VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

High Speed Circuit Design Based on a Hybrid of Conventional and Wave Pipelining

DISSERTATION SUBMITTED TO THE FACULTY OF THE
VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in
Electrical Engineering

by

Jos Budi Sulistyo

Advisor:

Dr. Dong S. Ha

Advisory Committee:

Dr. James R. Armstrong
Dr. F. Gail Gray
Dr. Jeffrey H. Reed
Dr. G. Q. Lu

Blacksburg, Virginia
September 2005

Keywords: high speed CMOS VLSI, pipelining,
wave pipelining, arithmetic circuits

High Speed Circuit Design Based on a Hybrid of Conventional and Wave Pipelining

Jos Budi Sulistyo

ABSTRACT

The increasing capabilities of multimedia appliances demand arithmetic circuits with higher speed and reasonable power dissipation. A common technique to attain those goals is synchronous pipelining, which increases the throughput of a circuit at the expense of longer latency, and it is therefore suitable where throughput takes priority over latency.

Two synchronous pipelining approaches, conventional pipelining and wave pipelining, are commonly employed. Conventional pipelining uses registers to divide the circuit into shorter paths and synchronize among sub-blocks, while wave pipelining uses the delay of combinational elements to perform those tasks. As wave pipelining does not introduce additional registers, in principle, it can attain a higher throughput and lower power consumption. However, its throughput is limited by delay variations, while delay balancing often leads to increased power dissipation.

This dissertation proposes a hybrid pipelining method called *HyPipe*, which divides the circuit into sub-blocks using conventional pipelining, and applies wave pipelining to each sub-block. Each sub-block is derived from a single base circuit, leading to a better delay balance and greater throughput than with heterogeneous circuits. Another requirement for wave pipelining to achieve high speed is short signal rise and fall times. Since CMOS wide-NAND and wide-NOR gates exhibit long rise and fall times and large delay variations, they should be decomposed. We show that the straightforward decomposition using alternating levels of NAND and NOR gates results in large delay variations. Therefore, we propose a new decomposition method using only one gate type. Our method reduces delay variations by up to 39%, and it is appropriate for wave pipelining based on standard-cells or sea-of-gates.

We laid out a 4×4 HyPipe multiplier as a proof of concept and performed a post-layout SPICE simulation. The multiplier achieves a throughput of 4.17 billion multiplications per second or a clock period of 2.52 four-load inverter delays, which is almost twice the speed of any existing multiplier in the open literature. When the supply voltage is reduced to 1.2 V from 1.8 V, its power consumption is reduced from 76.2 mW to 18.2 mW while performing 2.33 billion multiplications per second.

Acknowledgments

Special thanks are due to my advisor and commitee chairman Dr. Dong S. Ha, with whose patience, understanding, support, and invaluable advice, this work has been accomplished. I would also like to express my gratitude to Dr. James R. Armstrong, Dr. F. Gail Gray, Dr. Guo-Quan Lu and Dr. Jeffrey H. Reed for serving as my committee members.

I would also express my special thanks to the members and former members of the VTVT (Virginia Tech VLSI for Telecommunications) Group at the Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, namely – in no particular order – Dr. Nathaniel J. August, Woocheol Chung, Sajay Jose, Sweta Kalantri, Jina Kim, Dr. Suk Won Kim, Hyung-Jin Lee, Jong-Suk Lee, Kevin Marsden, Jonathan Perry, Venkataramanujam Srinivasan, Rajesh Thirugnanam, and Shen Wang, for all their help and suggestions along the course of this work.

Last but not least, my parents and my sister deserve a special kind of thanks. While they were not directly involved in this work, they nonetheless supported me with their encouragement, understanding and patience, particularly when I faced some particularly difficult problems. Without their support, it may not have been possible for me to accomplish this work. I do not know how I could properly thank them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The increasing capabilities of multimedia appliances demand higher speed for arithmetic circuits. As a result, circuit design techniques to push the speed of arithmetic circuits into the gigahertz range with reasonable power dissipation have been investigated intensively [3] [6][9][13][26][43][59]. A most commonly used technique is pipelining, which increases the throughput of a circuit at the expense of a longer latency [56]. Hence, pipelining is suitable for applications, where a high throughput is critical while a long latency is tolerable such as arithmetic circuits, in digital signal processing. For a given throughput, pipelining can also be used to reduce power consumption by reducing the supply voltage V_{DD} until the circuit meets the desired throughput [28][59]. In this approach, pipelining is used to increase the throughput attained by a circuit to be significantly higher than needed by the application. Afterward, the supply voltage is reduced until the throughput just satisfies the requirement; this reduction in V_{DD} is expected to reduce power consumption.

Two kinds of synchronous pipelining approaches, conventional pipelining and wave pipelining, are commonly employed for high speed circuits [5]. In *conventional pipelining*, clocked elements such as latches and flip-flops are used to divide the circuit's critical paths into shorter paths [56]. In *wave pipelining*, the delay of combinational elements is utilized for synchronization [4]. Both approaches have their advantages and shortcomings. Conventional pipelining tends to increase power consumption greatly, due to introduction of registers and associated clock power. Additionally, a large number of pipeline stages are necessary to attain a high throughput, and the latency of the circuit increases proportionally to the number of stages. In principle, wave pipelining avoids these problems, but the throughput is limited by the difference between the longest and the shortest path in the wave-pipelined circuit. Equalizing the path lengths may require the insertion of logic gates, which increase power dissipation. Further, a wave-pipelined circuit has two-sided operating speed limits which are determined by both the shortest and the longest paths. Inequality between those paths, particularly under process variations, limits the operable speed range, for both high and low ends of the speed, of a wave-pipelined circuit. Note that a flip-flop based circuit is bounded by only the longest path delay, which limits the highest operating speed, not the lowest.

This thesis proposes a hybrid pipelining method called *HyPipe*, which integrates the two methods, conventional pipelining and wave pipelining [45][47][48]. Conventional pipelining divides the circuit into smaller blocks, and then wave pipelining is applied to each sub-block. To balance delays of each sub-block, each sub-block is composed of only one gate type. The proposed approach leads to a better delay balance than the use of heterogeneous types of gates to offer potentially greater performance for wave pipelining. Another requirement of wave pipelining to achieve high speed is short rise and fall times of signals. We noticed that wide-input logic gates exhibit long worst-case delays and large delay variations. To address the problem, we investigated several methods to decompose CMOS wide-NAND and wide-NOR gates into narrower gates. We showed that while the straightforward decomposition using alternating levels of NAND and NOR gates results in short latencies, it generally results in large delay variations. We proposed a new method for decomposition, in which the width ratio between the NMOS and the PMOS transistors is held constant. Our method reduces the delay variation by a factor of up to 1.64 for circuits consisting of only NAND gates or only NOR gates. Thus, it is appropriate for wave pipelining based on standard-cells or sea-of-gates.

For a proof of concept, we applied the proposed HyPipe method to adder and multiplier circuits based on fully complementary static CMOS gates [48]. We laid out a 4×4 multiplier and performed a post-layout SPICE simulation. The simulation results show that the multiplier achieves a throughput of 4.17 billion multiplications per second or a clock period of $2.52 \times FO4$ delays (which is the delay of an inverter driving four other identical inverters), and the speed is almost two times the speed of any existing multiplier available in open literature. As expected, our multiplier exhibits a high latency of 28 clock cycles and consumes a relatively large power of 76.2 mW. When the supply voltage is reduced to 1.2 V from its original 1.8 V, the power consumption is reduced to 18.2 mW at a lower throughput of 2.3 billion multiplications per second.

The rest of this thesis is organized as follows. Chapter 2 provides preliminaries on conventional pipelining and wave pipelining. It discusses metrics used for quantifying the speed and energy efficiency of a CMOS VLSI circuit and presents the logical effort method for circuit design proposed by Sutherland and Sproull [49][50]. Chapter 3 describes the proposed HyPipe method and presents the design of a circuit based on HyPipe for proof of concept. In addition, it also analyzes several methods to decompose wide-NAND and wide-NOR gates into a network of narrower gates and attempts to find a decomposition method to result in the least delay variations. Chapter 4 presents and discusses simulation results. It compares the performance and power dissipation of HyPipe circuits with that of non-pipelined, conventional-pipelined, and wave-pipelined circuits. It also compares the performance attained by various wide-NAND/wide-NOR decomposition methods. Chapter 5 draws a few conclusions from our work.

