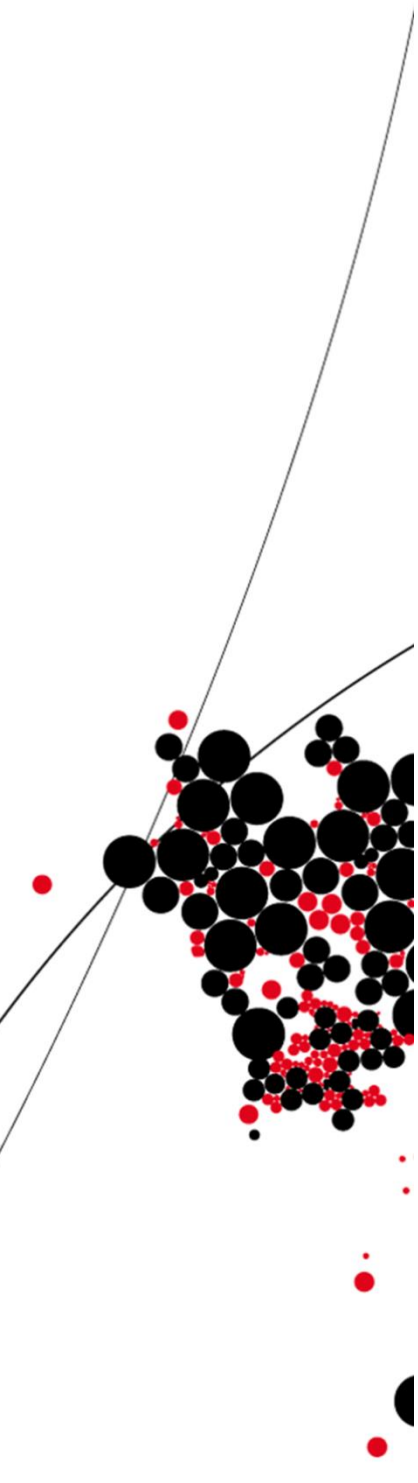
UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics and Computer Science

Embedded Systems

An Asynchronous Approach for Designing Robust Low Power Circuits



Committee University of Twente:
Anuradha Ranasinghe (M.Sc.)
Dr. Ir. S.H. Gerez
Dr. Ir. A.B.J. Kokkeler
Dr. Ir. M.S. Oude Alink
Dr. Ir. Jan Kuper

Student:
Shubham Yadav (s1911996)

UNIVERSITY OF TWENTE, THE NETHERLANDS

M.SC. THESIS

An Asynchronous Approach for Designing Robust Low Power Circuits

Author:
Shubham Yadav

Graduation Committee:
Anuradha Ranasinghe, M.Sc.
Dr. Ir. S.H. Gerez
Dr. Ir. A.B.J. Kokkeler
Dr. Ir. M.S. Oude Alink
Dr. Ir. Jan Kuper

Computer Architecture for Embedded Systems (CAES)
Faculty Of Electrical Engineering, Mathematics And Computer Science

August 30, 2019

"What is not started will never get finished."

Johann Wolfgang von Goethe

Abstract

This thesis work investigates asynchronous design techniques for robust low power digital circuits. Asynchronous circuits can be an alternative to the widely-used synchronous design style as they can operate at very low voltages and are mostly immune to PVT (Process-Voltage-Temperature) variations. Asynchronous designs are also well known for their event-driven functionality where the power consumption is ideally zero while the design waits for an event. There are multiple handshaking protocols, design templates and transistor-level cell designs which contribute to a large design space for asynchronous designs. In this thesis work, this design space is studied and careful design decisions are taken at every step. A complete flow of circuit design is presented along with assumptions taken while exploring the design space. An unsigned 64-bit Kogge-Stone adder is taken as a benchmark design. The investigation focuses on achieving performance improvements (speed, power, etc.) as well as the use of design tools to evaluate performance. To perform a fair comparison, this 64-bit adder is also designed using the synchronous style and comparison results are presented.

Acknowledgements

This research is the product of collective efforts put in by many people and I take this opportunity to acknowledge their contributions. First and foremost, I would like to thank my daily supervisor Anuradha Ranasinghe and my main supervisor Dr. Ir. Sabih Gerez who have been of immense help to me and without their guidance, this project would not have been possible. Their persistent encouragement and motivation along with their profound insight inspired me to achieve my best. My gratitude for their timely solutions for the problems I faced during the course of this work is boundless. I am also highly grateful to them for providing me with all the possible facilities required for the successful completion of the project.

I would also like to thank my committee members Dr. Ir. A.B.J. Kokkeler, Dr. Ir. M.S. Oude Alink and Dr. Ir. Jan Kuper for their valuable suggestions and their guidance during the work. They helped me by providing their opinions and evaluating my work from different perspective. Bert Helthuis, supporting staff of CAES group also helped me in providing technical support for the tools and also solved a lot of technical issues during my thesis work.

At last, I would like to acknowledge the efforts and support provided by my parents and my friends which helped me mentally and emotionally and kept me highly motivated during my entire journey at this university. I would also appreciate the efforts of teaching, administrative and technical staff of Computer Architecture for Embedded Systems (CAES) group at this university for aiding in this endeavor.

Contents

List of Figures

List of Tables

List of Abbreviations

ARM	Advanced Reduced Instruction Set Computing Machine
ASIC	Application Specific Integrated Circuit
ATPG	Automatic Test Pattern Generator
CAD	Computer Aided Design
CMOS	Complementary Metal Oxide Semiconductor
CPTL	Complementary Pass Transistor Logic
CSCD	Current Sensing Completion Detection
DCD	Data Control Decomposition
DI	Delay Insensitive
DIMS	Delay Insensitive Minterm Synthesis
DPCD	Datapath Completion Detector
DUV	Design Under Verification
EDA	Electronic Design Automation
EMI	Electro-Magnetic Interference
FDSTDL	Foot Driven Stack Transistor Domino Logic
FDVS	Full-range Dynamic Voltage Scaling
FIFO	First In First Out
GCD	Greatest Common Divisor
IEEE	Institute of Electrical and Electronics Engineers
IL	Integrated Latch
IoT	Internet of Things
LCD	Left Hand Completion Detector
MT-CMOS	Multi-Threshold CMOS
MT-NCL	Multi-Threshold Null Convention Logic
NCL	Null Convention Logic
NMOS	N-type Metal Oxide Semiconductor
NRZ	Non Return to Zero

PCFB	Pre-Charge Full Buffer
PCSL	Pre-Charge Static Logic
PCHB	Pre-Charge Half Buffer
PDK	Process Development Kit
PMOS	P-type Metal Oxide Semiconductor
PTM	Predictive Technology Model
PTL	Pass Transistor Logic
PVT	Process-Voltage-Temperature
QDI	Quasi Delay Insensitive
RCD	Right Hand Completion Detector
RL-NCL	Register Less Null Convention Logic
RSPCFB	Reduced Stack Pre-Charge Full Buffer
RSPCHB	Reduced Stack Pre-Charge Half Buffer
RTZ	Return To Zero
SA	Sense Amplifier
SAHB	Sense Amplifier Half Buffer
SCL	Sleep Convention Logic
SDDS-NCL	Spatially Distributed Dual Spacer Null Convention Logic
SDI	Scalable Delay Insensitive
SI	Speed Independent
TSMC	Taiwan Semiconductor Manufacturing Company
TT	Typical Typical
UMC	United Microelectronics Corporation
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WCHB	Weak Conditioned Half Buffer

Chapter 1

Introduction

Current era of industries present in the fields like *mobile communication, medicine, high performance computing, research, entertainment*, etc., heavily rely on the semiconductor industry. The development of these industries go hand in hand with the development of semiconductor industry. As the semiconductor industry developed, many challenges were faced at every step. These challenges involved wire resistance, high mutual capacitance, cross-talk and recently the small feature size of transistors which is reaching atomic limits [1]. As the need for more and more performance is being demanded by different industries, these roadblocks in the progress of semiconductor industry appear to be bigger than usual.

New research areas such as Internet of Things (IoT) require the remote devices to have enough computational power for the required job with least power consumption. This power consumption is governed by the ideology that remote devices should operate for longer duration of time: days, months or may be even years, with minimum power/battery management. New trend in digital design industry focuses on high performance and high throughput devices with low power consumption. Semiconductor industry has matured enough to provide us with high performance designs but research is still going on to make them low power.

At the top level, we have two broad categories of designing digital circuitry : *synchronous* and *asynchronous*. The predominant circuit design style is synchronous designing which uses Complementary Metal Oxide Semiconductor (CMOS) static logic gates and global clock signals to ensure synchronization and to regulate state changes in the design. However, as the circuit becomes complex and process variations increase, routing global clock signal across the design becomes increasingly problematic. Managing clock skew and jitter, fulfilling setup and hold time requirements in the design pose as a very big design challenge. Apart from these challenges, synchronous designs are power hungry too. Due to all these challenges, other design styles from asynchronous domain are becoming increasingly interesting alternatives.

In simple form, asynchronous designs remove the global clock signal in favor of distributed local handshaking signals to control data transfer and change of state. Synchronous designs follow a well-defined, rather simple constrained design styles. The relative simplicity of the constrained design style enables the development of Computer Aided Design (CAD) tools that automate large portions of the design process, significantly reducing design time [1]. Asynchronous designs, however, have varied design styles and a big design space, and, thus lack good CAD tools and testing techniques.

1.1 Synchronous Designs

Synchronous designing has been the dominant methodology since 1960s. In this design technique, the system consists of sub-systems controlled by one or more clock signals that synchronize tasks and the communication between those sub-systems. In traditional synchronous Application Specific Integrated Circuit (ASIC) flow, combinational logic is placed in between flip-flops(FF) to process the data, as shown in Figure 1.1. In this figure, "PI" and "PO" stands for primary inputs and primary outputs respectively.

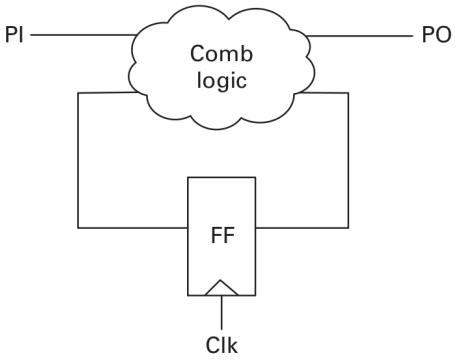


FIGURE 1.1: Structure for semi-custom synchronous designs [1]

As per the flow defined, combinational block must complete its operation within one clock cycle under all possible input combinations, from all reachable states, in the worst-case operating environment. In this way, the clock ensures synchronization among combinational blocks by guaranteeing that the output of every combinational block is valid and ready to be stored before the next clock period begins. In fact, the data at the inputs of the flip-flops may exhibit glitches or hazards, as long as they are guaranteed to settle before the sampling clock edge arrives [1]. In order to guarantee that the data is stable when sampled, the clock period should account for the worst-case delay including clock skew and all process variations. This requirement leads to introduction of safety margins for the clock period.

At a global level, we can look at the advantages and disadvantages of synchronous designs to get an idea about this designing approach. The advantages are as follows :

- Due to constrained design style, synchronous designs are easy to develop and can be mapped to any compatible technology.
- Global clock signal makes it very easy to synchronize events and communicate data at interfaces of various functional blocks.
- Longest combinational path viz. critical path along with some safety margins is considered to decide the period of clock signal. This safety margin consideration helps designer in avoiding hazards and glitches. Although there is a safety margin, glitches can still be present in the design, and, additional efforts are required to avoid glitches.

- Cell characterization for synchronous standard cells is straightforward for both front-end and back-end use. This reduces the design time and makes design analysis easier.
- There are a lot of CAD tools which focus on the designing of optimized synchronous circuits. Existence of wide variety of CAD tools promotes the usage of synchronous designs.
- Synchronous designs are easy to test for stuck-at-faults. There are tools such as Automatic Test Pattern Generator (ATPG) which generate specific test patterns to test the design.

The disadvantages are as follows :

- Global clock which makes designing circuits simpler is the major power consumer in the design. The entire clock generation and propagation circuitry can consume from 30% to 50% of total power provided to the design [2] [3].
- Presence of clock and safety margin in the clock period makes the circuit run at the worst case timing, which is decided by the longest combinational path of the circuit.
- Due to a periodic clock signal, there is high peak noise and Electro-Magnetic Interference (EMI) in the circuit.
- Designing and laying out high frequency clock signals with other components pose as a challenge to modern day designers.
- Synchronous designs are sensitive to Process-Voltage-Temperature (PVT) variations. Additional safety margin is considered to cope up with these variations.
- As transistor feature size has decreased, leakage power has increased and it is now very difficult for designers to meet required power budget.

1.2 Asynchronous Designs

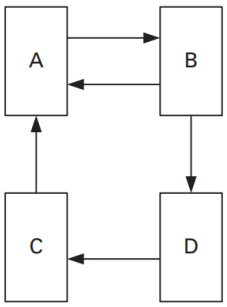


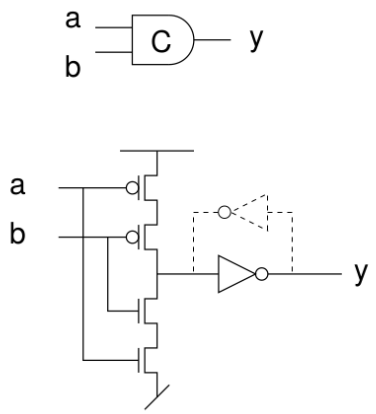
FIGURE 1.2: Asynchronous design using channels [1]

As the designing of synchronous circuits is getting complicated along with all the disadvantages, the trend is to shift to a newer approach for designing circuits. As mentioned earlier, major portion of power dissipation in a synchronous design is because of the clock circuitry which includes clock generation circuit, wires

and buffers. These wires are used to propagate the clock signal over the entire circuit and buffers are used to avoid skewing of the clock signal. Looking at this, one can say that removing clock and designing the circuit accordingly can help in reducing the overall power figure and might lead to faster executions. This is often the motivation for designing asynchronous circuits. In the absence of global clock signal, asynchronous designs rely on handshaking between functional blocks to transfer data. The group of data and handshaking signals is often termed as "Channel" [1]. The abstract design of an asynchronous system using channels is presented in Figure 1.2. Sub-systems "A", "B", "C" and "D" are the functional blocks whereas the arrows performing the communication between them are channels.

Various asynchronous design styles exist in literature based on different encoding schemes, timing assumptions and different handshaking protocols. There are several design flows [1] being used by designers to implement the asynchronous designs. They are :

- First flow involves designing leaf cells (smallest element that can perform handshaking) with minimum number of transistors possible and connecting them manually with other leaf cells to get a design. This entire process is manual and does not involve any specific tooling.
- Second flow involves using the pre-existing synchronous synthesis tools and translating synchronous gate level netlist into equivalent asynchronous netlist. This process involves removing clock signals and replacing them with handshaking signals.
- Third design flow involves usage of syntax-directed translation from high level language that has built-in primitives like "SEND" and "RECEIVE" for transferring data asynchronously. Verilog supports such primitives for asynchronous designing [1].



Some specifications:

1: if $a = b$ then $y := a$

2: $a = b \mapsto y := a$

3: $y = ab + y(a + b)$

4:

a	b	y
0	0	0
0	1	no change
1	0	no change
1	1	1

FIGURE 1.3: Muller C-element [4]

At the transistor level, some design approaches use standard synchronous cells while other approaches use non-standard custom cells. One such example of a non-standard cell is Muller C-element [4] which is presented in Figure 1.3. This cell changes the output state when inputs are identical, else no change is observed on the output terminal.

Similar to synchronous designs, advantages and disadvantages of asynchronous designs need to be listed to get an overall idea of this designing approach. The advantages are as follows :

- Unlike synchronous designs, asynchronous designs no longer operate at worst case timing, instead, system performance depends on the average delay which leads to higher performance. This high performance is always not guaranteed and depends on the design choices made.
- In general sense, absence of clock circuitry leads to lower power consumption. Also, asynchronous designs are event driven which leads to low standby power.
- Since there is no periodic signal, asynchronous designs have lower EMI. Activities in an asynchronous circuit are uncorrelated, resulting in a more distributed noise spectrum and lower peak noise.
- More performance and less chip area can be achieved by employing dynamic logic in asynchronous designs.
- Some asynchronous designs are insensitive to PVT variations which makes them more robust. This insensitivity and robustness enables aggressive voltage scaling from nominal to near-threshold and sub-threshold regions. This can further lower down the power consumption.

The advantages discussed above are highly based on the design choices made. Usage of dynamic logic or removal of clock circuitry, etc. comes with some overheads and assumptions. These assumptions and overheads will be discovered later in this report. Apart from the advantages, there are some disadvantages of asynchronous designs too. They are as follows :

- Absence of clock circuitry necessitates some additional hardware which is meant for performing local handshaking. This additional hardware along with high interconnect density can lead to larger chip area.
- Hazard free design satisfying all the timing constraints often leads to more area. To ensure hazard free behavior, additional transistors are required which contributes in increasing the overall power consumption of an asynchronous design.
- Lack of CAD tools to automate the process of designing asynchronous circuits.
- Asynchronous circuits are very hard to test. In synchronous counterparts, we have scan chains and ATPGs which are widely used to test the circuit. But in asynchronous domain , there is no such testing strategy. Asynchronous circuits are built on feedback loops which are not handled by these testing techniques. Additional hardware is needed to be placed with the existing design to enable testing of the asynchronous designs.

1.3 Summary

Advantages and disadvantages discussed earlier are mostly accurate, but there are some deviations in real scenarios. For example: In terms of performance,

asynchronous circuits are considered to be superior as they do not wait for the end of clock cycle to start a newer operation. Earlier this assumption was true, but with the advancements in technology and having shorter critical paths by means of pipelining have led to higher performance figures for synchronous designs. Now the performance of both the design techniques is comparable and sometimes higher for synchronous designs. Also the choice of designing asynchronous circuits affects the power figures. Some techniques such as quad rail design often lead to very high power figures. Upcoming chapters try to explore asynchronous domain of hardware designing and provide in-depth knowledge of the same.

1.4 Thesis Outline

Ultimate goal of this thesis work is to explore the asynchronous design space by providing a quantitative and qualitative analysis based on a sufficiently complicated baseline design (a 64-bit unsigned Kogge-Stone adder) against its synchronous counterpart. Apart from comparison with the design from synchronous domain, the asynchronous design will also be compared with its own asynchronous variants which are based on different timing assumptions and design techniques. This work is divided across multiple chapters and a short description of these chapters is discussed here.

Chapter 2 presents the widely accepted classification of asynchronous designs along with handshaking mechanisms and some basic concepts which will help in exploring the asynchronous domain. Chapter 3 presents the design templates and cell design approaches taken by designers to implement asynchronous circuits. A comparative study between all the approaches taken is also presented in the chapter. Chapter 4 explains the importance of hysteresis mechanism and indicatability posed by the functional blocks. It also discusses about the conditions required for removal of hysteresis and effects caused by it. Chapter 5 presents basic components used in the circuit designing like registers and completion detectors. This chapter also discusses a few template designs based on which asynchronous variants of Kogge-Stone Adder are implemented. Chapter 6 presents different transistor level circuit design styles and also mentions their advantages and disadvantages. Chapter 7 explains the implementation of designs, tooling and testing techniques used along with the simulation results. Chapter 8 is the last chapter which presents the conclusion and possible future work. This report also includes appendices which present schematics of some basic cells used in circuit design and ways to simulate an asynchronous design using Very High Speed Integrated Circuit Hardware Description Language (VHDL).

Chapter 2

Asynchronous Design : Classification and Concepts

Synchronous circuits designed and developed today are based on two fundamental assumptions :

- All the signals are encoded as binary signals i.e. logic high and logic low.
- All components share a common and discrete notion of time i.e. clock.

Unlike synchronous circuits, asynchronous circuits do not have a notion of time. Instead, they use handshaking signals to manage the synchronization and flow of data between components. In synchronous terms, these signals are termed as local clocks whose period is decided by the actual delay of components, unlike worst case delay in synchronous circuits. Absence of global clock has various advantages as already discussed in Chapter 1.

In asynchronous circuits, the handshaking is implemented by means of "request" and "Acknowledge" signals. These handshaking signals ensure that local clock pulses are generated where and when needed. This reduces and randomizes the clock pulse generation over time which leads to less electromagnetic interference. In the absence of a global clock in asynchronous circuits, sometimes signals are required to be valid at all time instants and all signal transitions have a meaning. This means that if any asynchronous circuit has to be developed, then hazards and glitches must be avoided at all cost [1] [4].

2.1 Classification of Asynchronous Designs

Asynchronous circuits are classified based on the assumption of the delay model which comprises of the delay information of gates and wires. Generally, less restrictive delay constraints lead to designs with robust nature against temperature, voltage fluctuations, manufacturing process variations, etc. This classification is discussed below.

2.1.1 Delay Insensitive (DI) Designs

Delay insensitive design is the most robust of all asynchronous circuit delay models. It makes no assumption on the delay of wires or gates, i.e. they can have zero to infinite delay and can be time varying. In other words, a circuit that operates “correctly” with positive, bounded but unknown delays in wires and gates is termed as delay insensitive. To understand more about delay-insensitivity in asynchronous circuits, consider a block diagram present in Figure 2.1. In this figure, d_A, d_B and

d_C corresponds to gate delays, whereas, d_1, d_2 and d_3 corresponds to wire delays. A DI circuit works correctly for arbitrary values of d_A, d_B, d_C, d_1, d_2 and d_3. Unfortunately, the designing of delay insensitive circuits is very difficult and the designs made by this model mostly comprises of Muller C-element and inverters [1].

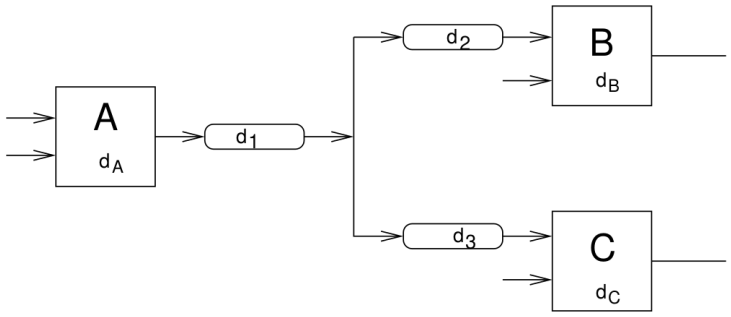


FIGURE 2.1: Circuit representing gates and wire delays [1]

To build delay insensitive circuits the smallest building block must itself be delay insensitive which makes it larger than a simple gate. As one example, a network of leaf cells (smallest building block) that communicate with delay-insensitive channels is a delay-insensitive circuit if we consider each leaf cell to be an atomic multi-output gate where delays within the cell are well verified to be hazard-free under all environmental conditions. These conditions include slewed input rates, output loading, and cross-talk due to over-the-cell routing [1]. If leaf cells can be designed taking care of delay insensitivity, then we can achieve robust DI designs.

2.1.2 Quasi Delay Insensitive (QDI) Designs

In a QDI circuit, as in a DI circuit, all gates and wires can have arbitrary delays, except for a set of designated wire forks labeled as isochronic. Refering to Figure 2.1, a QDI circuit works correctly for arbitrary values of d_A, d_B, d_C and d_1 with a condition that $d_2 = d_3$. This condition makes the wire fork isochronic. Isochronic forks, as the name suggests, have the additional constraint that the delay to the different ends of the fork must be the same. This strict and unrealistic definition is often interpreted as the difference in the times at which the signal arrives at the ends of an isochronic fork must be less than the minimum gate delay. In other words, the isochronic fork assumption can be relaxed to mean that the delay from one end of the fork to its terminal gate G must be less than the delay of any re-convergent fan-out path through any other end of the fork that also terminates at G. This assumption is made to have a hazard-free design.

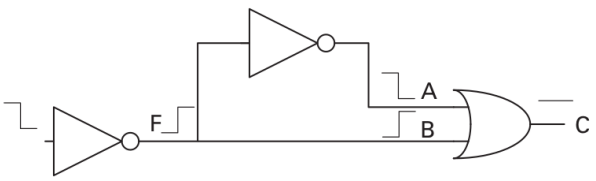


FIGURE 2.2: Isochronic Fork for re-convergent paths [1]

Consider the re-convergent circuit [1] depicted in Figure 2.2, if the fork F is isochronic, it is guaranteed that the rising transition at B arrives before the falling

transition at A. This means that there is no glitch at C. If the fork was not isochronic then a glitch at C could occur and the circuit would be hazardous. Apart from this assumption, no other timing analysis is needed to be carried out to ensure the correct functioning of QDI designs. DI/QDI designs are claimed to be immune to PVT variations, but guaranteeing the isochronicity of forks is difficult under these variations.

This isochronic fork assumption also simplifies the hardware (simpler completion detector circuitry) and reduces the overall transistor count. This reduced transistor count in the data path and in the control circuitry (handshaking signals) leads to a faster design when compared to DI designs.

2.1.3 Speed Independent (SI) Designs

According to this design style, gates can have arbitrarily large delays but all wire delays are negligible/zero. A speed independent circuit operates "correctly" assuming positive, bounded but unknown delays in gates and ideal zero delay wires. Refering to Figure 2.1, a SI circuit works correctly for arbitrary values of d_A, d_B, d_C with a condition that $d_1 = d_2 = d_3 = 0$. This means that all forks present in the design must be isochronic for correct operation of circuit. The SI model may be a practical model to do synthesis as long as a more detailed analysis of the underlying timing assumptions is used to verify correctness after physical design.

2.1.4 Scalable Delay Insensitive (SDI) Designs

Main idea [5] behind this methodology is to divide the entire system into parts of a reasonably smaller size, to design each part based on a more relaxed or optimistic delay model (like SI), and to design the interconnection parts or global parts based on the DI model. It is very realistic and provides with a nice base for the design of reasonably reliable (not entirely delay-insensitive) and high-performance asynchronous processors.

2.1.5 Bounded Delay Designs

In this model, delays are bounded by a minimum and maximum value. Circuit functions correctly if these bounds are met. Such timed circuits can have the advantages of being faster, smaller, and lower in power than their more robust counterparts, but their design and synthesis procedures are generally more complex. In addition, extensive post-physical-design analysis is needed to ensure that all timing bounds are met.

2.2 Handshaking Protocols [4]

Asynchronous circuits make use of handshaking protocols to ensure the correct flow of data. This section provides a classification of handshaking protocols used in asynchronous circuit designing.

2.2.1 Bundled Data Protocol

This protocol refers to a situation where the handshaking signals (Req - Request and Ack - Acknowledgment) are bundled with the data signals. It is also referred to

as *single-rail protocol*, which hints toward the usage of one wire to carry one bit of data. The signals between the atomic components in this protocol are presented in Figure 2.3. It can further be divided into two protocols based on the interpretations of handshaking signals. These protocols are : *4-phase bundled data and 2-phase bundled data*.

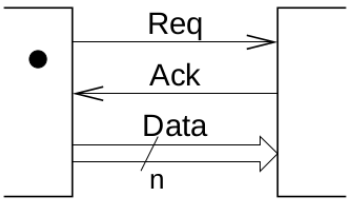


FIGURE 2.3: Signals in Bundled Data Protocol (Dot = Sender) [4]

4-phase bundled data protocol is also known as Return To Zero (RTZ) signaling or level signaling. Events happening in this protocol are presented in Figure 2.4 and can best be described as follows :

- Sender issues data and sets *Req* signal high.
- Receiver absorbs the data and sets *Ack* signal high.
- Sender responds by taking *Req* low (at this point, data is no longer guaranteed to be valid).
- At last, the receiver acknowledges this by taking *Ack* signal low. After this, the sender may initiate the next communication cycle.

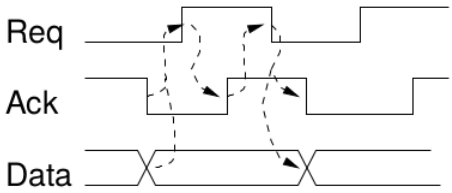


FIGURE 2.4: Signal transitions for 4-phase bundle data protocol [4]

Having simplicity as its advantage, this protocol has a disadvantage in its superfluous return-to-zero transitions that cost time and energy. If the time to process valid data (when "Req" signal is high) and time to process null data (when "Req" signal is low) are considered to be equal, then the resultant data rate or throughput gets reduced by a factor of 2.

To overcome these disadvantages, 2-phase bundled data protocol can be used which is also known as Non Return to Zero (NRZ) signaling or transition signaling. Now information on *Req* and *Ack* signal is transferred as transitions, and, there is no difference in $1 \rightarrow 0$ and $0 \rightarrow 1$ transition. Events happening in 2-phase bundled data protocol are shown in Figure 2.5 and can best be described as follows :

- Sender puts valid data on the bus and shows a transition (rising edge in Figure 2.5) on *Req* signal.

- Receiver accepts this data and shows a transition (falling edge in Figure 2.5) on *Ack* signal. This completes the communication and sender can now initiate a new cycle of communication.

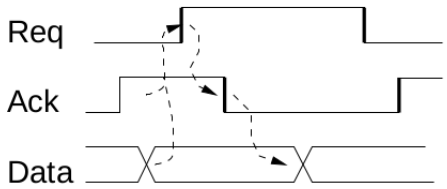


FIGURE 2.5: Signal transitions for 2-phase bundle data protocol [4]

Ideally the 2-phase bundled-data protocol should lead to faster circuits than the 4-phase bundled-data protocol (no time for processing null data, when "Req" signal is low), but often the implementation of circuits responding to transitions is complex and there is no general answer as to which protocol is best.

It is also important to note that all the bundled data protocols rely on *delay matching* for correct operation. Delay matching is a way for maintaining proper synchronization in asynchronous circuits by trying to match the delay of control circuitry with delay of datapath to ensure that correct data is read and written. The data should be valid before the *Req* signal is set high by the sender. This order should also be valid at the receiver, otherwise invalid data will be read by receiver. This order is ensured by delay matching. While designing the physical layout of a circuit, some problems are still faced in terms of delays, therefore, additional delay chains have to be introduced to cope up with the timing issue.

A 4-phase bundled data protocol is simple to implement using Muller's pipeline. Muller's pipeline is an arrangement of latches, combinational circuits, control circuits and delay chains which provides a basic setup for designing asynchronous circuits. A FIFO buffer with latches based on Muller's pipeline is presented in Figure 2.6. Also, an example circuit where a functional block is added in the path along with matching delays (in *Req* signal path) is presented in Figure 2.7. In spite of simple implementation, there a few drawbacks of 4 phase bundled data protocol: introduction of matching delays, slow speed and sensitivity to PVT variations.

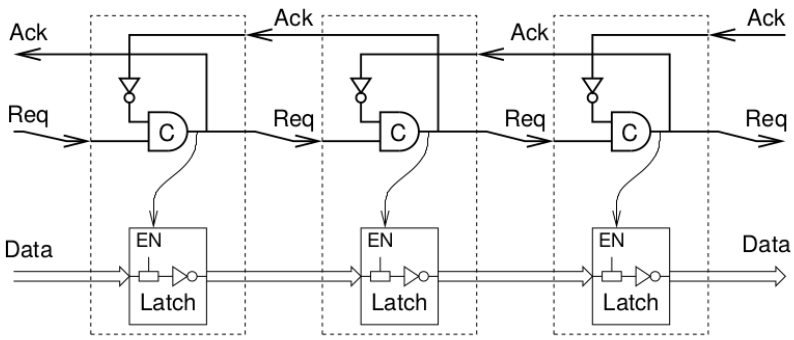


FIGURE 2.6: FIFO implementation based on Muller's pipeline [4]

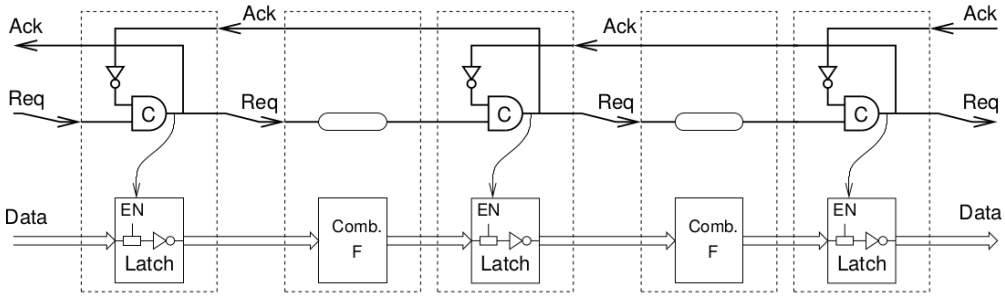


FIGURE 2.7: A general 4-phase Bundled Data Protocol implementation using Muller's pipeline [4]

A 2-phase bundled data pipeline also uses a Muller pipeline. But the control signals are interpreted as events or transitions, see Figure 2.8. For this reason special capture-pass latches are needed: events on the C and P inputs alternate, causing the latch to alternate between capture mode (capture input data) and pass mode (provide output data).

At the conceptual level, the 2-phase bundled-data approach is elegant and efficient; compared to the 4-phase bundled-data approach, it avoids the power and performance loss that is incurred by the return-to-zero part of the handshaking. However, edge sensitive devices are often more complicated than level sensitive devices. In addition to storage elements, conditional control logic that responds to signal transitions tends to be complex as well. Thus, the 2 phase bundled data approach may be the preferred solution in systems with unconditional data-flows and very high speed requirements.

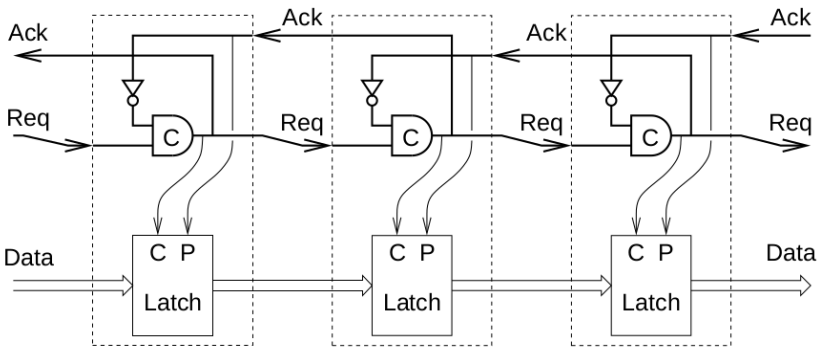


FIGURE 2.8: 2-phase Bundled Data Protocol implementation using Muller's pipeline [4]

2.2.2 Dual Rail / One-of-2 Protocol

In this protocol, the request signal is encoded into the data signal using two wires per bit of information. Similar to the bundled data protocol, this protocol is also divided into two other protocols : 4-phase dual rail and 2-phase dual rail. In 4-phase dual rail protocol, out of the two wires, one wire d.t (true rail) is used to signal logic 1 and other wire d.f (false rail) is used to signal logic 0. Interpretation of the logic levels and their respective meanings for 1 bit dual rail signal are presented in Table 2.1. "D.t" and "D.f" correspond to the "true" and "false" rails of 1 bit dual rail data.

D.t	D.f	Single-rail equivalent logic	Remark
0	0	-	Also referred as null value or spacer
0	1	'0'	Logic 0
1	0	'1'	Logic 1
1	1	-	Not used in dual rail protocol

TABLE 2.1: Interpreting 1 bit dual rail signal

Using two wires for one bit data makes this protocol robust and parties can communicate without worrying about delay matching, which makes this protocol delay-insensitive. In this 4-phase protocol, transition from one valid value to another valid value is not allowed and should always be separated with a null value. The steps followed in this protocol make this transition logic clear. The steps are :

- Sender issues a valid codeword (data).
- Receiver absorbs the codeword and sets acknowledge signal high.
- Sender responds by issuing the empty codeword (null value).
- At last, the receiver acknowledges this by taking acknowledge signal low. At this point the sender may initiate the next communication cycle.

The aforementioned steps can also be well understood with the waveform and state diagram presented in Figure 2.9.

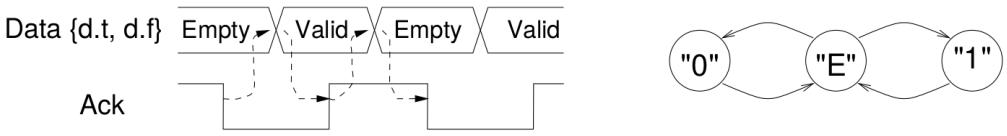


FIGURE 2.9: Waveform explaining dual rail 4-phase protocol[4]

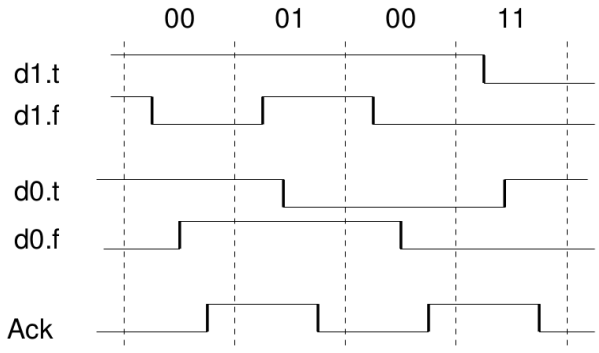


FIGURE 2.10: Waveform explaining dual rail 2-phase protocol[4]

In 2-phase dual rail protocol, the information is encoded as transitions on the pair of wire (d.t and d.f). Out of two wires, only one wire is allowed to perform a transition. Waveform for 2-phase dual rail protocol is presented in Figure 2.10.

