ENERGY-PERFORMANCE TUNABLE DIGITAL CIRCUITS

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Bita Nezamfar

December 2008

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____

(Mark A. Horowitz) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____

(Boris Murmann)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____

(Bruce A. Wooley)

Approved for the University Committee on Graduate Studies

Abstract

The continued scaling of CMOS technology has enabled incredible computing devices to be created, but also pushed these devices to their energy dissipation limits. Process, temperature and workload variation change the power and performance of a chip, making it impossible to create a single energy optimal design. As a result, adaptive energy and performance adjustment methods have emerged as attractive methods to improve the effective power efficiency of a circuit. Dynamic voltage scaling and adaptive transistor body biasing have been employed to control the dynamic and leakage power of the circuits, respectively. Unfortunately, in modern technologies body bias is not very effective in controlling leakage current.

In this thesis, we propose an alternative approach to the in situ adjustment of the energy-performance point of the design. We first show how the effective threshold of the transistors can be adjusted over a wide range using skewed supplies. Leveraging this method for pure static logic is difficult, so we create a new circuit architecture that is intrinsically faster than static circuits, and more importantly, can use skewed supplies to tune both dynamic and leakage power of the system over a wide range. As an example, we describe how the proposed architecture can be used to implement FPGA chips that can be programmed in the field to be low power or high performance depending on the application. Measured results from a 90-nm CMOS test-chip indicate that much wider tuning range is possible using the new programmable interconnects compared to static interconnects. The key to this new logic family is the use of static external signaling, but internal pulse logic. Using pulses, even if they are internal to the gates, does complicate system performance analysis. We also include a discussion of the issues involved with the use of the proposed logic for general circuits and how to analyze the resulting system and show that the required changes can be made small depending on application.

For the purpose of improving the performance of FPGA example, we also propose and evaluate a new fully pulse-mode architecture for FPGA. Regularity of FPGA structure allows one to find architectures that have less system level problems than general pulse mode circuits. This serves as an upper bound on the performance of FPGA among the conventional and proposed architectures.

Acknowledgements

My years at Stanford would not have been the same without the help of many great people.

First, I thank my advisor, Prof. Mark Horowitz, for being a great advisor and a role model of a researcher for me. He taught me how to correctly approach the problems, how to present, and how to get my ideas across. He always cared whether I enjoyed my work and was always supportive during the ups and downs of research. Working with him is definitely one of the things I miss most after leaving Stanford.

I am grateful to Prof. Murmann for being my co-advisor and for his comments on this thesis. I would also like to thank Prof. Wooley for his help and support in many occasions, for his excellent Analog to Digital course, and also for reading this thesis and for accepting to be on my orals committee.

Being a member of VLSI group, I have had the opportunity to work with many different people from Mark's group. Valentin Abramzon, Dinesh Patil, and Elad Alon particularly helped me out with different aspects of my research at Stanford. My special thanks goes to Dinesh for his help with using his convex optimization tool. A lot of Mark's former students also helped with reading my papers or being available for discussions. In particular, I would like to acknowledge the help of Don Stark, Ron Ho, Dean Liu, Sam Palermo and Azita Emami. I also thank Manoj Chirania and Philip Costello of Xilinx for their help and feedback about the interconnect architecture.

Teresa Lynn is a great admin for our group. All I had to do whenever I needed her help, was to send an email to her and she would take care of everything. Ann Guerra also made everything smooth in CIS. I thank them for their great job.

I was fortunate to have really good friends around me. Shadi Oveisgharan, Shirin Jalali, Vahideh Hosseinikhah, Neda Nategh, and Nafise Rahimzadeh have always been there for me and I thank them for their friendship. Parastoo Nikaeen and Mona Jarrahi were both my friends and colleagues at CIS, and I enjoyed doing many course projects with them. The Persian community at Stanford had a great role in making Stanford a great place like home for me. As I leave Stanford, I am carrying with me many great memories of the activities and events we had together.

I am grateful to my mother and father in law for their help and support and for treating me like their own child during these years. I also thank my brother in law, Arash, for his kindness and support.

Words cannot express my gratitude towards Amir. I cannot thank him for all the good he has brought to my life. I thank God, many times, for having him as my best friend, colleague, companion and husband all this time! I only hope that I can be as good for him as he has been for me all these years. He has always been really encouraging and has had so much belief in me, sometimes even more than myself!

Finally, I thank my father and mother, my sister, Shiva, and her family, my only brother, Hooman, and my younger sister, Gita, for all their love and support. Sacrifice wouldn't be a good enough word for all they have done for me, nor my simple thank you would be enough to acknowledge them. I don't think they can imagine how much happiness they bring into my life by just being there for me, despite the very far distance. This thesis, for all it is worth, is dedicated to them.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In the past few decades, digital circuit performance improved rapidly; mostly enabled by aggressive improvement of transistor performance. The exponential decrease in transistor feature sizes accompanied by exponential increase in transistor switching speed has led to incredible computing power in today's integrated circuits. However, this rapid scaling has also brought about significant challenges for the design of power-efficient devices that can deliver the expected power and performance levels.

One immediate side-effect of small feature sizes is increased variability [1]. In other words, even though it is possible to make transistors that are much faster than the previous generation, it is difficult to guarantee that every device created in the production line has the same performance. Hence, guaranteeing a high production yield has become a very important concern in modern technologies[2]. Consequently, the required margin to accommodate the change in performance resulting from variations can take away most of the improvements obtained from improving transistor technology.

Another side effect of atomic transistor dimensions is the increased leakage (or off-current) of the transistors [1]. As a result, even the parts of a circuit that are not active dissipate power. This idle power can become a large portion of the total chip power in today's multi-function integrated circuits where only a small fraction of

transistors are needed for any particular functionality. For optimal design, leakage power should be about 20%-40% of the active power. The problem arises when the active power is variable, since then one wants to modulate the leakage power, which is often difficult. If this cannot be done, then the amount of energy spent is not directly proportional to the work performed.

Therefore, even though smaller transistor sizes free up die area for putting more transistors on the chip, the power budget makes it harder to add more functionality. As a result, in today's designs, energy has become even a more important concern than before: people not only care about the performance that they get, but they very much care about the amount of energy they have to spend to achieve the performance.

In general, there are many different implementations of a system that may achieve a certain level of performance. But, clearly, not all different implementations are optimal. In fact, if you plot all possible designs, the leading edge of the feasible set of designs is the Pareto-optimal curve of all the energy efficient designs as shown in Figure 1.1.

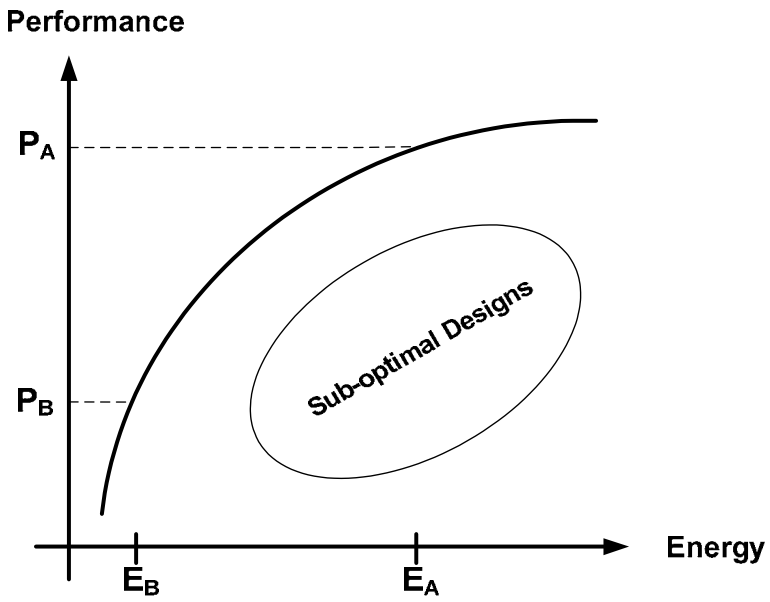

Figure 1.1. Energy-Performance Tradeoff curve

For example, if we are targeting low performance (P_B), considering all designs that meet this performance, we choose a simple design to achieve the least possible energy consumption of E_B. However, if we are targeting a high performance level (P_A), the previous design will not meet the performance spec, and a new design will be optimal with energy consumption of E_A.

In many systems, the expected performance from the system can change from time to time. For example, let's consider the case of a laptop; it may be in standby mode where the required performance is low; it may be running a few tasks and require a modest performance; or it maybe used for applications where very high performance is required. In addition to changes in workload, the characteristics of the transistors can change due to temperature variations, which affect both energy consumption and performance of the system. In such a system, the optimal energy-performance point of the design is a complicated function of the system workload pattern and the dependencies between leakage power, chip temperature, and process characteristics. Therefore, to stay on the pareto-optimal curve, we ideally would like to have the means to "tune" the hardware to adjust its energy consumption based on the target performance. This dissertation looks at one way to address this problem.

1.2 Organization

Many different approaches have been studied to adaptively change the energy and performance of a design. Energy-performance tunable circuits are attractive for achieving this goal since they enable the user to change the operating point of the transistors in the design after fabrication to compensate for process, temperature and workload variations, thus improve the effective power efficiency of the circuit. Most of the optimizations in this area fall in the category of adaptive voltage scaling and adaptive body bias. In Chapter 2, we provide a brief summary of the previous research in this area. While adaptive voltage scaling is quite effective, in scaled technologies body bias is not as effective in controlling leakage currents. Therefore, we explored ways to control leakage by more direct control of the threshold voltage. In Chapter 2,

we explain how the effective threshold of the transistors can be changed by skewing the supplies. We also point out the problems of applying this technique to regular static circuits and review previous related work in this area.

Even though skewing the supplies cannot be directly applied to conventional static digital circuits, it can be utilized to adjust transistor threshold if the correct logic family is used. In Chapter 3, we investigate different ways of creating a logic family that enable direct threshold control. The obvious place to start is dynamic and pulse-mode logic. Pulse-mode logic satisfy the circuit level requirements, but lead to many system design issues which are also described in Chapter 3. We then propose a new logic family that we call pseudo-static that avoids the fundamental system level problems of using pulse mode gates but still retains the potential for creating an energy-performance tunable circuit. The architecture of this circuit is explained in the same chapter and its characteristics are discussed. We show that using these gates in the system requires only minor modifications to the timing analysis, but does not impose any fundamental problems. As a result, this logic can replace the ordinary static gates with minimum required system change. The design approach for designing pseudo-static gates is also discussed in this chapter. One major cost of using pseudo-static circuit is the requirement for generation and distribution of extra supplies. The characteristics of these supplies are discussed in this chapter along with cost of different way of generating them.

We originally developed pseudo-static logic while we were working on improving the performance of FPGA interconnect circuits. In Chapter 4, we first explain why the FPGA application is important and why having pseudo-static interconnects is interesting. The rest of the chapter shows the implementation of FPGA interconnect using pseudo-static circuits and its performance. Chapter 4 will include description of our test chip and the measurement results. Looking at the FPGA architecture, we can take advantage of its regularity to implement a fully pulse-mode FPGA and handle the system complications with moderate cost. This is discussed in Chapter 5 and serves as an upper bound on the performance and Chapter 6 concludes this work.

Chapter 2

Adaptive Optimization Techniques

In this chapter, we overview some of the techniques used to change the energy and performance of a chip after fabrication. The simplest method was to adjust the supply voltage specifically for each chip to meet the required performance when the process variation is large. Adaptive techniques have also been used to become closer to the optimum energy consumption point when working condition and temperature of the chip changes. After discussing the strengths and the limitations of prior techniques, we propose a more direct way of post-fabrication control of transistor thresholds. Unfortunately, as described in the chapter, this technique cannot be applied to static CMOS circuits.

2.1 Increase yield by adjusting the supply

The standard approach to accomplish post-fabrication tuning is by adjusting the supply voltage. All high-end microprocessors use this technique to compensate for process variations by lowering the supply voltage for "fast" leaky chips to increase the number of chips that meet both the performance and power specifications. In Intel processors, for example, there is a voltage identification pin that determines the voltage for a specified frequency for that chip [3]. This voltage is determined after calibration of that specific chip.

The study in [4] shows that yield can significantly increase by adjusting the supply voltage. Figure 2.1(a) shows the scatter plot for the normalized frequency and power consumption of 5000 samples of a test circuit running from nominal supply of 1.1V in 0.1um CMOS technology and Figure 2.1(b) shows this scatter plot after adjustment of supplies. Figure 2.2 shows the supply voltage distribution to achieve this.

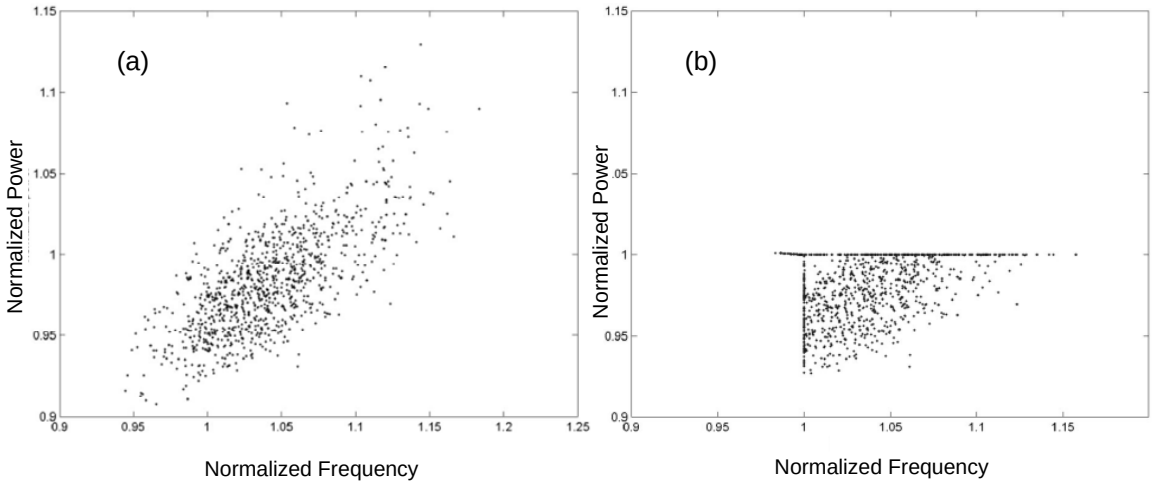

Figure 2.1. Scatter plot of normalized frequency and power (Figure from [4]).

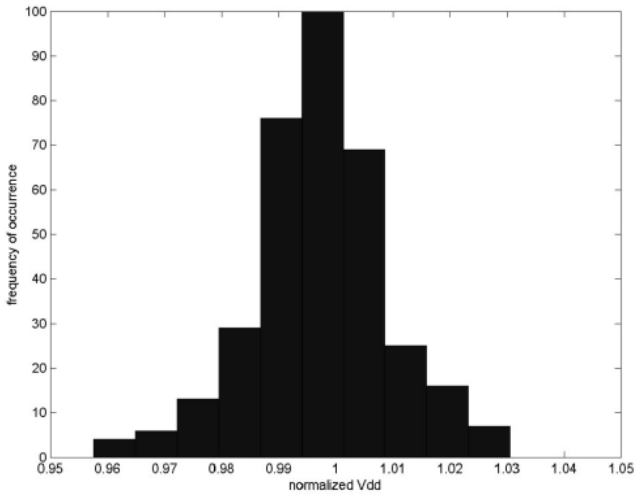

Figure 2.2. V_{dd} distribution after adjusting supply voltage (Figure from [4]).

From these figures, we can see that significant improvement in yield is possible with the adjustment of supply voltage. As shown in Figure 2.2, for many chips, the adjusted supply is only less than $\pm 100mV$ different than the nominal supply but for some of them, it has to change by $\pm 300mV$.

2.2 Adaptive voltage scaling

As explained in Chapter 1, in addition to process variation, the specific working condition of the chip, for example temperature and workload, also affects the optimum energy point. Most laptop processors go further than setting V_{dd} at test and dynamically switch to a lower operating voltage depending on workload [5],[6],[7]. This adjusting of the supply voltage based on the conditions during the operation of a chip (instead of having a fixed working voltage) is called Dynamic Voltage Scaling (DVS) and has been extensively used [8],[9]. DVS helps since changing the supply changes dynamic power quadratically but the inverse effect on delay is less significant. Therefore, when the required performance decreases, decreasing the supply can help saving power.

One of the most aggressive implementations of DVS was the Razor project by Blaauw and Austin [10],[11]. They scaled power supply voltage until the part occasionally made errors which the system could correct. It used the presence of these errors to indicate the correct supply voltage had been reached. Figure 2.3 shows the schematic for their flip flop which was the block that detected the errors.

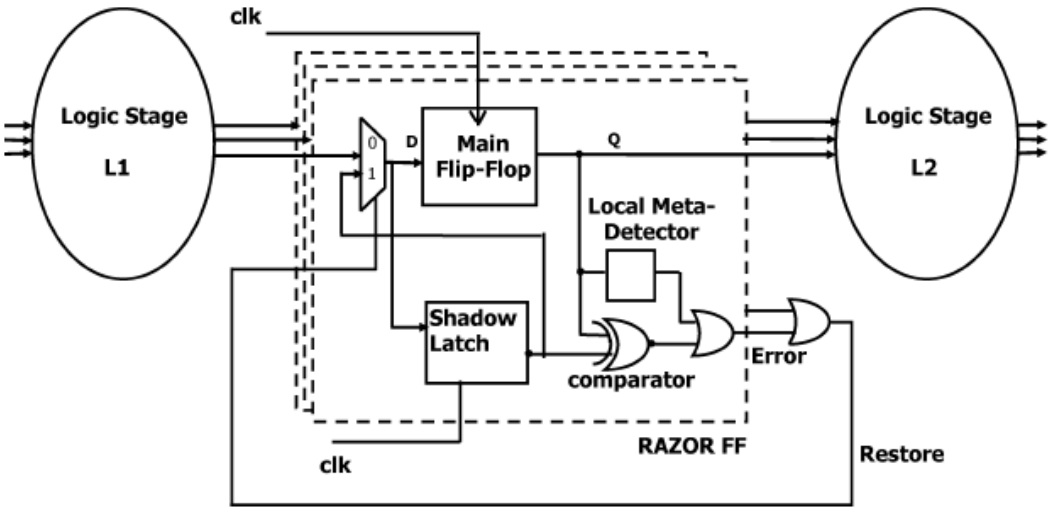

Figure 2.3. Abstract overview of Razor Flip Flop (Figure from [11])

In order to detect the lowest possible supply voltage, they used a fast path and a slow path for the signal and compared the outcome of these two. The fast path is a flop which is clocked by the main clock of the system while the slow path is a latch that is clocked with a delayed clock. Therefore, the output of the slow path is always correct after it is evaluated (delayed clock should be slow enough that the output is correct for the minimum supply voltage). If the output of the two paths were different, it meant that the supply voltage has become too low and should be increased. Using this technique, they eliminated the significant voltage margins required to ensure correct operation under the worst case condition.

While DVS is an effective technique for optimizing dynamic power; as the next section will show, in energy optimized systems, dynamic power and leakage power should be in balance. Therefore, in order to optimize circuit energy, leakage current should also be adaptively controlled. Section 2.4 describes different ways to reduce leakage power.

2.3 Balance in static and dynamic energy

Sakurai [12] has shown that for an energy optimized system, dynamic and leakage energy should be in balance, where static circuit consumes about 30% of the total energy. Looking at the equation for dynamic and leakage power[1]:

$$P_D = afC_L V_{DD}^2 \tag{2.1}$$

$$P_{Leak,\max} = I_0 e^{-V_{Th,\min}/N_S} V_{DD} \tag{2.2}$$

$$t_D = \frac{C_L V_{DD}}{\beta\left(V_{DD} - V_{th,\max}\right)^{\alpha}} \tag{2.3}$$

$$f = \frac{1}{L_D t_D} \tag{2.4}$$

Table 2-1. Parameters used in Equations 2.1 to 2.4

A	Activity factor
L_D	Logic depth of the critical path
F	Clock frequency
C_L	Load capacitance
$V_{th,min}$	Lowest V_{th} in temperature and process variation range
$V_{th,max}$	Highest V_{th} in temperature and process variation range
N_s	nkT_{max}/q (n:subthreshold slope factor)

[1] Here it is assumed that leakage current is dominated by the sub-threshold current. Although scaling has caused gate leakage to increase, sub-threshold current is still the major component of leakage current.

In these equations, $V_{th,min}$ is used to estimate worst case leakage power and $V_{th,max}$ is used to estimate worst case delay in order to consider take the effect of variations into account.

In the above equations, if we take the derivative of total power (sum of dynamic and leakage power) with respect to V_{dd}, obtain the optimum point and calculate the ratio of static power to the total power consumption, the optimum ratio will be around 30% for typical process values. It is shown in [12] that this ratio is not changed over a wide range of activity factor, logic depth and frequency. It makes sense that there should be a balance between dynamic and static energy in a chip. If the static power was very low, one could decrease the threshold voltage (increasing the small leakage power). This would allow one to decrease V_{dd}, and maintain performance which decreases the total power.

For example, assuming that the chip is working at its optimum energy point originally, if the required frequency increases, the threshold voltage does not need to change significantly since it will increase the leakage current exponentially. Thus the optimum threshold voltage only changes by 0.1V when the required frequency changes from 100 MHz to 300 MHz.

On the other hand, changing the activity factor does not change the leakage current while it does change the dynamic power. Thus, the ratio of dynamic power and leakage power changes, and in order to be close to optimum, threshold voltage should be adjusted properly. The optimum threshold voltage changes by 0.3V when activity factor changes from 1 to 0.01. Therefore, in order to get close to the optimum both supply voltage and threshold voltage should be adjusted adaptively.

2.4 Leakage reduction techniques

Different leakage components of a transistor are shown in Figure 2.4 and explained in Table 2-2:

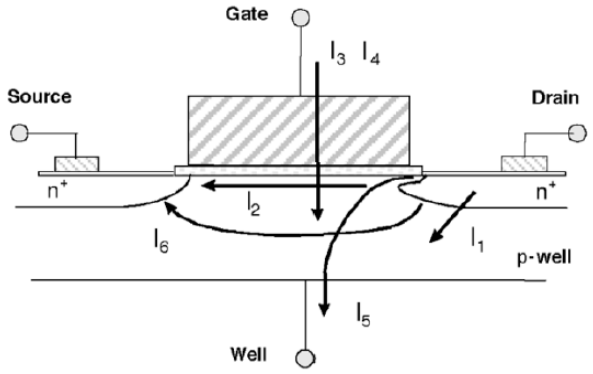

Figure 2.4. Leakage components of a transistor

Table 2-2. Different leakage components of a transistor

I_1	Reverse-bias pn junction current
I_2	Sub-threshold leakage
I_3	Oxide tunneling current
I_4	Gate current due to hot-carrier injection
I_5	Gate induced drain leakage (due to high field effect in the drain junction)
I_6	Channel punch-through (due to proximity of the source and drain depletion regions)

From these components, I_2, I_5 and I_6 are off-state leakage mechanisms. I_1, I_3 and I_4 occur in both on and off states. From the off-state leakage currents, the most important one has been the sub-threshold leakage current (I_2) so far and most of the circuit leakage reduction techniques are trying to decrease this component during the idle state. The sub-threshold current is expressed as:

$$I_{sub} = ke^{\left(V_{gs} - V_{th} / nV_T\right)} \quad (2.5)$$

The most effective way to reduce sub-threshold leakage is, therefore, changing the exponential component in equation (2.5) by either increasing V_{th} or reducing V_{gs}. The traditional way of increasing threshold is by changing the body bias.

2.4.1 **Adaptive body bias**

Changing body bias has been previously used to change the threshold of the transistors dynamically during operation [17][18][19]. The first order equation relating threshold voltage to back body bias is:

$$V_{th} = V_{th0} + \delta(\sqrt{|2\Phi_F + V_{SB}|} - \sqrt{|2\Phi_F|}) \quad (2.6)$$

where δ is given by:

$$\delta = \sqrt{2q\varepsilon N_{sub}} / C_{ox} \quad (2.7)$$

By changing V_{SB} in the above equation, the threshold of the transistor is changed. But, in recent technologies, two factors combine to make this technique less effective: body bias has a small effect on threshold since δ has become very small and also body bias causes additional drain leakage from band to band tunneling at the drain[20][21]. Band to band tunneling is caused by high electric field across the reverse bias p-n junction which causes the electrons to tunnel from the valence band of the p-region to the conduction band of the n-region. In scaled technologies, high doping concentrations and abrupt doping profiles causes significant band to band tunneling which increases by reverse body bias. Figure 2.5 shows the effect of body bias on different leakage currents of a 70nm predictive technology indicating that reducing substrate voltage does not necessarily reduce leakage current [21].

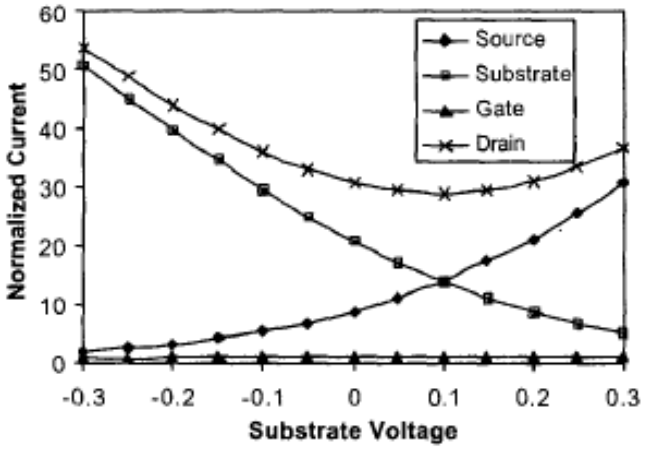

Figure 2.5. Effect of body bias on different leakage components (Figure from [21]).

Therefore, since changing body bias is not very effective in reducing leakage current in scaled technologies, people have looked for alternative methods to reduce leakage power. One approach is to use both high threshold and low threshold transistors and in fact, use low threshold transistors (that have high leakage) only when it is necessary in terms of performance. This is called variable V_{th} technology.

2.4.2 **Variable-V_{th} CMOS**

Mixed-V_{th} technologies reduce leakage by allowing use of transistors with different thresholds at different parts of the chip [16]. Figure 2.6 shows a very simplified circuit schematic using VTCMOS technology [16] where low V_{th} transistors are used in the critical path and high V_{th} transistors are used in the rest of the design.

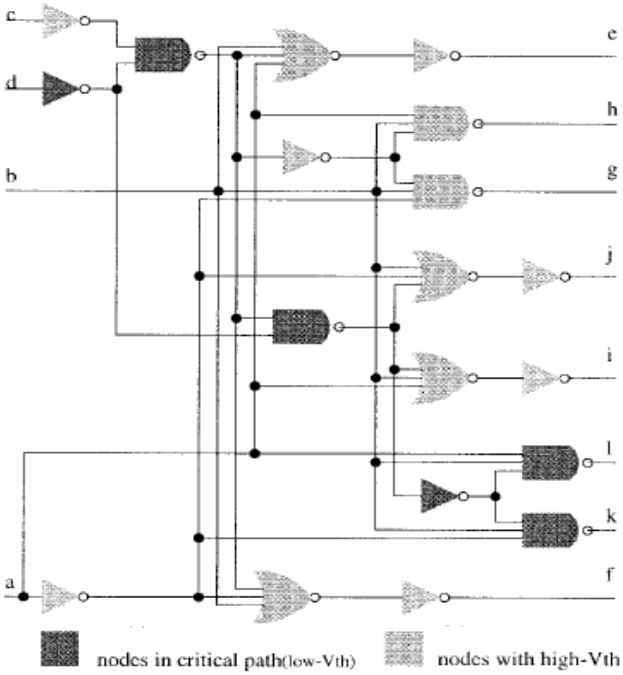

Figure 2.6. Simplified VTCMOS circuit scheme (Figure from [16]).

Using this technology, high leakage transistors are only used when they are needed from delay perspective and therefore, the overall leakage of the circuit is reduced (since the number of transistors that have high leakage during the idle mode is reduced). Although leakage power is reduced, it is still mainly determined by low threshold transistors. This means that if the circuit is not operating at its highest performance, the leakage power will be far from optimum. Also, the reduction of leakage takes place during the design time and it cannot adaptively change due to workload.

One alternative approach is to use power switches to shut off high leakage transistors when they are not needed.

2.4.3 **Multi-threshold CMOS**

Multi threshold technologies (MTCMOS) [14] provide the ability to control leakage power of the chip during design time by using high threshold transistors in

series with low threshold transistors to reduce the sleep leakage power [15]. Figure 2.7 shows an example of this idea.

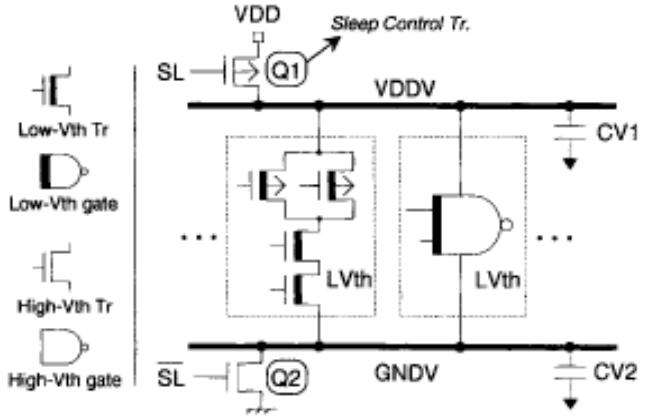

Figure 2.7. MTCMOS circuit scheme (Figure from [15]).

In active mode, SL is low and high-Vt transistors Q1 and Q2 provide the connection to the supply rails. These transistors should be sized large enough that their resistance is negligible. On the other hand, in sleep mode, Q1 and Q2 turn off and make the nodes connecting to the low-Vt transistors floating. The leakage of the circuit is mainly determined by the leakage of Q1 and Q2 and since these are high threshold transistors, overall leakage is greatly reduced. Therefore, although the threshold of the transistors have not changed using this method, but the overall standby leakage power is reduced since high leakage transistors are disconnected from the supplies and therefore do not leak. The overhead is that we need large high threshold transistors in series with the main low-threshold transistors of the circuit. Using these transistors has area overhead. Also some energy is spent in turning these transistors on and off during the normal mode and stand-by mode of operation.

Although multi-threshold technologies provide two leakage states, it can only crudely handle workload changes, and cannot really handle process variations.

An alternative approach to leakage control is to change the off state gate-source voltage of the transistors, which has the same effect as changing the threshold voltage. Therefore, we view this as changing the effective threshold voltage of the transistors.

The next section shows how one can use skewing the supply voltage to accomplish this.

2.5 Change the effective threshold by skewing supplies

The effective threshold of the transistors can be changed by changing the V_{gs} of an "off" transistor, which is normally 0V, to a different voltage. This can be accomplished by skewing the supplies of the gates with respect to each other. The basic idea is shown in the following figures for the case of a chain of inverters.