Chapter 2

Background

This work describes a new pipelining approach and a high–speed multiplier design which serves as proof of concept. This chapter reviews existing pipelining methods and existing high-speed multiplier designs with and without pipelining. Additionally, it also introduces a method for designing fast circuits, namely the *Logical Effort* (LE) method, as we will employ its simplified delay model and parts of its terminology in the rest of this work.

This chapter is divided into five sections. Section 2.1 explains the metrics used for quantifying the performance and energy efficiency of a CMOS circuit. While this information is not required to understand pipelining, it justifies the use of those metrics in later chapters. Section 2.2 describes conventional pipelining and analyzes its performance theoretically. Section 2.3 describes wave pipelining. Section 2.4 reviews previous works on high-speed multipliers using both pipelined techniques and non-pipelined techniques. Finally, section 2.5 introduces the Logical Effort (LE) method.

The main aim of this work is to attain the highest speed possible. However, as many applications require low power dissipation, this work also analyzes the energy cost of the various techniques.

2.1. Metrics: Latency, Throughput, Energy-Delay Product, Energy-Delay^2 Product, $FO4$ Delay

We first define the performance metrics that account for both power and speed. The existence of such metrics is important since one design may be faster yet consumes more power than another, hence it is difficult to judge relative performance and energy efficiency. Comparison is even more difficult between different technologies, since the difference in speed may be due to technology instead of design.

When comparing different technologies, *normalization factors* (linear or otherwise) are necessary. For delay or speed comparisons, normalizing the delay to the $FO4$ *delay* of the technology is a common practice. The $FO4$ delay is defined as the delay exhibited by an inverter driving four identical inverters, itself being driven by another identical inverter driving four identical inverters [14]. Another commonly used metric, called the delay of a NAND2 gate with three loads, is approximately 25% higher than the $FO4$ delay [14]. We use the $FO4$ delay due to the simplicity of the simulations; an inverter has one input, so we only need to check the two possible input transitions of rising and falling. This work uses a ring oscillator to find the $FO4$ delay of a technology. Further, the $FO4$ delay is taken as the average between rise and fall delays.

If the $FO4$ delay for a particular technology is not given and cannot be found through simulations, it could be estimated as given in [15], as:

$$FO4 \text{ delay} = 360 \text{ ps} \times \text{minimum gate length in } \mu\text{m} \qquad (2.1)$$

However, the above number was obtained for "typical" transistors. While "typical" parameters are adequate for predicting the results of postlayout simulations, the performance of the circuits, if actually fabricated, will also depend on the process variations, or more specifically, on the slowest transistors, for which the conversion factor could be as high as 500 ps/μm [15]. Our own simulations of a 0.18 μm technology gave *typical* $FO4$ delays of 82 – 95 ps, therefore conversion factors of 455 – 527 ps/μm. On the other hand, there is a 1 μm technology [17] with an $FO4$ delay of 264 ps. Due to this uncertainty, for the circuits which were actually fabricated, we feel that a more prudent estimate probably is using the median of the above numbers, or approximately:

$$FO4 \text{ delay} = 450 \text{ ps} \times \text{minimum gate length in } \mu\text{m} \qquad (2.2)$$

We will use (2.2) for fabricated circuits (such as [21]), while for simulated circuits (such as [35]) we will use (2.1), unless it is explicitly stated that the slowest transistor model is used.

In addition to normalization factors, it is also necessary to determine the appropriate *metrics* to compare speed and energy efficiency. In general, speed can be measured according to throughput or latency. Since the focus of this work is on attaining high throughput, we assume that throughput or equivalent measures (such as clock period or inter-input period) is the primary measure of speed or delay unless stated otherwise. Delays are normalized to $FO4$ delays to compare different technologies.

For energy efficiency, the most useful metric is the power consumption at just high enough a speed and V_{DD} to perform the task reliably. This metric assumes that there is a V_{DD} value which allows the circuit to operate with sufficient speed with some safety margin. However, this metric is inconvenient and impractical, since it requires exhaustive simulations at various V_{DD} levels. Further, different applications may have different speed requirements, and therefore, the corresponding metrics will vary even for the same circuit. Therefore, a meaningful metric, which does not vary for a given circuit, is needed.

The most common metric for energy efficiency is the energy-delay product (EDP) [15], or the product of energy per switching and the delay; the lower it is, the better. For a pipelined circuit, we can use the product of energy dissipated per clock cycle and the shortest clock period allowable. However, this metric depends on the supply voltage [31]. Another metric is the energy-delay^2 product (ED^2P), which is independent of V_{DD} for a circuit under any V_{DD} > $\max(V_{tn}, |V_{tp}|)$ assuming a long-channel CMOS model [31]. However, this product is not constant for wave-pipelined circuits, as is later shown in Chapter 4. For lack of better metrics, both EDP and ED^2P are used. Note that simply using power consumption as the base for comparison is unfair, even if we run the circuits at the same clock speed. If one circuit is capable of faster operation, its power consumption can be reduced by simply reducing its V_{DD} until it is just fast enough to operate at that particular clock speed [28]. Thus, a faster circuit may also offer greater power saving as its supply voltage can be reduced farther. Thus, the use of power consumption as a measure of energy efficiency is fair only if supply voltage is adjusted; however, such a case is usually not reported. Likewise, power-delay product (PDP), which is essentially equivalent to switching energy, is a poor indicator of energy efficiency. It can be drastically lowered by simply lowering V_{DD} until the circuit barely works, but the circuit will be unacceptably slow [15].

For power comparison across different technologies, no simple normalization is available. However, we assume a first-order constant field scaling model [56]. Assuming that the feature size of a technology is scaled by a factor of α, the delay, dynamic power at constant frequency, dynamic power at the highest attainable frequency, and PDP, and by extension switching energy, EDP and ED^2P, scale by the following factors:

$$D \propto 1/\alpha \tag{2.3}$$

$$P\big|_{f=const} \propto 1/\alpha^3 \tag{2.4}$$

$$P\big|_{f=f\max} \propto 1/\alpha^2 \tag{2.5}$$

$$E \propto 1/\alpha^3 \tag{2.6}$$

$$PDP \propto 1/\alpha^3 \tag{2.7}$$

$$EDP \propto 1/\alpha^4 \tag{2.8}$$

$$ED^2P \propto 1/\alpha^5 \tag{2.9}$$

Therefore, in absence of a PDP or $FO4$ delay of a technology, the above relations are used. Whenever they are given, however, EDP and ED^2P could be calculated as follows:

$$EDP \propto PDP \times FO4\ delay \tag{2.10}$$

$$ED^2P \propto EDP \times FO4\ delay \tag{2.11}$$

For comparisons involving different technologies, the EDP and ED^2P are scaled to give fair comparison. We build the example circuit in TSMC’s 0.18 μm technology, so all numbers are scaled to this technology.

In summary, the scaled EDP and the ED^2P are the energy metrics and the $FO4$-normalized delay is the delay metric.

2.2. Conventional Pipelining

A conventional pipelined system follows the structure in Figure 2.1. The definitions for $T_{CK\rightarrow Q}$ and T_{SU} (as belonging to successive register stages instead of as belonging to the same register) is neither arbitrary nor inconsequential. As they belong to different registers, a rising (falling) $T_{CK\rightarrow Q}$ is not associated with a rising (falling) T_{SU}. Therefore, the worst-case (best-case) path delay D_{MAX} (D_{MIN}) must be found by summing the worst-case (best-case) values of $T_{CK\rightarrow Q}$ and T_{SU}, which may represent opposite directions of signal transitions. The same reasoning and definitions will be followed in later discussions of wave pipelining and HyPipe. If $T_{CK\rightarrow Q}$ and T_{SU} were defined as belonging to the same register, they would transition in the same direction, and this may not necessarily give the actual D_{MAX} (D_{MIN}).