From this figure, assuming that out of 4 cycles presented, first cycle represents the state where data is “00”. In the next cycle, d1.f and d0.t show transitions which means "01" is present on the data bus. Similarly other cycles can also be interpreted based on the transitions of different wires.

Similarly, on an N-bit channel a new codeword is received when exactly one wire in each of the N wire pairs has made a transition. There is no empty value; a valid message is acknowledged and followed by another message that is acknowledged. Absence of coming back to "empty value" makes this protocol fast but similar to 2-phase bundled data protocol, designing edge sensitive circuits is always complex.

Implementation-wise, 4-phase dual rail pipeline is also based on Muller’s pipeline. Considering an example of an N bit wide pipeline to understand the implementation, we can say that an N bit wide pipeline can be implemented by using a number of 1 bit pipelines in parallel. Since all the one bit pipelines works independently, it is upto the functional block to implement additional bit parallel synchronization logic to accept correct data. If bit-parallel synchronization is needed, the individual acknowledge signals can be combined into one global acknowledge using a C-element.

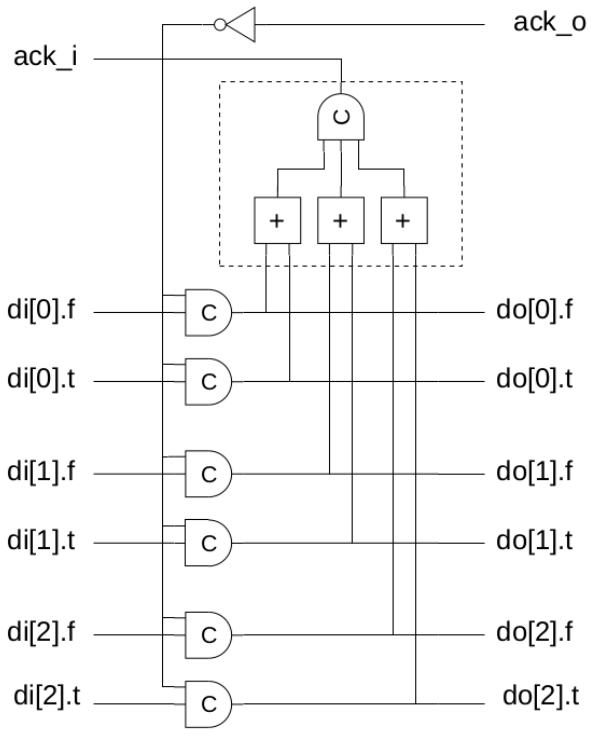
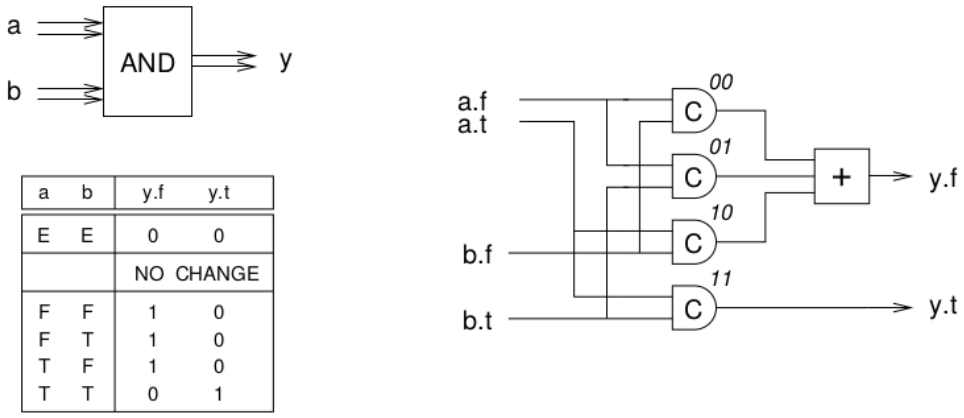


FIGURE 2.11: A N-bit latch with completion detection [4]

A N bit wide latch implementing this bit-parallel synchronization logic is presented in Figure 2.11. The OR gates and the C-element in the dashed box form a completion detector that indicates whether the N bit dual rail codeword stored in the latch is empty or valid. In some of asynchronous designs, apart from indicating the previous stage in the pipeline, signal “ack_i” is also used by functional blocks to start evaluating the valid (null) data impending on their input port. More information on this topic is presented in upcoming chapters.



a	b	y.f	y.t
E	E	0	0
		NO CHANGE	
F	F	1	0
F	T	1	0
T	F	1	0
T	T	0	1

FIGURE 2.12: A Dual rail AND gate using Muller C elements and 3 input OR gate [4]

It is also possible to design combinational circuits using 4-phase dual rail concept. Using Muller C-element and basic logic gates, we can design "dual rail gates". These dual rail gates can then be used to implement any combinational circuit (sum of minterms or sum of maxterms). Going this way leads to higher transistor count in basic dual rail gates and efficient implementation of these gates is also possible. One such example of a dual rail AND gate implemented with Muller C elements and 3 input OR gate is presented in Figure 2.12.

2.2.3 Higher radix / One-of-N protocol

Similar to Dual-Rail / One-of-2 protocol, One-of-4 protocol (aka Quad-Rail protocol) can also be considered, where the channel communicates 2 bits of data by changing only one data wire. This protocol leads to less power consumption than Dual-Rail channel with no additional data wire per bit. Other higher radix protocols also exist, like One-of-8 which requires more data wires per bit rather than two data wires per bit in Dual-Rail and Quad-Rail. Moving to higher radix leads to higher power consumption and more transistor count. Comparison between dual rail and quad rail design is presented in upcoming section.

2.2.4 Single-track One-of-N channel protocol

A single-track One-of-N channel protocol [6] has One-of-N data with no additional acknowledgment wire running between sender and receiver, and is implemented with two phases. The steps are mentioned below :

- Sender drives one of the N wires high, thereby sending a token.
- After receiving this token the receiver drives the same wire low.
- After driving the wire to its desired state, the sender and receiver(s) tri-state the wire to ensure that they do not try to drive it in opposite directions at the same time.

The two-phase single-track protocol avoids the overhead of the null phases without requiring the sender and receiver to react to different transitions, yielding substantially higher performance than four-phase protocols. A single-track protocol

with communication channel (along with *Ack* signal generation) is presented in Figure 2.13. From this figure, we can say that number of wires meant for handshaking reduces in this protocol (No separate *Ack* signal wire in the channel).

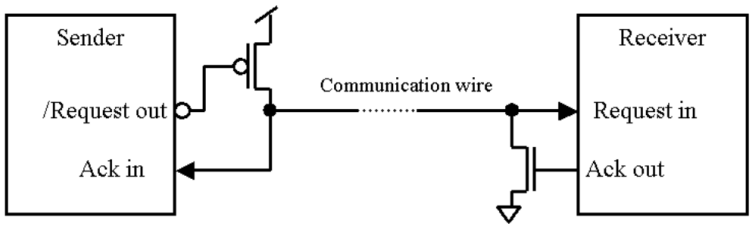


FIGURE 2.13: Handshaking signals for single track protocol [4]

In comparison with four-phase 1-of-N protocols there are fewer transitions per bit, resulting in substantially lower power. But in comparison with bundled-data channels, however, the number of transitions per bit is often larger, yielding a higher power consumption per bit transmitted.

2.3 Quantitative Figures

To get an idea about the estimated power consumption and transistor count for different handshaking protocols, an example of Thumb Decoder is considered here. The Thumb instruction set [7] is a subset of the most commonly used 32-bit Advanced Reduced Instruction Set Computing Machine (ARM) instructions. Thumb instructions are each 16 bits long, and have a corresponding 32-bit ARM instruction that has the same effect on the processor model. Thumb decoder is used to decode these subset of instructions.

Thumb Instruction decoder design [8] presents some numbers which are helpful to make choice of protocols while designing circuits. The logic was constructed using gates obtained from standard cell libraries. The single-rail implementation used a commercial static standard cell library whereas cell libraries for the dynamic dual-rail and 1-of-4 gates had to be constructed for the project [8]. From an encoding point of view, Figure 2.14 presents the figures for estimated area (wires/bit) and energy (transitions/bit). Also, Figure 2.15 presents the estimated transistor count and Figure 2.16 presents the power consumption figures for thumb instruction decoder. In Figure 2.16, power saver design corresponds to a modified design which uses bypass detectors. Bypass detectors avoids immediate processing and generates early completion detection signal. More information can be found in [8].

	Area wires/bit	**Energy** transitions/bit
single-rail	1	1/2 (average)
dual-rail (RTZ)	2	2
dual-rail (NRTZ)	2	1
1-of-4 (RTZ)	2	1
1-of-4 (NRTZ)	2	1/2

FIGURE 2.14: Comparison of Encoding [8]

	request activated	normalized size
single-rail	764	1.0
dual-rail	1201	1.6
1-of-4	1378	1.8

FIGURE 2.15: Number of transistors (following RTZ) [8]

	request activated	normalized power
single-rail	12 mW	1.0
dual-rail	38 mW	3.2
1-of-4	24 mW	2.0
1-of-4 (power saver)	16 mW	1.3

FIGURE 2.16: Datapath Power Consumption [8]

Note that the transistor count presented does not consider the transistors required for delay matching in single rail design. Also, additional transistors (196 transistors) are present in 1-of-4 encoding for half-word aligning the address offset (see [8] for more information on half-word alignment).

2.4 Summary

After studying the classification of asynchronous designs and different handshaking protocols, following points should be noted :

- 2-phase circuits are generally faster than the 4-phase equivalents but designing edge sensitive 2-phase circuits is cumbersome.
- Bundled protocols are highly dependent on the process variations, temperature and voltage fluctuations. Delay elements could also be vulnerable to PVT variations and therefore a bigger error margin is taken to cope up with this scenario.
- Designing data dependent circuits is difficult with bundled data protocol, because of the use of matched delays. Techniques like speculative completion (see [4]) can be introduced to choose between different matched delays based on the inputs. Other option is to go for worst case delay, which might lead to very slow circuits.
- Data dependent circuits work well with DI and QDI protocols because they embed request and data signals together and no additional delay chain is required.
- Single track protocol seems simple and power efficient, but designing the functional block is a tedious task. All the blocks should be hazard/glitch free.

2.5 Conclusion

After summarizing the qualities of different protocols, it can be concluded that a DI or QDI design (Dual-Rail) following 4-phase handshaking protocol will be better in terms of less designing and verification efforts and will also be immune to PVT variations. The efforts required for timing analysis is minimum for DI / QDI designs. The overhead due to the increased number of transistors and interconnect density can be reduced with the aid of template level optimizations and timing assumptions. This will be discussed in upcoming chapters.

Chapter 3

Asynchronous DI/QDI Pipeline Templates and Techniques

The QDI circuits work correctly regardless of the values of the delays in the gates and wires, except for the isochronic wire forks. By definition, the difference in the times at which a signal arrives at the ends of an isochronic fork is assumed to be less than the minimum gate delay. If these isochronic forks are guaranteed to be physically localized to a small region, this assumption can be easily met and the circuits can be practically as robust as DI circuits.

As concluded in the previous chapter, QDI circuits have less designing and verification steps and are immune to PVT variations, so, we will explore more into this area in current chapter. In the following sections, several DI / QDI design templates are discussed with their comparative study. There are a lot of DI / QDI templates present and discussed in the references mentioned but some of the templates do not fulfill the goal of low-power high-performance circuit design.

Within the DI / QDI approaches, there are two general asynchronous pipeline structures: 1) Data Control Decomposition (DCD) and 2) Integrated Latch (IL) pipelines. The DCD pipeline separates asynchronous controllers and datapaths where they can be designed individually and independently. Such pipelines are simple but less speed efficient as a large number of cells would be grouped together to form a block-level pipeline stage, hence resulting in a longer critical path. In IL pipeline, there is no clear distinction between controllers and datapath. Both these separate entities are joined together to form a microcell. In this chapter, focus is placed on both DCD and IL type of templates.

3.1 Weak Conditioned Half Buffer (WCHB) [1] [9]

WCHB is a 4-phase QDI template based on weak-conditioned logic. In this template, the validity (presence of data) and neutrality (absence of data) of the output data imply the validity and neutrality of the corresponding input data, which is also known as weak-conditioned / strong indicating logic. This type of logic for functional blocks requires a special mechanism of hysteresis at the gate level which helps in asserting the validity and neutrality of input data. More details on this strong indicating logic and hysteresis mechanism is presented in Chapter 4. In a 4-phase protocol, pipeline is filled with alternating null and valid values, which means that the left and right channels of a pipeline stage cannot simultaneously hold two distinct data tokens, so, this circuit is said to be a half buffer or to have a slack of 1/2. Due to these reasons, this template is also known as Weak Conditioned

Half Buffer (WCHB).

This WCHB template is used for implementing 1-of-N protocol (here, N = 2 : dual rail). The block diagram and the gate level implementation of WCHB template for dual rail protocol are presented in Figure 3.1. In this figure, 'W' corresponds to the functional block whereas Right Hand Completion Detector (RCD) corresponds to the completion detector circuit used for detecting the valid and null data. More complex weak-conditioned function blocks can be implemented by changing the internal structure of block 'W'. WCHB heavily relies on C elements which means if more complex function blocks are to be developed, the complexity of C elements will also increase. However, this means increasing the complexity of both the N-type Metal Oxide Semiconductor (NMOS) and P-type Metal Oxide Semiconductor (PMOS) stacks of the C-element. The two-input C-element (as shown in Figure 1.3) used in Figure 3.1 already has two PMOS transistors in series and a general rule of thumb is not to exceed that number. Therefore implementing any complex function with the WCHB template leads to poor performance.

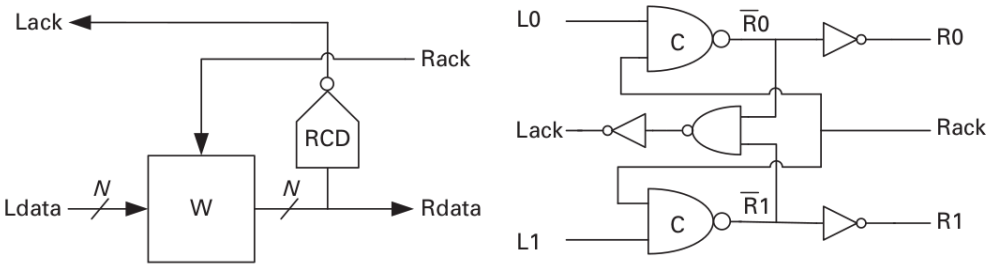


FIGURE 3.1: Block diagram and gate level implementation of WCHB pipeline template RCD [1]

The WCHB template mostly comprises of the buffering and control parts of an asynchronous design template. To have a fully functional design, some data processing logic style has to be adopted. WCHB being a weak-conditioned template requires the data processing part to follow weak-conditioned property. A frequent choice in this case is the Delay Insensitive Minterm Synthesis (DIMS) (discussed in later section), which together with WCHB forms the template WCHB / DIMS design template.

3.2 Pre-Charge Half Buffer (PCHB) [1] [9]

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 an input completion detector block (Left Hand Completion Detector (LCD)) also. Apart from this, PCHB also makes use of domino logic (a type of dynamic logic), which avoids the extensive stacking of PMOS transistor found in WCHB circuits. It should be noted that domino logic is not suitable for voltage scaling because of less immunity to noise. So, PCHB is not a good candidate for ultra-low power applications. Although, Foot Driven Stack Transistor Domino Logic (FDSTDL) [10] can be used for designing CMOS domino logic gates for the reduction in leakage power and improved noise performance.

PCHB template has an output completion detector, denoted as RCD. However, unlike WCHB, the validity and neutrality of the input channels are checked using

LCD. Moreover, the function block 'F' need not be weak-conditioned and instead can be implemented using one level of domino logic with two control inputs en (enable) and pc (pre-charge), as shown in Figure 3.2.

Because the function block need not be weak-conditioned, it can evaluate before all the inputs have arrived (if the logic allows). However, the template only generates an acknowledgment signal "Lack" after all the inputs have arrived and the output has evaluated. The implementation shown above 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 (discussed in next section).

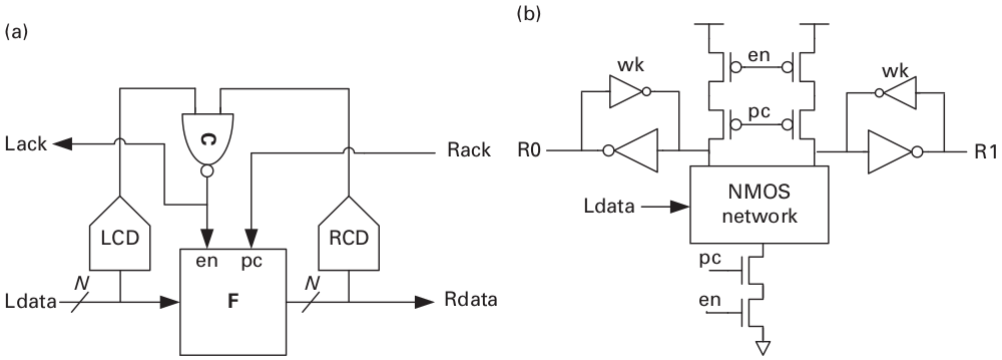


FIGURE 3.2: Block Diagram and Dual rail domino function block in PCHB pipeline template [1]

3.3 Pre-Charge Full Buffer (PCFB) [1]

The PCFB template is more complicated than the PCHB template because of two asymmetric C-elements, as shown in Figure 3.3. Asymmetric C-elements are similar to C-elements but have inputs that only effect the operation of the element when transitioning in one direction (either $1 \rightarrow 0$, or, $0 \rightarrow 1$). In this template, left and right side of the buffer can store two different data tokens which means that the value of slack is equal to 1 (thats why the name full buffer).

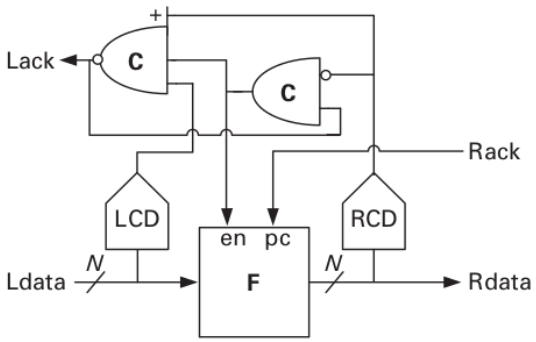


FIGURE 3.3: Block Diagram of PCFB template [1]

Optimizations : One optimization that can be applied to the PCHB and PCFB templates is to merge the LCD of one stage with the RCD of the other by adding

an additional request line to the channel. This is shown in Figure 3.4 for a PCHB template. The request line, at least from the channel point of view, may appear redundant but in fact it enables the removal of the input-completion detector, thereby saving area and reducing capacitance on the data lines. Moreover, the request line does not significantly impact performance and the template is still QDI. Therefore, the communication between stages remains delay-insensitive.

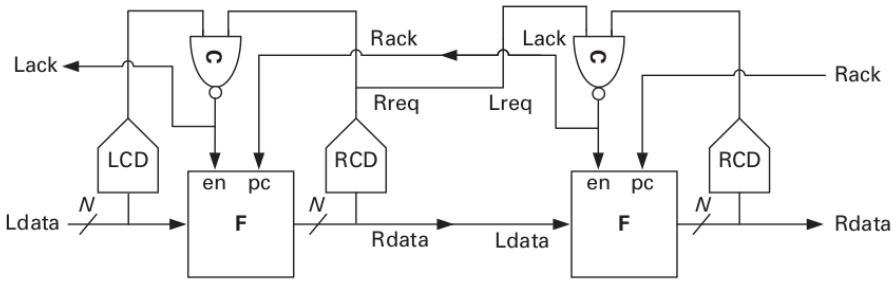


FIGURE 3.4: Optimized PCHB template (similar changes in PCFB) [1]

3.4 Reduced Stack Pre-Charge Half Buffer (RSPCHB) [1]

The motivation for a RSPCHB template is to eliminate the need for an enable signal "en" (or precharge signal "pc") from the domino function-block control, as seen in PCHB template. This enable signal is only needed to support concurrency that typically does not improve performance. RSPCHB eliminates internal "en" (pc) signal, thereby reducing the transistor stack sizes in the function block. This pipeline template is shown in Figure 3.5. Note that any one of the pc/en signal can be removed and logic can be modified accordingly. The impact of this change in RSPCHB is that the assertion and de-assertion of Rreq is delayed until after Lreq have been asserted and de-asserted respectively.

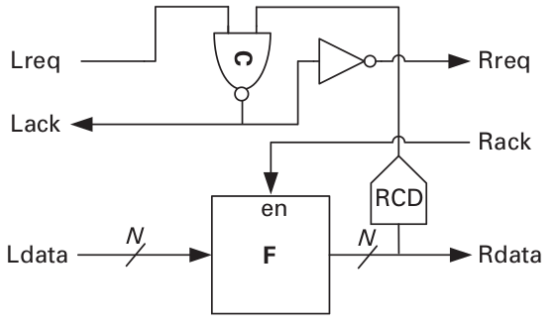


FIGURE 3.5: Block Diagram of RSPCHB template [1]

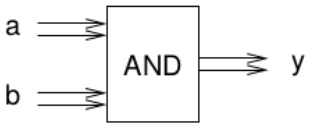
With these changes, RSPCHB is still QDI; however, the communications along the input channels to the joins become QDI instead of DI (other channels remain DI). The advantage of RSPCHB is that the lack of an LCD and the reduced stack size of the function block, which reduces the capacitive load, yielding significantly faster overall performance.

Similar to RSPCHB, which is a half buffer pipeline, there exist a full buffer pipeline template which is termed as Reduced Stack Pre-Charge Full Buffer

(RSPCFB). This pipeline template has its structure similar to RSPCHB. Additional hardware is required to make this pipeline full buffer, when compared to RSPCHB template. More inforamtion on this template can be found in [1].

3.5 Delay Insensitive Minterm Synthesis (DIMS) [11][4]

This section provides an approach for implementing functional blocks. This DIMS approach can be used with other pipelining templates like PCHB, WCHB to get a fully functional template. A technique to implement basic 4-phase dual rail AND gate is presented in Figure 3.6. Using the similar basic topology, it is possible to implement other simple gates such as OR, XOR, etc. As per dual-rail encoding, an inverter will just be a swap of two wires. Inverter converts logic '0' ("01" in dual rail encoding) into logic '1' ("10" in dual rail encoding), which means that in dual rail encoding, inverter can be implemented by swapping the true and false rails of data signal.



a	b	y.f	y.t
E	E	0	0
		NO CHANGE	
F	F	1	0
F	T	1	0
T	F	1	0
T	T	0	1

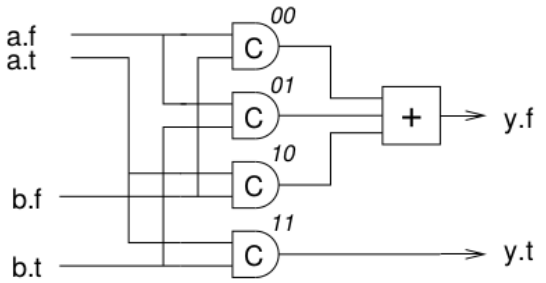


FIGURE 3.6: Symbol, Truth table and gate level implementation of dual rail AND gate [4]

Arbitrary functions can be implemented by combining gates in exactly the same way as when one designs combinatorial circuits for a synchronous circuit. The handshaking is implicitly taken care of and can be ignored when composing gates and implementing Boolean functions. This has the important implication that existing logic synthesis techniques and tools may be used, the only difference is that the basic gates are implemented differently. This approach of implementing Boolean expressions using dual-rail gates is termed as DIMS.

Usage of 4 C-elements and 1 3-input OR gate for a simple dual-rail AND gate (approx 56 transistors), which is 10 times more than conventional AND gate shows that this gate developing technique inevitably becomes area and power hungry. Larger functions can be implemented together which can reduce the overhead but the overall transistor count will still be very high. Implementation of DIMS complex gates to reduce hardware is still very hardware intensive. Hence, this approach of functional block implementation is generally avoided and approaches like Null Convention Logic (NCL) are considered (discussed later).

3.6 Sense Amplifier Half Buffer (SAHB) [11][12]

SAHB is a QDI cell design approach (not a pipeline template) which focuses on high operational robustness, high speed, and low energy. This cell design approach can be embedded with a suitable pipeline template to get a fully functional asynchronous QDI template. SAHB can be incorporated with IL type of pipelines like PCHB to get maximum performance with low power figures. Although, the cell design style promises robustness and low energy, the complexity of cell is very high and involves a lot of transistors. To give reader an idea, Figure 3.7 and Figure 3.8 presents a SAHB cell template and internals of a buffer cell.

FIGURE 3.7: SAHB cell template [12]

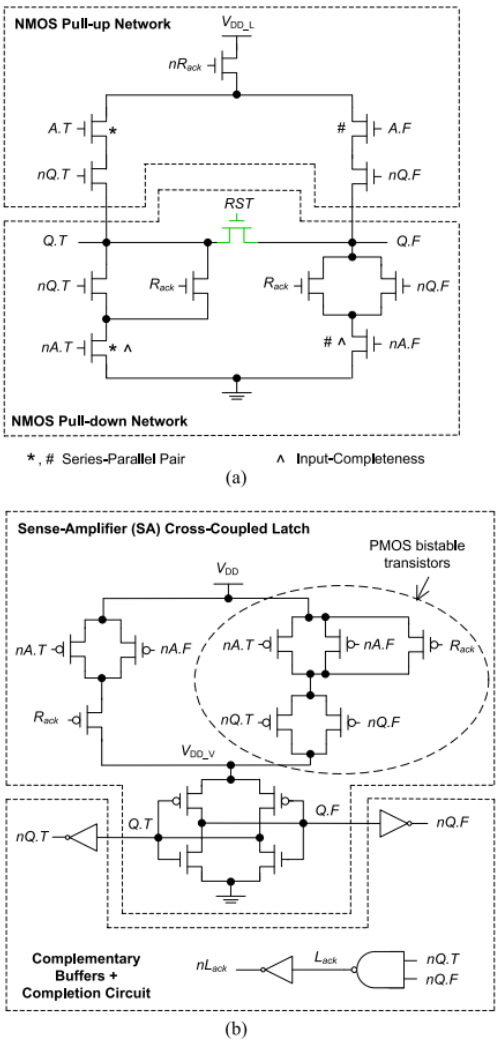


FIGURE 3.8: SAHB Buffer cell with evaluation and sense amplifier block [12]

From these figures, it can be observed that apart from normal signals, complementary input signals are also required for ensuring correct output generation and QDI operation. There are also several novel noteworthy features in SAHB. They are :

- SAHB cell incorporates two blocks : 1) An evaluation block, and, 2) A Sense Amplifier (SA).
- Evaluation block comprises of pull-up and pull-down network, both made up of NMOS transistors, for generating valid and null data output.
- The SA block embodies a cross-coupled latch with completion detection circuitry. This block also involves circuitry to generate complementary output signals.
- Both the pull-up and pull-down networks in the evaluation block comprise only of NMOS (instead of CMOS) transistors to reduce the parasitic capacitance and hence reducing power dissipation.
- Evaluation block and SA block are tightly coupled to reduce the number of switching nodes, resulting in short cycle time and low power dissipation.
- The SAHB cell is realized in static logic style and hence appropriate for Full-range Dynamic Voltage Scaling (FDVS) for supply voltages ranging from nominal voltage (1 V) to deep sub-threshold (0.3 V) regions.

Performance : On the basis of six library cells (i.e., buffer, two-input AND/NAND, two-input OR/NOR, two-input XOR/XNOR, two-input MUX/IMUX, and three input AO/AOI) at 1 V, 65nm CMOS process, the proposed SAHB approach outperforms the reported competing PCHB approach; *the PCHB library cells, on average, dissipate 2.8x more power, suffer 1.27x longer delay, and occupy 1.06x larger area than the SAHB library cells* [12].

Apart from certain advantages of SAHB approach, there are some disadvantages also. First disadvantage is that the SAHB cell requires dual power supply for operation. Laying dual power rails for individual cells is a complicated task. Second disadvantage is that the transistor count is still higher if compared with other implementations of buffer cells. Also, as seen in the DIMS approach, dual rail gates can lead to very high transistor count. So, it is better to have complex functions implemented at transistor level to reduce transistor count.

3.7 Null Convention Logic (NCL) [9][13]

This section does not mention a template, rather mentions a technique widely used for designing asynchronous circuits. NCL is a symbolically complete logic which expresses process (along with control information) completely in terms of the logic. It provides us with an easy way to conveniently express asynchronous digital circuits. Symbolically complete logic is complete in and of itself solely in terms of the relationships among input values in the logical expression [13]. It integrates the expression of data transformation and what is generally viewed as the expression of control in a single symbolically determined expression, without appealing to any extra-symbolic expression such as a clock or a controller. In simple terms, NCL avoids the usage of global clock but still provides necessary synchronization at

basic gate level.

The traditional form of Boolean logic is not symbolically complete, in the sense that it requires the participation of a fundamentally different form of expression, time in the form of the clock, which has to be very carefully coordinated with the logic part of the expression to completely and effectively express a process. To express a process completely, basic gates need to indicate about the validity and neutrality of the input data. In terms of NCL, this translates into a need for hysteresis mechanism at the gate level. More detail on this hysteresis mechanism will be presented in next chapter.

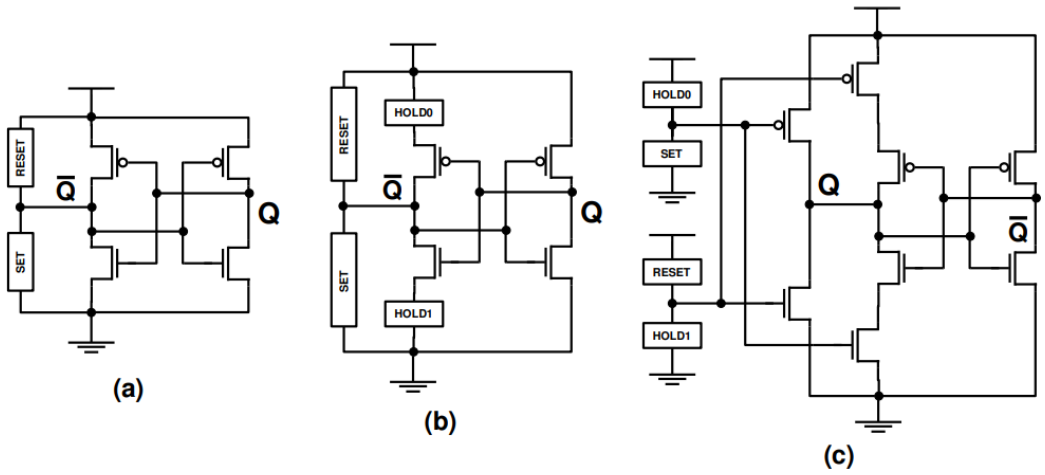


FIGURE 3.9: NCL Gate topologies, a) Martin, b) Sutherland, c) Moreira [13]

There are 27 fundamental gates [14] provided by the NCL. These gates provide the logical functionality along with hysteresis which helps NCL to achieve symbolic completeness.

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. All three topologies have their advantages and disadvantages. Martin's topology employs two main logic blocks (SET and RESET), and a pair of cross-coupled inverters: an output inverter and a weak-feedback one. The inverters are used to latch the output in case neither SET nor RESET logic blocks 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 control 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 NCL design even further to low-voltage scenarios. This topology proposes two main improvements:

lower driving capacitances for SET/HOLD0 and RESET/HOLD1 logic blocks and reduced short circuit currents while switching the output. Moreira's topology brings better speed, energy and leakage trade-offs.

3.7.1 NCL Basic Template

NCL pipeline stage has a structure similar to a WCHB stage, although the NCL template separates the latch logic from the combinational logic. For better understanding, Figure 3.10 illustrates the implementation of NCL pipeline stage with NCL register R_n, a completion detector CD_n, NCL combinational logic block F_n and two handshaking signals: ack(n) and ack(n + 1). Cells used in the register R_n are Muller C elements, whereas, the cell used in CD_n is a TH24 gate (refer [19] for more information) .Signal ack(n) indicates the previous stage of pipeline about providing new data or spacer, while signal ack(n+1) coming from next stage informs the current stage to provide new data or spacer. Using the atomic structures of basic NCL template, the pipeline is implemented as shown in Figure 3.11.

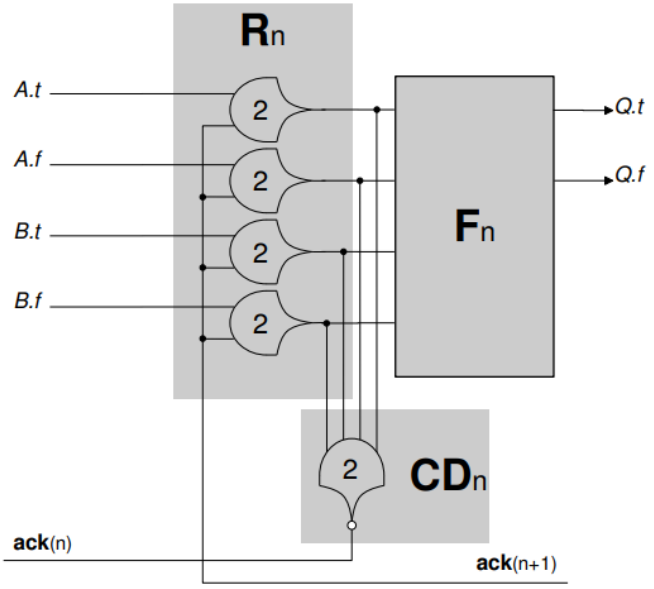


FIGURE 3.10: Basic NCL template [13]

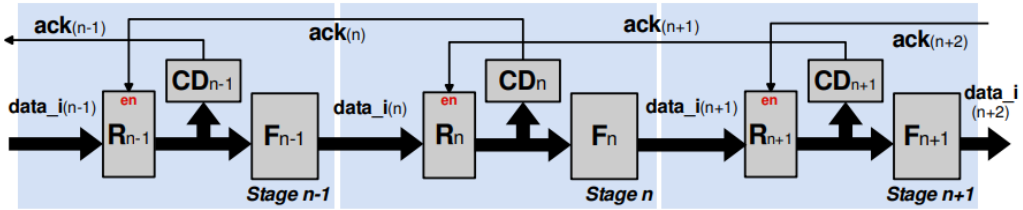


FIGURE 3.11: 3 stage pipeline using Basic NCL template [9]

The functional blocks present in the pipeline are implemented using the fundamental hysteresis gates governed by NCL. Note that these gates make the functional blocks symbolically complete.

3.7.2 Spatially Distributed Dual Spacer Null Convention Logic (SDDS-NCL) Template [15]

[15] presents a new QDI designing method which extends NCL asynchronous template. It also describes a custom design flow for the extended template which uses conventional Electronic Design Automation (EDA) tools to synthesize QDI circuits. Most synthesis tools rely on pattern based technology mapping i.e. synthesizing circuits as single rail version and then replace logic gates by NCL template to get a QDI version of circuit. This often prevents exploring optimizations enabled by logic sharing those specific to NCL. This template couples NCL with another family of logic, NCL+ [16]. This combination enables full potential of optimization for commercial EDA tools.

Note : This automated design flow is only meant for designing combinational circuits. Also, NCL approach can lead to less hardware expensive circuits when compared to DIMS.

Static Logic Implementation of NCL gates lead to large PMOS transistor stacking for complex gates which can degrade circuit performance and complicate transistor sizing.

3.8 Sleep Convention Logic (SCL) [9] [17]

SCL is an asynchronous design template inspired by NCL. Its structure presents two main characteristics: NCL with early completion and fine-grained Multi-Threshold CMOS (MT-CMOS) power-gating. In fact, SCL was initially labelled as Multi-Threshold Null Convention Logic (MT-NCL). Compared to NCL, SCL brings architectural and design modifications that may result in area and performance advantages. Figure 3.12 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.

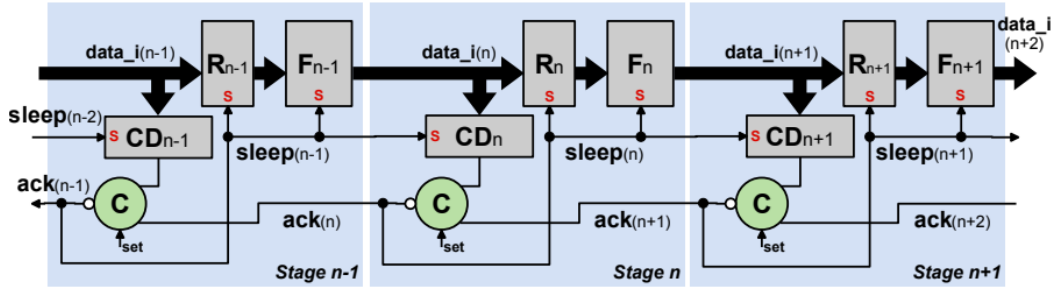


FIGURE 3.12: 3 stage SCL pipeline template [9]

The combinational logic blocks 'F' basically consist of SCL gates that couple a threshold function (refer NCL [13]) 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. The remaining logic blocks (HOLD1 and RESET) are replaced by the sleep logic, which is responsible for generating logic 0 at the output. This can be observed in Figure 3.13. The Boolean expression implemented in the figure is AB + AC + BC, which is the Boolean expression implemented by NCL TH23 gate.