Normally the supplies of all gates are V_{ss} and V_{dd} but here the supplies of the first gate and the third gate in the chain are shifted to $V_{ss}+\Delta V$ and $V_{dd}+\Delta V$. The effect of this skew in the supplies is that the V_{gs} of the colored transistors (marked by a '*') changes by ΔV from an unskewed configuration. This shift in V_{gs} is equivalent to an opposite shift in effective V_{th}. Figure 2.8 shows that in the off-state there is ΔV voltage at the gates source and Figure 2.9 shows that during the active state Vgs is increased by ΔV.[2]

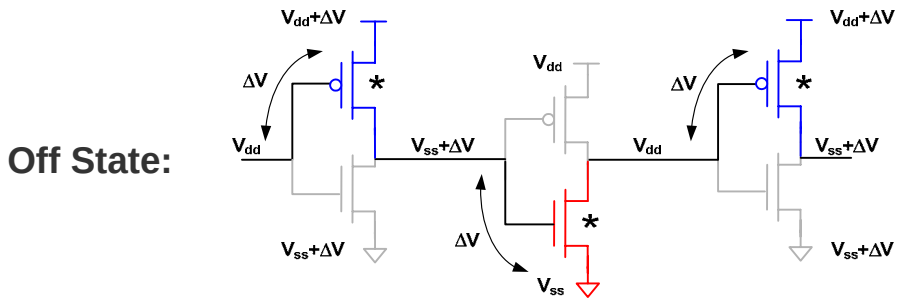

Figure 2.8. Off state for the colored transistors

[2] Substrate voltages are also shifted in triple-well technology (substrates are connected to the sources of the devices). However, if a triple well technology is not available, substrates should be connected to the main V_{dd} and V_{ss}. The shift in threshold caused by this is in the same direction of the shift caused by changing overdrive.

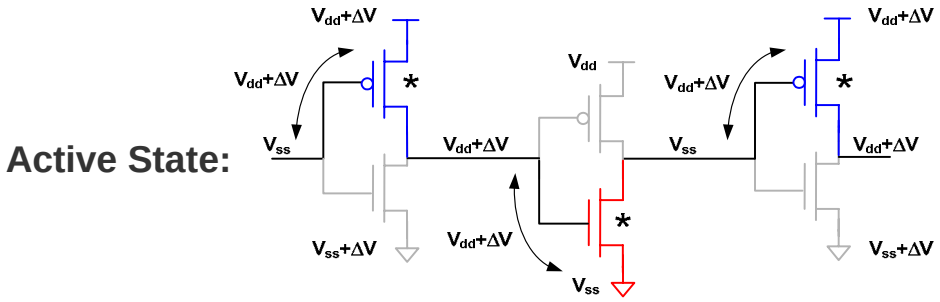

Figure 2.9. Active state for colored transistors

Positive ΔV leads to faster transitions and negative ΔV reduces leakage. However, this technique does not work for standard static CMOS gates because the effective threshold of the gray transistors changes in the opposite direction, making the other transition slower for positive ΔV and leakier for negative ΔV. Therefore, if this technique is used to achieve higher performance, it can only be used where the gate has a reset state and there is only one critical transition. If the purpose is to reduce leakage, the gate has to be in a state that leakage is determined by the colored transistors for most of the time.

Kohno [22] avoids the problem with static gates by separating the inputs to the nMOS and pMOS transistors to be able to have different standby levels for these signals. The logic is changed to a dual-rail logic with n-logic and p-logic gates that generate the inputs for nMOS and pMOS transistors of the next stages, respectively. The two outputs have the same polarity but the low or high levels are changed differently so that the effective threshold is not changed in the opposite direction for any of the transistors. Figure 2.10 shows the idea for a 'NAND' gate. V_L and V_H are the voltage levels that change the effective threshold for nMOS and pMOS transistors, respectively.

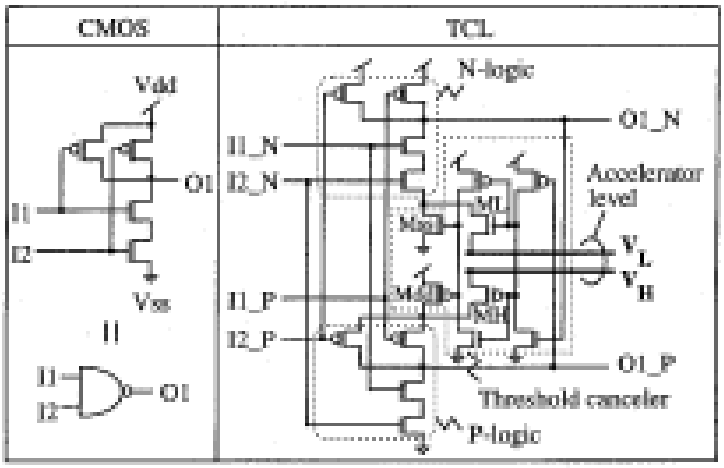

Figure 2.10. Using dual-line signals and n-logic and p-logic gates to adjust threshold (Figure from [22]).

Figure 2.11 shows the O1_N and O1_P outputs. In these gates, the effective threshold of the transistors can change in one direction (decrease). Therefore, the physical threshold is set for having low leakage and is decreased if higher performance is required.

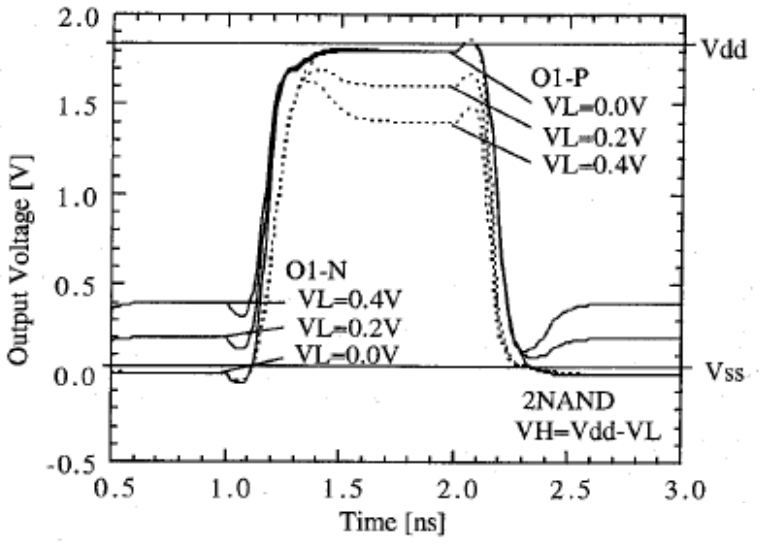

Figure 2.11. Output of n-logic and p-logic gates (Figure from [22]).

However, in order to achieve this, they had to double the signal wires, gates and energy consumption. Also, the level of the output signals are always changed to V_{dd} and V_{ss} and afterward the threshold canceling logic adjusts the levels. This also causes some extra power consumption.

The use of skewed supplies has also been studied for pre-charged gates. In the next chapter we will explain that although this is one possible way to create an adjustable threshold logic family, it does not achieve the full benefit. Pre-charged logic has extra power consumption and adjusting the effective threshold improves performance and does not reduce leakage power. After reviewing these, we discuss how we can create an adjustable logic family without these issues.

Chapter 3

Adjustable Threshold Logic Family

In this chapter, we investigate different ways of using skewed power supplies to create a logic family that enables the user to change the effective threshold of the transistors post fabrication. As described in Chapter 2, in order to be able to use this technique, we need to use logic families in which the gates have a default state. The simplest logic of this form is dynamic logic. After investigating the possibility of using dynamic logic and pulse-mode logic and describing the problems associated with each, we propose a new logic family, which we call pseudo-static, which addresses these issues.

3.1 Dynamic logic

Dynamic logic is used in high speed circuits to improve the performance by separating the set and reset paths in the circuit [23]. In dynamic logic, only one transition is optimized. The gates remain in a default pre-charged state and only evaluate in case output signal should be asserted. Pre-charging the nodes is done using a clock signal. An example of a dynamic gate is shown in Figure 3.1 where the nodes are pre-charged when the clock signal is low and at the rising edge of the clock, the gates will evaluate and some of the nodes will change their state. Since only one edge needs to be optimized for performance, it is potentially faster than standard static gates.

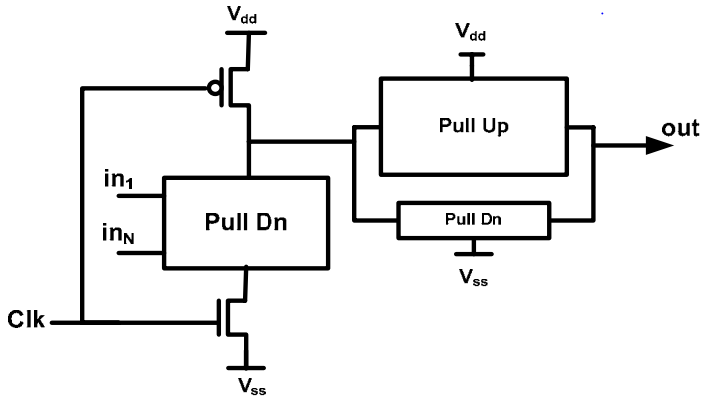

Figure 3.1. A simplified block diagram of dynamic logic.

Since the only transition that matters in terms of delay is the transition from the default state to the other state, we can use the positive effect of higher performance when we skew supplies to reduce V_{th}. The only constraint for the other transition is that pre-charging the nodes is done before clock goes high but since there is half a cycle time for this, the requirement is much more relaxed. In fact this is exactly the approach suggested by Solomatnikov [24] for skewed gates.

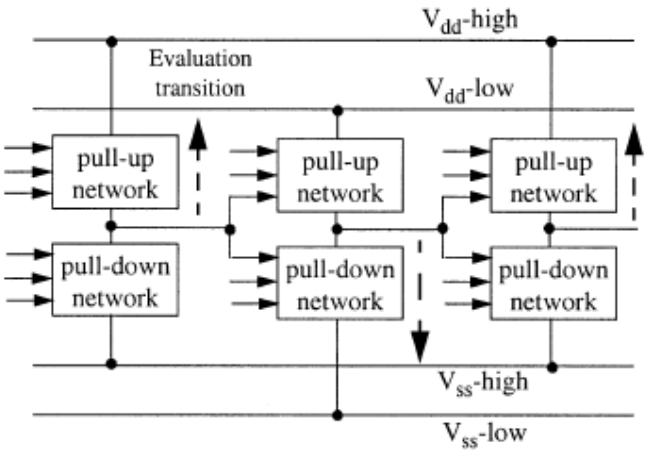

Figure 3.2. Use of skewed supplies for pre-charged gates (Figure from [24]).

In the above figure, V_{dd}-high and V_{ss}-high are used as supplies of every other gate and V_{dd}-low and V_{ss}-low are used as the supplies for the rest of the gates. In the first

stage shown, therefore, the pull-up network is strong, while in the second stage the pull-down network is strong.

Although this is one possible way to create an adjustable threshold logic family, it only helps in improving performance. The ability to reduce leakage power is limited since the gate will spend about ½ its time in the evaluated state where transistors have high leakage. In addition, dynamic logic has its own limitations:

- It is clocked, which means that even if inputs are constant, gates evaluate at the rising edge of the clock and nodes have high activity factor, causing the logic to consume extra power.
- There is an additional overhead for using dynamic logic in a system. The gates require monotonic signals at their inputs: Once the output is discharged because of an input combination, further changes in the inputs cannot change the output before the next cycle (that the nodes are pre-charged again). This is shown in Figure 3.3 for a simple case of a dynamic buffer. As it is shown a low-to-high transition of input during the time clock is high is acceptable (Figure 3.3 (b)) while a low-to-high transition of input during this time does not have any effect on the output and, therefore, is not acceptable (Figure 3.3(c)).

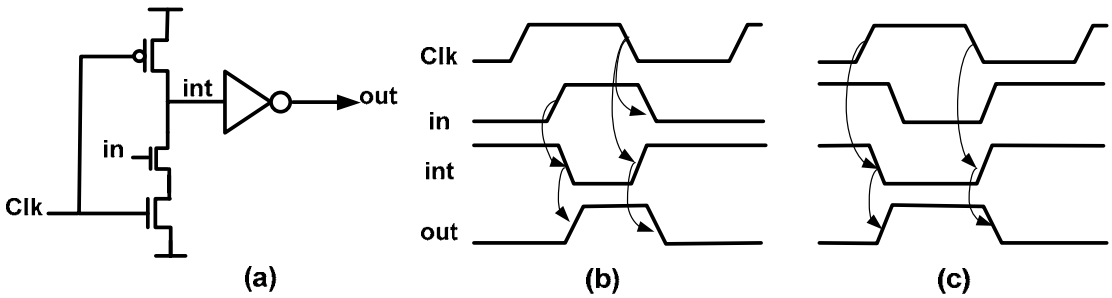

Figure 3.3. (a) A dynamic buffer, (b) with low-to-high transition of input during evaluation, (c) with high-to-low transition of input during evaluation.

Therefore, only monotonic transitions (transitions only in one direction during the evaluation time) are allowed at the input of the gates. This means that if the logic

requires both a signal and its complement, dual rail signaling should be used which in turn doubles the signal wires and consequently doubles the power consumption.

The issues of dynamic logic are particularly important in circuits where nodes do not normally have high activity factor. A good example of this is memory decoders [25]. In decoders, most of the gates do not transition at every clock cycle and therefore the overhead is large. In order to solve this, people have created a pulsed gate that is like a dynamic gate, but without a clock.

3.2 Pulse-mode logic

Pulse-mode logic is an alternative to dynamic logic where it has many of its good properties without using a clock. A simplified block diagram for pulse-mode logic (also known as post-charge or self-resetting logic) is shown in Figure 3.4. Similar to dynamic logic, all the nodes are in a default state but there is no clock signal. When an input comes, the gate evaluates. If the output is changed, the gate resets itself some delay after evaluation and remains in the default state until the next input changes.

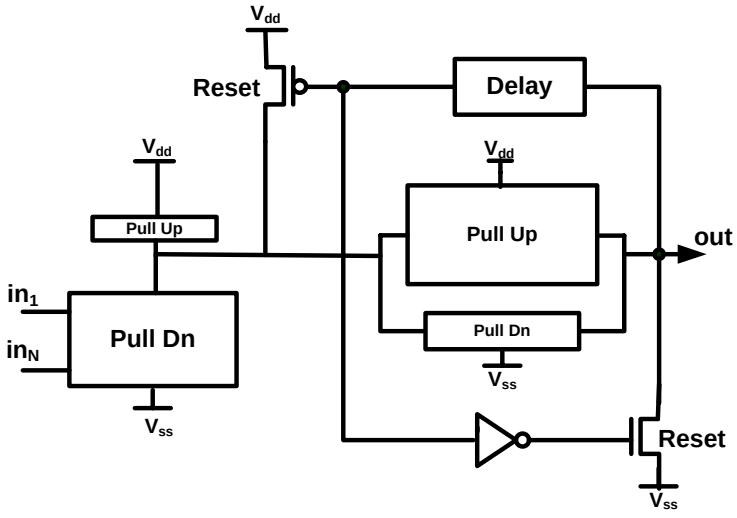

Figure 3.4. Simplified block diagram of a pulse-mode gate.

As mentioned earlier, pulse-mode logic is mostly used in fast memory decoders [25]. The logical function of the decoder is 2^n n-input 'AND'gates as shown in Figure

3.5. From all these 'AND' gates, only one is activated at any point of time (one memory address is chosen). This is the reason that using dynamic logic for decoders is not very efficient since there are many unchanged lines that will be clocked every cycle and therefore will have high power consumption. In fact, in memory designs having the wordline be a pulse can have additional power advantages. Since the bitlines have large capacitive loads and often sensed, pulsing the wordline can limit the total swing on these lines and save overall memory power.

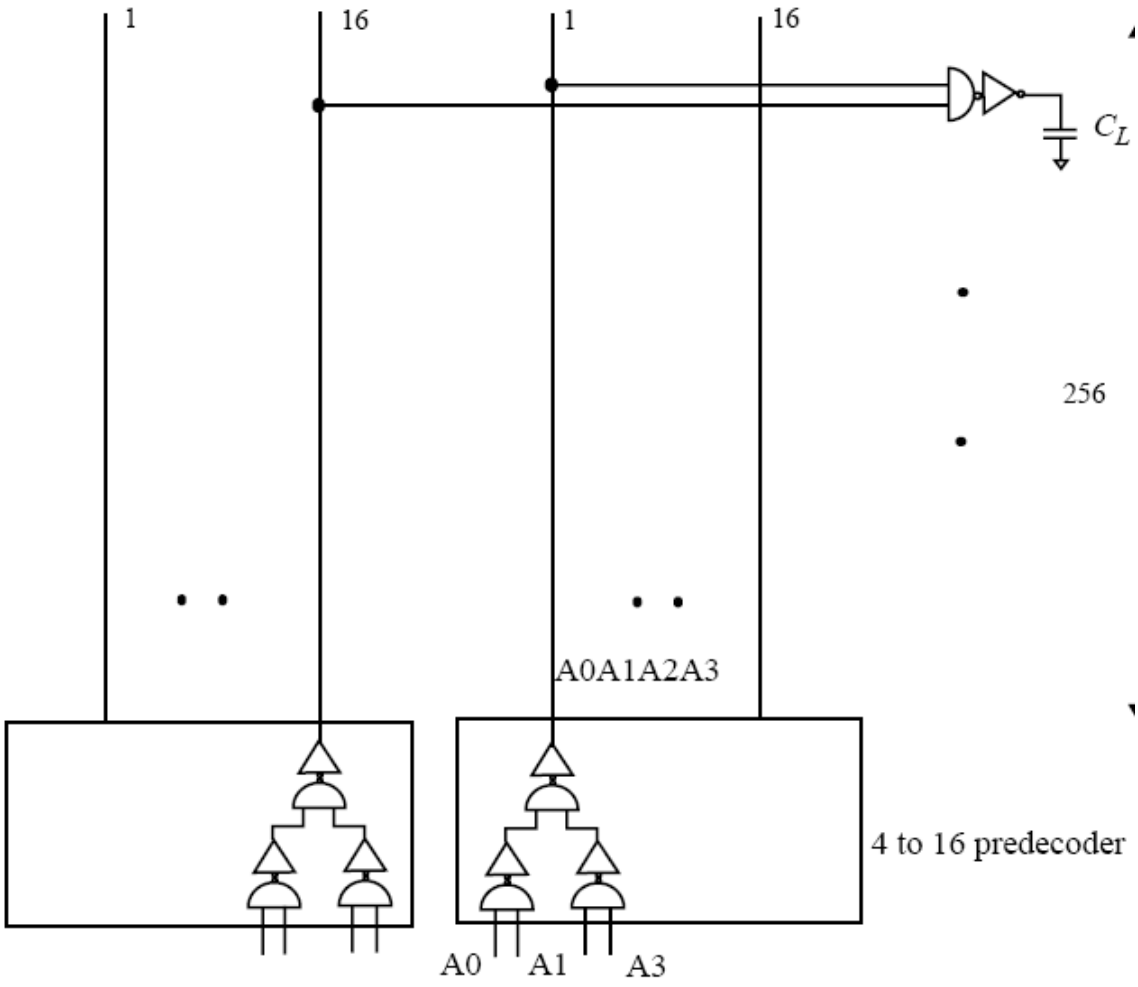

Figure 3.5. Schematic of a two level 8 to 256 decoder (Figure from [64]).

Since only one transition is critical in the forward path, the effective threshold of the critical transistors can be adjusted by skewing supplies. But using pulses has an

additional advantage (besides not requiring clock signal) over dynamic logic: short evaluation time. This means that, unlike dynamic gates, leakage of these gates can also be modulated for low power applications by skewing the supplies. Since the gate resets itself some small delay after evaluation (short evaluation time), the leakage of the circuit will be dominated by the leakage in the reset state.

On the other hand, using pulse-mode gates also causes system level problems. The main problem is that pulses corresponding to different inputs of a gate should overlap with each other to ensure correct logical operation. This makes designing circuit using pulse mode logic complicated since the timing is critical. Thus this type of logic is generally only useful in symmetric logic like decoders, where the delay of all paths match.

One way to get away from the tight timing requirements is to use pulses to communicate if a transition has occurred. Since pulse gates can only transition in one direction, a pulse in this system always shows that the corresponding signal has changed and this transition is propagated in the system instead of the true signal values. Conceptually, the true value of the signals should be stored at each gate, and a flip flop and a pulse generator should be added to the output of the gate to generate a pulse only if the state of the output has changed from its previous state. While this is a lot of overhead per gate, the overall system can be simplified and not all the gates have to do this conversion. One issue with this approach is higher power consumption since in a pulse mode gate each transition is a pulse: The gate is in default state, changes to the other state and resets to the default state again. Therefore, compared to a static circuit, the activity factor of the nodes is doubled. Using pulse-mode gates in this way, however, has two fundamental problems in addition to being power inefficient: meta-stability and pulse propagation which we will address when we talk about pulse-mode circuits in more detail in Chapter 5.

Alternatively, if we look at the source of these problems, the main reason is that the inputs of the gates are pulses with arbitrary arrival times. Therefore, if we can synchronize the inputs or avoid pulses at the inputs and outputs of the gates, we will avoid the problems. This is the basic motivation for pseudo-static logic.

3.3 Pseudo-static logic

While the previous solution has high overhead, inverting it by putting the pulses inside the gate, yields an interesting new design style. To achieve this, we separate the paths responsible for propagating the input rising and falling transitions into two different paths. By combining the outputs of the two paths to a single output, the gates will have static outputs and inputs while using pulsed nodes inside the gate. While these gates do use internal pulses, the overhead that these pulsed gates can cause is small, and is described in more detail in Section 3.5.

Splitting the internal gate paths has been done to improve the performance of IO buffers [26]. This splitting was done to shorten the delay of the circuit by using single edge optimized designs: while the internal gates are not pulse mode, the transistor sizes are skewed in opposite directions in the two paths, so the delay of the input falling transition is short on the bottom path and the delay of the input rising transition is short on the top path. These outputs are combined to have a single output at the end. The intermediate nodes of each path are combined using a simplified Muller-C circuit to choose between the two paths and eliminate contention current.[3]

We build on this idea to create pseudo-static logic which is shown in Figure 3.6. The top path and the bottom path are responsible for propagating the input rising and falling transitions respectively. Since there is a fast path and a slow path for each transition, the output of the two paths cannot directly be connected to each other (will cost both energy and performance since there will be a conflict at every input transition). Therefore, tri-state transistors have been used to prevent the conflict. Once the output has made a transition, the corresponding path is disabled and the other path is enabled by the delayed version of the output. The delay of the enable path has to be larger than the delay of the weak path to make sure that the second path has been

[3] The Muller C-element can suffer from meta-stability if the “wrong” timed pulse is input since it is deciding the value of its output based on the two input pulses.

resolved before its corresponding output driver is turned on. This sets the minimum output pulse for the gate.

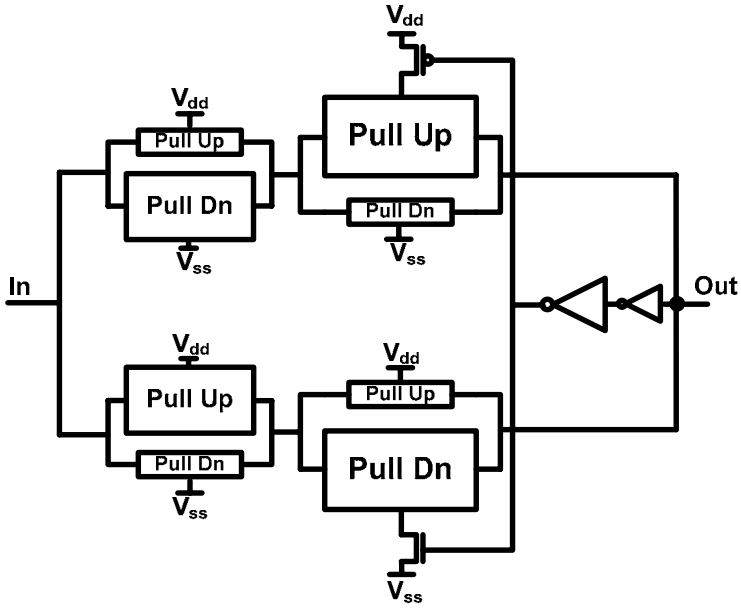

Figure 3.6. Block diagram of pseudo-static logic.

While this initial circuit implementation is functional, it can be improved by moving the tri-state away from driving the large output load. Having tri-state series transistors at the output stage costs both energy and performance. An improved alternative is shown in Figure 3.7, where the enable transistors are moved to the first stage. In this case, the inverted version of the output has been used to disable the driving path, reset the internal nodes and enable the other path after output transitions, essentially making the internal gates pulse mode gates. In this gate, however, since there is a loop from input to the output, the output can become meta-stable if the gates are not sized correctly.

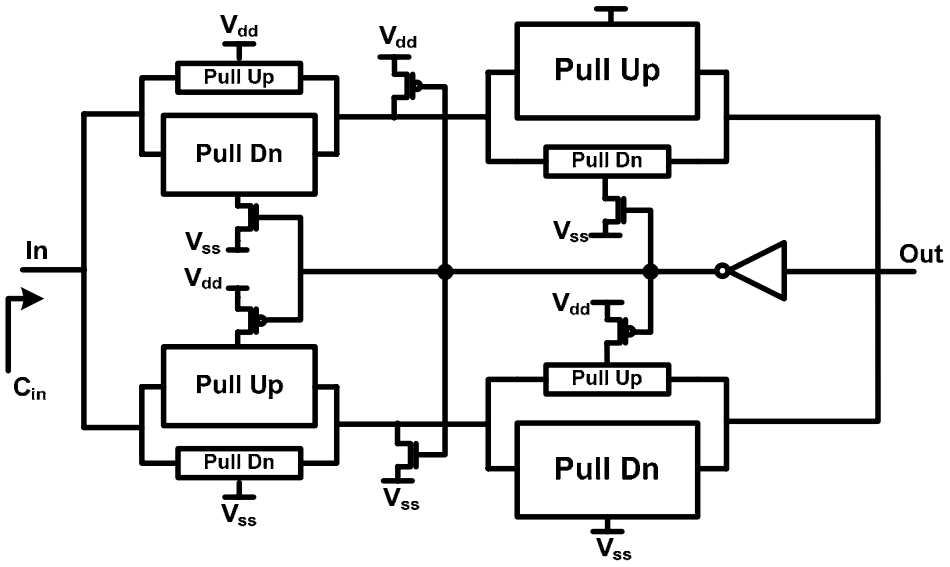

Figure 3.7. Moved tri-state transistor to the front stage.

Figure 3.8 shows an improved architecture without the meta-stability problem. The weak drivers in the last stage are removed and an additional weak path from input to output has been added to ensure that the output is always driven by the input, and eliminates any meta-stable conditions. This architecture is more efficient than the first one and since it does not need to wait for the weak path, the minimum possible pulse width is smaller than both the first architecture and the split I/O proposed by Hamzaoglu in [26].

The resulting pseudo-static circuits have the following characteristics:

- The input and output are static signals and therefore this gate can readily replace any static circuit in a system and no further change in the system is required.
- There is no extra power consumption at the load compared to the static circuit since outputs have the same activity factor as the static outputs.
- It is faster than the static circuit since each path can be optimized for one transition like a dynamic gate. It is slower than dynamic [or pulse mode logic] because the input essentially drives 2 pulse mode gates in the first stage. If we

increase the number of stages in the gate, it asymptotically approaches the speed of dynamic logic.

- There is no meta-stability since there is always a path from input to the output and no decision needs to be made.

The most important property of this gate is that, unlike the static gates, we can adjust the effective threshold of all critical transistors after fabrication as it is explained below.

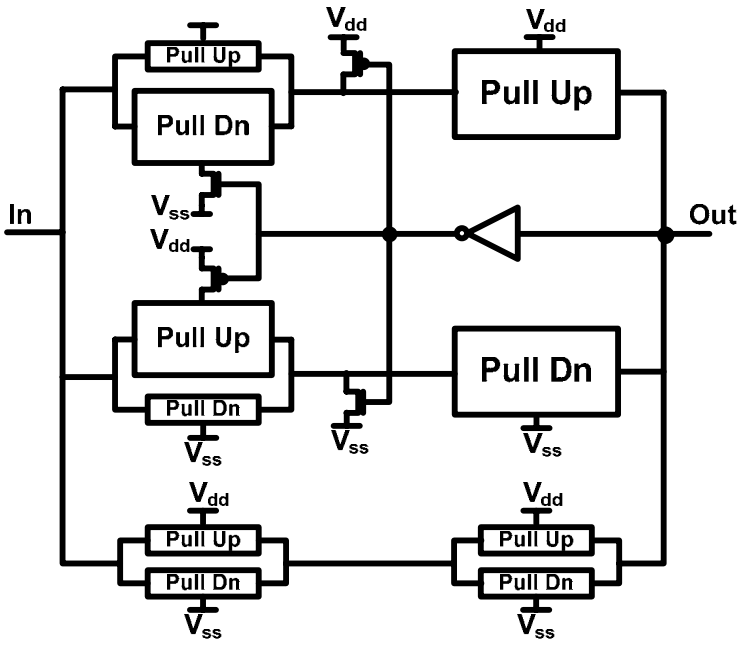

Figure 3.8. Improved pseudo-static architecture

Figure 3.9 shows the supply configuration used to adjust the effective threshold of the transistors. The input and output signal levels are V_{dd} and V_{ss} while the supplies of the first stage of the top path and the bottom path are shifted by $-\Delta V$ and $+\Delta V$ respectively. Since the output levels are V_{dd} and V_{ss}, from the system point of view, changing the supply is not seen by other parts of the system and happens internal to the gate. With the supply configuration shown, the effective threshold of all critical transistors (for both transitions) is reduced by ΔV. Positive ΔV boosts the performance

of the gates while negative ΔV reduces the leakage power. All the nodes have equal swing of (V_{dd}-V_{ss}) and dynamic power remains constant when ΔV changes.