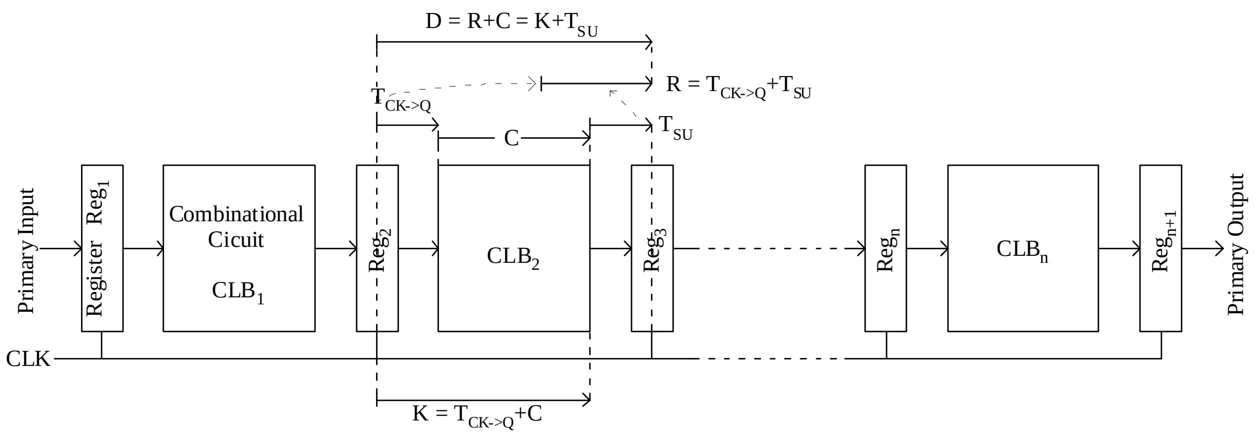


Figure 2.1. Structure of a Conventional Pipelined Circuit and Delay Definitions. Here R denotes register delays, C denotes combinational (CLB) delays, K denotes the delay from clock to and D the total path length

Given that the total delay between the output of a register and the input of the next register stage is:

$$D = C + R \tag{2.12}$$

Henceforth, the subscripts *MAX* and *MIN* signify the maximum and minimum values of a parameter displayed by a circuit. For instance, C_{MAX} denotes the longest combinational delay in a circuit. At any case, the shortest clock period is bounded from below by:

$$T_{CK} \geq D_{MAX} + (2 \times S_{CK}) \tag{2.13}$$

Here S_{CK} denotes the largest unintentional clock skew between two consecutive register stages. Ignoring clock skew, the minimum clock period is simply D_{MAX}.

The EDP and ED^2P of the pipelined circuit can be estimated as follows. The total delay of the combinational circuit, if not pipelined at all, is denoted by C_{NP}. Let us ignore leakage or static energy, and assume that a single operation takes E_C amount of energy. Hence, the EDP and ED^2P of the circuit are $E_C \cdot C_{NP}$ and $E_C \cdot C_{NP}^2$, respectively. Therefore, the lower bound of worst-case combinational delay between two registers for an *n*-level pipelined circuit (hence with *n*+1 register levels) is:

$$C_{MAX} \geq \frac{C_{NP}}{n} \tag{2.14}$$

therefore:

$$D_{MAX} \geq R_{MAX} + \frac{C_{NP}}{n} \tag{2.15}$$

Let us assume that one register stage is associated with energy dissipation of E_R. It follows that the total register energy, which includes both clock energy and internal energy dissipations of each register, is $(n+1) \cdot E_R$. This approximation does not account for the possibility differing register widths for each stage; hence, it should serve only as an approximation.

Assuming that a new input pattern is ready every clock cycle, the lowest average energy needed to process an input pattern could be approximated as:

$$E_{MIN} = E_C + (n+1) \cdot E_R \tag{2.16}$$

Here E_C stands for the amount of energy needed by the nonpipelined combinational circuit to process one input pattern. Taking throughput as our sole aim, we can use the minimum attainable clock period as our sole measure of delay. Therefore, the lowest EDP and ED^2P attainable are:

$$EDP_{MIN} = \left[E_C + (n+1) \cdot E_R\right] \cdot \left(R_{MAX} + \frac{C_{NP}}{n}\right) \tag{2.17}$$

$$ED^2P_{MIN} = \left[E_C + (n+1) \cdot E_R\right] \cdot \left(R_{MAX} + \frac{C_{NP}}{n}\right)^2 \tag{2.18}$$

As is apparent, the use of conventional pipelining has the potential for greatly decreasing the inter-input period, thus increasing a circuit's throughput. However, it could also increase the per-input energy greatly, and therefore if very high throughput (e.g., in GHz range) is desired, the resulting EDP and ED^2P could be prohibitively high.

In the next section, we analyze the wave-pipelined circuit and compare it with the conventional pipelined circuit.

2.3. Wave Pipelining

The basic wave-pipelined circuit is shown in Figure 2.2. At first glance, it looks like a nonpipelined circuit; however, it differs in that the CLB contains the signals caused by several successive input patterns, instead of only those caused by a single input pattern. Unlike the conventional pipelined circuit, a wave pipelined circuit synchronizes and partitions the signals using the delays of combinational circuits. It is required that $f_{CLKIN} = f_{CLKOUT}$, but they may be separated by a constant skew. Figure 2.3 shows the comparison between conventional- and wave-pipelined circuits when they are operating. For the conventional pipelined circuit, two consecutive register stages are separated by just one clock cycle; therefore, any input received by the first register in a clock cycle will be received by the second register in the very next clock cycle. However, in a wave pipelined circuit, the registers could be separated by several clock cycles; the dashed lines in Figure 2.3 shows the borders between the inputs from consecutive clock cycles as they propagate through the combinational block. As the registers are separated by three clock cycles, the wave-pipelined circuit in Figure 2.3 (a) is said to have a *wave number* N of 3. In contrast, the conventional pipelined circuit in Figure 2.3 (b) also contains the signals generated by three consecutive inputs, but only one in each CLB.

Wave pipelining is not pipelining in the ordinary sense of the word, however, since the above three datasets are contained by the same CLB, and not separated by registers.

Mathematically, N does not have to be an integer; *fractional* N could be attained by permanently inserting a skew between the clock signals of the input and output registers, hence using different clocks. If, for instance, k inputs are contained within the CLB and the output clock lags by $m \cdot T_{CLK}$ behind the input clock, with m being a fractional number, we could also say that $N = k - 1 + m$ in mathematical analysis. Note, however, that in some of the literature published, such as [4], such a situation is referred to as $N = k$ with skew.

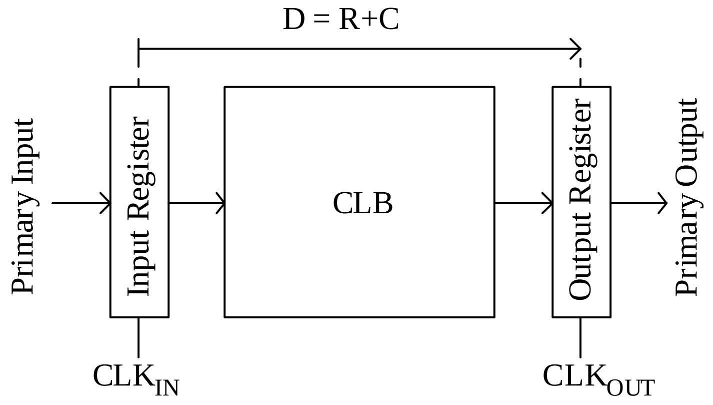


Figure 2.2. Basic Structure of a Wave-Pipelined Circuit

Excellent tutorials of wave pipelining are available in [4] and [57]. However, here we discuss several key points, a few of which were not mentioned in the references.

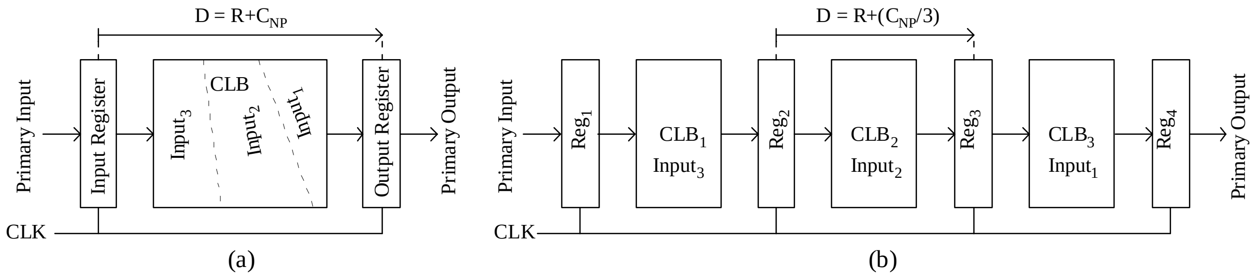


Figure 2.3. Comparison between (a) a Wave-Pipelined Circuit with $N = 3$ and (b) a Conventional Pipelined Circuit with Three Combinational Stages.