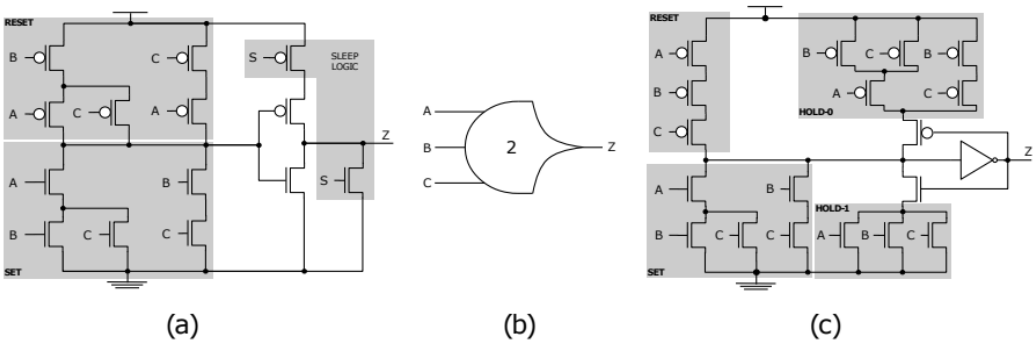


FIGURE 3.13: a) TH23 SCL gate, b) Symbol for both NCL and SCL gate, c) TH23 NCL gate [17]

Apart from less transistor count and sleeping mechanism to save power, there are a few disadvantages to this pipeline as well. The SCL template implements a longer forward latency because 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 functional block to compute and propagate new data. Also, the presence of sleep transistors (power gating transistors), makes this pipeline inherently slow when compared with other templates. The inverter which is placed at the output in CMOS logic is meant for providing buffering and a low resistance path to power rails. In the SCL cells, the inverter stage comprises of two PMOS transistors in series which increases the resistance of output path to power rails, thus slowing down the speed of gate.

The presence of sleep signal which is feeding each and every gate in a functional block leads to additional set of wires flowing in the design. This requires a lot of work at the physical layout level. These set of wires will switch with every cycle of valid data and null data. Operation at higher throughput will lead to higher switching activity and thus higher power consumption. Thus, this template will be well suited for low throughput applications.

3.9 Register Less Null Convention Logic (RL-NCL) [18]

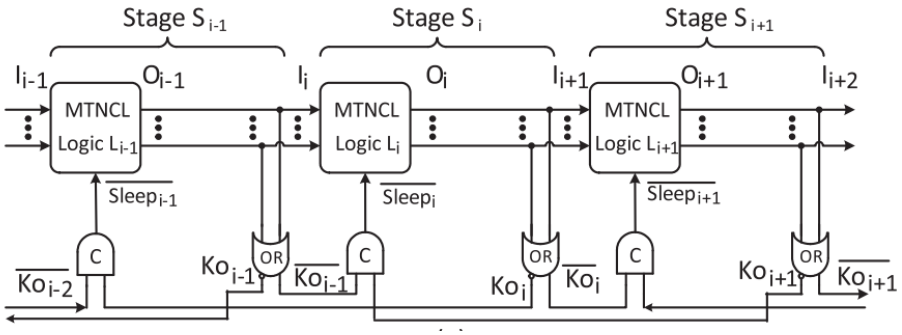


FIGURE 3.14: RL-NCL 3 stage pipeline [18]

The conventional NCL paradigm requires pipeline registers for separating two neighboring logic blocks, and those registers can account for up to 35% of the overall power consumption of the NCL circuit. RL-NCL design paradigm achieves

low power consumption by eliminating pipeline registers, simplifying the control circuit, and supporting fine-grained power gating (similar to SCL) to mitigate the leakage power of sleeping logic blocks. Figure 3.14 presents the 3 stage RL-NCL pipeline.

Absence of latches / registers in between surely saves power in the design. But presence of MT-NCL gates with sleep mechanism slows the pipeline down. As the author of [18] says : "The latency of a logic block is deteriorated by power gating, the MT-NCL implementation has a lower maximum sustainable throughput rate than the conventional NCL counterpart. But newer RL-NCL implementation can achieve the same maximum sustainable throughput rate as the conventional NCL counterpart by replacing completion detectors with much faster OR gates to countervail the latency deterioration due to power gating".

This usage of OR gate is based on the assumption that all primary inputs are available at the same time and some paths in functional block are longer and some are shorter, and, placing OR gate in the longest path serves as completion detector. But this is not always the case and this assumption fails when multiple paths have equivalent delays or inputs arrive at different times. This scenario did not seem to be considered by the authors during the timing analysis for OR gate placement. In those scenarios, we need completion detector circuits with Muller C elements. Also, an elaborate pre and post layout timing analysis is required to ensure the correct functionality of the design.

3.10 Comparative Study

WCHB has a short cycle time when compared to other QDI templates, although its implementation complexity is typically high and it is not feasible to implement complex functions with a single level of weak-conditioned logic. Building complex functions in the buffer leads to big stacks of PMOS and NMOS transistors, which impacts circuit performance significantly.

PCHB template presents better performance, because it utilizes domino logic, avoiding long chains of series PMOS transistors, but increases implementation complexity. 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.

A comparative study between variants of PCHB (PCFB, RSPCHB and RSPCFB) was carried out in [1]. For the half buffer pipeline, i.e. the PCHB and the RSPCHB, a linear dual-rail pipeline of buffers with 60 stages was constructed to achieve a static slack (number of valid data token present in the pipeline) of 30. For the full buffers, i.e. the PCFB and the RSPCFB, 30 stages were used to achieve the same static slack. HSPICE simulations were performed using 0.25um Taiwan Semiconductor Manufacturing Company (TSMC) process technology with a 2.5 V power supply at 25 degree C . The PCHB achieves maximum throughput of 772 MHz, whereas, the RSPCHB is faster with maximum throughput of 920 MHz. The throughput improvement is approximately 20%. For the full buffers, the PCFB achieves maximum throughput of 707 MHz. The RSPCHB is faster, with maximum

throughput of 1000 MHz. These figures mean that the reduced-stack templates are faster. The disadvantage lies in their higher transistor count. Also due to the presence of domino logic, the operation in sub-threshold region is somewhat questionable.

NCL is a good candidate for sub-threshold operation. It is possible to adopt the static gate 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 complicate transistor sizing. In the same context, SDDS-NCL utilizes the same gate implementations, along with the addition of NCL+ gates. 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. Moreover, optimizations provided by SDDS-NCL are compatible with conventional EDA tools. NCL as well as SDDS-NCL leads to a very high transistor count due to the presence of gate level hysteresis mechanism.

The SAHB cell design approach shows compatibility with low-voltage operations, due to the use of static logic. SAHB uses a fine-grained pipeline structure to maximize throughput. When compared to PCHB, SAHB presents performance and energy improvements, although its area consumption is 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.

The SCL template is also a possible alternative for circuits operating in the sub-threshold region. Its logic optimization significantly reduces area requirements. Also, sleep logic reduces leakage power, increasing its applicability in ultra-low power domains. Forward latency is significantly higher than that of other templates. The larger forward latency could impact performance, specially in sub-threshold operation. SCL works on additional timing assumptions which are to be verified during physical designing. SCL template can in principle be combined with SDDS-NCL, which can reduce even further leakage power consumption and increase energy efficiency.

Regarding RL-NCL approach, 50% - 60% power reductions are observed due to the removal of registers. This approach is better than NCL template, as per the power numbers. The design generates these power numbers using Predictive Technology Model (PTM). But the power gating transistors make this pipeline slow and sizing those power gating transistors is an additional effort in the designing process. There are requirements on the arrival of input data and there are some timing assumptions made which are to be verified at the physical layout level.

3.11 Conclusion

Multiple templates were considered and it was observed that every approach has its own advantages and disadvantages. Some of the templates discussed have made certain timing assumptions on the arrival of input data. These templates also involve performing timing analysis to detect the longest path in the functional block. All these assumptions and timing analysis slowly drift these templates away

from the true definition of asynchronous circuits and QDI delay model. The random arrival of inputs and minimal timing analysis in terms of isochronic forks should be satisfied by a asynchronous QDI template.

We need to have such a QDI design template which can run at full speed at a given voltage with least number of transistors and low power figures. Considering this, sleep logic and power gating templates will not be considered for final design. It can be concluded that a hybrid (or heterogeneous) implementation utilizing the aforementioned templates and cell topologies will be investigated in this thesis.

Chapter 4

Functional Blocks : Hysteresis and Indicatability

Synchronous circuits are designed with single-rail encoding logic (1 wire per bit of information) and require clock signals to control the flow of data across registers. Clock signal decides the time instances at which the output of combinational logic will be read and stored in registers. To decide these time instances, a detailed timing analysis is carried out on the synchronous design. This timing analysis is performed because the single-rail encoding logic is not capable of indicating the validity and neutrality of data. This timing analysis ensures that circuit behavior is verified under worst case scenarios, and, given a range of operating conditions, the synchronous circuit will work for a range of pre-determined clock frequencies.

Asynchronous circuits based on the DI and QDI delay model do not assume any timing information about the gates and wires except a few isochronic forks. Since, there is no assumption on the timing, it is required by the asynchronous design to function correctly under all operating conditions. This correct functionality requirement translates into another requirement, according to which, the design should indicate about its completion (validity and neutrality) of input and output data. Completion detection by observing input and output data is not possible with single-rail encoding, that is why, dual-rail encoding is chosen. Other encoding schemes can also be considered which are capable of indicating the validity and neutrality of data.

The indication of validity and neutrality of data classifies the asynchronous functional blocks into two categories : Strongly-Indicating and Weakly-Indicating. These terms have different meaning for basic dual rail gates and combinational logic comprising of these dual rail gates. *When combinational logic is considered, this indication property is required at the inputs, outputs, and intermediate nodes. Indication at the intermediate nodes necessitates the usage of hysteresis mechanism which will be explained in the upcoming sections.* Also, the type of functional blocks and indicatability used affect the complexity of design as well as the pipeline template. If certain timing assumptions are made, an opportunity to optimize the functional blocks and template opens up which reduces the power consumption and area. In some scenarios, it is also possible to remove the hysteresis behavior which is necessary to indicate data at intermediate nodes.

4.1 Hysteresis at Gate Level

Hysteresis is a mechanism present at the gate level which helps a logic gate to hold its output status (valid or null) till the desired inputs are received to change that status. This mechanism is one of the key feature of strongly-indicating functional blocks, which will be discussed shortly. To understand how the hysteresis mechanism works, Figure 4.1 presents a flow which explains the working of a typical asynchronous dual-rail gate in a 4-phase signaling system. The dual rail gate can implement any arbitrary Boolean expression.

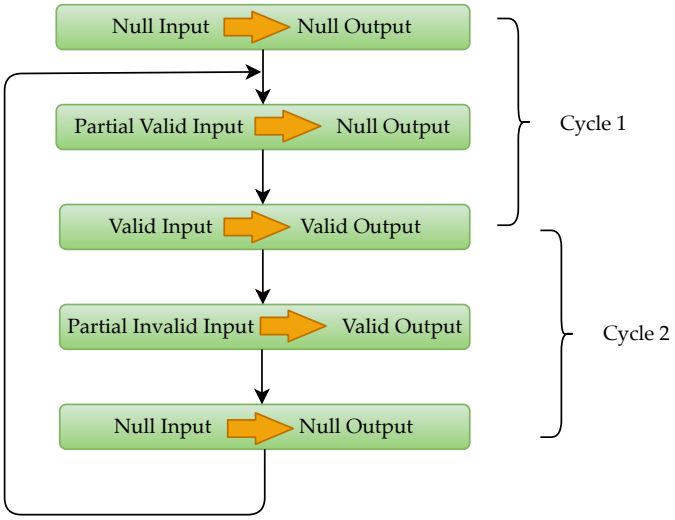


FIGURE 4.1: Steps involved in hysteresis for 4-phase signaling system

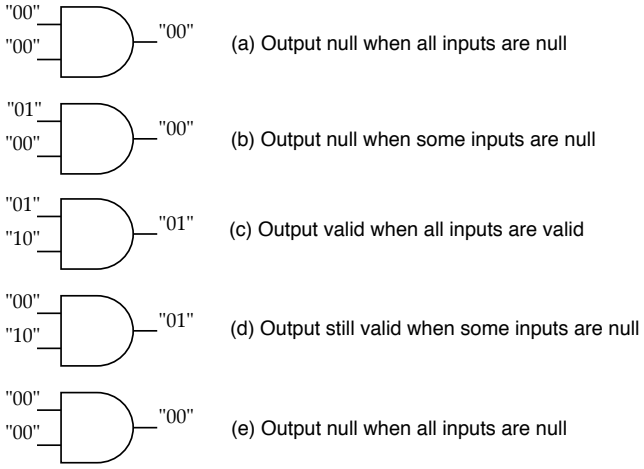


FIGURE 4.2: Hysteresis in dual rail AND gate

In cycle 1, when all inputs to a gate are null, the output produced is also null. The logic gate keeps on producing null value until all the inputs become valid. In cycle 2, when a valid output is produced after receiving all valid inputs, the logic gate holds the output till all of its input have become null. When all the inputs go to null state, the logic gate produces null output. Having different output for partial set of valid/null inputs based on the different cycles of 4-phase signaling is termed as hysteresis. This hysteresis mechanism is also explained with an example of dual

rail AND gate in Figure 4.2. Dual rail encoding has its usual meaning as presented earlier in Table 2.1.

4.2 Strongly-Indicating vs. Weakly-Indicating

As discussed earlier, the indicatability requirements in an asynchronous design have divided the functional blocks into two categories. There are different definitions presented by different authors for both strongly and weakly indicating type of logic implementation. Also, the interpretation of these definitions change when the indicatability is observed at gate level and at combinational logic level.

Considering gate level strongly-indicating type of logic first, we can say that a basic dual rail gate should produce a valid output when all of its input are valid, and, a null output when all of its input are null. For remaining input patterns, the logic gate should hold the previous state of output. In simple terms, inputs to a gate should be of same status to change the output to that status. This definition implies a need of hysteresis mechanism at the gate level, which we had already discussed earlier.

To understand more about this gate level strongly-indicating logic, an example of dual rail AND gate is presented. Truth table which represents a dual rail AND gate with hysteresis mechanism is presented in Table 4.1. The working principle of this strong-indicating dual rail AND gate is already presented in Figure 4.2.

A (A_t A_f)	**B** (B_t B_f)	**Output (True-rail False-rail)**
00	00	00
00	01	Hold Data
00	10	Hold Data
00	11	Pattern not allowed
01	00	Hold Data
01	01	01
01	10	01
01	11	Pattern not allowed
10	00	Hold Data
10	01	01
10	10	10
10	11	Pattern not allowed
11	XX	Pattern not allowed

TABLE 4.1: Dual rail AND gate truth table with hysteresis

It should be noted that input pattern "11" is invalid but it can be taken into consideration for simplifying the Boolean expressions. It is safe to assume that simplified Boolean expressions will implement the required functionality provided that the input pattern "11" is never encountered within the functional block. The Boolean expressions for true and false rail of AND gate output excluding hold patterns are presented below. These expressions suggest pure combinational logic, and if hold patterns are to be included, then hysteresis mechanism is required.

$$\text{True rail} = A_t.B_t$$

$$\text{False rail} = A_f.B_f + A_f.B_t + A_t.B_f$$

These expressions make sure that each output rail is computed when all the signals are available and valid. Expression for true rail is a single rail AND gate, but the false rail implements a complicated function. It should be noted that these expressions are to be implemented with hysteresis mechanism. Multiple ways of implementing hysteresis mechanism are already presented in NCL [13]. If fundamental NCL gates [19] are considered, then, the true rail can be implemented with so called TH22 threshold gate and the false rail with THand0 threshold gate.

At a combinational logic level, the definition of strongly-indicating logic stays the same but it has certain implications on the complexity of hardware involved. For a multi-input multi-output functional block, as shown in Figure 4.3, none of the outputs should change their status until all the inputs have reached the same status. As per [4], a functional block is said to be strongly-indicating if it waits for all the inputs to go to valid state before producing valid output, and, waits for all the inputs to go to null state before producing null output. Ensuring this requires a lot of additional circuitry and, thus, becomes a hardware-intensive approach to design functional blocks.

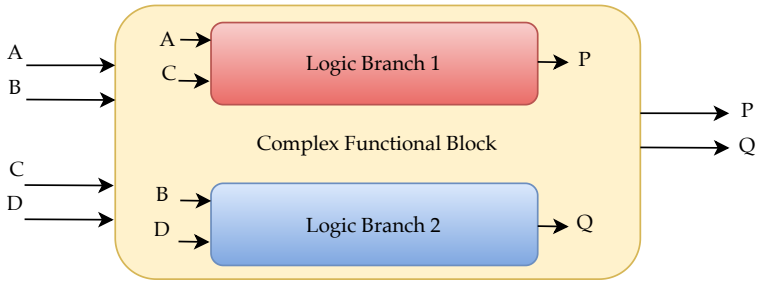


FIGURE 4.3: Complex functional block with isolated logic branches

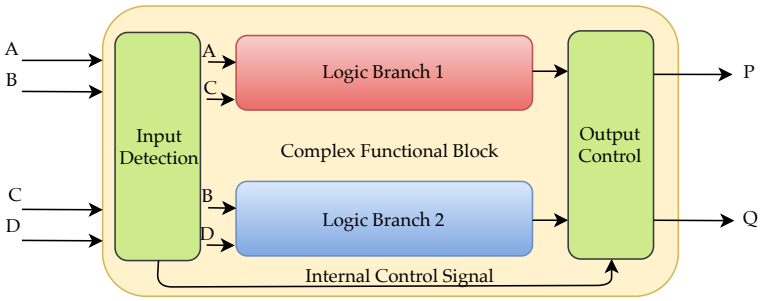


FIGURE 4.4: Additional circuitry to ensure strong-indicating logic in a combinational block

To understand the complexity and hardware intensity, an example of a functional block (see Figure 4.3) which has two isolated combinational logic branches will be taken. These isolated branches are working on different set of inputs. If one set of input changes to null, one logic branch will start producing null data, irrespective of input data on the other branch. This behavior of functional block does not conform with the definition of strongly-indicating type of logic and necessitates additional circuitry in the form of an "Input Detection" and "Output Control" components as presented in Figure 4.4.

"Input Detection" block checks when all the inputs are valid/null and accordingly indicates "Output Control" block using "Internal Control Signal" to produce complete valid/null data. The task of producing all valid/null values at the same time is given to "Output control" circuit. All these additional components are mentioned in detail in Chapter 5. This type of logic also requires intermediate nodes to indicate about the inputs. If all the intermediate nodes are indicating, then changes on output port means definite changes on all the input ports. This indicatability at intermediate nodes ensures that circuit is not going to outrun a slowly changing signal to produce valid/null output. It means that the functional block will not produce complete valid/null output if its slow intermediate nodes are still transitioning. And, to ensure the indicatability, this type of combinational logic should be made out of strongly-indicating gates which implement hysteresis mechanism. Presence of hysteresis mechanism, input detection and output control blocks make this approach hardware intensive.

Considering the definition of weakly-indicating logic [4] at gate level, valid output can be produced when some of the input signals are valid, and similarly, null output can be produced when some of the input signals are null. Now, the output signal of a gate is no longer able to indicate when all the input data is valid/null. Based on this definition, optimizations can be made in the Boolean expression which are meant for cycle 1 of 4-phase signaling. Since, the logic gate has already lost indicatability, there is no need to have indicatability in cycle 2 also. This means that hysteresis mechanism can also be removed at the gate level. This optimization comes at the cost of timing analysis at the design level to ensure that circuit behaves correctly. This timing analysis will be discussed in Section 4.4.

A (dual-rail)	B (dual-rail)	Output (dual-rail)
00	00	00
00	01	01
00	10	00
00	11	Pattern not allowed
01	00	01
01	01	01
01	10	01
01	11	Pattern not allowed
10	00	00
10	01	01
10	10	10
10	11	Pattern not allowed
11	XX	Pattern not allowed

TABLE 4.2: Dual rail AND gate truth table for weakly-indicating logic

To understand more about this gate level weakly-indicating logic, the same example of dual rail AND gate is presented. The truth table which represents a dual rail AND gate without hysteresis mechanism is presented in Table 4.2. For a 2-input single-rail AND gate, if one of the input is at logic '0', the output is always logic '0' irrespective of the second input. For the output to become logic '1', both inputs should be at logic '1'. Using this "single-rail" theory, the truth table presented in Table 4.2 is designed. For dual rail AND gate, the Boolean expressions obtained for weakly-indicating scenario after considering invalid input pattern "11"

for optimizations are presented below:

$$\text{True rail} = A_t.B_t$$

$$\text{False rail} = A_f + B_f$$

Simplified expressions obtained here indicate that dual rail expressions can be simplified to a greater extent with the aid of weakly-indicating logic. This leads to a faster implementation of cells with relatively lower transistor count. A similar "single-rail" theory can also be applied while designing other dual rail gates like OR, NAND, etc. It should be noted that this optimization is not always possible. When considering the example of dual rail XOR gate, the Boolean expressions obtained for both strongly and weakly indicating logic are same, offering no possibilities of Boolean expression simplification.

In this weakly-indicating logic at gate level, there is no indicatability mechanism which means that if these gates are to be used for asynchronous circuit design, additional circuitry will be required which will ensure that all valid/null outputs are passed when all inputs are valid/null. This additional circuit is similar to the "Input Detection" and "Output Control" blocks presented earlier. This arrangement for weakly-indicating dual rail AND gate is presented in Figure 4.5. This arrangement comes with an implicit timing assumption according to which the best case time required by input detection circuit should be sufficient enough for the actual logic block to process the latest data. If this timing assumption is not met, then there will be floating data in the functional block. More information on this timing assumption and floating data is presented with an example in Section 4.4.

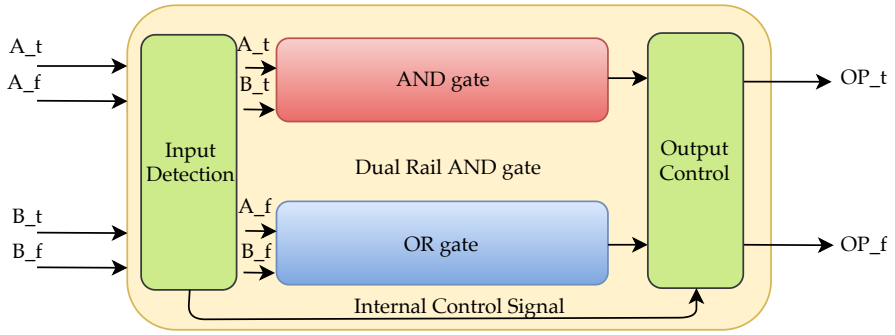


FIGURE 4.5: Dual rail AND gate with weakly-indicating type of logic

Finally, if this weakly-indicating logic is considered at combinational level, then as per [4] for multi-input multi-output functional block, some of the outputs can turn valid/null based on a subset of valid/null input signals. Some of the output signals need not wait for all the input signals to go valid/null completely. For this type of combinational logic to work correctly, it is required that a functional block never produces all valid outputs with partial valid inputs. There should be atleast one output which indicates about the status of all the inputs.

This type of weaky-indicating logic can be designed with strongly-indicating gates (with hysteresis mechanism), or, with weakly-indicating gates (without hysteresis mechanism). With strongly-indicating gates, intermediate nodes will be able to indicate about their input's status. This will make the design similar to strongly-indicating combinational design style with a simplification in terms of

partial valid or null output generation. If weakly-indicating gates are used to design this logic, then additional circuitry is required to assert complete valid/null status of inputs and to control the outputs, as presented earlier. Apart from this additional circuitry, as per [20], a Datapath Completion Detector (DPCD) is also required for correct functioning of the design. DPCD is used to keep a check of valid and null data on the internal nodes of a functional block and assert indicatability for them. This DPCD circuitry will be explained in detail in Chapter 5. A step-by-step guide to make a choice of indicatability based on the granularity required in the designing of functional blocks is presented below.

Step 1 : Determine the complexity of the functional blocks present in between the pipeline stages. If the functional block is just a single stage of gates, go to Step 2a. If the functional block comprises of combinational logic made up of multiple gates, go to Step 2b.

Step 2a : Single stage functional blocks can be made out from strongly-indicating as well as weakly-indicating gates. In the case of weakly-indicating gates, additional hardware is required in terms of input completion detector and the output control circuit for correct data and null propogation. Exit the guide.

Step 2b : Functional block comprising of combinational logic can be divided into two categories : strongly-indicating and weakly-indicating functional blocks. For strongly-indicating functional blocks, go to Step 3, and for weakly-indicating functional blocks, go to Step 4.

Step 3 : Strongly-indicating combinational functional block can only be made out of strongly-indicating gates with hysteresis mechanism. To make sure that all outputs turn valid/null at the same time based on the status of input signals, additional hardware is required in terms of input completion detector and the output control circuit. Exit the guide.

Step 4 : Weakly-indicating combinational functional blocks can be made out of both strongly as well as weakly-indicating gates. For weakly-indicating functional block with strongly-indicating gates, go to Step 5. For weakly-indicating functional block with weakly-indicating gates, go to Step 6.

Step 5 : When strongly-indicating gates are used for designing weakly-indicating combinational functional block, all the internal nodes have the ability to assert indicatability. No additional circuitry is involved apart from registers and completion detectors to develop a design based on these functional blocks. Exit the guide.

Step 6 : When weakly-indicating gates are used to design weakly-indicating combinational functional block, additional hardware is involved like DPCD to assert indicatability for intermediate nodes. Apart from DPCD, input completion detectors are also used for correct propogation of data and null values across the pipeline. Exit the guide.

4.3 Usage and need of hysteresis mechanism

Hysteresis is an important mechanism involved in asynchronous circuits. Implementing this mechanism at transistor level involves additional transistors. These additional transistors count up to 1x-1.5x of transistors required for logic implementation. These additional transistors increase capacitance at the internal nodes of a gate and thus slow down the speed of operation. To increase the speed and compensate for the increased capacitance, width of the transistors is increased. This increased count of transistors and up-sizing the width leads to more power consumption. Although there are certain difficulties and disadvantages involved, hysteresis still is very important and provides us with a simpler way to design asynchronous circuits.

Hysteresis is helpful in establishing indicatability for input and intermediate nodes. To understand more about this, a weakly-indicating functional block comprising of strongly-indicating (hysteresis) gates is considered as an example, as presented in Figure 4.6. All the gates present in the functional block have a latching mechanism to hold the output data. At the start of operation (t = t_0), all the input, output and intermediate nodes of the functional block are filled with null data ('N'). Output node "Z2" indicates about the status of all the inputs, as required by weakly-indicating logic.

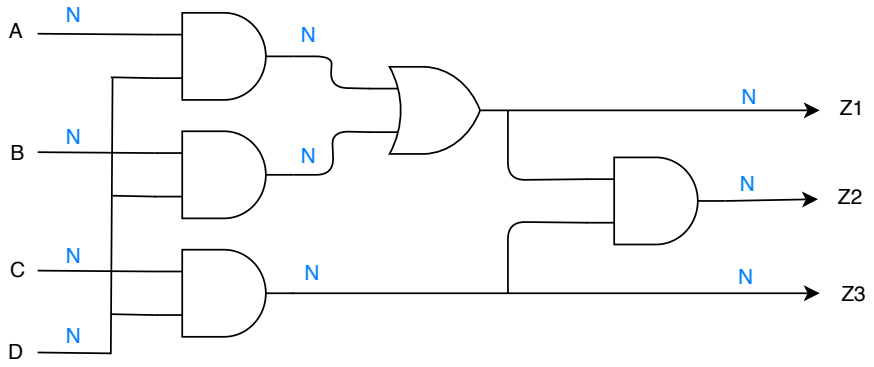


FIGURE 4.6: Weakly-indicating functional block with all nodes set to null (t = t_0)

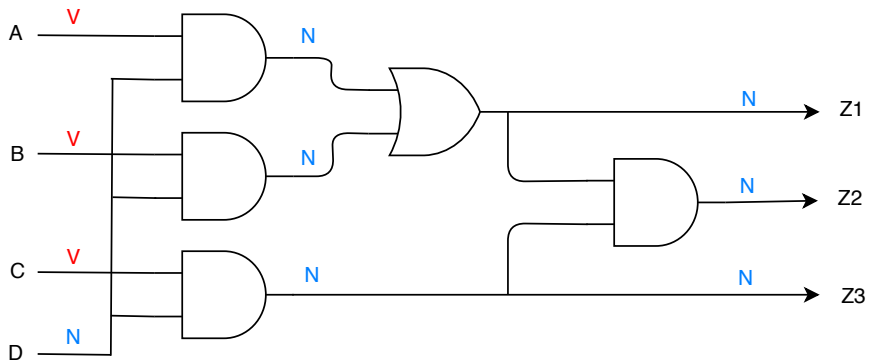


FIGURE 4.7: Partial valid inputs provided to functional block at t = t_1

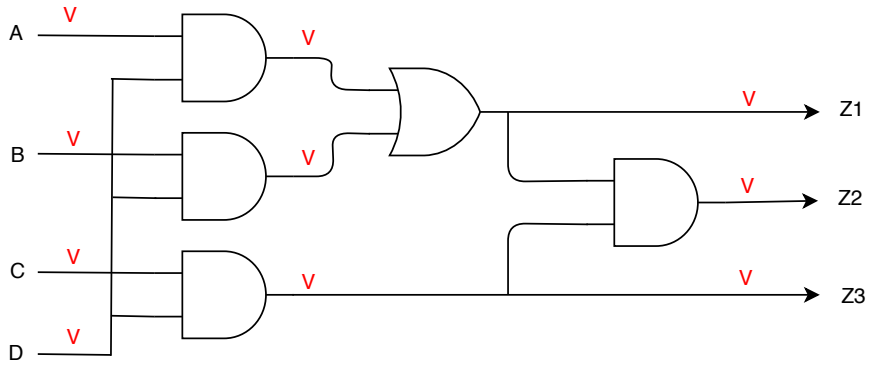


FIGURE 4.8: Valid output generation at t = t_2

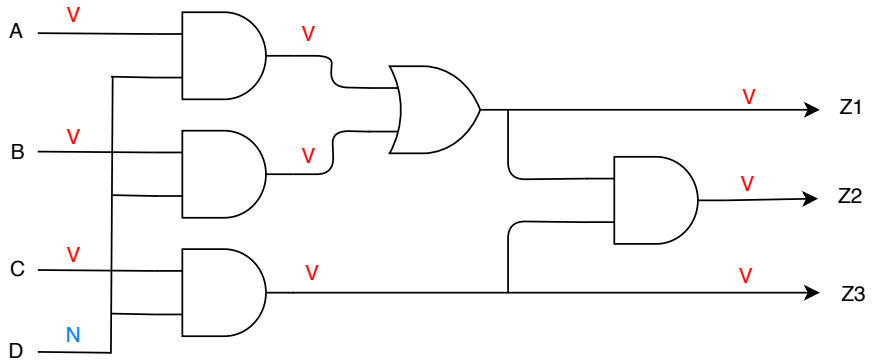


FIGURE 4.9: Partial null inputs provided to functional block at t = t_3

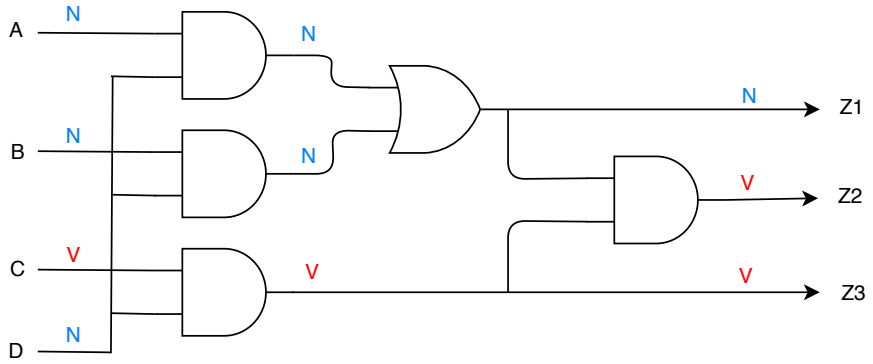


FIGURE 4.10: Partial null inputs provided to functional block at t = t_4

Since the functional block is meant to work asynchronously, the primary inputs can be provided at different time instants. Let us assume that input 'A', 'B' and 'C' turn valid ('V') but input 'D' is still null at time instant 't1'. This is presented in Figure 4.7. Since input 'D' is connected to all the AND gates, the intermediate nodes do not change their status and no valid output is produced.

At t = t_2, as shown in Figure 4.8, all the inputs and intermediate nodes switch to valid state and functional block starts producing valid output. The operation of functional block between time t_0 to t_2 corresponds to the cycle 1 of 4-phase signaling

logic.

In cycle 2 of 4-phase signaling, inputs will go to null state. Since there is no timing assumption on the inputs, their change of state can be asynchronous. At time $t = t_3$, input 'D' turns null when other inputs are still valid. This state of operation is presented in Figure 4.9. Due to the presence of hysteresis mechanism, intermediate nodes will not turn null until all the inputs to the AND gates are null.

At $t = t_4$, as shown in Figure 4.10, input 'A' and 'B' goes to null state along with input 'D' which forces some of the intermediate nodes to go to null state. This also leads output 'Z1' to go to null state. In the given state of operation, partial set of outputs are valid when partial set of inputs are null which conforms with the weakly-indicating logic (at combinational logic level), as discussed earlier. When input 'C' also goes to null state, all the intermediate and output nodes go to null state. This configuration will be similar to Figure 4.6. After this state, new set of inputs can be provided to the functional block. All these figures show that hysteresis is very important in establishing indicatability property for functional blocks.

4.4 Optimizations and Timing Assumptions

Hysteresis ensures that correct null and valid values are propagated across the functional block. It also ensures that a functional block does not outrun a slowly changing signal to produce outputs. This makes asynchronous circuit designing simpler but comes at the cost of increased number of transistors and power consumption. This section will focus on removing the hysteresis mechanism and analyzing the effects caused by hysteresis removal.

To analyze the effects of hysteresis removal, the functional block which is presented in Figure 4.6 having all the characteristics of weakly-indicating logic is taken as an example. *The only change made is that these strong indicating gates do not have hysteresis mechanism in them.* This means that the dual rail gates used in the design, follow Boolean expressions derived for strong-indicating scenarios. Without hysteresis, a dual rail gate will produce valid output when all inputs are valid, but will go back to null state when any one of the input goes to null state. This hysteresis removal will have no effect in cycle 1 but it will affect cycle 2 of 4-phase signaling logic.

When partially valid inputs are provided to the functional block, intermediate nodes and output nodes do not change their status and stay at null state, as shown earlier in Figure 4.7. When all valid inputs are available, intermediate nodes and output nodes go to valid state, as presented earlier in Figure 4.8. This behavior in cycle 1 stays similar to the case where hysteresis was present. In cycle 2, when some of the inputs go to null state (assume that input 'D' goes to null state), all intermediate and output nodes go to null state, as shown in Figure 4.7. All the outputs are at null state while some of the inputs are still valid. Output node "Z2" is no longer able to indicate about the inputs. When this functional block will be used in an asynchronous pipeline, subsequent stages will ask for new set of valid data after receiving null data as their inputs. This shows that subsequent stages can finish their operation while the current stage is still operating on its previous inputs. This might lead the asynchronous pipeline into a deadlock state, if suitable

arrangements are not made. These arrangements are made by modifying the pipeline structure or by adding extra control hardware.

To avoid this deadlock state, continuous monitoring of input signals is required. Output signals of a functional block should change their status to a certain state only if the inputs belong to the same state. Also, instead of having hysteresis mechanism at the gate level, a register which is made of muller C elements can be placed at the output of functional block to mimic similar hysteresis behavior. This register will store new valid/null data when all inputs to the functional block are at same status and next stage is ready to accept new valid/null data. This arrangement of registers and functional blocks is presented in Figure 4.11. "Input Detection" and "Output Detection" circuitry are used to monitor the status of inputs and outputs of a functional block. "Latching circuitry 1" and "Latching circuitry 2" involve arrangement of Muller C elements which help registers "REG1" and "REG2" in passing/holding valid/null value for the next stage. These additional elements present in the design are discussed in detail in upcoming chapters.

