention Logic

SM – State Machines

SVT – Standard Threshold Voltage

TPPN – Timed-Place Petri Nets

TTPN – Timed-Transition Petri Nets

VS – Voltage Scaling

WCHB – Weak-Conditioned Half-Buffer

SAHB – Sense Amplifier Half-Buffer

RL-NCL – Registerless Null Conventional Logic

Contents

1. Introduction and Motivation

For several decades, the semiconductor market has managed to reduce the minimum feature size of transistors and wires, which leads to increasing density and cost-effectiveness to build integrated circuits (ICs), among several other benefits. An excellent example of this increase is the SPARC M7 [LSK+15], developed by Oracle in 2015, a single chip that contains over 10 billion transistors. Apart from the benefit of reducing the minimum feature size are higher performance and lower power that can be obtained, due to the lower resistances and capacitances of smaller transistors and wires [BOF10]. However, these advances bring huge challenges to circuit and computer aided design (CAD) tool designers. As the transistor minimum feature size approaches fundamental atomic limits, electronic devices gradually behave less and less as ideal switches, and wires behave less and less as ideal electrical connections with negligible delay and impedance [ITR11]. In addition, increased manufacturing parameter variations bring uncertainties to the processes of estimating and/or predicting the timing and power characteristics of circuits [ITR11].

Nowadays, the predominant digital circuit design style adopted by the industry is the synchronous one. This style takes as fundamental assumption that all components share a common and discrete notion of time, which is guaranteed by the use of a global clock signal distributed throughout the circuit. The clock signal controls every sequential element in the design, typically flip-flops and/or latches. The value stored in these elements can only change when the clock switches its logic level in a given direction, or when it remains in a given active state. This is what enables the design of sequential blocks to deal with time as a discrete variable, allowing data to flow from one register to the next as the clock signal activates. This characteristic enables designers to ignore wire and logic gates delays, as long as the worst case delay between two registers is never longer than the period of the clock signal controlling them [RCN03]. However, despite the fact that synchronous design has abundant CAD support and is familiar to most designers, it also bring challenges with regard to clock signal distribution, skew and power consumption. Also, the current level of precision required on manufacturing processes operating conditions finally results in substantial variations on the electrical characteristics of fabricated devices, which in turn can lead to significant delay and power consumption variations. To cope with these problems, synchronous designs require margins in the period of the clock signal, which leads to increasing costs in performance, power, area and design time. Margins can indeed become the Achilles heel of synchronous design. For example, almost ten years ago Brej stated that industrial circuits could require up to 130% of overhead in the clock period due to the summation of all needed margins [Bre07], and the situation has only become worse since then.

Asynchronous circuits are an alternative to overcome issues faced by contemporary synchronous designers. Unlike the synchronous paradigm main assumption, the design

of these circuits do not rely on a discrete notion of time. In this way, the global clock signal is exchanged for local handshaking control blocks that are added between adjacent storage elements, which establish the synchronization, communication and sequencing of operations [SF01]. This fundamental assumption of local synchronization avoids clock-related problems and overheads, at the cost of extra hardware for local synchronization.

Among the constraints imposed by contemporary designs, power consumption is also an issue that has attracted growing attention. With the increasing investment on portable devices and, more recently, the Internet of Things (IoT), emerging applications such as distributed sensor networks and wearable devices define low energy consumption as a key factor during circuit design [CWC05] [GBMP13] [ILL+13]. In addition, with decreasing feature size, transistors have become increasingly leaky, augmenting static power dissipation, and challenging designers to meet power constraints [BOF10]. This has motivated the research of new design techniques for minimizing power as much as possible. These efforts usually focus on high performance strong inversion in the super-threshold region of operation of transistors, and are implemented at the architectural level, where designers can reduce the computational workload or improve the architecture to achieve better power optimizations [ILL+13]. At the circuit level, a compelling approach to lower power consumption is reducing the voltage supply, which is referred to here as *voltage scaling* (VS). As the supply voltage is quadratically related to the dynamic power, VS is a very effective low power design technique [CPR10]. Taking this design option to the extreme, some low power systems operate in the sub-threshold region of transistor operation [LM12]. This allows achieving drastic power reductions, although with heavy performance penalties.

This work contributes to the state of the art in low and ultra-low power circuit design, by exploring eight of the available quasi-delay-insensitive (QDI) asynchronous design templates proposed in the literature, including:

1. The Weak-Conditioned Half-Buffer (WCHB) [TBV12];
2. The Pre-Charged Half-Buffer (PCHB) [BOF10];
3. The Null Convention Logic (NCL) [FB96];
4. The Spatially Distributed Dual Spacer Null Convention Logic (SDDS-NCL) [MTMC14];
5. The Autonomous Signal-Validity Half Buffer (ASVHB) [HCGC15];
6. The Sense Amplifier Half-Buffer (SAHB) [CHL+17];
7. The Register-less NCL (RL-NCL) [CYP17];
8. The Sleep Convention Logic (SCL) [PSAA16].

The investigation mostly encompasses the study of near- and sub-threshold operation. Early work of the Author showed that the SCL template is particularly promising, for several reasons: (1) it is based on the Null Convention Logic (NCL) template [FB96], a well-established semi-custom way of implementing asynchronous circuits; (2) SCL employs a fine-grained power-gating mechanism that has potential to achieve significant power savings; (3) NCL has demonstrated potential to support VS in a graceful way [MAMC15] [JSL+10]; (4) SCL brings significant area reduction when compared to NCL. Despite the fact that the combination of NCL with sub-threshold operation brings significant benefits [JSL+10], NCL and other QDI templates still have a significant area overhead when compared to equivalent synchronous circuits. This area overhead compromises the adoption of QDI design in current commercial applications. This is another motivation to explore even further the benefits provided by SCL, enabling it as a suitable approach for ultra-low power applications.

1.1 Objectives

The objectives of this work were divided in two classes: strategic and specific. The strategic objectives were:

1. To dominate the field of low and ultra-low power circuit design of asynchronous circuits;
2. To contribute to the state of the art in circuit design, by making viable to plan asynchronous circuits that correctly operate under near- and/or sub-threshold supply regimes;
3. In general, to contribute to the dominion of digital circuit design techniques for operation under low and very low voltage supplies.

To achieve the listed strategic objectives, this work devised a set of specific objectives to reach:

1. To analyze a selected set of eight available asynchronous circuit design templates, with regard to their capacity to support operation under low and very low power supply (typically below nominal voltage and down to a few hundred mV), leading to low and ultra-low power consumption figures;
2. To select and/or propose an asynchronous QDI template to support ultra-low power design;
3. To compare one or more asynchronous design templates with the proposed QDI template, based on case study circuits;

4. To compare the proposed QDI template with competing QDI templates, under the point of view of figures such as power and energy consumption, area, performance and composite figures such as energy per operation (EPO) and energy squared-delay product (E^2DP);

5. To produce a set of guidelines for the design of low and ultra-low power asynchronous circuits.

1.2 Document Structure

The rest of this document is organized as follows. Chapter 2 introduces the main concepts relevant to this Thesis. Chapter 3 gives an overview of several asynchronous QDI design templates, with some emphasis on the Sleep Convention Logic (SCL) template. It also presents initial results of a theoretical comparison of several QDI templates based on marked graphs, a special form of Petri nets useful to analyze asynchronous circuit design templates. The Chapter ends by providing a comparison of asynchronous templates based on a circuit implementation in each of them. Chapter 4 highlights some of the SCL implementation issues that can be improved and which can otherwise compromise circuit functionality. This results in the proposal of two new QDI templates, called VELO and VELO+, and which constitute the main original contribution of this Dissertation. Details of the new templates include their pipeline structure, combinational and sequential logic implementation details and their completion detection organization. Next Chapter 5 approaches the proposed template and its main competitors at the cell-level and at the circuit level, with a set of extensive experiments, operating under representative supply voltages. Finally, Chapter 6 brings a set of conclusions, ongoing and future work on the subjects of the Dissertation.

2. Basic Concepts

This Chapter presents some basic concepts related to several topics covered in this Dissertation. First, it gives some definitions related to asynchronous circuit design with emphasis on the QDI family of asynchronous design templates, exploring a limitation of such techniques as well. Next, it explores some principles of transistor sub-threshold operation, and provides initial results of simulation analysis of logic gates from different technologies operating under several supply voltages, ranging from super-threshold to sub-threshold regimes. A third topic covered in this Chapter is power gating and its use in sub-threshold operation. The last part of the Chapter deals with the performance analysis of asynchronous pipelines, an important topic to enable the investigation of asynchronous circuit design templates.

Among the different asynchronous design templates available in the literature, most can be classified in one of two families: Bundled-Data (BD) and Quasi-Delay-Insensitive (QDI) design templates [BOF10]. BD templates usually assume data is represented with binary codes, just as in synchronous design, where a single wire carries exactly one bit of information. An advantage of BD design is that it can benefit to some extent from the use of conventional design tools and cell libraries, due to its similarity with synchronous design. The drawback, though, is that BD templates still require additional care in the definition and verification of timing constraints between data and control signals. This work accordingly focuses on QDI circuits only, its advantages and inconveniences, as Section 2.1 explores in more detail.

2.1 Quasi-Delay-Insensitive Design

An alternative to avoid the complex timing issues posed by asynchronous BD templates is to include the control flow information within data *channels*[1], which is the main strategy that characterizes delay insensitive (DI) and quasi-delay insensitive (QDI) designs. DI design is not useful in practice to create large systems [Mar90], but QDI design is, and it keeps most of the advantages of DI design [MN06], by adding a constraint on selected wire forks of the circuit, the so-called *isochronic forks* (see Section 2.1.1 for more details). Since isochronic forks can be limited in scope and designed to exist mostly inside basic design components (e.g. logic gates), QDI becomes a viable family of design techniques. In fact, QDI design is reported by Martin and Nyström and other authors as the most prac-

[1] Simply stated, the concept of channels comprises: (i) a bundle of wires; (ii) a protocol for synchronizing computation and communicating data between circuit blocks; (iii) are uni-directional and typically point-to-point. If there is bi-directional data communication between circuit blocks, it is necessary to employ two channels in opposite directions.

tical asynchronous design template, due to its relaxed timing constraints [MN06] [Mye01]. Its delay insensibility provides higher robustness against process, voltage and temperature (PVT) variations, single event effects (SEE) and permits very low electromagnetic interference (EMI) implementations. On the downside, QDI implementations requires extra hardware, which can lead to significant area and power overheads.

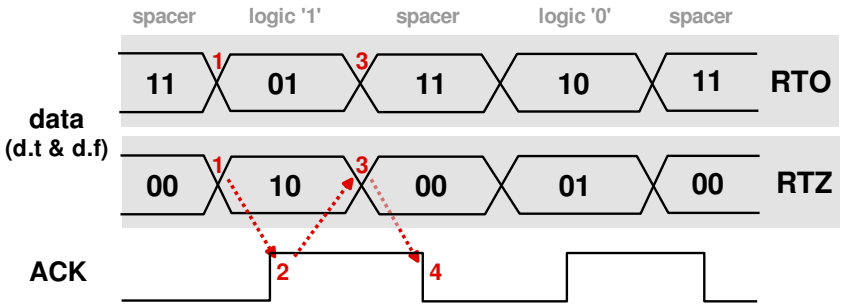
Any QDI template requires the choice of a handshaking protocol and of a delay-insensitive (DI) code to represent data and control flow information. One of the most used DI codes is called dual-rail [MN06]. Refer to Figure 2.1 (a), that presents the basic encoding for a 1-bit dual-rail channel. Each bit of data is coded in two wires called here *d.t* and *d.f*. Usually, the scheme relies on the classic return-to-zero (RTZ), 4-phase handshake protocol [SF01]. A receiver can obtain the equivalent of a request control signal directly from the codewords made available by the sender. In RTZ schemes, data tokens presence is identified by *d.t.* and *d.f.* being at different logic levels. To represent a high logic level, it is necessary to set *d.t* high (1) and *d.f* low (0). The representation of a low logic level is opposite: *d.t* is set low (0) and *d.f* high (1). When communicating any two consecutive valid data, a spacer must always be inserted between them. In the case of the RTZ protocol, a spacer is defined as all wires at logic low (0). This work uses the terms *spacer*, *NULL* and *NULL token* as synonyms. Note that the situation where both signals are set to logic high (1) is defined as an invalid and unacceptable value.

Beyond RTZ, the designer can adopt another handshaking protocol called Return-to-One (RTO) [MOPC13]. Similar to RTZ, the RTO protocol also identify data tokens by *d.t.* and *d.f.* being at different logic levels. However, in this case, to represent a high logic level, it is necessary to set *d.t* low (0) and *d.f* high (1). For low logic level representation, *d.t* is set high (1) and *d.f* low (0). Note that the logic level representation is practically the opposite of the RTZ representation – see Figure 2.1 (a) – including the spacer and invalid encoding. In RTO schemes, both signals in logic low is invalid while the spacer is defined as all wires set (1).

Figure 2.1 (b) illustrates the transmission of two data bits in sequence (a '1' bit followed by a '0' bit), using the RTZ and RTO handshake protocols. As an initial state, for RTZ, all data signals are reset in the beginning of the communication cycle, indicating a spacer. Then, the data channel presents a valid data codification – marked as 1 in Figure 2.1 (b). As a consequence, the *ack* signal is asserted, signaling that the data was received (2). Next, the data channel shows a spacer, indicating the absence of valid data (3). At last, the *ack* signal is reset, ending the communication cycle (4). This same behavior applies to the RTO protocol, only using a distinct data encoding.

Signals		Value	
d.t	*d.f*	RTZ	RTO
0	0	spacer	invalid
0	1	0	1
1	0	1	0
1	1	invalid	spacer

(a)



(b)

Figure 2.1: (a) Codification for a 1-bit dual-rail channel using RTZ/RTO protocols and (b) an example of data transmission through a 2 bits dual rail channel based on both protocols.

2.1.1 The QDI Limitation

In a QDI circuit, gates and wires can display arbitrary delays. However, differently from DI circuits, there is a set of designated wire forks that must respect an isochronic timing constraint. Such isochronic forks have the additional constraint that the delay to different ends of a fork must be the same [Mar90]. According to Sparsø and Furber [SF01], the behavior of an isochronic fork can be explained as follows. Figure 2.2 shows a circuit with three logic blocks (*B0*, *B1* and *B2*) that are interconnected by three wire segments, each with a given delay (*d0*, *d1* and *d2*). In this case, there is a fork *F* through which any value produced by the output of the logic block *B0* passes before reaching the respective inputs of blocks *B1* and *B2*. *F* begins after the wire delay *d0* and has two ends, each one with a wire delay: *d1* and *d2*. Following the definition presented by [Mar90], if the wire delays *d1* and *d2* are identical ($d1 = d2$), the circuit in Figure 2.2 respects the isochronic fork constraint and is thus called an *isochronic fork*. Despite of its elegance, this definition has been later refined to ease the practical implementation of QDI circuits and the verification of the fork isochronicity property.

In 1995-1996, Manohar and Martin presented a new definition of isochronic fork and of the *isochronicity assumption* [MM95]. Considering the same structure presented in Figure 2.2, according to Manohar and Martin [MM95] saying that fork *F* is isochronic means that some transitions on *F* need not to be acknowledged by a transition in both $o1$ and $o2$, the outputs of gates $B1$ and $B2$, respectively. For example, when a transition on the input of $B1$ (after delay $d1$) has been acknowledged by a transition on $o1$, then a transition on the input of $B2$ (after delay $d2$) has also completed, even though $o2$ may not have acknowledged it. This is called the *isochronicity assumption*. As an example, consider that a rising transition occurs in *F* ($F \uparrow$). Thus, this transition will cause $f1 \uparrow$ and $f2 \uparrow$ respectively after delays $d1$ and $d2$. Next, assume that only $B1$ generates a transition in its output ($o1 \uparrow$), while output $o2$ keeps the same logic level. Note that in this case it is not possible to visualize through

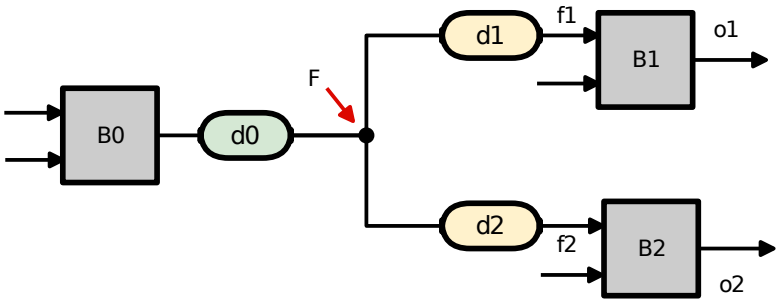


Figure 2.2: Representation of an isochronic fork with logic blocks and delay wires. Adapted from [SF01].

$o2$ whether the transition $f2 \uparrow$ was processed by $B2$. If the fork is isochronic though, after seeing an effect in $o2$ due to $F \uparrow$, it is safe to assume that $B2$ already processed transition $F \uparrow$, too. This clearly means that the delay values $d1$ and $d2$ are identical or that their difference is negligible.

Independently from the proposal of Manohar and Martin, van Berkel and others advanced an extended definition of isochronic forks [vBHP95]. For instance, the fork in Figure 2.2 respects the *extended isochronic fork* definition if the delay difference between $F \rightarrow o1$ and $F \rightarrow o2$ is less than the delays of the gates driven by the output nodes $o1$ and $o2$. That means that all output nodes must be stable when the following gates are triggered. An important aspect to consider is that this definition does not take into account the wire delays of the fork only, but also the gate delays. This is different from the original definition of the isochronic fork, where gate delays are at all disregarded. Moreover, the extended isochronic fork definition also comprises forks that employ more than one logic block in its branches. According to [vBHP95], a fork with only one logic block in each end is a fork of depth 1. For instance, the fork in Figure 2.2 has depth 1.

2.2 Sub-threshold Operation

The sub-threshold effect is present in MOS transistors when the gate to source V_{GS} voltage is equal to or lower than the threshold voltage V_{th}. When $V_{GS} \leq V_{th}$, ideally, the transistor should be off and no current would flow through the transistor drain. In practice, the transistor still leaks a small current, usually denominated *sub-threshold current*. In most digital applications, the sub-threshold current is caused by parasitic leakage currents and is, accordingly, undesirable. This is because it is seen as a deviation from the ideal switch-like behavior of the MOS transistor [LAHG12] [RCN03]. If a circuit is powered by a voltage source where $V_{dd} \leq V_{th}$, then the circuit is on sub-threshold operation – or it is operating in the sub-threshold region.

Sub-threshold operation is a well known technique to reduce both static and dynamic power consumptions [CWC05] [LAHG12] [LM12] [VJD14]. The reduction of the sup-

ply voltage leads to quadratic savings in dynamic power and linear savings in static power. Due to these power savings, sub-threshold operation fits well in digital applications that must rely on energy harvesting or on limited power supplies such as small batteries. Nonetheless, sub-threshold operation brings a significant performance degradation and higher sensibility to process, voltage and temperature (PVT) variations, which narrows the set of applications that can benefit from sub-threshold operation [PN13] [LAHG12].

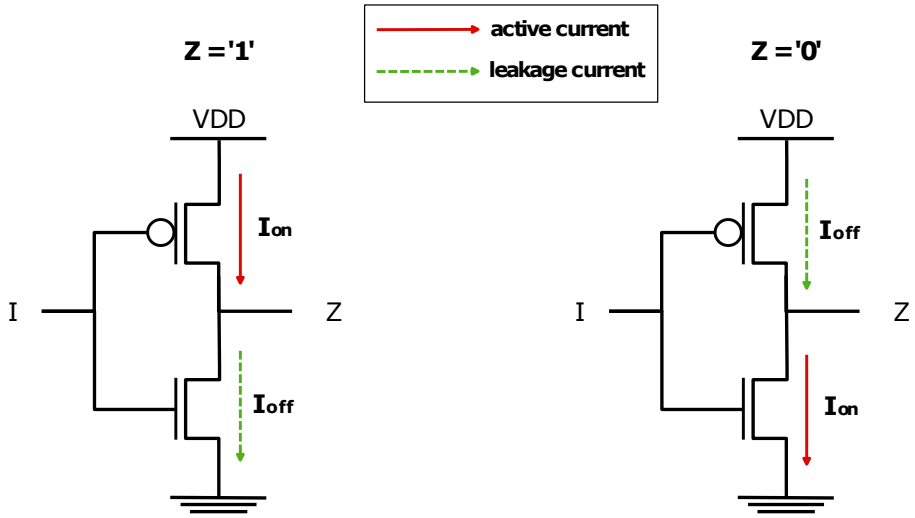


Figure 2.3: Active (I_{on}) and leakage (I_{off}) current representation in a basic CMOS inverter.

In a CMOS standard cell, the output fall and rise transitions rely on the I_{on}/I_{off} ratio, where I_{on} is the active current and I_{off} is the leakage current [PN13]. Figure 2.3 illustrates these currents for an inverter, depending on the output logic level. When $Z = 1$, the pull-up network (PUN) is enabled and is responsible to generate I_{on}, whereas the pull-down network (PDN) is disabled but stills generates I_{off}. When $Z = 0$, the roles change: the PUN is disabled (I_{off}) and the PDN is enabled (I_{on}). If the relation I_{on}/I_{off} is too small, the PDN or PUN may not have enough strength to drive the logic level of the output, making the circuit fail. Due to the use of reduced supply voltages and lower active currents, sub-threshold operation implies in smaller I_{on}/I_{off} ratios. This interferes in the performance of sub-threshold circuits, increasing delays by up to several orders of magnitude. Moreover, PVT variability, transistor sizing and channel doping (among other effects) are responsible for variations of the threshold voltage, which directly affects transistor currents and the I_{on}/I_{off} ratio [PN13].

2.2.1 An Evaluation of Sub-threshold Operation for Bulk and FDSOI CMOS Technologies

For several decades, planar bulk CMOS was the state of the art technology process for IC manufacturing. However, after technology nodes scaled to lower than 45nm-32nm geometries, new chip fabrication processes were needed and proposed. Examples are fully depleted silicon on insulator (FDSOI) CMOS and FinFETs. Such processes bring

new design challenges and opportunities, specially for ultra-low power applications that can benefit from sub-threshold operation. FDSOI CMOS and FinFETs provide better leakage control [VWC+10] [LXW+15] and may achieve better I_{on}/I_{off} ratios than traditional planar bulk CMOS technologies. Moreover, with the current trend to support the IoT domain, new process alternatives [VWC+10] have been developed to optimize the sub-threshold operation of transistors, supporting its use for ultra-low power applications. Currently, however, the adoption of these new fabrication processes implies in a significant higher production cost, which motivates the usage of traditional bulk CMOS technology nodes. Figure 2.4 shows an example of the technology node adoption for IoT applications between the years of 2007 and 2014. Note that in the first two years, older nodes are the most dominant technologies, whereas some early nodes are discretely or not even considered due to commercial unavailability. The next years shows higher adoption of newer nodes – specially for 90nm, 65nm and 40nm nodes – despite the fact that older technologies such as 180nm are still heavily used. From 2010, it is possible to see that the 28nm technology node initiates its contribution to the IoT domain, showing that companies are willing to face higher costs in order to leverage the power optimizations provided by early technologies.

As mentioned before, the power and performance optimizations provided by FDSOI CMOS and FinFET technologies could be attractive to ultra-low power applications, albeit higher production cost could compromise the applications commercially speaking. FDSOI CMOS presents itself as a more suitable alternative than FinFET as the fabrication process is cheaper due to its similarity to traditional planar bulk CMOS process and lower number of required masks. In fact, foundries have pinpointed FDSOI CMOS and FinFET as complementary technologies, where the first covers low-power, smaller designs and the latter focuses on very large, high performance designs.

In order to better understand the advantages of each technology process for sub-threshold operation, this Section presents a first contribution of this work, which includes a VS comparison between two distinct technologies: a 65nm planar bulk CMOS and a 28nm FDSOI CMOS, both from STMicroelectronics.

The evaluation considers 2- and 4-input NAND and NOR gates from the respective standard cell libraries provided by the technology vendor and its results are depicted in Figure 2.5. Note that the selected gates have similar drive strength. The same environment from [GMCM15] was used to simulate all gate arcs in several supply voltages, covering sub-, near- and super-threshold operations. In this case, supply voltages from 150mV to 1V (with 50mV steps) are covered. The results for the 2-input NAND gate can be visualized in Figure 2.5, which indicates the (a) transition delay, (b) transition energy, (c) leakage, (d) energy-delay product (EDP) and (e) leakage-delay product (LDP) for the 65nm planar bulk CMOS (using SVT transistors only) and 28nm FDSOI CMOS technology nodes. Regarding the 28nm technology node, two transistor types were considered: standard (SVT) and low (LVT) threshold voltage. As the other gates present similar results with minor changes, the

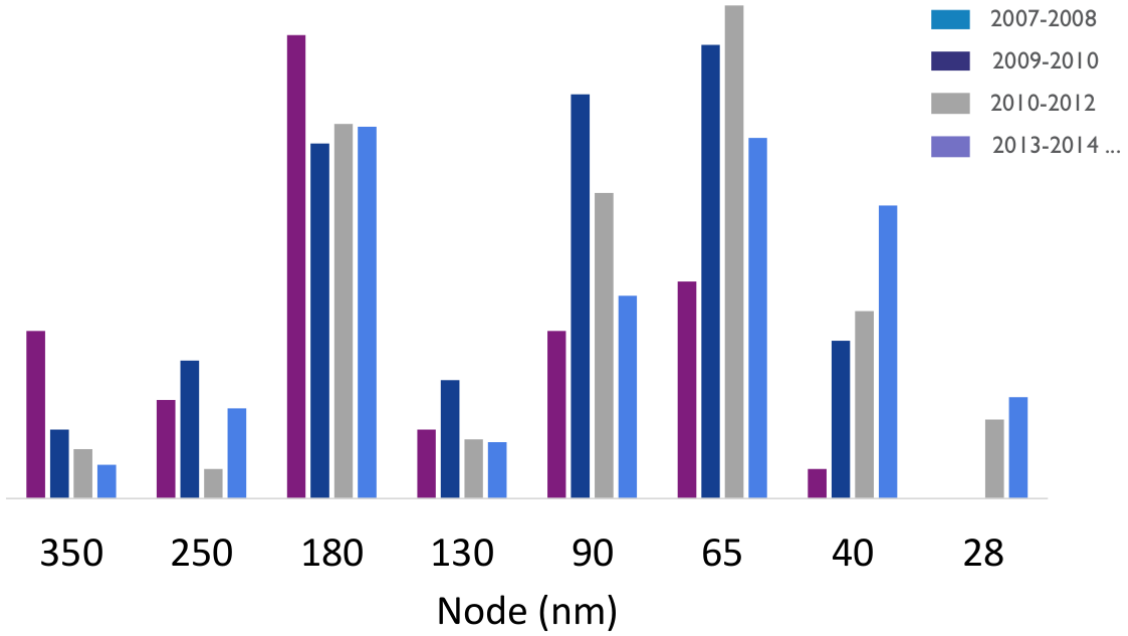


Figure 2.4: CMOS technology adoption for IoT applications. Chart courtesy of IMEC.

following discussion only covers the results obtained for the 2-input NAND – See Appendix A for results regarding the remaining cells employed in the experiment.

Under nominal supply voltage ($V_{dd} = 1\,V$), Figure 2.5(a) indicates that the FDSOI node provides lower transition delay than the bulk technology, which is expected. However, as the supply voltage reaches the sub-threshold region ($V_{dd} < 0.5\,V$), the FDSOI technology presents a larger delay degradation, pointing out that the 65nm technology has a better sub-threshold performance than the SVT and LVT FDSOI technologies. Despite the fact that this could be an interesting information to define which technology node is more suitable for sub-threshold operation, it is important highlight that ultra-low power applications usually do not consider performance as a primary constraint. Energy efficiency plays a huge role in this kind of application, implying that energy must be considered as well. In that way, Figure 2.5 (b) shows the transition energy for the previous mentioned technologies. Different from the delay analysis, the bulk technology produces higher transition energy under all supply voltages. Due to its lower threshold voltage, the LVT FDSOI presents higher short circuit currents during transition than the SVT FDSOI. This increases the transition energy but not significantly enough to reach levels of the bulk technology. In order to combine the results from (a) and (b) and have a better perspective regarding energy efficiency, Figure 2.5(d) shows the EDP for the 2-input NAND. In this case, the EDP curves clearly shows the higher energy efficiency of the FDSOI technology when compared to the bulk technology, specially under the nominal supply voltage and in the near-threshold region. In sub-threshold operation, it is possible to see that the delay degradation of the SVT FDSOI technology inflicts a huge impact on its EDP curve, making its energy consumption to achieve values near the bulk EDP results. This shows that the LVT FDSOI technology provides better energy-

efficient results in sub-threshold operation, while the SVT FDSOI technology has better results in near-/super-threshold operation. Regarding minimum EDP points, the 65nm bulk technology indicates the most energy-efficiency results at $V_{dd} \approx 0.6V$ and the 28nm FDSOI technology at $V_{dd} \approx 0.8V$.