Unlike the conventional pipelined circuit, whose clock frequency is bounded from below by the *length* of the critical path, the minimum clock period of a wave-pipelined circuit is instead bounded by the *difference* between the longest and shortest path between inputs and outputs. Denoting register hold times as T_H and rise/fall times of signals inside the CLB, including primary inputs and outputs of the CLB, as T_{rf}, the lower limit for clock period is given by the following formula:

$$T_{CK} \geq (K_{MAX} - K_{MIN}) + T_{H,MAX} + T_{SU,MAX} + T_{rf,MAX} + (2 \times S_{CK}) \qquad (2.19)$$

Therefore, unlike the clock period of conventional pipelines, the clock period of wave pipelines is also limited from below by the highest hold time and signal rise and fall times. However, if the sum of worst-case hold time and rise/fall times is less than the shortest clock-to-CLB-output delay K_{MIN}, as we shall see, wave pipelines could attain shorter clock periods than conventional pipelines.

As before, we ignore clock skew in the rest of this work. Further, because we use master-slave flip-flops, whose hold times are zero or negative [44], we conservatively assume that T_H = 0. Also, in various sources [4][5][21][40], rise and fall times T_r, T_f were ignored, because generally they targeted the cases where $D_{MAX} - D_{MIN} >> T_{rf}$. However, for the clock speeds targeted in this work, rise and fall times cannot be ignored.

At any rate, if rise and fall times are negligible, the above formula reduces to:

$$T_{CK} \geq (K_{MAX} - K_{MIN}) + T_{SU,MAX} \qquad (2.20)$$

Or, in a different form:

$$T_{CK} \geq D_{MAX} - K_{MIN} \qquad (2.21)$$

For internal nodes which do not drive any flip-flops, the constraint becomes:

$$T_{CK} \geq \left(K'_{MAX} - K'_{MIN}\right) + T_{MS} + T_{rf,MAX} + S_{CK} \tag{2.22}$$

Where K' denotes the total path length from the last clock to that internal node inside the circuit, and T_{MS} denotes the minimum duration for which the logic value in that node must remain unchanged for the next logic stage to function correctly.

The inequalities (2.19) – (2.22) must be satisfied by all internal nodes of the circuits in addition to the end nodes. However, here, we focus on the output nodes, under the assumption that the output nodes tend to exhibit the largest delay variation, and therefore determines T_{CK}. Additionally, as later seen, the actual operable frequency for wave-pipelined circuit is determined by the clock-to-output node delay and delay variations.

From (2.21), it is obvious that a wave-pipelined circuit can attain much higher speeds than a conventional-pipelined circuit in principle. In theory, if D_{MIN} and D_{MAX} are equal, and the flip-flop has a negligible T_{SU}, infinitely high speed can be attained. However, this is impossible, because the clock period cannot be less than the rise/fall times of the slowest rising/falling signals, nor can it be shorter than the minimum toggle time of the flip-flop or the maximum delay of the slowest logic gate.

$$T_{CK} \geq \frac{1}{2} \cdot \left(T_{Q\uparrow MAX} + T_{Q\downarrow MAX}\right)_{all\_ff} \tag{2.23}$$

$$T_{CK} \geq \max\left(T_{pLH}, T_{pHL}\right)_{all\_gates} \tag{2.24}$$

Equation (2.23) appears to not have been considered in other works so far; however, it may be conjectured that the right hand side of (2.24) may be represented by T_{MS} in (2.22).

To avoid confusion, a CMOS logic gate is defined as a logic circuit which contains just one node where the source or drain of a PMOS transistor electrically meets the source or drain of an NMOS transistor. The highest N attainable is:

$$N_{MAX} = \left\lceil \frac{D_{MAX}}{D_{MAX} - D_{MIN}} \right\rceil \tag{2.25}$$

Even with good delay balance and with the inequalities (2.19) – (2.24) satisfied, a high N – and hence, a high speed – may be unattainable in practice. Unless the delay balance is perfect, a higher N narrows the frequency range for correct operation.

$$\frac{D_{MAX}}{N} \le T_{CK} \le \frac{D_{MIN}}{N-1} \tag{2.26}$$

To see why this is the case, Figure 2.4 shows the time window for sampling data. As apparent, data applied at clock cycle k should be available by NT_{CK} later for sampling at the output register. Also, at that time, the data applied at the clock cycle $k+1$ must not have arrived. Those two requirements are expressed as:

$$D_{MAX} \le N \cdot T_{CK}$$

$$D_{MIN} \ge (N-1) \cdot T_{CK}$$

which combine into equation (2.26).

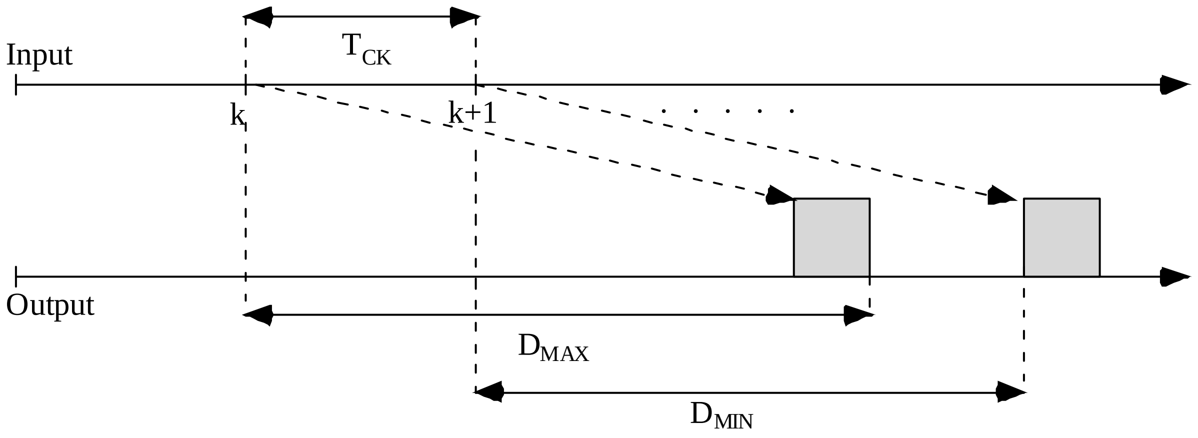


Figure 2.4. Timing Window Requirement for Wave-Pipelined Sampling of Data

Process variations may decrease D_{MIN} and increase D_{MAX} compared to values obtained in postlayout simulations. Accounting for process variations, equation (2.26) indicates that N is likely limited to about 3 [7]. To estimate the wave number needed to equal the performance of a conventional-pipelined circuit with n stages, we assume the system runs at its highest attainable speed. In that case, we could ignore the upper limit of T_{CK} for a given N and focus on the lower limit instead. Ignoring clock skew and hold times,

$$T_{CK} \ge \frac{R_{MAX} + C_{NP}}{N} \tag{2.27}$$

To derive the minimum N needed, we replace the greater-than/equal signs in (2.15) and (2.27), with equalities; hence, to allow the clock period for wave-pipelined circuits to be no more than the conventional pipelined case, the following condition must hold:

$$\frac{R_{MAX} + C_{NP}}{N} \le R_{MAX} + \frac{C_{NP}}{n}$$

$$N \ge 1 + \frac{n-1}{n} \frac{C_{NP}}{R_{MAX} + C_{NP}/n} \tag{2.28}$$

As a numerical example, if a combinational circuit with C_{NP} = 3 ns is split into a 5-stage pipeline using registers with R_{MAX} = 200 ps (for a 1.25 GHz clock), a wave number of at least 4 is needed for the same speed. This situation highlights a problem: since N frequently cannot exceed 3, a wave pipelined circuit is not guaranteed to attain higher throughput than a conventionally pipelined circuit.

The delay and energy efficiency of the circuit can be computed as follows. We assume that the per-pattern energy consumption of the CLB is $E_C^W \ge E_C$ due to the extra energy from the delay elements needed to equalize the delay paths [4][10][21][27]. We also assume that the maximum delay does not increase, since the delay equalization is performed by extending the shortest paths. With two register levels, the per-input-pattern energy consumption would be approximately $E_C^W + 2E_R$. Further, the best-case EDP and ED^2P could be estimated as:

$$EDP_{MIN} = \frac{\left(E_C^W + 2E_R\right) \cdot \left(R_{MAX} + C_{NP}\right)}{N} \tag{2.29}$$

$$ED^2P_{MIN} = \frac{\left(E_C^W + 2E_R\right) \cdot \left(R_{MAX} + C_{NP}\right)^2}{N^2} \tag{2.30}$$

Wave-pipelined circuits could be more energy efficient than nonpipelined circuits (whose EDP and ED^2P are, again, $E_C \cdot C_{NP}$ and $E_C \cdot C_{NP}^2$, respectively), since most likely $N > 1$. Wave-pipelined circuit could also be more energy efficient than conventional pipelined circuit, if it attained high enough N without increasing E_C^W excessively. Likewise, attaining high N would allow the circuit to attain very high throughputs.