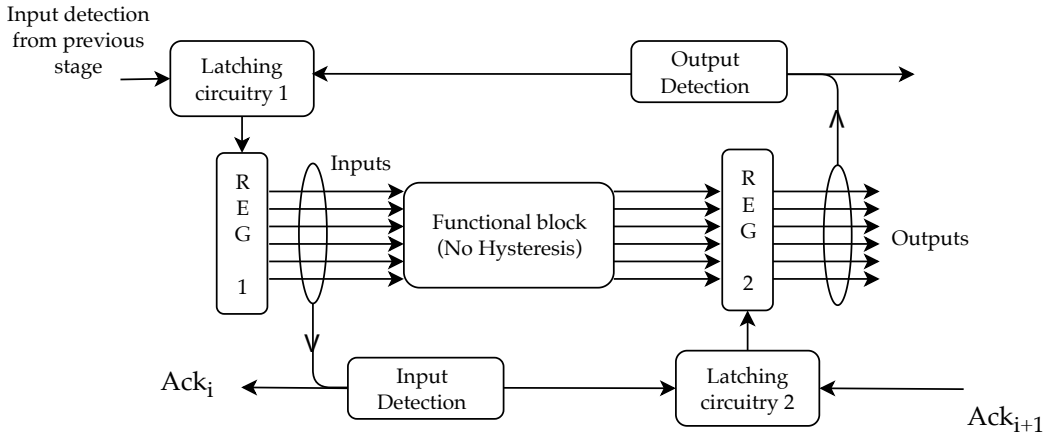


FIGURE 4.11: Additional circuitry present in absence of hysteresis mechanism for strongly-indicating gates

This solution for correct operation of asynchronous designs presented above comes with an implicit timing assumption. When the assumption is not satisfied, it leads to an incorrect output generation and might end up deadlocking the pipeline. For analyzing this timing assumption, the wire forks present between functional block and input detection circuit are considered to be isochronic, see Figure 4.11.

When input arrival to the functional block through "REG1" is asynchronous, the processing time of input detection circuit is overlapped with the processing time of functional block caused by the latest arriving signal. If the functional block is made up of strongly-indicating gates (without hysteresis), then in cycle 1, the functional block will wait for its input and intermediate nodes to go to valid state before producing valid outputs. If a multi-input multi-output functional block is considered, partial valid outputs can be produced with partial valid inputs, as shown in Figure 4.12.

After cycle 1, null data will be provided asynchronously to the functional block. Let us assume that the functional block is similar to the circuit (without hysteresis mechanism) presented in Figure 4.6. If one of the inputs to the functional block

(input 'D') goes to zero, all outputs go to null state which is presented in Figure 4.13. This null data will not be propagated further until the input detection circuit indicates that all inputs are null.

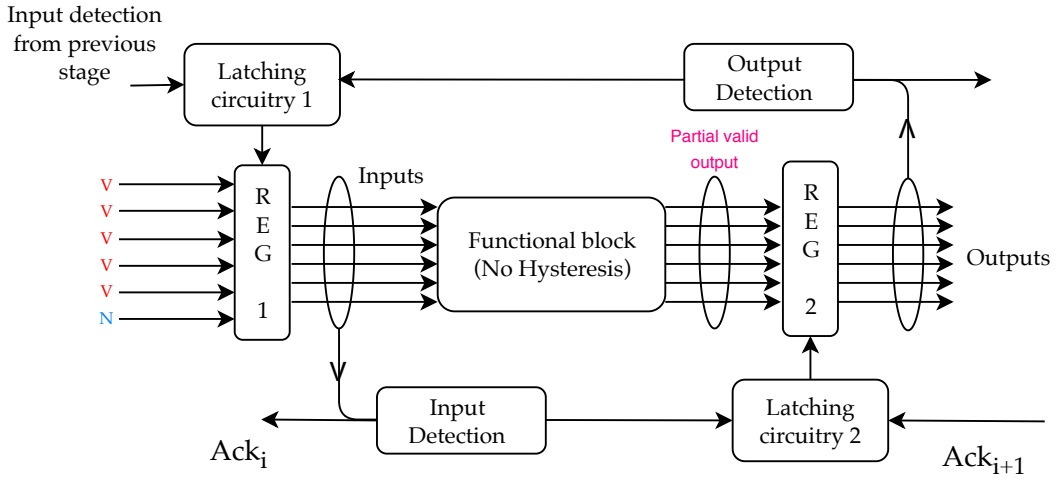


FIGURE 4.12: Pipeline behavior for valid data when input arrival is asynchronous

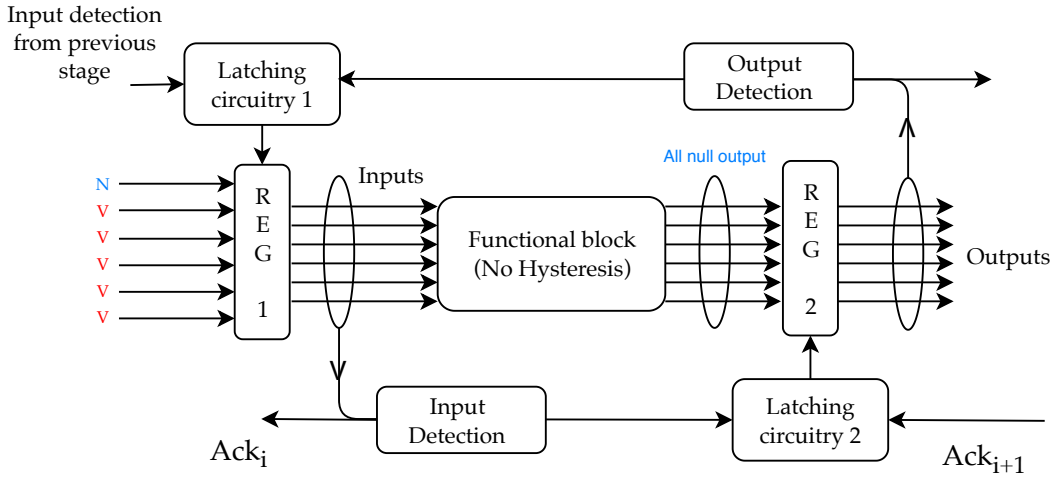


FIGURE 4.13: Pipeline behavior for partial null data when input arrival is asynchronous

The scenarios presented in Figure 4.12 and Figure 4.13 depict the general cases involving the functionality of pipeline under asynchronous arrival of inputs. To understand the effects of hysteresis removal, the overlap between the timing windows of functional block and input detection circuit should be studied. Lets assume that the time required for functional block to produce output and settle all its intermediate nodes is t_{FB}, and, the time required for input detection circuit to indicate about the inputs is t_{ID}. Based on the delay of the functional block and input detection circuits, there are two scenarios possible. They are :

- Delay of input detection circuit is greater than the delay of functional block, i.e., $t_{ID} > t_{FB}$.
- Delay of functional block is greater than the delay of input detection circuit, i.e., $t_{FB} > t_{ID}$.

In scenario 1, when functional block is faster, all the intermediate and output nodes will get settled till the latching circuitry acts on slow arriving signal from input detection circuitry. This slow input detection circuitry provides enough time frame for functional block to get settled internally. If this timing analysis is considered from the point of view of input signals, the time between the arrival of two data tokens (either valid after null or null after valid) should be sufficient enough for all internal signals of functional block to settle down. Mathematically this equates to :

$$t_{FB} < t_{settle}\ , \quad \text{where}$$

$$t_{settle} = t_{ID} + t_{Lat\_Ckt\_2} + t_{REG\_2} + t_{OD} + t_{Lat\_Ckt\_1} + t_{REG\_1}$$

In these equations, "t_{settle}" corresponds to time frame available for settlement of signals in functional block, "$t_{Lat\_Ckt\_1}$" and "$t_{Lat\_Ckt\_2}$" correspond to the delay of latching circuitry, "t_{REG_1}" and "t_{REG_2}" correspond to the delay of register and "t_{OD}" corresponds to the delay of output detection circuit. It should be noted that these timing values are worst case timing delays of each macro block. This delay path which is used to derive this equation is presented with red arrows in Figure 4.14. While calculating t_{settle}, it is assumed that other signals arriving from subsequent and previous stages are always available.

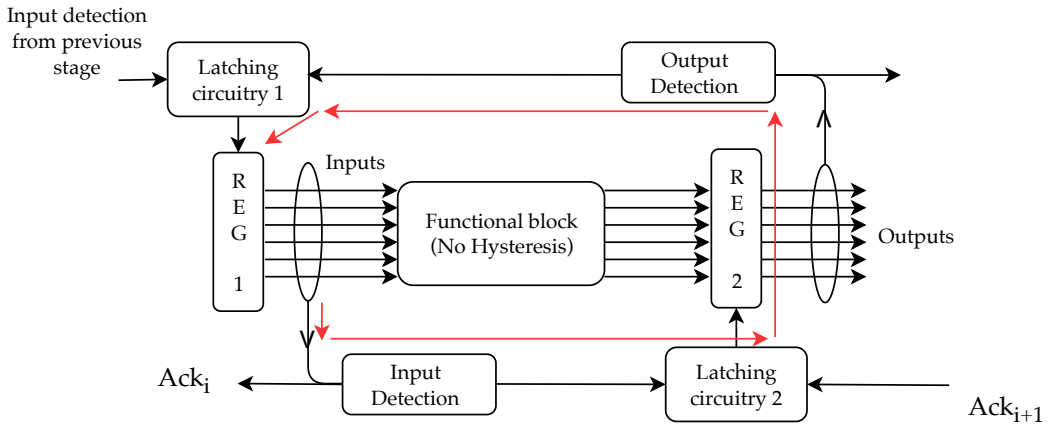


FIGURE 4.14: Delay path providing time frame for signals to settle

In scenario 2, when the input detection circuitry is faster or have nearly equivalent speed of operation when compared to the functional block, the latching circuitry will enable the passing of valid/null data across the register "REG" before the functional block has actually finished computation. The timing analysis performed earlier and value of t_{settle} obtained is still valid here. If the intermediate nodes in the functional block settle within this time, correct operation can be guaranteed. If the time frame is not sufficient, or when the delay of functional block is very large compared to input detection circuit, there is a possibility of floating data on the intermediate nodes. To understand more on how floating data is present on intermediate nodes, consider an example circuit presented in Figure 4.15. In this figure, the functional block is accompanied by additional logic block chains which have an average delay of 1ns and 100ns respectively. The functional block comprises of the circuit presented in Figure 4.6 where input 'D' is connected to the output of logic block with a shorter delay. The functional block comprises of strongly-indicating dual rail gates without hysteresis mechanism in them.

At the start of operation, all the inputs, intermediate and output nodes in the functional block and logic block chain are at null state. When valid inputs are provided, input 'D' receives valid value after 1ns, but functional block has to wait additional 99ns for remaining set of valid input data to produce valid outputs. This happens because the functional block comprises of strongly-indicating gates (without hysteresis). After 100ns, functional block receives all valid data and starts producing valid output. This valid output generation is presented in Figure 4.16.

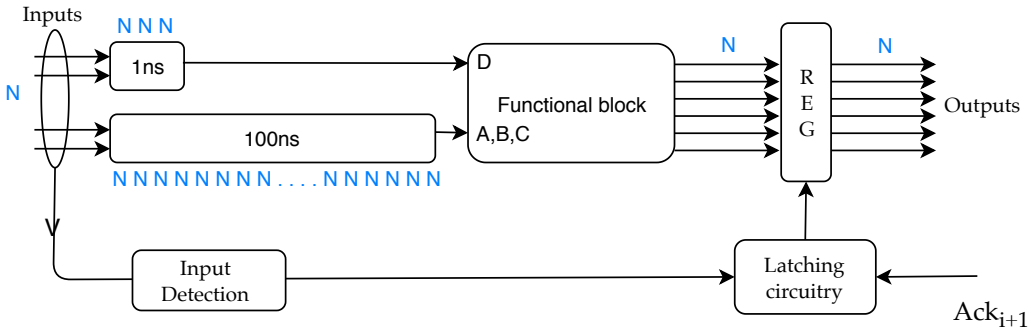


FIGURE 4.15: Circuit for floating data analysis (N = null data)

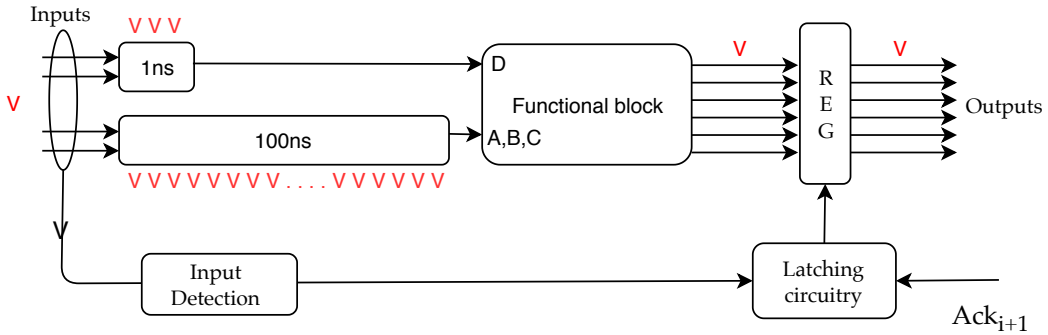


FIGURE 4.16: Valid output generation by circuitry meant for floating data analysis (V = valid data)

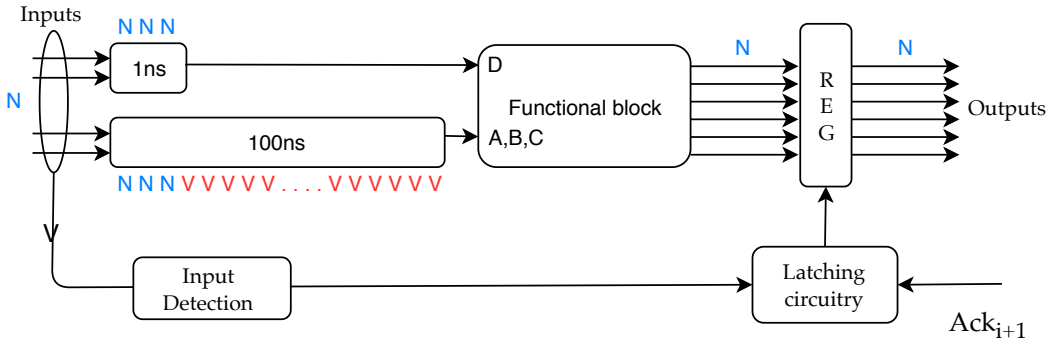


FIGURE 4.17: Null input data leading to null output generation

As per 4-phase signaling, after providing valid data, null data will be provided to the pipeline, as shown in Figure 4.17. Null data travels through shorter logic chain and reaches input 'D'. When input 'D' is at null state, all the outputs go to null state (as seen in Section 4.3), and, because of faster input detection circuit, this null value

is propagated and new valid data is requested by the pipeline. It should be noted that output is at null state while intermediate nodes are still valid. Generally this should be taken care of by the time frame discussed earlier, but it is not sufficient in this scenario.

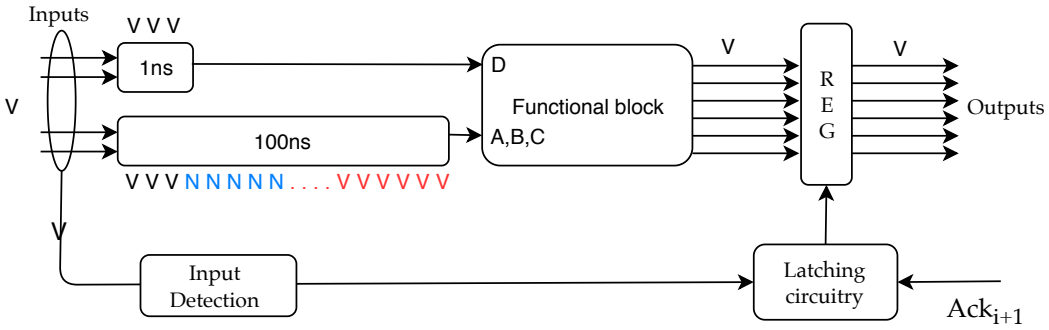


FIGURE 4.18: New valid input data clashing with older valid data

When the next set of valid inputs are provided as seen in Figure 4.18, floating data present in the logic block chain interferes with the newly arrived valid data. Functional block starts computing with valid set of data and produces incorrect results. This incorrect output generation is one of the example which shows the need for hysteresis mechanism. Due to absence of indicatability, fast moving signals produced valid output without waiting for slow changing signals. Careful study of this example reveals that incorrect output generation occurs because of faulty operation in cycle 2 where intermediate nodes did not turn null. Two solutions for this problem are presented here and both of them require additional hardware.

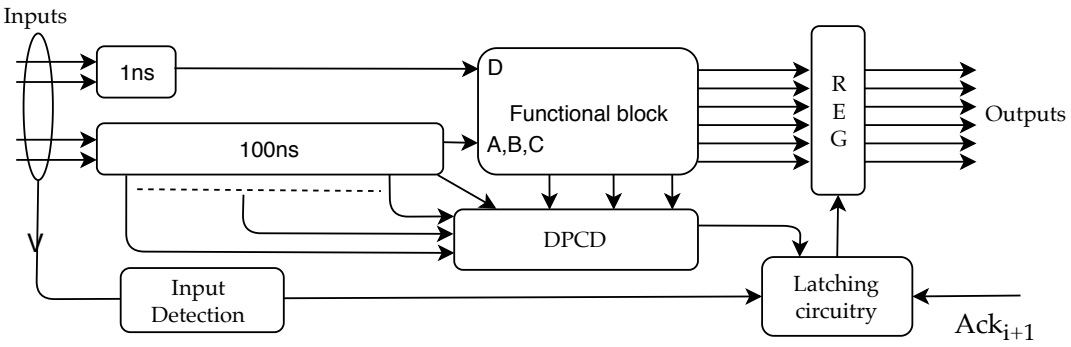


FIGURE 4.19: Introduction of DPCD to cope up with floating data problem

Solution 1 for the floating data problem involves monitoring of intermediate nodes apart from input and output nodes of a functional block. This monitoring can be done by Datapath Completion Detector (DPCD) circuit which is also presented in [20]. This DPCD circuit is simlar to the completion detector circuit and checks for valid and null data on the intermediate nodes of a functional block. This DPCD will help in establishing indicatability property for intermediate nodes and will also assist latching circuitry in passing valid/null data when input as well as intermediate nodes are at same status. This arrangement of DPCD is presented in Figure 4.19. In [20], basic dual rail gates are based on weakly-indicating logic (no hysteresis) which means that valid output can be generated with partial null inputs. In that scenario, DPCD works for both cycle 1 and cycle 2 of 4-phase signaling logic,

but in the example presented in Figure 4.19, DPCD is only used for cycle 2.

Solution 1 provides a QDI design technique which does not involve a hysteresis mechanism. Removal of hysteresis reduces transistor count, internal node capacitance and power consumption. This reduction leads to faster and compact designs. Complexity of DPCD is based on the complexity of functional blocks. Monitoring each and every intermediate node is not required and careful study of design can simplify the complexity of DPCD. This simplified DPCD can lower down the overall transistor count of the design, thus reducing net power consumption and making the circuit operate at faster speed. Also, instead of using strongly-indicating gates (no hysteresis), if weak-indicating gates (no hysteresis) are used, the same DPCD circuitry can be used to establish indicatability property in cycle 1 also, which will simplify the dual rail gates and will reduce the transistor count even further. This strategy is already implemented by the authors of [20].

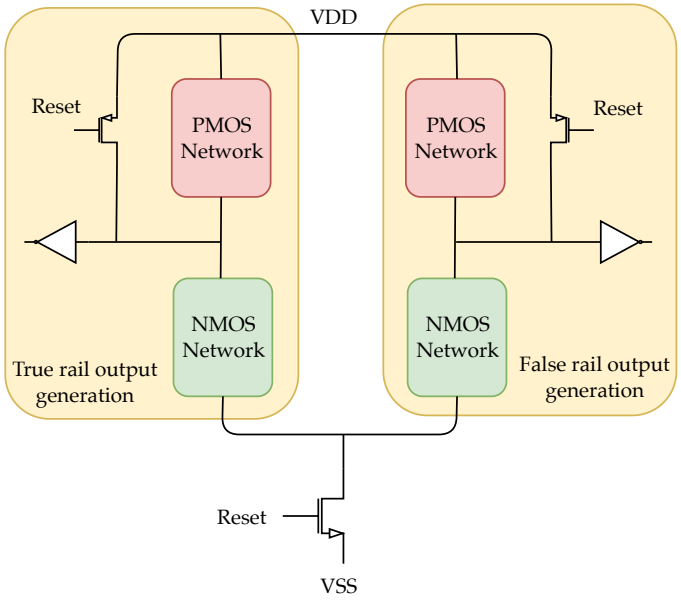


FIGURE 4.20: Abstract arrangement of PMOS and NMOS network for dual rail gates supporting reset mechanism

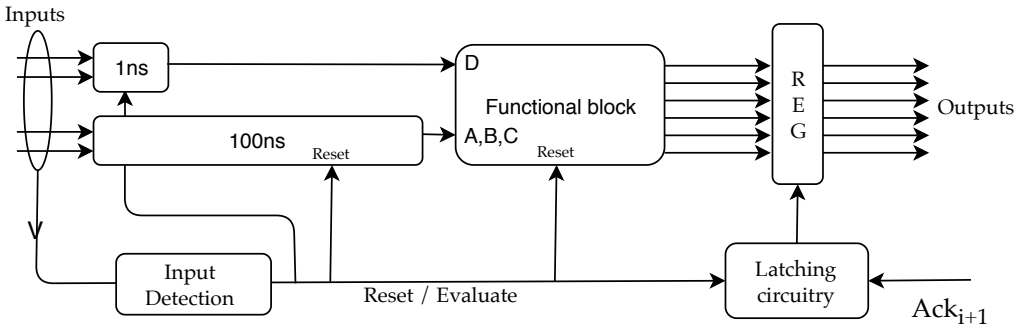


FIGURE 4.21: Pipeline arrangement with reset signal for avoiding floating data

Solution 2 for the floating data problem involves implementation of a reset mechanism for intermediate nodes. As per this mechanism, when primary inputs

to a functional block are null, the intermediate nodes will be reset to null state and thus the output generated will be null. This reset mechanism will make sure that there is no floating data inside the functional block. To reset the intermediate nodes, additional circuitry can be employed at the gate level. An abstract arrangement of PMOS and NMOS network in a dual rail gate with reset mechanism is presented in Figure 4.20. The reset signal shown in Figure 4.20 can be derived from the input detection circuit. The arrangement of functional block with reset signal in a pipeline is presented in Figure 4.21.

The resetting solution has its own advantages and disadvantages. The main advantage is speed gain which comes in the cycle 2 of 4-phase signaling. Null data instead of traveling from primary inputs to the outputs, is now generated instantly at every node by means of reset mechanism. This instant reset saves time for null data propagation and speeds up the design. The disadvantage of this solution is increased transistor count and power consumption. Implementing reset circuitry involves 2 PMOS and 1 NMOS transistors per dual rail gate. Also, the input detection circuit should have a buffered network of its output to drive these individual reset signals. Arrival time of reset signals to individual gates should not have a huge time difference. As the functional block grows larger, this network of buffers and reset signals also grows larger, which makes this solution equivalent to a synchronous design where the clock signal is propagated across the entire design using buffers. This solution will not work well with complex functional blocks and will end up consuming more power making this solution not feasible for design implementation.

Authors of [20] have implemented a mix of solution 1 and solution 2. This mix of solution will also be implemented for this thesis work and will become one of the asynchronous variants of Kogge-Stone adder. More details about the design and template based on these solutions are presented in Chapter 5.

In cases where primary inputs are provided synchronously (approximately at the same time) to the design, the processing in functional block due to all the input signals will overlap with the working of input detection circuit. The functional block has a time frame of t_{settle} time unit (as defined earlier) to settle its internal nodes. If primary inputs to the asynchronous pipeline are coming synchronously, then it is not guaranteed that inputs to the subsequent stages will remain synchronous. Due to uneven delay paths in the functional blocks, the outputs produced will be asynchronous. These asynchronous outputs will be fed to the next stage of pipeline and since the register present in between stages do not synchronize the data, the inputs to the next stage remains asynchronous. So, the general case discussed earlier with timing assumptions is applicable to all the scenarios.

4.5 Summary

Hysteresis plays a vital role in the design of asynchronous circuits by providing indicatability to the inputs as well as intermediate nodes of a design. The hysteresis mechanism requires additional 1x-1.5x transistors than the transistors required for actual logic implementation. These additional transistors increase power consumption and at the same time slow down the speed of operation.

Removal of the hysteresis mechanism results into loss of indicatability property. This loss can be taken care of by performing some timing analysis and adding some control circuitry. The timing analysis performed cannot be generalized and depends on the complexity of functional blocks involved. Instead of carrying out this analysis with strongly-indicating gates (without hysteresis), if weakly-indicating gates (without hysteresis) are considered, then indicatability is required in cycle 1 also. Some of the solutions mentioned already take care of the usage of weakly-indicating gates. General approaches of using DPCD and reset mechanism ensures that the design stays QDI and operate asynchronously. If usage of DPCD complicates the design and leads to more power consumption, then the granularity of functional blocks in a pipeline should be adjusted such that the time frame t_{settle} becomes sufficient for internal nodes to settle down. If granularity of functional blocks cannot be adjusted then suitable safety margin should be added in terms of delay chains to increase this time frame. Addition of delay chains will divert this design style from QDI delay model.

Removal of hysteresis saves some transistors but additional transistors are involved in implementing reset circuitry and DPCD. The effect on the power consumption and speed of operation because of hysteresis removal will be studied in upcoming chapters. The next chapter will focus on designing pipeline, and their associated join and fork mechanisms.

Chapter 5

Asynchronous Pipeline Template Design

The discussions presented in earlier chapters indicate that asynchronous designs sometime perform better (in terms of power) their synchronous counterparts. Avoiding clock signal should save significant amount of power making the asynchronous designs, implemented with handshaking mechanism, inherently run at maximum speed which is governed by gate delays and supply voltage specifications. In real scenarios, asynchronous designs are slower and consume more power at their highest operational speed in nominal voltage range. Asynchronous designs outperform synchronous designs only at very low throughput and low supply voltages, as presented in [20]. This different power saving profile at nominal and low supply voltages indicate a cross-over point. In this chapter, different pipeline templates will be presented based on the discussions presented in Chapter 4. If a cross-over point exist, it will be identified for the design implemented with one of the optimized templates.

As per Chapter 3, a mix of QDI templates and cell level designs can lead to an optimum design in terms of power and performance. Also, as per Chapter 4, different timing assumptions and type of indicatability chosen will lead to different designs. This choice will also affect the performance and power of a design. All these templates are studied and based on the timing assumptions discussed, template designs supporting all the control circuitry are presented in this chapter. *It should be noted that all the templates discussed in this chapter will support unidirectional flow of data within a functional block. If there is a loop within functional block, then, asynchronous registers along with required control circuitry should be used to break that loop.* Before discussing the templates, it is important to gauge the characteristics required by a pipeline. These characteristics are presented below :

- Pipeline should be an IL type of pipeline which promises higher performance when compared to DCD pipeline. Also, IL pipelines are easy to design, which means reduced design efforts.
- Choice of half buffer (4-phase) and full buffer (2-phase) pipelines has a direct impact on the complexity of functional blocks and handshaking circuitry. Half buffer pipelines are simple to design, making them a good choice for faster implementation.
- Strongly-indicating (weakly-conditioned) gates should be the approach for simple and easy way of designing, but, weakly-indicating (strongly-conditioned) gates can save some transistors. Choice of indicatability

can greatly affect the complexity of pipeline template. This will be observed in upcoming sections.

- Implementation of a sleeping mechanism in the pipeline by using power gating transistors should be avoided in high throughput applications.
- Pipeline should be dynamic in terms of performance. If high throughput required, pipeline should run at nominal voltage of 1.2 V (or as per the cell technology used). In low throughput scenarios, techniques like FDVS should be used to slow down the pipeline. FDVS make the operation of design in near-threshold and sub-threshold regions possible which saves quite a lot of power. This approach provides us with high performance as well as low power wherever and whenever required.
- There are a lot of implementations and optimization approaches at the transistor level. FDVS is best supported by static implementation of cells/gates. This static implementation is discussed with other approaches in next chapter.

Based on these characteristics, three pipeline templates will be designed and discussed in upcoming sections. These templates are based on both weakly-indicating and strongly-indicating type of gates.

5.1 Basic Elements

Before diving into the pipeline templates, it is important to understand the basic components used while designing an asynchronous pipeline. These basic components involve registers, input and output completion detectors, and, latching circuitry. All these components are explained in following subsections.

5.1.1 Register

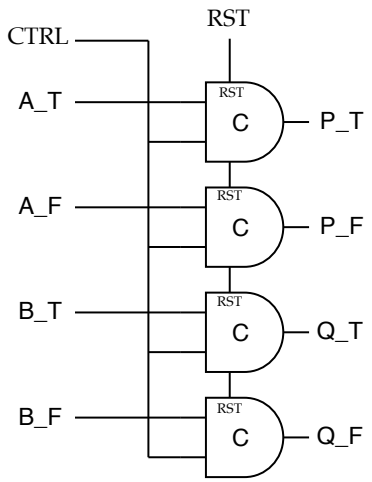


FIGURE 5.1: Asynchronous dual rail 2 bit register implemented using C elements

The asynchronous register, unlike its synchronous counterpart, does not save data. Instead, it controls the flow of valid and null values across the design based on a

control signal. It involves a parallel array of Muller C elements whose one terminal is tied to a control signal (CTRL) and other end is tied to input data. If required, reset/set functionality can also be provided to the Muller C elements which will assist the pipeline in starting from a predetermined state.

A 2-bit dual rail asynchronous register supporting reset feature is presented in Figure 5.1. When "CTRL" signal is at logic '0', only null value can pass through the register; when "CTRL" signal is at logic '1', only valid value can pass through the register. When control signal and input data are at at different logic levels, output data holds.

5.1.2 Completion Detection Circuit

Completion detection circuitry is used to detect the validity and neutrality of data. In Chapter 4, terms like input detection circuit, output detection circuit and DPCD correspond to this completion detection circuitry. It involves the usage of OR gates and Muller C elements. Figure 5.2 presents a completion detector for 2 bit dual rail signal.

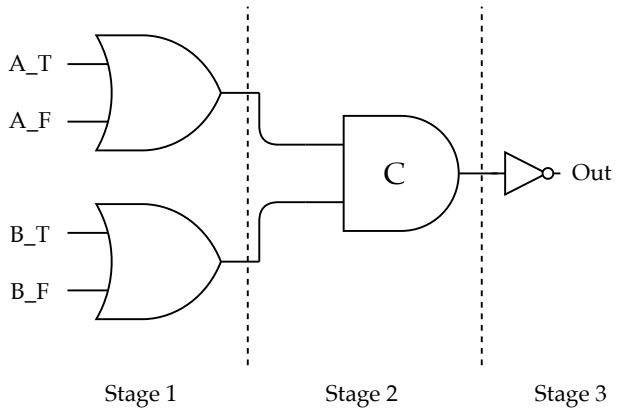


FIGURE 5.2: Completion detector for 2 bit dual rail signal

If the number of input signals increase, then changes required here are : increased parallel arrangement of OR gates at stage 1, and, multi-input Muller C element or tree-like arrangement of Muller C elements in stage 2. An inverter is placed in stage 3 which can be used if inverted signals are required in the pipeline. *It should be noted that dual rail data pattern "11" is illegal and should never be generated. If it gets generated due to incorrect logic implementation, efforts should be placed to modify the logic, or, instead of OR gates in stage 1, XOR gates should be used in stage 1 to filter out illegal data patterns (or, stall the pipeline).*

The logic used for designing completion detectors can be modified to reduce the transistor count and propagation delay of the circuit. There are two ways in which the circuit can be modified. They are :

- The NOT gate placed in stage 3 can be moved to the inputs of stage 2 Muller C elements. These NOT gates can then be combined with stage 1 OR gates to form NOR gates. This optimization reduces the transistor count and speeds up the completion detectors.
- Instead of having separate stages in the completion detector circuit, the entire logic can be developed as a complex gate which might save some transistors.

The presence of Muller C element complicates the implementation of this approach. Due to the hysteresis mechanism, capacitance on internal nodes increases tremendously which slows down the circuitry. Also, as the number of inputs increase, the completion detector circuit becomes complex and the number of PMOS and NMOS transistors in series keeps on increasing, making the complex gate very slow. Drawing layout for a complex gate is also difficult. Appropriate design decisions should be taken based on the performance required.

In some of the literature works (not presented in this thesis), there are numerous ways for detecting the validity and neutrality of data. Author of [21] presents a Current Sensing Completion Detection (CSCD) circuit which measures the current of functional block during operation. In the literature, it is stated that when a functional block is performing operation on data, more current is withdrawn in comparison to the current withdrawn in idle state. This difference in current magnitude helps in detecting the validity and neutrality of data. Operation of CSCD and additional circuitry involved can be referred from [21]. This current sensing approach will not be used for this thesis work.

In Figure 5.2, stage 2 comprises entirely of Muller C elements. Hysteresis gates are generally slower and area expensive when compared with basic static CMOS gates implementing the same Boolean expression (minimum sized transistors are considered for speed comparison). If speed of both type of gates has to be matched, transistor up-sizing is required which increases the gate capacitance as well as power consumption. To have a faster and lower power consuming completion detection circuit, efforts should be placed to use basic static CMOS gates. One such arrangement for completion detector is presented in Figure 5.3.

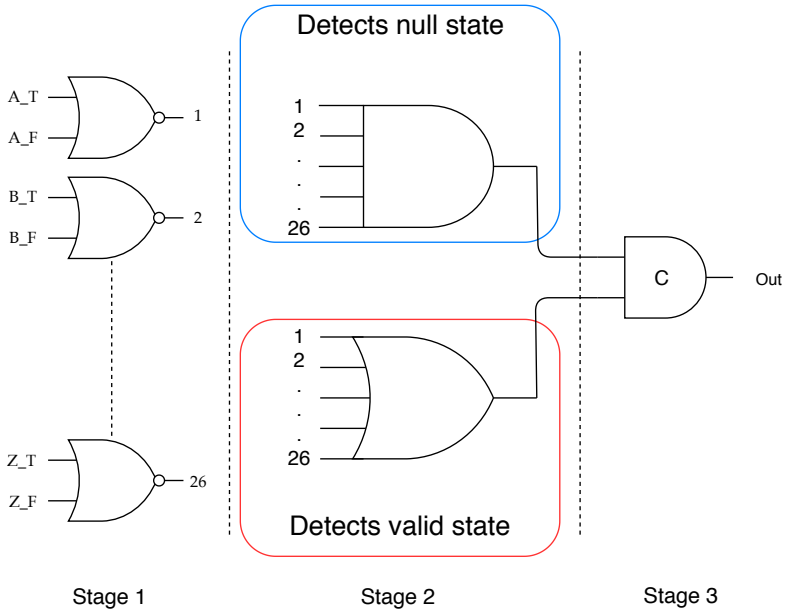


FIGURE 5.3: Completion detector using basic static CMOS gates

Stage 1 of this completion detector comprises of an array of NOR gates. Stage 2 comprises of two networks of AND gates and OR gates respectively. These networks are individually used to state the validity and neutrality of data. In stage 3, a 2-input Muller C element is used which is required to provide the required

hysteresis for completion detector circuit. It might be possible that various other completion detection techniques are available which are more area and power efficient, but the approach presented with basic static CMOS gates is simple and will be used for implementation.

A comparison between completion detectors made out of Muller C elements and basic static CMOS gates is presented in Table 5.1. This table presents the number of transistors used by completion detectors. This comparison assumes certain number of transistors for 2-input, 3-input and 4-input gates. The number of transistors assumed are presented in Table 5.2 and can be referred from Appendix A.

No. of inputs	With hysteresis gates	With basic CMOS gates
2	18	30
4	42	46
8	96	94
16	194	174
32	400	350
64	802	686

TABLE 5.1: Comparison of transistor count between completion detectors made out of hysteresis gates and basic CMOS gates

Gates	No. of transistors
2-input C element	12
3-input C element	18
4-input C element	26
2-input AND/OR gate	6
3-input AND/OR gate	8
4-input AND/OR gate	10
2-input NOR gate	4

TABLE 5.2: Number of transistors involved in gates used for completion detectors

From Table 5.1, it can be concluded that completion detector circuit made out of basic static CMOS gates is area efficient when compared to the completion detector circuit made out of hysteresis gates for larger number of inputs. Area efficiency can be directly linked to power efficiency if there are no other factors affecting both the designs. Since low power is the final aim, completion detector with basic static CMOS gates will be used for design implementation.

5.1.3 Latching circuitry

Latching circuitry is used in the pipeline to control the passing of valid and null values across the asynchronous registers. This latching behavior is controlled by observing the signals coming from completion detectors. Based on the type of pipeline and assumptions made, these completion detection signals may arrive from preceding and/or succeeding pipeline stages. In 4-phase signaling system, latching circuitry should enable passing of valid data when succeeding pipeline stages are ready to accept the valid data and current stage has produced valid data.

It goes similarly for the null data.

Latching circuitry works similarly to the completion detection circuit. If completion detectors are indicating desired status (valid or null), then, latching circuitry should pass data of same status (valid or null), else latching circuitry should hold the previous state. Latching circuitry gets complicated when pipeline have forks and joins. If a join with multiple incoming ends is considered, then, the complexity is extremely high and the transistors used to implement that latching circuitry follow the trend presented in Table 5.1. Examples of latching circuitry for different pipeline templates are presented in upcoming sections.