Another important aspect that should be evaluated for an ultra-low power application is the leakage power consumption. This kind of application often implies in systems using a battery as main supply and operating in standby mode for long periods of time. Even if the system is not executing any heavy workload, the leakage current can still cause battery discharge and compromise circuit functionality. Figure 2.5(c) presents the leakage power consumption with the same setup as the previous charts. Similar to what occurs in the energy consumption analysis, the bulk technology presents higher leakage power consumption under all supply voltages. Again, the lower threshold voltage of the LVT FDSOI technology generates leakier transistors, increasing the leakage power consumption when compared to the SVT FDSOI technology. Following an approach similar to that of the EDP analysis, Figure 2.5(e) presents leakage-delay product (LDP) curves of the considered technology nodes, combining the results from (a) and (c). As expected, the bulk technology presents an LDP curve that is above the ones for the FDSOI technology, due to its higher leakage power consumption. Both LVT and SVT FDSOI technologies provide better LDP curves, albeit LDP increases significantly when the supply voltage reaches the sub-threshold region. Similarly to the EDP results, the LDP curves of the 65nm bulk technology presents the most energy-efficient result at $V_{dd} \approx 0.65V$. For FDSOI technologies, LVT and SVT indicate the most energy-efficient result at $V_{dd} \approx 0.7V$ and $V_{dd} \approx 0.8V$, respectively.

2.3 Power Gating

In order to reduce power consumption, several techniques have been presented in the last years, notably driven by the increasing popularity of mobile devices [SAAR11] [LCGC09]. Despite the fact that dynamic power is still dominant in most digital circuits, leakage power is becoming a significant issue as fabrication processes reach into deep sub-micron (DSM) ranges [BDN05] [SAAR11] [MBM05]. This fact led to the proposition of several approaches to reduce leakage power, such as power gating, body biasing, transistor stacking, critical transistor-sizing, etc. Among these options, power gating is widely used to effectively reduce leakage power consumption [LCGC09].

Power gating is a transistor-based technique that consists in the idea of "disconnecting" the circuit from the power supply when it is not operating or is in an idle state. This is implemented by stacking a transistor between the circuit and the power supply. Figure 2.6 shows the three common types of gating transistor configuration [LCGC09]: PMOS Gating, NMOS Gating and Dual Gating. Most commercial designs use PMOS gating, due to their

easier design, especially when the application uses multiple power domains [Hen11]. However, there are two important aspects in using PMOS gating. First, leakage current is not eliminated, because the gating transistor also leaks. Second, The use of a stacked PMOS transistor implies in a voltage drop, which interferes in circuit performance, specially if the circuit is operating in the sub-threshold region. Hence, designers must consider these aspects while sizing the gating transistors, as larger transistors will reduce voltage drop but will increase leakage current. NMOS gating also contributes to the supply voltage drop, but consumes less area, as electrons typically have higher mobility than holes, reducing the size of transistors applied in the gating structure.

2.3.1 Power Gating Granularity

When analyzing power gating in a digital circuit, it is important to consider the granularity at which it is performed. Granularity can be classified in two types [Hen11]: temporal and regional. Temporal granularity considers how long the circuit is in idle periods, while regional granularity indicates how the circuit is divided in power-gating regions. Note that temporal and regional granularity are usually interdependent. For instance, a coarse-grained approach determines that power gating shuts down large logic blocks, such as a processor core or even an entire chip. This, however, forces the power-gating structure to switch large capacitances when shutting down or booting up the logic blocks. Consequently, the active and idle mode transition delays would increase and the power-gating structure demands higher energy consumption during these transitions. Because of that, coarse-grained power-gating is only considered when the logic blocks will stay in idle mode for a long period of time [Hen11]. On the other hand, a fine-grained approach focuses in the idea of shutting down small logic blocks. In a microprocessor, these blocks can correspond to Arithmetic Logic Units (ALUs), I/O blocks or memories. Different from the coarse-grained approach, fine-grained power gating operates with comparatively small capacitances. This implies that the logic blocks can be turned on and off for short periods of time without severely affecting circuit performance. However, this approach brings the need of a more complex power-gating structure, which must efficiently turn on and off each logic block without compromising the energy budget.

2.4 Synchronous and Asynchronous Pipeline Performance

For synchronous circuits, the performance of a pipelined design is characterized by throughput and latency figures, where the first is measured in terms of results per second and the latter is measured as the number of clock cycles to generate a results after the

inputs are available. The throughput is the inverse of the clock frequency for a synchronous system that generates a new result every clock cycle. So the clock period is a synonym of the *cycle time* concept. Obviously, this value can be multiplied if the system generates multiple results per clock cycle, and can be fractioned if multiple clock cycles are required to generate a single result. In pipelined implementations, the latency is a measure of the pipeline depth of the system.

Unlike synchronous designs, the cycle time of an asynchronous system is not defined by the clock frequency but by the time between successive output data tokens it produces. Since this time can vary between tokens, the cycle time is often associated with the average time between output tokens. In addition, asynchronous systems frequently have a warm-up period, during which there is irregular or no generation of output tokens at all. The average cycle time is a long-term average for which the system operates with a regular and constant flow of tokens. The throughput of a asynchronous system is simply the inverse of its cycle time.

The latency of an asynchronous system describes the time required for input tokens being consumed and output tokens being generated. Its measurement relies in the presentation of one set of input tokens in isolation. Isolating each input token avoids the possibility of congestion caused by previous tokens and, consequently, avoids interference during measurements.

The performance of asynchronous system depends on the performance of the communication blocks that compose the system, as well as on the performance of the protocols that dictate how communication proceeds. Hence, some metrics are used to associate communication and protocols with the system performance. In this Chapter, three metrics are covered: *forward latency*, *backward latency* and *local cycle time*. Moreover, this Chapter presents the basics of Petri nets, a modeling aid which is often used to model and evaluate asynchronous templates.

2.4.1 Asynchronous Pipelines Performance Metrics

Forward latency is defined as the time difference between the token arrival in a pipeline stage input channel and its propagation to the stage output channel. The input token must flow through the pipeline in isolation, to avoid that congestion due to previous tokens interfere with this measurement. Because of that, it is assumed that output channels are always ready to consume new tokens, which makes the measurement independent from the output environment. For blocks with multiple input and output channels, the forward latency from different input channels to different output channels may vary. In addition, forward latency can be data dependent or defined by the current state of the pipeline stage.

Backward latency is the time difference between the token arrival in a stage input channel and the moment when the pipeline stage resets and becomes available to receive a new token on that input channel. Usually, it captures the acknowledgement process that a block executes to inform the sender stage that the token has been consumed. As the backward latency is not related to the token propagation to the next pipeline stage, if the input token are sufficiently spread apart in time, the backward latency does not create a performance bottleneck. However, if input tokens are presented in a short amount of time, new input tokens could stall due to the backward latency.

Local cycle time is the shortest length of time for neighboring pipeline stages to complete a handshake process in a specific channel. As it incorporates the entire handshake process, the local cycle time can be associated to the sum of the forward and backward latencies. When back-to-back communication on a channel is required, the local cycle time represents the lower bound for the system cycle time. For instance, the cycle time of a linear pipeline is not smaller than the worst-case local cycle time of all channels in the pipeline. The local cycle time can be dependent on the sender and on the receiver of a single channel or on senders and receivers of neighboring channels. This happens because it is sometimes required for a block to receive a acknowledgement of an output channel before resetting the acknowledgement of a corresponding input channel. Typically, this case is present in many four-phase half-buffer implementations, which makes the local cycle time a function of the behavior of three identical blocks in sequence.

2.4.2 An Asynchronous Pipeline Performance Analysis Tool: Petri Nets

According to Sparsø and Furber [SF01], a Petri net is a graph composed of directed arcs and two types of nodes: transitions and places. Theses places can be marked with tokens and the Petri net model can move tokens by firing transitions. A transition is allowed to fire (or is *enabled*) only if all its input places have at least one token. When the transition fires, it removes a token from each of its input places and adds a token to each of its output places. As Petri nets present themselves as a suitable way to express choice and concurrency, many different concurrent systems can be modeled using different interpretations of these graphs.

Petri nets can be formalized as a four-tuple $N = (P, T, F, m_0)$, where P is a finite set of places p_i and T is a finite set of transitions t_i [BOF10]. $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation and $m_0 \in N^{|P|}$ is the initial marking, where N is the set of natural numbers. Figure 2.7 (a) illustrates a Petri net of a 2-input C-element with the timing diagram of Figure 2.7(b) being an example of correct behavior for this component. Figure 2.7 (c) represents a simpler form of the Petri net in Figure 2.7 (a), where places are omitted. Usually, Petri nets are represented as a bipartite graph where p_i and t_i are nodes. Considering two nodes x and y, if $(x, y) \in F$ then there is a directed arc from x to y. For instance, the Petri net in Figure 2.7

has $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$, $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$, $F = \{(p_1, t_1), (t_1, p_3), (p_2, t_2), (t_2, p_4), (p_3, t_3), (p_4, t_3), (t_3, p_5), (t_3, p_6), (p_5, t_4), (t_4, p_7), (p_6, t_5), (t_5, p_8), (p_7, t_6), (p_8, t_6), (t_6, p_1), (t_6, p_2)\}$ and $m_0 = [11000000]$. An assignment of tokens to places is called a *marking* and it represents the state of the system. For a place or transition $x \in P \cup T$, $\bullet x$ is a preset of x and $x\bullet$ is a postset of x. Formally, a preset can be defined as $\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$ and a postset as $x\bullet = \{y \in P \cup T \mid (x, y) \in F\}$. A transition t is enabled at a marking m if and only if every element of its preset is marked with at least one token. When enabled, the transition removes one token from each place that constitutes its preset, and adds one token to each place that composes its postset. Considering again the example in Figure 2.7, if p_3 and p_4 have tokens, firing t_3 removes one token from p_3 and p_4 and adds one token to p_5 and p_6.

While modeling a system with Petri nets, it is possible to use four basic constructs: fork, join, choice and merge [SF01]. The construct definitions are as follows. If a place p has in its postset more than one element, then p is a choice place. The same applies for merge, albeit in this case it considers the place preset. If p has more than element in its preset, p is a merge place. When a transition t has more than one element in its postset, then t is a fork transition. If t has more than one element in its preset, t is a join transition.

Petri nets can also be divided in multiple types. These types restrict/add specific characteristics from/to the main Petri net definition, to ease its use or make it compatible with the synthesis of certain types of systems. According to [BOF10], some of these system types useful in asynchronous design are State Machines (SMs), Marked Graphs (MGs), Timed-Place Petri nets (TPPNs) and Timed-Transition Petri nets (TTPNs). An SM is a type of Petri net where every transition has at most one element in its preset and at most one element in its postset, or more formally $\mid \bullet t \mid \leq 1 \wedge \mid t\bullet \mid \leq 1, \forall t \in T$. An MG is a type of Petri net where every place has at most one element its preset and at most one element in its postset, or more formally $\mid \bullet p \mid \leq 1 \wedge \mid p\bullet \mid \leq 1, \forall p \in P$. On the one hand, the restriction imposed by the SM definition allows it to model choice, but not concurrency. On the other hand, the MG definition permits the modeling of concurrency, but not of choice. TPPNs and TTPNs are just Petri nets with timing constraints. While TPPNs introduce timing awareness to places, TTPNs introduce these to transitions. The timing awareness introduced by the two last types can also be extended to MGs, for instance. The timing annotation in a *timed* MG can be related to either transitions or places. However, the association of delays with places is less constrained than the association with transitions [BOF10].

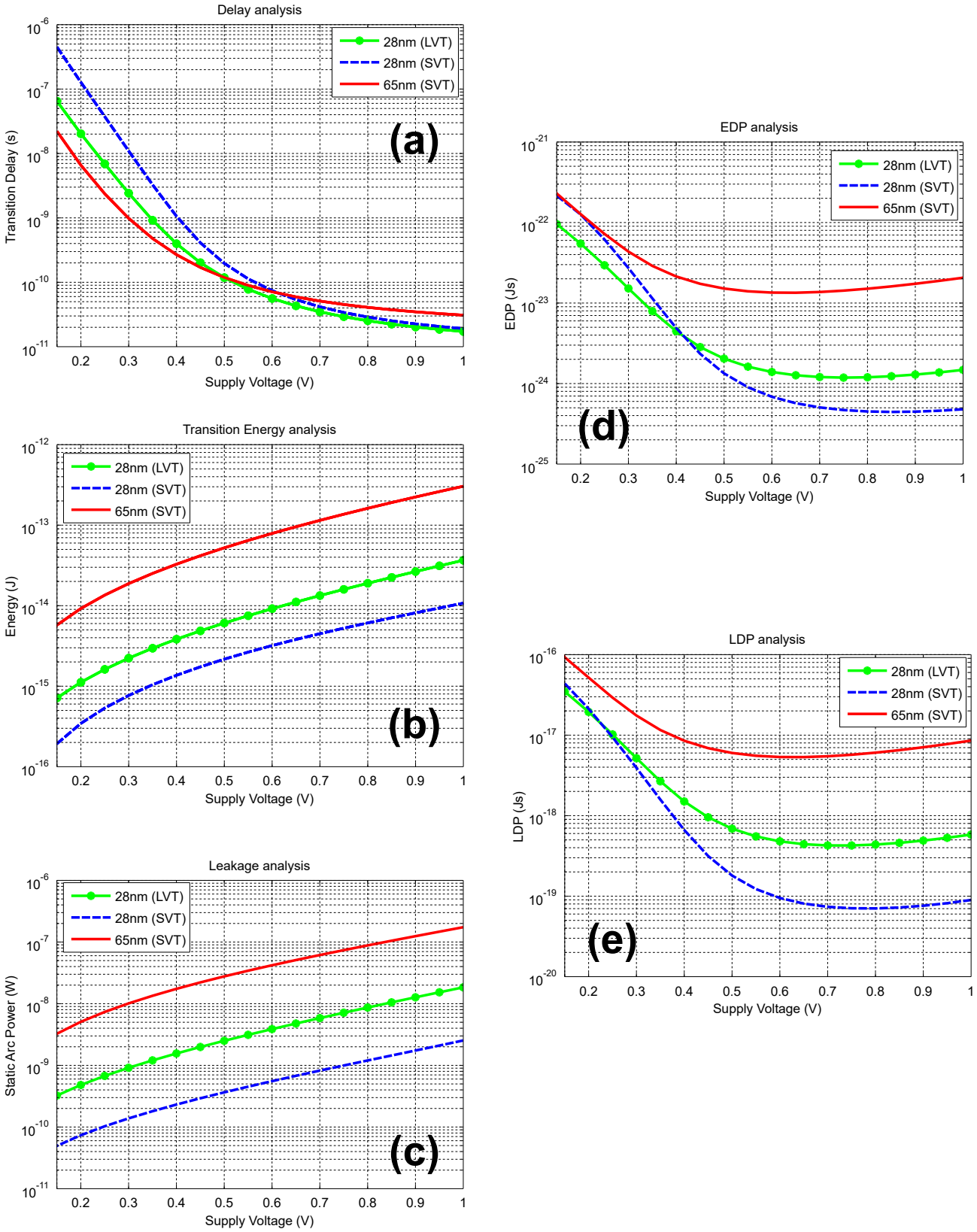


Figure 2.5: Voltage scaling analysis for 2-input NAND gate with similar drive in the 65nm bulk CMOS and 28nm FDSOI technologies. The later technology is analyzed for both SVT and LVT versions of the library. The experiments contemplate analysis of cell: (a) delay; (b) consumed energy; (c) leakage analysis; (d) EDP and (e) LDP.

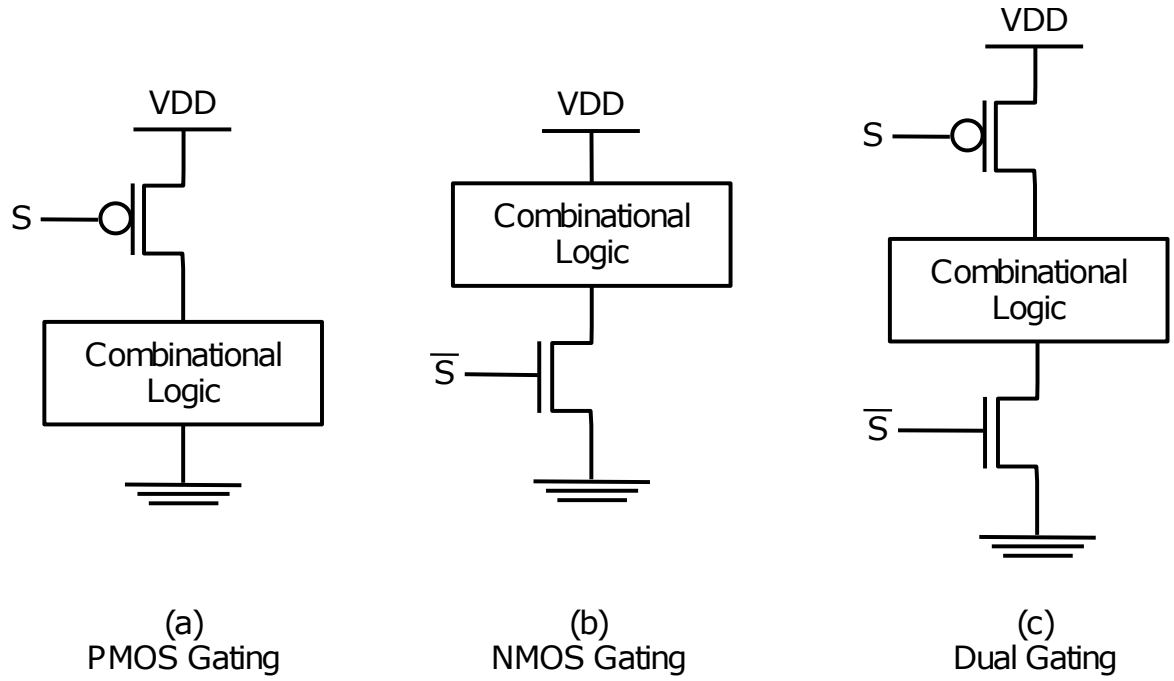


Figure 2.6: Power-gating structures: (a) PMOS Gating, (b) NMOS Gating and (c) Dual Gating. When S = 0, the gating transistor is ON and the combinational logic operates normally (active mode). When S = 1, the gating transistor is OFF and the combinational logic is isolated from the power supply (sleep mode).

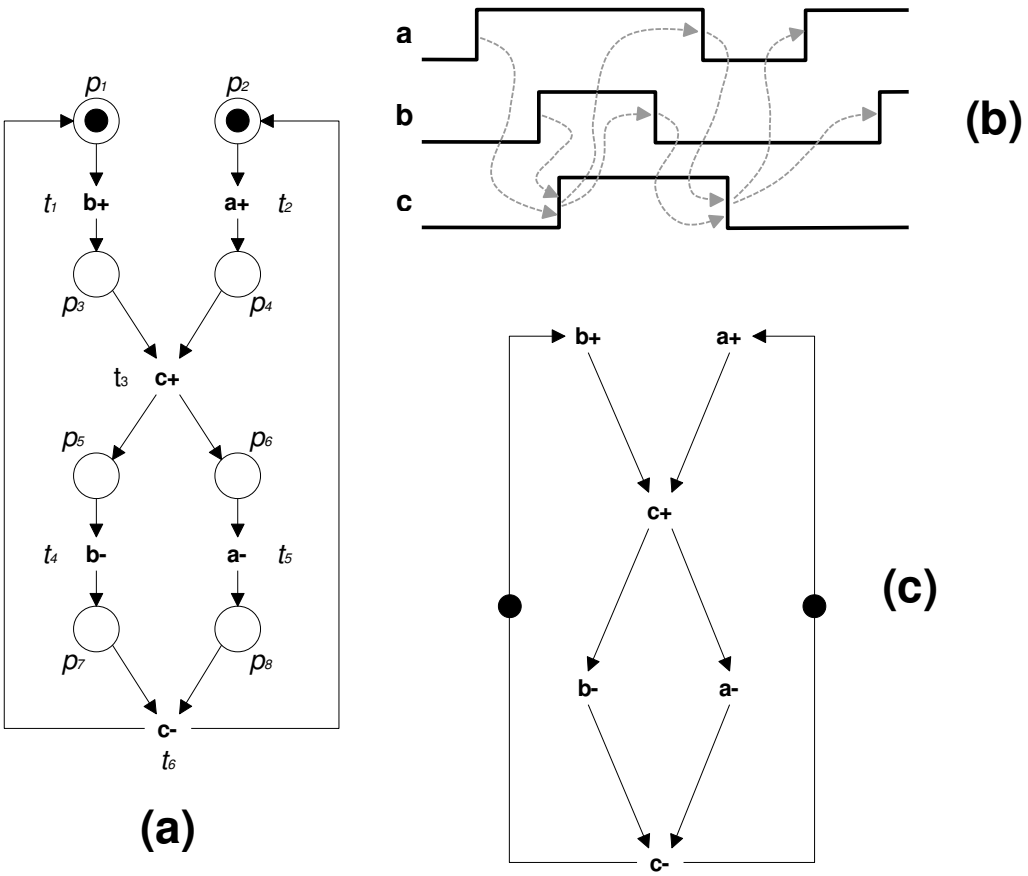


Figure 2.7: Petri net description (a) and timing diagram (b) of a 2-input C-element. (c) presents a simplified Petri net of (a) where places are omitted.

3. Selected QDI Templates: Overview and Analysis

This work assumes that the use of an asynchronous QDI template in the design of ultra-low power circuits is a choice that intrinsically brings several benefits. To support this assumption, this Chapter describes several QDI templates proposed in the literature, introducing their functionality and analyzing their behavior. The Chapter continues with a qualitative discussion of the relative gains and drawbacks of choosing any of these QDI templates as target for circuit design. Some of the initially chosen templates are then discarded from consideration, based on the fact that they are clearly not suitable to achieve the objectives of this work. A circuit implementation then helps in comparing the remaining templates and provides ground for a decision about the target template to address in this work.

Given the suitability of the Null Convention Logic family of gates [FB96] to semi-custom implementations of QDI circuits, several of the templates investigated here employ this gate family. Accordingly, Section 3.3 briefly explores the characteristics of this gate family before presenting the basic asynchronous QDI template based on it.

3.1 The Weak-Conditioned Half-Buffer (WCHB) Template

The Weak-Conditioned Half-Buffer (WCHB) is a four-phase QDI template based on weak-conditioned logic. Figure 3.1 shows the WCHB template in two views, its block diagram and an instance of a gate-level implementation. The WCHB block diagram is composed by a buffer *W*, a completion detector *RCD*, two DI-encoded channels (*Ldata* and *Rdata*) and two acknowledgement signals (*Lack* and *Rack*). As Figure 3.1(b) shows, in a dual-rail implementation for a single bit, *Ldata* employs two wires to represent the information data bit: *L*0 and *L*1. The same of course applies to *Rdata*. The initial state of a WCHB module consists in *Ldata* and *Rdata* containing spacers and *Lack* and *Rack* in logic 1.

The operation of a WCHB stage from the initial (empty or spacer) state follows five main steps, which are described below and illustrated in Figure 3.2.

1. *Ldata* switches from a spacer to a valid data token presented to block *W*; assuming *W* is available (i.e. *Rack* is asserted to logic 1, part of the initial state), *W* propagates *Ldata* to *Rdata*;

2. After *Rdata* has a valid token, *RCD* can compute the validity of *Rdata*, deasserting *Lack* to logic 0; The propagation of *Rdata* to the next stage will also provoke the deassertion of *Rack*;

3. After *Lack* is deasserted, *Ldata* can be reset to a spacer by the environment. However, the buffer maintains its output while *Rack* is not deasserted to logic 0;

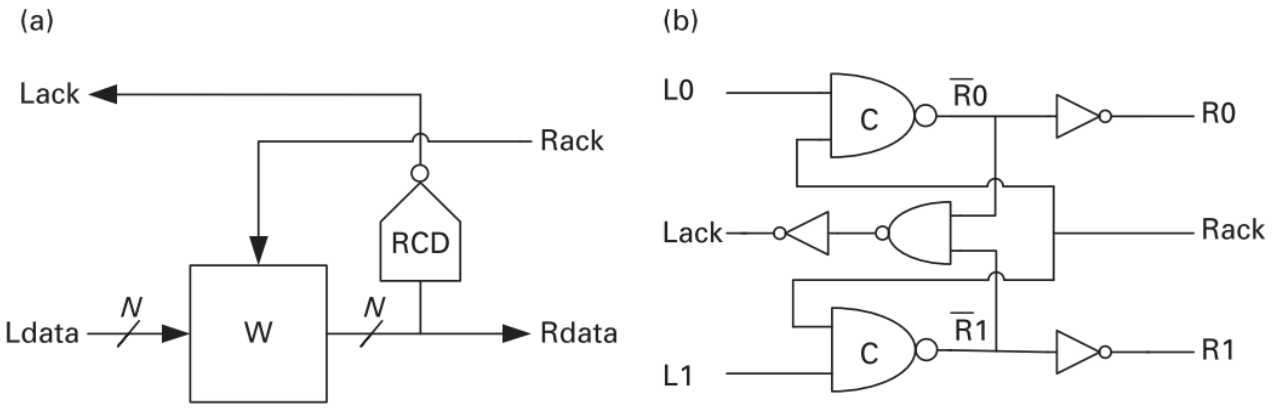


Figure 3.1: The Weak-Conditioned Half Buffer template: (a) block diagram with buffer *W* and completion detector *RCD*; N is the number of data bits of the channel and is represented using a DI code; (b) a dual-rail gate-level implementation for a single bit data word. The gates with the "C" letter inside are C-elements with inverted outputs. Note that this gate-level implementation incorporates both *W* and *RCD* blocks. Based on [BOF10].

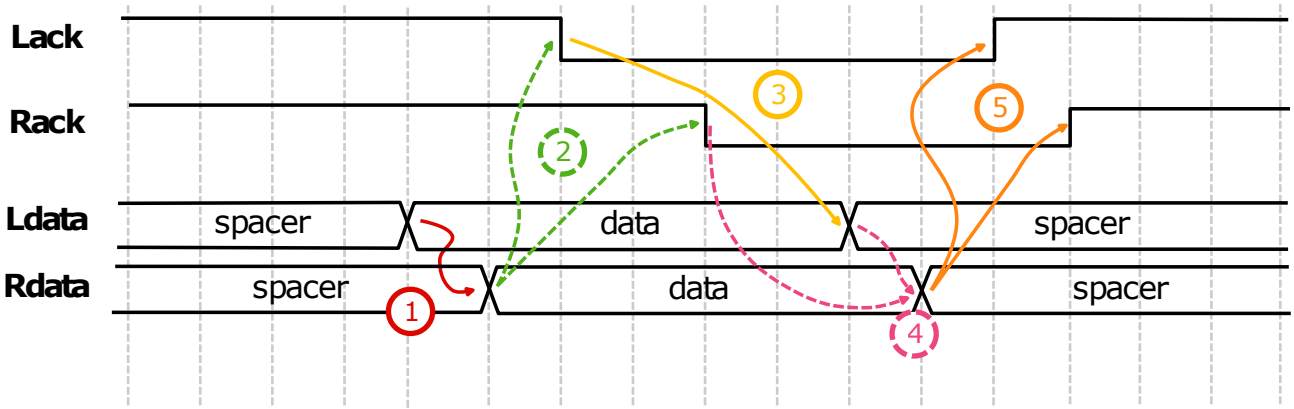


Figure 3.2: Example of the WCHB template operation when in the initial state and processing one data token. Numbers correspond to the bullet items list in the text.