However, delay equalization is far from trivial for large circuits, especially considering process variations. If we limit the operation to low N, ignoring register delays, the throughput of the wave-pipelined circuit could actually be lower than that of a deeply pipelined circuit. As an example, we could simply attain high throughputs for a 32-bit multiplier by using *bit-level* conventional pipelining, where we use an array multiplier and put a register after every full adder. Assuming that the delay of a register is similar with the delay of a full adder, a comparable throughput using wave pipelining could only be attained for $N > 21$. The required N could be lowered to 4 if we use a tree multiplier, which may need only 6 adder levels. However, even $N = 4$ may be excessive.

2.4. High Speed Multipliers: Literature Survey and Analysis

This section presents a survey of some of the fastest multipliers in the open literature. A multiplier is taken as a testbench circuit for the HyPipe high-speed design technique. This section compares the performance and power dissipations of wave pipelined, conventional pipelined, and nonpipelined designs. Later, we will compare the performance of *HyPipe*.

Table 2.1 lists the highest-throughput multipliers thus far; only those with $T_{CK} < 20 \times FO4$ delays are shown. The multipliers are ranked based on the shortest clock cycle (*the column in* **boldface**), normalized to their $FO4$ delays as calculated in (2.1) or (2.2).

Table 2.1. Clock Periods of the Highest-Throughput Multipliers Surveyed

Ref.	Year	Pipelines	$T_{CK,MIN}$ (in FO4 delays)	$f_{CLK,MAX}$ (MHz)	Feature Size (μm)	Estimated FO4 Delay (ps)	Input Word Length
[22]	1996	conventional	**4.61**	482.5	1	450	varies[1]/4
[40]	2001	wave	**4.92**	1612.9	0.35	126	8
[17]	2001	conventional	**5.3**	715	1	264	8
[10]	1995	wave	**5.56**	625	0.8	288	8
[21]	1994	wave	**6.35**	350	1	450	16
[55]	2000	conventional	**7.41**	625	0.6	216	8
[7]	2002	conventional	**7.94**	1000	0.35	126	8
[1]	1995	conventional	**15.38**	50	1.2	1300	16
[19]	2002	conventional	**15.7**	1538.5	0.13	41.4	54
[35]	2001	conventional	**17.86**	444.4	0.35	126	32
[30]	1996	conventional	**19.44**	285.7	0.5	180	56

Note: 1. Design in [22] accepts varying input widths.

Although all the three wave-pipelined multipliers rank among the top five performers in terms of throughput, a conventional pipelined multiplier attains the highest throughput. Therefore, it is possible for a fine-grained conventional pipelined circuit to surpass the throughput of a pure wave-pipelined circuit.

For the 3.08 GHz Pentium IV [19], the actual maximum throughput of the multiplier is not known. The throughput (and also, later, the latency) is estimated from the reported speed of the FPU, which is half the clock frequency of the integer unit, and from the latency of the FPMUL instruction, which is 7 clock cycles. The actual T_{CK} and latency of the multiplier itself is probably shorter.

Table 2.2. Latencies of the Various Multipliers Surveyed

Ref.	**Year**	**Pipeline**	**Input Size (bits)**	**Feature Size (μm)**	**Estimated FO4 delay (ps)**	**Latency (ns)**	**Latency (FO4)**	**Normalized Latency (FO4)**
[34]	1996	conv.	24	1	450	10.5	23.33	**10.18**
[12]	1998	conv.	54	0.25	90	2.7	30	**10.43**
[58]	1990	none	16	0.5	180	3.8	21.11	**10.56**
[37]	1993	conv.	16	1	450	11	24.44	**12.22**
[21]	1994	wave	16	1	450	11.4	25.4	**12.7**
[35]	2001	conv.	32	0.35	126	4.5	35.71	**14.29**
[25]	2001	unkn.	64	0.25	90	4	44.44	**14.81**
[11]	1997	none	54	0.25	90	4.1	45.56	**15.83**
[20]	2001	conv.	54	0.18	65	3.16	48.62	**16.9**
[29]	1996	none	54	0.5	180	8.8	48.89	**16.99**
[33]	1995	unkn.	54	0.25	90	4.4	48.89	**16.99**
[19]	2002	conv.	54	0.13	41.4	2.27	54.9	**19.08**
[40]	2001	wave	8	0.35	126	3.9	30.95	**20.63**
[24]	2002	none	54	0.25	90	6.9	76.67	**26.64**
[23]	2001	none	16	0.25	90	5.14	57.11	**28.56**
[22]	1996	conv.	4	1	450	13.47	29.95	**29.95**
[30]	1996	conv.	56	0.5	180	15.8	87.78	**30.23**
[39]	2002	conv.	59	0.13	41.4	4	96.62	**32.85**
[17]	2001	conv.	8	1	264	15.3	57.95	**38.64**
[7]	2002	conv.	8	0.35	126	15	119.05	**59.52**
[55]	2000	conv.	8	0.6	216	27.2	125.93	**62.96**
[1]	1995	conv.	16	1.2	1300	400	307.69	**153.85**

Note: *Pipeline types:* unkn. = unknown, conv. = conventional pipeline, wave = wave pipeline. The design in [22] accepts variable word width. If the size is not 4-bit×4-bit, its latency D is normalized further according to $D \propto m + n$ for array multipliers and $D \propto \log_2(m \times n)$ for tree multipliers

Table 2.2 shows the latency of the multipliers. For the wider multipliers, the latencies are normalized to the latencies for 4 bits. In general, the latency for array and tree multipliers depend linearly and logarithmically, respectively, on word width [18][41][56]. We assume that the latency of *m*-bit×*n*-bitarray multipliers is proportional to (*m*+*n*), whereas the latency of tree multipliers is proportional to $\log_2(m \times n)$. It appears that the multipliers with the highest throughputs generally do not have the shortest latency. Likewise, the multipliers which exhibit the best latencies are not necessarily the ones with the best throughputs as well. Among the five highest-throughput multipliers, only the 5th-highest throughput one even ranks in the top five in terms of latency, therefore confirming the general tradeoff between throughput and latency.

Table 2.3 gives both the throughputs and the latencies of the highest-latency multipliers already listed in Table 2.1. The poor latency of highest-throughput multipliers is again apparent; the three highest-throughput multipliers all rank in the bottom half in terms of latency.

Table 2.3. Latencies and Latency Rankings of Highest-throughput Multipliers

Ref.	Year	Pipelines	Input Word Length	$f_{CLK,MAX}$ (MHz)	$T_{CK,MIN}$ (in FO4 delays)	Normalized Latency (FO4)	Latency Rank
[22]	1996	conventional	varies[2]/4	482.5	**4.61**	**29.97**	16
[40]	2001	wave	8	1612.9	**4.92**	**20.63**	13
[17]	2001	conventional	8	715	**5.3**	**38.64**	19
[10]	1995	wave	8	625	**5.56**	unknown	unknown
[21]	1994	wave	16	350	**6.35**	**12.7**	5
[55]	2000	conventional	8	625	**7.41**	**62.96**	21
[7]	2002	conventional	8	1000	**7.94**	**59.52**	20
[1]	1995	conventional	16	50	**15.38**	**153.85**	22
[19]	2002	conventional	54	1538.5	**15.7**	**19.08**	12
[35]	2001	conventional	32	444.4	**17.86**	**14.29**	6
[30]	1996	conventional	56	285.7	**19.44**	**30.23**	17

Notes:

1. If the size is not 4-bit×4-bit, its latency D is normalized further according to $D \propto m+n$ for array multipliers and $D \propto \log_2(m \times n)$ for tree multipliers.
2. Design in [22] accepts varying input widths.

Finally, Table 2.4 attempts to estimate the EDP and ED^2P of the multipliers, normalized to 4-bit×4-bit and 0.18 μm technology. The data is sorted according to the ED^2P, which is theoretically independent of V_{DD} for long-channel transistors [30]. Since most of those multipliers were reported in different technologies, the estimates are made as follows. First, it is assumed that if the multiplier is "normalized" to 4-bit for pipelined multipliers, their throughput and / or latency change as in the notes. Second, it is assumed that the throughput in $FO4$ delays remains the same if scaled down (or up) to 0.18 μm. The energy-delay product is then estimated based on the estimated highest clock speed attainable at 0.18 μm, 4-bit×4-bit. Here the clock frequencies quoted are the clock frequencies at which the power measurements are actually performed in the references; they are not necessarily the highest clock speeds attainable.