5.2 Template 1 : Weakly-indicating functional block with hysteresis mechanism

The pipeline template presented in this section is one of the simplest asynchronous pipelines. It is similar to the template used for NCL gates, as presented in Chapter 3. In this pipeline, 'F' stands for functional blocks, "REG" stands for registers and "CD" stands for completion detectors. In this template, functional blocks are weakly-indicating combinational logic and are made up of strongly-indicating gates with hysteresis mechanism present inside them. The basic structure of pipeline template is presented in Figure 5.4.

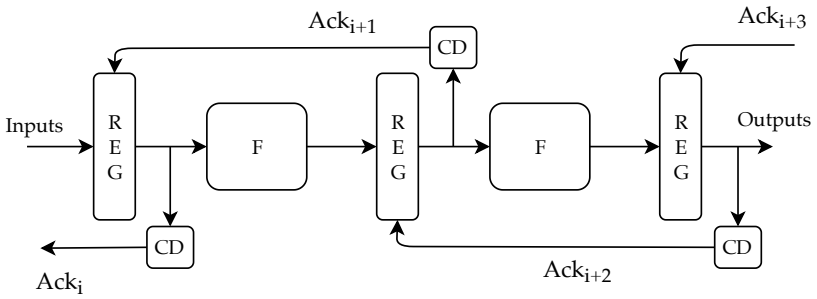


FIGURE 5.4: Pipeline template for weakly-indicating functional blocks with strongly-indicating gates

For proper initialization of the pipeline to all null state, the reset signal present in the registers can be asserted. Functional block follows indicatability property due to the presence of strongly-indicating gates with hysteresis. This simplifies the pipeline and there is no need of input detection circuit and complex latching circuitry. The pipeline presented in Figure 5.4 becomes complicated when components like fork and join are present in the design.

In the case of a 3-way fork, as presented in Figure 5.5, acknowledgment signals (Ack_1$_i$, Ack_2$_i$ and Ack_3$_i$) coming from the forked stages need to be combined for correct flow of valid and null data. This combining of acknowledgment signals is performed by a latching circuitry (3-input Muller C element). With this arrangement of components, the pipeline automatically adjusts its speed of operation as per the slowest forked stage.

In the case of a 3-way merge, as presented in Figure 5.6, the acknowledgment signal (Ack$_i$) coming from the merge stage is shared by the different incoming

pipeline stages. No latching circuitry is required for performing a merge operation. Also, in the merge operation, the merge stage automatically adjusts the speed of pipeline as per the slowest incoming pipeline stage.

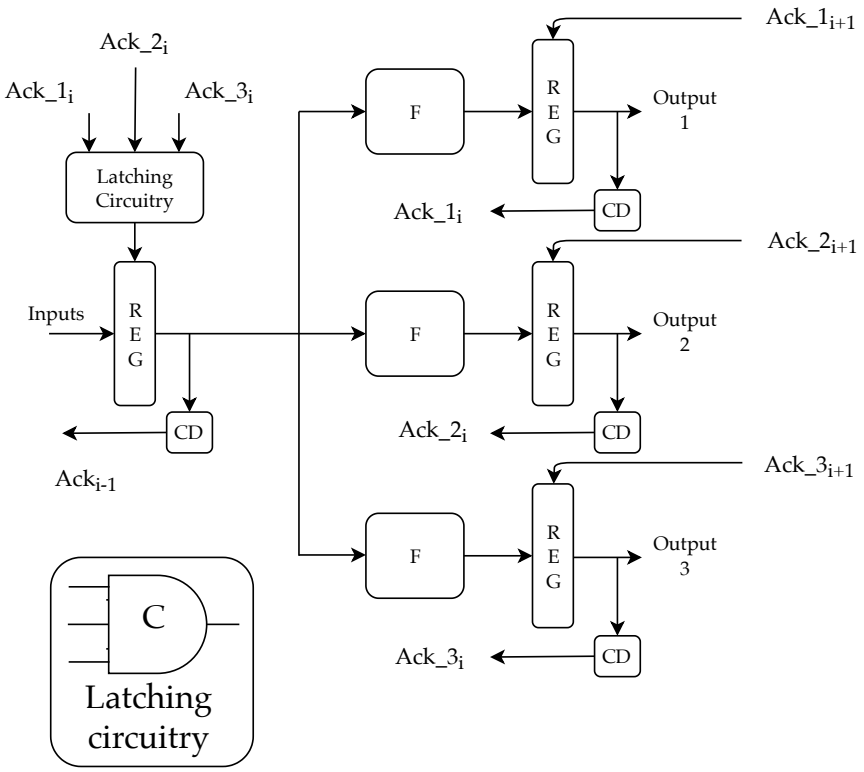


FIGURE 5.5: 3-way fork mechanism for pipeline template 1

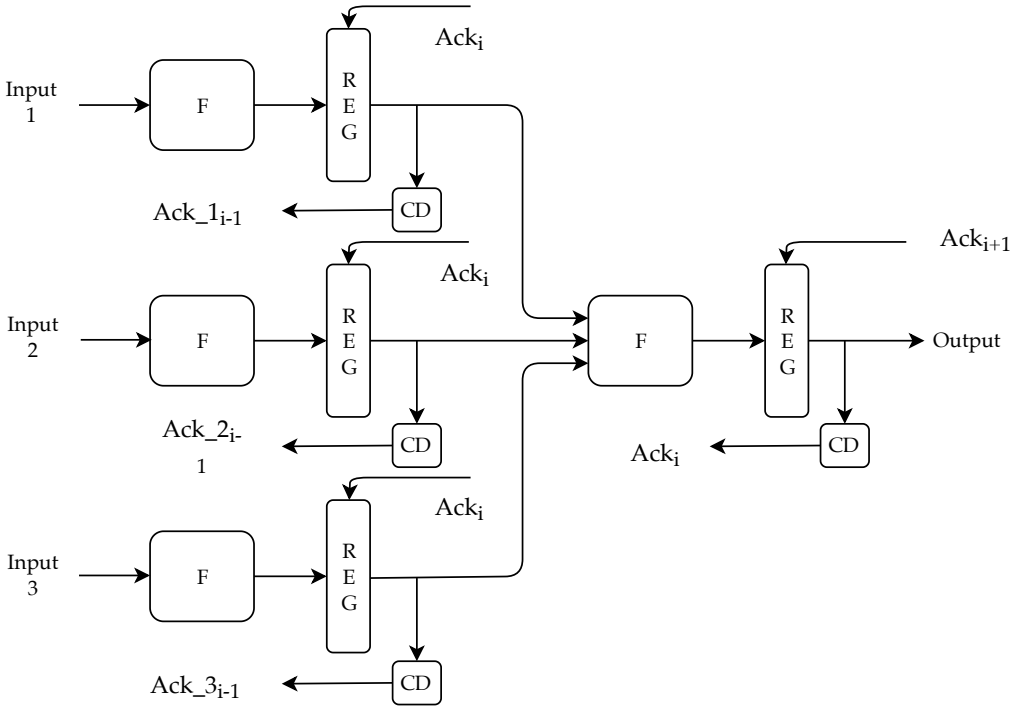


FIGURE 5.6: 3-way merge mechanism for pipeline template 1

5.3 Template 2 : Weakly-indicating functional block with weakly-indicating gates and DPCD[20]

The pipeline template presented in this section is based on the filter design discussed by the authors of [20]. In this pipeline, 'F' stands for functional blocks, "REG" stands for registers, and, "CD" and "DPCD" stands for completion detectors. In this template, functional blocks are weakly-indicating combinational logic and are made up of weakly-indicating gates without hysteresis mechanism. These weakly-indicating gates also support reset mechanism which is used for avoiding the floating data on the intermediate nodes of a functional block. This pipeline template implements both the solutions presented earlier in Chapter 4 for the floating data problem. The basic structure of pipeline template is presented in Figure 5.7.

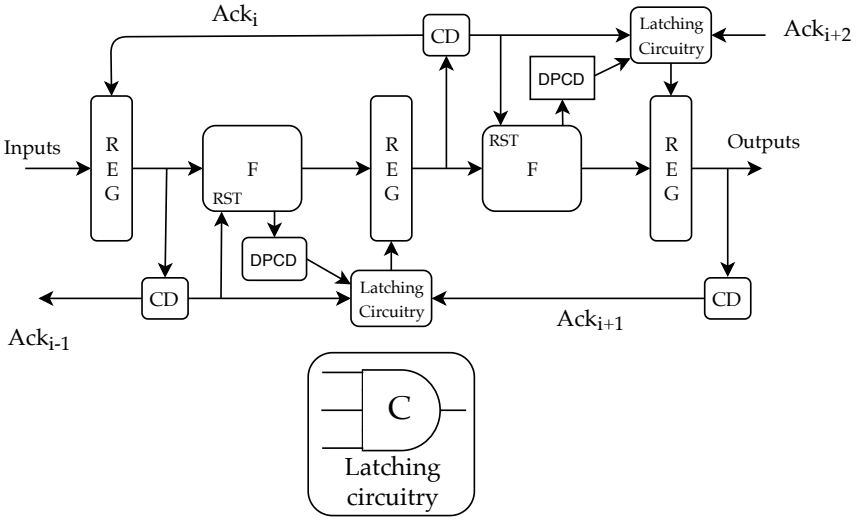


FIGURE 5.7: Pipeline template for weakly-indicating functional blocks with weakly-indicating gates and reset mechanism

From Figure 5.7, it can be observed that the control circuitry surrounding functional block gets complicated when the hysteresis mechanism is removed. The transistors saved by removing hysteresis are now used to implement additional control circuitry to ensure QDI nature of the design. Additional wires are laid down for propagating reset signals across the entire functional block. As the functional block gets complicated, more reset signals are required to be fed at the gate level which will further lead to the introduction of buffers and increase in power consumption. Presence of DPCD and completion detection signals from preceding and succeeding stages of the pipeline leads to the introduction of latching circuitry. For a linear pipeline, this latching circuitry is a 3-input Muller C element, but it gets complicated when components like fork and join are present in the pipeline.

In the case of a 3-way fork, as presented in Figure 5.8, the acknowledgment signal (Ack_{i-1}) travels to the preceding stage as well as to all the forked stages to drive their reset as well as latching circuitry. This requires the completion detector circuit generating Ack_{i-1} signal to have a higher driving strength. Also, acknowledgment signals Ack_1_i, Ack_2_i and Ack_3_i coming from the forked stages need to be combined for correct flow of valid and null data. This combining of acknowledgment signals is performed by a latching circuitry (3-input Muller C

element). With this arrangement of components, pipeline automatically adjusts its speed of operation as per the slowest forked stage. The latching circuitry for the input stage, as presented in Figure 5.8, gets complicated when the size of fork stage is increased, or, when the fork is present in between multiple pipeline stages. In the case where fork is present in between multiple pipeline stages, latching circuitry will then have to accommodate the completion detection signal and DPCD signal coming from preceding stage.

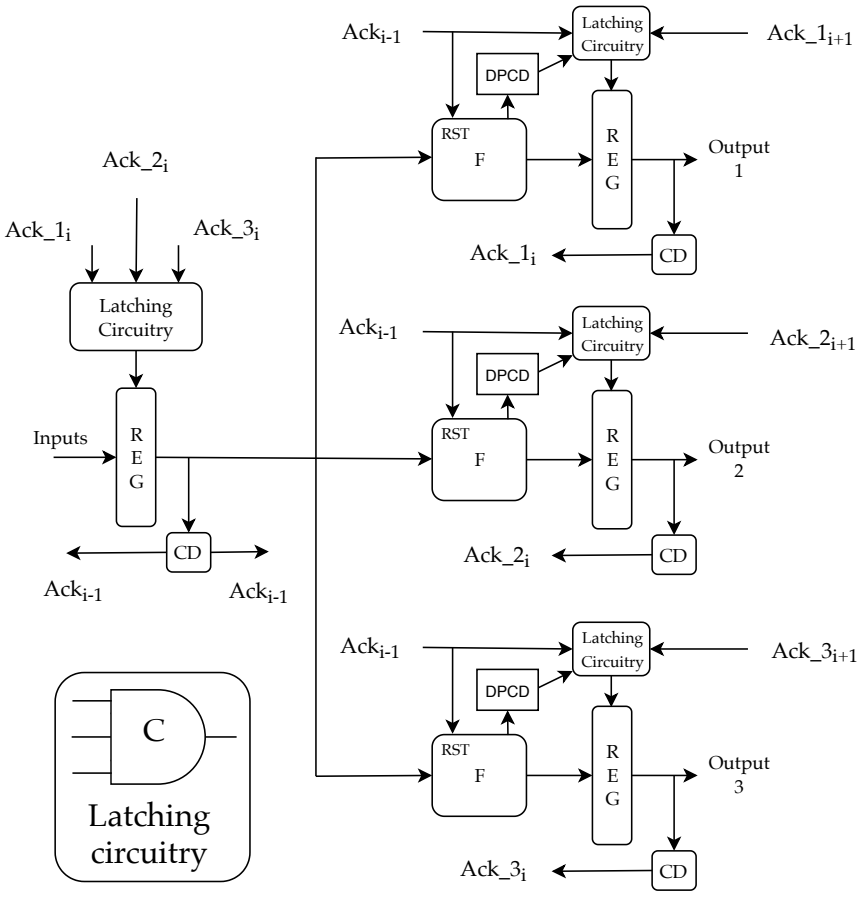


FIGURE 5.8: 3-way fork mechanism for pipeline template 2

In case of a 3-way merge, as presented in Figure 5.9, the acknowledgment signal (Ack_{i+1}) coming from the merge stage is shared by the different incoming pipeline stages. This shared incoming acknowledgment signal is then used by the individual latching circuitry (3-input Muller C element) along with their respective signal from DPCD and preceding stages to control the flow of valid and null data. This acknowledgment signal also makes sure that the pipeline automatically adjusts its speed of operation as per the slowest incoming pipeline stage.

From Figure 5.9, it can be observed that the merge stage has 3 incoming completion detection signals (Ack_1_i, Ack_2_i and Ack_3_i) from 3 preceding stages. These signals will be used to drive the latching circuitry as well as the reset signal of the functional block. These three signals have to be combined to form a single reset signal which will put the functional block into reset state when these signal indicate null data, and will lift off the reset signal when all the three signals indicate valid data. This reset circuitry can be implemented using a 3-input Muller C element. Apart from this reset circuitry, the merge stage also requires a complicated

latching circuitry which is capable of handling 5 control signals (Ack_1_i, Ack_2_i, Ack_3_i, Ack_{i+2} and Sig_{DPCD}). This latching circuitry can be implemented using a 5-input Muller C element. Implementation of a 5-input Muller C element is very complicated, so, an alternate implementation of latching circuitry is presented in Figure 5.10. This latching circuitry presented in Figure 5.10 is similar to the completion detector circuits (see Figure 5.3) without stage 1 of NOR gates.

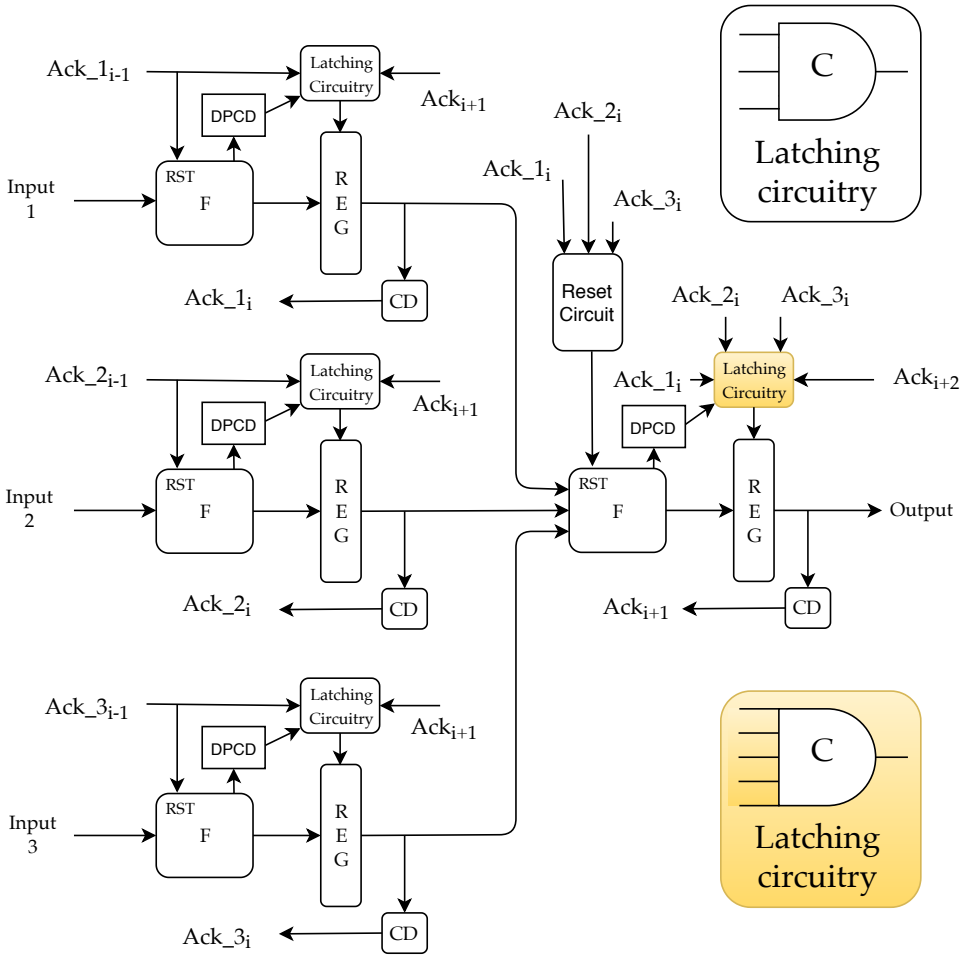


FIGURE 5.9: 3-way merge mechanism for pipeline template 2

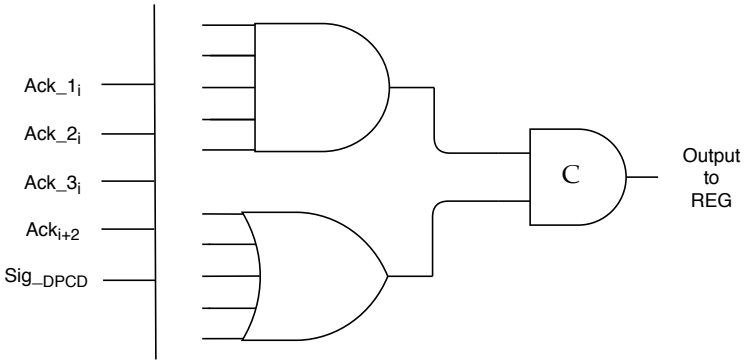


FIGURE 5.10: Latching circuitry for handling 5 control signals

5.4 Template 3 : Weakly-indicating functional block with timing assumption t_{settle}

The pipeline template presented in this section is based on the timing analysis discussed earlier in Chapter 4. As per the timing analysis, the functional block has a time frame of t_{settle} time units to settle its internal nodes. If the functional block is not able to satisfy the time frame, efforts should be placed to reduce the complexity of the functional block by dividing the implemented logic across multiple pipeline stages. Dividing logic across multiple pipeline stages will introduce additional hardware in terms of completion detectors and registers. Another solution to this problem is to increase the duration of time frame available for functional blocks. The duration of time frame can be increased by using programmable delay elements [22] in the delay path mentioned in Figure 4.14. These delay elements can be programmed to suit the requirements of a complex functional block.

This pipeline is based on timing assumptions and programmable delay elements which make this pipeline template deviate from QDI nature of asynchronous circuits. Circuits implemented with this template are no longer delay insensitive but they are capable of handling asynchronous arrival of data. If correctly programmed delay elements are present in the delay path, then the operating range for the design can be increased. This increased operating range will make this pipeline template as robust as QDI templates.

In this pipeline, 'F' stands for functional blocks, "REG" stands for registers, and, "CD" stands for completion detectors. In this template, functional blocks are weakly-indicating combinational logic and are made up of weakly-indicating gates without hysteresis mechanism. There is no gate-level reset mechanism present and the floating data problem is taken care of by the time frame. The basic structure of the pipeline template is presented in Figure 5.11.

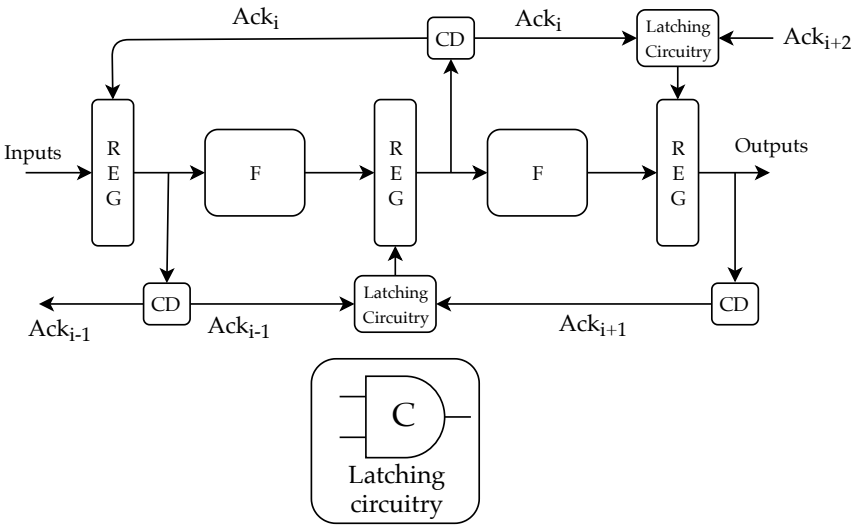


FIGURE 5.11: Pipeline template for weakly-indicating functional blocks with weakly-indicating gates (No Reset)

From Figure 5.11, it can be observed that the no additional control and DPCD circuitry is involved. This pipeline template looks similar to the pipeline template 1, with an exception of the latching circuitry. For a linear pipeline, this latching

circuitry is a 2-input Muller C element. Similar to previous pipeline templates, the latching circuitry gets complicated in case of fork and join mechanisms.

In the case of a 3-way fork, as presented in Figure 5.12, acknowledgment signal (Ack_{i-1}) travels to the preceding stage as well as to all the forked stages to drive their latching circuitry. Acknowledgment signals Ack_1_i, Ack_2_i and Ack_3_i coming from the forked stages need to be combined for correct flow of valid and null data. This combining of acknowledgment signals is performed by a latching circuitry (3-input Muller C element). With this arrangement of components, the pipeline automatically adjusts its speed of operation as per the slowest forked stage.

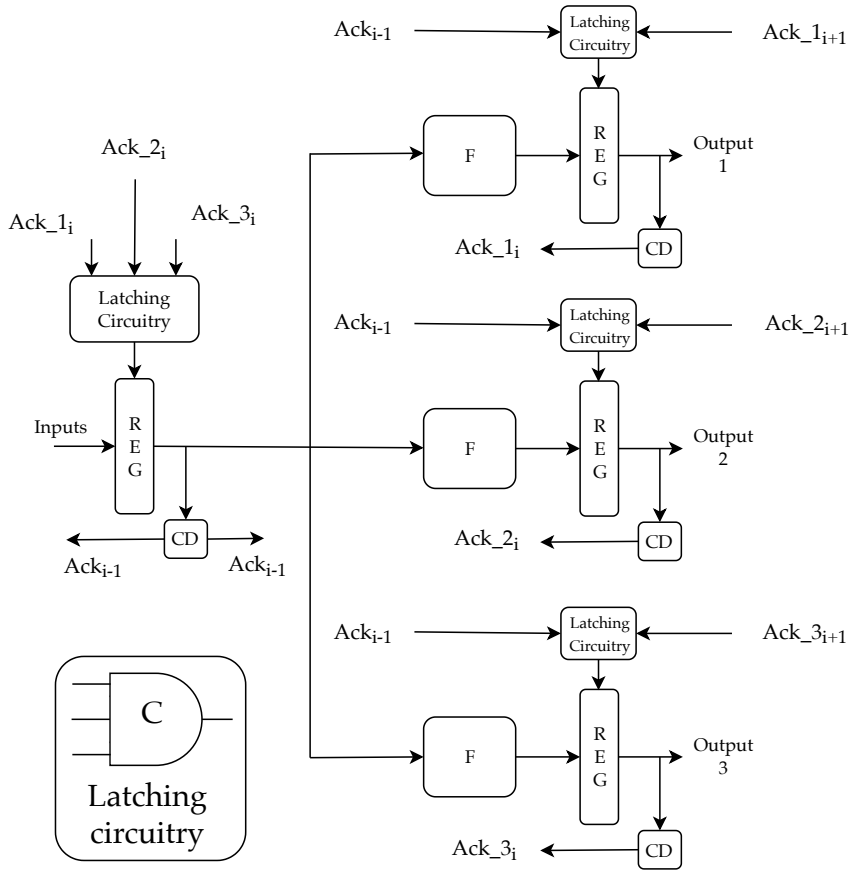


FIGURE 5.12: 3-way fork mechanism for pipeline template 3

In case of a 3-way merge, as presented in Figure 5.13, acknowledgment signal (Ack_{i+1}) coming from the merge stage is shared by the different incoming pipeline stages. This shared incoming acknowledgment signal is then used by the individual latching circuitry (3-input Muller C element) along with their respective signal from preceding stages to control the flow of valid and null data. This acknowledgment signal also makes sure that the pipeline automatically adjusts its speed of operation as per the slowest incoming pipeline stage.

From Figure 5.13, it can be observed that the merge stage have 3 incoming completion detection signals (Ack_1_i, Ack_2_i and Ack_3_i) from 3 preceding stages. These signals along with acknowledgment signal from the succeeding stage will be used to drive the latching circuitry of merge stage. This latching circuitry should be capable of handling 4 control signals (Ack_1_i, Ack_2_i, Ack_3_i and Ack_{i+2}). The

5-input latching circuitry discussed earlier can be modifed to get a 4-input latching circuitry for this merge stage.

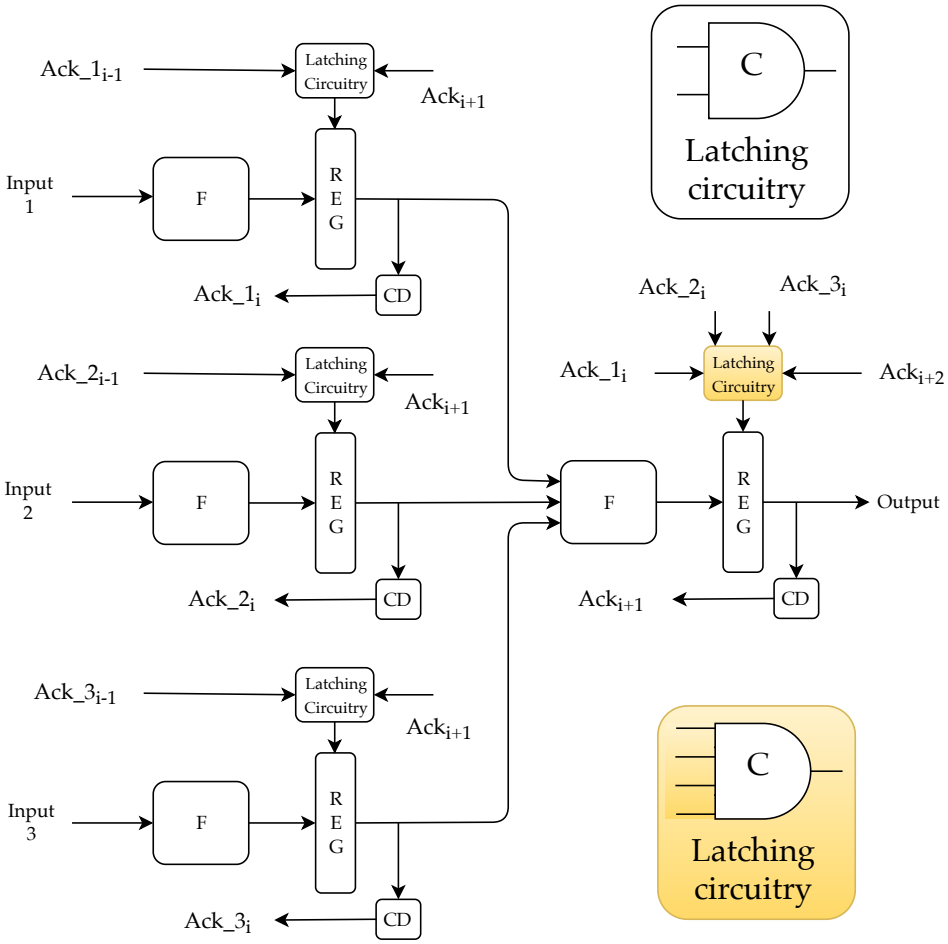


FIGURE 5.13: 3-way merge mechanism for pipeline template 3

5.5 Example Designs

Template 1 and 2 discussed earlier promise QDI class of designing, whereas, template 3 deviates from QDI model but can be as robust as QDI designs if the time frame is satisfied by the functional blocks in all the operating conditions. *Satisfying the time frame in all the operating conditions indicate towards the worst case performance. This is similar to the worst case performance given by synchronous designs. The major difference that helps distinguish asynchronous designs from their synchronous counterparts is that the asynchronous designs are capable of handling asynchronous inputs which are naturally not supported by synchronous designs.* So, these templates should be used in the scenarios where inputs are arriving asynchronously from external environment.

Aim of this thesis work is to find a design strategy for robust and low power circuit designing. The templates discussed in this chapter and the analysis performed in Chapter 4 provides multiple way of designing robust low power circuits. To quantify these design approaches, a 64-bit unsigned Kogge-Stone Adder was implemented. All the three designs were first verified at behavioral level using VHDL. VHDL simulations can be found in Chapter 7. After this, analog simulations were carried out in Cadence Virtuoso (More information on tooling is presented

in Chapter 7). To give an idea about the design results and type of structures involved, a 4 bit Kogge-Stone adder and a GCD circuit are taken as an example and were implemented at behavioral level using weakly-indicating type of logic. These designs and their implementation details are presented in the following subsections.

5.5.1 Design 1 : 64-bit Unsigned Kogge-Stone Adder

Addition is the basic element in most of the digital hardware designs. The ripple carry adder is the simplest adder but has a long critical path. To make addition faster, multiple addition techniques are used, out of which Kogge-Stone is one of them. Figure 5.14 presents the top-level view of 4 bit Kogge-Stone Adder. This Kogge-Stone adder does not have any pipeline stages and the entire Kogge-Stone adder is implemented as a complex functional block.

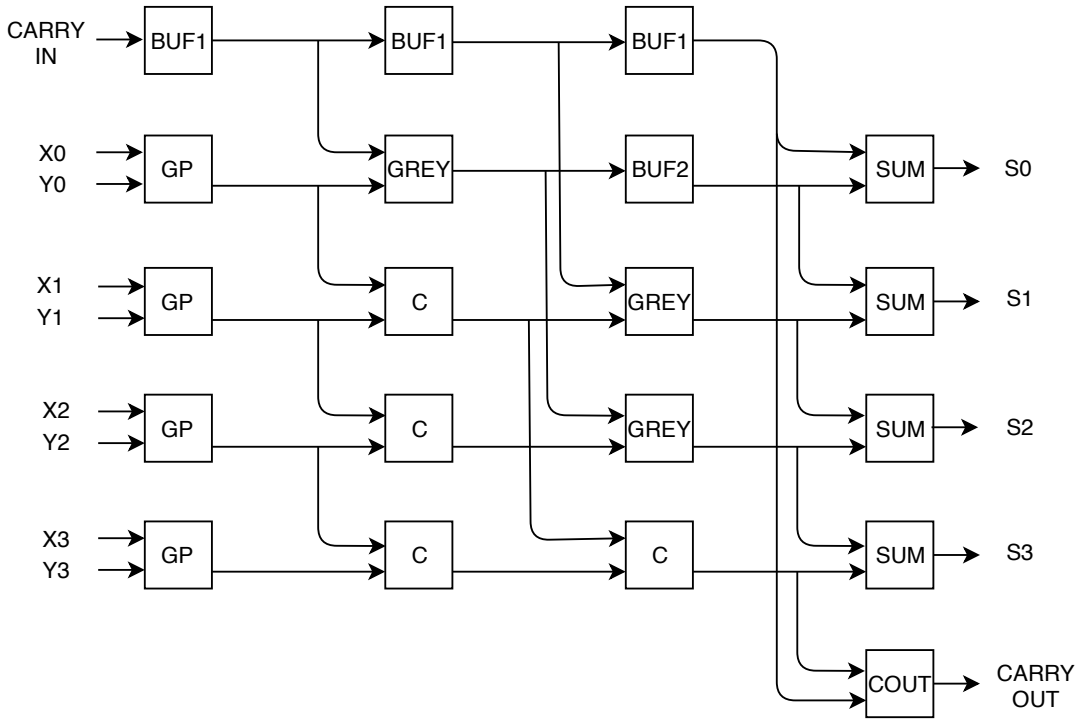


FIGURE 5.14: Top level view of 4 bit Kogge-Stone Adder

In Figure 5.14, handshaking signals are not shown to reduce clutter. But these signals can easily be drawn based on the fork and join mechanism presented earlier. There are 7 different types of leaf cells present in the design : BUF1, GP, GREY, C, BUF2, SUM and COUT. Each cell has its own unique function and will be discussed in following sections.

5.5.1.1 BUF1 leaf cell

This cell is used to propagate the primary carry signal which is further used by intermediate stages to compute intermediate carry signals, and, by the final stage to compute carry out signal. This cell comprises of buffer cells to provide required driving strength and to drive multiple gates which make use of primary carry signals.

5.5.1.2 GP leaf cell

This cell is used to compute the carry generate and carry propagate signals from the primary inputs. For synchronous designs, the generate signal is defined by $G_i = X_i$ **and** Y_i and propagate signal is defined by $P_i = X_i$ **xor** Y_i. For weakly-indicating type of logic (without hysteresis), the expression obtained for the generate and propagate signals are :

$$G\_T = X\_T \,.\, Y\_T$$

$$G\_F = X\_F + Y\_F$$

$$P\_T = X\_F \,.\, Y\_T + X\_T \,.\, Y\_F$$

$$P\_F = X\_T \,.\, Y\_T + X\_F \,.\, Y\_F$$

5.5.1.3 GREY leaf cell

This cell is used to compute the intermediate carry signals from generate-propagate and carry signal from previous stage. For synchronous designs, the intermediate carry signal is defined by $C = G_i + P_i \,.\, C_{in}$. The expressions obtained for the intermediate carry signal are :

$$C\_T = G\_T + P\_T \,.\, C_{in}\_T$$

$$C\_F = G\_F \,.\, P\_F + G\_F \,.\, C_{in}\_F$$

It should be noted that the C_{in} signal in the expression is the primary carry input for stage 1 of grey cells, and, intermediate carry signals for subsequent stages of grey cells. For more details on this, refer Figure 5.14.

5.5.1.4 CARRY (C) leaf cell

This cell is used to compute net generate-propagate signal from incoming generate-propagate signals. For synchronous designs, the net generate signal is defined by $G = G_2 + G_1 \,.\, P_2$, and, net propagate signal is defined by $P = P_1 \,.\, P_2$. The expressions obtained for the net generate-propagate signals are :

$$P\_T = P1\_T \,.\, P2\_T$$

$$P\_F = P1\_F + P2\_F$$

$$G\_T = G2\_T + P2\_T \,.\, G1\_T$$

$$G\_F = G2\_F \,.\, G1\_F + G2\_F \,.\, P2\_F$$

5.5.1.5 BUF2 leaf cell

This cell is used to propagate the primary propagate signal from GP cells and intermediate carry signals generated from GREY cells. This cell comprises of buffer cells to provide required driving strength and to drive multiple gates which make use of these signals.

If the Kogge-Stone adder is implemented as a non-pipelined combinational logic, then this cell is similar to the BUF1 leaf cell discussed earlier. In case of a highly pipelined design (pipeline register after every gate-level stage), BUF2 cell comprises

of registers and completion detectors of different sizes, when compared to BUF1 leaf cell.

5.5.1.6 SUM cell

This cell is used to compute final sum from the intermediate carry signals generated by GREY cells and the primary propagate signals which is being traveling down the pipeline from buffers. For synchronous designs, this signal is defined by $S_i = P_i$ xor C_{i-1}. The expressions obtained for the sum signal is :

$$S\_T = P\_F \,.\, C\_T \; + \; P\_T \,.\, C\_F$$

$$S\_F = P\_F \,.\, C\_F \; + \; P\_T \,.\, C\_T$$

5.5.1.7 COUT cell

This cell is similar to the GREY cell without an additional buffer for primary propagate signal. This cell generates carry out signal for a given set of primary inputs. This cell can also be omitted if output carry generation is not required.

5.5.2 Design 2 : Greatest Common Divisor (GCD)

GCD is one of the popular designs used as a base example for sequential circuits. GCD designing involves loop which is a bit difficult to implement with asynchronous logic, but careful logic design and proper placement of asynchronous registers is required to make design functional. This implementation of GCD circuit involves repeated subtraction till both operands are equal. This algorithm is known as Euclidean algorithm and is presented in Figure 5.15. Top level diagram for GCD circuit is presented in Figure 5.16.

```
function gcd(a, b)
    while a ≠ b
        if a > b
            a := a − b;
        else
            b := b − a;
    return a;
```

FIGURE 5.15: Euclidean algorithm for GCD calculation

The Kogge-Stone Adder designed in previous section is modified to operate as a subtractor. Mux Network is used to switch between primary inputs and result of subtractor; and feed the resulting data to the comparator cell. Comparator cell, as the name suggests, compares the input data and provide the result to subtractor and buffer chain. When input operands are equal, the comparator cell provides output data along with output validity signal. Buffer chain is used to buffer primary inputs of subtractor and comparison flags. These flags are then used by asynchronous controller to control the mux.