4. When both *Lack* and *Rack* are deasserted, the spacer can propagate from *Ldata* to *Rdata*;

5. The spacer in *Rdata* eventually guarantees that the control signals will be reasserted back to the initial state, *Lack* = 1 and *Rack* = 1. Then, the stage indicates it is available for receiving new data.

In WCHB the validity and neutrality of the output *Rdata* implies the validity and neutrality of the input *Ldata*, respectively. This kind of logic is called *weak-conditioned logic* and represents one of the main features of the WCHB template. The QDI nature of WCHB relies on the isochronic fork assumption for the *Rack* signal, which will always fork to at least two points in practical implementations (See for example Figure 3.1(b)). Another aspect of this template is that channels *Ldata* and *Rdata* cannot simultaneously hold two distinct data tokens. Due to the nature of the protocol, a buffer can only store a new data token if its output currently holds a spacer. This aspect implies that the template is a *half buffer* or has

a *slack of 1/2* [BOF10]. Thus an n-stage WCHB pipeline always contain at most $\lceil n/2 \rceil$ data tokens.

As is evident from the above explanation, the WCHB comprises the buffering and control parts of an asynchronous design template only. In practical implementations, this implies a choice of some type of DI data processing logic style, to form a complete design template. A frequent choice in this case is the Delay-Insensitive Minterm Synthesis (DIMS), which together with WCHB forms the template WCHB/DIMS design template [Mor16]. For shortness any mention here to WCHB implies in fact the WCHB/DIMS template.

For better illustration, Figure 3.3 shows the implementation of a linear 3-stage WCHB pipeline. Each pipeline contains a function logic block F_i and a completion detection block CD_i. The data signals *data_i* are DI-encoded, usually using m-of-n such as dual-rail or multiple 1-of-4 channels. On the other hand, all handshake signals ack_i are represented by a single bit. F_i is responsible to compute its inputs and latch the output when necessary, whereas CD_i detects the presence of valid data tokens in the data channel. Note that F_i is controlled by the handshake signal of the next pipeline stage (ack_{i+1}), which dictates whether the F_i can compute its inputs or should keep the current output.

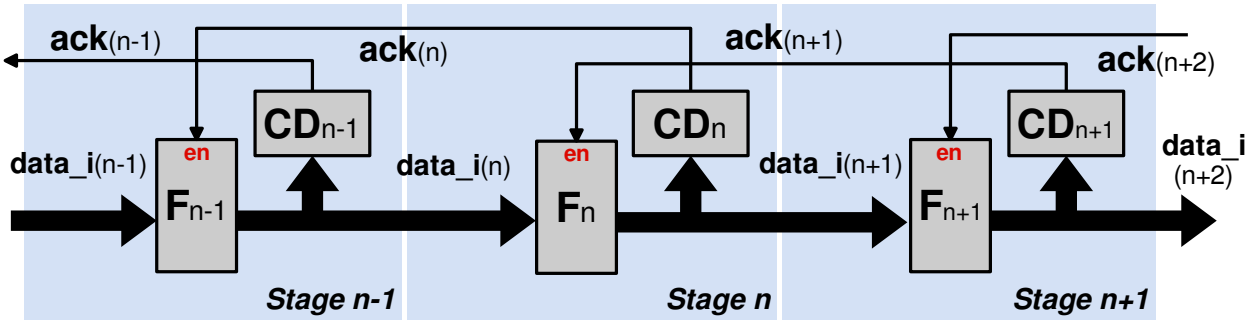


Figure 3.3: Block diagram of a 3-stage WCHB pipeline. Thick lines represent DI-encoded data signals. The remaining lines represent single bit (control) signals.

To abstractly represent the WCHB functionality, Figure 3.4 shows the marked graph for a 3-stage pipeline built with this template. As will be the case for all analyzed templates, the initial marking assumes the pipeline is initially empty (spacers in all stages) and that the environment can accept a newly arrived output data. Also, the $F_i eval$ and $F_i rst$ phases correspond to complete data propagation and complete spacer propagation to the output of stage F_i, respectively.

Being WCHB a half-buffer template, the first pipeline stage can only complete the handshake process if data has propagated to the third stage and the second stage has reset. The cycle time, formed by the thick red lines in the graph, evidences that F_{1eval} will only have tokens in its preset places if the previous token has propagated to the third stage. Equation 3.1 represents the cycle time of the circuit depicted by the MG in Figure 3.4 (To simplify the expressions, it is assumed that all $F_i eval$, CD_i and $F_i rst$ are respectively identical). The

WCHB forward latency employs three *Feval* elements, while the backward latency employs two *CDs* and one *Frst*. Equations 3.2 and 3.3 indicate the elements that compose the forward (FL) and backward (BL) latencies, respectively.

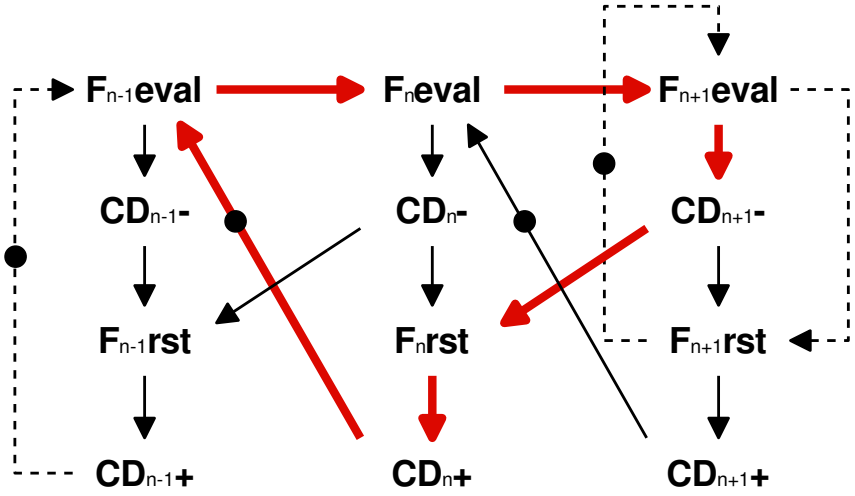


Figure 3.4: Marked Graph of a 3-stage WCHB pipeline. Transitions that compose the cycle time are highlighted with thick red lines. Dashed lines represent external interactions with the circuit environment (e.g. neighboring pipeline stages).

$$CT_{WCHB} = 3 \times t_{eval} + 2 \times t_{CD} + t_{rst} \tag{3.1}$$

$$FL_{WCHB} = 3 \times t_{eval} \tag{3.2}$$

$$BL_{WCHB} = 2 \times t_{CD} + t_{rst} \tag{3.3}$$

3.2 The Precharged Half-Buffer (PCHB) Template

The Precharged Half Buffer (PCHB) is also a four-phase QDI template and it was introduced as an alternative to the WCHB template. PCHB utilizes the same basic concepts from WCHB but incorporates the use of domino logic, which avoids the extensive stacking of PMOS transistor found in WCHB circuits. Figure 3.5 illustrates the PCHB block diagram and a 1-bit, dual rail gate-level implementation of the *F* block.

Unlike what occurs in WCHB, in this template the validity and neutrality of inputs and outputs are separately computed. Completion detector *LCD* is responsible for the verification of the input validity, whereas completion detector *RCD* ensures the output data validity. Both validity signals are synchronized by the inverted C-element, which controls *Lack* and the load of data in block *F*.

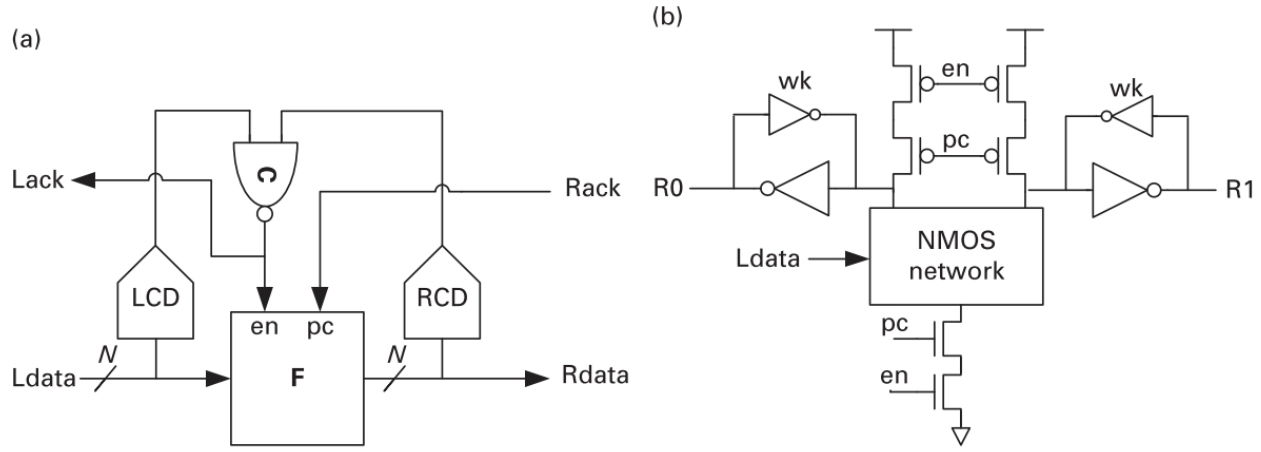


Figure 3.5: Precharged Half-Buffer template structure: (a) block diagram with combinational block *F* and input and output completion detectors, *LCD* and *RCD*; (b) An N=1 bit, dual-rail gate-level implementation of *F*. Here, *Ldata* comprises two wires, *L*0 and *L*1. Based on [BOF10].

Regarding Figure 3.5(b), note that block F_i implements a one-level domino logic with two control inputs: *en* and *pc*. The precharge *pc* signal is controlled by *Rack* and enable *en* is driven by the inverted C-element. A distinctive feature of PCHB compared to WCHB is that a typical stage in the former comprises both buffering and processing logic, besides the control signals.

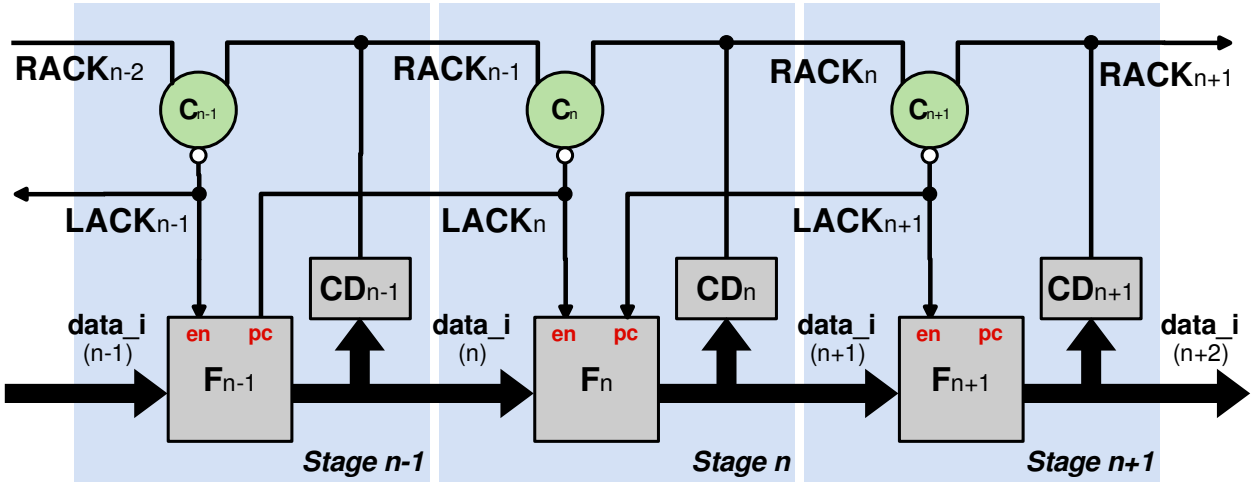


Figure 3.6: Block diagram of a 3-stage PCHB pipeline. Note that the pipeline merges modules LCD with the previous stage's RCD generating a single CD module, to optimize area. Thick lines indicate DI data signals. Usually, these data signals are encoded with an *m-of-n* encoding such as *1-of-2* (dual-rail) or *1-of-4*. The remaining lines represent single bit signals.

The implementation shown in Figure 3.5(b) indicates that PCHB brings about a large area overhead, due to the use of two completion detectors (*LCD* and *RCD*). Fortunately, this overhead can be reduced by merging completion detectors. The *LCD* of one

pipeline stage merges with the *RCD* of a neighbor pipeline stage and adds a request signal to the channel. Figure 3.6 shows a 3-stage PCHB pipeline with the LCD/RCD merging optimization, indicated as *CD*. Without the optimization, on the one hand, the output of the RCD_i block would only connect to the input of C_i (see Figure 3.5(a)). On the other hand, as Figure 3.6 points out, CD_i sends its output to C_i and to the next inverted C-element C_{i+1}, where the latter comprises the new request signal. The addition of this request line does not significantly impact performance, keeps the template with the QDI property and the communication between stages as DI. However, the completion detection sharing implies that the fork formed by the input of a CD_i and the input of the next stage's logic block F_{n+1} must be isochronic [BOF10].

Figure 3.7 shows the marked graph of the 3-stage PCHB pipeline depicted in Figure 3.6. Assumptions here are similar to those for the WCHB discussion, *mutatis mutandis*. Again, the cycle time is highlighted with thick red lines and is computed based in Equation 3.4. The token propagation to the next stages is similar to what occurs in the WCHB template, as the PCHB forward latency (Equation 3.5) has the same transitions as the WCHB template. However, it is important to note that PCHB uses dynamic logic, which is faster than traditional complementary logic. Consequently, even with similar forward latencies, PCHB can provide lower overall latency than WCHB. In addition, Equation 3.6 shows the backward latency of the PCHB template. Comparing this backward latency to the WCHB one, the PCHB template has additional transitions, due to the C-element which controls the enable signal. This brings an additional delay overhead to the PCHB cycle time, which is nonetheless usually not significant.

$$CT_{PCHB} = 3 \times t_{eval} + 2 \times (t_{CD} + t_C) + t_{pre} \tag{3.4}$$

$$FL_{PCHB} = 3 \times t_{eval} \tag{3.5}$$

$$BL_{PCHB} = 2 \times t_{CD} + 2 \times t_C + t_{pre} \tag{3.6}$$

3.3 The Null Convention Logic (NCL) Gate Family and its Basic Template

Theseus Logic, Inc. proposed the NCL logic gate family [FB96] to support the implementation of QDI asynchronous circuits. Since then, NCL has been applied to deal with the design of low power, high speed and fault tolerant circuits, among other circuit classes. Employing NCL gates permits power-, area- and speed-efficient QDI design with standard cell-based approaches, as opposed to other asynchronous templates that require full-custom approaches. NCL gates couple a threshold function with positive integer weights

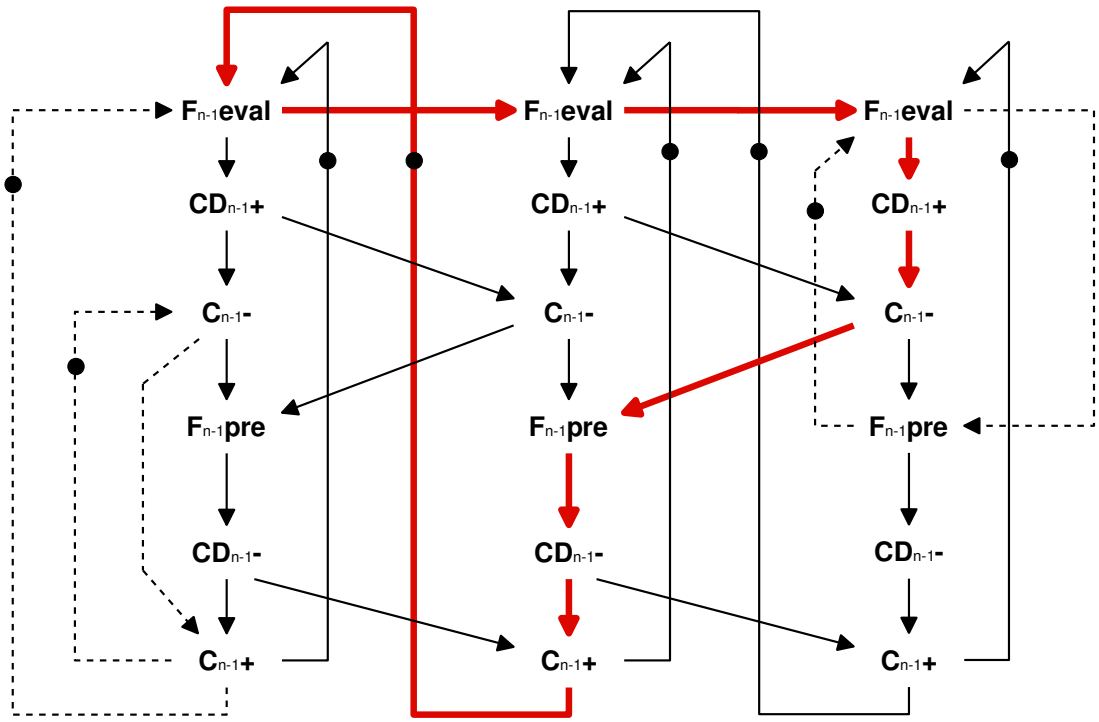


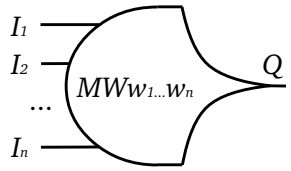
Figure 3.7: Marked graph of a 3-stage PCHB pipeline. The transitions that comprise the cycle time are highlighted with thick red lines. Dashed lines represent external interactions with the environment (e.g. with neighbor pipeline stages that are not shown).

assigned to inputs with the use of a hysteresis mechanism[1]. Figure 3.8 (a) shows a generic symbol for an NCL gate with a threshold function M and N inputs, each one with a weight w_i. According to the NCL gate function, each input can have a different weight value. If the weights are suppressed in the gate notation, weight 1 is assumed for all inputs.

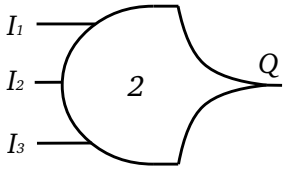
The output of an NCL gate switches according the following premises: (1) a high-to-low transition only occurs after all inputs go to logic level 0; (2) a low-to-high transition occurs when the sum of weights for inputs at logic level 1 reaches a value bigger than or equal to the gate threshold M. In case the inputs do no satisfy the threshold function, the output holds its previous state (either 0 or 1). These characteristics demonstrate how NCL gates are similar to a classic asynchronous component, the C-element. In fact, a basic C-element is a special case of NCL gate where $M = N$ and all input weights are 1. Figure 3.8 (b) illustrates a specific NCL gate with $M = 2$ and $N = 3$ with all weights equal to 1. Moreover, Figure 3.8 (c) shows the truth table for the gate in Figure 3.8 (b), indicating where the gate sets, resets and holds its output value (Q_{i-1}).

Regarding NCL gate implementations, Figure 3.9 shows the generic implementation of three often used NCL gate topologies: (a) Martin's (weak-feedback); (b) Sutherland's and (c) Moreira's [MAGC14]. Martin's topology employs two main logic blocks (SET and

[1]The original definition of threshold logic gates (TLGs) includes the threshold function and the weights, but not the hysteresis effect of NCL gates. In original TLGs, if the weight is not reached by the combination of inputs, the gate output is 0.



(a)



(b)

I_1	I_2	I_3	Q_i
0	0	0	**0**
0	0	1	Q_{i-1}
0	1	0	Q_{i-1}
0	1	1	**1**
1	0	0	Q_{i-1}
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

(c)

Figure 3.8: Elements of the NCL logic gate family: (a) a generic NCL gate with threshold M and weights $w_1 ... w_N$; (b) an actual instance of an NCL gate, with $M = 2$ and weights $w_1 = w_2 = w_3 = 1$; (c) the truth table of the gate depicted in (b).

RESET), and a pair of cross-coupled inverters: an output inverter and a weak-feedback one. The SET and RESET blocks are responsible for forcing logic values 1 and 0 to the output, respectively, while the inverters are used to latch the output in case nor SET neither RESET are active. Despite the fact that this topology is the most straightforward implementation, the weak-feedback structure compromises output integrity, specially in low-voltage supply scenarios. The Sutherland's topology, on the other hand, introduces two additional logic blocks to the feedback structure: HOLD0 and HOLD1. HOLD0 and HOLD1 controls the feedback circuit, to account for input combinations that hold the output logic value at one of the two possible logic values, i.e. the hysteresis behavior. Consequently, the Sutherland's topology mitigates the integrity issue of Martin's topology, at the cost of extra hardware. Finally, the Moreira's topology is a recent proposal to adapt even further NCL design to low-voltage scenarios. This topology proposes two main improvements: lower driving capacitances for SET/HOLD0 and RESET/HOLD1 logic blocks and reduced shorts while switching the output. Moreira's topology brings better speed, energy and leakage trade-offs, albeit initial experiments have pinpointed higher sensibility to SEEs in lower supply voltages [MAGC14].

3.3.1 The NCL Basic Template

An NCL pipeline stage has a structure similar to a WCHB stage, albeit the NCL template separates the latch logic from the combinational logic. For better understanding, Figure 3.10 illustrates the implementation of an NCL pipeline stage with an NCL register R_i, a completion detector CD_i, an NCL combinational logic block F_i and two handshaking signals: $ack(i)$ and $ack(i+1)$. Note that the stage has two dual-rail inputs A and B and one dual-rail output Q, assuming that F_i computes a function of two inputs. The NCL register is usually

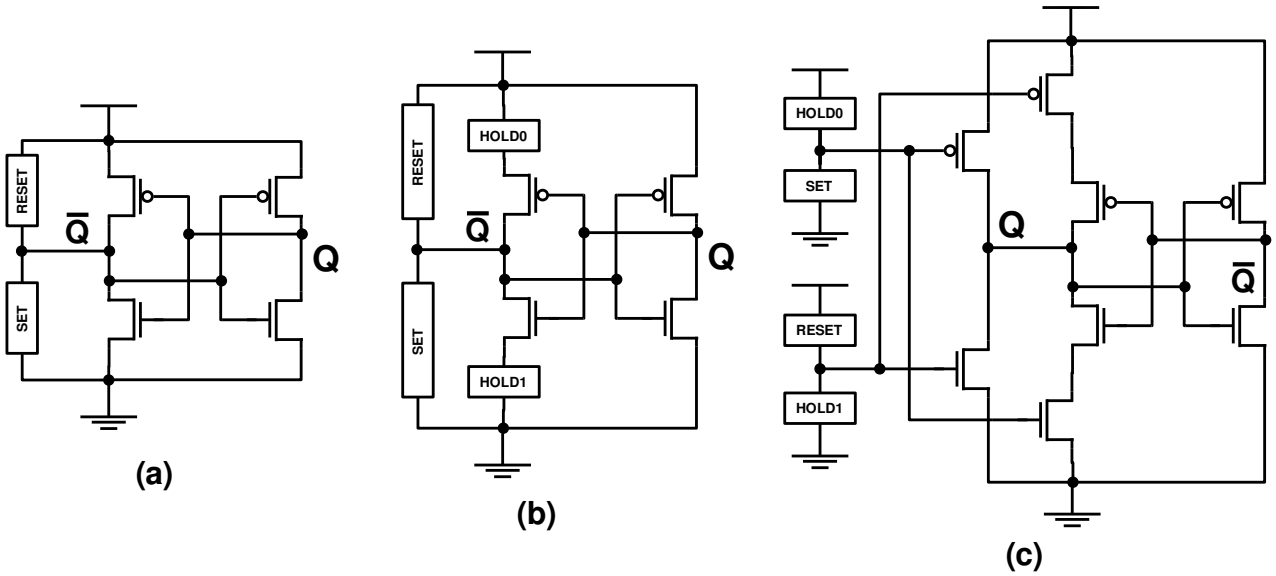


Figure 3.9: NCL gate topologies: (a) Martin's (weak-feedback); (b) Sutherland's and (c) Moreira's.

implemented with 2-input C-elements (equivalently, an NCL gate with $M = N = 2$). For each gate, an input connects to the input data and the other connects to the acknowledgement signal $ack(n+1)$ from the next stage. Basically, $ack(i+1)$ controls the status of the register, dictating when the register propagates its input or latches it. When $ack(i+1) = 1$, all register are able to propagate data tokens to their outputs. On the other hand, when $ack(i+1) = 0$, all registers are in the opaque mode and cannot propagate any data tokens – only NULL tokens. When data/NULL tokens propagate to all registers' outputs, the completion detector CD_i will signal through $ack(i)$, informing the previous stage the respective data/NULL token has been successfully propagated. Note that the NCL gate used in CD_i employs inverted logic, as CD_i lowers $ack(i)$ in the presence of a data token, and asserts $ack(i)$ when a NULL token is present.

To visualize how neighbor pipeline stages interconnect, Figure 3.11 shows the block diagram of a 3-stage NCL pipeline. As in previous templates, thick lines represent DI-encoded data channels and narrow lines indicate single bit signals. In the initial state, all data channels contain NULL tokens and all handshaking signals are set to 1. Consider that a data token arrives in $data\_i(n-1)$. As $ack(n) = 1$, register R_{n-1} can propagate the data token to its output. At this point, two parallel events occur: (i) CD_{n-1} identifies the presence of a data token, lowering $ack(n-1)$. By lowering $ack(n-1)$, stage $n-1$ signals to the previous stage – in this case, the external environment – that a data token has been stored. As a consequence, the environment will propagate a NULL token to $data\_i(n-1)$. Note, however, that R_n is not able to propagate the NULL token as the $ack(n)$ is still set; (ii) F_{n-1} computes the data token, propagating another data token to $data\_i(n)$. the register R_n in the next stage will store the propagated data token, indicating through $ack(n) = 0$. When $data\_i(n-1)$ contains a NULL token and $ack(n) = 0$, R_{n-1} can reset its output, propagating

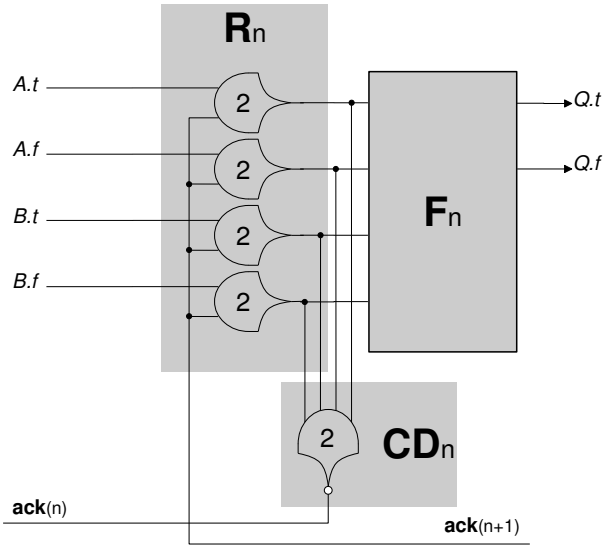


Figure 3.10: Example of a typical NCL pipeline stage. In this case, the stage has two dual-rail inputs A and B and one dual-rail output Q.

a NULL token to the entire stage. Consequently, $ack(n-1)$ is asserted by CD_{n-1} and F_{n-1} propagates a NULL token to $data_i(n)$, finalizing the communication cycle of stage $n-1$. Eventually, $ack(n)$ will be asserted again, making possible the propagation of a new data token through stage $n-1$.

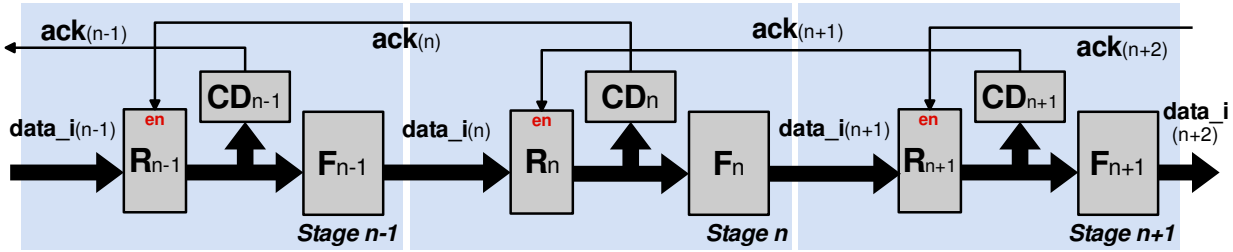


Figure 3.11: Block diagram of a 3-stage NCL pipeline.