Table 2.4. Energy Efficiencies and Power Dissipations of the Multipliers Surveyed

Ref.	**Year**	**Type / Pip'd**	**Max Clock Speed (MHz)**	**ED^2P, $\times10^{-30}$ J.s.**	**EDP, $\times10^{-21}$ J.s.**	**Power (mW)**	**At Clock Speed (MHz)**	**Input Word Length**	**Feature Size (μm)**
[8]	2003	Tree,	1200	**0.04**	0.09	16.81	1000	13	0.35
[22]	1996	Serial, Y	482.5	**0.12**	0.31	69	482.5	$4^{(5)}$	1
[55]	2000	Array, Y	625	**0.27**	0.57	52.4	300	8	0.6
[35]	2001	Tree, Y	1612.9	**0.35**	1.08	150	1500	8	0.35
[21]	1994	Tree, Y	350	**0.44**	0.85	1360	300	16	1
[7]	2002	Tree, Y	1000	**0.90**	1.76	100.52	1000	8	0.35
[29]	1996	Tree, N	114	**2.72**	1.94	540	100	54	0.5
[11]	1997	Tree, N	244	**7.84**	5.99	2.23	1	54	0.25
[24]	2002	Tree, N	145	**11.06**	5.01	111	100	54	0.25
[30]	1996	Tree, Y	285.7	**13.49**	10.71	5.1	1	56×8	0.5

Notes:

1. EDP and ED^2P are normalized to the estimated values for an 0.18 μm implementation, according to formulae (2.3)-(2.9)
2. If the size is not 4-bit×4-bit, energy per operation is normalized down further to 4-bit×4-bit by assuming that for an *m*×*n*-bit multiplier, $E \propto m \times n$
3. If the size is not 4-bit×4-bit and not pipelined, its clock period D is normalized further according to $D \propto m + n$ for array multipliers and $D \propto \log_2(m \times n)$ for tree multipliers
4. [31] multiplies a 32-bit pixel data with a 32-bit FP, using a 56×8 multiplier (Booth recoded, possibly tree topology)
5. Design in [22] accepts varying input widths.

For multipliers other than the one listed in Table 2.4, no data regarding power is available.

Many assumptions are employed here, and the numbers in Table 2.4 are our best attempt at a reliable comparison.

2.5. Logical Effort: A Technique for Circuit Analysis and High-Speed CMOS Design

The latency of a circuit depends partly on the sizes of its transistors. As a first order approximation, resizing all the transistor widths inside a chip by the same factor will not change its delay. Resizing will give benefit in speed only if done selectively. For the special case of an inverter chain consisting of inverters with identical rise and fall delays, driving a fixed load, and driven by a signal source with a fixed, nonzero output impedance, the solution for the sizing problem is well known [56].

For a more general logic circuit, a simplified method for delay calculation and transistor sizing is also known, namely the *Logical Effort* (LE) method [49][50]. This section introduces the parts of the LE method which are relevant for our work. The LE method is well developed and also includes methods for minimizing the delay of asymmetrical gates (logic gates with different rise and fall delays) and specialized circuit families such as domino and transistor-gate logic, or for finding the delay parameters for logical gates which do not exhibit a linear delay-to-load-capacitance relation. Such specific topics are not discussed here. Readers desiring to obtain more information may refer to [50] for an excellent in-depth treatment of this subject. Here we will introduce only the basic form of the LE method, in which each logic gate is assumed to have equal worst-case (greatest) rise and fall delays and output resistance.

The LE method is a quick-and-dirty method to design the fastest circuit possible. It focuses first on finding the optimum transistor size, and second, on finding the optimal number of stages inside a path. If there are alternative topologies for a path, this method could also be used for choosing among alternative topologies.

Both transistor sizing and number of stages in a path would indirectly affect load levels: if a logic gate is resized, the other gates which drive it will see change in load levels, and if the number of stages is changed, as we will later discover, the sizes may have to be changed. Further, changing the number of stages in a path may mean that we have to use a different logic gate, namely one with a different topology. Thus, this method requires a gate delay model which at the very least considers the effects of gate topology and load toward delay.

The problem of minimizing the delay of a chain of logical gate is viewed as the problem of minimizing the delay of an *amplifier* chain. The problem is reduced to the problem of properly distributing the gain of the amplifier among the stages. Here, the following terms are defined.

The term *Logical Effort*, in addition to denoting the name of the method, also means the ratio between the input capacitance of a logic gate to that of an inverter of the same worst-case delay. Alternatively, the logical effort can also be defined as the ratio of its output resistance to the output resistance of an inverter with the same input capacitance. In other words, this parameter indicates *how much worse this gate is at driving loads, compared with an inverter* with the same input capacitance. This parameter is denoted with g for a logic gate and G for a path or multi-gate circuit. For a path, the "input nodes" of that path are the primary inputs of the first stage, while the "output nodes" are the primary outputs.

For a logic gate, g depends only on the gate's topology; for a path, G depends only on the topology of the path and of the logic gates in the path. Thus, it is independent of the transistor sizes, and as we will later see, it is this size independence which makes this quantity useful in transistor size optimization. However, "size independence" does not mean that g remains unchanged if each transistor inside a logic gate is sized arbitrarily. Instead, it means that g remains unchanged if all transistors inside a logic gate are scaled by the same factor. The assumption that a gate has the same worst-case rise and fall delays and output resistance (hence the same rising and falling g) necessitates a specific width ratio between any two transistors inside each logic gate of a given type. For instance, if the proper width ratio (PMOS width vs. NMOS width) for an inverter is 2:1, for a 3-input NAND gate it will approximately be 2:3. The reason is that in the 3-input NAND gate, there are three NMOS transistors in series, thus the resistance of the NMOS would triple. On the other hand, since the PMOS are in parallel, in the worst case, if only one of them turns on, their resistance would remain unchanged. To equalize both, the NMOS has to be made three times as wide as in the inverter, so as to reduce its resistance by a factor of three.

We can define g for both a single pin and a bundle of pins; for a bundle of pins, we use the sum of input capacitance of all the pins in calculation. However, unless noted otherwise, in this work, we will always use per pin g values.

The term *electrical effort*, denoted with h for a logic gate and H for a path, denotes the ratio of load capacitance driven by a logic gate (or path) and the gate's input capacitance:

$$h \equiv \frac{C_L}{C_{in}} \tag{2.31}$$

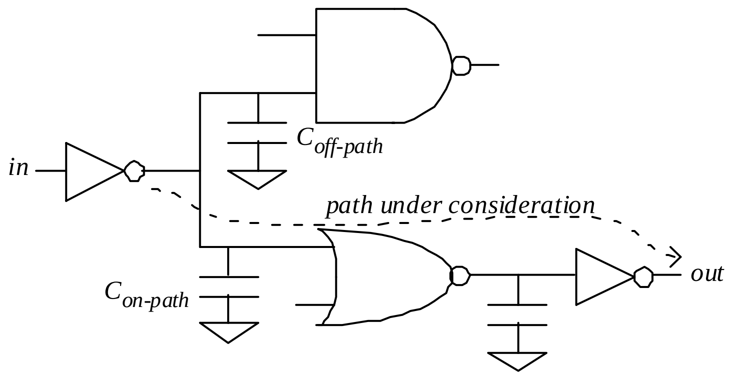


Figure 2.5. Circuit with Branch; only part of the capacitance in the branching node actually leads to the pertinent output (and therefore "useful")

Another term is *branching effort*, which is respectively denoted with b and B for a logic gate and a path. For the path which includes branches, as shown in Figure 2.5, b is defined as:

$$b \equiv \frac{C_{on-path} + C_{off-path}}{C_{on-path}} = \frac{C_{total}}{C_{useful}} \tag{2.32}$$

Thus, for a given output node of a logic gate driving a total load of C_{total}, of which C_{useful} actually leads to the point we are concerned with, b signifies how many times bigger C_{total} is than C_{useful}; it denotes the amount of load which has to be driven to drive a unit size of pertinent load. For a branchless circuit, b = 1. Also, b is normally calculated *at the input pin of a logic gate*; thus, b = 1 as well for all output pins. In this example, b is calculated at the input pin of the NOR gate in the second level.