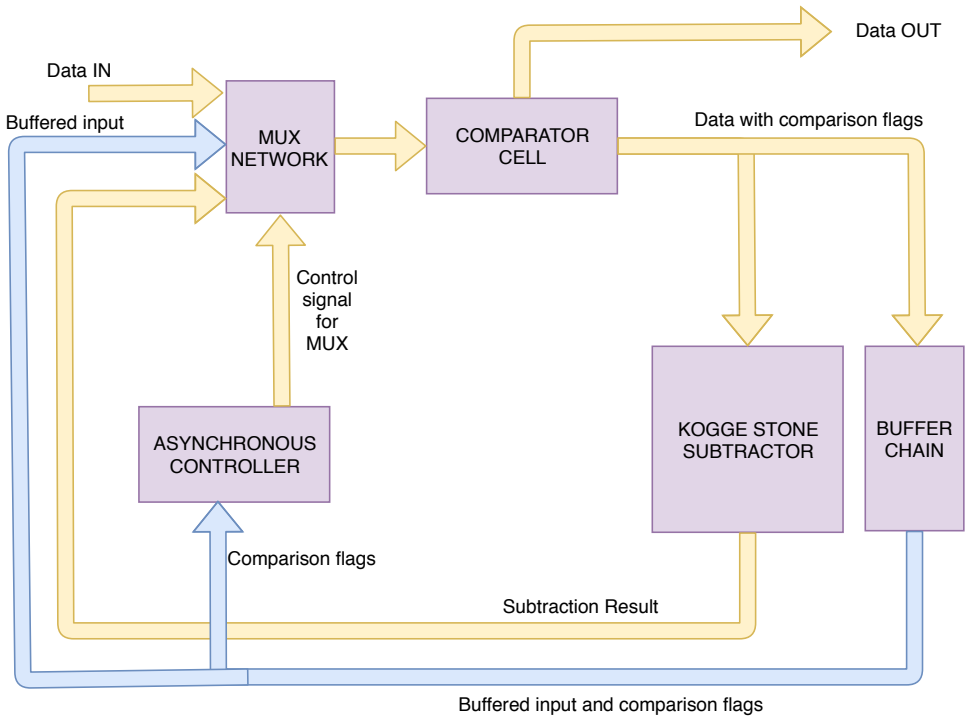


FIGURE 5.16: Top Level Diagram for GCD

5.6 Summary

This chapter discusses the characteristics required by an asynchronous template for providing performance as well as low power figures. Basic components involved in the pipeline design along with their optimized variants is also presented in this chapter. Different pipeline template designs based on the type of indicatability and logic blocks used are presented along with their respective fork and join elements.

With the explanation provided in this chapter for different templates, it is evident that Template 1 and Template 2 promise QDI nature at the cost of additional hardware and also have slower speed of operation. This additional hardware will lead to increased power consumption. More conclusive results on power consumption will be presented in Chapter 7. In contrast to this, Template 3 does not follow QDI nature but can be used to build robust asynchronous circuits.

At last, a 4-bit unsigned Kogge-Stone adder is presented with basic building blocks to give an idea about the functionality. This adder will be scaled to a 64-bit version to perform actual quantitative analysis. Along with this adder, a GCD block diagram is also presented which gives an idea on how to design sequential circuits using asynchronous pipelines. In the upcoming chapters, cell level and transistor level designing will be considered and the challenges faced in those areas will be discovered.

Chapter 6

Cell Design at Transistor Level

The selection of pipeline template and type of indicatability for functional blocks provides a direction towards the next step of design implementation. This next step involves selection of the most appropriate transistor design style based on the requirements dictated by operating conditions. There is a need of high performance, low power dissipating and less area intensive cell design style along with the capabilities of Full-range Dynamic Voltage Scaling (FDVS). FDVS is required to ensure that design works at nominal voltage for providing high performance (as required by the application), and, in near-threshold and sub-threshold regions to achieve lower performance (if operating conditions allow) with low power figures. There are various transistor design styles which need to be studied and compared based on the aforementioned requirements. Apart from these requirements, other requirements like driving strength, low leakage current, less static power, etc. should also be considered during the comparison. Following sections discuss different cell design styles and mention their advantages and disadvantages.

6.1 Static CMOS style [23][24][25]

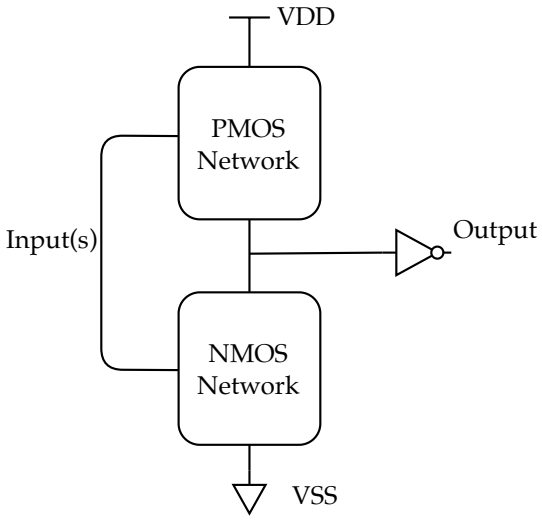


FIGURE 6.1: Complementary design in static CMOS gates

Static CMOS is a transistor design technique where the output is either strongly driven by logic '1' (VDD) or strongly driven by logic '0' (GND). Static here means that the output is a logic function of inputs, and, given stable inputs, output does not change over time. This type of circuit comprises of two networks : Pull up

network which is entirely made up of PMOS logic and pull down network which is entirely made up of NMOS logic. This arrangement is presented in Figure 6.1.

Logical expressions implemented by these networks make them operate in a mutually exclusive way. This mutually exclusive operation is the reason behind passing strong one and strong zero signal. Also, in a steady state, there is no direct path between supply voltage and ground, which ideally leads to zero short circuit current. This design style has full rail to rail swing and very high noise margin. Arbitrary Boolean logic can be implemented using static CMOS logic without using any additional gates (for example, NOT gate for complementing the inputs). The only disadvantage of this logic is that more than 2N transistors are involved in the design for N inputs. Some design styles exist where less than 2N transistors are used for a N input design. These design styles are discussed in upcoming sections. Figure 6.2 presents a sample implementation of a static CMOS AND gate where 6 transistors are used for a 2 input design (> 2N transistors).

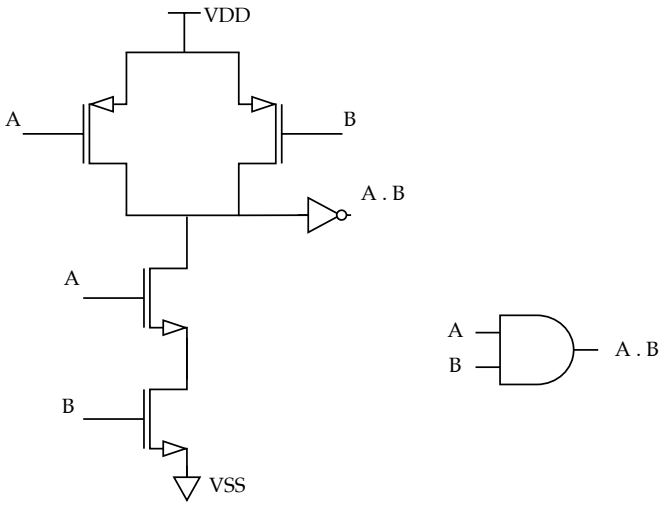


FIGURE 6.2: Static CMOS AND gate with logic symbol

6.2 Dynamic CMOS style [26][27][28]

Dynamic CMOS circuits, in contrast to static CMOS circuits, rely on temporary storage of signal values on the capacitance of high impedance nodes. These dynamic circuits work in two phases : pre-charge phase and evaluation phase. In pre-charge phase, high impedance nodes are charged to logic '1' (VDD). In evaluation phase, these charged nodes either get discharged or stay charged based on the logic function implemented. A logic function is implemented by the pull down network of NMOS transistors. There is no pull up PMOS network involving primary input signals in dynamic CMOS gates. Due to absence of pull up PMOS network, only N+2 transistors are required for N input Boolean logic, which also leads to smaller area than static CMOS circuits.

To put a circuit into pre-charge and evaluation phase, an additional signal is required. In synchronous designs, this signal will be a clock signal, whereas in asynchronous designs, this signal can be derived from the input/output detection

circuit. An abstract arrangement of pull down network along with additional pre-charge signal (Clk) is presented in Figure 6.3.

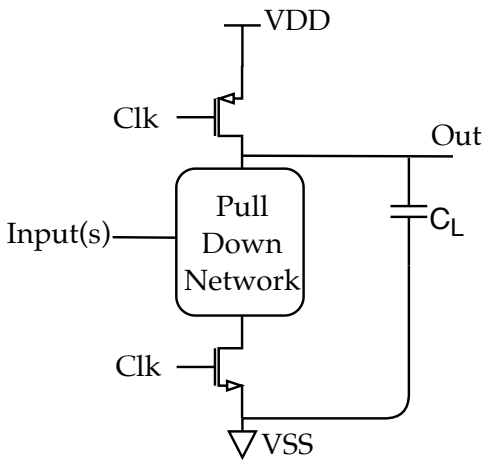


FIGURE 6.3: Abstract design of a dynamic CMOS gates

Dynamic logic provides full swing outputs and faster switching speeds due to reduced transistor count and load capacitance. It appears that power consumption will reduce due to reduced number of transistors, but, due to extra loading on "Clk" signal and continuous charging of each and every node in every clock cycle increases the overall power consumption. Apart from high power consumption, there are other disadvantages too. Once the output of a dynamic gate is discharged, it cannot be charged again until next pre-charge phase. There are multiple possibilities of discharging the dynamic gate before evaluation phase, one of them is stray noise. Also, because of leakage current through NMOS network, dynamic logic cannot hold the charge for longer duration of time, which simply translates into a minimum throughput requirement. This unwanted discharging of gates might lead to incorrect output generation. Another disadvantage is that the dynamic logic does not work well in low voltage region because of low noise immunity. To get an idea about designing dynamic CMOS gates, an AND gate is presented in Figure 6.4.

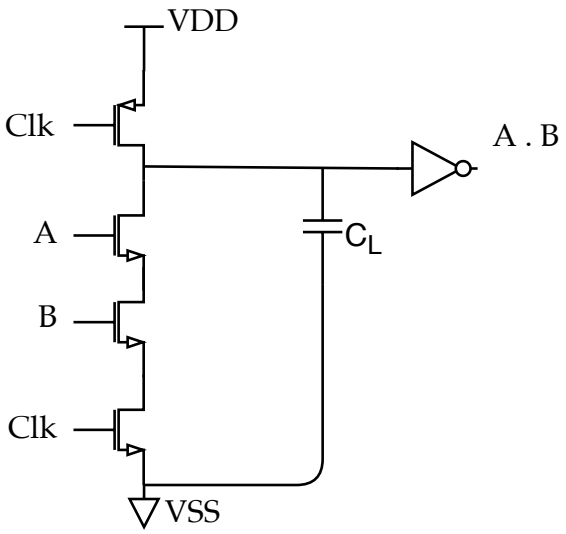


FIGURE 6.4: Dynamic CMOS AND gate

6.3 Pass Transistor Logic (PTL) [29][30][31][32]

In this cell design style, main focus is placed on reducing the transistor count by using primary inputs to drive gate terminals, as well as, source and drain terminals of a transistor. In contrast to PTL, primary inputs in CMOS logic only drive gate terminals. In this style, output port always has a low-impedance path to both supply rails (Power and GND) under all input combinations. An AND gate design using PTL is presented in Figure 6.5.

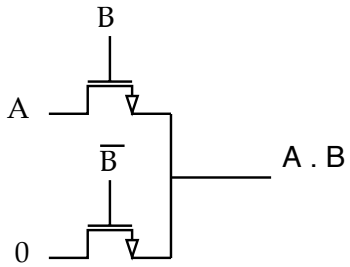


FIGURE 6.5: PTL AND gate

When input B is '1', top NMOS turns on and copies the input A to output port. When input B is '0', bottom NMOS turns on and passes logic '0' to the output port. The presence of the switch driven by $\bar{B}$ is essential to ensure that the gate is static and provides a low-impedance path to supply rails. The major advantages of this design style are lower transistor count, lower capacitance due to reduced number of transistors and no static power consumption. This type of logic only comprises of NMOS gates, which passes strong logic '0' but weak logic '1'. Due to weak passing of logic '1', full swing is not obtained at the output port. Absence of full swing makes this design style not suitable for operation in near-threshold and sub-threshold regions. Implementation of PTL gates require complementary signals which introduces additional transistors for performing inversion. Also, if multiple PTL logic stages are cascaded, then a level restoration circuit is required at the output to bring the voltage back to VDD.

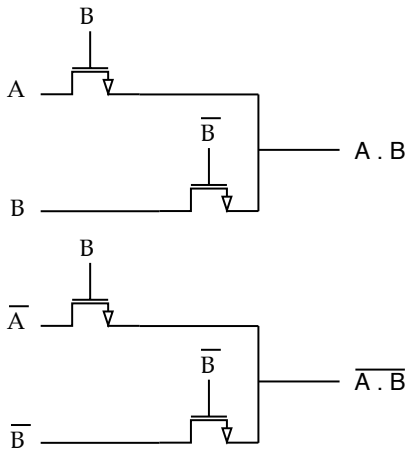


FIGURE 6.6: CPTL AND/NAND gate

To overcome these issues, different variants of PTL are invented. First variant is Complementary Pass Transistor Logic (CPTL) which provides both normal and complementary outputs, saving the inverters for obtaining complements of input

signals. A NAND/AND gate design using CPTL logic is presented in Figure 6.6.

This CPTL approach also suffers from the problem of passing weak logic '1' and requires level restoration circuit after certain cascaded stages. This approach is also not suitable for operation in sub-threshold or near-threshold region. This problem of weak passing of logic '1' can be overcome by the use of PMOS transistors. This usage of PMOS transistors leads to another approach known as transmission gates. In this approach, NMOS transistors are used to pass strong logic '0' and PMOS transistors are used to pass strong logic '1'. A transmission gate involves a pair of PMOS and NMOS transistor which controls the flow of input based on a control signal. This transmission gate is presented in Figure 6.7.

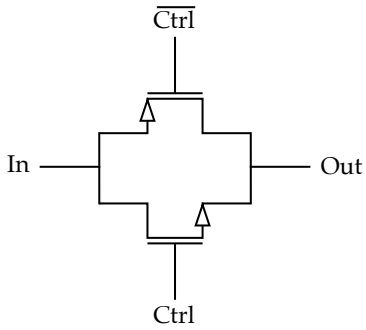


FIGURE 6.7: Transmission gate

Transmission gates pass strong logic levels but implementation of complex Boolean logic using them becomes complicated. Apart from transistors required for actual logic implementation, inverters are required to generate complementary signals. Sometimes, the number of transistors used for implementing actual logic is less than or comparable to the number of transistors required for generating complementary signals. This excessive use of transistors do not make this approach a feasible way for implementing basic gates or complex functions.

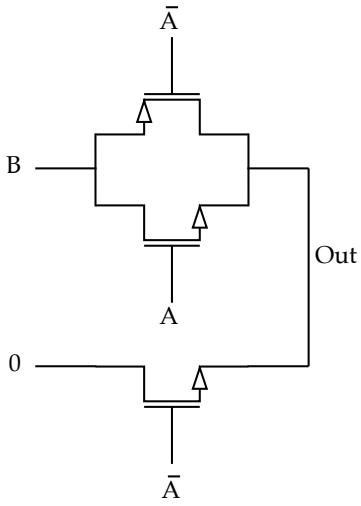


FIGURE 6.8: AND gate (no inverter at the output) implemented with transmission gates

Another disadvantage that comes with transmission gates is the absence of isolating/buffering stage in between the cascaded stages (like inverters in static

CMOS style). Although transmission gates pass strong logic '1' and logic '0', cascading multiple stages together increases the impedance to the power rails and slows down the speed of operation. The isolation stage comprising of inverters help in reducing the impedance to the power rails. This isolation stage also helps in providing different driving strengths to a cell. Usage of inverters in transmission gates for providing isolation and driving strength is possible, but, it complicates the logic implemention and introduces additional inverters. AND gate implementations without, and, with an isolation stage are presented in Figure 6.8 and Figure 6.9 respectively.

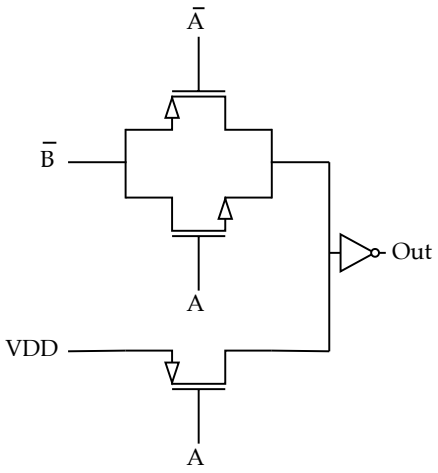


FIGURE 6.9: AND gate (inverter at the output) implemented with transmission gates

In the implementation of AND gate, 5 transistors are used for the design without isolation stage, and, 9 transistors are used for the design with isolation stage. In contrast to this, static CMOS style only requires 6 transistors for AND gate implementation. This example shows that transmission gates are not always a suitable choice for implementation.

Pass transistor logic and its variants mentioned above are all based on static logic, which means the output is connected to either "VDD" or "GND" for all input combinations. Apart from this static logic, dynamic counterparts of PTL also exist which suffer from the same problems as mentioned earlier for dynamic CMOS logic.

6.4 Choice of Transistor Design Style for Final Implementation

Low power operation and high noise margin are some of the important characteristics required for the design of Kogge-Stone adder. FDVS should be plausible which will ensure that the design operates in sub-threshold and near-threshold region. High fan-out gates are also present in the adder design which necessitates the usage of isolation stage for providing sufficient driving strength to the cells. All these requirements are best fulfilled by the static CMOS implementation. Although the number of transistors is more than other design styles, static nature of gates, FDVS and presence of isolation stage makes this design style suitable for cell implementation.

Authors of [12][19][20] provide some cell design styles and their comparisons which will be useful in implementing static CMOS cells for different type of pipelines discussed in earlier chapters. [12] presents the SAHB cell design style which is already discussed in Chapter 3. [19] provides a way to implement NCL directed Boolean expressions (with hysteresis) using static and semi-static style. Figure 6.10 presents a static version and Figure 6.11 presents a semi-static version of TH23 gate. TH23 gate is a 3 input gate which implements following Boolean expression : AB + AC + BC.

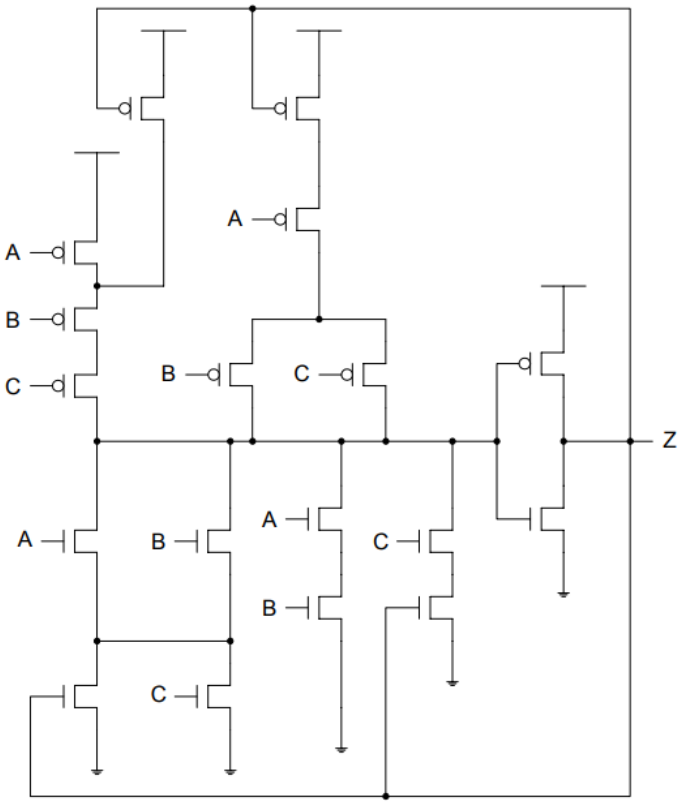


FIGURE 6.10: Static implementation of TH23 NCL gate [19]

In Figure 6.10, output of the gate is fed back to the input side and in Figure 6.11, a cross-coupled inverter arrangement is used to provide the necessary hysteresis behavior which is dictated by the NCL logic. If these strong-indicating gates are to be converted into weak-indicating ones, then, suitable changes are to be made. These changes involve removal of hysteresis mechanism and inclusion of reset circuitry. Authors of [20] provide a general arrangement for designing weak-indicating dual rail gates with reset mechanism. Instead of using the term "Reset", authors name the reset terminal as "Req" (Request). In addition to this, authors of [20] coined the term Pre-Charge Static Logic (PCSL) for these weak-indicating dual rail gates.

To compare the gates presented in [20] with other cell design approaches, some quantitative figures are also presented by the authors, which are shown in Figure 6.12. From the data presented in the figure, it can be observed that PCSL gates are superior to other approaches like DIMS, NCL1 (normal NCL implementation presented earlier) and NCL2 (NCL gates with fast-reset mechanism, not presented

here) in terms of energy, delay and chip area. *It should also be noted that PCSL gates do not implement hysteresis mechanism, whereas other cell design styles like DIMS and NCL have hysteresis mechanism in them. Drawing conclusions based on the comparison presented in the table does not truly reflect the energy, power and chip area of different design styles.*

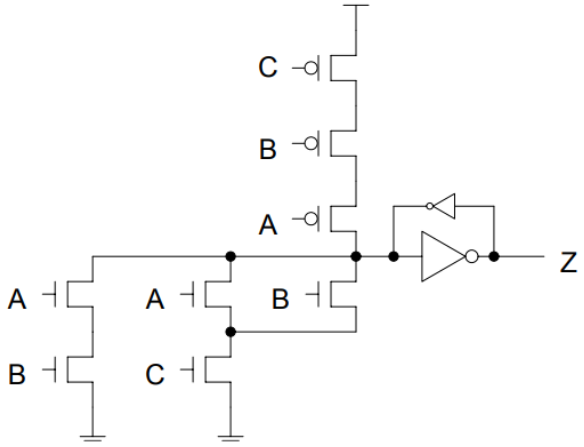


FIGURE 6.11: Semi-static implementation of TH23 NCL gate [19]

	E_{per} (normalized to PCSL (10^{-18}J))				Delay (normalized to PCSL (ns))				IC Area (normalized to PCSL (μm^2))			
	PCSL	DIMS	NCL1	NCL2	PCSL	DIMS	NCL1	NCL2	PCSL	DIMS	NCL1	NCL2
AND/NAND	1 (185)	3.2	0.9	1.2	1 (663)	3.2	1.0	1.2	1 (21)	4.5	1.6	1.6
OR/NOR	1 (185)	3.2	0.9	1.3	1 (661)	3.2	1.0	1.2	1 (21)	4.2	1.5	1.5
AO/AOI	1 (229)	6.8	2.0	2.2	1 (696)	7.0	2.2	2.3	1 (26)	6.8	3.1	3.3
OA/OAI	1 (229)	6.9	2.0	2.2	1 (696)	7.0	2.2	2.3	1 (26)	6.8	3.1	3.3
XOR/XNOR	1 (330)	1.8	2.0	2.1	1 (997)	2.1	2.3	2.3	1 (34)	2.9	3.1	3.3
MUX	1 (326)	1.8	2.0	2.1	1 (1021)	2.1	2.3	2.3	1 (32)	3.1	3.2	3.4
Average	1 (247)	4.0	1.6	1.9	1 (789)	4.1	1.8	1.9	1 (27)	4.7	2.6	2.7

FIGURE 6.12: Quantitative figures comparing PCSL approach with DIMS and NCL design styles [20]

6.5 Summary

In this chapter, different design styles were studied and some sample designs published by different authors were presented. After studying these styles, it can be said that static implementation is the best choice for cell design. Static design style supports low voltage operation and provides higher noise margin. In static implementation, making a choice between PTL and its variants, or, CMOS logic is highly based on the type of logical expressions being implemented. In all the practical designs, some gates are required to drive high fan-out paths which necessitates the usage of isolation/buffering stage. If PTL is chosen then number of transistors (for inverters) required to generate complementary signals and provide required driving strength is sometimes more than the transistors required for logic implementation. Considering all the points mentioned, it can be concluded that static CMOS style is the best way to design basic cells. Different static CMOS design styles mentioned by the authors will be used as references to implement basic cells for Kogge-Stone adder. Implementation of different cells, their schematics and optimizations performed are explained in detail in Appendix A.

Chapter 7

Implementation, Testing and Results

Previous chapters of this thesis present the advantages and disadvantages of asynchronous designs along with their well-known classification based on the delay models. A design decision was taken earlier to develop the baseline application (Kogge-Stone Adder) with QDI delay model which follows 4-phase signaling protocol. The choice of indicatability and timing assumptions involved were also discussed which resulted into three different templates. A choice of static CMOS design style for developing basic asynchronous cells was made in Chapter 6 to support the operation of design in sub-threshold and near-threshold regions.

This chapter presents the simulation results of the Kogge-Stone adder based on the design decisions made earlier. Preliminary simulations meant for the functional verification of the templates were carried out using VHDL. After these preliminary VHDL simulations, analog simulations were carried out to gather all the quantitative data meant for the analysis of the templates.

For performing analog simulations, 65-nm process technology was chosen and the transistors present in the Process Development Kit (PDK) were chosen to design the basic asynchronous cells. Upcoming sections of this chapter discuss about the tools involved in the design flow along with their software versions and licensing information. This chapter also presents information about the testing environment chosen to verify the designs. Simulation waveforms obtained from VHDL simulations were also presented along with data acquired from analog simulations.

7.1 Tools and their licensing details

The adder designs based on the templates discussed earlier were first modeled in VHDL. The VHDL used for the modeling follows the revised standard 1076-2002. The models defined in VHDL were then simulated in the simulation environment provided by Questa Sim-64 10.6e tool. This tool is developed by Mentor Graphics and is provided to the university under Europractice.

For performing analog simulations, a 65-nm process technology was chosen which is provided to the university by United Microelectronics Corporation (UMC) under Europractice. The UMC65 PDK comprises of configurable transistors which were used for designing basic cells. Cadence Virtuoso 6.1.7-64b was used for designing the schematics and layout of the cell, whereas, Cadence Spectre 18.10.077

was used for performing the analog simulations. Cadence tools were also provided to the university under Europractice.

Next step after cell designing is the conversion of VHDL model into a gate-level netlist which comprises of the basic asynchronous cells. *VHDL models were imported directly into Cadence environment but the netlist generation failed due to unknown reasons.* To overcome this problem, asynchronous cells were grouped together to form an asynchronous cell library. Cadence Liberate 16.14.122 is used for the characterization of asynchronous cells and creation of a cell library. Some of the asynchronous cells contained latching mechanism which lead to the incorrect characterization of cells. Although incorrect characterization results were given by the Liberate tool, a cell library was still generated which is good enough for the netlist generation.

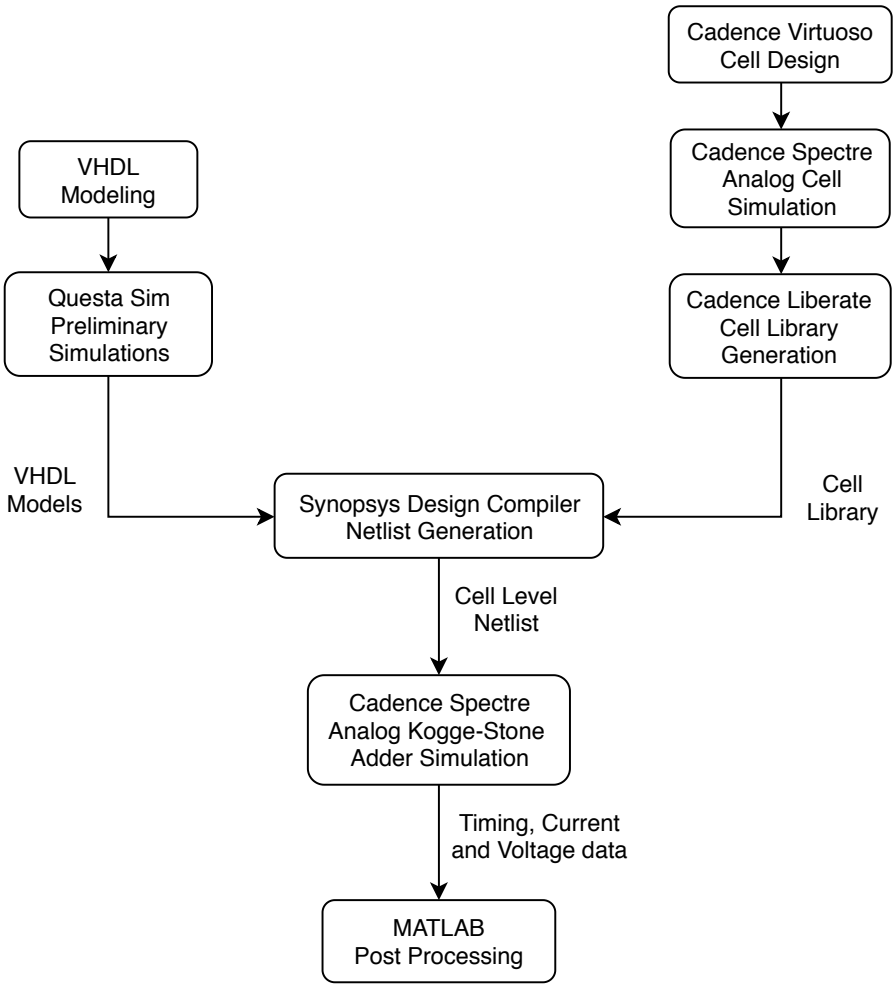


FIGURE 7.1: Step-by-Step usage of tools

After cell library generation, Synopsys Design Compiler O-2018.06 was used to generate gate-level netlist using VHDL models and asynchronous cell library. These generated netlists were then directly imported into Cadence simulation environment for carrying out analog simulations. The quantitative data (Time, Voltage and Current) obtained from the analog simulations was further processed by MATLAB R2018b. This processing by MATLAB involved calculation of average current, power and energy. This step-by-step usage of tool is also presented in Figure 7.1. All the tools used in this thesis work are obtained from the Europractice and more

licensing information on this can be obtained from [33].

7.2 Testing Environment

Testing and verification of different designs discussed earlier requires a uniform testing strategy for performing a fair comparison between the designs. This testing strategy involves a uniform testing environment which provides similar simulation parameters to all the designs. These simulation parameters include load capacitance, rise and fall time of input signals, supply voltage, body bias voltage and junction temperature. This testing environment for asynchronous and synchronous designs is presented in Figure 7.2 and Figure 7.3 respectively.

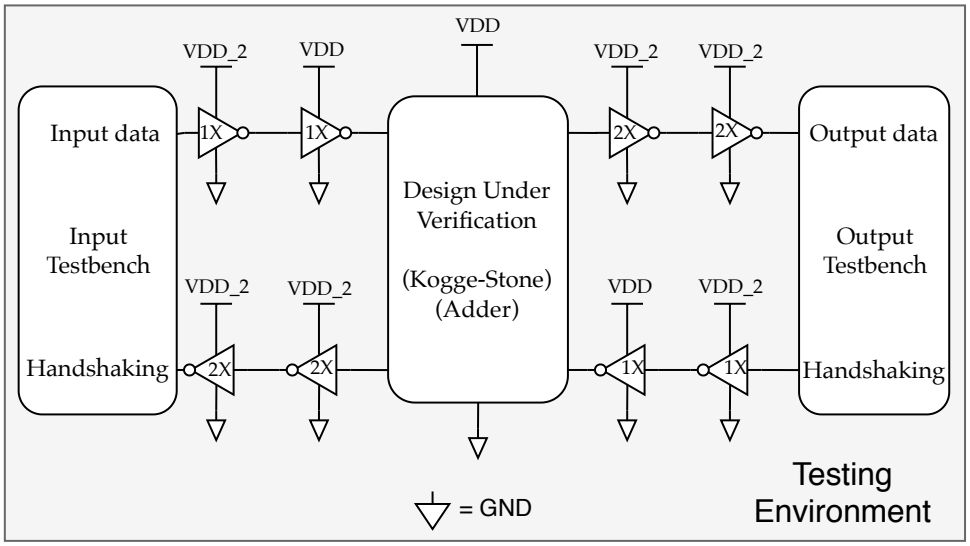


FIGURE 7.2: Testing environment for asynchronous designs

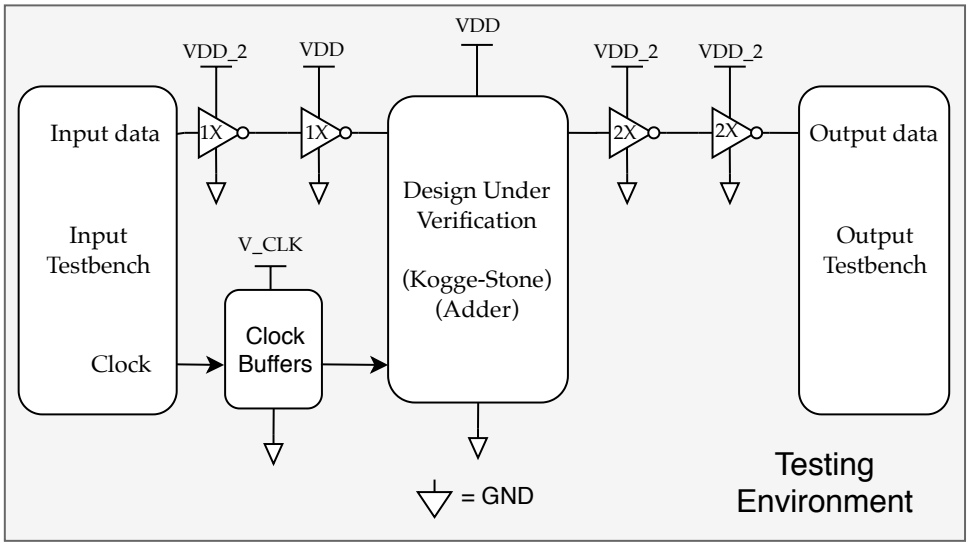


FIGURE 7.3: Testing environment for synchronous design

These testing environments have inverters on both the input and output terminals of the Design Under Verification (DUV). Inverters with 1X driving strength are present on the input side, whereas, inverters with 2X driving strength

are present on the output side of the design. Inverters placed on the output side provide necessary load to the DUV, whereas, inverters on the input side are meant for doing accurate power measurements (to consider current flowing through the input terminals). It should be noted that different power supply is provided to the inverters which make sure that power measurements are done accurately for the DUV. For synchronous testing environment, clock buffers have their separate power supply (V_CLK) which will be useful in identifying the power consumption by the clock circuitry alone. These testing environments also include input and output testbenches which are working independently on input data generation and output data monitoring respectively. Input testbench reads input patterns (discussed later) from an input file and provides it to the DUV in desired dual-rail or single-rail format. Output testbench keeps on monitoring the "SUM" and "CARRY_OUT" signals coming out from the DUV and checks them against reference result values stored in another file. This helps to ensure that the design is functionally correct. For analog simulations, these testbenches are written in Verilog-A, whereas for preliminary simulations in Questa Sim, these testbenches are written in VHDL.