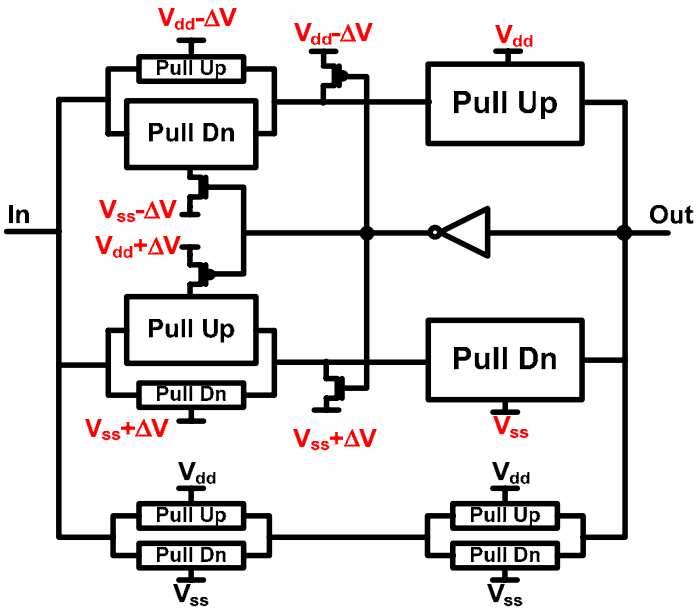

Figure 3.9. Supply configuration for adjusting threshold

The downside of this approach is that 4 additional supplies are required to adjust the threshold of all critical transistors. We can also reduce this number to 2 with the supply configuration shown in Figure 3.10, where we only shift the ground of the first stage of the top path to V_{ss}-ΔV and the supply of the first stage of the bottom path to V_{dd}+ΔV. In this scheme, the effective threshold of all the critical transistors are reduced by ΔV with the difference that the swing of the internal nodes, int_f and int_r, is increased by ΔV. The effect of this increase in the swing is that it increases the delay of the first stages in each path. The net effect is that in order to get the same performance as the previous case, we need to use slightly higher ΔV. As we will show in the measurement results, the amount of this increase in ΔV is not large.

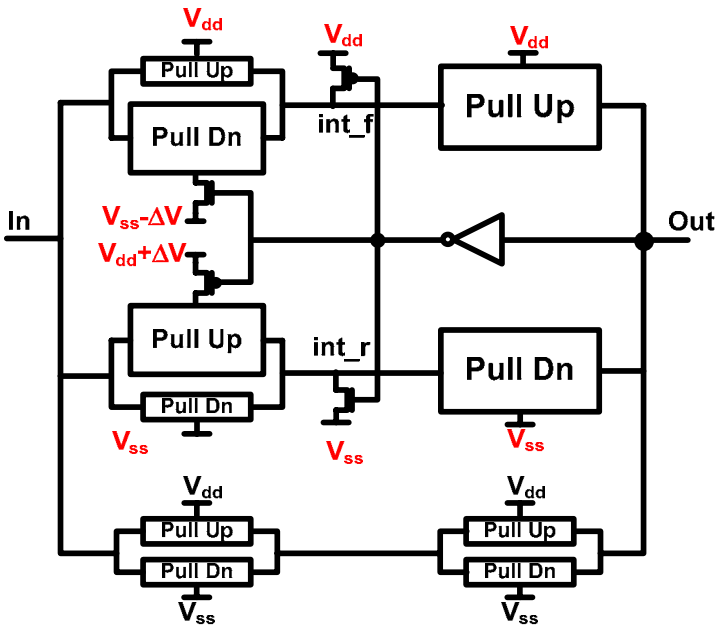

Figure 3.10. Adjusting threshold with fewer number of supplies

3.4 Design of pseudo-static gates

In this section, we briefly consider the energy and performance of a pseudo-static circuit and compare it to other logic families. Then we discuss the general considerations in designing these gates by providing an example of an "AND" gate.

3.4.1 Logical effort

In order to compare the performance potential of different circuits, we can compare the logical effort [27] of the gates with each other.

Let us consider a simple inverter chain as an example. The inverter chain consisting of static gates is shown in Figure 3.11. The logical effort[4] for each static inverter is defined as 1. A pulse-mode inverter chain is shown in Figure 3.12. The gray

[4]Logical effort is a measure of the strength of a gate compared to an inverter: $LogicalEffot = \frac{C_{in\_gate} \times R_{out\_gate}}{C_{in\_inv} \times R_{out\_inv}}$)

part is the reset circuitry. Here we have assumed that the strength of the weak path is 1/5 of the strength of the strong path. The two differently skewed inverters are shown with two different colors.

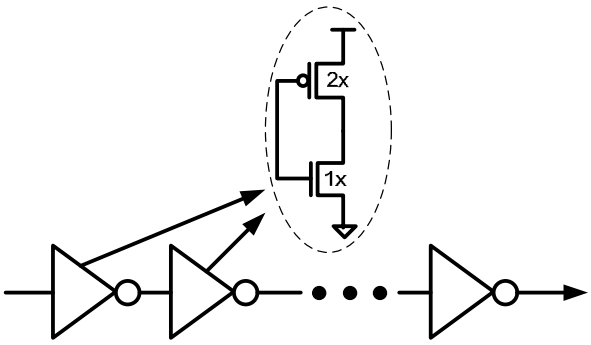

Figure 3.11. Static inverter chain.

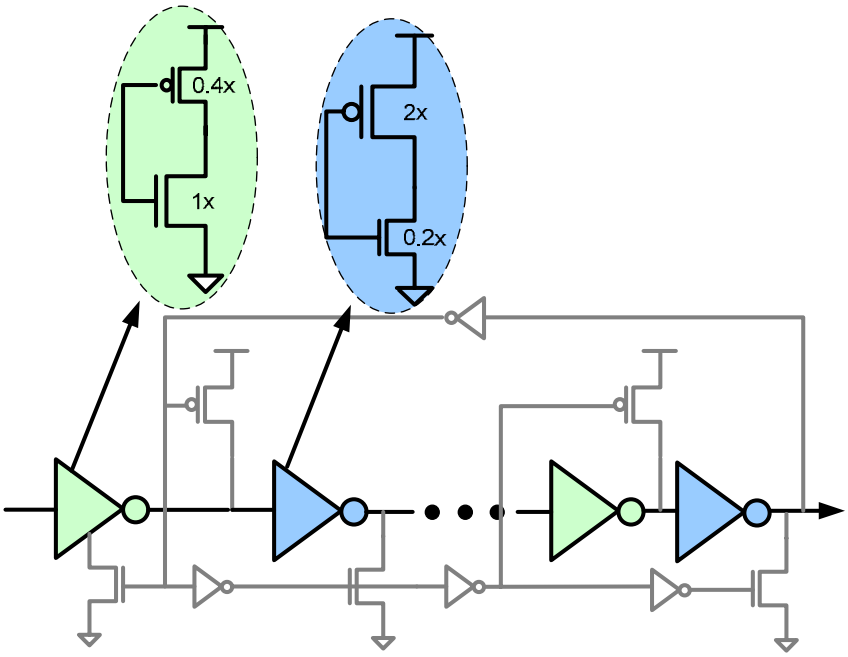

Figure 3.12. Pulse-mode inverter chain.

The logical effort for the odd inverters in this chain is $\frac{1.4}{3}$ and for the even inverters is equal to $\frac{2.2}{3}$. Therefore, the average logical effort for every stage is $\sqrt{\left(\frac{1.4}{3}\right)\left(\frac{2.2}{3}\right)} = 0.58$. For an inverter chain of length N, the effort of the path will be $(0.58)^N$.

The logical effort for the pseudo-static gates is similar to the logical effort of the pulse-mode gates except for the branching of two at the input and the load of the weak path for the input.

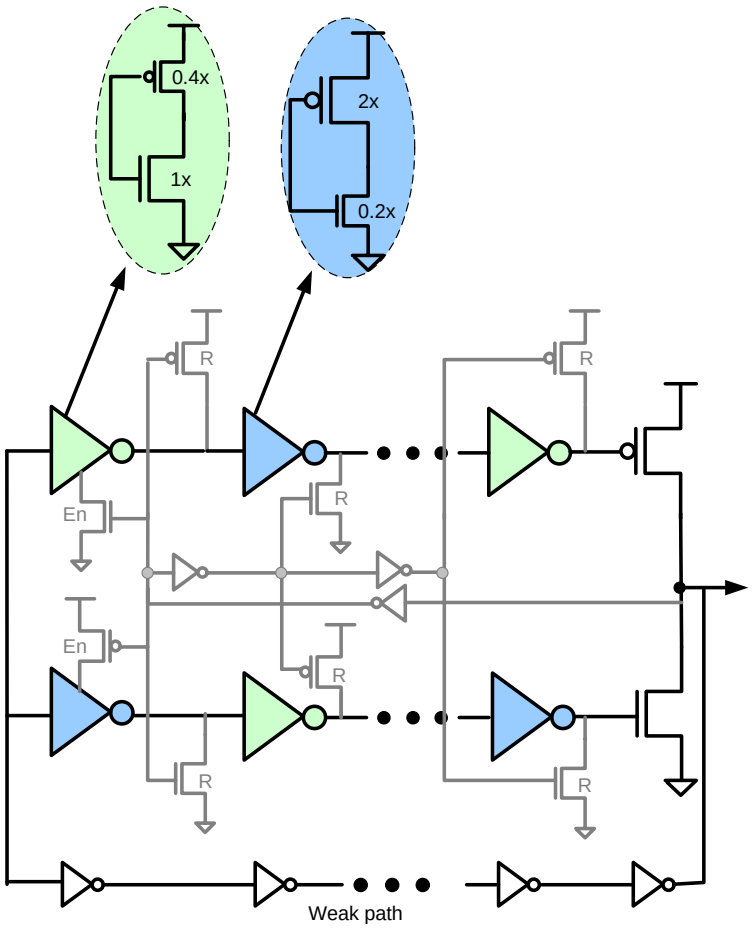

Figure3.13. Pseudo-static inverter chain.

For the intermediate stages, the logical effort is equal to $\frac{1.4}{3}$ and $\frac{2.2}{3}$ except for the last stages. Logical effort is equal to $\frac{2}{3}$ and $\frac{1}{3}$ for output pMOS and nMOS drivers respectively. Assuming we have N stages:

Effort of the top path is calculated in Equation (3.1):

$$\left(\frac{1.4}{3}\times\frac{2.2}{3}\right)^{(N-2)/2}\times\frac{1.4}{3}\times\frac{2}{3} \tag{3.13}$$

Effort of the bottom path is calculated in Equation (3.2):

$$\left(\frac{2.2}{3}\times\frac{1.4}{3}\right)^{(N-2)/2}\times\frac{2.2}{3}\times\frac{1}{3} \tag{3.2}$$

In theory, we size the top path and the bottom path such that both of them have the same delay:

Delay of the top path:

$$D_1=\left(\frac{C_{out}}{C_{in1}}\right)\times(0.58)^{(N-2)}\times\frac{2.8}{9} \tag{3.3}$$

Delay of the bottom path:

$$D_2=\left(\frac{C_{out}}{C_{in2}}\right)\times(0.58)^{(N-2)}\times\frac{2.2}{9} \tag{3.4}$$

Equalizing delay of the two paths:

$$D_1=D_2\Rightarrow C_{in1}=1.27\times C_{in2} \tag{3.5}$$

Assuming that the weak path input capacitance is 0.2 times the input capacitance of the two main branches:

$$C_{in} = 1.2 \times (C_{in1} + C_{in2}) = 2.724 \times C_{in1} \quad (3.6)$$

Substituting Cin1 in the path delay equation (3.6) gives:

$$D_1 = \left(\frac{C_{out}}{C_{in}}\right) \times 2.724 \times (0.58)^{(N-2)} \times \frac{2.8}{9} = \left(\frac{C_{out}}{C_{in}}\right) \times (0.58)^{(N-2)} \times 0.85 \quad (3.7)$$

If the we define the effort of the path as the average of these two and include the branching factor of 2.2 (2 for the two main branches + 0.2 for the weak path with strength of 1/5), the total effort of the path will be $2.2 \times (0.58)^{(N-2)} \times 0.275$. Therefore we can define the average LE of each stage as: $\left(2.2 \times (0.58)^{(N-2)} \times 0.275\right)^{1/N}$. This shows that as we increase the number of stages in a path, the effect of the branching factor becomes less significant. Figure 3.14 shows the effective LE of each stage based on the number of stages:

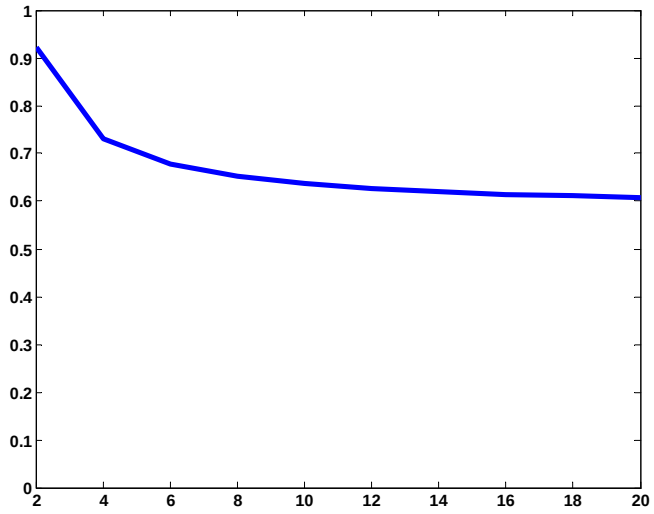

Figure 3.14. Effective LE per stage vs. number of stages for pseudo-static logic

Based on this figure, if we use 6 or more stages of logic in a pseudo-static block, the effect of branching of the input becomes insignificant and the effective LE is 0.67 for 6 stages of logic. For only 2 stages, the effective LE is 0.92 which is slightly still better than the logical effort of the static gate but worse than pulse mode.

3.4.2 **Activity factor**

For the static circuit all the nodes have at most one transition. Therefore, assuming equal number of one and zero transitions for the input, the activity factor is 0.5. For the pulse mode circuit, the nodes in the critical path have two transitions each time and the activity factor is 1. For the pseudo-static circuit, each time the gate has a transition, half of the nodes in the main path have two transitions while the other half are constant. Therefore, the average activity factor for pseudo-static signal nodes is 0.5. The reset driving nodes make one transition each time and also have activity factor of 0.5.

3.4.3 **Pseudo-static AND gate design example**

Figure 3.16 shows the schematic of a pseudo-static AND gate. The size of the weak transistors should be determined based on the noise margin requirement of the system. For example for our proposed circuit in Chapter 5, noise coupling is limited (since the circuit has high fanout) and, therefore, we used weak transistors which had 1/5 the strength of the strong transistors. All weak transistors (in the top path, the bottom path and the output driving transistors) used the same 1/5 ratio.

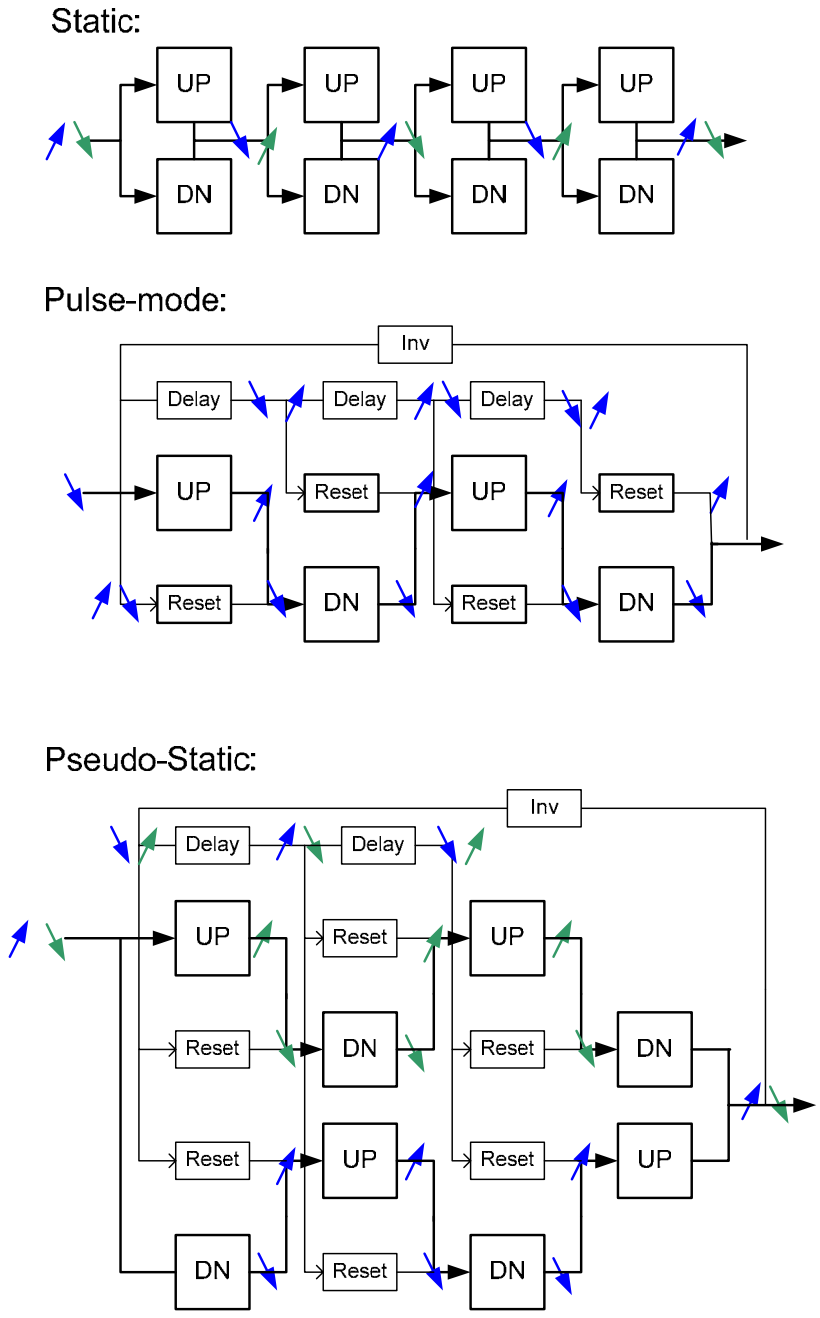

Figure 3.15. Activity factor for different logic styles. Each color arrow shows transitions of each node for a particular input transition. As we can see each node in pulse mode circuit has two transitions per for one input transition. In static and pseudo-static each node has one transition.

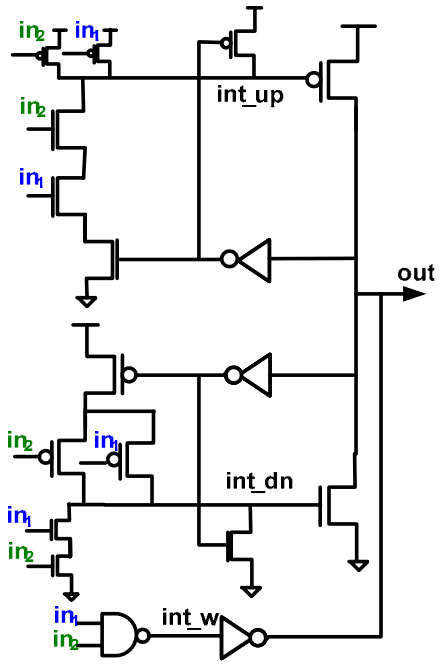

Figure 3.16. A pseudo-static AND gate.

The transistors responsible for propagating the input signals to the output can be sized for power efficient or high performance the same as any other logic. In order to prevent any contention between the weak path and the strong path we should have:

$$\text{int\_}w.Trise \leq \text{int\_}dn.Trise \quad [5] \tag{3.8}$$

$$\text{int\_}w.Tfall \leq \text{int\_}dn.Tfall \tag{3.9}$$

For the reset transistors, there is a tradeoff between how much energy and area should be spent for reset and how much glitch overhead the circuit has for single glitches.

In terms of performance there are four paths to consider:

[5] int_w.Trise means time that rising edge of a transition at int_w node gets to half supply voltage.

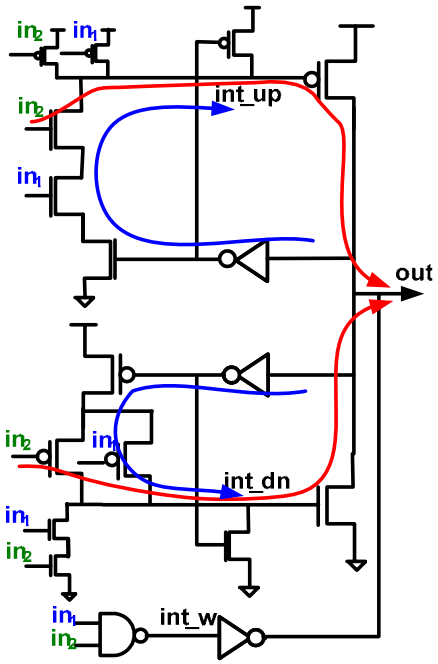

Figure 3.17. Different paths that has to be considered for delay and overhead.

The red paths shown on Figure 3.17 are the forward paths from inputs to the output that has to be considered to minimize delay of the circuit given the energy constraint. The blue paths are the paths from output to the reset of the internal nodes that has to be considered to minimize the glitch overhead.

In summary, pseudo-static logic has many advantages. It looks like static logic from the system point of view, can provide slightly higher speed, and enables the user to optimize for both dynamic and leakage power after fabrication.

With these advantages come a few issues that needs to be considered before using this circuit in a real design:

Noise coupling is a potential issue since the weaker default driver means slightly larger noise coupling. Highly skewed gates in general have larger noise sensitivity than static circuits (but better than pre-charged gates). The sensitivity to this issue depends on the specific applications. As it will be discussed in Chapter 4, for the FPGA interconnect circuits implemented using these gates, our measured results

indicate that the noise margin is sufficient for this not to be an issue, mainly because the large fan-out of the interconnect makes the coupling capacitor of the wire a small fraction of the overall load. For a general circuit, the noise considerations determine the ratio of the weak to strong drivers.

3.5 Pseudo-static timing

While pseudo-static logic uses normal CMOS level signals for the input and output, it does use pulses internally. We need to analyze the limitations in performance that these pulsed gates cause. This section shows that while there is a potential issue with the second edge of a glitch being delayed and slowing the critical signal, for the pseudo-static logic, the maximum slow down is small and is independent of the logic depth. Thus, it can be easily handled in a system timing analysis. After setting up the analysis framework, the next two sub-sections describe glitch expansion for a single gate and then a string of gates.

As mentioned earlier, pseudo-static gates do not have a meta-stability problem since the output is a level and can take any value (there is no decision required). But the internal pulse gates can slow down the 2^{nd} edge, leading to glitch expansion. The output transition rate in pseudo-static gates is limited by the reset path delay. After the output makes a transition, the corresponding path should reset and the other path should be enabled to be ready for the next input transition. Therefore, input glitches which have a width of less than the reset path delay will be expanded at the output. This glitch expansion can affect the worst case delay situation since the delay of a gate is defined as the delay from the last input to the last output transition. For static gates, this problem does not exist since static gates have the nice property that the propagation delay for the second edge is never more than the propagation delay for the first edge. Figure 3.18 shows the concept. To evaluate the glitch overhead in the measurement setup, a simple buffer is assumed. An input pulse goes through the static and pseudo-static circuits. If the input pulse is very short, both gates will filter the pulse and there is no transition at the output of the gates. If the input pulse width is

wide, both gates propagate the pulse to the output with the same width. However, for intermediate input pulse widths, the output pulse width of the static circuit is always less or equal to the input pulse width while the pseudo-static output can have a wider pulse width.

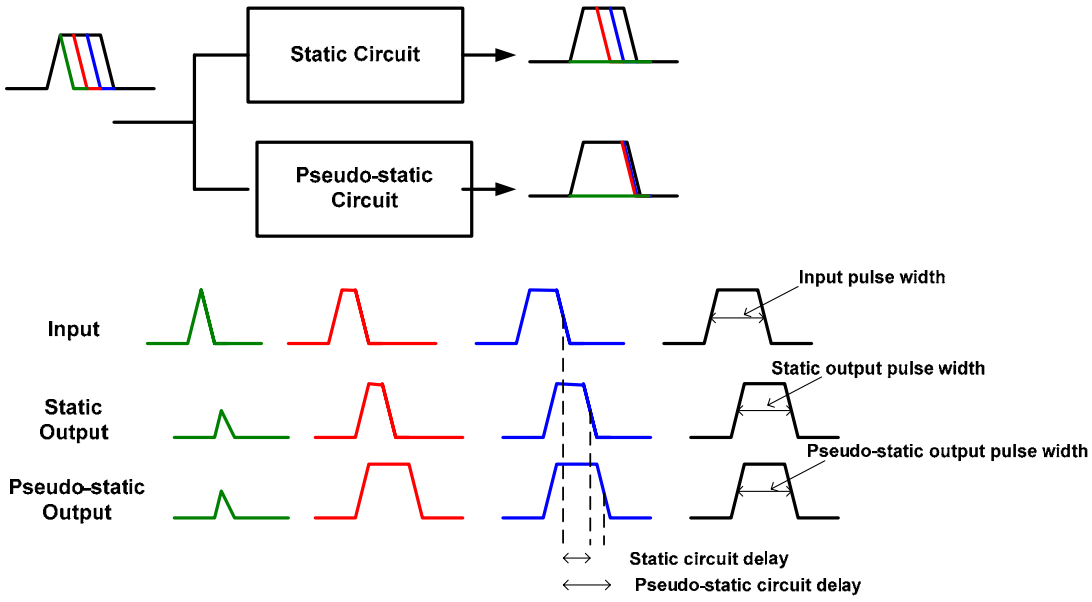

Figure 3.18. Hypothetic illustration of pulse expansion in pseudo-static circuit

The important question here is that what the overall effect of this glitch expansion to the system is. If, for example, only one of the gates in the critical path expands a glitch, the overall effect in not significant. If, however, the cascading of the glitches in the worst case causes unbounded overhead, this makes the logic non-practical. We first consider the problem of pulse expansion for the two cases of having one-input one-output gates and having multiple input gates. For a single input gate, we use the example of pseudo-static and static interconnect that we implemented. The implementation of these circuits is discussed in Chapter 4.

3.5.1 Glitch Expansion for the single input gates

The simulated response of a simple CMOS buffer to an input pulse is shown in Figure 3.19 Figure 3.19(a) shows the measured delay for the last input to last output transition versus input pulse width and Figure 3.19(b) shows the output pulse width for different input pulse widths. As expected, we can see in this figure that very narrow input pulses are filtered and there is not any transition at the output (delay is defined as zero). As input pulse width increases output pulse width also increases and eventually becomes equal to the input pulse width.

Figure 3.20 shows the same plots for a pseudo-static buffer circuit for the two cases of zero skew and 200mV skew in the supplies. Since the overhead in the pseudo-static circuit is determined by the reset path delay, we reduced this overhead by increasing the cost of the reset path in optimization. Shortening the reset path delay would decrease the glitch overhead since it shortens the required time for switching between the two paths. However, the reset delay should be long enough that the output transitions full rail before the driving transistor is turned off. As this figure shows, for the case of zero skew, the overhead is not significant. However, for the case of 200mV skew between the supplies, since the ratio between the reset path delay and the main path delay changes, the gate will have some delay overhead for input pulse widths between 40ps and 80ps. From Figure 3.20, we can also see that even in the case of 200mV skew, the delay with overhead is still less than nominal delay of the case without skewing.

The cost of this glitch expansion depends on the circuit's context. For some circuits, like the FPGA interconnect example described in the next chapter, one cannot create glitches in the intermediate stages since the interconnect blocks have only one active input. Thus, in the cascade of these blocks, in the worst case, only the first stage can expand an input glitch (which comes from a logic block), but the rest of the stages will just pass the expanded glitch and will have full performance benefit. However, most gates have more than one input, which requires a more careful analysis.

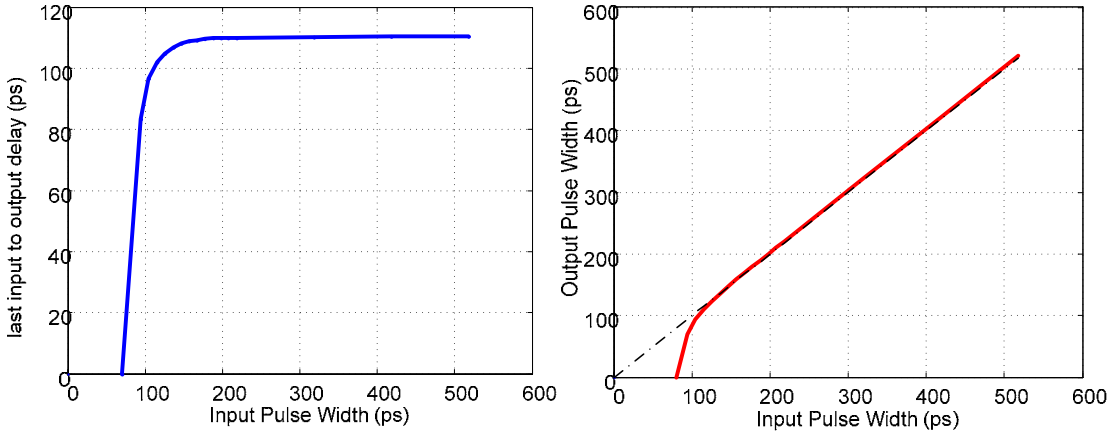

Figure 3.19. Response of a static gate to a glitch.

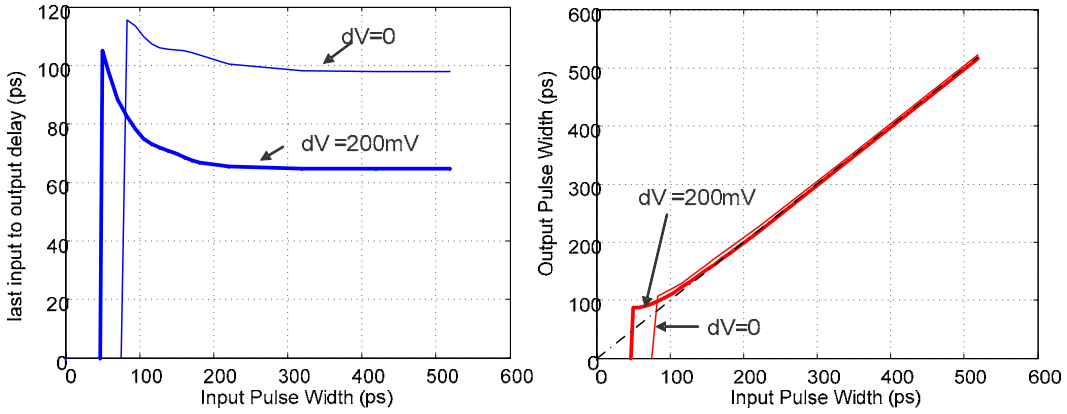

Figure 3.20. Response of a pseudo-static gate to a glitch.

3.5.2 **Glitch cascade**

The potential worst case situation is when one input to a gate has a glitch, and another input of the gate makes a transition. Depending on the arrival time of the inputs with respect to each other and the logic of the gate, we can have multiple close transitions at the output and need to determine the worst case overhead. It is this situation in pure pulse mode circuits that can add a glitch overhead for each stage in the logic. Let us assume a two input gate and the case that there is already an input pulse on one of the inputs.