For a path, the efforts are the product of the efforts along the path. Here, the uppercase-symbols G, B, H stand for logical, branching, and electrical efforts of a path.

$$G = \prod_{path} g_i \tag{2.33}$$

$$BH = \prod_{path} h_i \tag{2.34}$$

We could also define the parasitic delay of a path; here, it is the sum of all the parasitic delays along the path:

$$P = \sum_{path} p_i \tag{2.35}$$

Part of a logic gate's delay is caused by the presence of load; such a delay is called *effort delay*, denoted with f for a logic gate and F for a path, respectively:

$$f = gbh \tag{2.36}$$

$$F = GBH = \prod_{path} f \tag{2.37}$$

And the total delay, both for a logic gate and a path, is the sum of parasitic and effort delays.

$$t_{gate} = f + p \tag{2.38}$$

$$t_{path} = F + P \tag{2.39}$$

In [50], it is shown that for a path with n_s stages, the shortest worst-case delay possible is attained if the effort (or, alternatively, gain, to make an analogy with amplification problems) is evenly distributed along the path. Denoting the per-stage effort delay at which this shortest delay possible as $\hat{f}$, we have:

$$\hat{f} = F^{1/n_s} \tag{2.40}$$

If the logic gate which drives the path is fixed, and the load driven by the path is also fixed, then both G and H for the path are already fixed, regardless of the path's circuitry. Thus, we could use (2.40) to calculate $\hat{f}$. Since g and p depend only on a gate's topology, assuming fixed circuit topology and hence fixed b for each stage, all which could be adjusted for each stage is h. In turn, for fixed path and logic gate topology, h could only be adjusted by adjusting the sizes of the transistors inside the logic gates. Thus, this method gives us a way to size the transistors to attain best speed possible. This method is simple enough to be done by hand, although the resulting sizes may still need some fine-tuning with the help of simulation tools.

The LE delay model, upon which the delay minimization technique is based, is built on finding the per-pin input capacitance of a logical gate which is designed to exhibit the same worst-case output resistance as an inverter. The simplest delay model is based on the following simplifications:

1. Input capacitance is assumed to be comprised entirely of gate capacitance of transistors, which is assumed to be proportional to the width of the transistors; thus, wire capacitance and perimeter capacitances of the transistors are ignored;
2. The delays are modeled purely as RC delays;
3. The delay of logic gates are assumed to be independent of the slew rate (or rise and fall times) of its inputs;
4. Output capacitance is lumped at the output node of the logical gate, and is assumed to depend linearly on the total widths of the transistors; thus, the drain / source capacitance of other transistors are ignored;
5. The resistance of a transistor is assumed to be proportional to W/μ, where W is the transistor's width and μ is the charge mobility of that type of transistor.

The delay of a logic gate is expressed as:

$$t_{gate} = p + R \cdot C_L \tag{2.41}$$

where R is the output resistance of the logic gate and C_L is the load capacitance it drives.

By comparing equations (2.38) and (2.41), it may be seen that:

$$f = gh = p = R \cdot C_L \tag{2.42}$$

Here, b = 1 since, as mentioned before, it is calculated at the input pin of the next stage. Since $C_{out} = \left(C_L/C_{in}\right) \cdot C_{in}$, and $h \equiv C_L/C_{in}$, it follows that:

$$g = R \cdot C_{in} \tag{2.43}$$

Thus, g could be determined from the topology of a logic gate. It could be further noted that to simplify calculations, usually the p and g of a logic gate are normalized to that of an inverter.

As a calculation example, let us use the inverter in Figure 2.6. In this section, we use the simplest form of LE method, where all logic gates are assumed to be designed to exhibit equal worst-case rise and fall delays, using minimal-length transistors only. Here, the inverter is assumed to have NMOS width W_n = 1; thus, we could represent the falling resistance R_{HL} = $1/W_n$ = 1. The proper width and resistance units could be ignored, because we will normalize all other widths and resistances to these values. At any case, to attain the same rising resistance ($R_{LH} = R_{HL}$), we must have $W_p / \mu_p = W_n / \mu_n$. Let us define:

$$\gamma \equiv \frac{\mu_n}{\mu_p} \tag{2.44}$$

Thus, γ denotes the charge mobility ratio. The width of the PMOS width could then be found as $W_p = \gamma$.

The input capacitance could be found as:

$$C_{in,INV} = k_{in} \cdot (W_{n,INV} + W_{p,INV}) = k_{in} \cdot (1+\gamma) \tag{2.45}$$

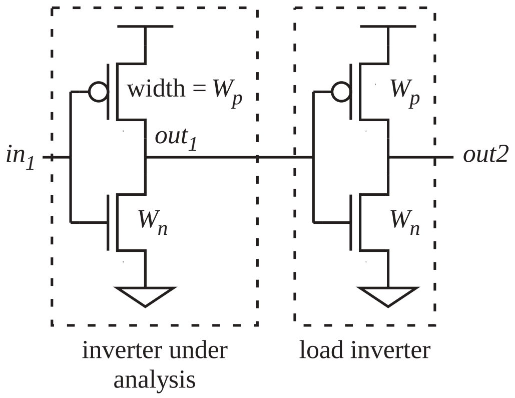


Figure 2.6. The Inverter Analyzed

All input capacitances will be normalized to this value; they are relative to the input capacitance of the inverter. Thus, in LE, we set:

$$k_{in} \equiv \frac{1}{1+\gamma} \tag{2.46}$$

Thus, $C_{in,INV} = 1$.

For instance, if it drives another inverter as in Figure 2.6, we would have $C_L = C_{in}$ of the second inverter. Thus, p is the delay exhibited by the first inverter in the absence of any load; it is also termed *parasitic delay*. It is also modeled as an RC delay:

$$p = R \cdot C_{out} \tag{2.47}$$

where C_{out} is the output capacitance of the inverter. As previously mentioned, it is modeled as lumped to the output pin, and proportional to the total sum of width of transistors attached to the output node; thus:

$$C_{out,INV} = k_{out} \cdot (W_{n,INV} + W_{p,INV}) = k_{out} \cdot (1+\gamma) \tag{2.48}$$

Therefore, for the inverter:

$$p_{inv} = k_{out} \cdot (1+\gamma) \tag{2.49}$$

In general, the value of k_{out} has to be determined individually for each process technology, whether by experiments, by simulations, or by hand calculations. If it is not known, a simple and reasonable approximation for hand calculation is:

$$k_{out} \approx k_{in} \tag{2.50}$$

Therefore, $p_{inv} \approx 1$. Thus, the output capacitance of an inverter is approximately the same as its input capacitance, and the delay of an unloaded inverter is approximately the same as the incremental delay caused by the presence of a load of the same size. This approximation will be used exclusively in the rest of this work, although the method followed should still be valid in other cases, if necessary adjustments are made.

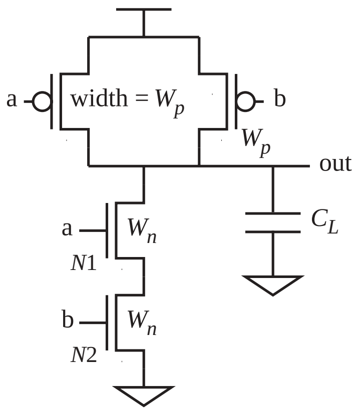


Figure 2.7. The 2-input NAND Gate Analyzed

As another example, for the 2-input NAND gate in Figure 2.7, assuming that the two NMOS transistors have the same width, and hence the same resistance:

$$R_{HL,NAND2} = R_{N1} + R_{N2} = 2R_{N1} \tag{2.51}$$

However, since we expect that $R_{HL,NAND2} = R_{HL,INV} = 1/W_n$, we have:

$$2R_{N1} = \frac{2}{W_{n,NAND2}} = \frac{1}{W_{n,INV}}$$

$$W_{n,NAND2} = 2W_{n,INV} \tag{2.52}$$

Therefore, to attain the same worst case (= highest) output resistance as the inverter we have to use NMOS transistors twice as wide as what is used for the inverter because we have two NMOS in series here. However, for the PMOS transistors, $W_{p,NAND2} = W_{p,INV}$, since in the worst case, only one of the two PMOS transistors conducts, thus each transistor only needs to be the same width as the inverter's PMOS. It could be shown that as the result, at the worst case, $R_{LH,NAND2} = R_{HL,NAND2} = 1$.