Figure 3.12 represents the marked graph of a 3-stage NCL pipeline. Note that the cycle time is highlighted with thick red lines. Compared to WCHB, the NCL cycle time has three additional transitions, due to the use of separate register and combinational logic – see Equation 3.7 and 3.8. Fortunately, this separation avoids implementations with large transistor stacks in latching logic, implying that the additional transitions not necessarily bring performance overheads. These three additional transitions are included in the NCL forward latency, as can be seen in Equation 3.8. The forward latency covers the t_R and T_{eval} of all three stages, having a total of six transitions. Similar to WCHB, the NCL backward latency has only three transitions, covering two t_{CD} and one t_R.

$$CT_{NCL} = 4 \times t_R + 3 \times t_{eval} + 2 \times t_{CD} \tag{3.7}$$

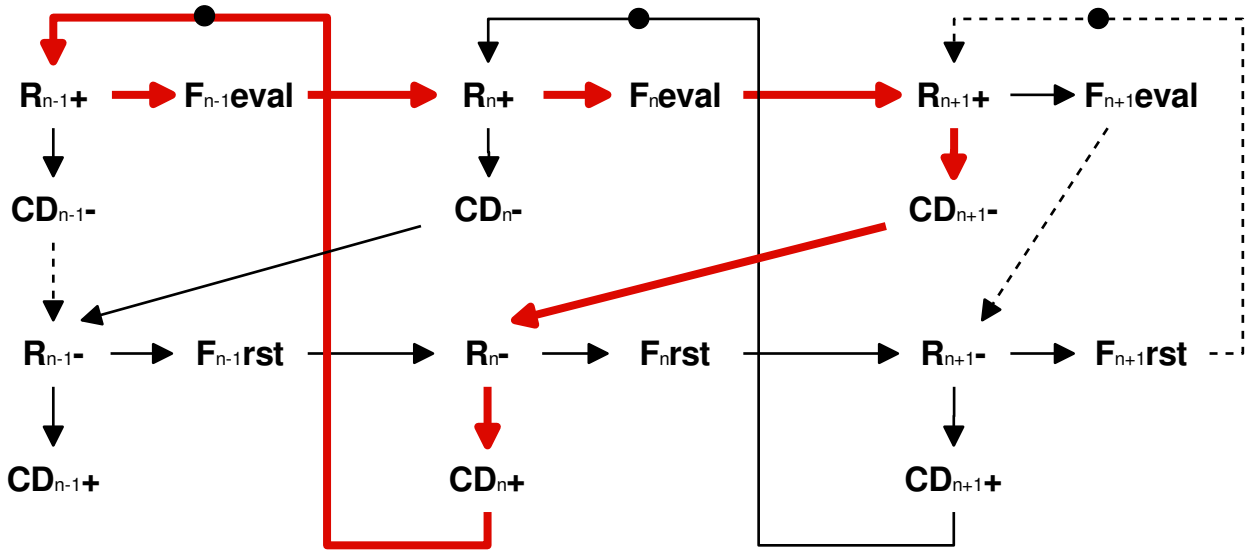


Figure 3.12: Marked graph of a 3-stage NCL pipeline. The transitions that comprise the cycle time are highlighted with thick red lines. Dashed lines represent external interactions with the environment (e.g. with neighbor pipeline stages that are not shown).

$$FL_{NCL} = 3 \times (t_R + t_{eval}) \tag{3.8}$$

$$BL_{NCL} = 2 \times t_{CD} + \times t_R \tag{3.9}$$

3.3.2 The Spatially Distributed Dual Spacer Null Convention Logic (SDDS-NCL) Template

The Spatially Distributed Dual Spacer NCL (SDDS-NCL) is a recently proposed QDI template [MTMC14], which brings the full power of synchronous logic optimizations tools to the design of NCL circuits [FB96]. To enable this, SDDS-NCL employs two types of gates: the conventional NCL family and the NCL+ family, proposed in [MOPC13]. The difference between the NCL and NCL+ families is that the latter relies on a different handshake protocol called return-to-one (RTO) [MGC12]. NCL+ gates have a functionality similar to NCL, but the assumption of the RTO protocol mandates the switching function of an NCL+ gate to be the reverse of its NCL counterpart. The gate output will only switch to 1 when all inputs are at 1 and to 0 when the threshold M is reached by the inputs at 0, subject to the same input weight rules. For other input combinations, the output keeps its previous value, exhibiting a similar hysteretic behavior.

The symbol to represent NCL+ gates is identical to the NCL symbol in Figure 3.8(a), except for a "+" symbol on its top right corner. When designing NCL+ gates, any topology of Figure 3.9 can be used. However, the SET/HOLD0 and RESET/HOLD1 blocks are implemented differently. Compared to its NCL gate counterpart, an NCL+ gate swaps RESET with

SET, and swaps HOLD0 with HOLD1. This exchange avoids big stacks of PMOS transistors that are present in NCL gates, which are implemented here with NMOS transistors.

Classically, no negative unate function was supported in either NCL or NCL+. However, SDDS-NCL enables to eliminate this restriction [MTMC14], since it mixes both RTZ and RTO protocols. Thus, to every conventional NCL and NCL+ gate there is a corresponding negative unate version gate, respectively INCL and INCL+. This can be useful for circuit synthesis optimizations, as internal inverters already present in these cells can be reused.

3.4 The Autonomous Signal-Validity Half Buffer (ASVHB) Template

The proponents of the Autonomous Signal-Validity Half Buffer (ASVHB) QDI template claim a focus on ultra-low power operation [HCGC15]. This template displays three main characteristics: (1) its structure employs integrated autonomous validity signals, which are used to simplify the circuit implementation; (2) it utilizes a fine-grained gate-level approach, which increases throughput by propagating data through a single-cell datapath pipeline; (3) it implements static logic only, which increases node output stability and circuit robustness.

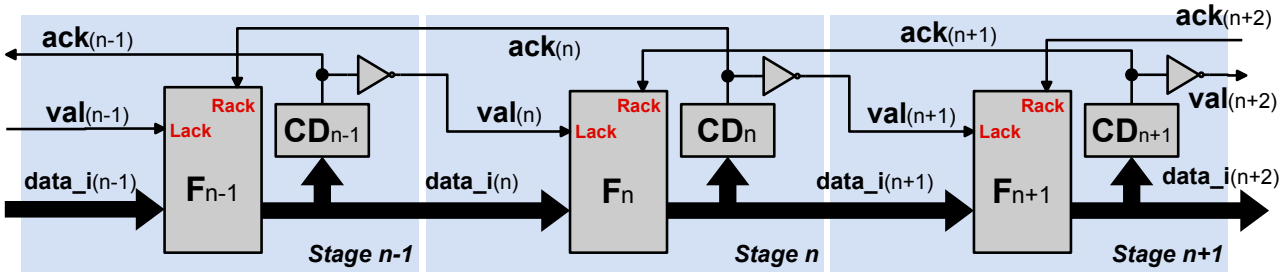


Figure 3.13: Block diagram of a 3-stage ASVHB pipeline. Each *val* signal represents the validity of each code block in the data input. Thick lines indicate DI-encoded data signals, usually an *m-of-n* code such as dual-rail or multiple *1-of-4* channels. The remaining lines represent single bit signals.

Figure 3.13 shows a 3-stage ASVHB pipeline. Regarding handshaking signals, each stage has two acknowledgement signals: *val* and *ack* – where the first, *val*, is in fact a set of signals, each of which provides the individual validity of a *code block* in the data input[2]. The completion detectors generate a single wire output, which is the Lack signal for the previous pipeline stage, and also the code block validity detectors (*val*) for the next

[2]A *code block* of a DI signal has a structure that depends on the code used. Assume for example the dual-rail code for representing two bits of data. We need four wires, implying two 2-wire code blocks. The same information can be represented using the 1-of-4 code, but in this case we would have a single 4-wire code block. For a more convoluted example, assume we want to represent an 8-bit information (for example, all characters of the ASCII-E code) using a number of 3-of-5 code blocks. The reader can verify that a single 3-of-5 code block can represent only seven distinct data in 5 wires (00111, 01011, ..., 11100). Thus, we need three 3-of-5 code blocks to represent the 256 valid codes in an equivalent 8-bit binary code, since 7x7=49, but 7x7x7=343. In this case, we are left with three 5-wire code blocks

stage. Different from other templates, ASVHB introduces individual validity signals *val* for each input data block. For example, a 2-operand, 1-bit ALU represented in dual rail code in ASVHB typically requires three validity inputs: one for each of the operands and one for the ALU operation input. Validity inputs in ASVHB have a function equivalent to the PCHB input completion detector *CD* block. In Figure 3.13, the pipeline stage is depicted with multiple input channels and only one output channel, which is an assumption of the template. When $data_i$ presents complete data and the corresponding *F* block processes this data, the completion detector *CD* will lower its output, acknowledging the preceding pipeline stage as well as notifying the succeeding pipeline stage that the data channel holds a data token. Note that acknowledging is signaled with 0 and validity with 1, which implies the use of an inverter before each *val*. At this point, input channels are free to change and produce a spacer. When all *val* = 0 and the succeeding pipeline stage acknowledges the output capture with *ack* = 0, output *data* is reset, inserting a spacer at the output as well.

The implementation of the ASVHB template is similar to that of other QDI templates, except for the validity signals to reset the output. For instance, Figure 3.14 illustrates a complete dual-rail implementation of a NAND/AND gate using ASVHB logic. The completion detection block and the inverter in the output are the same presented in Figure 3.13. The Pre-charge block is responsible for resetting the output, whereas the Evaluate block sets the output. If either Pre-charge or Evaluate are active, the output keeps its value unchanged. This happens due to the hysteresis mechanism implemented with the cross-coupled inverters. As ASVHB employs static logic, the "Hold 0" and "Hold 1" are used to guarantee the integrity of the output node when the hysteresis mechanism is the only active block.

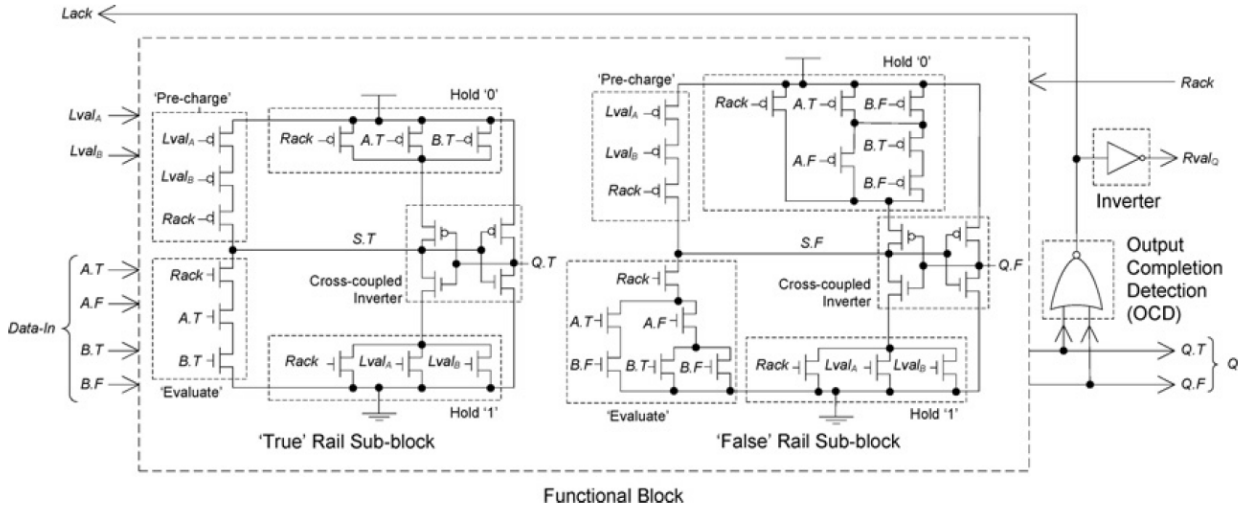


Figure 3.14: Dual-rail implementation of a NAND/AND gate using ASVHB logic. Extracted from [HCGC15].

Regarding the template behavior, Figure 3.15 shows the marked graph of the 3-stage ASVHB pipeline in Figure 3.13. As in the analysis of previous templates, the cycle time is highlighted through the three stages and is represented in Equation 3.10. The

ASVHB template presents a token propagation through the pipeline with three transitions (see Equation 3.11), which is similar to what occurs with the WCHB and PCHB templates. Moreover, Equation 3.12 describes the ASVHB backward latency. Similar to WCHB, ASVHB has a backward latency with three transitions albeit its precharge uses a smaller number of transistors, implying that ASVHB can provide a smaller backward latency than WCHB.

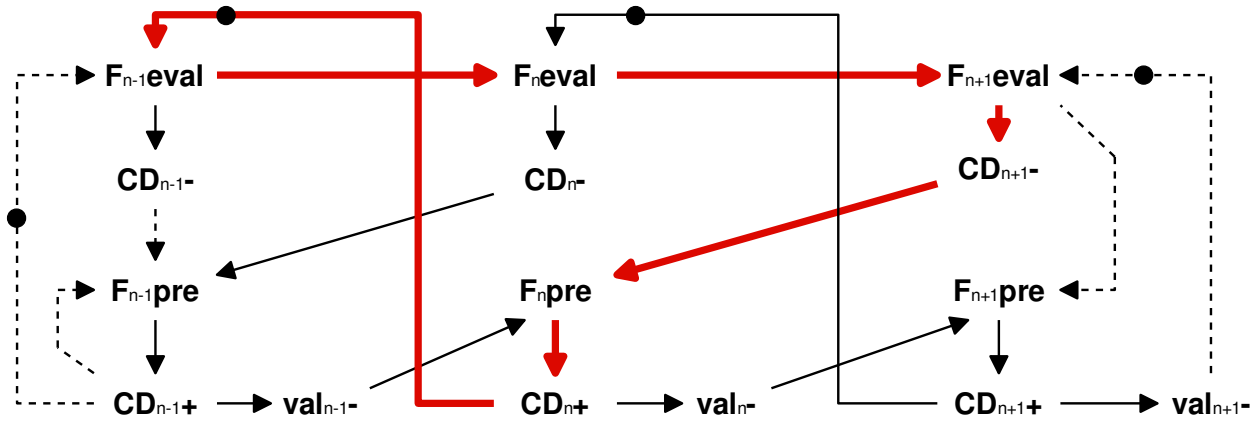


Figure 3.15: Marked graph of a 3-stage ASVHB pipeline. The transitions that compromise the cycle time are highlighted with thick red lines. Traced lines represent external interactions between neighbor pipeline stages that are not shown.

$$CT_{ASVHB} = 3 \times t_{eval} + 2 \times t_{RCD} + t_{pre} \tag{3.10}$$

$$FL_{ASVHB} = 3 \times t_{eval} \tag{3.11}$$

$$BL_{ASVHB} = 2 \times t_{RCD} + t_{pre} \tag{3.12}$$

3.5 The Sense Amplifier Half-Buffer (SAHB) Template

Introduced in [CHL+17], the Sense Amplifier Half-Buffer (SAHB) template is a recent alternative to implement high performance QDI circuits. Its main aspects can be summarized by five main points: (1) SAHB employs a standard 4-phase RTZ handshake protocol; (2) all SAHB cells employ sense-amplifier cross-coupled latches with positive feedback to optimize output evaluation; (3) it uses NMOS pull-up and pull-down networks with minimum transistor sizing; (4) evaluation and sense amplifier blocks are coupled, reducing internal switching; (5) all SAHB cells uses CMOS static logic, which makes this template an alternative for aggressive, dynamic VS. According to results in [CHL+17], SAHB presents better power (~64% lower), performance (~21% faster) and area (~6% smaller) results than PCHB. Moreover – differently from PCHB – the SAHB template is able to operate from super-threshold (1V) to sub-threshold operation (~0.3V), due to the use of static logic. The authors

in [CHL+17] consider the possibility to supply the evaluation and sense-amplifier latch blocks with distinct voltages. This allows an interesting approach for performance and energy optimization using VS. However, for the sake of simplicity, this Section focuses in the main SAHB concepts.

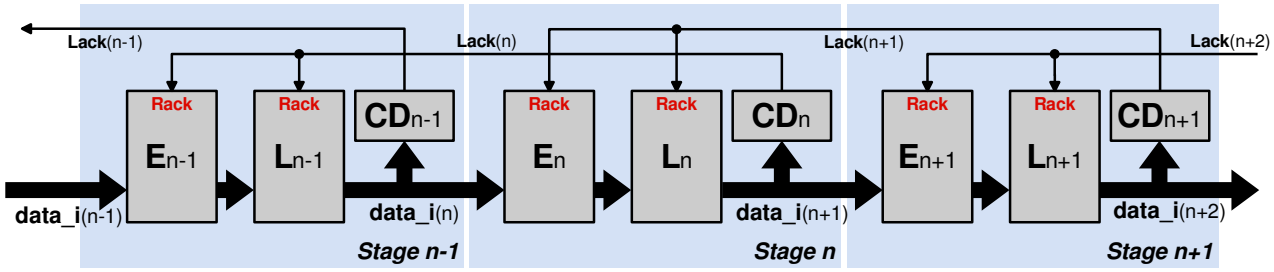


Figure 3.16: Architecture of a 3-stage SAHB pipeline. Thick lines indicate DI data signals. Usually, these data signals are encoded with an *m-of-n* encoding such as dual-rail. The remaining lines represent single bit signals. Note that *data_i* and *Lack* labels represent both positive and inverted values of its respective signals.

Figure 3.16 illustrates a 3-stage SAHB pipeline. Similar to ASVHB, SAHB also employs a fine-grained pipeline, as each SAHB cell receives and generates individual handshake signals. This type of structure reduces the circuit cycle time, as the critical path does not comprise multiple cells but just one. Consequently, a higher throughput can be achieved, albeit additional handshake circuitry is needed to ensure synchronism. Each stage comprises three main blocks: an evaluation block *E*, a sense-amplifier latch *L*, and a completion detector *CD*. There are also two handshake signals: *Lack* and *Rack*. Considering the three main blocks, Figure 3.17 illustrates the schematic of a SAHB buffer cell with: (a) its evaluation block and (b) a sense-amplifier latch integrated with output completion detection. The evaluation block is responsible to implement the circuit logic, evaluating and resetting the dual-rail output (*Q.t* and *Q.f*). Note that the evaluation block implements both pull-up and pull-down networks with NMOS transistors to reduce parasitic capacitance. The sense-amplifier latch is responsible for amplifying and latching the evaluation block output *Q.t* and *Q.f*, generating complementary output signals *nQ.t* and *nQ.f*. These complementary signals are propagated with their primary outputs and are also used by the evaluation block to reduce short-circuit currents [CHL+17]. Moreover, *nQ.t* and *nQ.f* are used as inputs to the completion detector, which generates the handshake signals *Lack* and *nLack*. Following a similar approach as the ASVHB template, the SAHB completion detector only employs a NAND and an inverter cell to generate all required signals. Of course, if more than one handshake signal exists in the pipeline stage and require synchronization, additional hardware is needed to implement the completion detection circuit.

In order to understand this template communication cycle, consider the following example. Initially, all *data_i* buses contains spacers and all handshake signals *Lack* are reset. For instance, consider that Stage $N-1$ already processed its input and propagates a valid data token to *data_i*(n). As $Lack(n+1) = 0$, E_n can evaluate the valid data token and

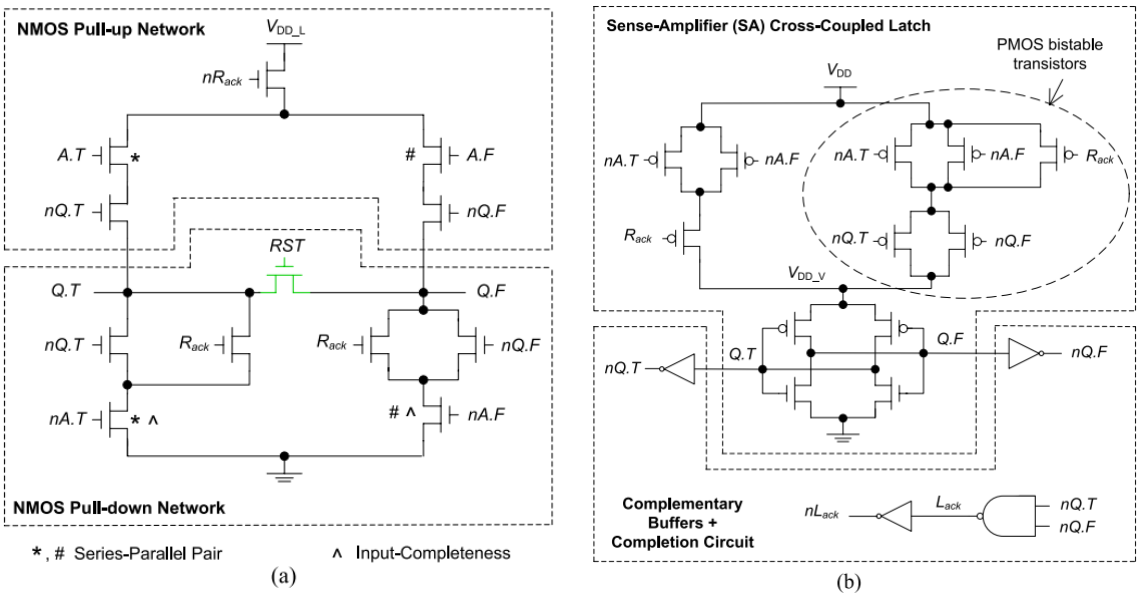


Figure 3.17: Schematic of a SAHB buffer cell: (a) evaluation block E_n and (b) sense-amplifier latch L_n with a output completion detector. Note that each block can be powered separately. In this example, the evaluation and sense-amplifier latch is powered by V_{DD_L} and VDD, respectively. Extracted from [CHL+17]

propagate to L_n, which latches a valid token in $data\_i(n+1)$. At this point, CD_n detects the valid token in its input and asserts $Lack(n)$, whereas the same token propagates to stage $N+1$. With $Lack(n) = 1$, stage $N-1$ can propagate a spacer, resetting $data\_i(n)$. At the same time, stage $N+1$ evaluates $data_i(n+1)$ and asserts $Lack(n+1)$. When both events occur – $data\_i(n) = NULL$ and $Lack(n+1) = 1$ – stage N can reset and propagate a spacer to $data_i(n+1)$. When stage $N+1$ resets $Lack(n+1)$, stage N has finished its communication cycle and is ready to process a new data token. This communication cycle can also be visualized in Figure 3.18, which shows the marked graph of the 3-stage SAHB pipeline in Figure 3.16. Note that the template cycle time is highlighted with thick red lines and corresponds to Equation 3.13. In total, the SAHB cycle time comprises 10 transitions, covering the evaluation phase of all three stages and the reset phase on the second stage. The forward latency is essentially composed by the evaluation and latching phases performed by the three evaluation and latch blocks. The backward latency corresponds to the reset phase of the second stage. Equations 3.14 and 3.15 depicts the forward and backward latencies, which respectively comprise six and four transitions.

$$CT_{SAHB} = 3 \times t_{eval} + 3 \times t_{L\_ON} + 2 \times t_{CD} + t_{rst} + t_{L\_RST} \tag{3.13}$$

$$FL_{SAHB} = 3 \times t_{eval} + 3 \times t_{L\_ON} \tag{3.14}$$

$$BL_{SAHB} = 2 \times t_{CD} + t_{rst} + t_{L\_RST} \tag{3.15}$$

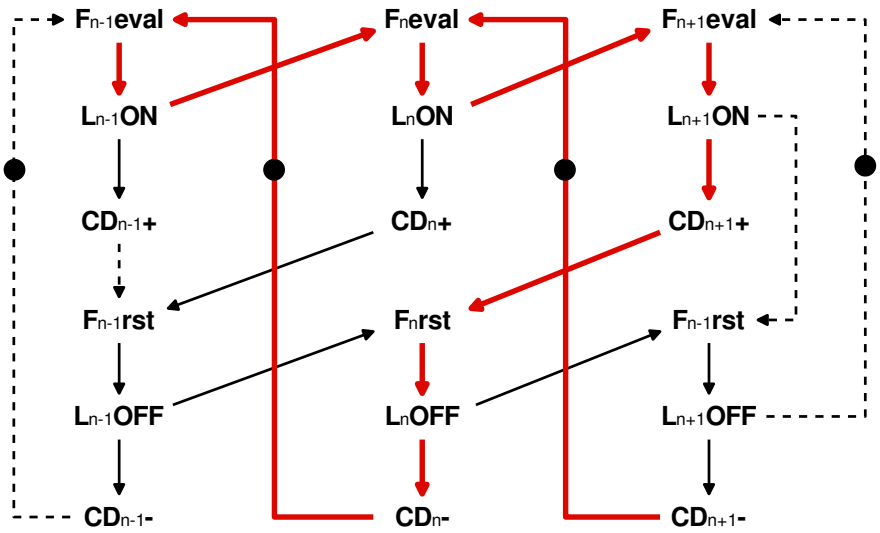


Figure 3.18: Marked graph for a 3-stage SAHB pipeline. The transitions that compose the cycle time are highlighted with thick red lines. Dashed lines represent external interactions between neighbor pipeline stages or the environment, and are not detailed.

3.6 The Register-Less Null Convention Logic (RL-NCL)

The RL-NCL template is a recent template proposal [CYP17] based on the NCL template and on MTCMOS [MDM+95] circuit design techniques. According to the Authors, registers can be responsible for a reasonable amount of the overall power and area consumption of an NCL circuit. To mitigate this overhead, RL-NCL suggests to optimize circuits by removing register circuitry and reducing the completion detection logic. By removing pipeline registers, RL-NCL introduces the handshaking structure in Figure 3.19, where a C-element is used in each stage to control the combinational logic block F_n. Different from the SCL template (see Section 3.7), the sleep signal is only used to control the combinational logic, leaving the completion detector block *CD* only sensitive to the *data_i* bus. The combinational logic blocks *F* in RL-NCL are similar to the ones used in the SCL template as both templates derived from MTCMOS. With the integrated sleep logic, the structure of logic gates comprises only the two main circuit blocks (*hold-0* and *set-to-1*) highlighted in Figure 3.20 (a). The *set-to-1* is responsible to set the output node during the evaluation phase, whereas *hold-0* only keeps the output node in 0 after the circuit is released from the *sleep mode*. Note that the logic responsible for resetting the output node is a NOR gate that drives the output node. When *sleep* = 1, the gate is in *sleep mode*. On the other hand, the is in *active mode* when *sleep* = 0. For a more detailed example, Figure 3.20 (b) shows the schematic of an MTCMOS threshold gate TH23. Different from traditional NCL gates, the RL-NCL avoids the cross-coupled inverters, significantly reducing area consumption. Area consumption is also reduced with the proposed completion detection scheme that only employs a single OR gate connected at the end of the critical path. This means that the

completion detector only detects a data token in the critical path. However, the analysis of this template reveals it contains several problems, which make it an unreasonable target for circuit implementations, as the discussion in the rest of this Section demonstrates.