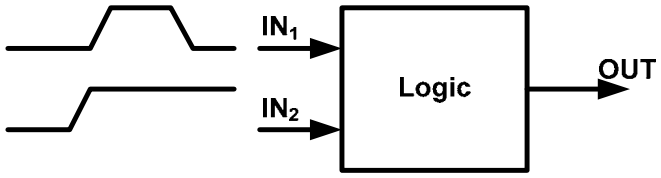

Figure 3.21. Worst case glitch consideration for multi-input gates.

If only one of the inputs changes the output, the situation is the same as for single input gates. If both of the inputs change the output, cascading can occur. In this situation, if the glitch input to the gate is the latest input and arrives when the output of the gate has already settled, this gate will just pass the glitch to the output and will not add any delay overhead. If the glitch input is early, delay is determined by the other input and therefore the glitch overhead is actually reduced. The potential worst case situation is when the non-glitch input arrives early, changes the state of the output and each of the edges of the glitch input change the state of the output as well. In this case, output will have multiple transitions and can further delay the edge that is already delayed. To make these three timing scenarios clearer, each is described in more detail below:

1. IN_1 arrives sooner and changes the output:

- IN_2 arrives after IN_1 pulse has finished: At most one glitch overhead is added to the output depending on the arrival time on IN_2. In this case, although the second edge of IN_1 was delayed, the output delay is mainly determined by IN_2. The only way the glitch hurts is when it becomes close to IN_2 and causes delay in propagating IN_2 but this is still one glitch overhead from the latest input (IN_2).

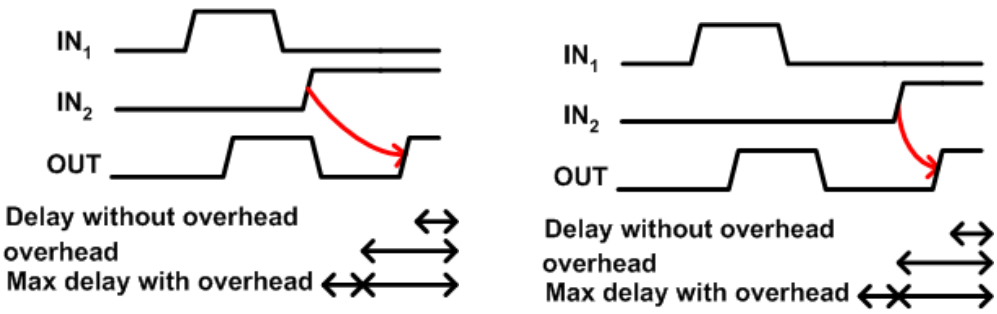

Figure 3.22. IN_1 arrives early, both IN_1 and IN_2 change the output.

- IN_2 arrives somewhere in the middle of the glitch: No additional overhead in this case.

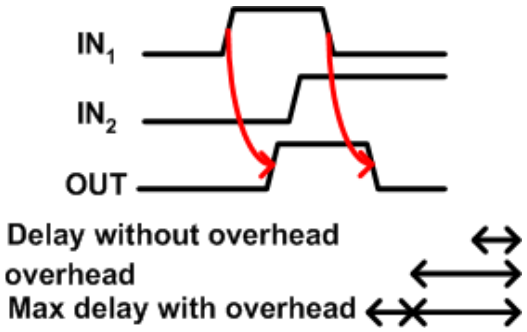

Figure 3.23. IN_1 arrives early, IN_2 arrives in the middle of the glitch.

2. IN_2 arrives earlier, changes the output. IN_1 arrives a long time after output has changed. Delay is just delay of IN1 and therefore no additional overhead.

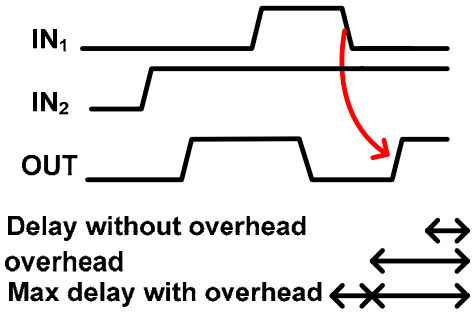

Figure 3.24. IN_2 arrives early but does not change the output.

3. IN_2 arrives sooner, changes the output, IN_1 arrives with short time delay and changes the output again. This is the worst case since the edge that is already delayed gets further delayed.

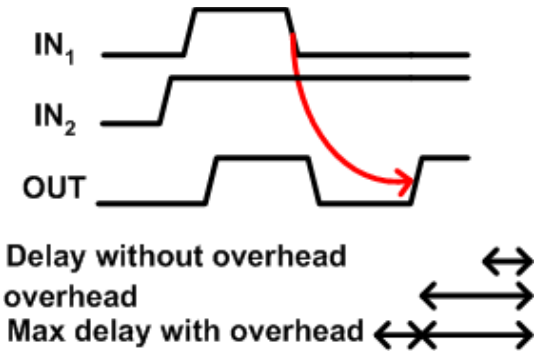

Figure 3.25. Worst case situation.

The probability of having this worst case is not very high since it requires specific timing between the inputs and also depends on the logic of the gate (output should change with every input transition). In other words, the logic of the gate filters the glitches to some extent. For example, for single glitches, the only glitch that is possible at the output of an ‘or’ or ‘nand’ gate is a ‘101’ glitch. Similarly, the only single glitch possible at the output of an ‘and’ or ‘nor’ gate is a ‘010’ glitch. Therefore, if, for example, we implement a decoder using pseudo-static gates, the overhead that we can get for the whole decoder is just the overhead of a single glitch and the rest of the stages will have full performance benefit since decoder only consists of a cascade of ‘AND’ gates that will not generate additional glitches.

The most problematic gates are the XOR gates where the output changes for every input transition. Figure 3.26 shows the simulation waveform for the case that an extended glitch arrives at the input of an XOR gate. The blue waveforms correspond to the case that only one input has a glitch and the other input is constant. The green waveforms correspond to the case where the first input has a glitch but the second input has a transition as well. As it can be seen from this figure, the additional overhead for cascading glitches is much less than the overhead of having a single glitch.

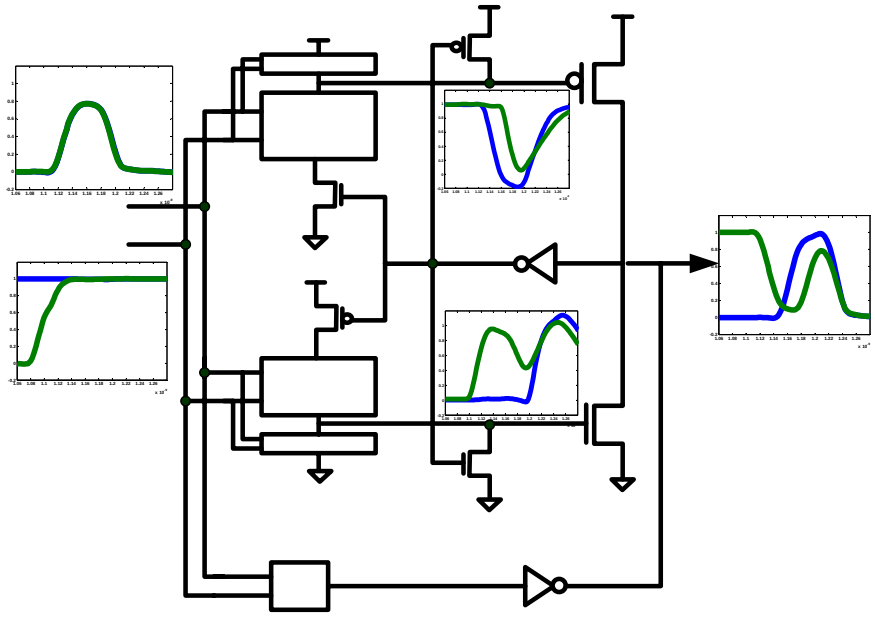

Figure 3.26. Simulation waveforms for a two input XOR gate

The reason is that for a single glitch each transition was driven by a different path. After the first transition, some time is spent for the reset circuitry to disable one path and enable the other path and after that the second path can propagate the transition. However, in case of multiple glitches, every other transition is driven by the same path. For example in case of the three transitions in Figure 3.26, the first and third transitions are driven by the bottom path and the second transition is driven by the top path. Since the reset time is small, the internal nodes are not completely reset before being set again and therefore the propagation delay for the third transition is much smaller than the propagation delay for the first transition. In other words, each path filters the propagated transitions on its own and since each path drives multiple transitions the later ones are filtered. This means that, although pseudo-static gate does not have the filtering property of a static gate for same width single glitches, for multiple glitches, it does filter since each path is the same as a static path for its own transitions.

Therefore, glitch expansion and glitch cascading can happen in pseudo-static gates, but the overhead is limited and is not a fundamental problem for timing analysis of the system. Our simulations show that in estimating overall delay of the logic, we

can ignore the glitch expansion completely and add two times overhead of a single glitch to the total estimated delay to consider the worst case situation.

3.6 Additional supplies requirement

In order to be able to change the effective threshold we use multiple supplies that are skewed with respect to each other. Therefore, additional supplies (V_{dd}+dV and V_{ss}-dV) are required. Using additional supplies has certain issues that need to be considered. The coupling between these supplies and generation and distribution of the extra supplies are the major considerations. In this section, we discuss different ways that these supplies can be generated. By looking at the characteristic of these supplies we provide some simple possible ways to deal with them.

The simplest way is to bring the supplies from outside the chip. Figure 3.27 shows the coupling between the supplies. If we want to bring the supplies from outside we can use four planes for the supplies with the order shown in Figure 3.28 to maintain good coupling between them.

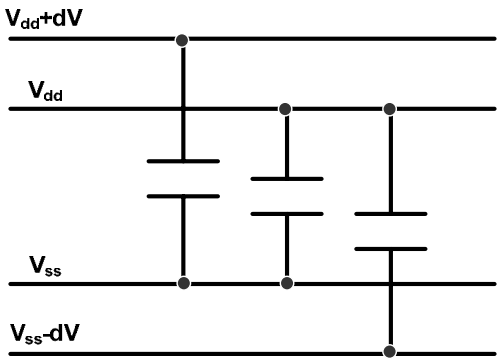

Figure 3.27. Coupling between different supplies

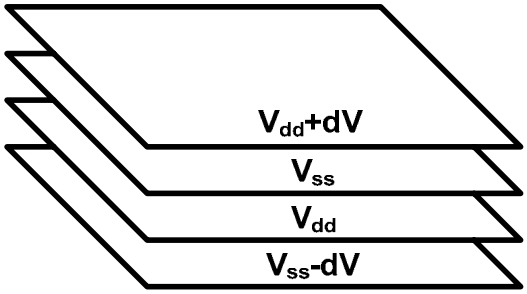

Figure 3.28. Supply planes

The alternative to avoid the cost of having four supply planes is to generate the supplies on the chip. Before going through the problem of generating and distributing the supplies, we look at the characteristics of these additional supplies.

3.6.1 **Characteristics of the additional supplies**

These additional supplies (V_{dd}+ΔV and V_{ss}-ΔV) only provide a small percentage of the total current of each pseudo-static gate (less than 15% for the FPGA interconnect circuit example). The current is driven from these supplies only during the evaluation and these supplies are not needed during the reset mode. The load that is driven by these supplies is completely internal to the gate. The wires and the output load are driven by the main supply of the chip. Therefore, the load of these supplies is determined during the design and is independent of place and route and wiring. These characteristics could be used to relax the generation and distribution of the additional supplies compared to a regular supply.

3.6.2 **On chip generation of the supplies**

In general, there are three main methods to generate the supplies on the chip: Linear regulators [28], switching regulators [29][30] and charge pumps [31]. A switching regulator is the most efficient method but has the draw back that needs large inductors and capacitors. Since on chip inductors occupy a large area and also have a low quality factor, usually off chip inductors are used. Linear regulator is the easiest to integrate. However, since it only works from a higher voltage, it will not be as

efficient as a switching regulator. In our case, if we only want to bring one supply from outside, it should be higher than all the supplies that are required on the chip. The efficiency is proportional to the ratio of the required supply to the generated supply. The draw back of this linear regulator is that since the off chip supply is higher than nominal supply of the chip, the nominal supply also must be generated by the regulator. However, most of the current of the chip is driven from this supply and therefore, the degradation in efficiency of generation of this supply significantly affects the overall efficiency of the chip.

The third method is using a charge pump to generate the higher and lower voltage supplies. One possible way to further relax the supply generation and distribution is by using distributed charge-pumps that locally transfer charge from the main supply to the additional supply during the idle mode of the gate. In this way, we can further decrease the amount of current that these supplies have to provide. Figure 3.29 shows the block diagram for this. For each gate, we draw some charge on a local capacitor during the evaluation mode and pump charge back to $V_{dd}+\Delta V$ supply during the reset mode. The amount of this cap should be proportional to the load inside the gate, so that the average current can become close to zero. The efficiency of each of these charge pumps are not very high but since they only provide a small amount of overall current, the effect on overall efficiency is small.

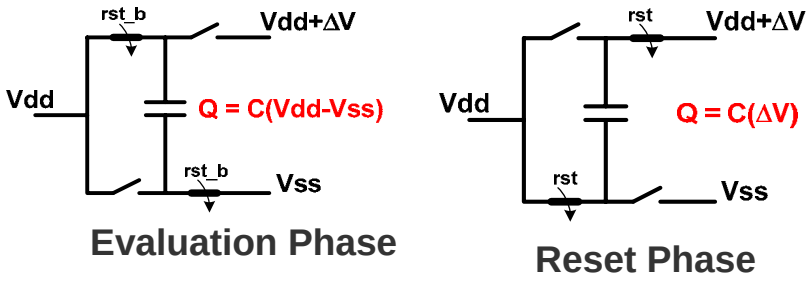

Figure 3.29. Local charge pumps for transferring charge from V_{dd} to $V_{dd}+\Delta V$.

3.7 Summary

In summary, the pseudo-static logic presented in this chapter provides good characteristics compared to other logic families. Compared to dynamic logic, the overhead is much less since there is no clock required. Noise margin is also better since the outputs are static and therefore, even if the output becomes incorrect because of noise, it will be corrected by the gate. Pseudo-static gates do not require monotonic signals and therefore there is no need for dual-rail signaling. Compared to pulse-mode logic, system implementation is much easier mainly because of the static nature of inputs and outputs. Pseudo-static circuits are basically two pulse-mode gates combined with each-other, where at each point of time only one of the paths are active. Since for the pseudo-static gates, outputs are combined to a single static output, the activity factor for the output of the gate and connecting wires is half of the pulse mode outputs and therefore more energy efficient. Compared to static logic, pseudo-static circuits are potentially faster. Even for a two stage pseudo-static circuit, the effective LE is 0.92 times the effective LE of the static circuit. For really low power designs, the static circuit can be more efficient since it does not have the overhead of the reset circuitry, but as the required power increases, pseudo-static circuit becomes more efficient. Another way to look at this is that a pseudo-static circuit is the same as a static circuit split in two paths. Each path is faster since the load in the internal nodes is smaller. It has comparable energy consumption, larger area and more complex circuitry. The most important advantage is that it can adjust the effective threshold of the transistors post fabrication and is, therefore, suitable for adaptive systems and can be optimized for either performance or power based on situation.

Chapter 4

Pseudo-Static Logic for FPGA

A Field Programmable Gate Array (FPGA) is a platform that enables the user to program a chip after fabrication to implement the logic of interest. Compared to Application Specific Integrated Circuit (ASIC), FPGA has much lower time to market, short bug fix cycles and lower non-recurring engineering cost. For custom circuit design, any mistake in the design (discovered after the tape-out) can cause months of delay in manufacturing, while in an FPGA the cost is just re-programming the chip making design cycle short. In many cases, new ideas are first proved on FPGA and then implemented on a custom design chip [32][33] if the product volume is large enough.

Not only are FPGAs used for prototyping, they find their way into many medium volume data parallel applications, from networking to signal processing. Signal processing applications such as neural networks [34][35], automated target recognition [36], software radio [37], wireless transceivers [38] and radar receivers [39] have all been built using FPGAs. In general, circuits that have regular parallel data structures such as multipliers [40][41] and application specific processors [42] have very good performance on FPGAs.

The main limitations of FPGA are the larger area, higher cost and worse performance (lower speed and higher performance/function) than ASICs which are both results of being highly programmable [43]. However, performance is still a limiting factor in using FPGAs since there are orders of magnitude difference between

ASIC and FPGA performance [44]. This chapter describes how pseudo static logic can both increase the performance of the programmable wires in an FPGA and allow the user to tune the speed/power trade-off of the resulting system after fabrication.

To understand why the circuits for the programmable wires are critical, we first describe the architecture of FPGAs showing the factors that limit performance. We then review previous proposed performance improvement methods. Since the current circuits are simple and well optimized static circuits, pseudo-static circuits fit in naturally since they are faster than static circuit and do not require significant architecture changes. The rest of the chapter then discusses the improvement achieved by using pseudo-static gates in FPGA interconnects.

4.1 FPGA Architecture

The most common FPGA architecture used today is an island-style architecture which consists of a large number of programmable logic blocks and each surrounded by programmable routing resources [45]. A simplified block diagram is shown in Figure 4.1.

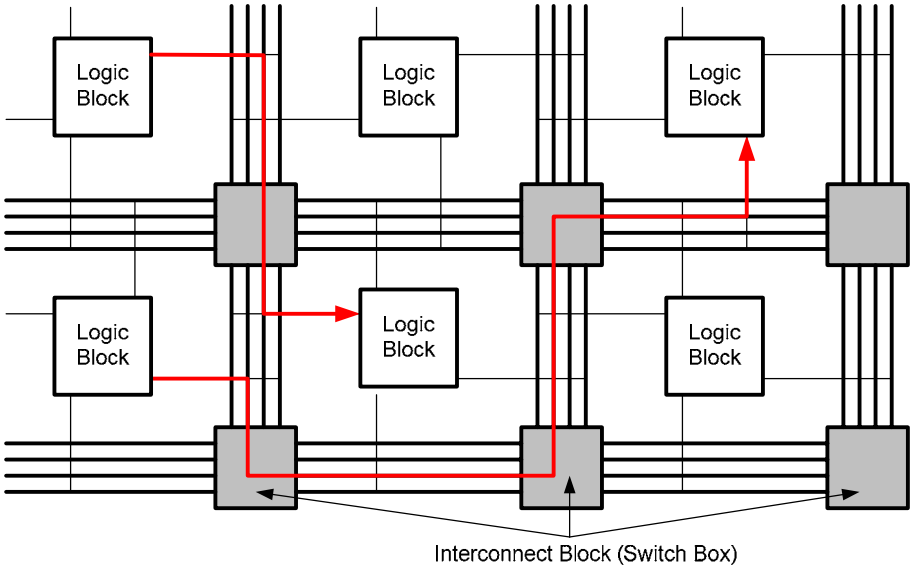

Figure 4.1. Simplified block diagram of FPGAs.

Logic blocks are programmable look up tables that implement the logic of interest and switch blocks are programmable multiplexers that provide the connectivity between logic blocks.

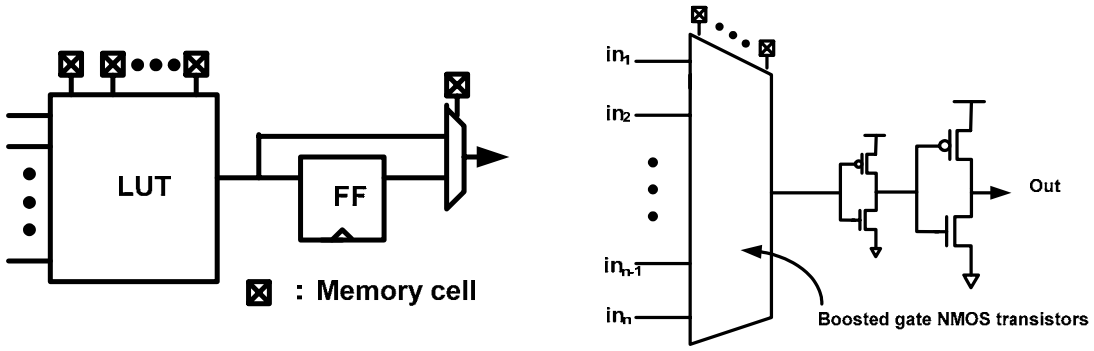

Figure 4.2. shows the block diagram of a logic block and an interconnect block. To provide programmability, as shown in Figure 4.1, logic blocks are connected to each other through a number of high fan-in high fan-out switch blocks to provide wiring flexibility.

A lot of work has been done on optimizing the routing network for an FPGA. Even with this structure, wire delays are usually dominated by switch box delay. A significant portion of the delay in FPGA is due to the interconnect [46] where this interconnect delay, or “wire” delay, is actually dominated by the delay of the switch blocks and not by the RC of the connecting wires. If we look at Figure 4.1, the output signal of one logic block goes to a number of switch blocks before it arrives to the next logic block. To minimize this delay, many different hierarchical interconnect architectures are proposed [46][47][48][49]. Different wire lengths for the interconnect has been used to maximum routing efficiency. For example in state of the art FPGAs [50], interconnect blocks with length of 2, 6 and 24 CLB’s are used.

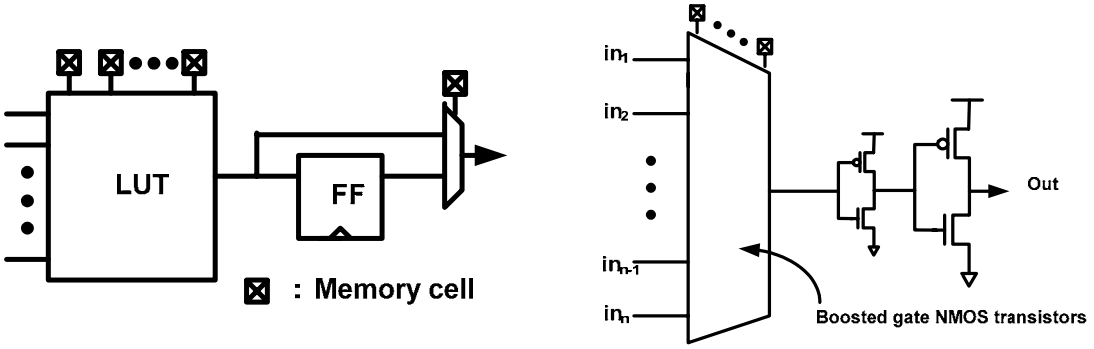

Figure 4.2. Left: logic block architecture consists of a look up table and a Flip Flop followed by a multiplexer, Right: interconnect architecture consists of a multiplexer followed by a buffer.

Since an FPGA has a regular architecture with a few building blocks, we can consider using more advanced design method for these blocks. The few building blocks can be custom designed and the whole chip can be generated by automatic tools. The other point is that redundancy in the FPGA hurts in two ways. First, they decrease the performance of the chip. Second, the unused blocks on the chip have static leakage energy and should be optimized considering that. Besides that, the required energy-performance from an FPGA depends on the specific application it is used for. For some applications, high performance is required while for some other applications, the challenging part is the power consumption. Since there are many used blocks in the architecture, leakage is an issue for FPGAs that needs to be carefully considered.

4.2 Previous work and design caveats

In order to narrow the gap between FPGA and ASIC, many alternative architectures have been proposed. As we will see, none of these removes the need for better wire switching circuits. In [51], regenerative repeaters are used to enhance the performance of interconnects. In this case, both input and output of a regenerative repeater are connected to the signal wire. After a change is detected in the wire, a positive loop in the regenerative repeater makes the transition much faster. However this technique works with pulses with arbitrary arrival times and it suffers from meta-

stability problems. In order to increase power efficiency, architectures such as dual supply [52] and low swing interconnect [47] have been considered as well. Different leakage reduction techniques such as using a mix of low-V_{th} and high-V_{th}, adjusting body bias and negatively biasing the gate terminals of off state multiplexer transistors have also been used [53][54].

High level changes to the architecture have also been considered. Asynchronous FPGA [55] tries to achieve higher performance by ignoring the clock and taking advantage of pipelining. Wave pipelining has also been used to take advantage of having similar blocks along the signal path [56]. In 3D FPGA, the overhead of interconnect is reduced by having different layers and therefore shorter interconnects [57][58].

While these solutions help to increase the overall efficiency, they still require programmable wires and hence switch blocks. As a result, a faster switch block is essential to improve the performance of all array architectures and will be considered next.

4.3 Interconnect architectures

Unfortunately, as Figure 4.2 shows, the architecture of a static interconnect is simple and hard to improve -- the multiplexer is implemented using boosted gate nMOS transistors followed by a buffer [47].

In this section, we explore different interconnect architectures and compare them. Since pulsed gates are usually the fastest circuit topologies, we include them in our interconnect architectures even though they have high power and system overheads.

4.3.1 **Static interconnect**

Figure 4.3 shows different ways of implementing a 2:1 multiplexer.

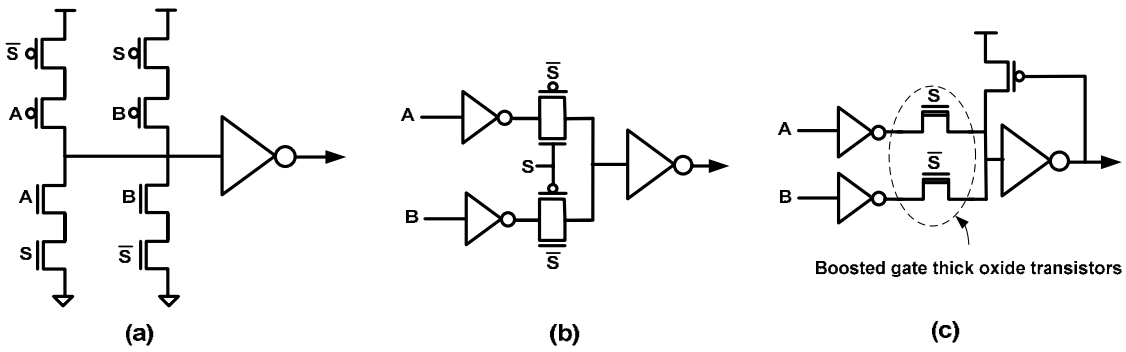

Figure 4.3. Different ways of implementing a 2:1 static multiplexer.

In order to both save area and decrease the parasitic delay, recent FPGA's use boosted gate nMOS transistors to implement the multiplexer [53] and pMOS transistors are eliminated. In order to be able to pass both zeros and ones with reasonable delay, the gate voltage is boosted for the pass transistors (Figure 4.3(c)) [59]. In an FPGA, using a high voltage to control the multiplexer is ok, since these controls are connected to configuration bits and are constant after initial configuration. Therefore, having boosted gate does not increase the operation power of the circuits. Also, due to device reliability issues, thick oxide transistors must be used for the pass transistors. The capacitance of these devices is larger than normal devices (~1.5X for the same width), but since the pMOS is eliminated, the overall parasitic capacitance is still less than before (also the effective resistance is less since the gate is boosted).

A 16:1 multiplexer is implemented as two cascaded stages of 4:1 muxes consisting of boosted gate nMOS transistors as shown in Figure 4.4.

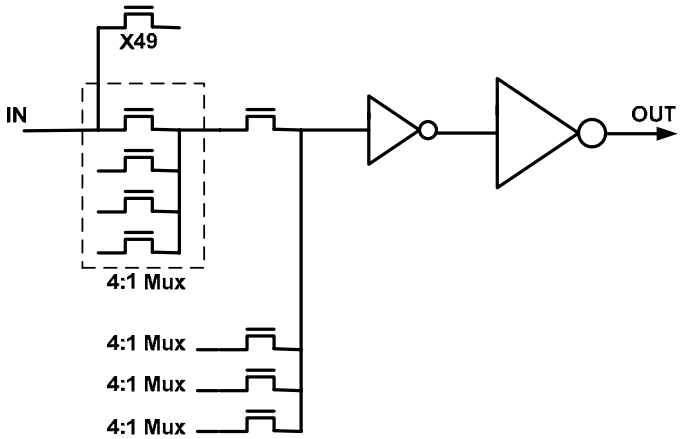

Figure 4.4. 16:1 static interconnect multiplexer.

We use a circuit optimization tool (described in Appendix A) to compute the energy delay optimal trade-off curve for this circuit. For CMOS circuits, delay can only be consistently calculated if the input change drives the gate of a transistor, and the output drives the gate of another transistor. The collection of transistors that connect the input transistor to Vdd and Gnd and the output is generally called a channel connected components (CCC). Since we require CCC components and the input of the mux goes to the source of the transistors, we need to consider the optimization path from the previous mux which is shown in Figure 4.5.

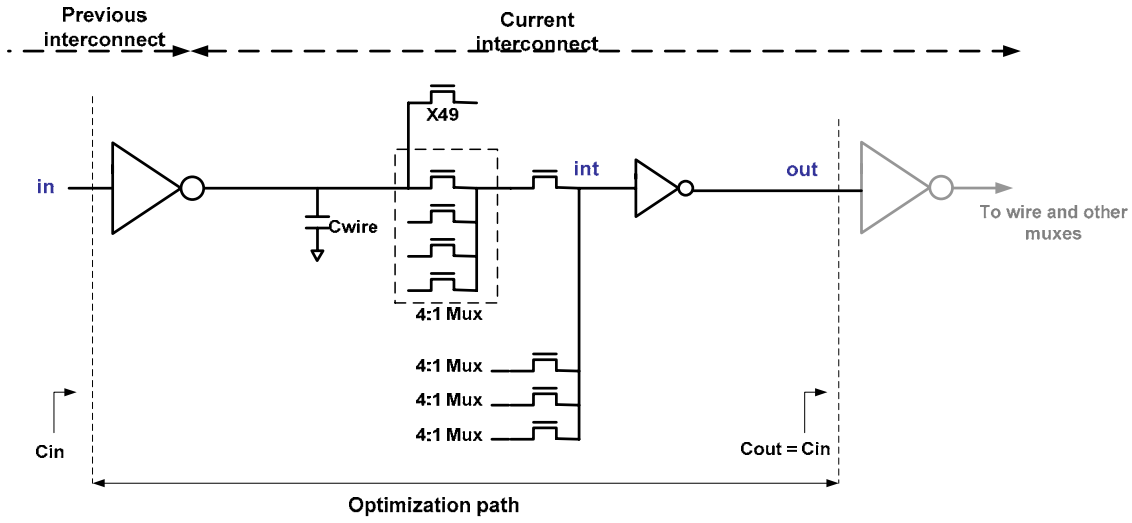

Figure 4.5. Optimization circuit for static interconnect. The optimization path is considered from the last stage of the previous mux to have source connected devices.