Asynchronous adder designs make use of asynchronous cells which have minimum size transistors (as dictated by the process technology) on the input side of the gate. The output of these cells is generated by an inverter (refer to static CMOS gates in Chapter 6) which is required to provide necessary driving strength to the cells. This output inverter is scaled accordingly to provide cells with 1X, 2X and 4X driving strengths. All the asynchronous and synchronous cells used in the different variants of adder are designed at the layout level in Cadence. These cells contain information about the parasitics which is helpful in performing near-accurate analog simulations. *It should be noted that Kogge-Stone adder is not designed at the layout level, rather the cells are developed at layout level. These cells are used to perform gate-level simulations.*

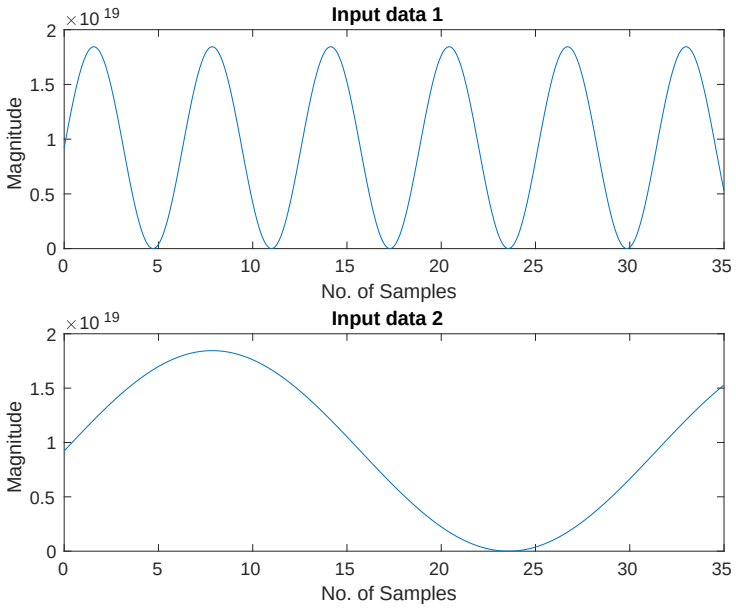


FIGURE 7.4: Input test patterns for testing the adder design

To test the functionality and gather quantitative data for a 64-bit adder, it is not possible to provide exhaustive input patterns. Instead, two sinusoidal signals of

different frequency with magnitude spanning from 0 to $2^{64}-1$ were chosen which try to cover the extreme ends for this adder. These test patterns try to cover all the path and excite most of the nodes present in the design. These test patterns are generated in MATLAB and are displayed in Figure 7.4. These input patterns are provided to the DUV with a rise time and fall time of 20ps. Also, the body bias terminals (meant for adjusting the threshold voltage) are connected to the respective power rails. The body bias terminal of PMOS transistors are connected to VDD, and, the body bias terminal of NMOS transistors are connected to GND. The junction temperature is set to 27°C in all the simulations.

7.3 Preliminary VHDL Simulations

Designs modeled in VHDL were simulated using Questa Sim to verify certain aspects of asynchronous circuit designing. Apart from functional correctness, these aspects include verification of 4-phase signaling protocol, hysteresis mechanism and random feeding of input signals by the testbench. Ensuring functional correctness for the adder design is already taken care of by the output testbench. This testbench continuously monitors the output signals and verifies them against reference outputs. These reference outputs are generated by MATLAB for given set of inputs and are stored in a file.

To verify the 4-phase signaling protocol, the adder design based on template 1 was chosen and simulated for the input patterns discussed earlier. The waveform presented in Figure 7.5 shows the 4-phase signaling protocol followed by the design. Cursor 1 points to a time instance when the adder is in null state (READY_TO_ACCEPT_INPUT (Ack_i) = 0). In this state, null inputs are provided to the design. At the time instance pointed by cursor 2, design is ready to accept valid data (READY_TO_ACCEPT_INPUT (Ack_i) = 1). When the adder has accepted valid data, it requests for null data from the input testbench (READY_TO_ACCEPT_INPUT (Ack_i) = 0) which is depicted by cursor 3. When the adder has processed the null data, it can now start new cycle of 4-phase signaling protocol by requesting new valid data (READY_TO_ACCEPT_INPUT (Ack_i) = 1) form the testbench, as presented by cursor 4. The explanation of this 4-phase signaling protocol has been presented in Chapter 2.

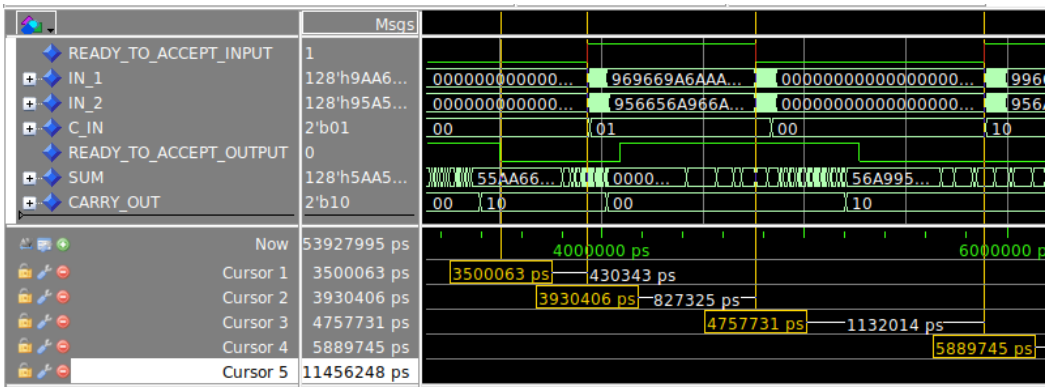


FIGURE 7.5: 4-phase signaling protocol followed by the design

For the verification of hysteresis mechanism, functioning of a 2-input Muller C element (present in registers) is considered, refer Figure 7.6. The two input terminals

are named as “ctrl_int_reg” and “IN_1(126)”, whereas, the output terminal is named as “temp_input1(126)”. These names originate because the Muller C element is embedded in a larger design. When input logic levels are different, as shown by cursor 6, output holds its previous logic level (logic ‘0’). At the time instance depicted by cursor 7, both the inputs go to same logic level (logic ‘1’) which changes the output to logic ‘1’. At the time instance pointed by cursor 8, input logic level changes again, but the output holds its state (logic ‘1’). Output goes back to logic ‘0’ when both the inputs go to logic ‘0’, as depicted by cursor 9.

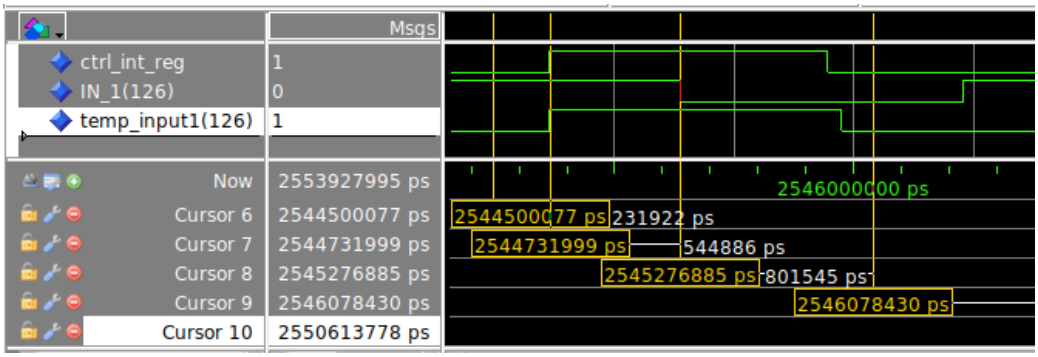


FIGURE 7.6: Hysteresis behavior followed by a Muller C element

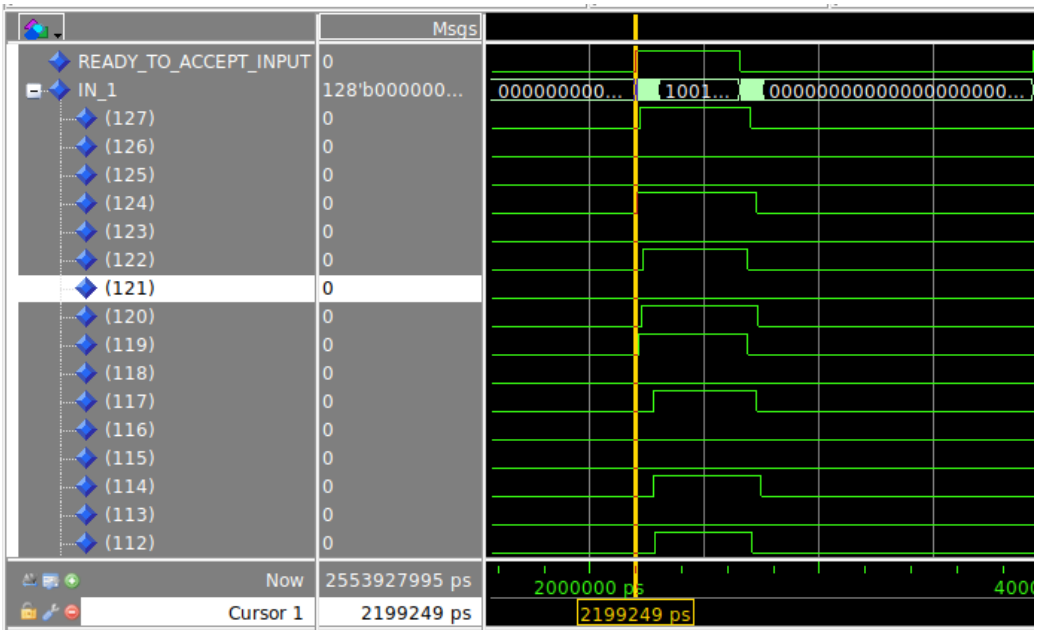


FIGURE 7.7: Random input arrival to the adder design

Random feeding of input signals to the design can be seen in Figure 7.7. When the design is ready to accept valid data (READY_TO_ACCEPT_INPUT (Ack_i) = 1), testbench starts providing input data at random time instances. All the signals in IN_1 vector turn valid at random time instances, as depicted by cursor 1 in Figure 7.7. Even though the inputs are arriving at random time, correct output is generated, which validates the functioning of the templates. Multiple simulations were carried out by considering large ranges of delay variations in the input arrival for template verification. For more information on generation of inputs at random time instances, please refer Appendix B.

All these aspects were also verified for other variants of Kogge-Stone adder. From the simulations which were carried out for all other variants, it can be established that these asynchronous models work perfectly and can be ported to Cadence environment for carrying out analog simulations.

7.4 Analog Simulations and Result Analysis

After performing preliminary simulations, VHDL models were ported to Cadence environment as per the tool flow mentioned in Figure 7.1. All the three variants of asynchronous adder and their synchronous counterpart were simulated at a nominal voltage of 1.2V and sub-threshold voltage of 0.3V. Input patterns discussed earlier for VHDL simulations were also used for the analog simulations. The testing environments presented in Figure 7.2 and Figure 7.3 were followed with a slight exception in the behavior of these testbenches. Instead of providing data at random instances of time, testbenches will now provide data synchronously to the DUVs. This synchronous arrival of data is done to make sure that the comparison between all the adder variants is fair.

Simulation results presented in this section make use of some symbols to indicate about the parameters which are acquired from the simulations. These symbols along with their formulas and meanings are presented in Table 7.1. Simulation results for all the three adder variants at nominal and sub-threshold voltage at 27°C for Typical Typical (TT) process corner are presented in Table 7.2 and Table 7.3.

Symbols	Formula	Meaning
d_{min}		Minimum delay between two valid inputs provided to the design
d_{max}		Maximum delay between two valid inputs provided to the design
d_{avg}		Average delay between two valid inputs provided to the design
V		Supply voltage
f_{min}	$(1/d_{max})$	Minimum frequency of operation
f_{max}	$(1/d_{min})$	Maximum frequency of operation
f_{avg}	$(1/d_{avg})$	Average frequency of operation
I_{avg}		Average current withdrawn
P_{avg}	$(V * I_{avg})$	Average power consumption
t		Simulation runtime
E_{avg}	$P_{avg} * t$	Average energy consumption
P_{avg}/MHz	P_{avg}/f_{avg}	Average power consumed per MHz
N		Number of operations carried out
E_{avg}/Op	E_{avg}/N	Average energy consumed per operation

TABLE 7.1: Symbols and their meanings

Parameters	Template 1	Template 2	Template 3	Units
d_{min}	5.884	2.920	2.244	ns
d_{max}	6.806	3.032	2.313	ns
d_{avg}	**6.392**	**2.971**	**2.271**	ns
f_{min}	146.929	329.728	432.338	MHz
f_{max}	169.952	342.442	445.632	MHz
f_{avg}	**156.445**	**336.485**	**440.334**	MHz
I_{avg}	2.6	4.6	4.3	mA
P_{avg}	**3.12**	**5.52**	**5.16**	mW
t	2.2739	1.0768	0.8313	us
E_{avg}	7.094	5.944	4.289	nJ
P_{avg}/MHz	**19.94**	**16.4**	**11.71**	uW/MHz
N	351	351	351	-
E_{avg}/Op	**20.21**	**16.93**	**12.21**	pJ

TABLE 7.2: Results for asynchronous adder variants simulated at 1.2V (TT process corner), 27°C

Parameters	Template 1	Template 2	Template 3	Units
d_{min}	17.249	8.902	7.340	us
d_{max}	20.149	9.159	7.410	us
d_{avg}	**18.847**	**9.013**	**7.370**	us
f_{min}	49.630	109.182	134.952	KHz
f_{max}	57.974	112.334	136.239	KHz
f_{avg}	**53.058**	**110.950**	**135.685**	KHz
I_{avg}	710.62	1058.3	1184.7	nA
P_{avg}	**213.186**	**317.49**	**355.41**	nW
t	6.681	3.24	2.66	ms
E_{avg}	1.4243	1.0286	0.9454	nJ
P_{avg}/MHz	**4.018**	**2.861**	**2.619**	uW/MHz
N	351	351	351	-
E_{avg}/Op	**4.057**	**2.930**	**2.693**	pJ

TABLE 7.3: Results for asynchronous adder variants simulated at 0.3V (TT process corner), 27°C

From the results presented above, it can be observed that the adder design based on Template 1 is slower (refer to parameter "f_{avg}") in comparison to the adder designs based on Template 2 and Template 3. This slow performance is observed for both nominal and sub-threshold voltage of 1.2V and 0.3V respectively. This slow

performance can be explained by the argument that asynchronous cells used in the design based on Template 1 have hysteresis mechanism. This hysteresis mechanism provides indicatability property but leads to a very complex cell structure involving bigger stacks of PMOS and NMOS transistors, when compared to the cells without hysteresis mechanism. This stacking of transistors lead to slower operation of the cells, and thus, slower operation of the adder design.

When adder designs based on Template 2 and Template 3 are compared in terms of speed, adder design based on Template 3 is faster. This faster operation is observed because of the absence of reset functionality (lesser transistors in basic asynchronous cells) and the absence of DPCD circuitry. Absence of DPCD circuitry reduces the capacitance on the internal nodes of the functional block, thus, leading to a faster speed of operation. This faster speed is observed for both nominal and sub-threshold voltages.

In terms of power (refer to parameter "P_{avg}/MHz"), the adder design based on Template 3 consumed least amount of power. Also, energy consumed per operation (refer to parameter "E_{avg}/Op") for Template 3 design is lower when compared to other adder variants based on Template 1 and Template 2. These lower power and energy figures are observed due to simple cell structure, lesser number of transistors in the cells and lesser capacitance on internal nodes (absence of DPCD circuitry). These results show that Template 3 is capable of providing designs with high performance and low power figures. Extensive simulations for the adder design based on Template 3 were carried out for different voltage levels in sub-threshold and near-threshold regions. The process corner and junction temperature for these extensive simulations were kept same. The results for these simulations are presented in Table 7.4 and Table 7.5.

Parameters	**0.5 V**	**0.45V**	**0.4V**	**0.35V**	**Units**
d_{min}	112.34	271.8	754.4	2144	ns
d_{max}	113.79	274.54	763.47	2167	ns
d_{avg}	**112.96**	**273.12**	**758.78**	**2155**	ns
f_{min}	8.788	3.642	1.309	0.461	MHz
f_{max}	8.901	3.679	1.325	0.466	MHz
f_{avg}	**8.852**	**3.661**	**1.318**	**0.464**	MHz
I_{avg}	43.077	16.864	6.008	2.285	uA
P_{avg}	**21.539**	**7.588**	**2.403**	**0.799**	uW
t	40.554	98.018	272.59	832.53	us
E_{avg}	873.49	743.76	655.03	665.19	pJ
P_{avg}/MHz	**2.433**	**2.072**	**1.823**	**1.721**	uW/MHz
N	351	351	351	351	-
E_{avg}/Op	**2.488**	**2.118**	**1.866**	**1.895**	pJ

TABLE 7.4: Results for asynchronous adder based on Template 3 simulated at different voltages (TT process corner), 27°C

Parameters	0.3 V	0.25V	0.2V	0.15V	Units
d_{min}	7.340	23.95	78.32	308.56	us
d_{max}	7.410	24.17	79.1	325.28	us
d_{avg}	**7.370**	**24.06**	**78.58**	**317.28**	us
f_{min}	134.952	41.37	12.64	3.074	KHz
f_{max}	136.239	41.75	12.76	3.240	KHz
f_{avg}	**135.685**	**41.56**	**12.72**	**3.151**	KHz
I_{avg}	1.184	0.753	0.623	0.661	uA
P_{avg}	**355.41**	**188.25**	**124.6**	**99.15**	nW
t	2.66	8.655	28.3	114.1	ms
E_{avg}	945.4	1629.3	3526.1	11313	pJ
P_{avg}/MHz	**2.619**	**4.529**	**9.795**	**31.46**	uW/MHz
N	351	351	351	351	-
E_{avg}/Op	**2.693**	**4.641**	**10.04**	**32.23**	pJ

TABLE 7.5: Results for asynchronous adder based on Template 3 simulated at different voltages (TT process corner), 27°C

These figures obtained for asynchronous design based on Template 3 indicate that the energy consumed per operation decreases initially with the decrease in supply voltage. But as the supply voltage is decreased further, the energy consumed per operation increases. The power (P_{avg}/MHz) also follows the same trend.

Parameters	1.2 V	0.5 V	0.45V	0.4V	0.35V	Units
Frequency	440.334	8.852	3.661	1.318	0.464	MHz
I_{avg} (Clock)	390.603	2.832	1.04	0.335	0.1049	uA
I_{avg} (Rest)	788.9	6.447	2.398	0.786	0.2606	uA
I_{avg} (Total)	1179.503	9.279	3.438	1.121	0.3655	uA
P_{avg} (Clock)	468.72	1.416	0.468	0.134	0.036	uW
P_{avg} (Rest)	946.68	3.223	1.079	0.314	0.091	uW
P_{avg} (Total)	1415.4	4.639	1.547	0.448	0.127	uW
E_{avg}	1.128	0.1839	0.148	0.119	0.096	nJ
N	351	351	351	351	351	-
E_{avg}/Op	3.214	0.524	0.422	0.339	0.273	pJ
Clock Power %	33.11	30.52	30.25	29.91	28.34	%

TABLE 7.6: Results for synchronous adder simulated at different voltages (TT process corner), 27°C

A comparison of asynchronous adder based on Template 3 is also performed with its synchronous counterpart. The synchronous design is simulated at all the voltage levels which were earlier considered for the asynchronous design. The

frequency for synchronous design is set to the average frequency of operation obtained earlier for asynchronous design. The results obtained are tabulated in Table 7.6 and Table 7.7. In these tables, value of current (power) is separately mentioned for clock circuitry and rest of the elements present in the design. Also, Clock Power % represents the percentage of power consumed by the clock circuitry.

Parameters	**0.3 V**	**0.25 V**	**0.2V**	**0.15V**	**Units**
Frequency	135.985	41.56	12.72	3.151	KHz
I_{avg} (Clock)	27.86	8.625	3.332	1.714	nA
I_{avg} (Rest)	81.42	34.5	20.45	14.47	nA
I_{avg} (Total)	109.28	43.125	23.782	16.184	nA
P_{avg} (Clock)	8.358	2.156	0.666	0.257	nW
P_{avg} (Rest)	24.426	8.625	4.09	2.17	nW
P_{avg} (Total)	32.784	10.781	4.756	2.427	nW
E_{avg}	84.808	91.052	131.23	270.35	pJ
N	351	351	351	351	-
E_{avg}/Op	0.241	0.259	0.373	0.77	pJ
Clock Power %	25.49	19.99	14	10.59	%

TABLE 7.7: Results for synchronous adder simulated at different voltages (TT process corner), 27°C

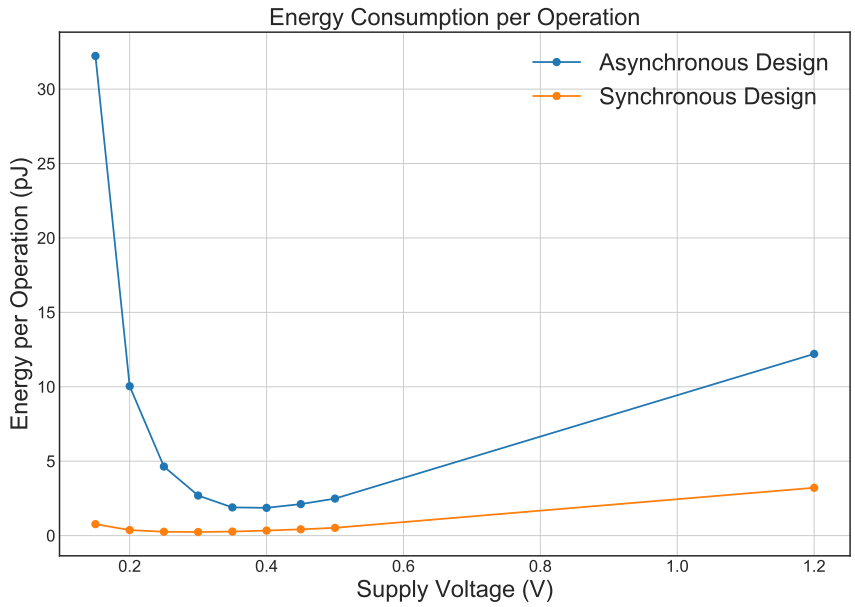


FIGURE 7.8: Energy consumed per operation for synchronous and asynchronous design

The figures obtained for synchronous design in Table 7.6 and Table 7.7 indicate that the energy consumed per operation decreases initially with the decrease in

supply voltage. But as the supply voltage is decreased further, the energy consumed per operation increases. This trend in energy consumption is similar to the figures obtained from asynchronous design. In initial chapters it was discussed that clock circuitry consumes 30%-50% of the total power of a synchronous design. This fact was verified with the simulations carried out for synchronous design. At nominal voltage, clock circuitry consumed ~33.2% of the total power, whereas, at sub-threshold voltage of 0.3V, clock circuitry consumed ~25.5% of the total power.

A plot presenting the comparison between the energy consumed per operation for synchronous and asynchronous design is presented in Figure 7.8. From this plot, it can be observed that the difference in energy consumption for both the designs decreases initially. But, as the supply voltage is decreased further, this difference increases which indicates that there is no cross-over point for this design. The minima for asynchronous design was observed at 0.4 V , whereas, the minima for synchronous design was observed at 0.3V. Below 0.25 V, the energy consumption per operation for asynchronous design starts increasing rapidly, and, the energy consumption at 0.15 V is ~2.7X more than the energy consumption at nominal voltage of 1.2V.

7.5 Summary

This chapter started with the introduction of tooling involved for the asynchronous circuit designing along with the relevant licensing information. A flow of tool usage was also presented to guide the reader through the entire designing process. Different test environments for synchronous and asynchronous designs were also discussed along with the test patterns for testing the functionality of the circuit. These test environments were also used for gathering quantitative data for both synchronous and asynchronous designs.

Preliminary simulations were carried out in VHDL to validate the behavior of the asynchronous pipelines. These VHDL models were then simulated in the Cadence environment. From these simulations, it was observed that the design based on Template 3 performs better, both in terms of power and speed, in comparison to the adder variants based on Template 1 and Template 2. This comparison between the templates show that Template 3 can be used to develop robust, high performance and low power consuming asynchronous circuits.

The discussions made about the power consumed by the clock circuitry was also verified. Authors of [20] mentioned a cross-over point for synchronous and asynchronous designs at lower supply voltages. An attempt was made to achieve that cross-over point, but it was unsuccessful.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

Chapters presented earlier in this thesis work provide a systematic top-bottom design approach for developing asynchronous circuits. The discussion of this design approach started with the selection of signal encoding, handshaking protocol and signaling logic which is present between the asynchronous blocks. This selection was made keeping in mind that the resultant design should be easy to develop and is robust in nature. Multiple cell design styles were also discussed and the best design style was chosen to achieve robust nature. Based on the discussions and design choices mentioned earlier, three asynchronous pipeline templates were selected. These asynchronous pipeline templates were studied and evaluated based on the complexity of their join and fork mechanism. This evaluation was further carried out by implementing and simulating the baseline design of an unsigned 64-bit Kogge-Stone adder. The design decisions made and the quantitative figures acquired from the simulations were studied.

Signal encoding chosen to represent the data affects the overall complexity and the power consumption of the design. Delay-insensitive dual rail encoding chosen to achieve robustness against PVT variations leads to an increase in the hardware usage (higher transistor count and higher interconnect density) and power consumption by ~2X times. Along with signal encoding, signaling logic also affects the quantitative figures. 2-phase logic leads to faster designs but it is hardware intensive when compared to 4-phase logic. Designs based on 4-phase logic run at approximately half the speed due to the presence of null cycle (return to zero part of 4-phase signaling logic). If a design is accepting valid data at the rate of 1 MHz, then it is actually operating at 2 MHz (1 MHz valid data + 1 MHz null data).

QDI delay model provides robustness to the designs similar to the DI delay model. QDI designs provide insensitivity to PVT variations and also reduce the hardware usage by making use of isochronic forks. Although, isochronic fork is a harmless simplification, isochronicity of forks cannot be guaranteed with process variations. Extensive post-layout timing analysis along with some timing assumptions are required to ensure the QDI nature of the design.

Different cell design styles were also discussed earlier and it was concluded that static CMOS style is best suited for the operation of design in low voltage region. This design style provides noise immunity which enables the operation of design in sub-threshold and near-threshold regions. Other design styles like dynamic logic, PTL, etc. were also discussed, but they were rejected due to minimum throughput

requirements, less noise immunity and higher transistor count.

Template 3 discussed earlier in Chapter 5 provides an area and power efficient technique to design asynchronous circuits. Addition of configurable delay element in the control circuitry makes this template more robust by giving an opportunity to dynamically increase the time frame (t_{settle}). This increase in the time frame can be used to accommodate the behavior of asynchronous design under harsh PVT variations.

Hysteresis is an important mechanism which helps in establishing the indicatability property for functional blocks. However, the presence of this mechanism at the cell level slows down the speed of operation and increases the power consumption of the design. Hence, different techniques were employed to remove hysteresis mechanism. The designs based on these techniques are as robust as the designs with hysteresis mechanism. Functional blocks used for Template 3 designs need not have hysteresis and reset mechanism. Atomic entity for these functional blocks can be dual rail gates based on weakly-indicating logic. Implementing dual rail gates with weakly-indicating logic reduces the transistor count drastically. This reduced transistor count increases the speed of operation and lowers down the power consumption, as seen in Chapter 7. These weakly-indicating dual rail gates can also be developed by using synchronous cells obtained from the standard cell library, which effectively reduces the design efforts and the time for development.

Tool flow described earlier mentions the use of pre-existing synchronous CAD tools which makes the designing of asynchronous circuits simple and straightforward. However, some efforts are still required to structurally model the asynchronous circuit in VHDL or Verilog. The tool flow after modeling is similar to the flow followed by synchronous designs. Design of functional blocks which use weakly-indicating dual rail gates can be assisted by the design compilers, for e.g. Synopsys Design Compiler. Since, the weakly-indicating dual rail gates are based on synchronous cells, the design compiler can make use of the characterization data available for synchronous cells to appropriately select the driving strength of the cells. This selection of driving strength will make sure that the design operates with optimum power and performance figures with appropriate rise and fall time parameters.

VHDL and analog simulations carried out in Chapter 7 are helpful in deciding the template for asynchronous designs. Based on the correct output generation for random arrival of inputs, it can be concluded that the designs based on Template 3 can be as robust as QDI designs. Adder design based on this template was faster than the designs based on other templates. In terms of power consumption per MHz also, Template 3 outperformed other templates. This comparison between the templates show that Template 3 can be used to develop robust, high performance and low power consuming asynchronous circuits. It was discussed earlier that the clock circuitry in a synchronous design consumes around 30%-50% of the total power. Based on the synchronous design simulated, this fact was verified. An attempt to find a cross-over point between the synchronous and asynchronous design (based on Template 3) at lower supply voltages was made. As the voltage was dropped from nominal supply voltage, the gap between the energy per operation for synchronous and asynchronous design widens. The attempt to find

the cross-over point failed.

This thesis work started with the discussion of DI/QDI designs. Based on the hardware usage and slower performance of these designs, some assumptions and changes (removal of hysteresis, addition of reset mechanism) were made and based on those assumptions Template 2 was obtained. This Template 2 ensures robustness in the design behavior but considers some safety margins. Some additional timing assumptions (time frame t_{settle}) were made on the Template 2 and after those assumptions, Template 3 was obtained. Template 3 removes the DPCD and reset circuitry which were present in Template 2. Using Template 3, the power and performance figures were improved further. These additional assumptions provided favorable results at the loss of QDI behavior of the design. If this trend is followed and more assumptions are made, then at some point the new design template obtained will be similar to the synchronous designs. Based on all the discussions and quantitative figures presented in previous chapters, it is evident that synchronous design outperforms asynchronous designs, both in terms of power and performance. This comparison assumed that all the inputs to the design are arriving at the same time. Although, the synchronous design is better performing, it fails when input arrival is random in nature. In comparison to synchronous design, asynchronous design works perfectly for random input arrivals. This indicates that the area of application of both the design classes is different and performing a comparison between them on common grounds is very difficult. If a comparison still needs to be made between synchronous and asynchronous designs, then asynchronous designs might outperform synchronous designs when input arrival is in bursts. In scenarios where no clock and power gating is considered, power consumption of synchronous circuits in idle state (no burst data received) will be more due to the continuous ticking of clock signal. In contrast to this, the only power consumed in idle state for asynchronous circuits will be leakage power. This leakage power is expected to be much less when compared to the power consumed by clock circuitry.

8.2 Future Work

The work presented in this thesis provides basic concepts for asynchronous circuit designing, and, a baseline design to prove and analyze those concepts. The time frame available for this thesis work was not enough to cover the wide scope of asynchronous design domain. There are still a few more tasks left which are to be implemented to provide an overall view to the design approach presented in this thesis. These tasks are listed below:

- Template 3 presented earlier can further be modified to speed up the design. This increase in performance will be achieved by reducing the time frame (t_{settle}). This reduction is achieved by modifying the delay path used to calculate t_{settle} (keeping the complexity of functional block same). This new template design is presented in Figure 8.1.
- In the current adder design, basic cells are designed at the layout level. These cells are then used to perform gate-level analog simulations. In the next step, interconnects and wirings between these cells should be done to get accurate quantitative figures.

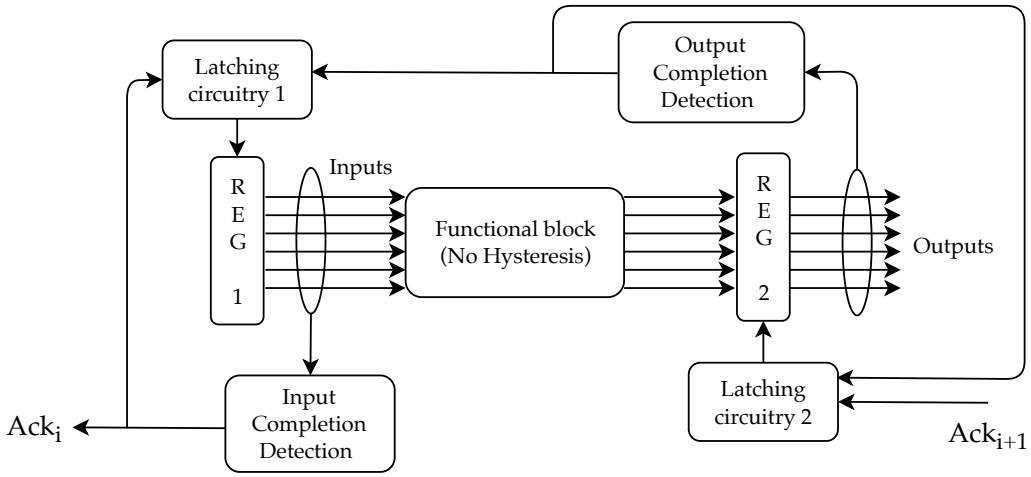


FIGURE 8.1: Modified template 3 to achieve faster speed of operation

- The Kogge-Stone adder design presented earlier is only tested for TT process corner at 27°C. This design should be simulated at all the process corners, and, at lower and higher temperatures to fully understand the functioning of the design. These simulations will help in establishing the robustness of the design against PVT variations.
- In the previous chapter, a speculation was made that the asynchronous circuits might consume less power than their synchronous counterparts when the inputs are arriving in bursts. This speculation should be verified with a suitable testing environment.
- In the current design, true and false rails of the dual rail gates are implemented separately by synchronous cells from standard cell library. An effort should be placed to combine those individual cells and create a complex cell which reduces the number of transistors involved.
- A sufficiently complex circuit which works with random input arrival should be considered and designed till the layout level to fully understand the advantages of asynchronous design domain.

Bibliography

[1] Marcos Ferretti Peter A. Beerel Recep O. Ozdag. *A Designer's Guide to Asynchronous VLSI*. Cambridge University Press, New York, 2010.

[2] George Conover. "Asynchronous vs. Synchronous Microporcessors". In: 2015.

[3] Jingwei Lu, Wing-Kai Chow, and Chiu-Wing Sham. "Fast power-and slew-aware gated clock tree synthesis". In: *IEEE transactions on very large scale integration (VLSI) systems* 20.11 (2012), pp. 2094–2103.

[4] Jens Sparsϕ. *Asynchronous Circuit Design, A Tutorial*. Technical University of Denmark, 2006.

[5] A. Takamura et al. "TITAC-2: an asynchronous 32-bit microprocessor based on scalable-delay-insensitive model". In: *Proceedings International Conference on Computer Design VLSI in Computers and Processors*. 1997, pp. 288–294. DOI: 10.1109/ICCD.1997.628881.

[6] M. Ferretti and P. A. Beerel. "Single-track asynchronous pipeline templates using 1-of-N encoding". In: *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*. 2002, pp. 1008–1015. DOI: 10.1109/DATE.2002.998423.

[7] *ARM7TDMI Technical Reference Manual*. URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0210c/CACBCAAE.html (visited on 04/01/2019).

[8] D. W. Lloyd and J. D. Garside. "A practical comparison of asynchronous design styles". In: *Proceedings Seventh International Symposium on Asynchronous Circuits and Systems. ASYNC 2001*. 2001, pp. 36–45. DOI: 10.1109/ASYNC.2001.914067.

[9] Ricardo Aquino Guazzelli. "QDI Asynchronous Design and Voltage Scaling". MA thesis. Pontificial Catholic University of Rio Grande Do Sul, 2017.

[10] Sandeep Garg and Tarun Kumar Gupta. "Low power domino logic circuits in deep-submicron technology using CMOS". In: *Engineering Science and Technology, an International Journal* 21.4 (2018), pp. 625 –638. ISSN: 2215-0986. DOI: https://doi.org/10.1016/j.jestch.2018.06.013. URL: http://www.sciencedirect.com/science/article/pii/S2215098617316257.

[11] R. A. Guazzelli, M. T. Moreira, and N. L. V. Calazans. "A comparison of asynchronous QDI templates using static logic". In: *2017 IEEE 8th Latin American Symposium on Circuits Systems (LASCAS)*. 2017, pp. 1–4. DOI: 10.1109/LASCAS.2017.7948103.

[12] K. Chong et al. "Sense Amplifier Half-Buffer (SAHB) A Low-Power High-Performance Asynchronous Logic QDI Cell Template". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.2 (2017), pp. 402–415. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2016.2583118.

[13] Karl M. Fant and Scott A. Brandt. *Null Convention Logic by Theseus Logic*. 1997.

[14] S. Smith and J. Di. *Designing Asynchronous Circuits using NULL Convention Logic (NCL)*. Synthesis Lectures on Digital Circuits and Systems. Morgan & Claypool Publishers, 2009. ISBN: 9781598299823. URL: https://books.google.nl/books?id=xYdiAQAAQBAJ.

[15] M. T. Moreira et al. "NCL Synthesis With Conventional EDA Tools: Technology Mapping and Optimization". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65.6 (2018), pp. 1981–1993. ISSN: 1549-8328. DOI: 10.1109/TCSI.2017.2772206.

[16] M. T. Moreira et al. "NCL+: Return-to-one Null Convention Logic". In: *2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS)*. 2013, pp. 836–839. DOI: 10.1109/MWSCAS.2013.6674779.

[17] R. A. Guazzelli, M. T. Moreira, and N. L. V. Calazans. "A comparison of asynchronous QDI templates using static logic". In: *2017 IEEE 8th Latin American Symposium on Circuits Systems (LASCAS)*. 2017, pp. 1–4. DOI: 10.1109/LASCAS.2017.7948103.

[18] M. Chang, P. Yang, and Z. Pan. "Register-Less NULL Convention Logic". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 64.3 (2017), pp. 314–318. ISSN: 1549-7747. DOI: 10.1109/TCSII.2016.2557812.

[19] *Fundamental NCL Gates*. URL: https://www.ndsu.edu/pubweb/~scotsmit/NCL_gates.pdf (visited on 03/12/2019).

[20] T. Lin et al. "An Ultra-Low Power Asynchronous-Logic In-Situ Self-Adaptive V_{DD} System for Wireless Sensor Networks". In: *IEEE Journal of Solid-State Circuits* 48.2 (2013), pp. 573–586. ISSN: 0018-9200. DOI: 10.1109/JSSC.2012.2223971.

[21] L. Nagy and V. Stopjaková. "Current sensing completion detection in dual-rail asynchronous systems". In: *2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*. 2012, pp. 38–41. DOI: 10.1109/DDECS.2012.6219021.