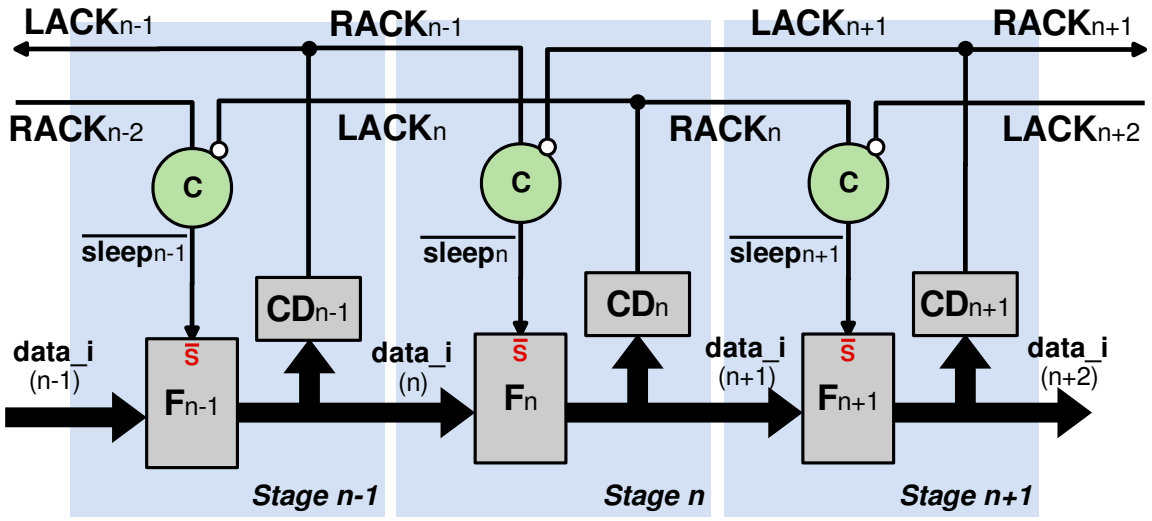


Figure 3.19: Architecture of a 3-stage RL-NCL pipeline. Thick lines indicate DI data signals. Usually, these data signals are encoded with a *m-of-n* encoding such as *1-of-2*. The remaining lines represent single bit signals. Note that is this template, the sleep signal is active-low.

The proponents of RL-NCL pinpoint that all threshold gates of the combinational blocks begin the evaluation/reset process at the same time, as all gates are sensitive to the same sleep signal. In that way, it would be possible to assume that when the output bit of the critical path has been processed, all output bits are already processed and stable. This assumption brings some important aspects that should be considered. First, if all gates begin the evaluation/reset process at the same time, this implies that the sleep signal *must* reach all gates at the same time. In order to guarantee this, the designer should treat each sleep signal as a clock signal, by implementing a balanced delay tree to control the skew of each sleep signal. Second, if PVT variations are considered, there is a possibility that the estimated critical path generates its output before others paths, compromising the input completeness of the circuit.

Regarding pipeline operation, the RL-NCL pipeline behavior can be described as follows. In the initial state, all pipeline stages are in *sleep mode* ($\overline{sleep} = 1$), generating NULL tokens in all *data_i* buses; Consider now that a valid data token arrives in *data_i*(n) and $RACK_{n-1}/LACK_{n-1}$ is asserted. With $RACK_{n-1} = 1$ and $Lack_{n+1} = 0$, $\overline{sleep_n}$ deasserts, removing Stage n from *sleep mode*. Consequently, F_n is able to compute its input and propagate a data token to *data_i*($n+1$). Next, the completion detector CD_n detects the data token and asserts its output ($LACK_n = RACK_n = 1$). As $RACK_n = 1$, the C-element in stage $n+1$ forces $\overline{sleep_{n+1}} = 0$, allowing this time to F_{n+1} to compute *data_i*($n+1$) and propagating a data token to *data_i*($n+2$). The data token in *data_i*($n+2$) can now be detected by CD_{n+1},

asserting this output ($LACK_{n+1}$ = $RACK_{n+1}$ = 1). Assuming that $RACK_{n-1}$ have already been reset – indicating that stage $n-1$ has been reset and $data_i(n)$ contains a NULL token – $LACK_{n+1}$ = 1 will force $\overline{sleep_n}$ = 1, putting Stage n in *sleep mode* again, finishing the communication cycle.

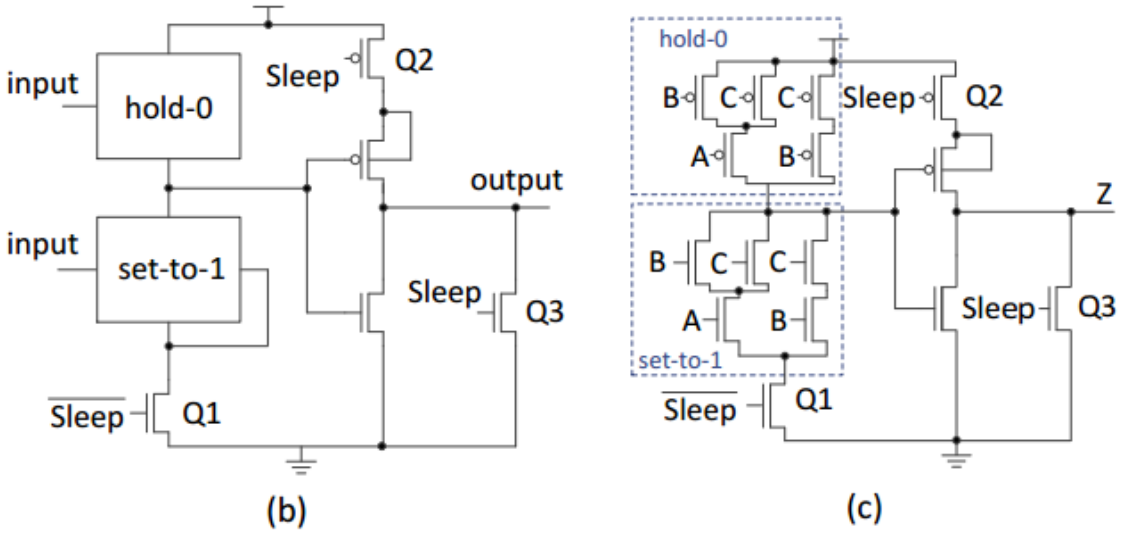


Figure 3.20: RL-NCL combinational logic: (b) generic schematic highlighting the main logic blocks (set-to-one, hold-0 and sleep logic) and (c) schematic of a MTCMOS threshold gate TH23. Extracted from [CYP17].

The pipeline behavior previously described describes the operation of the RL-NCL pipeline assuming that the pipeline always consumes new data tokens – producing no stalls. However, if a stall occurs in the RL-NCL pipeline, there is a possibility that data be overridden while stalling. In order to understand this possible hazard, consider the scenario described next. First, consider that the external environment in Figure 3.19 is not consuming and data token in $data\_i(n+2)$, simulating a possible environment stall. Because of that, $LACK_{n+2}$ is fixed in 0, which locks stage $n+1$ in the *active mode*. In addition, due to the presence of a data token in $data\_i(n+2)$, $LACK_{n+1}$ is set, allowing stage n to enter *sleep mode*. At this point, if $data\_i(n)$ receives a NULL token, a data override may occur. With $data\_i(n)$ = *NULL*, $RACK_{n-1}$ resets and force stage n to *sleep mode* ($\overline{sleep_n}$ = 0) – remember that $LACK_{n+1}$ = 1. Consequently, a NULL token is generated in $data\_i(n+1)$. As stage $n+1$ is locked in *active mode* it can process its input. As F_{n+1} is a purely combinational logic, the NULL token will propagate to $data\_i(n+2)$, erasing the data token before it has propagated. Due to this hazardous behavior, the RL-NCL is discarded from analysis herein.

3.7 The Sleep Convention Logic (SCL) Template

SCL is an asynchronous design template [BZF+08] [PSAA16] inspired on NCL [FB96]. Its structure presents two main characteristics: NCL with early completion [Smi02] and fine-

grained Multi-Threshold CMOS (MTCMOS) power-gating [ZSD10]. In fact, SCL was initially labelled as Multi-Threshold Null Convention Logic (MTNCL) [BZF+08]. Compared to NCL, SCL brings architectural and design modifications that may result in area and performance advantages [PSAA16]. Figure 3.21 shows a 3-stage SCL pipeline. Each stage contains a combinational logic block *F*, a register *R*, a completion detector *CD* and a settable C-element *C*[3].

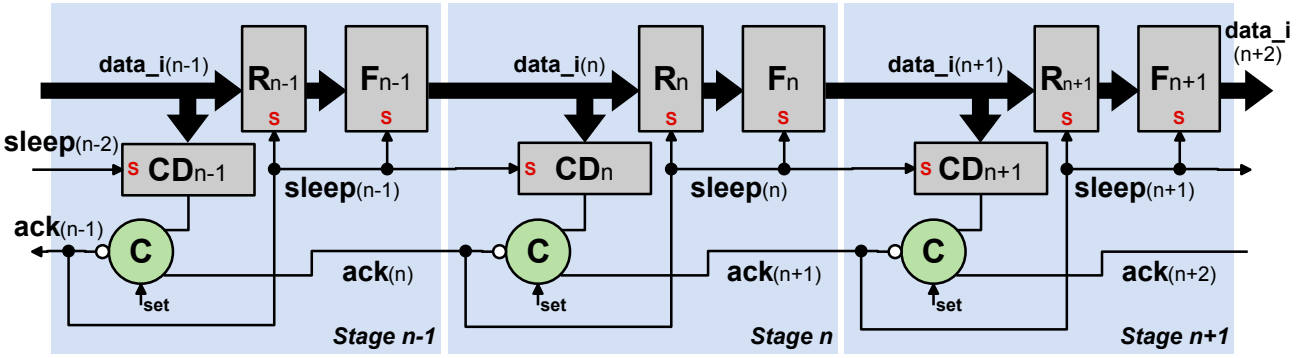


Figure 3.21: Architecture of a 3-stage SCL pipeline. Thick lines indicate DI data signals. Usually, these data signals are encoded with an *m-of-n* encoding such as *1-of-2* (dual-rail) or *1-of-4*. The remaining lines represent single bit signals.

The SCL template integrates the acknowledgement signal *ack* with a sleep mechanism. Basically, when a pipeline Stage n acknowledges a previous stage $n-1$, it also resets the sleep signal, which takes stage n out of sleep mode. To better understand the SCL pipeline operation, consider the structure in Figure 3.21 at a "sleep" state: all *data_i* = *NULL* (contain spacers) and all *ack* = *sleep* = 1, i.e. all pipeline stages are in sleep mode. Initially, $sleep(n-2)$ is reset, awaking CD_{n-1} to check the validity of $data\_i_{(n-1)}$. When $data\_i_{(n-1)}$) presents a valid data token, CD_{n-1} asserts its output, which forces the output of C_{n-1} to 0 ($ack_{(n-1)} = 0$). Remember that initially $ack_{(n)} = 1$. At this moment, two concurrent events take place, as the output of C_{n-1} forks to two paths: (i) the acknowledge signal $ack_{(n-1)}$ is lowered, signaling that stage $n-2$ confirms the data validity and (ii) the stage $n-1$ sleep signal is disabled ($sleep_{(n-1)} = 0$), awaking R_{n-1}, F_{n-1} and CD_n. Enabled, register R_{n-1} and the combinational block F_{n-1} can now propagate their inputs, generating a valid data token in $data\_i_{(n)}$, which will be detected by CD_{n-1}. Next, Stage n executes the same process described above for Stage $n-1$, and so on. After propagating a data token, stage $n-1$ must wait for two event to enter again in sleep mode: (i) the acknowledge signal $ack_{(n)}$ is lowered, confirming that stage n received a data token and (ii) the sleep signal $sleep_{(n-2)}$ is asserted, generating a NULL token in $data\_i(n-1)$ and resetting the output of CD_{n-1}. These two event are synchronized by the C-element, which asserts $ack_{(n-1)}$ and switches the stage $n-1$ to sleep mode. In sleep mode again, stage $n-1$ ends its communication cycle and can receive a new data token.

[3] In [PSAA16], Authors refer to this C-element as "resettable", but this is inaccurate. In a reset state, the SCL template must set all C-element outputs to 1 to maintain the pipeline stages in sleep mode ($sleep(n) = 1$).

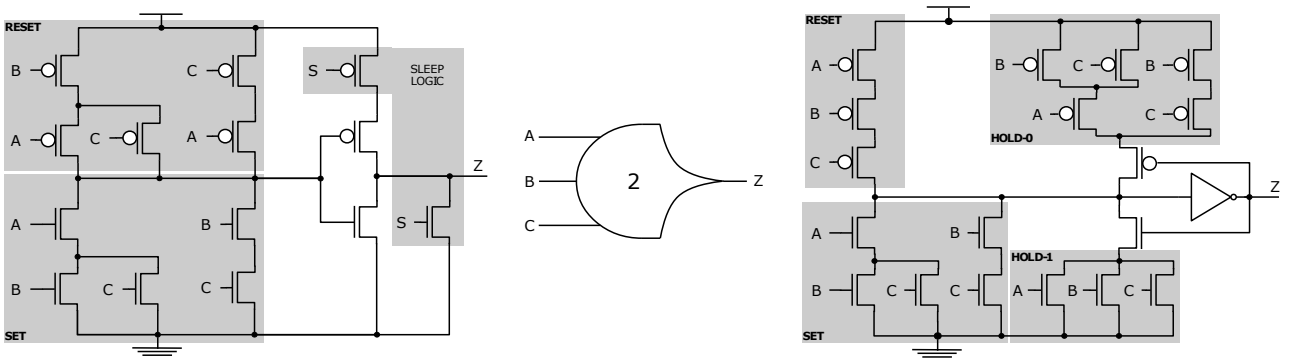


Figure 3.22: A comparison of SCL and NCL gates using the TH23 gate as example: (a) transistor-level implementation of the TH23 SCL gate; (b) TH23 symbol; (c) transistor-level implementation of the TH23 NCL gate. Note that both SCL and NCL implementation use the same symbol, but the SCL version has an additional input: the sleep signal, not shown in the symbol.

The combinational logic blocks F basically consist of SCL gates that couple a threshold function with positive integer weights assigned to inputs and power-gating logic. Compared to NCL gates, SCL gates present significant area reduction, due to the fact that the latter only uses two logic blocks from the original definition of NCL gates: the HOLD0 and SET [FB96]. The remaining logic blocks (HOLD1 and RESET) are replaced by the sleep logic, which is responsible for generating logic 0 at the output. This logic optimization removes the sequential characteristic that are present in all NCL gates, due to the removal of the hysteresis mechanism. An SCL gate is named according to its threshold value, equivalent to the convention stated for NCL gates. Following the terminology presented in [PSAA16], an SCL gate is denoted by THMWw_1, ..., w_N, where N is the number of inputs, M is the threshold of the gate, and w_1, w_2, ..., w_N are the input weights. If all weights are 1, their mention is omitted from the gate name. As an illustration, Figure 3.22 shows the transistor-level implementation of a TH23 SCL and a TH23 NCL gates, each with threshold=2, 3 inputs, and all input weights=1. Weight information is accordingly omitted from the gate denomination.

A completion detection block CD_i is responsible for identifying valid data and spacers in its $data(i)$ input. The CD_i implementation is similar to completion detectors from NCL, except for the fact that SCL completion detectors use SCL gates, which allow incorporating the sleep logic and further reduce static power consumption. Figure 3.23 shows the a gate-level implementation of an SCL completion detector. The first logic level is composed by TH12 gates, which operate as (and are, in fact) OR gates, and the following logic levels utilize THnn gates ($2 \leq n \leq 4$), i.e. n-input C-elements. The TH12 gates check the validity of each dual-rail signal while the following logic levels combine the results of all TH12 gates into a data validity control signal.

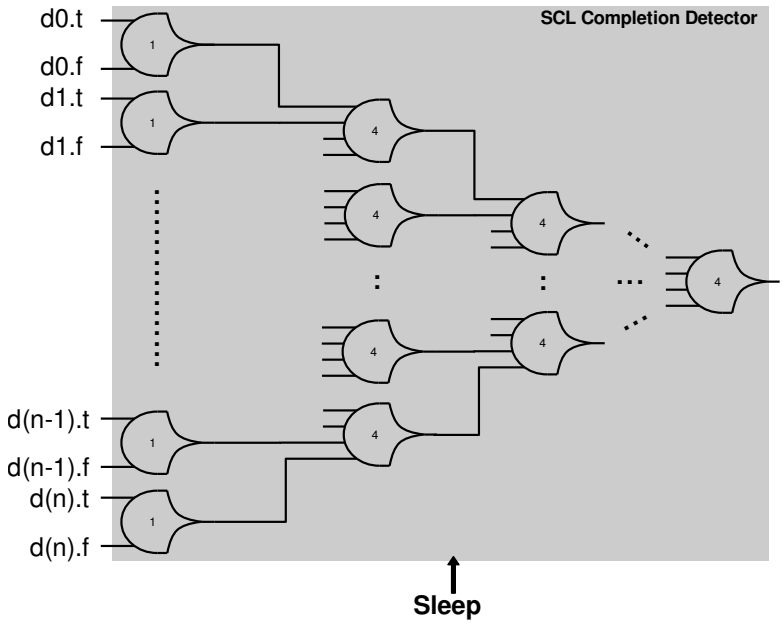


Figure 3.23: Gate-level implementation of a SCL completion detector.

Figure 3.24 presents the structure of a single bit SCL register. The SCL register employs a two-level logic with sleep transistors and an unconventional latch. Figure 3.24(a) shows the gate-level representation of the register. Basically, the SCL register uses two NOR gates, where the first NOR gate has an NMOS power gating transistor and the second NOR gate partially implements the sleep logic. Note that the first NOR receives the output Q as input, implementing the latch feedback signal. In addition, Figure 3.24(b) depicts the transistor-level representation of the register, highlighting the sleep logic and the feedback structure (HOLD-1).

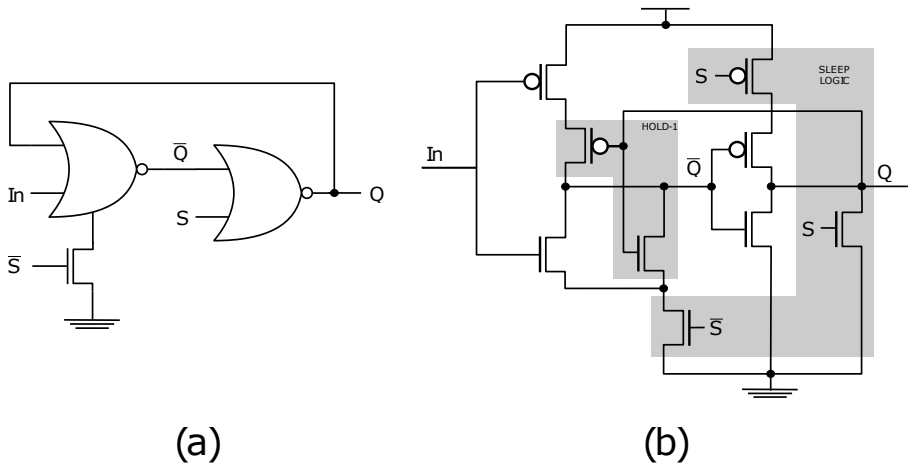


Figure 3.24: A single bit SCL register: (a) gate-level implementation; (b) transistor-level implementation. Note that the NMOS gating uses the inverted value of S ($\overline{S}$), which implies that an inverter must be added to the structure.

Due to the register structure depicted in Figure 3.24, register input *In* can only write 1 or keep 0 in output Q. In case Q = 1, the HOLD-1 structure disables the PUN of the first NOR gate. This implies that the sleep logic must be asserted to reset the output.

For instance, consider the register in Figure 3.24 operating under the stimuli showed in Figure 4.1: the initial input values are $In = 0$, $S = 1$. With these inputs, the sleep logic is enabled and is forcing $Q = 0$. Next, In is asserted but Q maintains its value since $S = 1$. When S is reset, the register can compute In and write $Q = 1$. At this point, the register will maintain $Q = 1$ even if In is reset. Therefore, the register has to wait for $S = 1$ to reset Q.

Figure 3.25 illustrates the marked graph of a 3-stage SCL pipeline with its cycle time indicated by thick red lines. It is possible to see that the cycle time structure differs significantly from previous templates. Usually, asynchronous design templates are meant to optimize the propagation of the data through the pipeline – in other words, to optimize the forward latency. This explains why the forward latencies of previous templates all have similar transitions. The SCL template, however, implements a longer forward latency and a shorter backward latency, which are represented in Equation 3.16 and Equation 3.17, respectively. Notice the the forward latency has 10 transitions – different from the 3 transitions of WCHB/PCHB – and incorporates the completion detector, evaluation blocks, a register and a C-element. This happens due to the fact that when new data arrives at an SCL stage, the stage is sleeping and must wake up. To do that, new data must be detected to lower the stage sleep signal, allowing this to compute and propagate new data. Regarding backward latency, the scenario is completely different. After the third stage detects the presence of new data in its input and wakes up, the only gate in the acknowledgement path is a single C-element. Compared to WCHB and PCHB, which have backward latency with 3 and 5 transitions respectively, SCL has a significant lower backward latency counting only one transition. Equation 3.18 shows the SCL cycle time, combining both forward and backward latencies.

It is important to emphasize that the SCL cycle time is related to a possible issue regarding the pipeline functionality. In fact, this issue is further discussed in Chapter 4 which covers the analysis of the SCL isochronic fork assumptions.

$$FL_{SCL} = 3 \times t_{CD} + 3 \times t_C + 2 \times t_{Ren} + 2 \times t_{eval} \tag{3.16}$$

$$BL_{SCL} = t_C \tag{3.17}$$

$$CT_{SCL} = 3 \times t_{CD} + 4 \times t_C + 2 \times t_{Ren} + 2 \times t_{eval} \tag{3.18}$$

3.8 Discussion

The QDI templates presented in this Chapter introduce interesting operation aspects and unique features, and also bring specific trade-offs that need to be evaluated before committing to use one of these in the design of a circuit. In order to synchronize with the context of this work, it is important to consider the trade-offs of each QDI template regarding

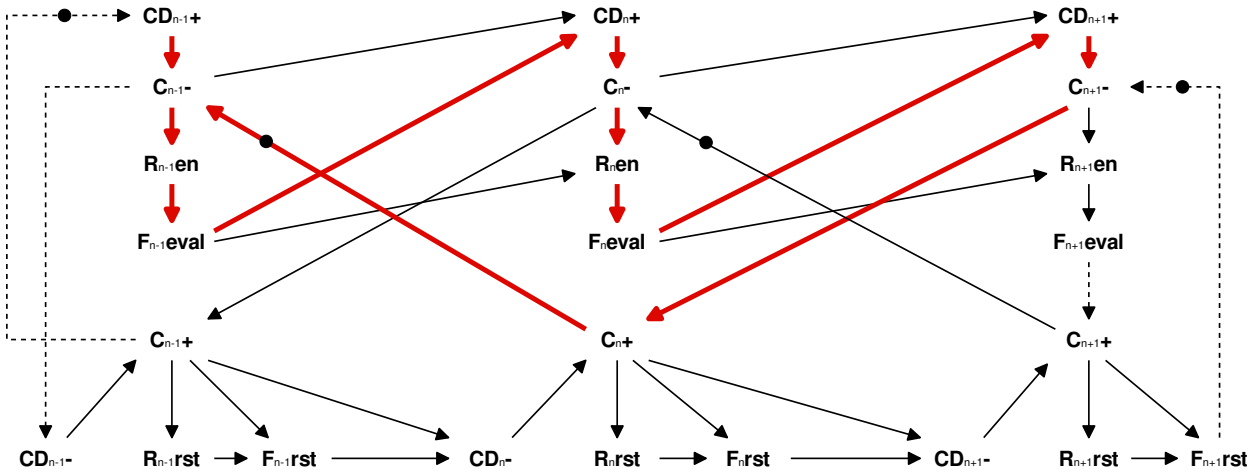


Figure 3.25: Marked graph for a 3-stage SCL pipeline. The transitions that compose the cycle time are highlighted with thick red lines. Dashed lines represent external interactions between neighbor pipeline stages or the environment, and are not detailed.

their applicability to an ultra-low power environment, specially in sub-threshold operation. To support the discussion, Table 3.1 indicates the main qualitative and quantitative aspects of each template, focusing on its use in ultra-low power scenarios. Table 3.1 also informs the lowest reported supply voltage for each template and the presence of CAD support in the literature.

WCHB is one of the most straightforward QDI templates to implement. Its buffer has a short cycle time when compared to other QDI templates, albeit its implementation complexity is typically high and it is not feasible to implement complex functions with a single level of weak-conditioned logic [BOF10]. Building complex functions in the buffer leads to big stacks of PMOS and NMOS transistors, which impacts circuit performance significantly. Because of that, WCHB is often used to implement buffers and control logic only, leaving the combinational logic to be implemented with DIMS or NCL, for instance.

Compared to WCHB, the PCHB template presents better performance, because it utilizes domino logic, avoiding long chains of series PMOS transistors, but increases implementation complexity and has multi-output gates that makes semi-custom design more challenging. Moreover, PCHB comprises fine-grained gate-level pipeline design, implementing single-cell pipeline stages and increasing the circuit throughput. Its implementation relies on staticized dynamic logic. Applying this type of logic may compromise circuit operation under very low supply voltages and is thus not recommended for sub-threshold operation.

NCL can be a good candidate for sub-threshold operation as pinpointed in [JSL+10]. Despite the fact that all NCL gates employ a hysteresis mechanism in its logic, it is possible to adopt the static logic implementation to guarantee the output integrity while operating under very low supply voltages. On the other hand, static logic implementations lead to large PMOS stacking for more complex gates, which can degrade circuit performance and com-

plicate transistor sizing. In the same context, SDDS-NCL includes similar trade-offs as NCL, because it also utilizes the same gate implementations, albeit with the addition of NCL+ gates. According to [MNM+14], the use of NCL+ provides lower leakage power and better energy efficiency, at the cost of an increase in forward propagation delay, when compared to NCL. This trade-off is interesting for ultra-low power design as the leakage power is a significant negative side-effect of sub-threshold operation. Moreover, it is important to pinpoint that the optimizations provided by SDDS-NCL are compatible with conventional electronic design automation (EDA) tools [MNM+14].

The authors of [HCGC15] indicate that the ASVHB template is suitable for sub-threshold operation, reporting a 32-bit Arithmetic and Logic Unit (ALU) that operates under a $V_{dd} = 0.2V$ supply. As advantages, this template improves circuits in two aspects: throughput and implementation. ASVHB improves circuit throughput by using single-cell pipeline stages, which are hard to achieve when using e.g. WCHB/DIMS. Regarding implementation, ASVHB utilizes the input validity data signals to simplify its implementation. Similar to NCL, this template also employs a hysteresis mechanism with a staticized logic implementation. Again, this type of implementation improves the output value integrity but employs big stacks of PMOS transistors if the circuit implements complex logic functions.

The SAHB template shows compatibility with low-voltage operations, due to the use of static logic. This is highlighted in [CHL+17] with a prototyped 64-bit Kogge-Stone adder operating in $V_{dd} = 0.3V$. Similar to ASVHB, SAHB uses a fine-grained pipeline structure to maximize throughput. When compared to PCHB, SAHB presents performance and energy improvements, albeit its area consumption keeps relatively high and basic dual-rail gates employ a large number of stacked transistors. For instance, an XOR/XNOR SAHB gate has five stacked NMOS transistors, which is not recommended when operating in low voltage scenarios [ILL+13] [PN13]. To optimize even further the design, the same Authors introduce the Template-based Cell-Interleave Pipeline (TCIP) [HLN+16], which mixes four QDI templates: WCHB, PCHB, ASVHB and SAHB. Using a initial set of performance constraints, these Authors define which template will be used for each pipeline stage, to keep area overhead at its lowest.

The SCL template is also a possible alternative for circuits operating in the sub-threshold region. It is in fact the template considered most promising among all evaluated in this work. Its logic optimization significantly reduces area requirements and removes the hysteresis mechanism that is present in other QDI templates. In addition, the fine-grained sleep logic reduces leakage power, increasing its applicability in ultra-low power environments. Differently from other templates, SCL brings an optimized backward latency albeit its forward latency is significantly higher than that of other templates. The larger forward latency could impact performance, specially in sub-threshold operation, and the low backward latency could be related to a possible timing assumptions in the handshake process, suggesting that the template must be evaluated and enhanced if chosen. Considering SCL

optimizations, the SCL template can in principle be combined with the advantages provided by SDDS-NCL, which can reduce even further leakage power consumption and increase energy efficiency. The proposal of an SDDS-SCL QDI template can provide an optimized point for ultra-low power operation. These advantages point to the SCL template as an elegant approach to design ultra-low power circuits, albeit the SCL template brings timing assumptions and performance overheads that need to be evaluated in order to determine the template stability and feasibility for ultra-low power operation. The present work contributes by investigating the issues related to the SCL in the next Chapter.

Table 3.1: Comparison table with main qualitative and quantitative aspects of the selected QDI templates. NA = Information Not Available.