The objective in optimization is to minimize the rise and fall time of the output given a constraint on the total energy dissipated for each transition. So our optimization program is:

minimize POMAX;

max(out.Trise, out.Tfall) < POMAX;

E_total < Specified_Energy

4.3.2 **Pulse-mode interconnect**

Since only one edge transition is critical for pulse-mode interconnect, we do not need the strong pMOS transistors that caused significant parasitic delay in static case Figure 4.3(a). Therefore, we can consider different circuit topologies for the multiplexer in a pulse-mode interconnect. After exploring many options, described in more detail in the next chapter, we discovered that the boosted gate nMOS multiplexer remained the best alternative, so the only real change was the design of the buffer. The best buffer was a delayed buffer which is a type of pulse-mode gates that have minimum overhead in the reset circuitry [64]. A delayed reset circuit resets its output only after its input are have been reset. Figure 4.6 shows the pulse-mode interconnect architecture we use. Adjusting the threshold for this circuit is achieved by using (V_{dd1} = V_{dd} + ΔV and V_{ss1}= V_{ss}+ΔV) or (V_{dd1} = V_{dd} + ΔV and V_{ss1} = V_{ss}), supply configurations for one and two additional supplies respectively.

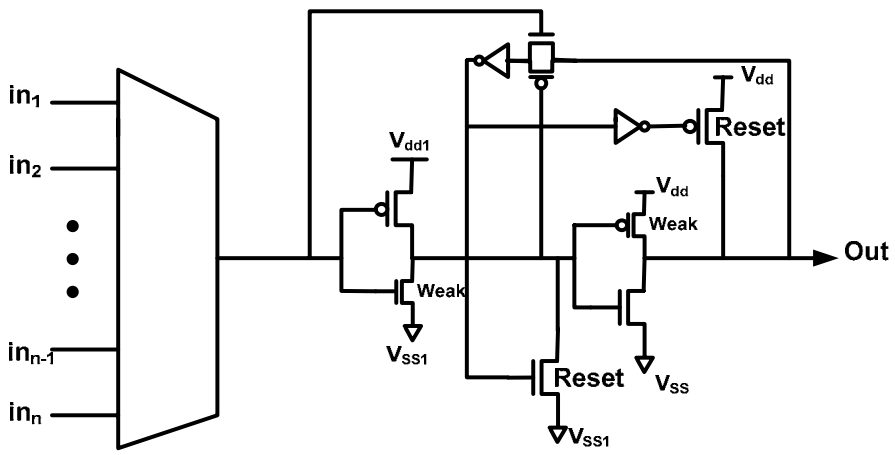

Figure 4.6. Pulse-mode interconnect architecture.

The optimization path is shown in Figure 4.7. The gray parts are the parts in an interconnect block which are not included in the optimization path (since optimization is started from the previous interconnect stage). The reason for the blue line, which are not in the real interconnect, is to enable one to estimate the delay of the block even though the inputs are source/drains. In our optimization path, output of the gate is not included since the output of the previous gate is included to drive the mux inputs. Therefore, in order to model this part, we include another input (inR_2) instead of the output. In order to model the loading of the gray circuit connected to the res_2 point, we connect this node to inR_1. Therefore, the functionality of this circuit is not correct but delay equations for the interconnect are correct. We constrained the weak transistors inside the gate to have a strength 1/5 of the strong transistors. Also, for reset, we only want to make sure that reset is done in a reasonable time and not necessarily the fastest time. Therefore, we included some relaxed constraints on falling of the output and rising and falling of the reset nodes. The optimization constraints are:

Minimize POMAX;

Max(out.Trise) < POMAX;

The above constraint is to minimize the critical delay of the circuit which is the delay from rising transition of ‘in’ node to rising transition of ‘out’ node in Figure 4.7.

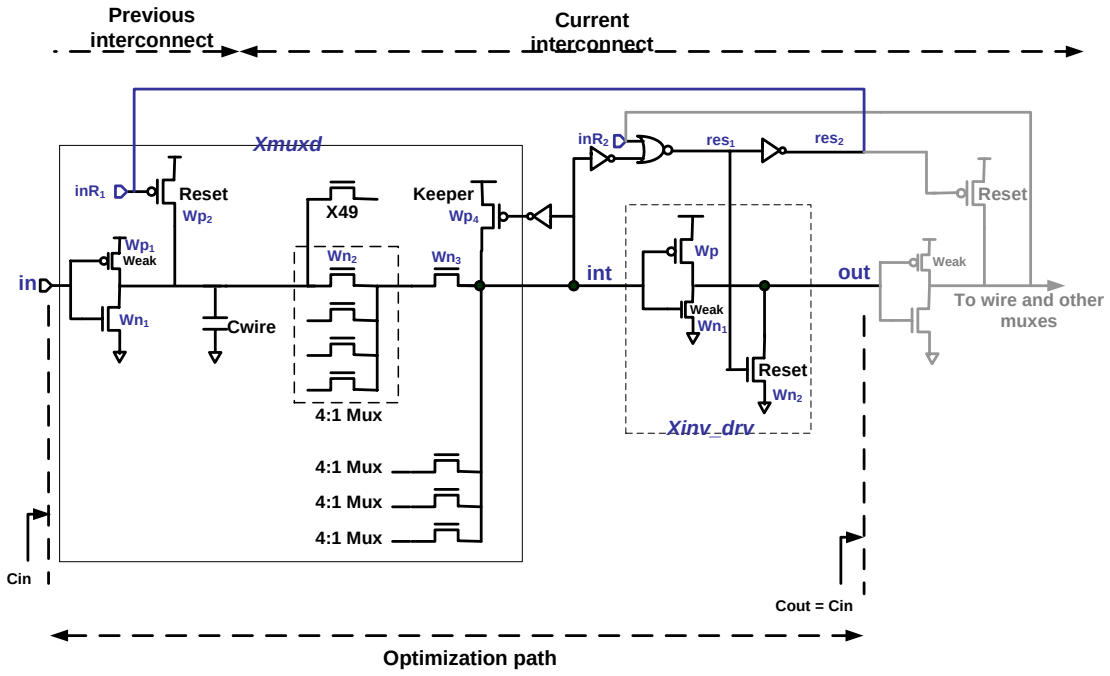

Figure 4.7. Optimization path for pulse mode interconnect

Since in a pulse-mode circuit only one edge is critical, only that edge is optimized for minimum delay. However, the delay of the other edge and the delay of the reset circuitry are also important since they need to be bounded and less than a certain amount for the circuit to work properly. The following equations provide the constraints for the reset signals and the falling edge of the output.

$$out.Tfall < 6 \times out.Trise \tag{4.1}$$

$$res_1.Tfall < 6 \times out.Trise \tag{4.2}$$

$$res_1.Trise < 6 \times out.Trise \tag{4.3}$$

$$res_2.Trise < 7 \times out.Trise \tag{4.4}$$

$$res_2.Tfall < 7 \times out.Trise \tag{4.5}$$

$$out.Tfall < res_2.Tfall \tag{4.6}$$

Constraints on the strengths of the weak transistors:

$$\frac{1}{5} \times 2 \times Xmuxd.Wn_1 < Xmuxd.W_{p1} \tag{4.7}$$

$$\left(\frac{1}{2} - \frac{1}{5}\right) \times 2 \times Xmuxd.Wn_1 < Xmuxd.Wp_2 \tag{4.8}$$

$$\frac{1}{4} \times Xmuxd.Wn_3 < Xmuxd.Wp_4 \tag{4.9}$$

$$\frac{1}{5} \times \frac{1}{2} \times Xinv\_drv.Wp < Xinv\_drv.Wn_1 \tag{4.10}$$

$$\left(\frac{1}{2} - \frac{1}{5}\right) \times \frac{1}{2} \times Xinv\_drv.Wp < Xinv\_drv.Wn_2 \tag{4.11}$$

Equations (4.7) and (4.10) make sure that the week path in the mux has 1/5 of the strength of the strong path. Equations (4.8) and (4.11) make sure that reset has ½ strength of the strong path. Since reset is only done when input is reset, the week path helps the reset path and therefore reset can be weaker. Equation (4.9) is to set the strength of the keeper.

4.3.3 **Pseudo-static interconnect**

The study of pulse-mode interconnect showed that the best architecture is to use boosted gate nMOS transistors in the multiplexer and change the buffer to a delayed reset buffer. Therefore, for the pseudo-static circuit, we also implement the multiplexer using boosted gate nMOS transistors and change the buffer to a pseudo-static buffer. Figure 4.8 shows the architecture of the pseudo-static interconnect.

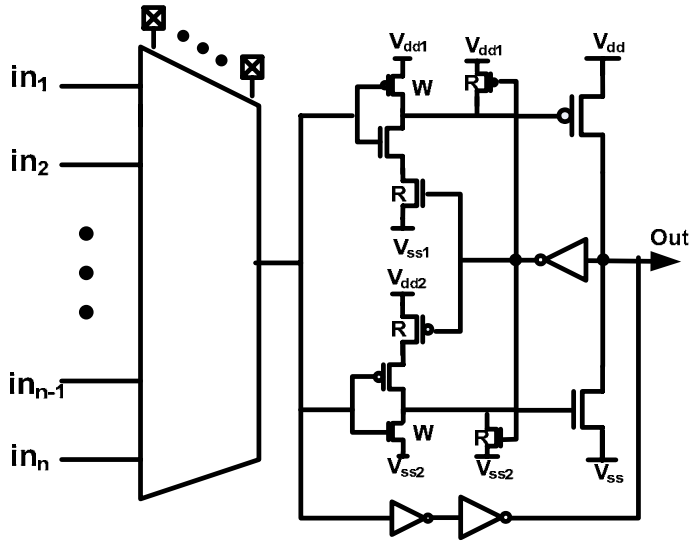

Figure 4.8. Pseudo-static interconnect architecture with the different supplies.

Adjusting the threshold for this circuit is achieved by using ($V_{dd1} = V_{dd}$, $V_{ss1} = V_{ss} - \Delta V$, $V_{dd2} = V_{dd} + \Delta V$ and $V_{ss2} = V_{ss}$) or ($V_{dd1} = V_{dd} - \Delta V$, $V_{ss1} = V_{ss} - \Delta V$, $V_{dd2} = V_{dd} + \Delta V$ and $V_{ss2} = V_{ss} + \Delta V$) supply configurations for two and four additional supplies respectively.

The optimization path for the pseudo-static interconnect is shown in Figure 4.9. To optimize this circuit, like the other two architectures, the optimization path is considered from the driving stage of the previous interconnect to be able to write delay equations for the pass transistors. In this case, we have three inputs, since we have three paths in the interconnect (paths for pull-up and pull-down and the weak path). Input and output capacitors should be equal. For the optimization constraint, we want to minimize delay of falling edge of outf, rising edge of outr and both edges of outw.

Also, in order to reduce the overhead of reset and glitch expansion overhead, we added two constraints on the reset path delay. outr_reset and outf_reset constraints are for this purpose. The other two constraints are for avoiding the conflict when one path resets and the other path is enabled.

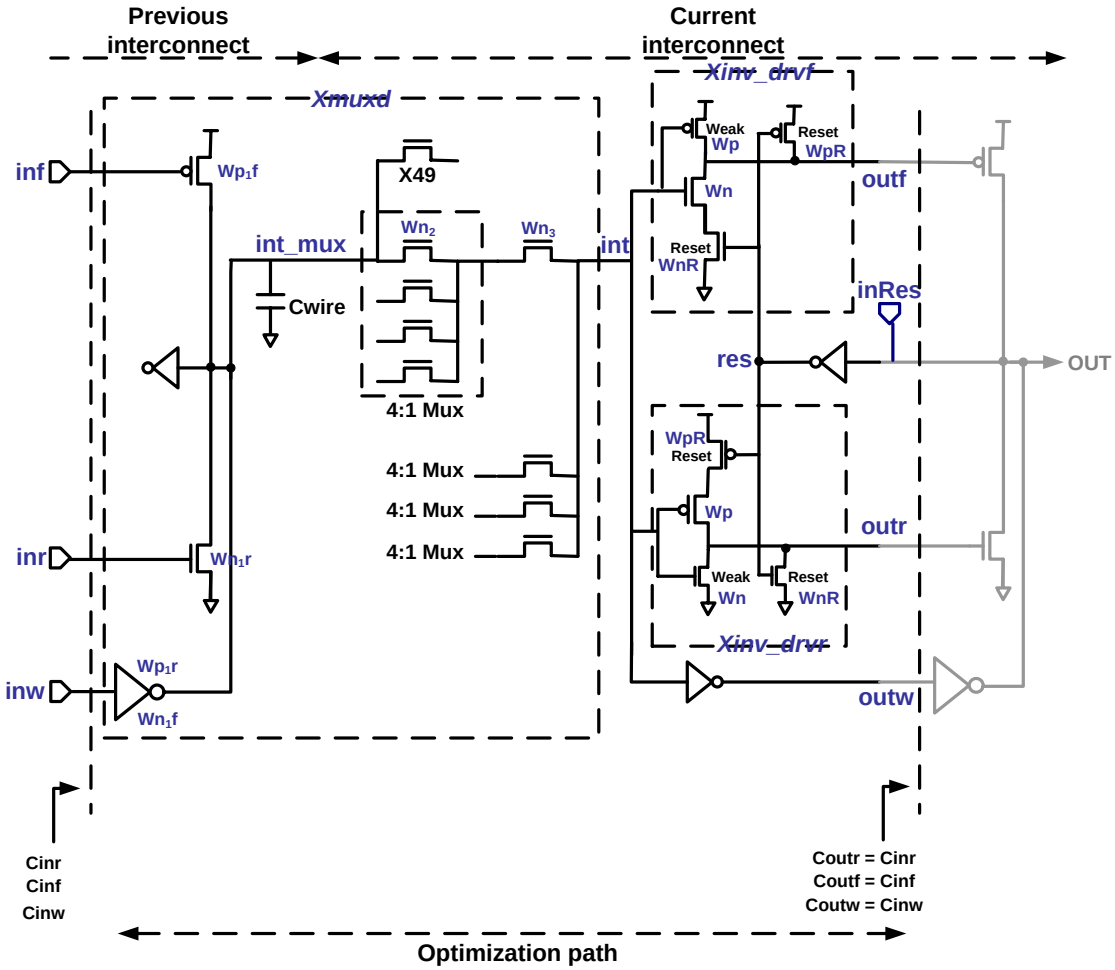

Figure 4.9. Optimization path for the pseudo-static interconnect: It starts from driving stage of the previous interconnect to the input of the driving stage of the current interconnect. Input and output capacitors are therefore equal.

Minimize POMAX;

Max(outf.Tfall,outr.Trise, outw.Trise, outw.Tfall, outf_reset, outr_reset) < POMAX;

outf.Tfall and outr.Trise are the main paths. outw.Trise and outw.Tfall are for the weak path and have to be optimized in order to avoid conflicts between weak path and strong path. outr_reset and outf_reset are for glitch overhead and are provided in the next equations.

$$\frac{1}{5} \times Xmuxd.Wp_{1f} < Xmuxd.Wp_{1r} \tag{4.12}$$

$$\frac{1}{5} \times Xmuxd.Wn_{1r} < Xmuxd.Wn_{1f} \tag{4.13}$$

Equations (4.12) and (4.13) are to set the sizes of the weak driver to have 1/5 strength of the main drivers.

$$delay(res \uparrow \rightarrow outr \downarrow) < delay(res \uparrow \rightarrow outf \downarrow) \tag{4.14}$$

$$delay(res \downarrow \rightarrow outf \uparrow) < delay(res \downarrow \rightarrow outr \uparrow) \tag{4.15}$$

Equations (4.14) and (4.15) are to make sure there is no conflict when one path is enabled and the other path is disabled.

$$delay(res \downarrow \rightarrow outr \uparrow) + delay(in\,\mathrm{Re}\,s \uparrow \rightarrow res \downarrow) + delay(inr \uparrow \rightarrow \mathrm{int}\_mux \downarrow) < outr\_reset \tag{4.16}$$

$$delay(res \downarrow \rightarrow outf \uparrow) + delay(in\,\mathrm{Re}\,s \downarrow \rightarrow res \uparrow) + delay(\mathrm{inf} \downarrow \rightarrow \mathrm{int}\_mux \uparrow) < outf\_reset \tag{4.17}$$

Equations (4.16) and (4.17) are to minimize reset path delay and the glitch overhead.

4.3.4 **Comparison of different interconnect architectures**

A convex optimization framework [60] was used to find the globally optimum transistor sizes and the supply voltage that achieve the best performance for a given total energy consumption for each architecture. This optimization is done for different energy points to obtain energy-performance tradeoff curves and the results are verified

using SPICE simulations. In the optimization for pseudo-static and pulse-mode architectures, zero skew between the supplies has been considered. Therefore the comparison between the blocks in this optimization is the comparison between the architectures without considering the effect of tunability.

Figure 4.10 shows simulated energy-performance optimization results for the three different architectures. The blue dot shows the energy-performance point for a static interconnect circuit used in a state of the art FPGA in the same technology. As the figure shows, at the same energy level as a static interconnect, a pseudo-static interconnect achieves 10% higher performance. For this same energy level, pulse-mode interconnect is as efficient as the traditional static design because the activity factor of the pulse-mode interconnect is roughly double that of the static design. However, as the desired performance (and hence energy/operation) increases, the inherent speed advantages of the pulse-mode interconnect make it more efficient than the static design. Another way to look at the results is to observe that pulse mode interconnect achieves roughly 30% better performance than the traditional static interconnect at twice energy consumption. This is perhaps a more meaningful comparison considering the twice activity factor of the pulse mode logic compared to static logic. The next section describes the results from a test chip we built to compare these circuits, including the effects of skewing supplies to change the effective threshold voltage.

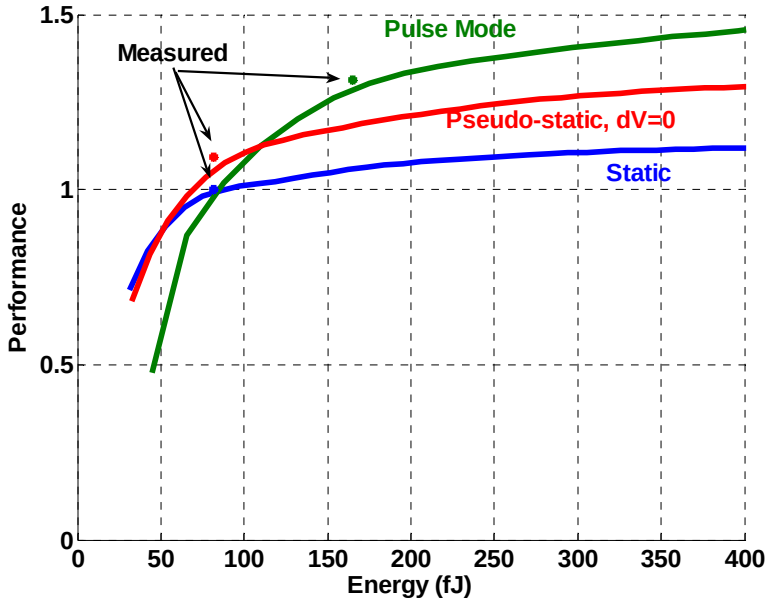

Figure 4.10. Energy-performance trade off curve for different architectures based on the sizes and supply voltage provided by the optimization using a convex optimization tool.

4.4 Measurement Results

The static, pseudo-static and pulse-mode circuits were prototyped on a 90nm CMOS test-chip. The prototype for static circuits is chosen on the knee of the optimization curve (at 80FJ energy). For the pseudo-static circuit, we chose a prototype with the same energy consumption. The pulse mode circuit was optimized for twice energy consumption. Interconnect blocks are connected to 200µm of wire and to the input of 45 other interconnects to realistically model the interconnect blocks in a state of the art FPGA. Two interconnect blocks with adjacent output wires are placed next to the main interconnect blocks (with twice minimum wire spacing of the technology, which is same as state of the art FPGAs [61]) to measure the effect of the worst case coupling on the output.

4.4.1 **Test Chip Design**

Figure 4.11 shows the test configuration for each of the three architectures. For each architecture, we cascaded four interconnect blocks and monitored input and output of the first interconnect stage and output of the third interconnect stage. All the aggressors are connected to a separate input so that the aggressors can turn on when them main input is off. The 'NAND' stage before the interconnect input is to shape the input coming from outside and also enables inputting a glitch with variable width to the interconnect when the two inputs of 'NAND' are the same frequency square waves with different phases. The nodes that are marked as sampled are used to monitor the inputs and outputs' waveforms and measure the delay by sub-sampling.

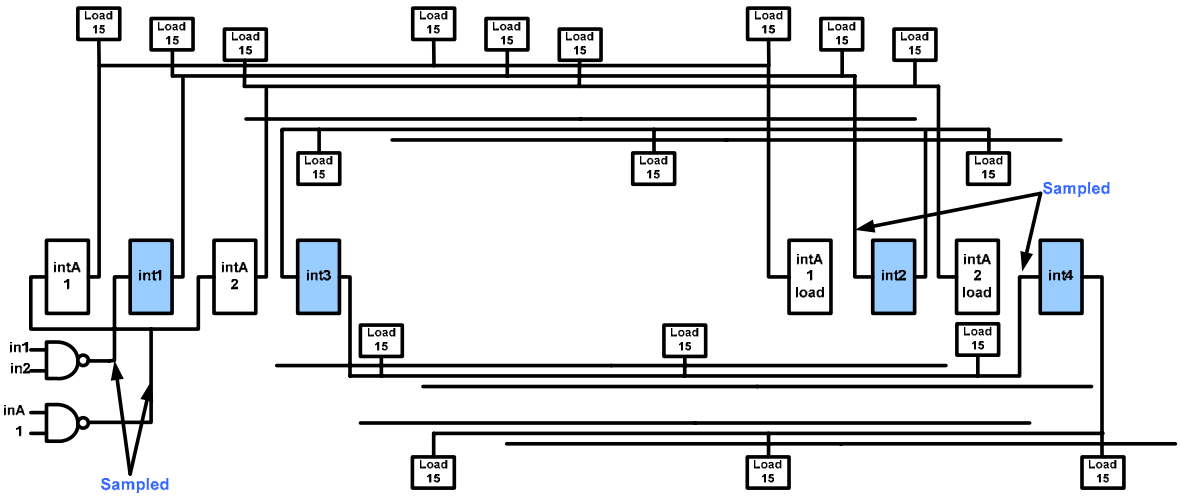

Figure 4.11.Test configuration for each architecture. Four interconnect stages are cascaded and two aggressors are provided to have worst coupling scenario.

Figure 4.12 shows the sampler architecture based on the sampler presented in [62].

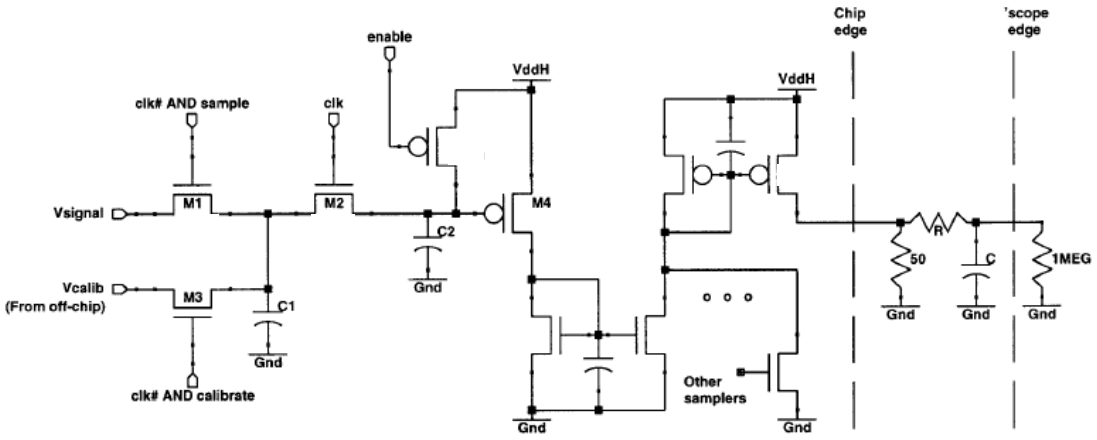

Figure 4.12. Sampler architecture as used in [62].

The sample and hold transistors are very small. The sampling capacitor is just the parasitic capacitor of these transistors. Since we are sampling digital signals with high swing, small sampling capacitor values are tolerable (kT/C noise is in the sub-mV range compared to 1V swing of the signals). In order to avoid variation in the small transistors, each of the samplers is calibrated. The output of the samplers are then converted to analog current and sent to the outside of the chip. Since we are sampling full rail signals, either the voltage of the samplers should be higher than the nominal V_{dd} or the input should be divided down. Since we have a higher than nominal voltage for the boosted transistors, we used the same voltage for the samplers as well. Thus VddH goes to the 1.5V supply that controls the passgates. The 3 samplers of interconnect inputs and the 6 samplers of the outputs are multiplexed so that we only have two analog outputs for the samplers. Each sampler has an enable bit that is programmed using scan chain.

In this test-chip, all the required supplies for the pseudo-static logic are supplied externally. For supply distribution, we used the same supply grid on M6 and M7 layers and divided the grid between different supplies based in the ratio of the required currents (Figure 4.13). Figure 4.16 shows a die micrograph of our testchip.

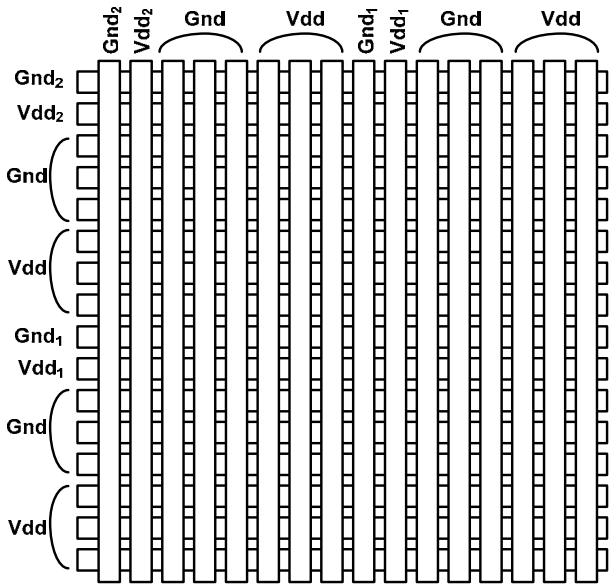

Figure 4.13. Supply grid used in the test chip for routing different supplies of the pseudo-static circuit.

4.4.2 **Energy-Delay Results**

The dots on Figure 4.10 show the measured energy/delay points for each circuit which closely match the simulation results. Figure 4.14 shows the measured gate delay vs. ΔV for 2-additional-supply and 4-additional-supply pseudo-static interconnects and 1-additional-supply and 2-additional-supply pulse-mode circuits, respectively. It is seen from this figure that by skewing the supplies a wide range in energy-performance can be achieved. Also, the same performance can be achieved by using a fewer number of supplies at a slightly higher skew.

The post-fabricated energy/delay trade-off curves for these circuits are obtained by changing ΔV and V_{dd} for pseudo-static and pulse-mode circuits and by changing V_{dd} for the static circuit, are shown in Figure 4.15. Each point in this figure is a measurement of the circuit at a particular Vdd and ΔV. These measurements were taken with an input frequency of 180MHz.

Comparing pseudo-static and static interconnect we can see that by tuning both V_{dd} and ΔV, pseudo-static circuit achieves 20% higher performance at no energy cost

compared to static circuits. If we compare the points at the same performance, it consumes 65% the energy of the static circuits. The tuning also allows it to cover a 2X wider range in performance over the same energy interval than a simple static circuit. We also notice that except at very high power, the pseudo-static gate matches the performance of a pulse-mode running at the same energy.

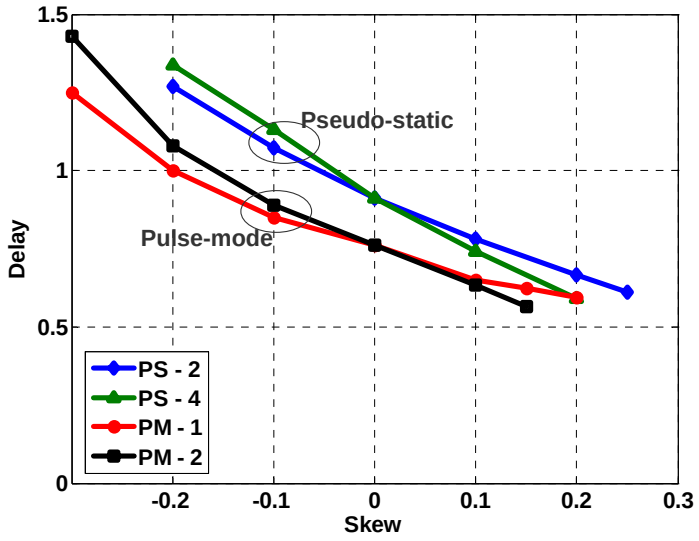

Figure 4.14. Delay vs. skew (ΔV) for pulse-mode and pseudo-static architectures. PS-2 shows pseudo-static with two supplies, PS-4 shows pseudo-static with 4 supplies, PM-1 and PM-2 show pulse-mode with one and two supplies respectively.

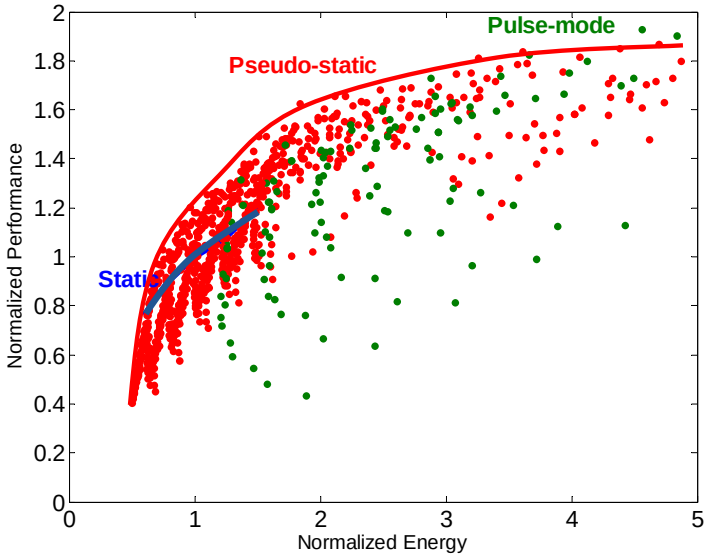

Figure 4.15. Overall energy-performance trade off curve obtained by changing both supply and skew for pseudo-static circuit and supply voltage for static circuit.