[22] M. Maymandi-Nejad and M. Sachdev. "A digitally programmable delay element: design and analysis". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 11.5 (2003), pp. 871–878. ISSN: 1063-8210. DOI: 10.1109/TVLSI.2003.810787.

[23] Aleksandar Milenkovic. *Complementary CMOS Logic Gates*. URL: http://www.ece.uah.edu/~milenka/cpe527-07F/lectures/CMOS_Static.pdf (visited on 04/02/2019).

[24] *Static CMOS*. URL: https://en.wikichip.org/wiki/cmos/static (visited on 04/02/2019).

[25] Prof. Bruce Jacob. *Static CMOS Logic*. URL: https://ece.umd.edu/courses/enee359a/enee359a-CMOS.pdf (visited on 04/02/2019).

[26] R. W. Knepper. *Dynamic Logic Circuits*. URL: http://people.bu.edu/rknepper/sc571/chapter5_ckts_C.ppt (visited on 04/02/2019).

[27] J Rabaey. *Dynamic CMOS*. URL: http://staff.iium.edu.my/zahirulalam/za/media/courses/ece4121/Dynamic%20CMOS.pdf (visited on 04/02/2019).

[28] Dewansh. *Advantages and Disadvantages of a Dynamic CMOS Circuit over a Static CMOS Circuit*. URL: https://www.vlsifacts.com/advantages-disadvantages-dynamic-cmos-circuit-static-cmos-circuit/ (visited on 04/02/2019).

[29] *Pass-Transistor-Logic*. URL: https://www.electronics-tutorial.net/Digital-CMOS-Design/Pass-Transistor-Logic/ (visited on 04/03/2019).

[30] *Pass-Transistor Logic*. URL: https://ece.uwaterloo.ca/~mhanis/ece637/lecture11.pdf (visited on 04/03/2019).

[31] Diwaker Pant. *Pass Transistor Logic*. URL: https://www.slideshare.net/diwakerpant/pass-transistor-logic (visited on 04/03/2019).

[32] J Rabaey. *Pass Transistor Logic*. URL: http://www.cs.tufts.edu/comp/103/notes/Lecture07(PassTransistorLogic).pdf (visited on 04/03/2019).

[33] *CAES-Doc*. URL: https://caesdoc.ewi.utwente.nl/ (visited on 08/12/2019).

[34] M. Shams, J. C. Ebergen, and M. I. Elmasry. "A comparison of CMOS implementations of an asynchronous circuits primitive: the C-element". In: *Proceedings of 1996 International Symposium on Low Power Electronics and Design*. 1996, pp. 93–96. DOI: 10.1109/LPE.1996.542737.

[35] Jose A. Torres. *Math real library*. URL: https://www.csee.umbc.edu/portal/help/VHDL/math_real.vhdl (visited on 06/26/2019).

Appendix A

Implementation of static CMOS cells

In chapter 3 and chapter 6, multiple cell design styles and techniques were studied and it was concluded that static CMOS style is the best choice for implementing basic asynchronous cells. The Kogge-Stone adder design was studied and basic cells required for the different adder variants were identified. Driving strength required by each and every cell was also estimated. Based on this study, only 7 basic asynchronous cells (driving strength not considered) are required to design the Kogge-Stone Adder for all the templates. Apart from these asynchronous cells, some cells from standard synchronous library were also used for the design. NOTE : While designing the schematic for different cells, the body bias terminal for every PMOS and NMOS transistor is taken out. This body bias terminal can be used to adjust the threshold voltage of the transistors. Upcoming sections will provide the schematics of all the basic cells used in the asynchronous designs.

A.1 Two-input Muller C element w/o Reset

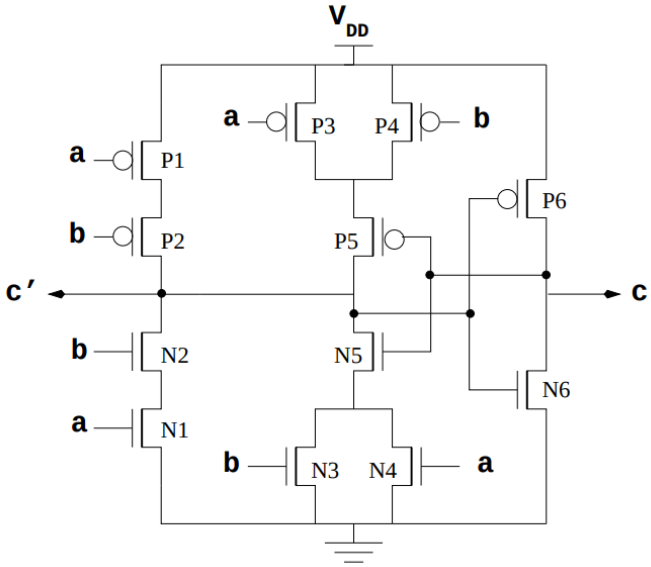


FIGURE A.1: Conventional Pull-up Pull-down implementation of C-element [34]

There are various ways to implement one of the most basic and important cell (Muller C element) present in asynchronous designs. Muller C element is a

hysteresis gate implementing a logical AND (a.b) operation. *In NCL, this cell is also known as TH22 gate* [19]. Basic truth table for Muller C element is already presented in Chapter 1. Authors of [34] present three different ways of implementing Muller C element.

First way, presented in Figure A.1, is a conventional pull-up pull-down realization presented by Sutherland [34]. Transistors P1, P2, P6, N1, N2 and N6 are used for generating the output while transistors P3, P4, P5, N3, N4 and N5 are used to provide the feedback and hold the state of output when inputs do not match. Figure A.2 presents the C-element implementation by Martin [34]. This circuit uses an arrangement of cross-coupled inverters to maintain the state of output when inputs do not match. This design is not efficient because it provides resistance for switching the output terminal. Also, while output terminal is switching, a lot of short-circuit current is observed through the transistors. For making this circuit work, efforts must be placed to resize the transistors in feedback path to make the output switching smooth and less power consuming.

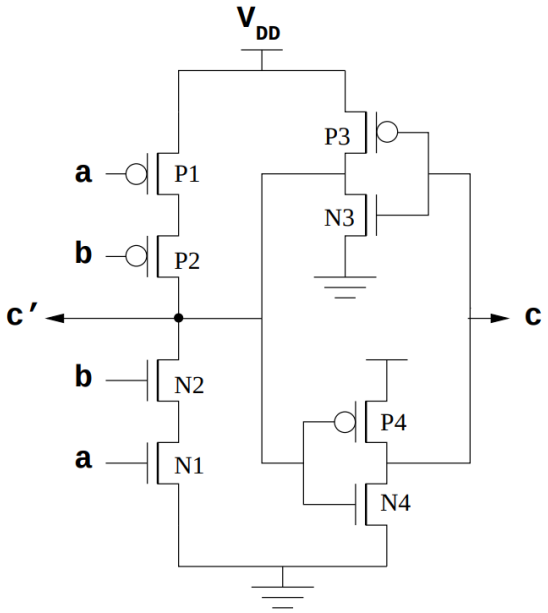


FIGURE A.2: CMOS implementation of C-element using cross-coupled inverters [34]

Figure A.3 presents the C-element implementation by Van Berkel [34]. This implementation also contains a feedback path in both pull-up and pull-down network. This circuit provides least amount of resistance for switching of output terminal and there is no short circuit current flowing through the transistors. In a comparison presented in [34], Van Berkel's implementation saves around 58% energy over Martins's implementation which is higher than the Sutherlands's saving of 42%. In terms of delay, Van Berkel's implementation obtained same delay for very less energy when compared to Martin's and Sutherland's implementations. All this information indicate that Van Berkel's implementation is the best way for the implementation of Muller C element. Hence, this way of implementation is followed and it is used to design the stage 3 of completion detector circuit in all the three asynchronous adder variants.

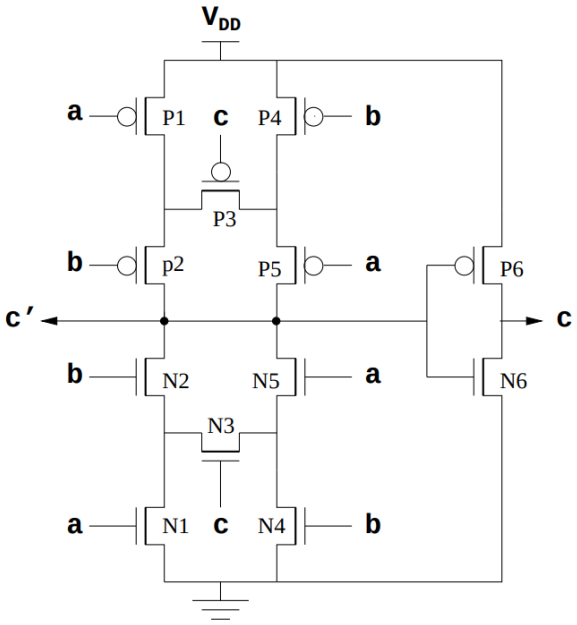


FIGURE A.3: CMOS implementation of C-element by Van Berkel [34]

A.2 Two-input Muller C element with Reset

Two-input Muller C element cell with reset functionality is based on Van Berkel's implementation, as presented in Figure A.3. This cell contains an additional PMOS transistor to provide the active-low reset functionality. This cell is used to design asynchronous registers ("REG") as presented earlier in Chapter 4 and Chapter 5. The reset functionality available in the cell helps in initializing the asynchronous pipeline from a pre-determined state. Figure A.4 presents the Muller C element with active-low reset signal.

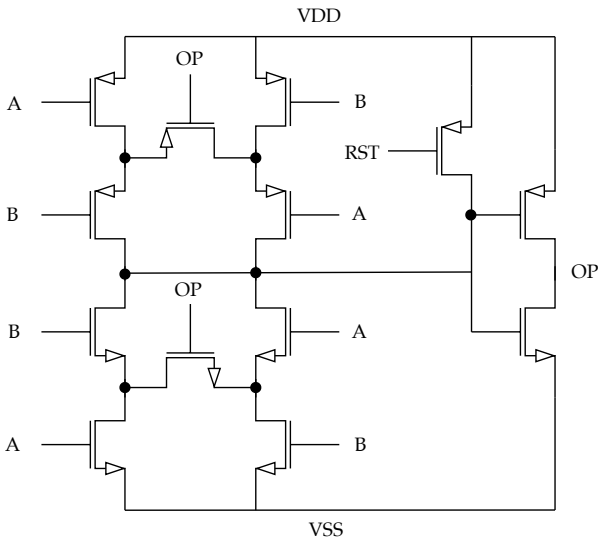


FIGURE A.4: CMOS implementation of C-element by Van Berkel with additional reset signal

By looking at Figure A.4, it can be argued that when "RST" signal is asserted low and signal "A" and "B" are asserted high, supply rails will get connected to each

other with a low resistance path. This low resistance path will lead to short-circuit current. Figure A.5 provides a solution to avoid this short-circuit current by adding an NMOS transistor to the NMOS network. This additional NMOS transistor will make sure that power rails never get connected to each other. Modified schematic presented in Figure A.5 is used for the implementation of asynchronous registers.

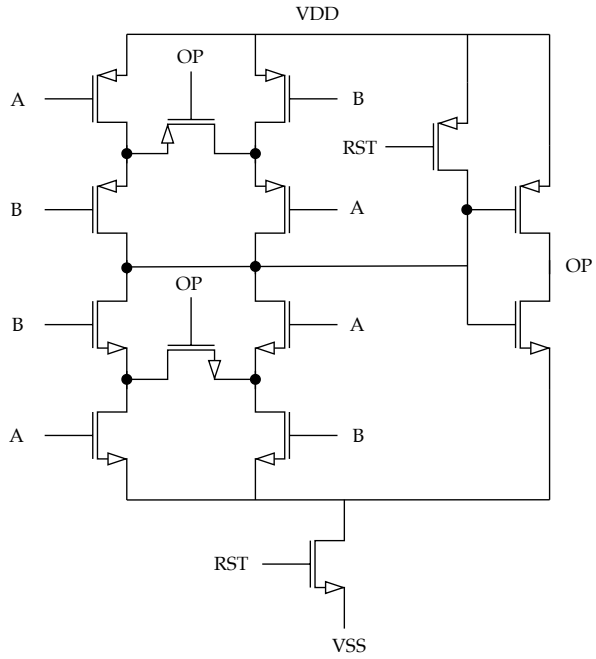


FIGURE A.5: Modified CMOS implementation of C-element by Van Berkel with additional reset signal

A.3 Dual rail XOR gate (No Hysteresis)

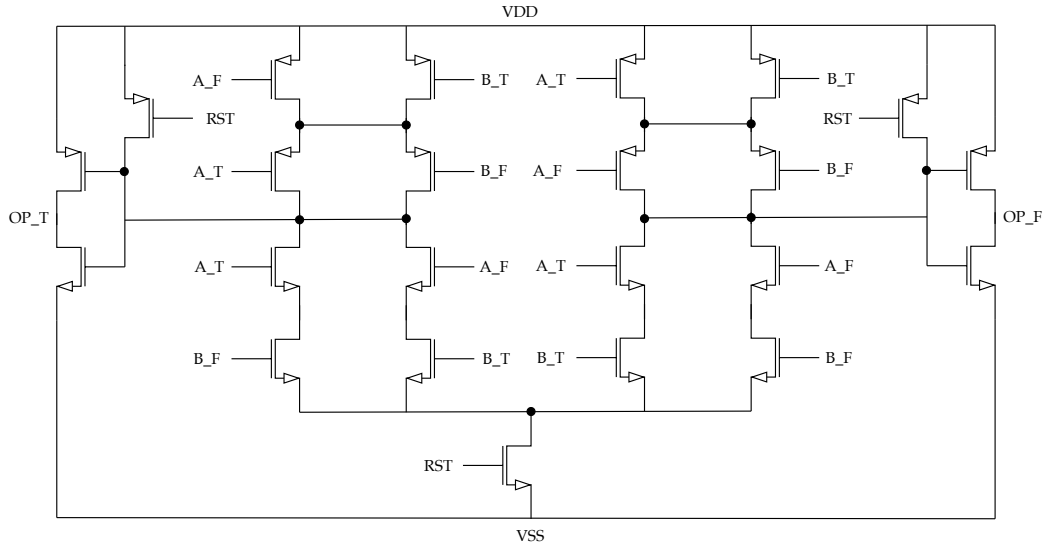


FIGURE A.6: Dual rail XOR gate with gate-level reset functionality

While designing basic gates for asynchronous circuits, it is observed that the Boolean expressions for true rail and false rails for some gates are same in

strongly-indicating as well as weakly-indicating type of logic. One such example is dual rail XOR gate. The Boolean expressions obtained for true and false rails are:

True rail (OP_T) = (A_T **and** B_F) **or** (A_F **and** B_T)

False rail (OP_F) = (A_T **and** B_T) **or** (A_F **and** B_F)

Figure A.6 presents the dual rail XOR gate with gate-level reset functionality. It should be noted that this gate does not have a hysteresis mechanism. This cell is used for implementing the GP cell in the adder variant based on Template 2.

A.4 Dual rail TH23W2 gate (No Hysteresis)

In synchronous logic, there is a gate viz. "AO21" which implements the following Boolean expression: A **or** (B **and** C). In NCL, this gate is known as TH23W2 gate which implements the same Boolean expression with a hysteresis mechanism. If this Boolean expression has to be implemented for weakly-indicating dual-rail scenarios, then the Boolean expressions obtained for true and false rails will be:

True rail (OP_T) = A_T **or** (B_T **and** C_T)

False rail (OP_F) = (A_F **and** B_F) **or** (A_F **and** C_F)

Based on the Boolean expressions mentioned above, Figure A.7 presents the weakly-indicating dual-rail TH23W2 gate without hysteresis mechanism. This cell contains reset functionality which makes it suitable for the use in adder variant based on Template 2.

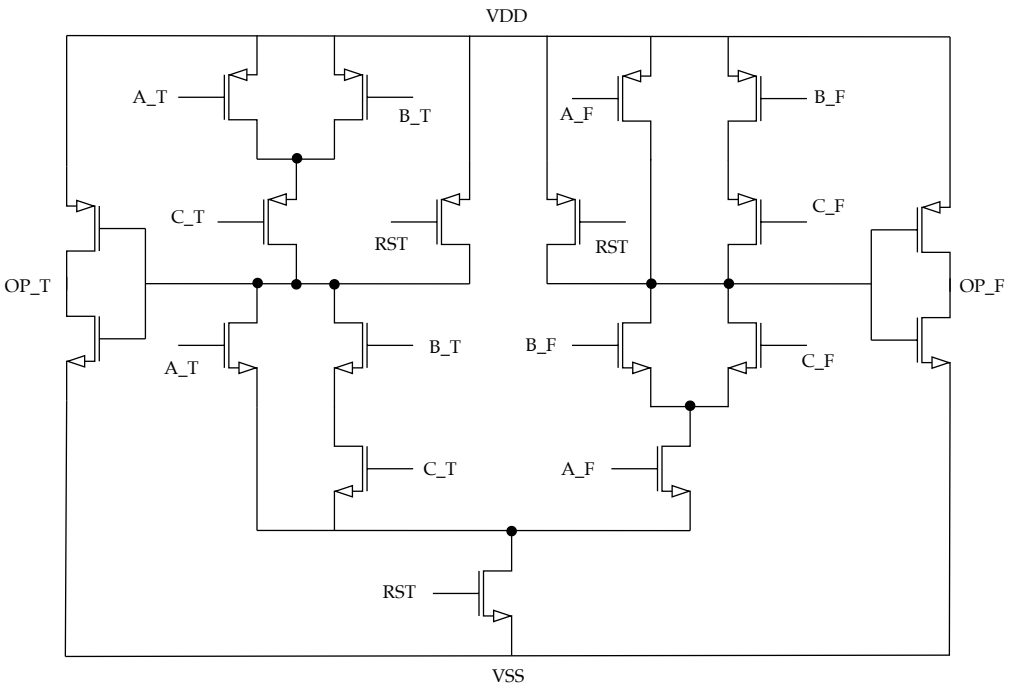


FIGURE A.7: Weakly-indicating dual-rail TH23W2 gate (implementing A + BC)

Instead of weakly-indicating logic, if strongly-indicating type of logic is required, then the Boolean expressions become very complicated and involves complement of

some input signals. To give an idea about the complexity, the Boolean expressions for true and false rails for strongly-indicating logic are presented below :

True rail = (A_T **and** (B_T **or** B_F) **and** (C_T **or** C_F)) **or** (A_F **and** B_T **and** C_T)

False rail = A_T′ **and** A_F **and** (B_T **or** B_F) **and** (C_T **or** C_F)

These complicated Boolean expressions lead to stacking of PMOS and NMOS transistors which slows down the speed of operation. Designing layout for such complicated cell also becomes very difficult. If strongly-indicating dual rail TH23W2 gate is required then it should be implemented as a combinational logic of AND and OR gate (A + BC).

A.5 Weakly-Indicating Dual-Rail AND gate (No Hysteresis)

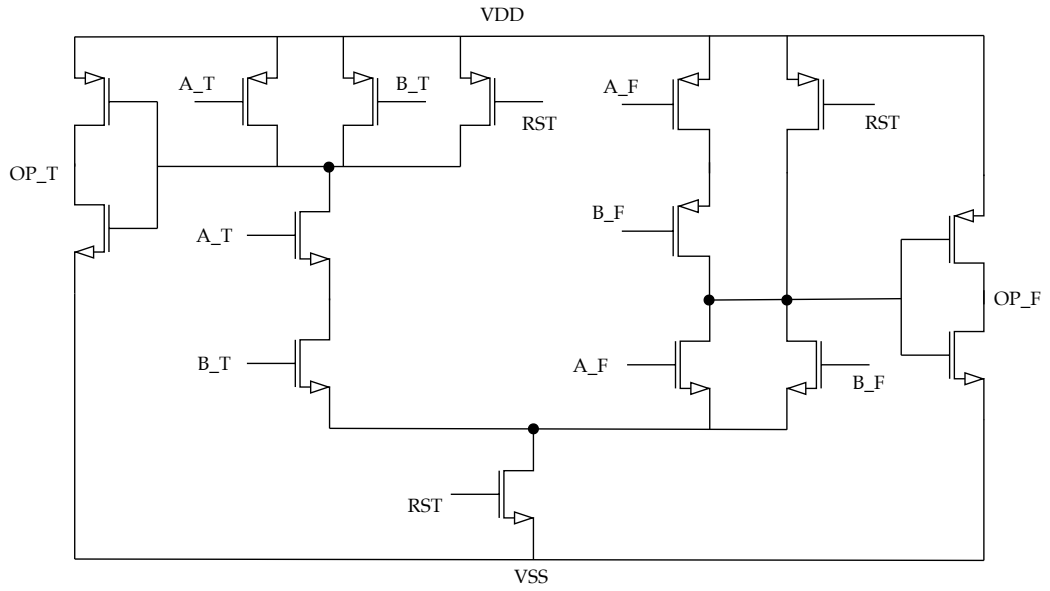


FIGURE A.8: Weakly-indicating dual-rail AND gate

The weakly-indicating dual rail AND gate without hysteresis mechanism is presented in Figure A.8. This dual rail gate contains gate-level reset functionality which makes it suitable for the adder design based on Template 2. This gate is similar to the PCSL AND gate, as presented in [20]. The true and false rail of this gate implements following Boolean expressions:

True rail (OP_T) = (A_T **and** B_T)

False rail (OP_F) = (A_F **or** B_F)

If the input and output terminals in the Boolean expressions are interchanged, then, this same cell can be used to implement weakly-indicating dual rail OR gate. To give an idea to the reader, the Boolean expressions involved for a weakly-indicating dual rail OR gate are presented below:

True rail (OP_T) = (A_F **or** B_F)

False rail (OP_F) = (A_T **and** B_T)

A.6 NCL THand0 gate (with hysteresis)

NCL THand0 gate is one of the 27 fundamental NCL gates presented in [19]. This gate contains hysteresis mechanism which helps in establishing the indicatability property for a strongly-indicating as well as for a weakly-indicating design. This gate implements following Boolean expression :

$$OP = (A \textbf{ and } B) \textbf{ or } (B \textbf{ and } C) \textbf{ or } (A \textbf{ and } D)$$

This gate along with TH22 gate (2-input Muller C element) can be used to form a dual rail AND gate which supports hysteresis mechanism. TH22 gate will be used to implement true rail whereas this gate will be used to implement false rail of a dual rail AND gate. The expression for true rail and false rail output are:

$$\text{True rail} = A_T \textbf{ and } B_T \text{ (TH22 gate)}$$

$$\text{False rail} = (A_F \textbf{ and } B_F) \textbf{ or } (A_F \textbf{ and } B_T) \textbf{ or } (A_T \textbf{ and } B_F) \text{ (THand0 gate)}$$

The schematic of this THand0 gate can be found in Figure A.9.

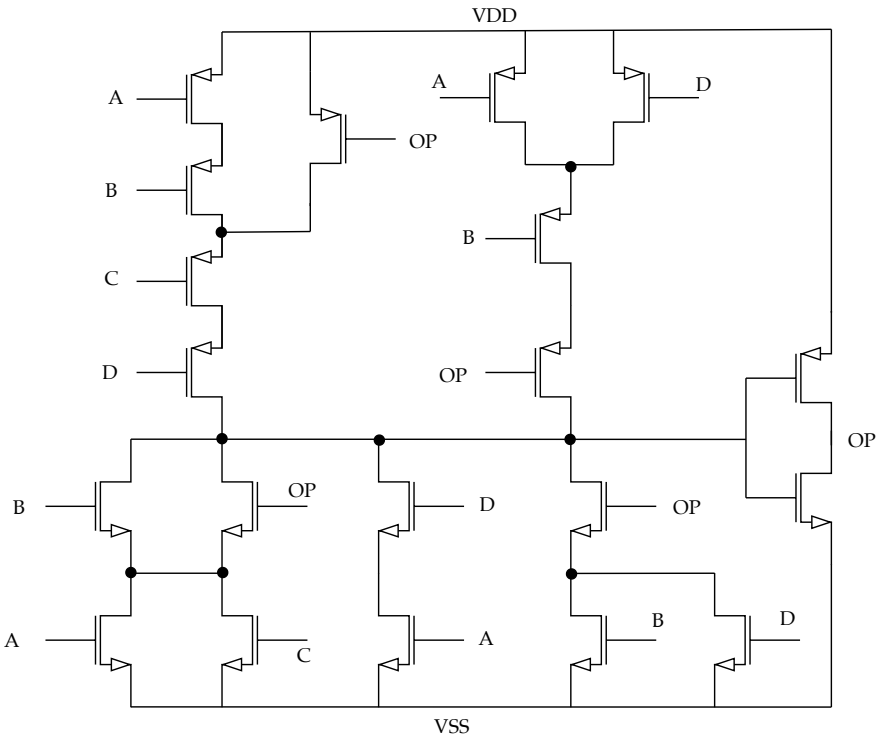


FIGURE A.9: NCL THand0 gate (with hysteresis)

A.7 NCL THxor0 gate (with hysteresis)

NCL THxor0 gate is also one of the 27 fundamental NCL gates presented in [19]. This gate also contains hysteresis mechanism which helps in establishing the indicatability property. This gate implements following Boolean expression :

$$OP = (A \textbf{ and } B) \textbf{ or } (C \textbf{ and } D)$$

This gate can be used to form a dual rail XOR gate which supports hysteresis mechanism. The Boolean expressions implemented for true and false rail of XOR gate are presented below.

$$\text{True rail} = (A_T \textbf{ and } B_F) \textbf{ or } (B_T \textbf{ and } A_F)$$

$$\text{False rail} = (A_T \textbf{ and } B_T) \textbf{ or } (A_F \textbf{ and } B_F)$$

The schematic of this THxor0 gate can be found in Figure A.10.

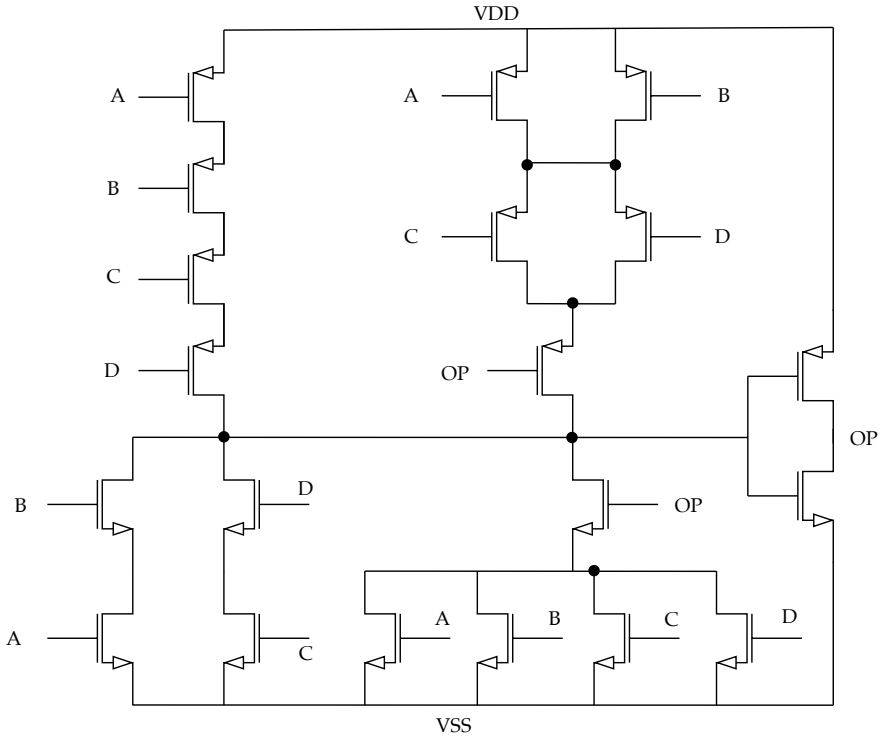


FIGURE A.10: NCL THxor0 gate (with hysteresis)

A.8 Synchronous Cells

Apart from the asynchronous cells discussed earlier, there are a few conventional synchronous cells which are taken up from the standard synchronous library. These cells are used for designing completion detector circuits, and for the combinational part of the adder design based on template 3. The cells used for the adder designs are as follows:

- 2-input, 3-input and 4-input AND gate.
- Buffer cells.
- Inverter.
- 2-input NOR gate.
- 2-input, 3-input and 4-input OR gate.
- "AO22M1RA" cell which implements following Boolean expression : AB + CD.

These cells do not support gate-level reset functionality and hysteresis mechanism. Also, the schematics for these cells is not presented here and can be referred from the manual of standard cell library (if available).

Appendix B

Simulating asynchronous designs with VHDL

Although no mathematical proofs are provided, the asynchronous pipelines presented in this thesis are supposed to be correct on the basis of reasoning about the arrival of signals, detection of valid/null signals, etc. Behavioral simulations based on structural compositions of basic gates were performed to increase confidence in their correctness. These simulations also assist in detecting the possibility of a pipeline ending up in a deadlock state. Example of the basic gates involved for designing are presented in Appendix A. This modeling of designs can be done by using hardware description languages like VHDL or Verilog. In this thesis work, VHDL is chosen as the language for modeling.

Asynchronous QDI designs assume no delay information on the gates and wires. The delay parameter can have any unbounded positive value with time-varying characteristics. For performing the simulations using VHDL as a modeling language, it is necessary to first create basic gates which are capable of following this timing characteristics. All the gates should be able to adjust their propogation delay randomly during simulations. Since the design is asynchronous, the input arrival can also be asynchronous and random in time. This means that testbench should also be able to provide data to the design at random time instances. It should be noted that feeding valid or null data to the design at random instances is possible only after the design is ready to accept new valid or null data.

To generate random delays at the gate level, a non-IEEE library named "math_real" is used. This library belongs to the IEEE VHDL Math Package Study Group from University of Maryland, Baltimore County, U.S.A. [35]. In this library, a VHDL procedure namely "uniform" is defined which returns a pseudo-random number with uniform distribution in the interval 0.0 to 1.0. The signature of this procedure is presented below :

```
procedure uniform (variable Seed1 , Seed2 : inout integer; variable X out real);
```

As per the library, seed values (Seed1 and Seed2) must be initialized to values in the range [1, 2147483562] before making first call to the procedure. With every subsequent call seed values are modified automatically. Also, this random number generator is portable which means it can work for both 32-bit and 64-bit machines and it has a period of approximately 2.30584 x (10^{18}) for every set of seed values [35].

An example to understand the usage and give readers an idea to scale this example for entire asynchronous design, is presented on the next page. The example

presents a conventional AND gate with random gate delay.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.math_real.all;

entity AND_GATE is
    generic (max_delay_ns : real := 10.0);
    port(A : in std_logic;
         B : in std_logic;
         Z : out std_logic;
end AND_GATE;

architecture behv of AND_GATE is
begin

    P1 : process(A , B)
        variable seed1 : positive := 1;
        variable seed2 : positive := 16565;
        variable rand : real := 0.0;
    begin

        uniform(seed1 , seed2 , rand);
        Z <= A and B after rand * max_delay_ns * 1 ns;

    end process P1;
end behv;
```

This VHDL code was simulated in Modelsim by providing input patterns at random time. The simulation starts with providing logic '0' to both the terminals of AND gate at 0 ns, as seen in Figure B.1. The simulation resolution taken here is in nanoseconds. Input A becomes logic '1' at 10 ns whereas input B becomes logic '1' at 15 ns. When both inputs are at logic '1', AND gate produces logic '1' at 21 ns, which means that the delay of AND gate is 6 ns.

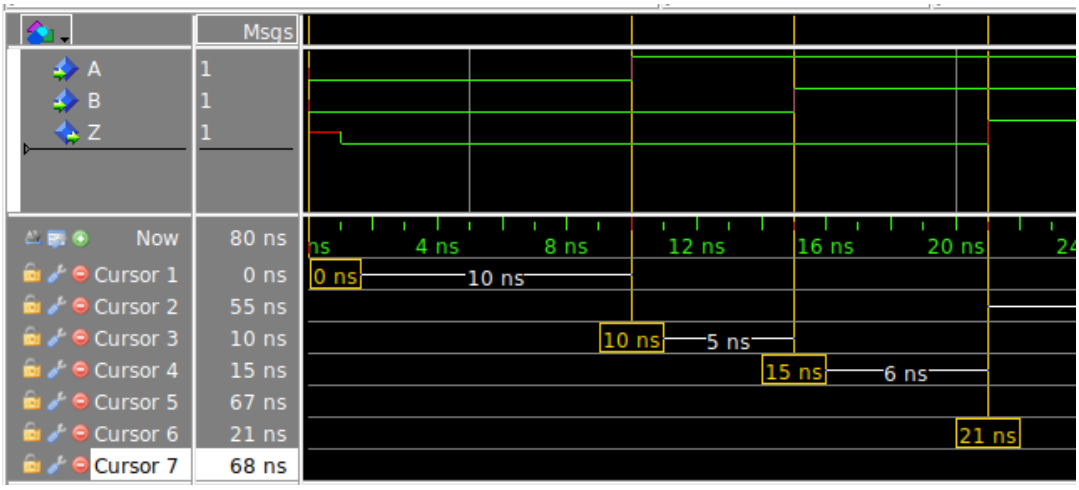


FIGURE B.1: AND gate operation with a delay of 6ns

Input terminals which were at logic '1' earlier turns to logic '0' at 30ns and 32ns respectively, as shown in Figure B.2. This change in the logic state of input terminals is reflected on the output terminal at 36ns, which means that the delay of AND gate is now 4ns.

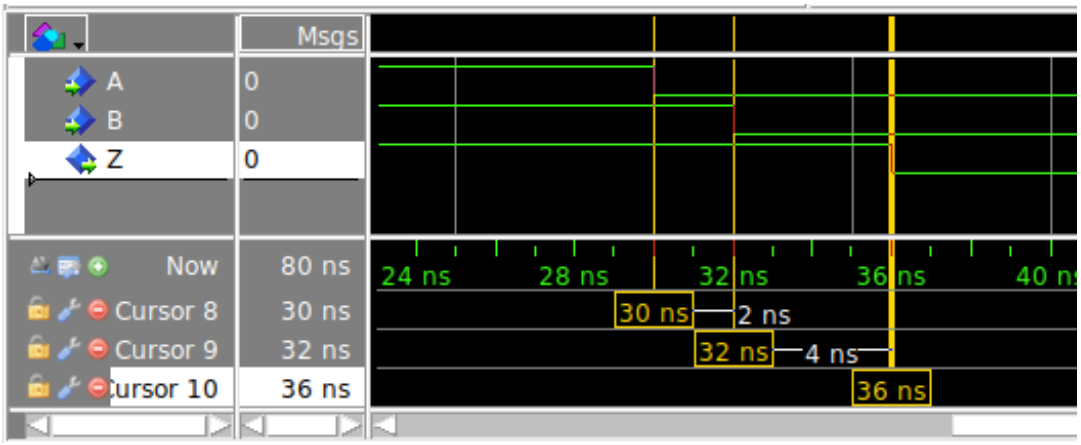


FIGURE B.2: AND gate operation with a delay of 4ns

In another case, these input terminals again go back to logic '1' at 55ns and 67ns respectively. This change in input is reflected at the output of AND gate at 68ns, which means that the delay of AND gate is now 1ns. This scenario is presented in Figure B.3.

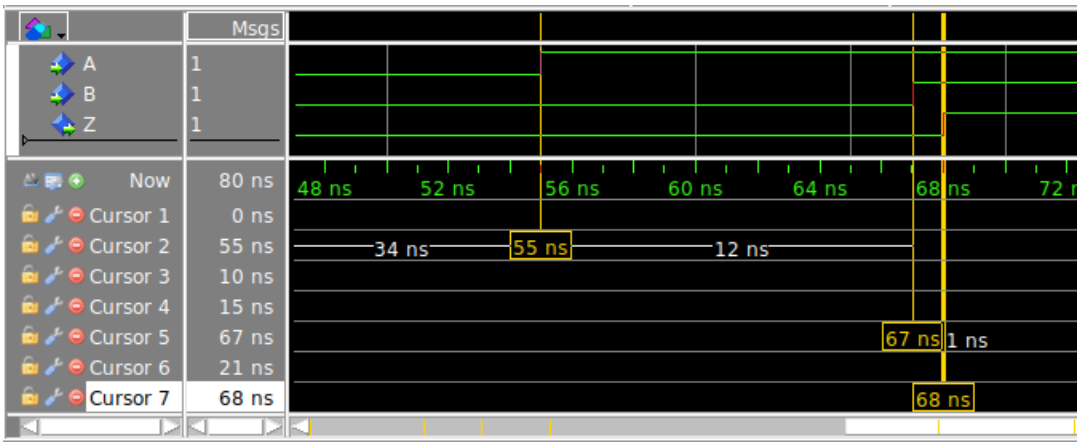


FIGURE B.3: AND gate operation with a delay of 1ns

All the three cases presented earlier shows that the AND gate displayed variable delays during simulation. This simple case of AND gate can be easily extended to other logic gates, as well as, to the testbenches. This random delay during simulation will help in testing delay-insensitivity property for the templates. The properties tested with this random delay generation technique are presented in Chapter 7.