	WCHB	**PCHB**	**ASVHB**	**SAHB**	**NCL**	**SDDS-NCL**	**SCL**
Implementation	Buffer and control only	Domino only	Autonomous validity signals	Sense amplifier based	Static threshold logic	RTZ/RTO protocol mixing	Fine-grained sleep logic
Pipeline Design	Fine-grained, gate-level				Coarse-grained, block-level		
Power-Gating	No						Yes
External Latches	Required	Not Required			Required		
Reported Minimum V_{dd} @ 65nm	0.2V	0.2V	0.2V	0.3V	0.15V	0.15V	NA
Hysteresis Mechanism on Logic	Yes						No
Reported CAD Support	Yes	Yes (proprietary)	No	No	Yes (proprietary)	Yes (combinational logic only)	No

3.9 A Circuit-based Quantitative Template Comparison

To assess the trade-offs of the chosen QDI templates, this works evaluates performance and power characteristics through a case study circuit using four selected templates: NCL, NCL+ ASVHB and SCL. The experiments rely on the design of an 8-bit, dual-rail Kogge-Stone adder operating in super-threshold and sub-threshold regions. Figure 3.26 illustrates a single-rail implementation of this adder with its three basic logic blocks. Besides the adder itself, the case study also includes an 8-bit input register, a completion detector and extra handshake hardware, which were implemented according to each template specification. For instance, the SCL-based adder uses SCL registers to implement the input registers, whereas the NCL-based one uses a conventional WCHB implementation. These additions allow a better estimation regarding cycle time, power and energy as the case study includes not only the combinational part but also sequential and synchronization parts. All four implementations were described at the transistor level using the SPICE language and addressing the 65nm Bulk CMOS technology from STMicroelectronics. The adopted transistor sizing for these experiments follows the strategy described in [HCGC15], which employs minimum width sizing to reduce leakage and area.

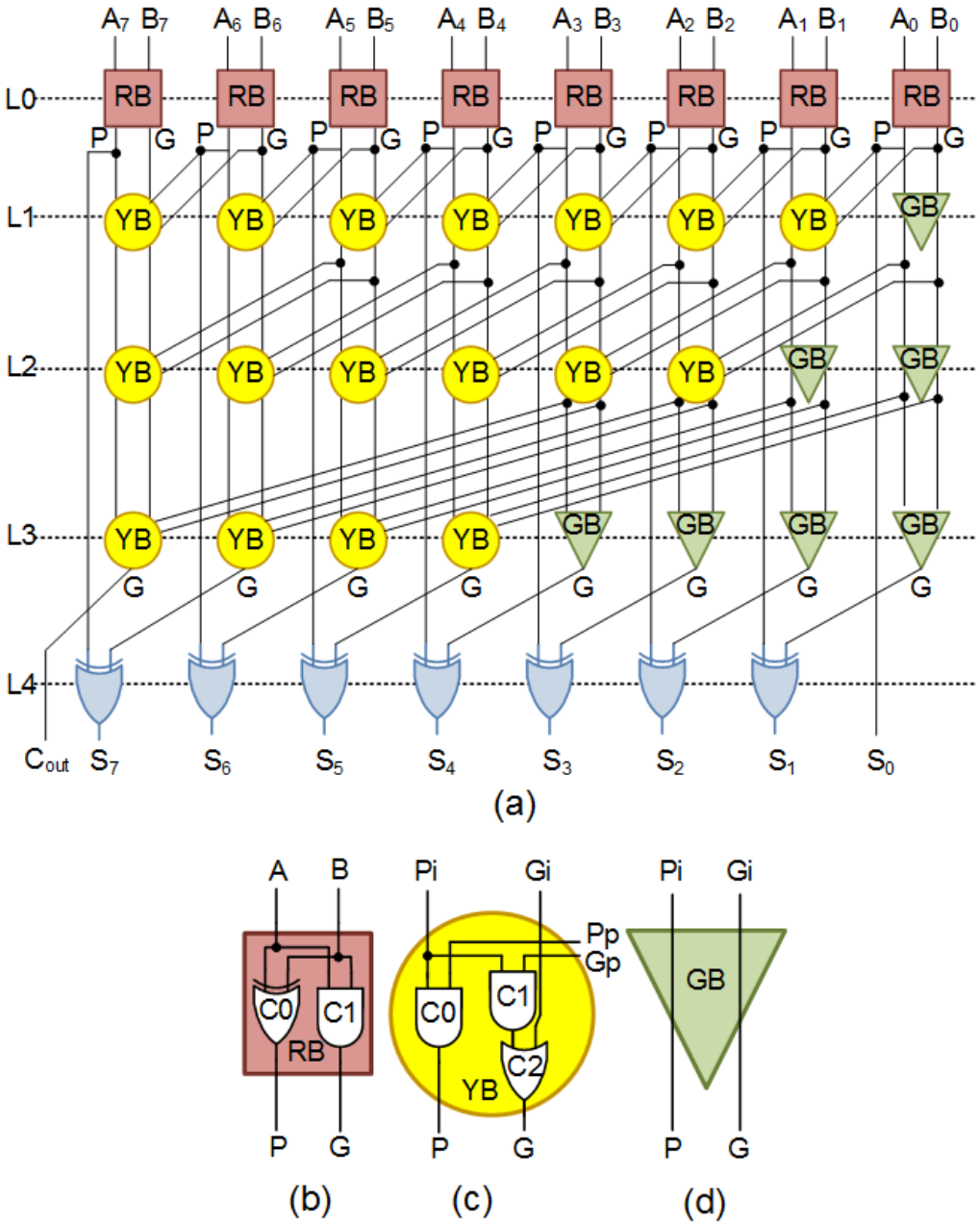


Figure 3.26: Single-rail implementation of the 8-bit Kogge-Stone adder and its basic blocks: (a) block diagram; (b) red box; (c) yellow box; (d) green box.

Figure 3.27 shows an overview of the simulation environment structure. This simulation environment incorporates a mixed-signal (VHDL-AMS) tesbench, which instantiates the SPICE description of each case study and where verification blocks are described in SystemC. The verification block generates the input stimuli, checks the outputs correctness and measures the cycle time characteristics (forward and backward latency). Moreover, the environment also captures leakage power, dynamic power and energy consumption using measurements from the SPICE setup. As some templates have different I/O signals and/or data encodings, the simulation environment implements dedicated interfaces and stimuli sets for each template.

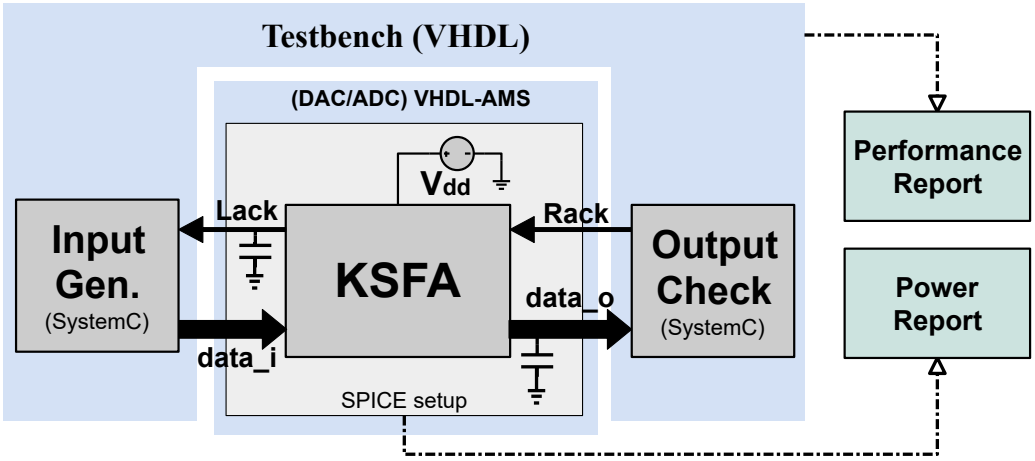


Figure 3.27: Simulation environment structure. The AMS-VHDL environment implements the ADC/DAC interfaces, allowing the communication between the case study circuit and the verification logic.

During simulation, the environment evaluates all implementations with nominal and sub-threshold supply voltage, in a typical process corner (TT) and at room temperature (25℃). The obtained simulation results are collected in Table 3.2, which indicates performance and power characteristics of each QDI template operating with $V_{dd} = 1V$ and $V_{dd} = 0.2V$. Note that all results are normalized to the results of the NCL-based case study, which is thus used as reference in the discussion. The cycle time captures not only the data propagation latency through the case study circuit itself but also the extra handshake latency, giving a better perspective on the templates relative performance. Area estimation is approached using the circuit transistor count for each template. Energy per operation (EPO) and leakage power consumption charts provide a power analysis regarding all covered templates. While EPO pinpoints the energy consumption when the circuit is active, leakage power shows how much the circuit consumes in standby mode. The Energy-Delay Product (EDP) provides a vision of the trade-off between performance and energy consumption. Similarly, the Leakage-Delay Product (LDP) focuses on the trade-off between performance and leakage power consumption.

The obtained cycle time results in super-threshold suggest that almost all templates present slightly better results than NCL, except the SCL template. More specifically, NCL+ and ASVHB have 4% and 8% lower cycle time than NCL, respectively. Meanwhile, the SCL overhead reaches around 76%. This is explained by the fact that the SCL cycle time is substantially penalized by the completion detector performance, usually regarded as the bottleneck of QDI templates. In sub-threshold operation, all templates present similar cycle time results. Differently from the super-threshold case, the SCL template does not show performance overheads in sub-threshold operation. This suggests that the performance degradation due to the low supply voltage affects much more other templates, making all of them end up with similar cycle time results.

Table 3.2: Simulation Results of the 8-bit Kogge-Stone Adder in super-threshold and sub-threshold operation. All results of cycle time (CT), leakage power (Leak), dynamic power (Dyn), energy per Operation (EOP), transistor count (TC), Energy-Delay Product (EDP) and Leakage-Delay Product (LDP) are normalized to NCL results.

V_{dd}	Template	CT	Leak	EPO	TC	EDP	LDP
1V	NCL	1	1	1	1	1	1
	NCL+	0.96	0.84	1.01	1	0.97	0.81
	ASVHB	0.92	1.53	1.53	1.35	1.40	1.41
	SCL	1.76	0.52	1.21	0.64	2.14	0.92
0.2V	NCL	1	1	1	1	1	1
	NCL+	0.95	0.59	0.99	1	0.94	0.56
	ASVHB	0.95	1.45	1.40	1.35	1.33	1.37
	SCL	0.98	0.59	0.98	0.64	0.96	0.58

Regarding power consumption characteristics, NCL+ and SCL showed 16% and 48% less leakage power than NCL in super-threshold operation, respectively. The NCL+ results are related with the usage of the RTO protocol – as covered in [MNM+14] – and the SCL results derive from the significantly smaller transistor count. In sub-threshold operation, the same applies to both templates, except the fact that NCL+ and SCL present an increase of leakage reduction (40%). However, the same does not apply to ASVHB. As ASVHB needs the implementation of handshake logic between each logic gate, extra hardware is required for proper synchronization. This aspect translates into an area overhead of 35%, which also explains the leakage power increase of 53% and 45% for super- and sub-threshold operation, respectively.

Concerning EPO figures, NCL provides interesting results. Both NCL and NCL+ templates have similar energy consumption under both supply voltages. ASVHB on the other hand increases EPO by 53% with $V_{dd} = 1V$ and 40% with $V_{dd} = 0.2V$. Interestingly, SCL provides 21% more EPO than NCL in super-threshold operation, but provides equivalent EPO in sub-threshold operation. Also, the SCL-based case study registers the highest EDP among all templates operating with $V_{dd} = 1V$. This is justifiable by the fact that the SCL template has significant cycle time and EPO overheads, which are the main components of EDP. As these overheads are mitigated in sub-threshold operation, SCL presents and slightly lower EDP – around 4%. Unfortunately, the same is not applicable to ASVHB. The ASVHB presents the highest EDP results in both supply voltages. For a more quantitative analysis, ASVHB EDP values show an increase of 40% with $V_{dd} = 1V$ and 33% with $V_{dd} = 0.2V$ when compared to NCL. The NCL+ EDP remained similar to that observed for the NCL template, following the trend of results collected for cycle time and EPO figures. Analyzing LDP in super-threshold operation, NCL+ and SCL achieve 8% 9% better results than NCL. These results are improvements when both templates operate in the sub-threshold region. For NCL+, LDP is decreased by 44%, while SCL achieves a 42% LDP reduction. The leakage reduction provided from NCL+ and SCL implies that the combination of both templates

might achieve further leakage reduction. Because of that, a RTO-based SCL template called SCL+ is covered in Chapter 5, allowing an understanding of the possible optimizations due to the template combination. On the other hand, the higher area consumption of ASVHB implies higher LDP results, which translate to an overhead of around 40% when compared to NCL in both supply voltages. The area estimation using transistor count highlights the logic reduction achieved by the SCL template. In fact, the SCL-based case study uses 36% and 47% less transistors than its NCL/NCL+ and ASVHB counterparts, respectively.

4. Analyzing and Improving the SCL Template

This Chapter analyzes three aspects of the SCL template. First, it discusses the SCL register, shows an internal hazard is possible to occur during normal operation of the register structure, and proposes a simple modification that do not change the register behavior, eliminates the possibility of hazard and which has a smaller cost in area and power. Second, it analyzes timing issues caused by the early completion scheme, highlighting that not only the SCL template has to respect a timing constraint but creates a performance overhead. These timing issues motivated the proposal of a new version of the SCL template, which this Chapter presents as the VELO template. The VELO template proposes the employment of a late completion scheme, which avoids the timing constraint and performance overhead recently mentioned. Finally, The Author briefly discuss the requirements of a possible SDDS implementation using the SCL template. This discussion pinpoints the necessity of positive and negative unate functions and proposes a simple modification on the implementation of the SCL combinational logic, providing a low-area implementation for both functions.

4.1 The SCL Register NMOS gating

During the SCL analysis conducted in this work, an issue was detected in the SCL register implementation as proposed in [PSAA16]. Under at least one specific situation, it is possible for the internal node $\overline{Q}$ to float. Refer to Figure 3.24 and to Figure 4.1. If all inputs in Figure 3.24(a) are set (S = 1 and In = 1), neither the PUN nor the PDN of the first NOR gate (see Figure 3.24(b)) are activated. To illustrate this scenario, the waveform in Figure 4.1 demonstrates the operation of the SCL register in four modes: RESET, SET, HOLD-1 and HOLD-0. During RESET and SET modes, the output is driven by the sleep signal S and the data signal In, respectively. The HOLD-1 mode is established when Q = 1 but neither S nor In drive the output. This mode is useful as it avoids the propagation of a '0' from In as the sleep signal is the only responsible to reset Q. During HOLD-0, both data and sleep signals are set. This scenario is possible, as a data token can arrive in In before S is reset – the register is then waiting to be changed to *active mode*. Due to the NMOS power-gating in the first NOR (the transistor driven by $\overline{S}$ in Figure 3.24(b)), the internal node $\overline{Q}$ is not driven by either V_{dd} or G_{nd}. Considering that the PUN/PDN are sized to achieve balanced currents, the internal mode $\overline{Q}$ will tend to reach $V_{dd}/2$ through the P and N ON transistors driving it. This dynamic behavior is not capable to change Q and compromise the template functionality, but it will increase the register leakage current uselessly – an undesirable side effect.

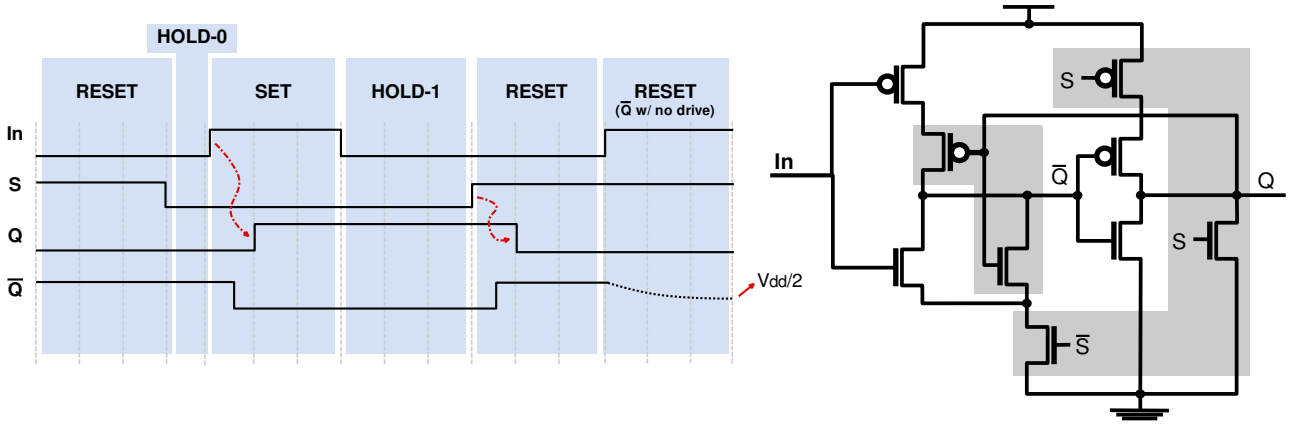


Figure 4.1: Operation of the SCL register according its input and output values. The $(V_{dd}/2)$ end value of $\overline{Q}$ considers balanced sizing of PUN and PDN of the first NOR gate.

For super-threshold operation, this side effect is expected to be negligible, because the leakage current is not able to provide enough voltage noise to disturb the output node. However, for near-/sub-threshold operation, noise margins for voltage levels are short and small voltage noise effects can set the output node to an unknown voltage value, possibly compromising the logic level of downstream logic cells. Fortunately, this side effect can be eliminated by simply removing the NMOS gating transistor, without changing the register functionality. In fact, without the NMOS gating, the register implements a conventional RS latch with two cross-coupled 2-input NORs, as Figure 4.2 illustrates. Moreover, the removal of NMOS gating avoids the need of an additional inverter responsible for producing the negated value of the input sleep signal $\overline{S}$, further reducing area consumption.

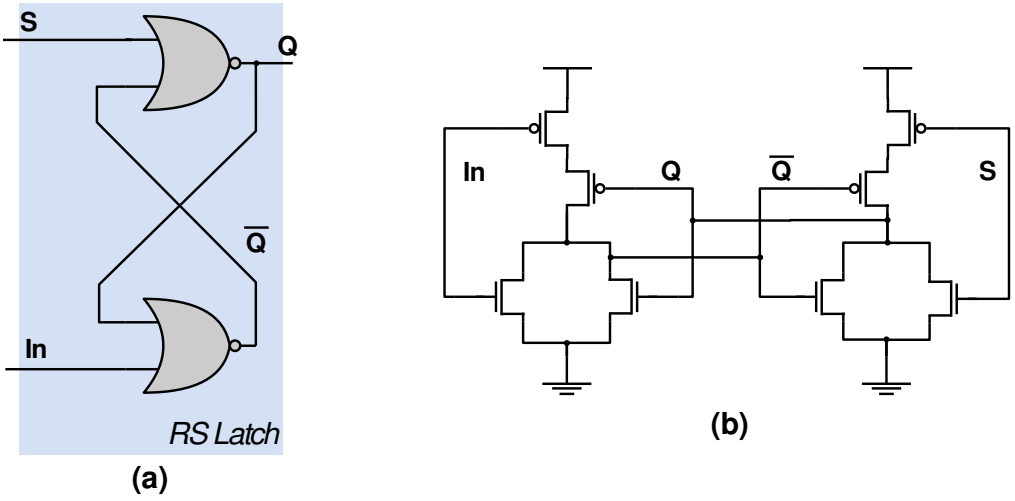


Figure 4.2: A suggestion for a new SCL register – a conventional RS Latch using two cross-coupled 2-input NORs: (a) gate-level design and (b) transistor-level schematic.

4.2 Timing Constraint Issues in the SCL Template

Parsan et al. in [PSAA16] pinpoint that SCL brings throughput optimization, as the template employs early completion to detect data and NULL tokens through the pipeline. Indeed, the adoption of early completion brings this performance advantage as it performs completion detection while data is propagated to the stage register. However, this brings additional timing constraints that must be addressed to avoid indication of acknowledgement without proper data propagation. Nonetheless, the analysis conducted in this work showed that in its current form SCL can violate these timing constraints, as this Section shows. The Section addresses a problem not covered in the original proposition of SCL, equates the associated timing constraints qualitatively and quantitatively, shows a set of experiments that demonstrate under which situations an SCL circuit can fail, and propose a solution to dimensioning SCL pipelines to respect the raised constraints. Next, Section 4.3 proposes a modification of SCL that mitigates the problems of the template and constitute a new asynchronous circuit design template.

To start with, refer to the SCL pipeline in Figure 3.21. When a data token arrives at an SCL pipeline stage input, the only logic block that processes this token is the completion detector – the rest of the pipeline stage is in *sleep mode*. Consequently, the pipeline stage has to wait for the completion detector to propagate its computation through the settable C-element, which can then drive the sleep signal and awake the rest of the pipeline stage. In other words, the SCL behavior implies that tokens have to propagate through the completion detector before propagating to the pipeline stage itself, bringing a forward latency overhead. In fact, this overhead was detected and already discussed in Section 3.9. The SCL use of early completion brings new timing constraints that must be addressed to keep the implementation a QDI circuit. The idea to acknowledge the previous pipeline stage before fully storing data token in another stage is what can compromise the template functionality. This can happen if the previous pipeline stage resets before the register of the current pipeline stage stores a valid data token.

The problem can be visualized in Figure 4.3, which focuses on the first two stages of the SCL pipeline in Figure 3.21. Assume that Stage 1 receives valid data at its input and CD_1 detects it. Consequently, the inverted C-element C_1 will eventually lower its output. At this point, two concurrent sequences of events take place, as the output of C_1 forks to two paths:

1. Stage 1 sleep signal is disabled, awaking pipeline Stage 1 to store and process its input data, from the output of Stage 0;

2. An acknowledge signal is sent to the previous pipeline stage (Stage 0), forcing it to sleep and reset all its logic blocks to NULL (i.e. to spacers).

If Stage 1 awakes before the reset process of Stage 0 takes place, register R_1 should sample and store valid data, and the pipeline operates correctly. However, if Stage 0 is able to sleep and reset all its logic before Stage 1 wakes up, register R_1 may not be able to correctly store the new data. This possible functionality hazard implies that SCL implementations must respect an timing constraint, and this was not covered in the template original proposition. Another consequence of this hazard is that the race between signals affecting the input of block R1 in Figure 4.3 may cause the output of this block to become metastable, which can have catastrophic consequences for downstream and upstream logic.

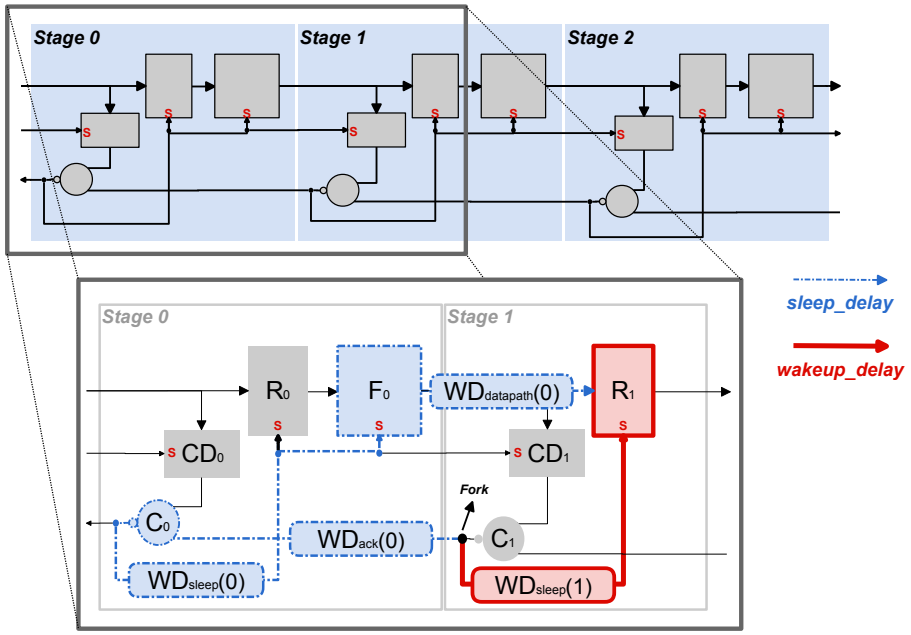


Figure 4.3: A scheme for delay analysis of the SCL isochronic fork between two pipeline stages.

From this analysis, it is possible to observe that there is a timing constraint between the awaking of registers and the sleeping time of some logic blocks. In fact, recalling Section 2.1, this constraint relies on the isochronicity of the fork in the output of the settable inverted C-elements. Note that both branches include not only wire delay but gate delays, which make this fork incompatible with the original isochronic fork definition. Fortunately, the extended definitions proposed in [MM95] and [vBHP95] allow a more flexible analysis on isochronic forks, making possible to assume that the circuit can maintain its QDI properties whether the extended definition of the isochronic fork is respected. To more thoroughly understand this timing constraint, Figure 4.3 shows the two concurrent paths between Stage 0 and Stage 1 with a thick continuous red line and a dashed blue line. The red line represents the delay path to *wake up* Stage 1 and incorporates three delay elements: a wire delay $WD_{sleep}(1)$, the rise propagation delay of register $R_1 \uparrow$ and the register hold time R_{n_hold}. Hence, it is possible to indicate that the wake up delay of a Stage n is represented by:

$$wakeup\_delay(n) = WD_{sleep}(n) + R_n \uparrow + R_{n\_hold} \tag{4.1}$$

On the other hand, the dashed blue line represents the delay path to *put* Stage 0 *to sleep* and incorporates four delay elements: (i) the wire delays $WD_{ack}(0)$; (ii) $WD_{sleep}(0)$; (iii) the propagation delay of the inverted C-element C_0; (iv) the fall delay of the combinational logic block $F_0\downarrow$; and (v) the wire delay $WD_{datapath}(n)$. Note that, in this case, the propagation delay of R_0 is not considered. As both register and combinational logic are reset during the sleep process, the combinational logic will reset its output independently of the delay of the register. In that way, the sleep delay of a Stage n can be represented by:

$$sleep\_delay(n) = WD_{ack}(n) + C_n + WD_{sleep}(n) + F_n\downarrow + WD_{datapath}(n) \tag{4.2}$$

Finally, the timing constraint between pipeline stages can be computed by:

$$wakeup\_delay(n) \leq sleep\_delay(n-1) \tag{4.3}$$

The Equation states that the *wake up delay* of pipeline stage n must be equal or lower than the *sleep delay* of the previous pipeline stage $(n-1)$. In this way, combining Equation 4.1 and 4.2, it is possible to determine which delay element of a delay path should be manipulated to satisfy the timing constraint in Equation 4.3. This combination can be more easily observed in Equation 4.4.

$$(WD_{sleep}(n) + R_n\uparrow + R_{n\_hold}) \leq (WD_{ack}(n-1) + C_{n-1} + WD_{sleep}(n-1) + F_{n-1}\downarrow + WD_{datapath}(n-1)) \tag{4.4}$$

This Equation is the same as Equation 4.3 but considers all delay elements values of the fork.

In case the circuit does not respect the isochronic fork, it is necessary to manipulate the delay elements of the fork, increasing (or decreasing) one or more delays. This could be done by loading timing constraints into an EDA tool, to add (or optimize) delay elements of the fork. However, it is important to understand the trade-offs that could arise as more (fewer) delay elements are added to the fork paths. Hence, each delay element is analyzed individually to point the negative and positive aspects of increasing (decreasing) its value. We analyze two alternatives to satisfy the timing constraint proposed in Equation 4.3: (i) increasing delay values of the $sleep\_delay(n-1)$; and (ii) decreasing delay values of the $wakeup\_delay(n)$.