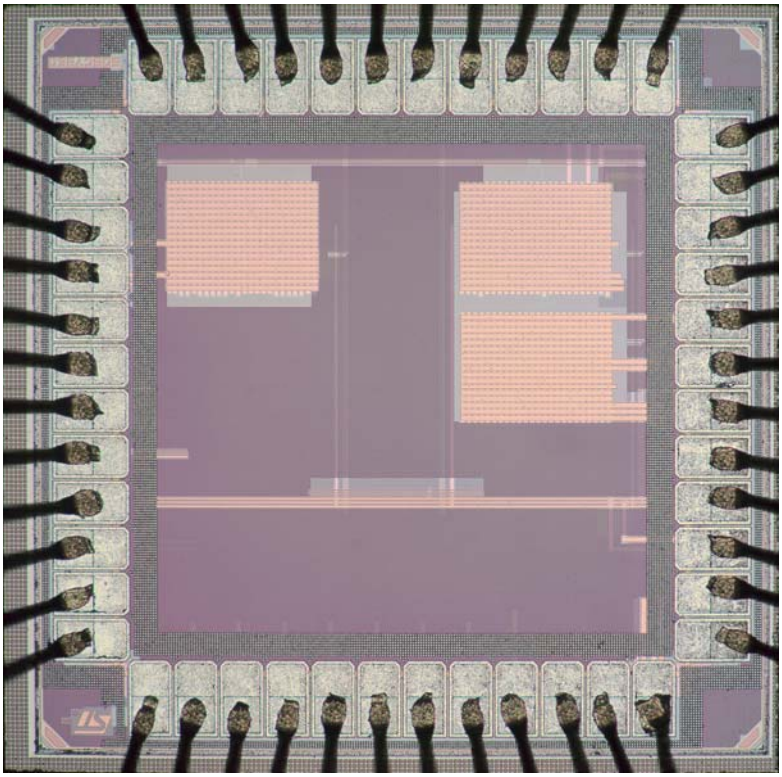

Figure 4.16. Die micrograph of the testchip.

4.4.3 Coupling noise

We also used the testchip to explore the effects of noise on the highly skewed gates in pseudo-static logic. Figure 4.17 shows the waveforms when the aggressors are on and the main input is off to show the effect of coupling. As it can be seen from this figure, the coupling noise is very small because the input capacitance is dominated by grounded capacitance of the multiplexer structure.

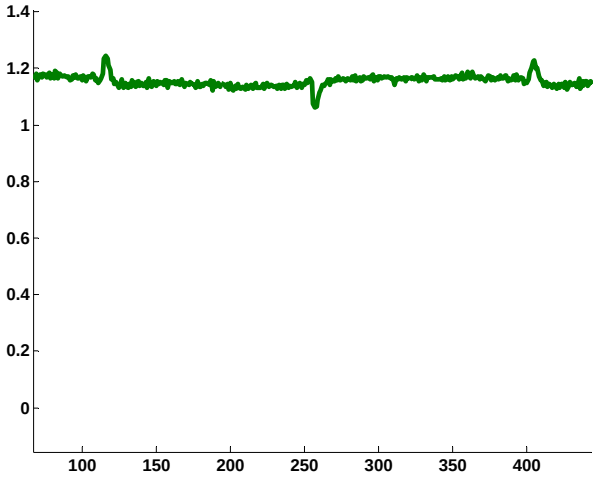

Figure 4.17. Coupling on the interconnect output when aggressors are on.

4.4.4 Glitch propagation

The NAND gates in the first stage of the interconnect circuits were very useful since they allowed us to create narrow pulses as input to the interconnect. In order to measure the glitch overhead and glitch propagation, we generated a narrow pulse input and measured the outputs of different interconnects in the cascade. We did this experiment for different input pulse widths. Figure 4.18 shows the measured waveforms for input and outputs of a first stage and third stage pseudo-static interconnect in a cascade with 200mV skew of the supplies. If the input pulse is too narrow (as in the case of Figure 4.18(a)), the pulse is filtered and there is no transition at the output of the third interconnect. As input pulse width increases, we see some glitch overhead at the output of the first stage and third stage (as in the case of Figure

4.18(b) and (c)). When the input pulse is wide (as in the case of Figure 4.18(d)), we do not see any overhead and all pulse width are equal. This confirms that glitch overhead only occurs for certain pulse widths and also it only happens at the first stage and the consequent stages just propagate the expanded pulse.

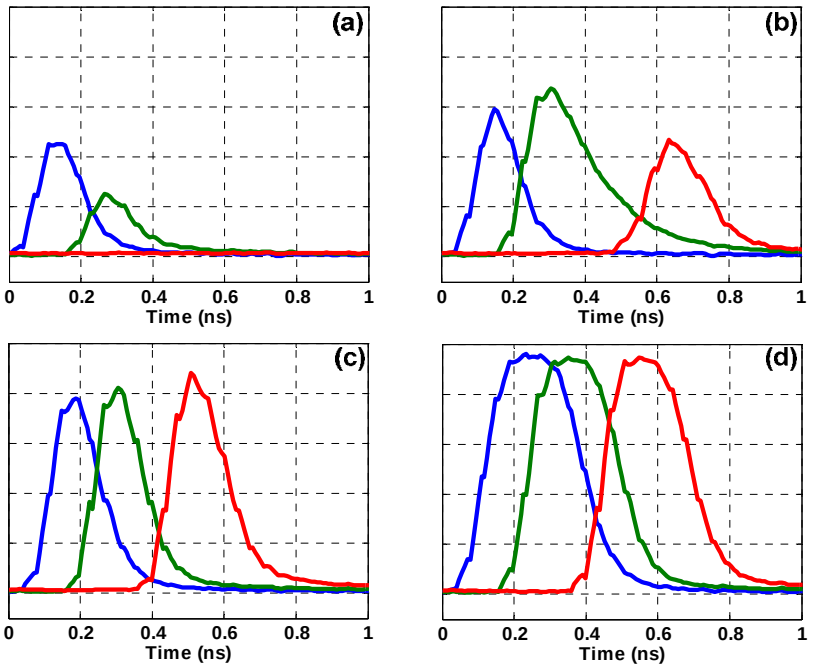

Figure 4.18. Input and output waveforms for pseudo-static interconnect with 200mV skew with different input pulse widths. Blue waveforms are input signals, green waveforms are outputs of the first stage in a cascade and red waveforms are outputs of the third stage in a cascade.

4.4.5 **Cost of generating additional supplies**

In our test chip, we brought the supplies directly from outside the chip. However, we evaluated the cost of generating the supplies on the chip as explained in Chapter 3. Our simulated show that the area overhead of the required capacitor (which is proportional to the gate's internal load) is less than 5% of the overall area of the gate.

In summary, in this chapter we showed that pseudo-static circuits are good candidates for FPGA interconnect circuits. The regularity of the architecture and high fanout of the gates make use of pseudo-static gates easier. As we showed, noise coupling is not an issue for these gates since the ratio of the wire capacitor to the load capacitor is small. Since interconnects are single input single output gates, overhead of glitches only exist for the first gate in a cascade and the rest of the gates will have full performance benefit. In terms of area, overhead is small since an interconnect gate consists of array of memory cells (that program the multiplexer) and boosted gate nMOS transistor for the multiplexer and the buffer. We are only changing the buffer architecture and, therefore, the area overhead is less than 20%. On the other hand, using pseudo-static circuits provide much wider tuning range in energy performance space compared to conventional static circuits.

In the next chapter, we consider changing the architecture of FPGA to a fully pulse mode architecture.

Chapter 5

A Fully Pulse-Mode FPGA

In this chapter, we investigate the possibility of having a fully pulse mode FPGA. As explained in Chapter 3, one can construct gates that create pulses when the gate output transitions. While this is in general inefficient, since FPGAs are regular architectures with only a few building blocks, the changes to the architecture may have a tolerable overhead. In this discussion, we first look at optimizing the pulse-mode interconnect block. In order to do this, we briefly overview different pulse mode gates and then talk about optimizing the interconnect architecture. Next, we look at different look up table architectures that are compatible with pulse-mode interconnect, calculate the overall performance and discuss the system level problems and how they could be addressed.

5.1 Pulse mode circuits review

In a pulse mode gate, there are three different stages of operation: evaluation, reset and standby. In the standby mode output stays at a default value. Every time an input pulse comes, the gate evaluates and the output may change based on the logic. The gate resets after some delay after the output changes and goes back to the default state. Different type of pulse-mode gates exist based on the way resetting of the gate is done. These are explained in detail in [64] and are briefly mentioned here.

There are three paths that have to be considered for a pulse mode circuit: forward path, reset path and keeper path. The keeper is usually a weak version of the logic to keep the intermediate node in standby mode. Therefore, if the evaluation path is

implemented using a strong pull down path, keeper would be a weak pull up path and vice versa. In cases that the logic is too complicated, sometime a half-latch with pMOS pull up transistor is used. The forward path is either pull-up or pull-down depending on which edge we are optimizing for. The reset path is the one that is mainly different between different architectures.

5.1.1 **Self resetting logic (SRCMOS)**

Self resetting logic uses the output to reset the gate [65],[66]. A block diagram is shown in Figure 5.1. The output pulse width is determined by the reset path delay. After resetting the output, the new level of the output is again propagated through the reset path, disables the reset transistors and the gate is ready for the next input transition.

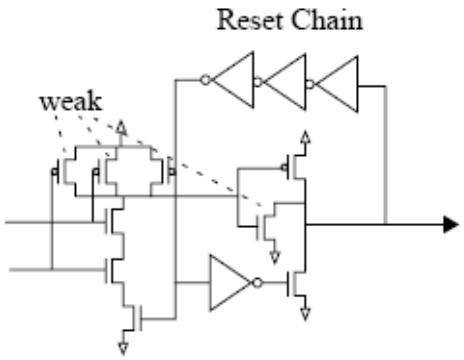

Figure 5.1. SRCMOS gate block diagram. Foot transitor at the bottom of the nMOS stack in the first stage disconnects input from the gate during reset.

The main constraints are:

1. A foot transistor is generally inserted in the bottom of the input stage to disconnect input from the logic during reset to avoid any contentions. In cases that it is guaranteed that input pulse width is smaller than reset path delay, this foot transistor can be eliminated to increase the speed.

2. Input pulse width should be less than twice the reset path delay to avoid multiple evaluations for the same inputs.

5.1.2 **Delayed reset logic (DRCMOS)**

In delayed reset logic, the second edge of the input pulse is used to reset the circuit. Therefore, there is no need for the foot transistor as in SRCMOS gates and the gate can potentially be faster. However, if multiple inputs exist, this may not be optimum since static version of the logic should be implemented in the reset path. Figure 5.2 shows the block diagram for the delayed reset architecture (which cheats and only waits for one of its inputs to reset). Using this logic in a gate, it is guaranteed that the output pulse width is longer than the input pulse width. Therefore, in cascade of these gates, the pulse width keeps growing. Usually the chains of DRCMOS gate are broken by an SRCMOS gate to adjust the pulse width.

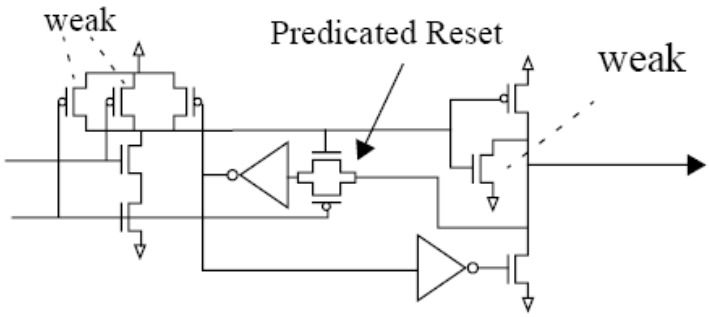

Figure 5.2. DRCMOS block diagram.

5.2 Pulse-mode interconnect

Since only one edge transition is critical for pulse-mode interconnect, we can consider different multiplexer architectures both with and without boosted gate transistors. For the keeper of the multiplexer, it is better to use a half latch since it is a simpler circuit. Since the interconnect mux has only one active input at each time, we do not need to worry about different input pulses overlapping for enough amount of time.

5.2.1 SRCMOS pulse mode interconnect

Since the multiplexer is a multi-input gate, with normal nMOS transistors that only pass low levels strongly, using DRCMOS is not efficient since we do not know which input is going to be active beforehand and therefore, we will have to implement a static mux in the reset circuitry which is too complicated. For a self-reset architecture, the output pulse width is determined by the reset path delay. The output of one interconnect block is the input of the next one and since the blocks are the same, for a self reset circuit, nominally the input and output pulse widths are the same. Because of the process variation, the actual pulse width may be different than the nominal pulse width and this can cause contention current during the reset. Therefore, we need to use foot transistors to disable the input path during reset and avoid any contention.

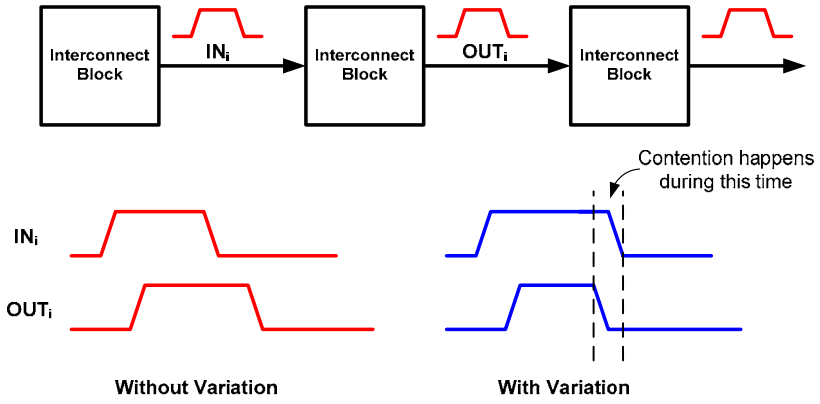

Figure 5.3. Input and output pulse width for cascaded self-reset interconnect blocks.

Since the nominal value of input and output pulses are the same, the input pulse will be finished before the reset is finished and therefore multiple evaluations does not happen.

The simplest form of the pulse-mode interconnect is implementing the multiplexer using nMOS transistors. Figure 5.4 shows a two stage implementation of the multiplexer. Assuming we have 16 selects for the multiplexer, the first stage of the gate just implements the multiplexer as 4 input multiplexers and the second stage combines four 4-input multiplexers. In the multiplexer architecture, from a delay

perspective, selects of the multiplexer should be placed in the bottom of the stack since they do not change. However because of the charge-sharing problem, we should put them on the top (especially since the gate has high fan-out). If we calculate delay based on logical effort for this gate:

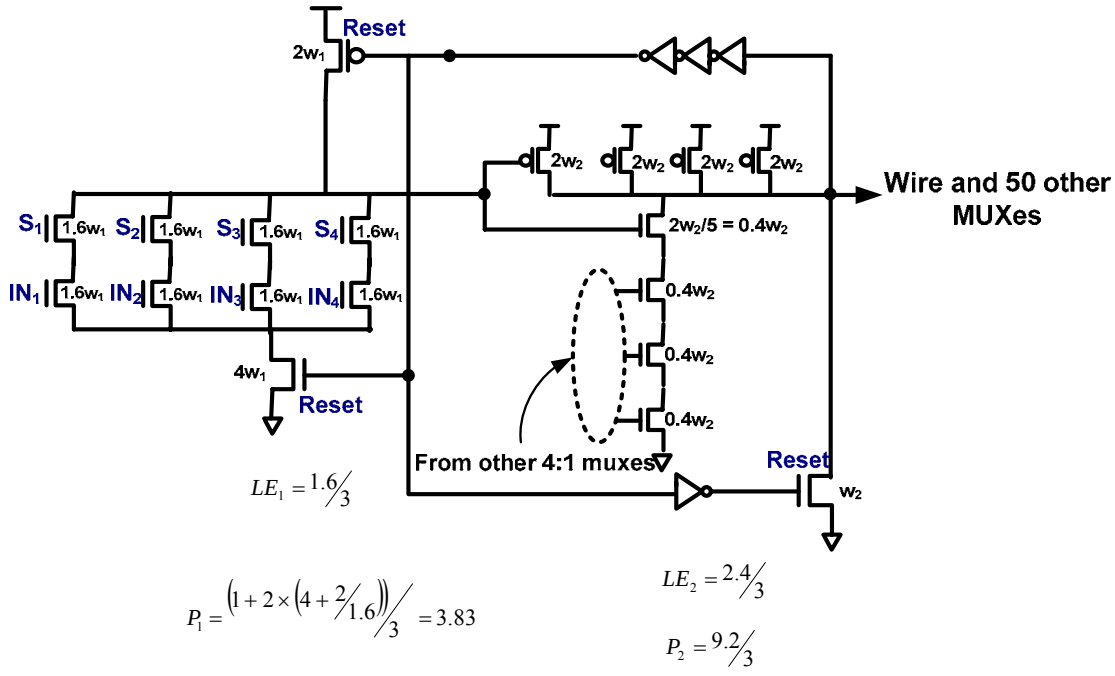

Figure 5.4. Two stage pulse-mode interconnect

Assuming reset transistor is big enough that does not affect delay to the first order (3-4 times other transistors) and using velocity saturated models:

$$Path\_Effort = LE_1 \times LE_2 \times FO = 1.6/3 \times 2.4/3 \times 50 \quad (5.1)$$

$$Effective\_Fanout = \sqrt{Path\_Effort} = 4.6 \quad (5.2)$$

$$Delay = Effective\_fanout \times 2 + Parasitic\_delay = 4.6 \times 2 + 3.83 + 3.1 = 16.13 = 3.27FO4 \quad (5.3)$$

Our study of different possible two stage multiplexers shows that none of them were faster than the above architecture. Since the effective fan-out is more than the

optimum, we consider increasing the number of stages and look at an architecture with 4 stages as shown in Figure 5.5 (optimum fan-out for dynamic gates is around 3). In this case we only need 8 selects for the multiplexers.

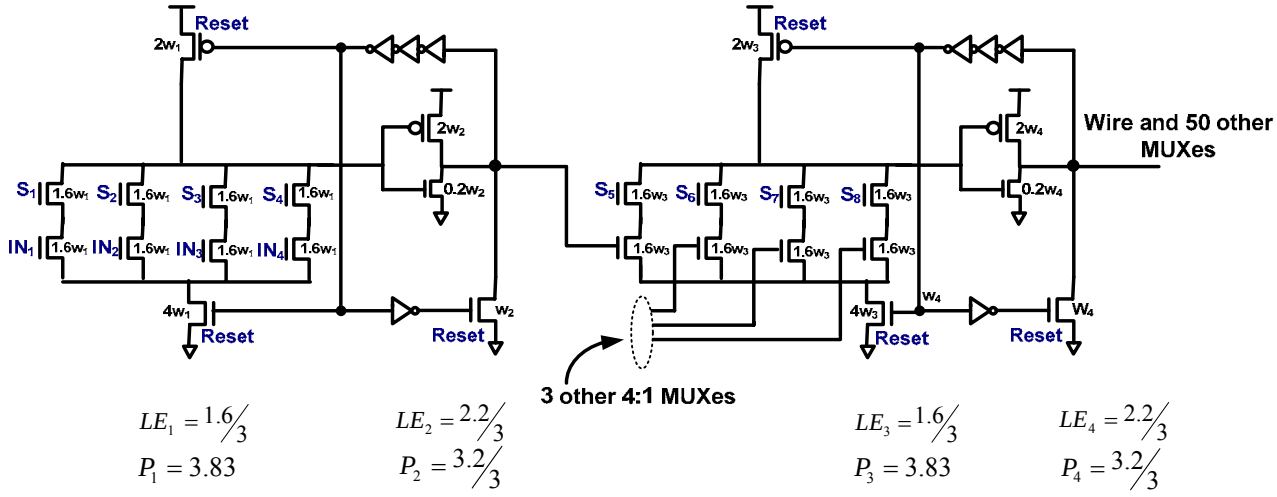

Figure 5.5. Four stage pulse-mode interconnect

$$Path\_Effort = LE_1 \times LE_2 \times LE_3 \times LE_4 \times FO = \left(1.6/3 \times 2.2/3\right)^2 \times 50 \tag{5.4}$$

$$Effective\_Fanout = \sqrt{Path\_Effort} = 1.66 \tag{5.5}$$

$$Delay = Effective\_fanout \times 2 + Parasitic\_delay = 1.66 \times 4 + (3.83 + 3.1) \times 2 = 16.43 = 3.29FO4 \tag{5.6}$$

In this case, the effective fan-out is below the optimum and the overall delay is about the same (the parasitic delay is increased since the number of stages has increased). Since the fan-out for 2-stage was more than optimum and for 4-stage was less than the optimum, the optimum number of stages should therefore be 3. But since the multiplexer and buffer combination is a non-inverting gate, it is hard to build the gate with three stages. However, if we use pass transistors in the multiplexer architecture, we can use a source coupled architecture where the inputs to the gate go to the source of the transistors instead of going to the gate of the transistors. Pulse-mode gates need a reset circuit but let us ignore the reset circuitry to obtain the upper

bound on achievable performance by skewing the basic static architecture. Figure below shows the architecture with skewed gates:

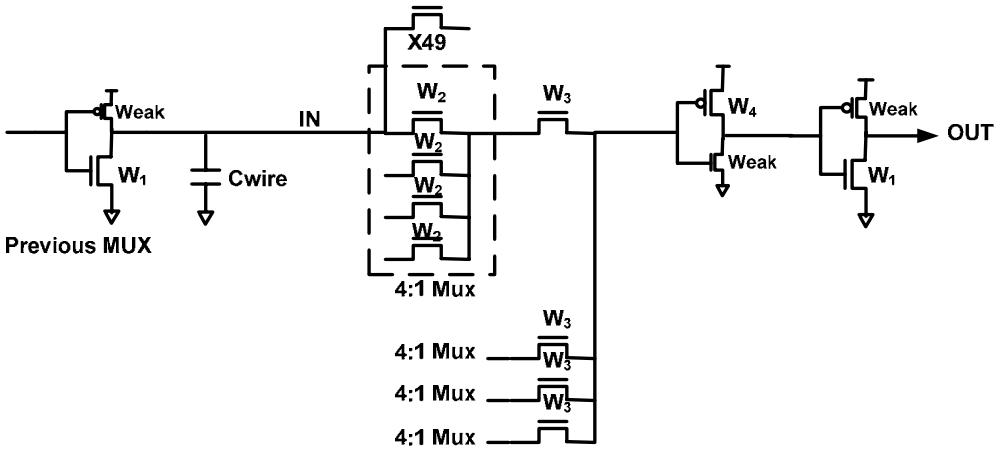

Figure 5.6. Source coupled multiplexer circuit without considering reset circuit

$$Total\_delay = \left(1/w_1\right)(50w_2 + 1.2w_1) + \left(1/w_1 + 1/w_2\right)(w_3 + 4w_2) + \left(1/w_1 + 1/w_2 + 1/w_3\right)(2.1w_4 + 4w_3) + \left(1/w_4\right)(1.2w_1 + 2.1w_4) \quad (5.7)$$

$$Path\_Effort = \left(1/w_1\right)(54w_2 + 5w_3 + 2.1w_4) + \left(1/w_2\right)(5w_3 + 2.1w_4) + \left(1/w_3\right)(2.1w_4) + \left(1/w_4\right)(1.2w_1) \quad (5.8)$$

$$Parasitic = \left(1/w_1\right)(1.2w_1) + \left(1/w_2\right)(4w_2) + \left(1/w_3\right)(4w_3) + \left(1/w_4\right)(2.1w_4) \quad (5.9)$$

MinDelayEquations:

$$\left(54w_2 + 5w_3 + 2.1w_4\right)/w_1 = 1.2w_1/w_4 = \left(1/w_1 + 1/w_2 + 1/w_3\right) \times \left(2.1w_4\right) = f \tag{5.10}$$

$$54w_2/w_1 = \left(5w_3 + 2.1w_4\right)/w_2 \tag{5.11}$$

$$\left(1/w_1 + 1/w_2\right) \times \left(5w_3\right) = 3w_4/w_3 \tag{5.12}$$

$$Delay = 3f - \left(3w_4\right)/w_1 + \left(5w_3\right)/w_2 + Parasitic\_Delay \tag{5.13}$$

From equation (5.10), the delay from the output stage driving the load and all the capacitances of the pass transistors should be equal to the delay of the second stage driving the output stage and should be equal to the delay from all the resistances along the pass transistor path driving the load at the end of the pass transistor. By simplifying equations (5.10) and (5.11), the optimum fan-out is around 2.3 and the optimum delay of the gate is around $2.45FO_4$.

The important point here is that since the gate we are implementing is a multiplexer, the pass transistor sizes cannot be increased. The load is proportional to the sizes of the pass transistors and increasing the size of them will increase the load and increase delay. Therefore, delay through pass transistor will also be an important portion of the overall delay and the gate effectively has three stages. Therefore, between different architectures, using a source-coupled architecture is closer to optimum. In order to make a complete self-reset gate, the following changes should be made in the above circuit:

1. Add one transistor in the main path to disconnect input from the circuit during reset. Doing this will create a stack of 4 NMOS transistors in the critical path and increases delay. However, since the select inputs of the multiplexer do not change during operation, we can combine one level of selects with the reset signal to reduce the number of stacked transistors. To prevent charge sharing, reset signal is combined with the first level of selects.

2. Keepers added to increase noise margin due to coupling.

Figure 5.7 shows the complete self reset circuit.

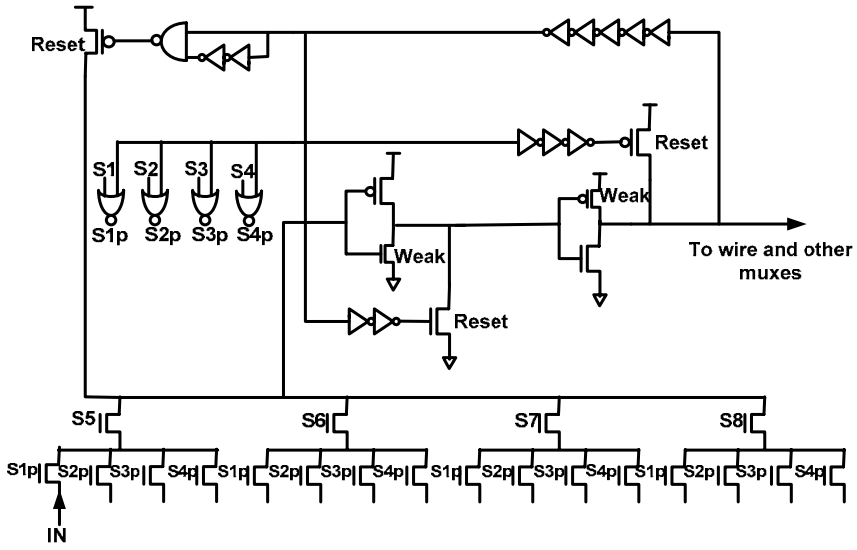

Figure 5.7. Complete self reset source coupled architecture.

In the source-coupled architecture, the default of the input to the interconnect is high and only goes to low upon evaluation. In the other cases the default is low and it goes high upon evaluation. This is actually good since the stage that drives the load has strong nMOS transistor (instead of strong pMOS transistor).

The source coupled architecture is very similar to the static architecture except that normal threshold transistors are used and the gates are skewed since we only care about one transition. Now let us consider using boosted gate nMOS transistors for the pulse mode interconnect as well.

5.2.2 **DRCMOS pulse mode interconnect**

If we use boosted gate nMOS transistor for the multiplexer, we can use them to pass both edges of the input. In this case, output of the multiplexer will have both transitions by the boosted gate transistors and does not need to be reset. Since there is only one output for multiplexer (one input for the buffer), we can use a delayed reset

buffer. In a delayed reset circuit, at every stage the output pulse width is larger than the input pulse width and therefore it is hard to cascade logic. In FPGA, this is not a problem since the cascade of interconnect blocks is always broken by logic blocks, and the cycle time is long enough that the growing pulse width is not an issue either. Figure 5.8 shows the block diagram of the proposed interconnect circuit.

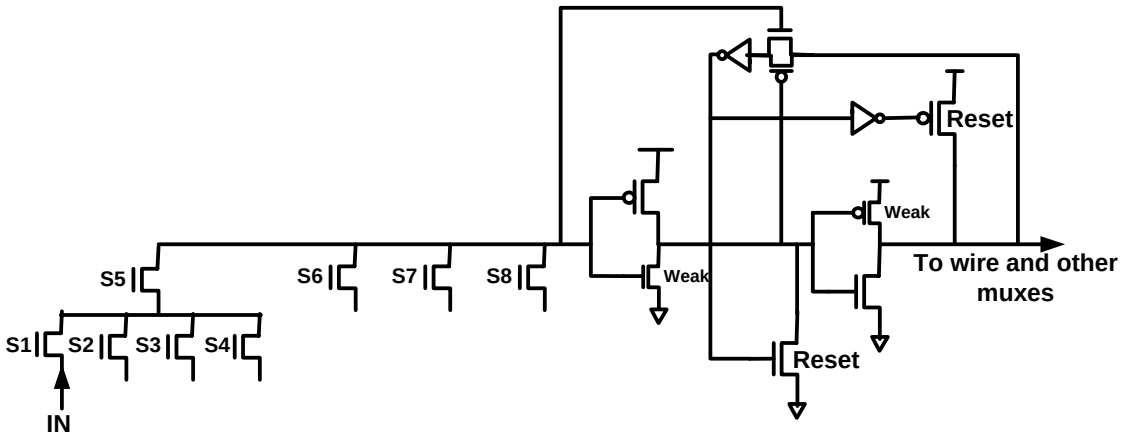

Figure 5.8. DRCMOS interconnect

Using delayed reset for the buffer is very efficient since there is only one input and it is easy to detect input transition to initiate the reset phase. It saves the power overhead of delaying the assert edge to generate the reset signal and also eliminates the need for the foot transistor that is generally necessary in self-reset circuits to avoid multiple evaluations and crossbar current. Also the effective resistance of the boosted gate transistors is much smaller than normal transistors (especially for pull down which is the edge that we care more). Figure 5.9 briefly shows why this is true. For the case of normal pass transistors with nominal supply voltage on the gate, the transistor is off during standby mode. When the input (source of the pass transistor) changes from one to zero, the transistor turn on and propagates the transition. However, in case of thick oxide transistors with boosted supply, the transistor is on during standby mode since it has 1.7V at its gate and 1V at its source. When the source changes from 1 to 0, the pass transistor propagates the transition faster since it is already on. Therefore,

although for thick oxide transistors, the resistance is generally higher, in this particular case, the overall equivalent resistance is smaller.

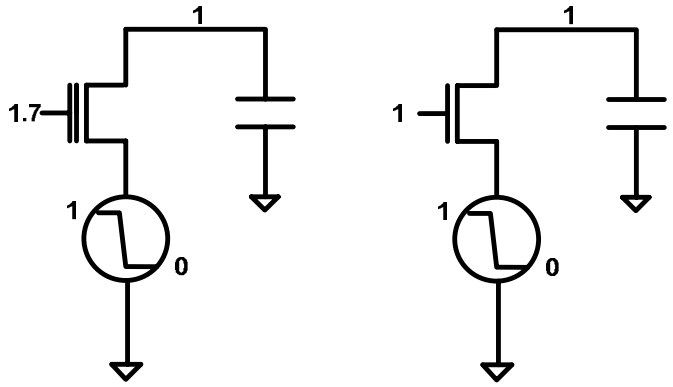

Figure 5.9. Comparison between thick oxide and normal transistors. Thick oxide transistor is always on but the normal transistor has to turn on before propagating a one to zero transition.