Focusing on $sleep\_delay(n-1)$, consider increasing the wire delay $WD_{ack}(n-1)$, which connects the C-elements. This can be a valid option at the cost of a higher ACK propagation delay. This same trade-off happens whether the delay propagation of the previous stage C-element (C_{n-1}) is increased. Another option is to increase the sleep delay of the combinational block F_{n-1} to reset its output. F_{n-1} is out of the ACK path and will not affect the ACK propagation delay. However, this implies that the sleep logic in the SCL gates should be resized and, consequently, will affect the datapath delay. Note that the same effect happens

whether $WD_{datapath}$ is increased. As a last option, the $WD_{sleep}(n-1)$ could be increased. In this case, neither the ACK propagation delay nor the datapath delay would be affected. Unfortunately, $WD_{sleep}(n)$ composes the ($wakeup\_delay(n-1)$) and its increase may also affect the isochronic forks of previous stages. For instance, if $WD_{sleep}(n-1)$ is increased to satisfy $wakeup\_delay(n) \leq sleep\_delay(n-1)$, it also increases $wakeup\_delay(n-1)$. This increase may affect the previous isocrohonic fork $wakeup\_delay(n-1) \leq sleep\_delay(n-2)$, requiring more delay manipulations in previous isochronic forks and causing a ripple performance degradation effect on previous stages. All these points suggest that manipulating $sleep\_delay(n-1)$ brings performance trade-offs, specially when interdependent delay elements between fork paths are manipulated.

The second alternative is to decrease the delay value of $wakeup\_delay(n)$. In this case, there are only two options: decrease $WD_{sleep}(n)$ or/and R_n. As before, manipulating $WD_{sleep}(n)$ may be a complex option, due to its interdependence with the fork paths. Decreasing $WD_{sleep}(n)$ will not only decrease $wakeup\_delay(n)$ but also decrease $sleep\_delay(n)$, wherein the latter must be a value higher than $wakeup\_delay(n+1)$. This manipulation forces the next stage isochronic forks to have lower delay values and may not be achievable if the next fork paths cannot be optimized. Register R_n could be optimized whether multiple drive strengths are available in order to decrease the register delay.

Other possible problems while operating with the SCL isochronic fork are the issues caused by current variations, which imply that only guaranteeing that $wakeup\_delay(n) \approx sleep\_delay(n-1)$ may not be enough. The delays inside the circuit change arbitrarily due to current variations and the isochronic fork may not respect the timing constraint defined by Equation 4.3 if delay variations are too large. This implies that it is necessary to consider a margin between delay paths to guarantee the functionality of the circuit, even with a certain amount of delay variations on its logic and wires. In order to support current variations, Equation 4.5 represents the same constraint as Equation 4.3, but it adds a margin Var_{margin} to the $wakeup\_delay(n)$ branch.

$$wakeup\_delay(n) + Var_{margin} \leq sleep\_delay(n-1) \tag{4.5}$$

In super-threshold operation, current variations can be represented by a Gaussian distribution $N(\mu, \sigma)$, where μ represents the mean value of the distribution and σ is the standard deviation. Frequently the confidence interval is defined by a $n \times \sigma$ distance from μ, which can define Var_{margin} in Equation 4.5. According to [GAN12], using $\mu \pm 3\sigma$ as end points defines a confidence internal that covers all samples with probability of 99.7%. For sub-threshold operation, however, the relation of end points with the standard deviation is not straightforward. As sub-threshold currents have an exponential dependency on the threshold voltage, the sub-threshold current variations translate to a log-normal distribution [LAHG12]. In that way, it would be inaccurate the usage of a confidence interval such as $\mu \pm 3\sigma$ to

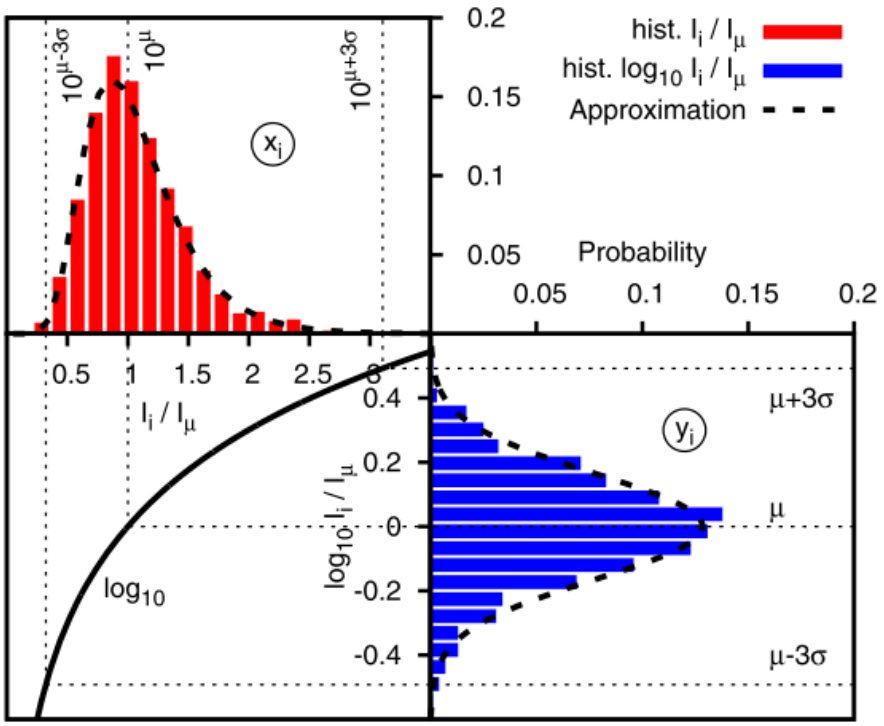


Figure 4.4: Example of correlation of $\mu \pm 3\sigma$ confidence interval for sub-threshold currents assuming a Gaussian distribution of transformed current samples. Extracted from [GAN12].

determine the end points. Fortunately, it is possible to overcome this issue by transforming a log-normal distribution to a Gaussian distribution, allowing correlation between $n \times \sigma$ with a log-normal confidence interval. Considering a 3σ range, the following approach can be adopted [GAN12]:

1. Any random log-normal distribution X with samples x_i is transformed to a random Gaussian distribution $Y(\mu, \sigma)$ with samples y_i by applying $y_i = log_{10}(x_i)$;
2. The confidence interval of $Y(\mu, \sigma)$ with probability of 99.73% is $[\mu - 3\sigma, \mu + 3\sigma]$;
3. Mapping the end points of $Y(\mu, \sigma)$ to X, Equation 4.6 shows the corresponding confidence interval of X.

$$X_{\mu \pm 3\sigma} = 10^{\mu \pm 3\sigma}, \quad where \quad \mu = \frac{\sum log_{10}(x_i)}{n}; \quad \sigma^2 = \frac{\sum (log_{10}(x_i) - \mu)^2}{n-1} \tag{4.6}$$

Figure 4.4 illustrates the correlation between a log-normal distribution with its corresponding transformed Gaussian distribution for MOS transistor currents. Note that current variations directly impact the performance of gates. Thus, if currents vary according to a log-normal probability distribution, the performance of gates will vary in the same way.

4.2.1 Experiments

The following experiments examine timing constraint discussed in this section. Their objective is to test the feasibility and overheads while implementing SCL circuits. The experiments present a set of assumptions to correlate logic and wire delays. Also, they consider operation parameters such as technology nodes and supply voltage. Moreover, the experiments use Monte Carlo simulation to take into account within-die variations, i.e. process corner mismatch.

In this analysis, two regular SCL pipeline stages are considered. They are sequentially connected as highlighted in Figure 4.3. As the fork comprises several delay components, some assumptions were considered to simplify the analysis. First, all existing wire delays in the fork are assumed equal ($WD_{sleep}(n) = WD_{ack}(n-1) = WD_{sleep}(n-1) = WD_{datapath}(n-1)$). Second, all delays in the fork are normalized to the delay of the SCL sleep logic ($WD_{norm} = WD/F_{n-1}\downarrow$, $R_{norm} = R_n\uparrow/F_{n-1}\downarrow$, $C_{norm} = C_{n-1}/F_{n-1}\downarrow$, $F_{norm} = F_{n-1}\downarrow/F_{n-1}\downarrow = 1$), which is always implemented as a NOR gate in the output of all SCL logic – see Figure 3.24 and Figure 3.22. This normalization allows to relate all fork delays, including gates and wires. The analysis considers three different technology nodes: 180nm bulk CMOS, 65nm bulk CMOS and 28nm FDSOI-CMOS. $F_{n-1}\downarrow$, $R_n\uparrow$ and C_{n-1} are implemented in three different technology nodes. Nominal and sub-threshold operation (30% of nominal supply voltage) are considered, as well as delay variations due to process corner mismatch. Considering nominal and sub-threshold operation allows to determine the delay relations among logic gates (R_{norm} and C_{norm}) and the delay variation Var_{margin} in both supply voltages.

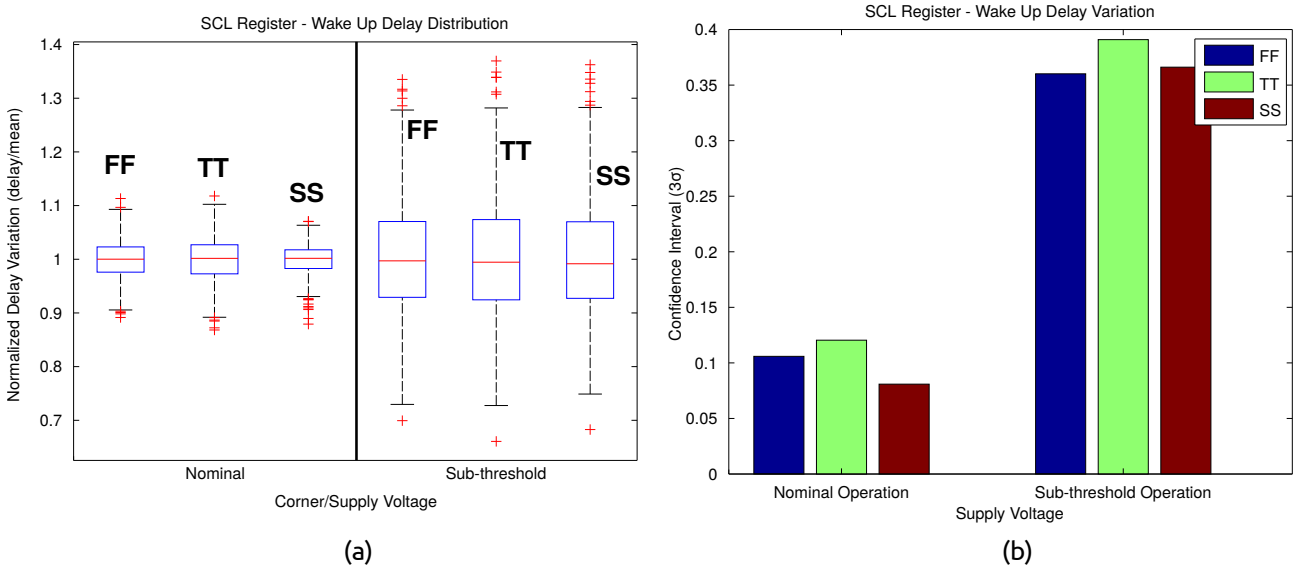


Figure 4.5: Results for Monte Carlo simulations for delay variation analysis: (a) represents the propagation delay variation of a SCL register. All distributions are normalized to the mean, three process corners are considered (FF, TT, SS) as well as two supply voltages (nominal and sub-threshold). (b) indicates the interval distance 3σ of the distributions in (a).

Figure 4.5(a) illustrates the propagation delay variation of the SCL register implemented when targeting the 65nm bulk CMOS technology. Results were obtained from 1,000 Monte Carlo simulations using three different process corners (SS, TT and FF) and two supply voltages (1V and 0.3V). As expected, the SCL register presents a higher delay variation in sub-threshold than under nominal operation and corners with the same supply voltage display small differences among them. These comparisons can be seen in Figure 4.5(b), which shows the confidence distance 3σ of the distributions in Figure 4.5(a). While the three process corners in nominal operation present a distance around 10% of the mean value, the distance in sub-threshold reaches around 35% and 40%. This higher delay variation implies that if the isochronic fork is targeted to sub-threshold operation, higher Var_{margin} must be considered to the isochronic fork.

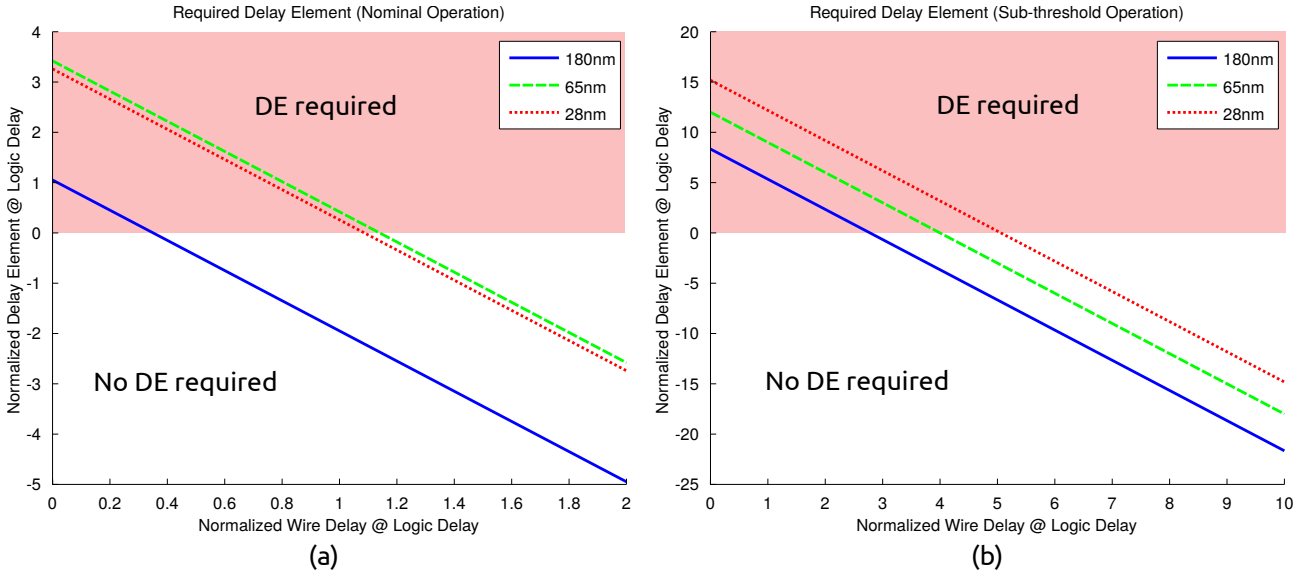


Figure 4.6: Required Delay Element (DE) for three technology nodes: 180nm bulk CMOS, 65nm bulk CMOS and 28nm FDSOI CMOS. Axis X represents the delay value of all wires in the fork (WD_{sleep}, WD_{ack}, WD_{sleep}, $WD_{datapath}$) and axis Y indicates the required DE. (a) shows results in nominal operation and (b) in sub-threshold operation. Note that both axes are normalized to the SCL sleep logic delay.

Having established the delay variation parameters of each target technology and the relationship among all logic delays, Figure 4.6(a) and (b) indicate when a Delay Element (DE) is required in nominal and sub-threshold operation, respectively, depending on the wire delay contribution in the isochronic fork. Note that both axes are normalized to the SCL sleep logic delay. The "DE required" region covers positive values of the delay element, while "No DE required" region covers negative values. If the DE value is negative, the fork is guaranteed to be isochronic. Otherwise, a DE must be inserted according to the normalized DE value. For 180nm, the fork will easily fulfill the isochronic constraint under nominal supply. If the wires have a delay which is 40% the delay of the SCL sleep logic or more, no DE is required. Now, a lesser wire delay contribution ($\leq$ 40%) in the fork requires the addition of a DE. Fortunately, the overheads are low, as the normalized DE value only reaches 1. In other

words, adding one or a couple of buffers in the fork would guarantee the fork is isochronic. Regarding 65nm and FDSOI 28nm technology nodes under nominal operation, the required margin increases slightly. In these cases, no DE is required if wires have the same or higher delay than the SCL sleep logic. This type of scenario is feasible, since the delay of wires has historically not decreased as much as the delay of gates in recent technology nodes. In fact, [HE00] points out that wire delays can exceed gate delays in several critical paths of benchmark circuits and this trend increases with circuit size. However, if the fork is physically composed of small wires only, a DE is required and its implementation may comprise multiple buffers.

As the supply voltage reaches the sub-threshold region, Figure 4.6(b) suggests that significantly higher margins are required to guarantee the fork isochronicity. Considering the 180nm technology node, for instance, the isochronic fork requires the insertion of a DE if the wire delays are 3× or less smaller than the SCL sleep logic delay. For 65nm and 28nm, wire delays must be around 4× and 5× smaller, respectively. This pinpoints that the SCL template will hardly respect the isochronic fork constraint without the insertion of DEs in sub-threshold operation. In addition, the implementation of a DE in sub-threshold operation can comprise a high number of buffers, increasing area overhead.

4.3 Section Suppressed for the Purpose of Intellectual Property Protection

4.4 Section Suppressed for the Purpose of Intellectual Property Protection

5. In-Depth Evaluation of QDI Templates for Voltage Scaling

Chapter 3 of this Dissertation investigated several asynchronous QDI templates, with target on their use for operation under voltage scaling. A subset of these templates was selected and evaluated, qualitatively and quantitatively. Results showed that the SCL template [PSAA16] is one of the most adequate compromises to develop the target applications.

(The rest of this Section was Suppressed for the Purpose of Intellectual Property Protection.)

5.1 The ASCEnD Cell Design Flow

The ASCEnD Design Flow [MNM+14] is an automated environment specialized to generate asynchronous standard cell libraries. The flow comprises commercial and in-house tools. The latter were developed in recent years. ASCEnD has already been used to develop libraries containing C-elements, mutexes, NCL/NCL+ gates, PCHB cells and is under constant update by the GAPH group. Figure 5.1 is an overview of the ASCEnD Design Flow. Currently, this flow includes four main steps: Cell Library Template (Definition), Cell Sizing, Cell Layout and Cell Characterization. These steps are connected through in-house scripts that provide data conversion, adaptation and controls to facilitate the tasks taken by the designer throughout the entire flow[1].

The first flow step helps in the specification of a Cell Library Template. This template provides information such as the target standard cell format (cell height, position of supply rails, etc.) and library/technology parameters. Library/technology parameters are defined by an initial configuration file, which contains basic information such as technology files, target supply voltages, maximum NMOS/PMOS widths, available driving strengths, etc. In-house scripts retrieve information from this configuration file to generate input files for each design flow step, allowing fast transition between these. The target standard cells are individually specified through template files using the SPICE language added with variables and macros. These variables and macros allow the designer to define sizing, layout and characterization information that will be used by further steps in ASCEnD. As an example, Figure 5.2(a) shows the cell template of an NCL TH22 gate – Figure 5.2(b) illustrates the equivalent schematic of the example cell template. Note that this gate implements the functional behavior of a 2-input C-element. The circuit is described as a traditional SPICE SUBCKT with additional comment lines and variables for specific transistor parameters. In ASCEnD, these variables are responsible for indicating which transistors will be dimensioned

[1] The work developed in this Chapter took advantage of the two first flow steps (Cell Library Template and Cell Sizing), which are detailed here. For more information regarding these and the remaining flow steps, the reader should refer to [Mor16].

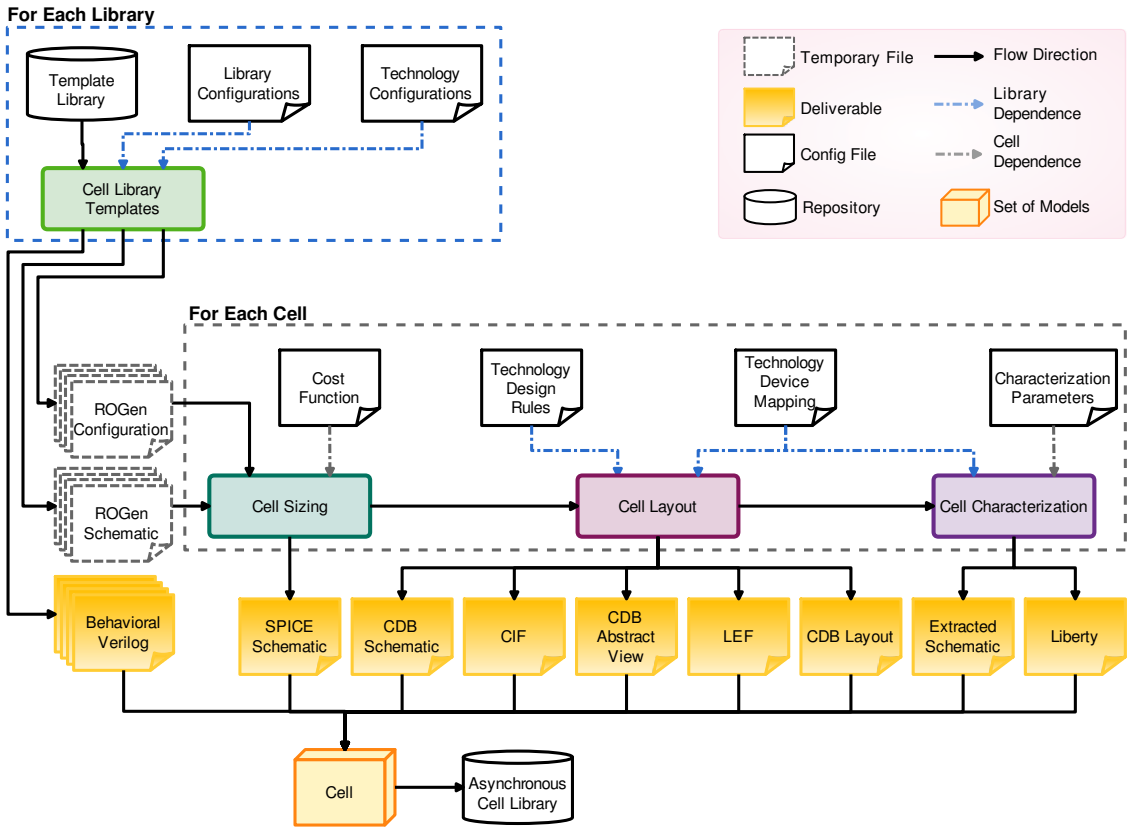


Figure 5.1: Overview of the ASCEnD Design Flow.

during the sizing step and which transistors must employ a special sizing strategy (e.g. for transistors on feedback lines). Currently, cell templates allow variables on transistor width, finger and multiplier – transistor length is fixed and typically set to the minimum length value allowed by the underlying technology. After sizing, all variables are replaced by fixed values. The additional comment lines (see lower left box in Figure 5.2) define the ASCEnD macros that are used through the entire design flow by both in-house tools and scripts – refer to Table 5.1 for a list of the available ASCEnD macros. Basically, ASCEnD macros indicate cell information that cannot be explicitly defined in or inferred from the SPICE description. For instance, considering the NCL TH22 gate in Figure 5.2, it is not trivial to identify the gate threshold value by parsing the cell description. On the other hand, the threshold value of an NCL gate can be easily defined by .ROGen:THR <integer>. Having specified all cell templates and library/technology parameters, an in-house script creates input files for the next flow step: Cell Sizing.

The Cell Sizing step of ASCEnD employs two in-house tools (ROGen and CeS) and requires the use of an electrical simulator[2]. ROGen uses the input files provided in the previous step to generate a simulation environment, which extracts power and performance information for all target cells. This simulation environment uses the configurations provided by the user to vary transistor sizes and collect performance figures during sim-

[2]Currently, Cell Sizing only supports the Cadence SPECTRE Simulator.

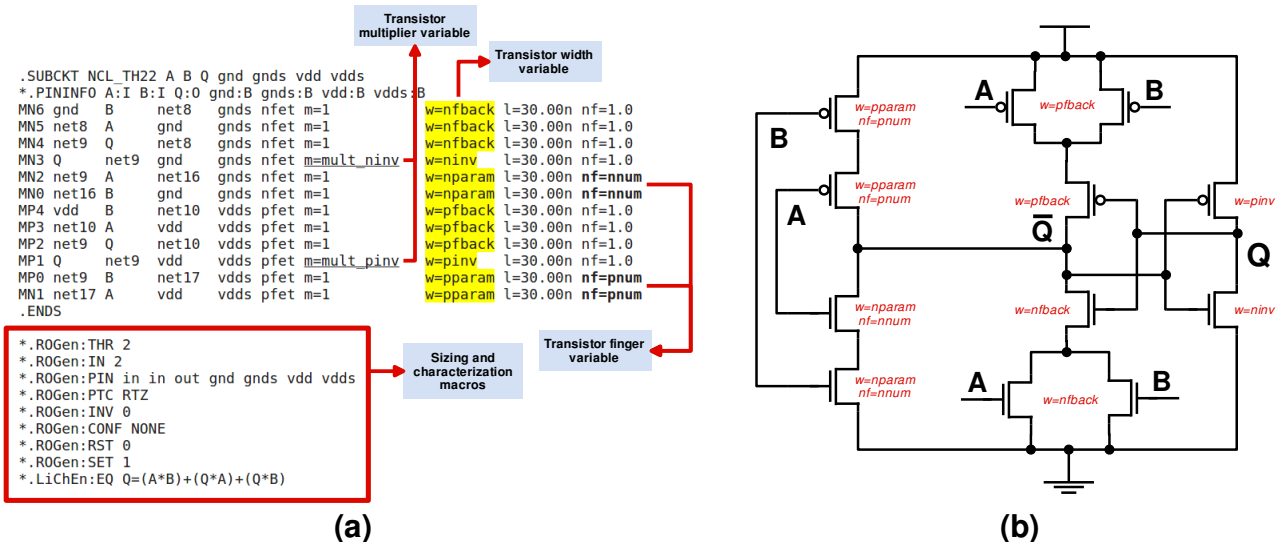


Figure 5.2: Cell template for an NCL TH22 gate (a 2-input C-element): (a) SPICE description and (b) equivalent schematic. ROGen stands for Ring Oscillator Generator, one of the ASCEnD sizing tools. LiChEn stands for Library Characterization Environment, the flow characterization tool [Mor16].

ulation. 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 of which are specified in the ASCEnD library/technology configuration file. For these variations, the tool combines all NMOS and PMOS dimensions available. Having the performance and power information of all sizing possibilities, CeS (the Cell Specifier tool) can read the output file provided by the ROGen simulation and define the best transistor sizing according to the provided cost function. The designer has a wide range of parameters to define cost functions, such as propagation delays (*prop_rise*/*prop_fall*), transition delays (*tran_rise*/*tran_fall*), dynamic power (*dynpwr*) and leakage power (*leakpwr*). ASCEnD has available some predefined cost functions. Currently, three such cost functions are available: high performance (Equation 5.1), energy efficiency (Equation 5.2) and low power (Equation 5.3).

$$EQ_{HP} = \frac{1}{prop\_rise + prop\_fall} \tag{5.1}$$

$$EQ_{EFF} = \frac{1}{(prop\_rise + prop\_fall) \times dynpwr} \tag{5.2}$$

$$EQ_{LP} = \frac{1}{dynpwr} \tag{5.3}$$

(The rest of this Section was Suppressed for the Purpose of Intellectual Property Protection.)

Table 5.1: ASCEnD macros for cell template specification.

Macro	Description	Usage
.ROGen:THR	Threshold value of a threshold logic. This macro is required by C-elements and NCL gates.	.ROGen:THR <integer>
.ROGen:IN	Number of Inputs.	.ROGen:IN <integer>
.ROGen:PIN	Pin direction definition (input, output or supply). Follows the same sequence as defined in the SUBCKT. Available types: in, out, gnd, gnds, vdd, vdds	.ROGen:PIN <set_of_pins>
.ROGen:PTC	Handshake protocol (only four-phase).	.ROGen:PTC [RTZ|RTO]
.ROGen:INV	Output Unatess. Positive = 0 Negative = 1	.ROGen:INV [0|1]
.ROGen:CONF	Additional logic (reset and/or set).	.ROGen:CONF [NONE|RST|SET]
.ROGen:RST	Reset activation. Active-low = 0 High-low = 1	.ROGen:RST [0|1]
.ROGen:SET	Set activation. Active-low = 0 High-low = 1	.ROGen:SET [0|1]
.LiChEn:EQ	Output equation.	.LiChEn:EQ < equation > Example: .LiChEn:EQ Q=(A*B)+(Q*A)+(Q*B)

Table 5.2: Suppressed for the Purpose of Intellectual Property Protection.

Thres. Logic	SCL	SCL+	XXXX	YYYY
TH22	X	X	X	X

5.2 Section Suppressed for the Purpose of Intellectual Property Protection

5.3 Section Suppressed for the Purpose of Intellectual Property Protection

5.4 Section Suppressed for the Purpose of Intellectual Property Protection

6. Contributions, Conclusions and Further Work

This Dissertation contributed in several topics to enhance the research of asynchronous circuit design, with specific focus on voltage scaling operation. A first contribution covers comparing bulk and FDSOI CMOS technologies on voltage scaling scenarios, identifying the main trade-offs of each technology type. Using 65nm Bulk CMOS and 28nm FDSOI technologies, the evaluation comprises a set of basic CMOS logic gates (NAND and NOR) to retrieve performance and power information. Results indicate that traditional bulk CMOS technologies are still a feasible option for voltage scaling, albeit FDSOI technologies can bring large energy efficiency benefits in low-voltage scenarios. In addition, a brief discussion reinforces the common knowledge that FDSOI technologies are more commercially appropriate for ultra-low power applications than FinFET technologies. Its lower fabrication cost and characteristics more similar to traditional technologies encourage foundries to focus on FDSOI for lower power and/or lower cost applications, making it a suitable option for IoT and similar markets.

This Dissertation also contributes to the asynchronous research, by reviewing several QDI templates proposed in the literature, analyzing their compatibility to voltage scaling scenarios and low and ultra-low power applications. To understand the benefits of each template, the QDI analysis advanced here covers implementation aspects, such as basic logic blocks, pipeline and handshaking structure, and operation behavior. The template behavior is addressed with examples of the propagation of data/NULL tokens through a linear pipeline, and also with marked graphs, pinpointing which transition set composes the template forward and backward latencies. Combining all aspects covered in this analysis gives a solid basis to discuss how each QDI template can effectively benefit from voltage scaling usage and indicate possible implementation problems. This qualitative discussion encouraged the Author to compare a set of QDI templates (NCL, NCL+, ASVHB and SCL) using a case study circuit, leading to elaborate comparisons among templates. At the end of this analysis, the Author could provide a set of basic conclusions: (i) the ASVHB provides throughput improvement due to its fine-grained pipeline structure, albeit its implementation brings power overheads; (ii) both NCL+ and SCL provide better leakage power reduction, which is interesting for ultra-low power applications; (iii) as NCL+ employs a different four-phase protocol – the RTO protocol – it would be appropriate to combine the same protocol with SCL, generating a RTO-based variation of the SCL template: the SCL+ template; (iv) the SCL template is indicated as a suitable alternative to implement low and ultra-low power circuits, due to the power and area consumption reductions it enables. Nonetheless, the in-depth analysis of SCL also indicates that this template has performance and implementation issues, suggesting that the SCL template still needs improvement to enable its use for voltage scaling applications.

The next contribution focuses on the proposal of three main modifications to the SCL template. The first one covers a modification to the original SCL register, avoiding the use of a NMOS gating. Second, the SCL completion detection scheme analysis reveals a major weakness, highlighting that use of early completion negatively affects the template performance and worse than this, implies a timing constraint in the acknowledgement/sleeping structure, which can potentially lead to violations of the QDI main principle. Due to this timing constraint and its complexity, the Author reviewed the concept of isochronic fork, which is stated as the only class of timing restriction imposed in QDI designs. The review brought about a more flexible definition of isochronic forks, making the SCL timing constraint compatible with the extended isochronic fork definition. Having this compatibility, the SCL template can keep its status as a QDI template, considering that the design respects the imposed timing constraint. Still regarding this same topic, the Dissertation analyzed the SCL timing constraint along three technology nodes, to determine the relative criticality of the SCL timing constraint in current commercial nodes. Estimating logic and wire delays, the experiments suggests that, under nominal supply operation, the SCL timing constraint is easily solved and avoids extra overheads. In sub-threshold operation, however, PVT variations have more impact on the circuit, implying that a larger margin is required to guarantee that the design respects the SCL timing constraint.

(The rest of this Section was Suppressed for the Purpose of Intellectual Property Protection.)

6.1 Future Work

(Section Modified for the Purpose of Intellectual Property Protection)

In recent years, the GAPH group gained ASIC prototyping experience. This fact motivated the Author to specify and develop an asynchronous QDI standard cell library for the TSMC 180nm bulk CMOS technology to support the design of circuits based asynchronous templates. The library is currently under development. As a long term objective, this library provides prototyping support to the group evaluate multiple QDI templates on silicon. This action has the potential to attract graduate and undergraduate students to design asynchronous circuits, from specification to tape-out.

Regarding the ASCEnD Design Flow, the sizing step can be improved and extended. Currently, the sizing process is executed by ROGen + SPECTRE tools, which exhaustively test all possible sizing options according to the maximum/minimum transistor width and sizing steps imposed by the designer. During the development of this work, it was detected that having a large number of simulation iterations with SPECTRE can cause a significant execution time increase. This has consequently forced the use of larger sizing steps and closer maximum/minimum values, reducing the precision of the ASCEnD flow

sizing step. Considering that the W_p/W_n ratio between PMOS and NMOS transistors are higher for sub-threshold operation [BLR10], it is necessary to optimize the sizing step, allowing the obtainment of more accurate results. Because of that, the Author considers two main approaches that can lead to improvement of the ASCEnD Design Flow:

- Flexible sizing step: allows the sizing process to cover a wide range of options with a larger sizing step. As the sizing process reaches near the optimum results, the sizing step is decreased, increasing the precision of the final sizing result;
- Sub-threshold sizing by methods like the one proposed by Liu and Ashouei [LAHG12]: this implies in changing the sizing process by an analytical and process-aware sizing strategy with target on sub-threshold operation.

It is important to highlight that parts of the work developed in this Dissertation were already published in the 8th IEEE Latin American Symposium on Circuits and Systems (LASCAS'17). Due to this paper quality, its Authors were invited to submit an extended version of this for publication in the IEEE Transactions on Circuit and Systems I (TCAS-I). Due to an ongoing patent filing process, no further text production has been performed so far.

References

[BDN05] Blaauw, D.; Devgan, A.; Najm, F. “Leakage Power: Trends, Analysis and Avoidance”. In: Asia and South Pacific Design Automation Conference (ASP-DAC), 2005, pp. T–2, Embedded Tutorial T-2.

[BLR10] Blesken, M.; Lütkemeier, S.; Rückert, U. “Multiobjective Optimization for Transistor Sizing of Sub-threshold CMOS Logic Standard Cells”. In: 2010 IEEE International Symposium on Circuits and Systems (ISCAS), 2010, pp. 1480–1483.

[BOF10] Beerel, P. A.; Ozdag, R. O.; Ferretti, M. “A Designer’s Guide to Asynchronous VLSI”. Cambridge University Press, 2010, 339p.

[Bre07] Brej, C. “High Performance Asynchronous Circuit Design Method and Application”. In: UK Async Forum, 2007, pp. 5p.

[BZF+08] Bailey, A.; Zahrani, A. A.; Fu, G.; Di, J.; Smith, S. “Multi-Threshold Asynchronous Circuit Design for Ultra-Low Power”, *Journal of Low Power Electronics*, vol. 4–3, Dec 2008, pp. 337–348.

[CHL+17] Chong, K. S.; Ho, W. G.; Lin, T.; Gwee, B. H.; Chang, J. S. “Sense Amplifier Half-Buffer (SAHB): A Low-Power High-Performance Asynchronous Logic QDI Cell Template”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25–2, Feb 2017, pp. 402–415.

[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.

[CWC05] Calhoun, B.; Wang, A.; Chandrakasan, A. “Modeling and Sizing for Minimum Energy Operation in Subthreshold Circuits”, *IEEE Journal of Solid-State Circuits*, vol. 40–9, Sep 2005, pp. 1778–1786.

[CYP17] Chang, M. C.; Yang, P. H.; Pan, Z. G. “Register-Less NULL Convention Logic”, *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. Early Access since April 2016, 2017, pp. 1–5.

[FB96] Fant, K. M.; Brandt, S. A. “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), 1996, pp. 261–273.

[GAN12] Gemmeke, T.; Ashouei, M.; Noll, T. G. “Noise Margin Based Library Optimization Considering Variability in Sub-threshold”. In: International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS), 2012, pp. 72–82.

[GBMP13] Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. “Internet of Things (IoT): A vision, architectural elements, and future directions”, *Future Generation Computer Systems*, vol. 29–7, Sep 2013, pp. 1645–1660.

[GMCM15] Guazzelli, R. A.; Moraes, F. G.; Calazans, N. L. V.; Moreira, M. T. “SDDS-NCL Design: Analysis of Supply Voltage Scaling”. In: Symposium on Integrated Circuits and Systems Design (SBCCI), 2015, pp. 2:1–2:7.

[HCGC15] Ho, W.-G.; Chong, K.-S.; Gwee, B.-H.; Chang, J. S. “Low Power Sub-threshold Asynchronous Quasi-delay-insensitive 32-bit Arithmetic and Logic Unit based on Autonomous Signal-validity Half-buffer”, *Circuits, Devices Systems, IET*, vol. 9–4, 2015, pp. 309–318.

[HE00] Huang, Z.; Ercegovac, M. D. “Effect of Wire Delay on the Design of Prefix Adders in Deep-Submicron Technology”. In: Signals, Systems and Computers, 2000. Conference Record of the Thirty-Fourth Asilomar Conference on, 2000, pp. 1713–1717 vol.2.

[Hen11] Henry, M. B. “Emerging Power-Gating Techniques for Low Power Digital Circuits”, Ph.D. Thesis, Virginia Polytechnic Institute and State University, Blacksburg, 2011, 104p.

[HLN+16] Ho, W. G.; Liu, N.; Ne, K. Z. L.; Chong, K. S.; Gwee, B. H.; Chang, J. S. “High Performance Low Overhead Template-based Cell-Interleave Pipeline (TCIP) for Asynchronous-logic QDI Circuits”. In: 2016 IEEE International Symposium on Circuits and Systems (ISCAS), 2016, pp. 1762–1765.

[ILL+13] Ieong, C.-I.; Li, M.; Law, M.-K.; Mak, P.-I.; Vai, M.-I.; Mak, P.-U.; Wan, F.; Martins, R. “Standard Cell Library Design with Voltage Scaling and Transistor Sizing for Ultra-low-power Biomedical Applications”. In: IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC), 2013, pp. 1–2.

[ITR11] ITRS. “2011 Edition - Design”, Technical Report, International Technology Roadmap for Semiconductors, 2011, 48p, available at https://www.dropbox.com/sh/r51qrus06k6ehrc/AABOCwgRzrxrgRpnWsvoVqoCa/2011Chapters/2011Design.pdf?dl=0.

[JSL+10] Jorgenson, R. D.; Sorensen, L.; Leet, D.; Hagedorn, M. S.; Lamb, D. R.; Friddell, T. H.; Snapp, W. P. “Ultralow-Power Operation in Subthreshold Regimes Applying Clockless Logic”, *Proceedings of the IEEE*, vol. 98–2, Feb 2010, pp. 299–314.

[LAHG12] Liu, B.; Ashouei, M.; Huisken, J.; Gyvez, J. P. d. “Standard Cell Sizing for Subthreshold Operation”. In: ACM/IEEE Design Automation Conference (DAC), 2012, pp. 962–967.

[LCGC09] Lin, T.; Chong, K.-S.; Gwee, B.-H.; Chang, J. S. “Fine-Grained Power Gating for Leakage and Short-Circuit Power Reduction by using Asynchronous-Logic”. In: IEEE International Symposium on Circuits and Systems (ISCAS), 2009, pp. 3162–3165.

[LM12] Lotze, N.; Manoli, Y. “A 62 mV 0.13µm CMOS Standard-Cell-Based Design Technique Using Schmitt-Trigger Logic”, *IEEE Journal of Solid-State Circuits*, vol. 47–1, Jan 2012, pp. 47–60.

[LSK+15] Li, P.; Shin, J.; Konstadinidis, G.; Schumacher, F.; Krishnaswamy, V.; Cho, H.; Dash, S.; Masleid, R.; Zheng, C.; Lin, Y.; Loewenstein, P.; Park, H.; Srinivasan, V.; Huang, D.; Hwang, C.; Hsu, W.; McAllister, C. “A 20nm 32-Core 64MB L3 Cache SPARC M7 Processor”. In: IEEE International Solid-State Circuits Conference (ISSCC), 2015, pp. 72–73.

[LXW+15] Li, J.; Xie, Q.; Wang, Y.; Nazarian, S.; Pedram, M. “Leakage Power Reduction for Deeply-Scaled FinFET Circuits Operating in Multiple Voltage Regimes Using Fine-Grained Gate-Length Biasing Technique”. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2015, pp. 1579–1582.

[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: IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2014, pp. 93–100.

[MAMC15] Moreira, M. T.; Arendt, M. E.; Moraes, F. G.; Calazans, N. L. V. “Static Differential NCL Gates: Towards 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: Sixth MIT Conference on Advanced Research in VLSI, 1990, pp. 263–278.

[MBM05] Mui, M. L.; Banerjee, K.; Mehrotra, A. “Supply and Power Optimization in Leakage-Dominant Technologies”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24–9, Sep 2005, pp. 1362–1371.

[MDM+95] Mutoh, S.; Douseki, T.; Matsuya, Y.; Aoki, T.; Shigematsu, S.; Yamada, J. “1-V Power Supply High-speed Digital Circuit Technology with Multithreshold-Voltage CMOS”, *IEEE Journal of Solid-State Circuits*, vol. 30–8, Aug 1995, pp. 847–854.

[MGC12] 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: Symposium on Integrated Circuits and Systems Design (SBCCI), 2012, pp. 1–6.

[MM95] Manohar, R.; Martin, A. J. “Quasi-delay-insensitive Circuits are Turing-Complete”, Technical Report, California Institute of Technology - CalTech, 1995, 14p, TR no. CS-TR-95-11, available at http://vlsi.cornell.edu/∼rajit/ps/qdi.pdf.

[MN06] Martin, A. J.; Nyström, 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: IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2014, pp. 53–60.

[MOPC13] Moreira, M. T.; Oliveira, C. H. M.; Porto, R. C.; Calazans, N. L. V. “NCL+: Return-to-one Null Convention Logic”. In: 56th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), 2013, pp. 836–839.

[Mor16] Moreira, M. T. “Asynchronous Circuits: Innovations in Components, Cell Libraries and Design Templates”, Ph.D. Thesis, PPGCC-Faculty of Informatics, PUCRS, Porto Alegre, 2016, 276p.

[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, Sep 2014, pp. 313–320.

[Mye01] Myers, C. “Asynchronous Circuit Design”. John Wiley & Sons, Inc., 2001, 419p.

[PN13] Pons, M.; Nagel, J. “Ultra Low-power Standard Cell Design using Planar Bulk CMOS in Subthreshold Operation”. In: International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2013, pp. 9–15.

[PSAA16] Parsan, F. A.; Smith, S. C.; Al-Assadi, W. K. “Design for Testability of Sleep Convention Logic”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24–2, 2016, pp. 743–753.

[RCN03] Rabaey, J. M.; Chandrakasan, A.; Nikolic, B. “Digital Integrated Circuits: A Design Perspective”. Prentice Hall, 2003, 2nd ed., 761p.

[SAAR11] Sethuram, R.; Arabi, K.; Abu-Rahma, M. “Leakage Power Profiling and Leakage Power Reduction using DFT Hardware”. In: 29th IEEE VLSI Test Symposium (VTS), 2011, pp. 46–51.

[SF01] Sparsø, J.; Furber, S. “Principles of Asynchronous Circuit Design – A Systems Perspective”. Springer, 2001, 354p.

[Smi02] Smith, S. C. “Speedup of Self-timed Digital Systems using Early Completion”. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2002, pp. 98–104.

[TBV12] Thonnart, Y.; Beigné, E.; Vivet, P. “A Pseudo-Synchronous Implementation Flow for WCHB QDI Asynchronous Circuits”. In: IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2012, pp. 73–80.

[vBHP95] van Berkel, K.; Huberts, F.; Peeters, A. “Stretching Quasi Delay Insensitivity by means of Extended Isochronic Forks”. In: IEEE Conference on Asynchronous Design Methodologies (ASYNC), 1995, pp. 99–106.

[VJD14] Vangal, S. R.; Jain, S.; De, V. “A Solar-Powered 280mV-to-1.2V Wide-Operating-Range IA-32 Processor”. In: IEEE International Conference on IC Design Technology (ICICDT), 2014, pp. 1–4.

[VWC+10] Vitale, S. A.; Wyatt, P. W.; Checka, N.; Kedzierski, J.; Keast, C. L. “FDSOI Process Technology for Subthreshold-Operation Ultralow-Power Electronics”, *Proceedings of the IEEE*, vol. 98–2, Dec 2010, pp. 333–342.

[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 – Bulk and FDSOI CMOS Evaluation for Sub-threshold Operation (Results)

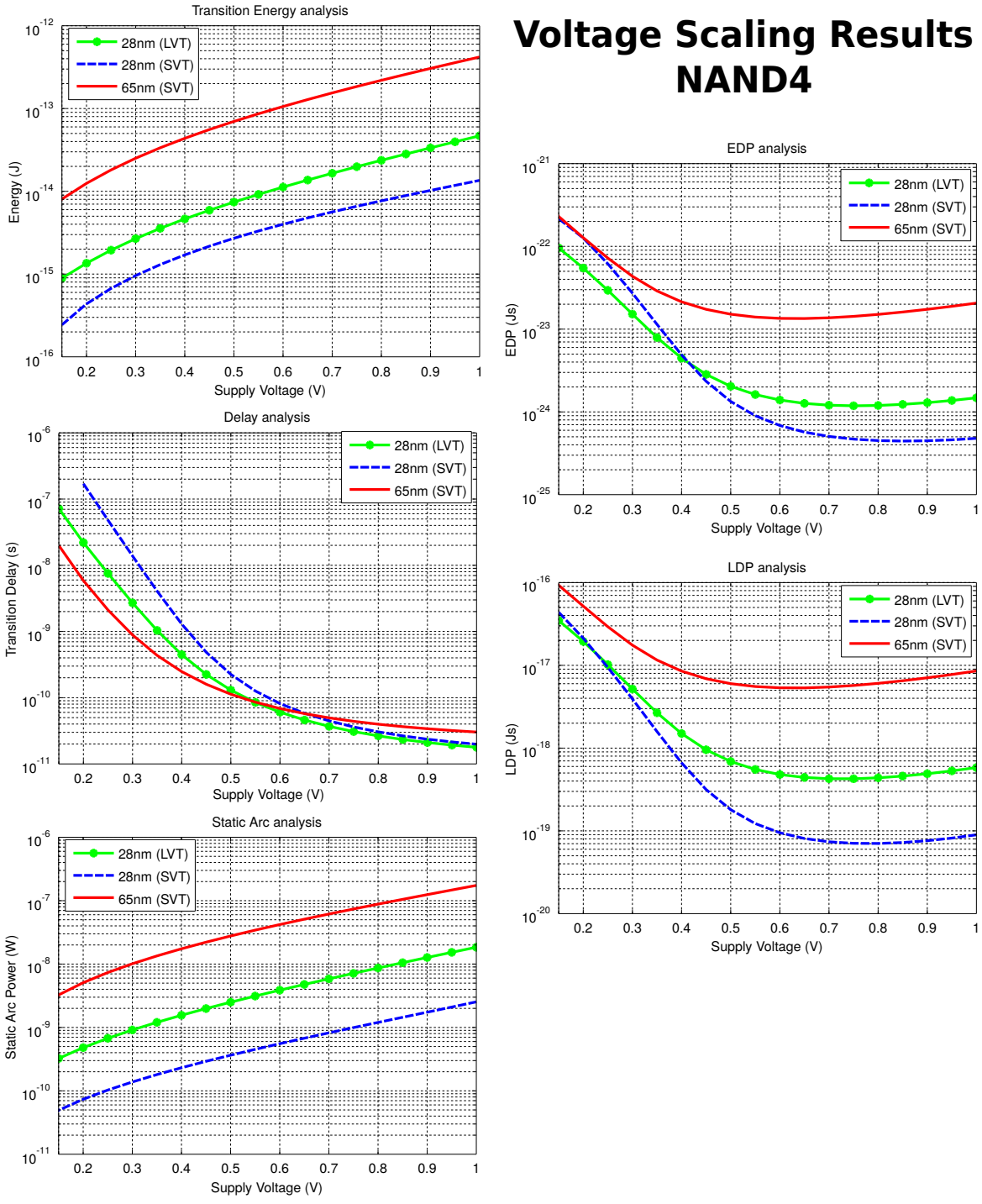


Figure A.1: VS analysis on a 4-input NAND gate covering delay, energy, leakage, EDP and LDP results.

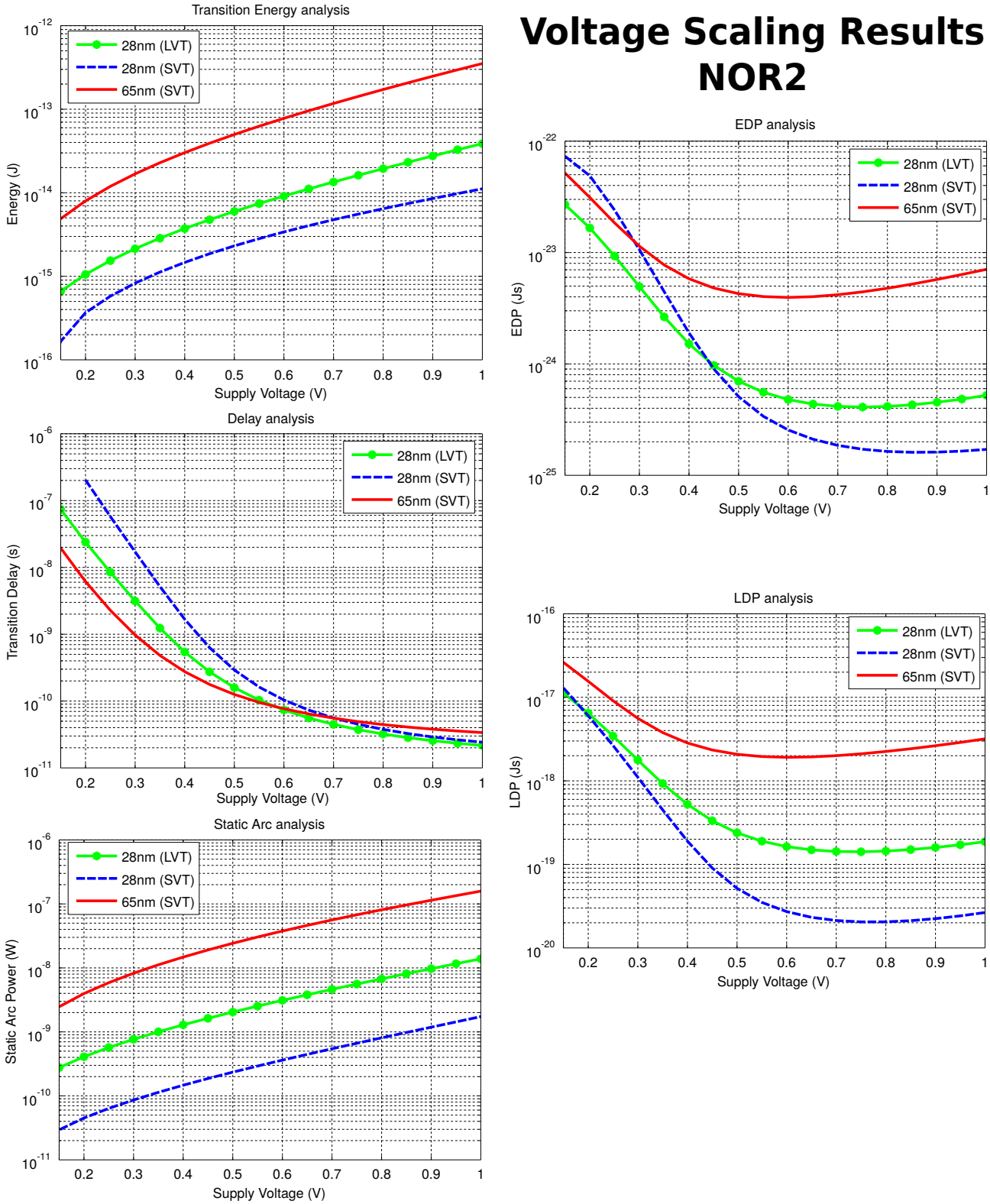


Figure A.2: VS analysis on a 2-input NOR gate covering delay, energy, leakage, EDP and LDP results.

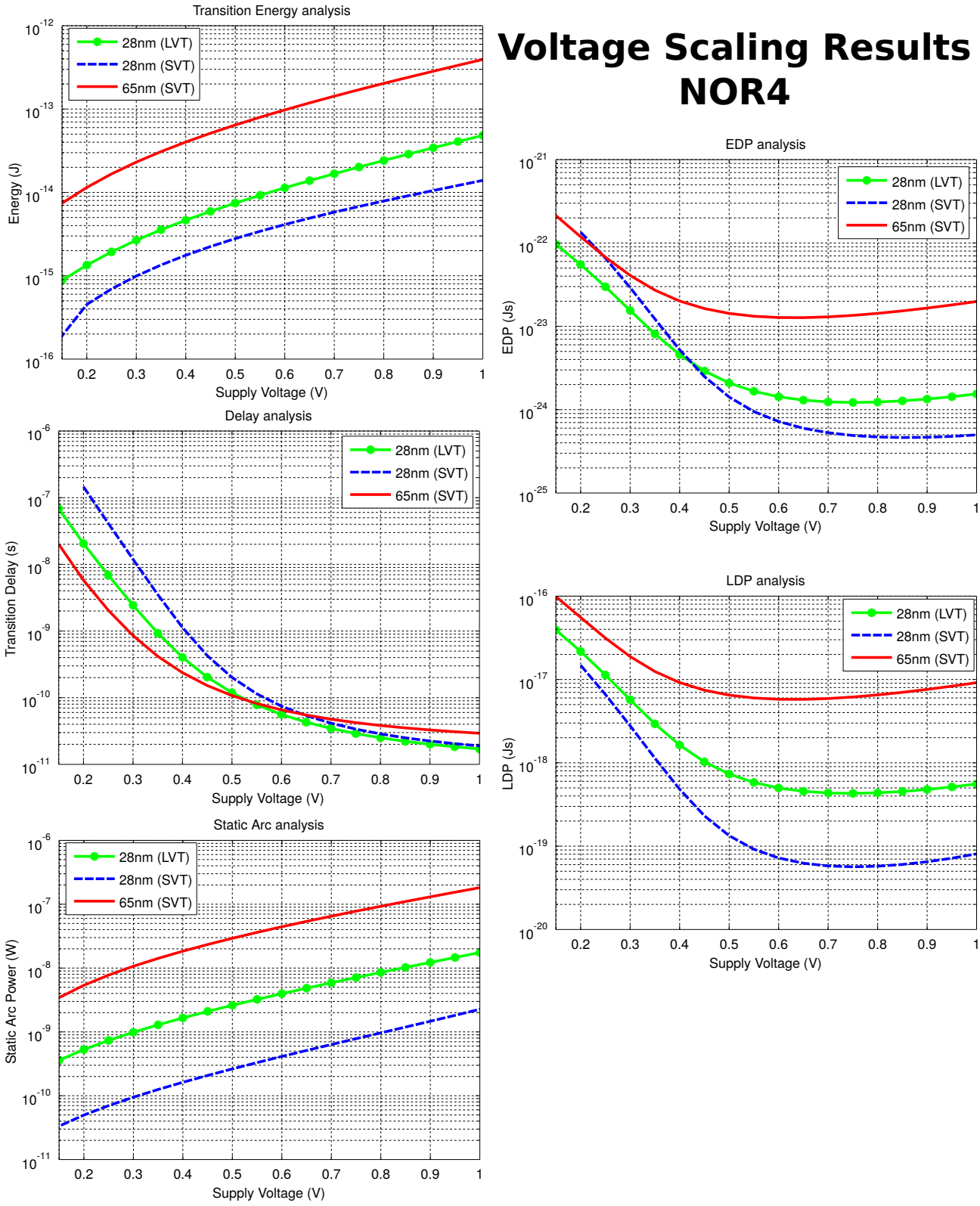


Figure A.3: VS analysis on a 4-input NOR gate covering delay, energy, leakage, EDP and LDP results.

Appendix B – Suppressed for the Purpose of Intellectual Property Protection