5.2.3 **Best pulse-mode architecture**

A comparison between different architectures shows that the source-coupled architecture has the best performance. Also, between the two cases of a DRCMOS source-coupled interconnect and a SRCMOS source-coupled interconnect, the first one has better performance because of two reasons: 1. Resistance of the boosted gate transistors are smaller than the normal transistors even for the falling edge. 2. The reset circuit overhead does not exist for DRCMOS gates. Therefore the best pulse-mode architecture is the DRCMOS source-coupled architecture.

5.2.4 **Low Swing interconnect**

The power breakdown for the pulse mode circuit indicates that about 70% of the power is consumed in driving the load capacitance. Therefore, a 30% reduction in the output swing can lead to 35% reduction in total power (Since power is quadratically related to voltage, total power consumption will be 0.7^2*70%+30% = 65%). This motivates considering low swing architecture for the interconnect. Figure 5.10 shows the block diagram of a low-swing variation of the pulse-mode interconnect. Since

multiplexers are implemented using boosted-gate pass transistors, their output swing is the same as their input. Transistor M_{L2H} has been added to convert the output of the multiplexers to high-swing. Figure 5.11 shows the corresponding energy-performance trade-off curves along with the curve for the original high-swing circuit with optimized supply in a 90nm technology. Based on this figure, the energy performance of the low swing interconnect is worse than the original circuit for all swing levels. The main reason for this behavior can be attributed to the resistance overhead of the low to high converter on the delay. Compared to the resistance of a transistor when gate is connected to V_{dd}, the effective resistance of M_{L2H} is 5X more for V_{low} of 0.6V and 3.3X more for V_{low} of 0.8V. To reduce the effect of this resistance the gate of M_{L2H} can be connected to V_{dd} while V_{low} is reduced to the level that standby leakage is negligible and noise margin is in the acceptable range. Results indicate that about 5-10% better performance is achievable at low energies but the performance is still worse at high energy consumptions and therefore using low swing architecture does not provide better energy for a fixed level of performance.

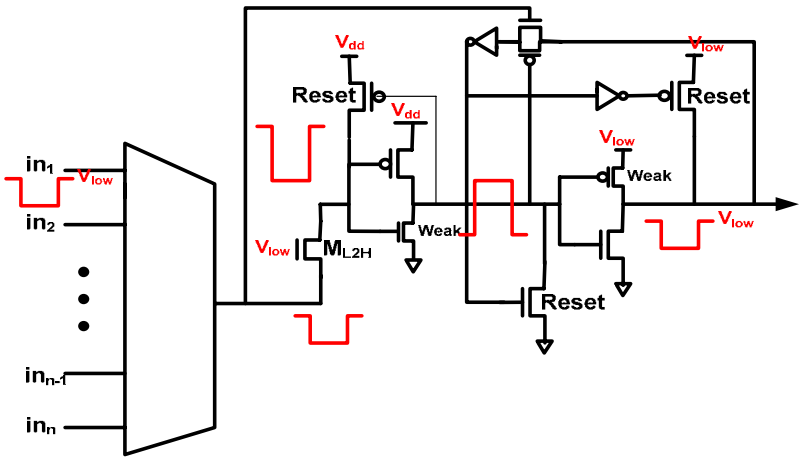

Figure 5.10. Low swing interconnect, a switch acts as a low to high converter after the multiplexer.

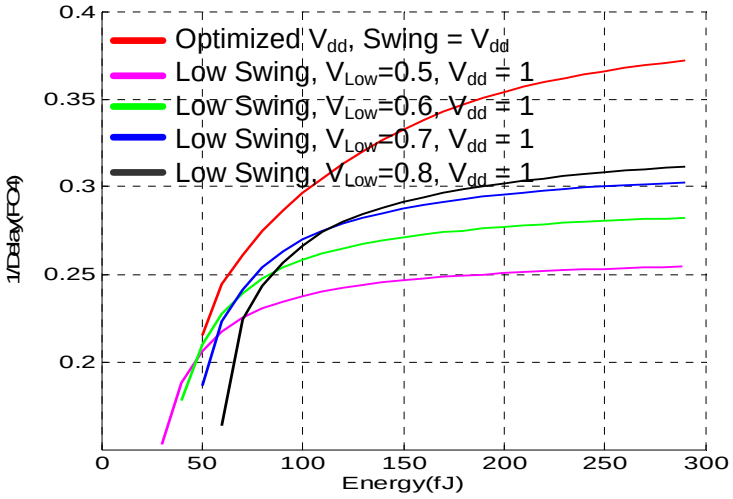

Figure 5.11. Comparison between low swing architecture and normal architecture. Different low voltage has been considered for low swing case.

5.3 Static and pulse-mode interconnect comparison

In order to compare the static architecture, we looked at one real state of the art FPGA interconnect. We calibrated the technology, optimized both static and pulse-mode interconnects and compared the performance. For each of these architectures, we used both low threshold and normal threshold devices. At the end of the optimization, transistor level SPICE simulations where used to verify the results. Figure 5.12 plots the energy delay results obtained from the spice simulations.

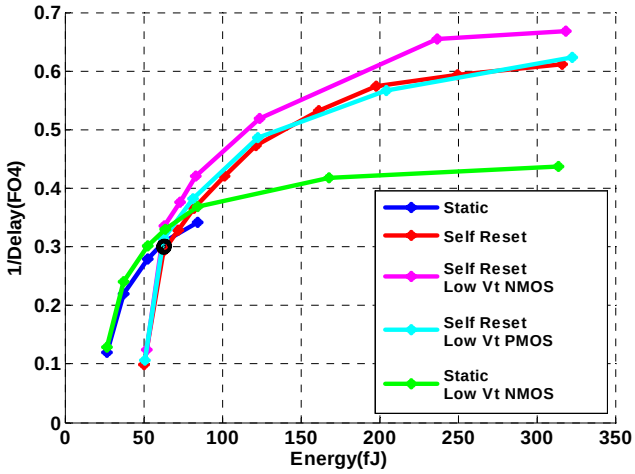

Figure 5.12. Energy-performance for interconnect based on a state of the art FPGA technology

If we look at Figure 5.12, we see that for low energies, static interconnect is more efficient than pulse-mode. As we increase the energy, pulse-mode becomes more efficient. The main reason for pulse mode being less efficient in low energies is because of the twice activity factor of pulses. Figure below confirms this by showing the comparison between a pulse mode circuit with a static circuit at twice power consumption.

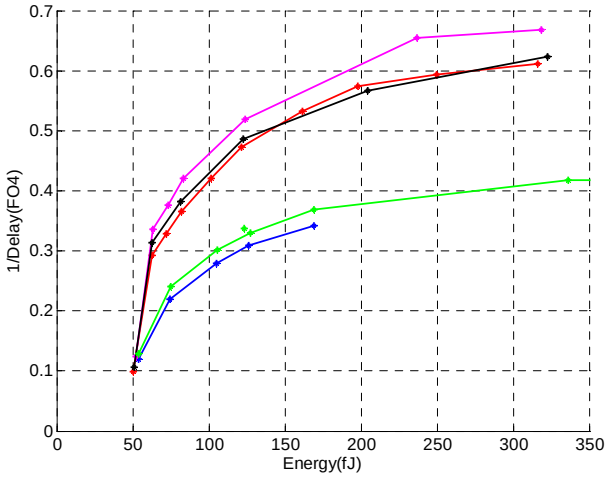

Figure 5.13. Energy-performance curve with energy of static circuit doubled to make a comparison between architectures without considering twice activity factor of pulse-mode.

We consider three variations of this circuit. One uses normal devices in the buffer, one uses low Vt NMOS transistors, and the final one uses low Vt PMOS transistors. The black dot in Figure 5.12 shows the energy-performance point for a static interconnect circuit used in a state of the art FPGA in the same technology.

As shown Figure 5.12 in at an energy cost of around 60fJ, pulse mode interconnect can be used instead of static interconnect without any power penalty, if there are system-level advantages for using pulses. Pulse-mode interconnect can achieve 1.6X better performance at energy cost of around 120fJ. In general, since in pulse-mode logic every transition is a pulse, activity factor is higher than static logic and this is the main reason pulse mode circuit consumes more energy to deliver better performance. Given that dynamic power (interconnect and logic) generally accounts for less than 50% of total FPGA power, the energy cost would be a smaller fraction of the total system energy.

Therefore, pulse-mode interconnect provides 1.3X to 1.5X performance improvement (at 2X dynamic energy) compared to a static interconnect. Given that we have faster interconnects, we now look at the compatible look up table architecture and how a complete pulse mode system would work.

5.4 Compatible Look up table architectures

A static N-input look up table (LUT) architecture is shown in Figure 5.14:

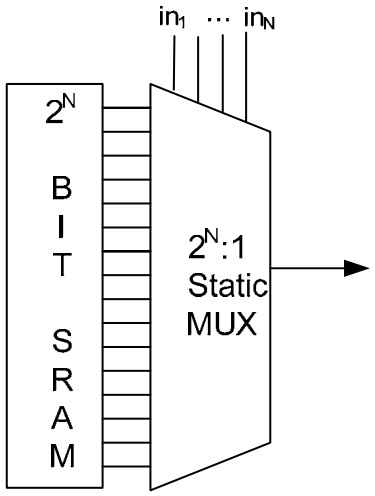

Figure 5.14. Conventional static look up table architecture.

In this look up table, when any of the inputs change, a different memory location is selected and the value of the output changes based on the programmed value in the memory. If interconnect is changed to a pulse-mode interconnect, the incoming pulse to the look up table (from the interconnect) only shows that the value of the corresponding input of the LUT is changed from its previous value. Therefore, state of the signals should be locally stored for all LUT inputs and one flip-flop per input is required where these flops are clocked with the input pulses. Output of the look up table is also signal transitions instead of signal levels. An output pulse is generated if the state of the output is changed. The generated pulse should be wide enough that all the receiving LUTs sense the transition.

5.4.1 **Static LUT with pulse generator**

This is the most straightforward way of implementing the look up table. This is basically the static architecture just with the additions required to be able to work with pulse-mode interconnect:

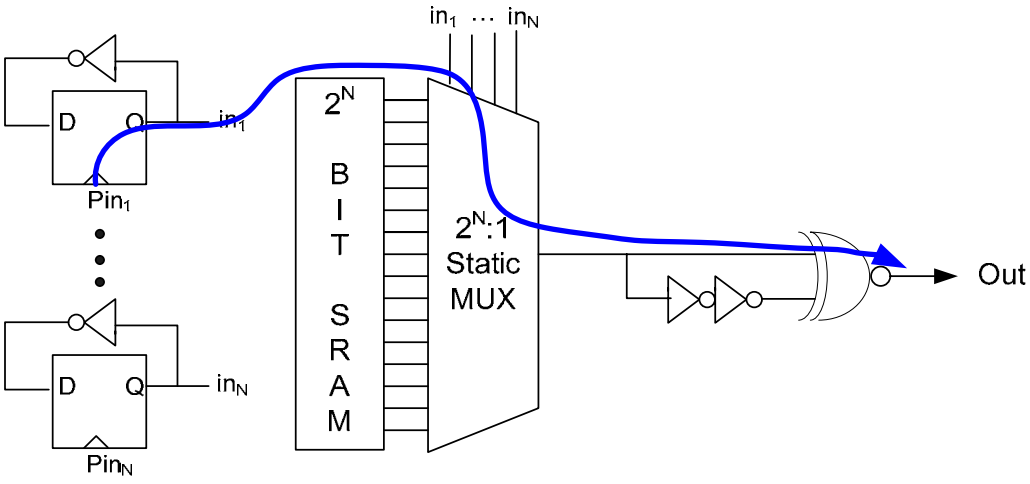

Figure 5.15. Modified static LUT to work with pulse-mode interconnect: Has storage elements for inputs and a pulse generator at the output.

Comparing to a static case, the delay of the lookup table is increased by clock-to-Q delay of the flops and the delay of the pulse generator. The blue line in the figure shows the critical path delay from the arrival of one pulse to the output of the block.

5.4.2 **Self Reset LUT**

We can change the architecture of the static mux in the LUT to a self reset architecture to make it faster. Instead of a pulse generator at the end, we just have the reset circuitry and every time output changes, a pulse is generated at the output. The delay is clk-q delay of the flop plus delay of the self-reset mux.

5.4.3 **Dynamic with pre-evaluation**

Alternatively, we can take advantage of having pulses at the input of the LUT that show one of the inputs has changed. This implementation has the most difference with the conventional logic block. The top level block diagram is shown below:

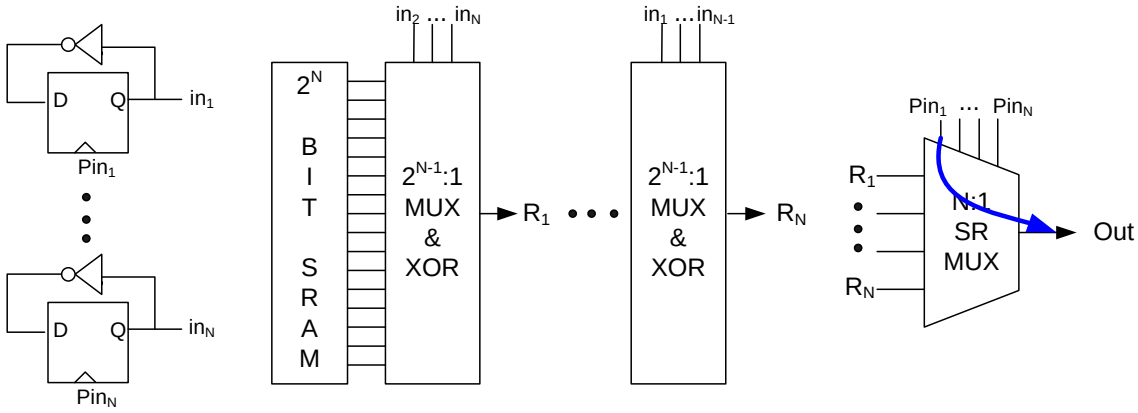

Figure 5.16. Dynamic with pre-evaluation LUT: Has storage elements for the inputs, pre-evaluation logic and a N:1 dynamic multiplexer at the output.

The way this look up table works is that it pre-evaluates the output for all possible one-bit changes in the input combination.[6] The output stage is a self-reset multiplexer that connects one of these pre-evaluated values to the output when the corresponding input pulse arrives. The pre-evaluated values are evaluated again after arrival of each input.

The blue line in the figure shows the critical path from the arrival of one input pulse to the output, assuming that all other inputs are constant. Compared to the conventional static lookup table, the critical path delay of this architecture is significantly decreased. For the static N-input LUT, delay is delay of a 2^N:1 multiplexer where in this case delay is reduced to delay of an N:1 multiplexer. Compared to the previous two architectures (that were compatible with pulse mode interconnect), it has the additional advantage that the clock-to-Q delay of the flops is not in the critical path either. However, since this architecture only considers one input change at each time, there are certain issues that should be considered for this architecture:

[6] There are some problems associated with the assumption that only one of the input pulses arrives at any time and will be addressed later.

- Reevaluation delay: After one input comes, the pre-evaluated values should be generated again. This takes some time and the output of the LUT may be wrong if some input comes before the evaluation is done.
- Tracking inputs: If one input comes, which causes the output to generate a pulse, further changes in the inputs will not be effective until the pulse ends. If some other input pulse comes during this time and goes away before the output pulse is reset (and the gate look at the inputs again) then the output may be wrong.

In order to solve these issues, we propose using an additional slow static path from input to the output to solve both above problems. This static path acts as the correction path that makes sure the output will eventually be correct. This architecture, therefore, has dynamic evaluation and static correction. A pulse is generated at the output in two cases: 1. One or more transitions at the input (which cause output transition in the logic). 2. The stored value of the dynamic output and static output are different after the output pulse has finished. To be functional, the delay of the static mux should be less than one pulse period. This additional static mux, does not increase the complexity or power consumption of the system since it can be combined with the pre-evaluation logic and therefore adding the static path has minimum complexity overhead. The LUT input stage (assuming four input LUT) is then:

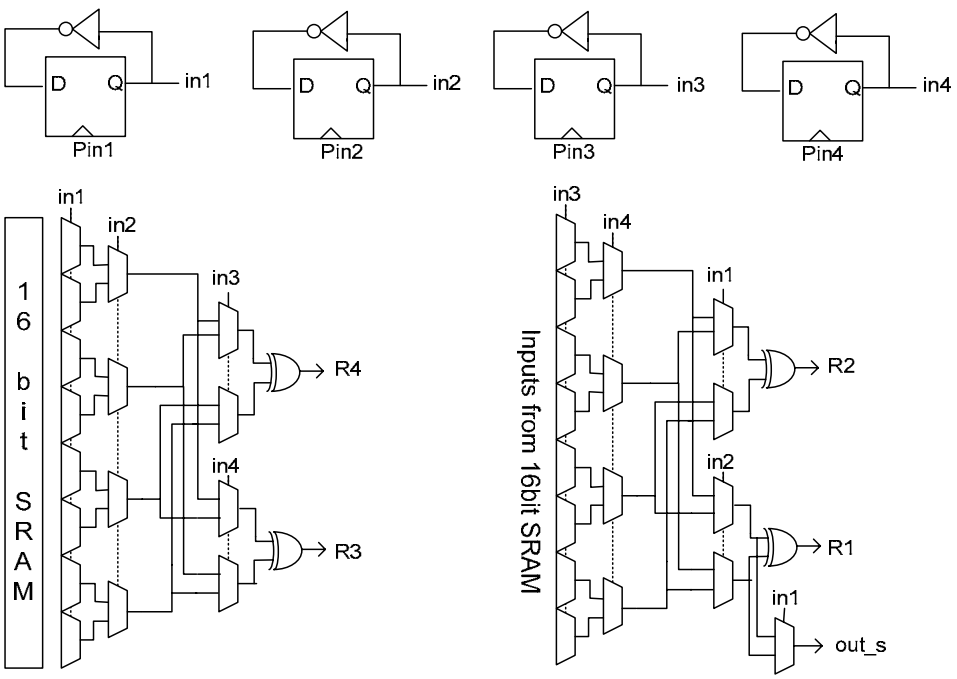

Figure 5.17. LUT input stage

R1 to R4 in the above figure are the pre-evaluated values. In order to add the static output (out_s), just one 2-input multiplexer is added.

LUT output stage is shown below:

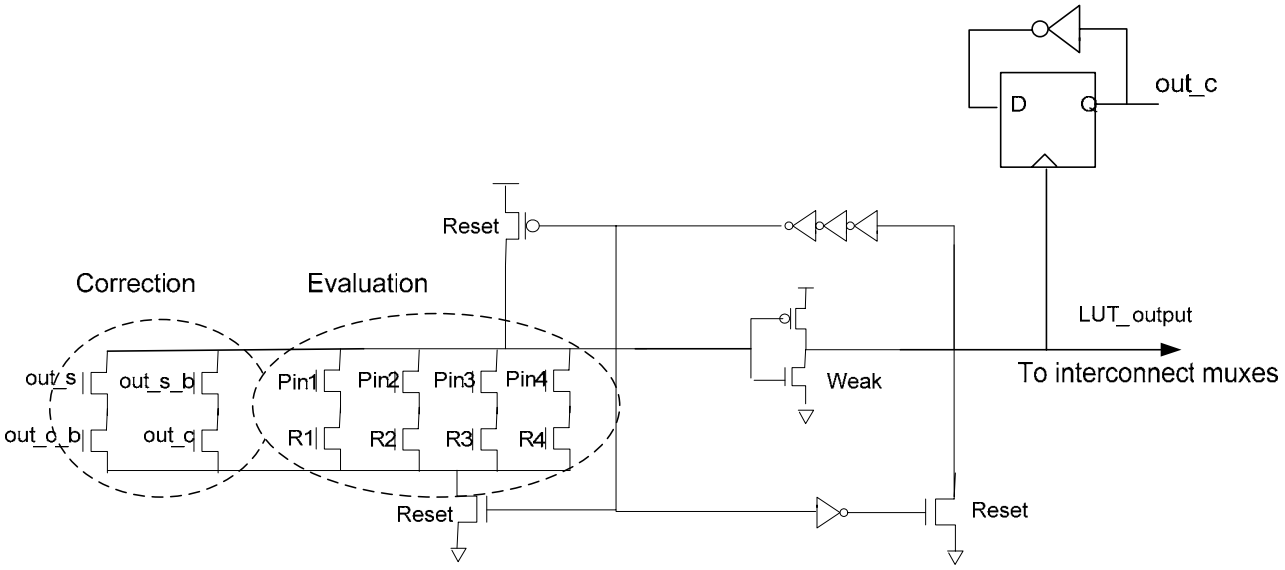

Figure 5.18. LUT output stage

5.4.4 **Complexity comparison:**

For an N-input LUT, both architectures require 2N memory bits. The static LUT, requires (2N-1) Mux2 (Mux2: 2-to-1 multiplexer). For the same number of inputs, dynamic with pre-evaluation, requires (N(2N-1+1)+1) Mux2 in addition to (N+1) flip-flops that are clocked by the input and output pulses. Therefore, dynamic with pre-evaluation LUT requires approximately (N/2) times more Mux2 and the overhead increase linearly with the number of inputs. For a 4-input LUT the complexity is roughly twice.

5.4.5 **Speed comparison:**

In order to do the comparison between different LUT architectures mentioned above, we obtained the energy-performance curve for both static and dynamic with pre-evaluation LUT and compared the speeds on the knee of the curves. For the static mux, the speed for different input is different but for dynamic with pre-evaluation all inputs have the same speed assuming only one input transition. The comparison shows that dynamic with pre-evaluation LUT is on average 2.5X faster than the static LUT.

5.5 System level architecture

The pulse mode FPGA architecture has the two fundamental problems of meta-stability and pulse-propagation mentioned in Chapter 3. Both of these problems can cause the system's performance to be less than expected, as is described next.

5.5.1 **Meta-stability**

In a pulse-mode gate, each transition is a pulse and, therefore the circuit has to decide whether to generate a pulse or not based on the inputs and the logic of the gate. If there is a very short glitch on its input, the pulse generator will ignore it. If the glitch is large, we will get two pulses, one for each transition of the input glitch. But this means that for some size of input pulse that is right on the decision point for this

circuit, the gate can become meta-stable. Figure 5.19 shows the concept. If for example, the in_1 input changes and wants to change the output, but before the output transition is completed another input, in_N, turns the gate off, one of the three cases may happen:

1. A pulse is already generated at the output and finished. The second input transition, therefore, causes another pulse at the output to correct the previous transition.
2. The output has not yet generated the first pulse. The second input will kill the pulse and therefore nothing happens at the output.
3. The output was generating a pulse but the arrival of the second input caused it to stop and therefore output is now at an ambiguous state.

From a system perspective both cases 1 and 2 are fine but case 3 has made the output meta-stable. Having a meta-stable point in a design is not desirable since it leaves some of the nodes in an intermediate (non-digital) state. These nodes will eventually go to the rails (due to the noise in the circuits) but the delay will be arbitrarily long, and the next gate will not necessarily interpret this as a correct pulse. More importantly, if this output goes to the input of multiple gates, different gates may interpret the intermediate level differently and this causes the logic to have incorrect output.

The reason for this problem is that the inputs of the gate have arbitrary arrival times. For many pulsed based circuits, the meta-stability problem is avoided by carefully controlling the timing of the inputs and ensuring that glitches do not happen. However, trying to convert a block of combination logic to use pulses is problematic.

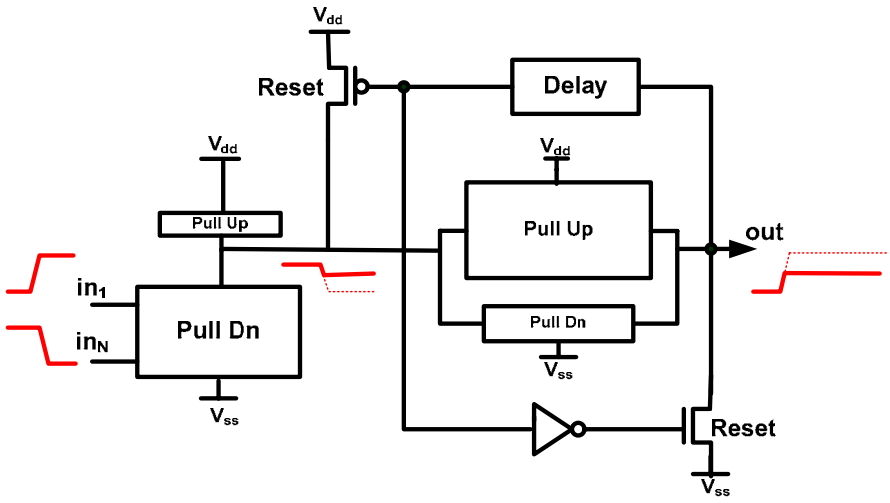

Figure 5.19. Example of meta-stability in a pulse-mode gate

The meta-stability problem is unavoidable since the circuit has to make a decision based on the inputs and inputs have arbitrary arrival times. However, what we can do is to determine the cases that meta-stability can happen. Then make sure that the circuit will eventually resolve to a correct value in case of meta-stability (even if it takes a long time) and make sure that the probability of meta-stability is low enough for our system. Figure below shows how the output stage of the LUT has to change in order to achieve this goal:

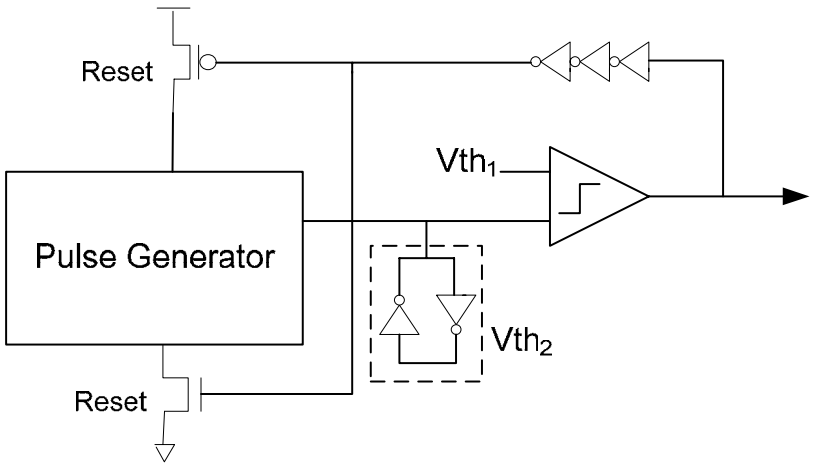

Figure 5.20. Modified LUT output stage to reduce meta-stability effect: Two circuits with different thresholds.

Since the comparator and the latch have different threshold values, the gate will eventually resolve. If a pulse is generated at the output, the amplitude of the pulse should be such that all the receiving gates see the pulse. Therefore, the threshold of the comparator should be high enough that if a pulse is generated at the output, it goes full rail.

5.5.2 **Pulse Propagation**

Pulse mode gates have an internal minimum cycle time constraint, meaning that a new pulse cannot start unless the previous pulse has finished and the gate has reset. Therefore, the time distance between two consecutive output changes is at least twice the reset delay of the gate. This means that the 2nd edge of a glitch can be delayed waiting for the gate to recharge (Figure 5.21).

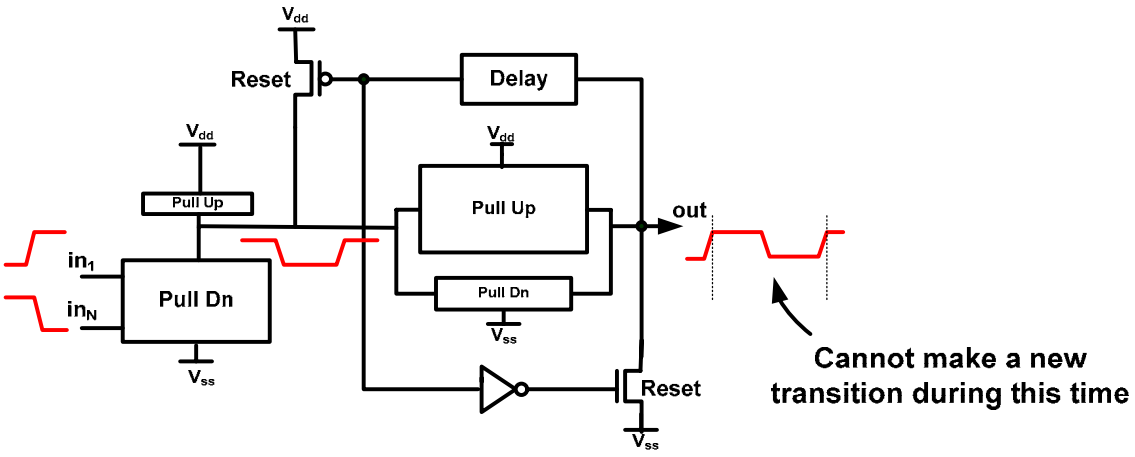

Figure 5.21. Minimum time constraint in pulse-mode gates

Since it is this last edge that contains the correct value, the worst-case delay through a gate can be much larger than its nominal delay, which makes guaranteeing timing difficult in these circuits.

While a single pulse overhead in a system in not very large, the main problem is that, in the worst case, each gate can add a pulse which significantly increases the overall delay of the system. This scenario happens when the inputs have transitions

close to each other and the logic is such that output changes with every input change (for example XOR gates). An example of this is shown in Figure 5.22(a). Assuming that the delay of the gate itself is zero, a zero-to-one transition of in_{1_1} changes out_1 immediately but the later transition of in_{1_2} cannot change out1 until time T has passed from its first transition (this is the time a pulse is generated and the gate has been reset, so it can launch a new transition). Therefore, the second transition of out_1 is delayed by $T-\Delta_1$. Now if we look at the second XOR gate, while in_{2_2} changes the output immediately, out_1 cannot change it right away. Since the gate has an internal timing, out_2 can only change every T seconds and this means that the transition corresponding to the second edge of out1 is now delayed by $T-\Delta_2$. Therefore, if we consider the cascade of the two XOR gates as a block of logic with inputs in_{1_1}, in_{1_2} and in_{2_2}, the last output edge is delayed by $2T-\Delta_1-\Delta_2$ from the last input edge (which is in1_2). The same scenario for a static gate would have had zero delay assuming the gates have zero internal delays.

For static gates, a glitch is either sent without distortion or it is filtered. In pulse mode gates, the output stage amplifies the glitch to a good shape pulse so that all the receiving gates receive a good shape pulse.

In order to solve this problem, we need to find a way to send the second edge as fast as the first edge. The solution we found for this is to send a correction pulse as fast as the main pulse. Figure 5.22(b) shows the same situation as Figure 5.22 (a) when we have another signal to propagate the second edge. By adding a second line to the output that determines if the correction is required or not, the glitches will not propagate and the only overhead will be one glitch overhead for the whole signal path. Most straightforward way of implementing this second line, is to have another interconnect architecture for this signal. We can also code this information in the voltage levels by using tri-level interconnect. (Basically we need to send two bits of information and can code this in voltage or have two separate lines).

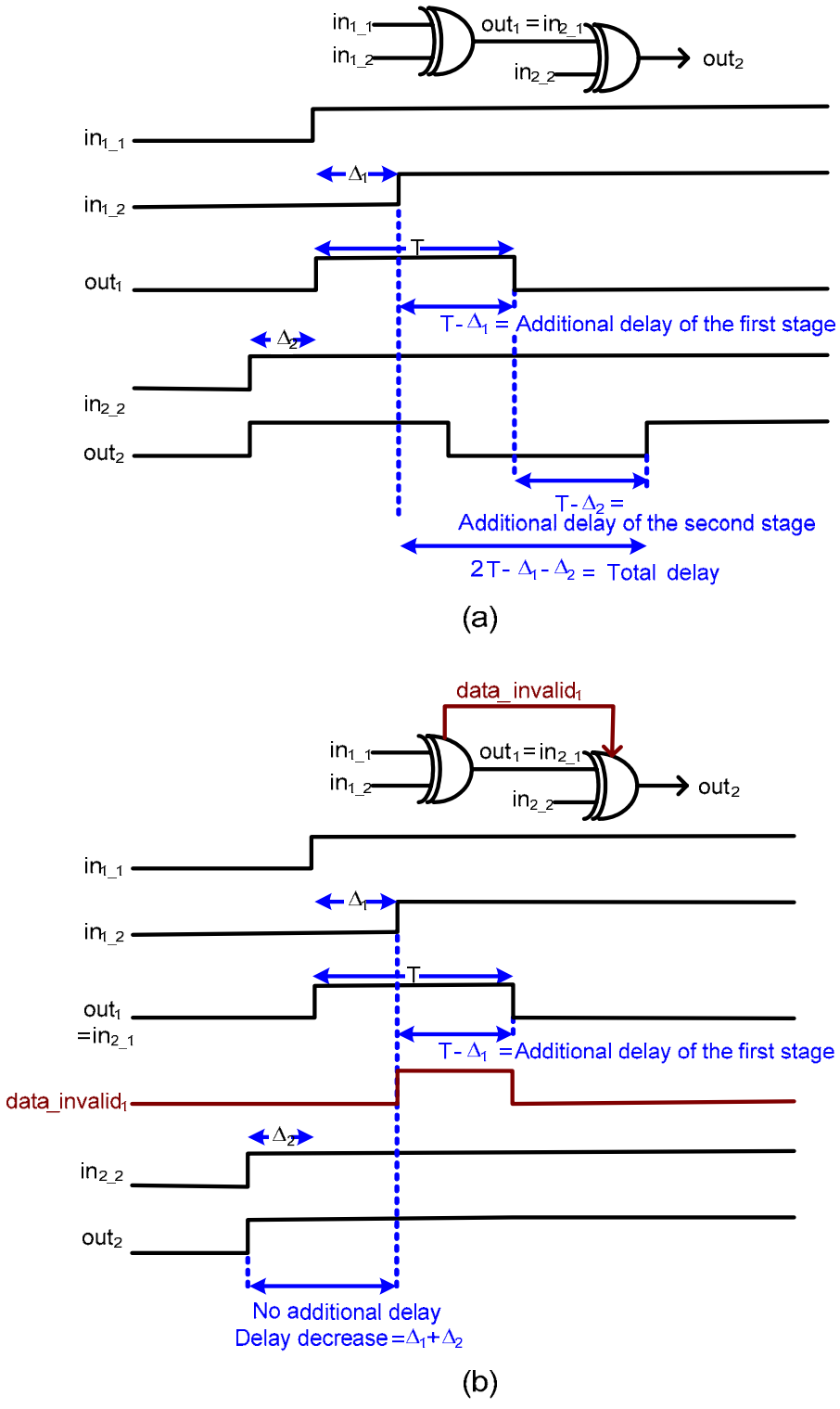

Figure 5.22.(a) Example showing glitch propagation in cascade of gates, (b) solving glitch propagation by using another signal indicating that a glitch has occurred.

5.6 Overall pulse mode system performance

In order to estimate the overall performance of the pulse-mode FPGA, we consider the critical path. Usually, there are 3 interconnect stages between look up tables in the critical path and delay breakdown in approximately 50% in look up tables and 50% in interconnect. Table below defines the notations for the calculations:

Table 5-1. Notations for delays of different blocks

Delay of pulse-mode interconnect	d_{int_pm}
Delay of static interconnect	d_{int_s}
Delay of pulse-mode LUT	d_{lut_pm}
Average delay of static LUT	d_{lut_s}

Based on the previous sections, we have:

$$d_{\text{int}\_s} = 1.5 \times d_{\text{int}\_pm} \tag{5.14}$$

$$d_{lut\_s} = \left(2.5 \times d_{lut\_pm}\right) \text{ or } d_{lut\_s} = \left(d_{lut\_pm}\right) \text{(if correction is needed)} \tag{5.1}$$

$$d_{lut\_s} = 3 \times d_{\text{int}\_s} \tag{5.2}$$

For the best case, that there is no glitch propagation:

$$delay\_s = 0.5 \times d_{\text{int}\_s} + 0.5 \times d_{lut\_s} = 0.5 \times d_{\text{int}\_s} + 0.5 \times 3 \times d_{\text{int}\_s} = 2 \times d_{\text{int}\_s} \tag{5.3}$$

$$delay\_pm = 0.5 \times d_{\text{int}\_pm} + 0.5 \times d_{lut\_pm} = 0.5 \times \left(\frac{1}{1.5}\right) d_{\text{int}\_s} + 0.5 \times \left(\frac{1}{2.5}\right) d_{lut\_s} = 0.93 \times d_{\text{int}\_s} \tag{5.4}$$

$$delay\_pm = 0.47 \times delay\_s \tag{5.5}$$

In the worst case, each stage can add a pulse and the pulse width is 4.5 times delay of the interconnect. Therefore, if we look at the delay from one LUT to the next one:

$$delay\_pm = 3 \times d_{\text{int}\_pm} + d_{lut\_pm} + 4.5 \times d_{\text{int}\_pm} = 7.5 \times \left(\frac{1}{1.5}\right) d_{\text{int}\_s} + \left(\frac{1}{2.5}\right) d_{lut\_s} = 6.2 \times d_{\text{int}\_s} \tag{5.6}$$

while for the static case delay from one lut to the next one is:

$$delay\_s = 3 \times d_{\text{int}\_s} + d_{lut\_s} = 6 \times d_{\text{int}\_s} \tag{5.7}$$

$$delay\_pm = 1.03 \times delay\_s \tag{5.8}$$

Therefore, the overall performance of the pulse mode FPGA is 2.12X static FPGA performance in the best case and 0.97X static FPGA performance in the worst case. For general circuits, the performance is somewhere between these two. We need to either find a delay estimation algorithm that can predict the delay from the logic or use the additional interconnect line to prevent glitch propagation (have 2X higher performance). The power consumption is also 2X power consumption of the static FPGA (dynamic power is 2X).

Chapter 6

Conclusions

Energy-performance tunable circuits have become attractive since they enable the user to change the operating point after fabrication. In this way, the user can optimize the chip taking into account process, temperature and workload variations. This is specially useful for systems with variable workload. In these systems being able to adjust the energy to be close to optimum is essential in optimizing overall power consumption. Adaptive systems usually use dynamic voltage scaling and adaptive body bias to achieve this goal. However, since in scaled technologies adaptive body bias is not very effective in changing threshold voltage, leakage power is not moved into the optimum range.

After looking at different ways of more directly controlling threshold of the transistors, we propose an alternative method to change the effective threshold of the transistors by changing the overdrive voltage. By using skewed supplies, the effective threshold of the transistors can be adjusted. The amount of change in the threshold is equal to the amount of skew between the supplies (the relation is 1:1 – which is much greater than changing body bias). This technique, however, cannot be applied to normal static gates. We investigated the possibility of making an adjustable threshold logic family based on this idea. From the known and normally used logic families, dynamic logic can only tune the circuits in one direction to increase performance, but cannot save leakage. Pulse-mode logic has the potential for tuning of both

performance and leakage. For some symmetric circuits such as decoders using pulse-mode logic is simple and beneficial and further tuning can be achieved by skewing the supplies. However, as is investigated in detail in this thesis, forming a general logic from pulse-mode gates has fundamental system level problems that make them non-practical as an adjustable logic family. Pseudo-static logic, introduced in this work, is an alternative approach that is compatible with normal static gates but retains the ability to tune the effective threshold as well as V_{dd}. Our comparison between static and pseudo-static circuits shows that pseudo-static circuit provides higher performance at the same energy consumption even without skewing the supplies.[7] If we allow skewing the supplies, pseudo-static circuit provides a much wider tunability range in energy-performance space.

FPGA interconnect circuits are a good candidate for pseudo-static circuits. While there is a strong need to improve the performance of FPGA, the design space is small since the circuits are simple and greatly optimized. For this reason, pseudo-static circuits are good candidates since they provide higher performance with minimum change in the architecture. Also, they enable creating a field-tunable FPGA which adds another degree of freedom to the FPGAs. Our study of different interconnect architectures show that pseudo-static interconnect can provide 20% higher performance at the same energy consumption, 35% lower power at the same performance. It also provides tuning range that is twice as wide as today's circuits. These measurements are for a 90nm CMOS technology where leakage power is negligible compared to dynamic power in each interconnect. In a complete system where leakage power is more significant portion of the total power consumption, the tunability range will also increase.

This work leads to the conclusion that pseudo-static circuits have advantages over static gates for interconnect circuits. We also showed that although pseudo-static circuits have pulse gates inside them, they do not pose any fundamental problems

[7] This is true for most of the circuits. If the fan-out of the gate is really small such that the overhead of having weak transistors and slightly larger wires are significant, the pseudo-static architecture itself may not have higher performance than static architecture. Even in this case, by skewing supplies it will provide higher performance.

when they are used in a system. Unlike pulse-mode gates the overhead of glitches in these gates are bounded and moderate. However, interconnect circuits are single input gates, and in order to generalize the use of pseudo-static gates to more general logic we also looked at the glitch cascade problem for multi-input gates. We showed that minor modifications to the timing analysis will take care of the glitch overhead.

Pseudo-static logic can serve as an energy-performance tunable logic family and are suitable for energy adaptive systems. It provides a wide tuning range over the energy-performance space and can easily be handled in the system. Pseudo-static gates have higher area and complexity than static gates, but since they are compatible with static gates, by using them in performance-energy critical parts of a system, these overheads can be made small.

6.1 Future work

There are some issues that still need to be addressed before pseudo-static circuits can be used as a general logic family:

Pseudo-static circuits require two additional supplies that should also be generated on the chip or come to the chip from the outside. A brief discussion on the supply generation is provided in this thesis with a simple proposed technique. We showed that since only a small portion of the total current comes from these supplies and it is internal to the gates, by using a local charge pump, we can use the standby time of the gates to give charge from the main supply of the chip to the additional supplies. The required decoupling capacitor for these supplies is not significant if the activity factor of the gates is not very high. However, a more detailed analysis has to be done on the generation and distribution of these supplies based on the requirement of the specific system and the effect on the area and performance should be evaluated.

The next step would be to implement this circuit style for a more complicated logic block. One good candidate is a look up table, since the combination of pseudo-static look-up table and interconnect creates a completely tunable FPGA. By implementing a pseudo-static look-up table or adder, for example, a set of design rules

can be obtained. These design rules can include questions such as how the logic should be portioned into different pseudo-static and static blocks, what the optimum number of gates in a pseudo-static block is, how much the overhead of area is and what the effect of having longer wires due to increased area is.

Appendix A

Use of Stanford Convex Optimization Tool

In order to compare different interconnect architectures, we used a convex optimization framework using the Stanford Convex Optimization Tool (SCOT) tool implemented by Patil at Stanford [60]. The purpose of the software is to obtain device sizing, supply and threshold voltage optimization for optimal delay, area or energy. In order to achieve this, a digital circuit netlist is converted into a constrained optimization problem. The problem is formulated using analytical delay, area and energy models. The optimization is to obtain the minimum delay subject to constant energy consumption:

Max(Trise, Tfall) < Tmax

Subject to E < Emax

Delay equations can be written as geometric functions of the size of the transistors, supply and threshold voltages. Energy consists of dynamic and static energy where dynamic energy for each node is equal to $\sum CV_{dd}^{2}$ and leakage depends on the off-state current of each circuit and the activity factor. The activity factor for interconnect is easier than regular circuits: For static interconnect, activity factor is equal to 0.5 while for pulse-mode it is equal to 1. For pseudo-static circuit, the activity factor is also 0.5 since each node has either two transitions or zero transition each time the state of the output changes.

Energy and delay equations should be written for channel connected components (CCC). This means that all the transistors that have their source or drain connected to each other should be in one component. For each component, input comes to a transistor gate and output goes to the gate of a transistor in the next component. Since all these equations can be expressed as convex equations, the overall optimization is a convex optimization.

In order to obtain delay equations, the technology that is used should be first calibrated. Different capacitors of the transistors should be estimated. Since we have both delay and power equations the capacitor values should be calibrated for both of them. This can be done in SPICE, for example, by using two circuits one with ideal capacitor and the other one with the transistor. The value of the ideal capacitor is changed until the delay or power is equal between the two circuits. This capacitor value shows the equivalent component capacitor. After estimating the capacitor values, delay equations are also calibrated by fitting the delay of the circuit (measured in SPICE) to an Elmore delay type equation. For this, we did several simulations with different transistor sizes and different supply voltages and obtained the best fitted equation. These equations were used in the optimization.

After sizing the circuit with the optimizer, we ran SPICE simulations with the new sizes and measured energy and delay of the circuit. We changed the fitted equations so that the error between the measured results and the predicted results from the equations were less than 10%. We optimized the circuit for different energy values to obtain the energy-delay trade off curve.

Bibliography

[1] Saxena, S.; Hess, C.; Karbasi, H.; Rossoni, A.; Tonello, S.; McNamara, P.; Lucherini, S.; Minehane, S.; Dolainsky, C.; Quarantelli, M., "Variation in Transistor Performance and Leakage in Nanometer-Scale Technologies," *Electron Devices, IEEE Transactions on* , vol.55, no.1, pp.131-144, Jan. 2008.

[2] Keh-Jeng Chang, "Accurate on-chip variation modeling to achieve design for manufacturability," *System-on-Chip for Real-Time Applications, 2004.Proceedings. 4th IEEE International Workshop on* , pp. 219-222, 19-21 July 2004.

[3] Intel Celeron processor 200^{Δ} sequence datasheet, intel website: "http://download.intel.com/design/processor/datashts/31854602.pdf", page 18.

[4] Chen, T.; Naffziger, S., "Comparison of adaptive body bias (ABB) and adaptive supply voltage (ASV) for improving delay and leakage under the presence of process variation," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.11, no.5, pp. 888-899, Oct. 2003.

[5] Burd, T.D.; Pering, T.A.; Stratakos, A.J.; Brodersen, R.W., "A dynamic voltage scaled microprocessor system," *Solid-State Circuits, IEEE Journal of* , vol.35, no.11, pp.1571-1580, Nov 2000.

[6] Nowka, K.J.; Carpenter, G.D.; MacDonald, E.W.; Ngo, H.C.; Brock, B.C.; Ishii, K.I.; Nguyen, T.Y.; Burns, J.L., "A 32-bit PowerPC system-on-a-chip with support

for dynamic voltage scaling and dynamic frequency scaling," *Solid-State Circuits, IEEE Journal of* , vol.37, no.11, pp. 1441-1447, Nov 2002.

[7] Intel Pentium M processor on 90 nm process with 2-MB L2 Cache, "http://download.intel.com/design/mobile/datashts/30218908.pdf"

[8] Pering, T.; Burd, T.; Brodersen, R., "The simulation and evaluation of dynamic voltage scaling algorithms," *Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on* , vol., no., pp. 76-81, 10-12 Aug 1998.

[9] Burd, T.D.; Brodersen, R.W., "Design issues for Dynamic Voltage Scaling," *Low Power Electronics and Design, 2000. ISLPED '00. Proceedings of the 2000 International Symposium on* , pp. 9-14, 2000.

[10] Ernst, D.; Nam Sung Kim; Das, S.; Pant, S.; Rao, R.; Toan Pham; Ziesler, C.; Blaauw, D.; Austin, T.; Flautner, K.; Mudge, T., "Razor: a low-power pipeline based on circuit-level timing speculation," *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on* , pp. 7-18, 3-5 Dec. 2003.

[11] Das, S.; Roberts, D.; Seokwoo Lee; Pant, S.; Blaauw, D.; Austin, T.; Flautner, K.; Mudge, T., "A self-tuning DVS processor using delay-error detection and correction," *Solid-State Circuits, IEEE Journal of* , vol.41, no.4, pp. 792-804, April 2006.

[12] Nose, K.; Sakurai, T., "Optimization of V_{DD} and V_{TH} for low-power and high-speed applications," *Design Automation Conference, 2000. Proceedings of the ASP-DAC 2000. Asia and South Pacific* , pp.469-474, 2000.

[13] Tschanz, J.W.; Narendra, S.; Nair, R.; De, V., "Effectiveness of adaptive supply voltage and body bias for reducing impact of parameter variations in low power and high performance microprocessors," *Solid-State Circuits, IEEE Journal of* , vol.38, no.5, pp. 826-829, May 2003.

[14] Mutoh, S.; Douseki, T.; Matsuya, Y.; Aoki, T.; Shigematsu, S.; Yamada, J., "1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS," *Solid-State Circuits, IEEE Journal of* , vol.30, no.8, pp.847-854, Aug 1995.

[15] Hyo-Sig Won; Kyo-Sun Kim; Kwang-Ok Jeong; Ki-Tae Park; Kyu-Myung Choi; Jeong-Taek Kong, "An MTCMOS design methodology and its application to mobile computing," *Low Power Electronics and Design, 2003. ISLPED '03. Proceedings of the 2003 International Symposium on* , pp. 110-115, 25-27 Aug. 2003.

[16] Wei, L.; Chen, Z.; Roy, K.; Johnson, M.C.; Ye, Y.; De, V.K., "Design and optimization of dual-threshold circuits for low-voltage low-power applications," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.7, no.1, pp.16-24, Mar 1999.

[17] Tschanz, J.W.; Kao, J.T.; Narendra, S.G.; Nair, R.; Antoniadis, D.A.; Chandrakasan, A.P.; De, V., "Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage," *Solid-State Circuits, IEEE Journal of* , vol.37, no.11, pp. 1396-1402, Nov 2002.

[18] Kim, C.H.; Roy, K., "Dynamic V_{TH} scaling scheme for active leakage power reduction," *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings* , pp.163-167, 2002.

[19] Nose, K.; Hirabayashi, M.; Kawaguchi, H.; Seongsoo Lee; Sakurai, T., "V_{TH}-hopping scheme to reduce subthreshold leakage for low-power processors," *Solid-State Circuits, IEEE Journal of* , vol.37, no.3, pp.413-419, Mar 2002.

[20] Ming-Jer Chen; Huan-Tsung Huang; Chin-Shan Hou; Kuo-Nan Yang, "Back-gate bias enhanced band-to-band tunneling leakage in scaled MOSFET's," *Electron Device Letters, IEEE* , vol.19, no.4, pp.134-136, Apr 1998.

[21] Neau, C.; Roy, K., "Optimal body bias selection for leakage improvement and process compensation over different technology generations," *Low Power*

Electronics and Design, 2003. ISLPED '03. Proceedings of the 2003 International Symposium on , pp. 116-121, 25-27 Aug. 2003.

[22] Kohno, I.; Sano, T.; Katoh, N.; Yano, K., "Threshold cancelling logic (TCL): a post-CMOS logic family scalable down to 0.02 μm," *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International* , pp.218-219, 459, 2000.

[23] Jan M. Rabaey, Anantha Chandrakasan and Borivoje Nikolic. *Digital Integrated Circuits, A Design Perspective*. New Jersey: Pearson Education, Inc. , 2003.

[24] Solomatnikov, A.; Somasekhar, D.; Sirisantana, N.; Roy, K., "Skewed CMOS: noise-tolerant high-performance low-power static circuit family," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.10, no.4, pp. 469-476, Aug 2002.

[25] Amrutur, B.S.; Horowitz, M.A., "Fast low-power decoders for RAMs," *Solid-State Circuits, IEEE Journal of* , vol.36, no.10, pp.1506-1515, Oct 2001.

[26] Hamzaoglu, F.; Stan, M.R., "Split-path skewed (SPS) CMOS buffer for high performance and low power applications," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on* , vol.48, no.10, pp.998-1002, Oct 2001.

[27] Neil H. E. Weste and David Haris. *CMOS VLSI Design, A Circuit and Systems Perspective.* Pearson Education, 2005.

[28] Endoh, T.; Sunaga, K.; Sakuraba, H.; Masuoka, F., "An on-chip 96.5% current efficiency CMOS linear regulator using a flexible control technique of output current," *Solid-State Circuits, IEEE Journal of* , vol.36, no.1, pp.34-39, Jan 2001.

[29] Gu-Yeon Wei; Horowitz, M., "A fully digital, energy-efficient, adaptive power-supply regulator ," *Solid-State Circuits, IEEE Journal of* , vol.34, no.4, pp.520-528, Apr 1999.

[30] Jaeseo Lee; Hatcher, G.; Vandenberghe, L.; Chih-Kong Ken Yang, "Evaluation of Fully-Integrated Switching Regulators for CMOS Process Technologies," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.15, no.9, pp.1017-1027, Sept. 2007.

[31] TianRui Ying; Wing-Hung Ki; Mansun Chan, "Area-efficient CMOS charge pumps for LCD drivers," *Solid-State Circuits, IEEE Journal of* , vol.38, no.10, pp. 1721-1725, Oct. 2003.

[32] Ha, V.H.S.; Sung-Kyu Choi; Jong-Gu Jeon; Geon-Hyoung Lee; Won-Kap Jang; Woo-Sung Shim, "Real-time audio/video decoders for digital multimedia broadcasting," *System-on-Chip for Real-Time Applications, 2004.Proceedings. 4th IEEE International Workshop on* , pp. 162-167, 19-21 July 2004.

[33] Guillaud, M.; Burg, A.; Mailaender, L.; Haller, B.; Rupp, M.; Beck, E., "From basic concept to real-time implementation: prototyping WCDMA downlink receiver algorithms-a case study," *Signals, Systems and Computers, 2000. Conference Record of the Thirty-Fourth Asilomar Conference on* , vol.1, pp.84-88 vol.1, 2000.

[34] Reay, D.S.; Green, T.C.; Williams, B.W., "Field programmable gate array implementation of a neural network accelerator," *Hardware Implementation of Neural Networks and Fuzzy Logic, IEE Colloquium on* , pp.2/1-2/3, 9 Mar 1994.

[35] Bastos, J.L.; Figueroa, H.P.; Monti, A., "FPGA implementation of neural network-based controllers for power electronics applications," *Applied Power Electronics Conference and Exposition, 2006. APEC '06. Twenty-First Annual IEEE* , pp. 6 pp.-, 19-23 March 2006.

[36] Hutchings, B.L.; Nelson, B.E., "Gigaop DSP on FPGA," *Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on* , vol.2, pp.885-888 vol.2, 2001.

[37] Abeysekera, S.S.; Charoensak, C., "FPGA implementation of a sigma-delta (Σ-Δ) architecture based digital I-F stage for software radio," *ASIC/SOC Conference, 2002. 15th Annual IEEE International* , pp. 341-345, 25-28 Sept. 2002.

[38] Yin-Tsung Hwang; Jih-Cheng Han, "A novel FPGA design of a wireless block transmission channel equalizer," *ASICs, 2000. AP-ASIC 2000. Proceedings of the Second IEEE Asia Pacific Conference on* , pp.119-122, 2000.

[39] Walke, R.L.; Dudley, J.; Sadler, D., "An FPGA based digital radar receiver for soft radar," *Signals, Systems and Computers, 2000. Conference Record of the Thirty-Fourth Asilomar Conference on* , vol.1, pp.73-77 vol.1, 2000.

[40] Mangal, S.K.; Deshmukh, R.B.; Badghare, R.M.; Patrikar, R.M., "FPGA Implementation of Low Power Parallel Multiplier," *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on* , pp.115-120, 6-10 Jan. 2007.

[41] Man Yan Kong; Langlois, J.M.P.; Al-Khalili, D., "Efficient FPGA implementation of complex multipliers using the logarithmic number system," *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on* , pp.3154-3157, 18-21 May 2008.

[42] Fritz Mayer-Lindenberg; Valerij Beller, "An FPGA-based floating-point processor array supporting a high-precision dot product," *Field Programmable Technology, 2006. FPT 2006. IEEE International Conference on* , pp.317-320, Dec. 2006.

[43] Kuon, I.; Rose, J., "Measuring the Gap Between FPGAs and ASICs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.26, no.2, pp.203-215, Feb. 2007.

[44] Zuchowski, P.S.; Reynolds, C.B.; Grupp, R.J.; Davis, S.G.; Cremen, B.; Troxel, B., "A hybrid ASIC and FPGA architecture," *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on* , pp. 187-194, 10-14 Nov. 2002.

[45] Chow, P.; Soon Ong Seo; Rose, J.; Chung, K.; Paez-Monzon, G.; Rahardja, I., "The design of a SRAM-based field-programmable gate array-Part II: Circuit design and layout," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.7, no.3, pp.321-330, Sep 1999.

[46] Betz, V.; Rose, J., "Circuit design, transistor sizing and wire layout of FPGA interconnect," *Custom Integrated Circuits, 1999. Proceedings of the IEEE 1999* , pp.171-174, 1999.

[47] DeHon, A.; Rubin, R., "Design of FPGA interconnect for multilevel metallization," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.12, no.10, pp. 1038-1050, Oct. 2004.

[48] Yi-Ru He; Wai-Kei Mak, "Optimal Buffering of FPGA Interconnect for Expected Delay Optimization," *Field-Programmable Technology, 2007. ICFPT 2007. International Conference on* , pp.289-292, 12-14 Dec. 2007.

[49] Lemieux, G. and Lewis, D. "Circuit design of routing switches," *Proceedings of the 2002 ACM/SIGDA Tenth international Symposium on Field-Programmable Gate Arrays,* Feb 2002.

[50] Shang, L., Kaviani, A. S., and Bathala, K., "Dynamic power consumption in Virtex™-II FPGA family," *Proceedings of the 2002 ACM/SIGDA Tenth international Symposium on Field-Programmable Gate Arrays,* Feb 2002.

[51] Dobbelaere I.; Horowitz M.; El Gamal A., "Regenerative feedback repeaters for programmable interconnections, " *Solid State Circuits, IEEE Journal of,* vol. 30, no. 11, pp.1246-1253, Nov. 1995.

[52] Fei Li; Yan Lin; Lei He, "FPGA power reduction using configurable dual-Vdd," *Design Automation Conference, 2004. Proceedings. 41st* , vol., no., pp. 735-740, 2004.

[53] George, V.; Hui Zhang; Rabaey, J., "The design of a low energy FPGA," *Low Power Electronics and Design, 1999. Proceedings. 1999 International Symposium on* , vol., no., pp. 188-193, 1999.

[54] Rahman, A. and Polavarapuv, V., "Evaluation of low-leakage design techniques for field programmable gate arrays," In *Proceedings of the 2004 ACM/SIGDA 12th international Symposium on Field Programmable Gate Arrays*, Feb 2004.

[55] Manohar, R., "Reconfigurable Asynchronous Logic," *Custom Integrated Circuits Conference, 2006. CICC '06. IEEE* , vol., no., pp.13-20, 10-13 Sept. 2006.

[56] Mak, T.; D'Alessandro, C.; Sedcole, P.; Cheung, P.Y.K.; Yakovlev, A.; Luk, W., "Implementation of Wave-Pipelined Interconnects in FPGAs," *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on* , vol., no., pp.213-214, 7-10 April 2008.

[57] Lin, M.; El Gamal, A.; Lu, Y.-C.; Wong, S., "Performance Benefits of Monolithically Stacked 3-D FPGA," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.26, no.2, pp.216-229, Feb. 2007.

[58] Singh, A.; Mukherjee, A.; Macchiarulo, L.; Marek-Sadowska, M., "PITIA: an FPGA for throughput-intensive applications," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.11, no.3, pp. 354-363, June 2003.

[59] Krishnan, R.; de Gyvez, J.P., "Low energy switch block for FPGAs," *VLSI Design, 2004. Proceedings. 17th International Conference on* , vol., no., pp. 209-214, 2004.

[60] Patil, D.; Yun, S.; Kim, S.-J.; Cheung, A.; Horowitz, M.; Boyd, S., "A new method for design of robust digital circuits," *Quality of Electronic Design, 2005. ISQED 2005. Sixth International Symposium on* , vol., no., pp. 676-681, 21-23 March 2005.

[61] Query from www.chipworks.com from a report on a recent FPGA chip.

[62] Ron Ho; Amrutur, B.; Ken Mai; Wilburn, B.; Mori, T.; Horowitz, M., "Applications of on-chip samplers for test and measurement of integrated circuits," *VLSI Circuits, 1998. Digest of Technical Papers. 1998 Symposium on* , vol., no., pp.138-139, 11-13 Jun 1998.

[63] Chiricescu, S.; Leeser, M.; Vai, M.M., "Design and analysis of a dynamically reconfigurable three-dimensional FPGA," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.9, no.1, pp.186-196, Feb 2001.

[64] Amrutur B., "Design and analysis of fast low power SRAMs ", *PhD thesis*, Stanford University, Aug 1999.

[65] Chappell, T.I.; Chappell, B.A.; Schuster, S.E.; Allan, J.W.; Klepner, S.P.; Joshi, R.V.; Franch, R.L., "A 2-ns cycle, 3.8-ns access 512-kb CMOS ECL SRAM with a fully pipelined architecture," *Solid-State Circuits, IEEE Journal of* , vol.26, no.11, pp.1577-1585, Nov 1991.

[66] Park, H.-C.; Yang, S.-K.; Jung, M.-C.; Kang, T.-G.; Kim, S.-C.; Sohn, K.-M.; Bae, D.-G.; Kim, S.-S.; Kim, K.-H.; Sohn, B.-S.; Kim, H.-S.; Byun, H.-G.; Shin, Y.-S.; Lim, H.-K., "A 833 Mb/s 2.5 V 4 Mb double data rate SRAM," *Solid-State Circuits Conference, 1998. Digest of Technical Papers. 1998 IEEE International* , vol., no., pp.356-357, 464, 5-7 Feb 1998.