Also:

$$C_{in,NAND2} = k_{in} \cdot \left(W_{n,NAND2} + W_{p,NAND2}\right) = \frac{2+\gamma}{1+\gamma} \tag{2.53}$$

$$C_{out,NAND2} = k_{out} \cdot \left(W_{n,NAND2} + 2W_{p,NAND2}\right) = k_{out} \cdot (2+2\gamma) = 2\,C_{out,INV} \tag{2.54}$$

$$p_{NAND2} = R_{NAND2} \cdot C_{out,NAND2} = 2\,p_{inv} \tag{2.55}$$

Table 2.5 shows the results for inverters, NAND, and NOR gates of n_i inputs, taken from pages 71 and 81 of [50].

As we have seen, the gate delay model is a simplified RC delay model. The LE method is particularly simple to use if the gates are designed to exhibit the same worst-case rising and worst-case falling delays (or "symmetrical delays"). The applications of the LE method to asymmetrical gates is more involved [50].

Table 2.5. Transistor Widths, Logical Efforts, and Parasitic Delays of n_i-input Logic Gates

	INV	**NAND**	**NOR**
W_p	$\frac{\gamma}{1+\gamma}$	$\frac{\gamma}{n_i+\gamma}$	$\frac{n_i\gamma}{1+n_i\gamma}$
W_n	$\frac{1}{1+\gamma}$	$\frac{n_i}{n_i+\gamma}$	$\frac{1}{1+n_i\gamma}$
g per pin	1	$\frac{n_i+\gamma}{1+\gamma}$	$\frac{1+n_i\gamma}{1+\gamma}$
p	p_{inv}	$n_i\,p_{inv}$	$n_i\,p_{inv}$

In Chapter 3, we will use an essentially identical RC delay model for analyzing the delays of logic gates and using them in wave pipelining. The main difference is that we will also calculate best-case delays.

Chapter 3

HyPipe: An Approach for Very High-Throughput Pipelined CMOS Circuits Design

In Chapter 2, conventional and wave pipelining were described. Their respective advantages and shortcomings were also described. Additionally, the method of Logical Effort (LE) was introduced. It was also asserted that while fine-grained conventional pipelining achieves high performance, it uses a large number of flip-flops and has correspondingly high power dissipation. Further, the resulting throughput is limited by register delays. To achieve high performance, a conventional pipelined circuit uses very fine-grained pipelining (e.g., gate-level), which requires a very large number of flip-flops, increasing both clock power and difficulty of clock routing.

Wave pipelining reduces the need for flip-flops; in principle, only input and output registers are needed for synchronization. Further, in theory, it potentially attains higher throughput than conventional pipelining. However, equalizing delays is not a trivial task, particularly for long paths, so delay inequalities result in low N and limited throughput. As the survey of fast multipliers indicates, the throughput of a fine-grained, conventionally pipelined circuit could surpass that of wave pipelined circuits. To attain the best throughput for wave-pipelined circuits, better delay balance than attainable in practice is required.

This chapter proposes a hybrid pipelining solution called HyPipe. The idea is to break a circuit into shorter paths using conventional pipelining, and then to equalize the delay paths of each now shorter path. Since the path to be equalized is shorter, a better delay balance should be attainable. Further, a technique to transform a class of nonpipelined circuits into HyPipe circuits through buffer and flip-flop insertion and a limited amount of transistor sizing is proposed and analyzed.

This chapter is organized as follows. Section 3.1 describes the proposed idea. Section 3.2 performs some theoretical analysis of decomposition methods for wide-NAND/wide-NOR gates, focusing on finding the best one for wave pipelining. Section 3.3 attempts to find the minimized two-level Boolean functions which result in the worst delay imbalance if implemented straightforwardly. Section 3.4 shows how the proposed approach is applied to the design of a multiplier as an example, and compares the performance of the resulting circuit with the performance of a handcrafted 4-bit×4-bit HyPipe multiplier. Section 3.5 describes the simulation procedure needed for HyPipe circuits, focusing on the design of the testbench for the multiplier as a specific example.

3.1. HyPipe Architecture Description and Theoretical Analysis

The basic structure of a HyPipe circuit is shown in Figure 3.1. As shown, the circuit is divided into shorter segments through conventional pipelining; each segment, or CLB (combinational logic block), is wave pipelined. Since every CLB is smaller than the full circuit, its paths are shorter, and it should be easier to equalize the paths. Further, every CLB could have a different N, and every register could have a different clock. For simplicity, we only analyze the case where all CLBs have the same N.

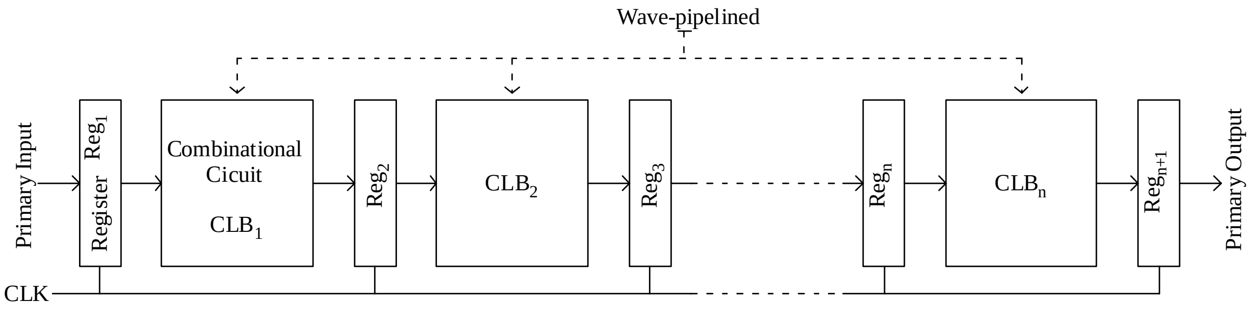


Figure 3.1. Basic Structure of a HyPipe Circuit

To analyze the circuit, let us assume that all CLBs exhibit equal D_{MIN} and D_{MAX}, and they have equal N as well. Assuming that the entire combinational circuit without pipelining has a delay of C_{NP} as in Section 2.2, from equation (2.15), we have:

$$D_{MAX} \geq R_{MAX} + \frac{C_{NP}}{n}$$

Therefore, for any given N we have:

$$T_{CK} \geq \frac{R_{MAX} + (C_{NP}/n)}{N} \tag{3.1}$$

Here, we assume that the circuit will operate at, or close to, its highest clock frequency for a given N. Therefore, only the upper limit of T_{CK} needs to be considered, not its lower limit. Obviously, for the same n, any $N > 1$ will result in improved throughput over conventional pipeline. Hence, compared with the pure wave-pipeline case of Section 2.3, we are less likely to require unattainably high wave numbers. However, note that the lower limits for clock periods as given by equations (2.20) – (2.23) also apply here.

To obtain energy efficiency figures of merit, let us denote the energy needed by all CLBs, after the adjustments, with $E_C^H \geq E_C$. Therefore, a lower bound for per-clock cycle energy consumption is:

$$E_{MIN} = E_C^H + (n+1) \cdot E_R \tag{3.2}$$

Therefore, the following formulae could be used for calculating lower bounds of the EDP and the ED^2P of the HyPipe circuit.

$$EDP_{MIN} = \left(E_C^H + (n+1) \cdot E_R\right) \cdot \frac{R_{MAX} + C_{NP}/n}{N} \tag{3.3}$$

$$ED^2P_{MIN} = \frac{\left(E + (n+1) \cdot E_R\right) \cdot \left(R_{MAX} + C_{NP}/n\right)^2}{N^2} \tag{3.4}$$

For large N, the energy efficiency of a HyPipe circuit could exceed that of a nonpipelined, conventional pipelined, or wave pipelined circuit, due to the high clock speed it potentially attains at any given V_{DD}.

Here, some words of caution must be given. Although the scheme depends on shortening the paths, there is an unknown lower limit on the path length for effective HyPipe operation. In the extreme case, each CLB consists of just one logic level, and it would be impossible to use wave pipelining. As the inequalities (2.19) – (2.24) show, the clock frequency cannot be made shorter than a gate delay, nor can it be made shorter than the signal rise/fall times, which are normally longer than gate delays. Therefore, gate delays and signal rise/fall times should be short. This observation precludes the use of large, complicated gates, and favors the use of small, simpler gates. As general rules of thumb,

– If CMOS is used, fully complementary static CMOS (FCCMOS) should be used for combinational circuits.

– No logic gates should have four or more inputs.

– Three-input gates should be used sparingly and limited to NAND3 and NOR3.

More complicated gates, such as majority gates, are best avoided, as they tend to lead into long rise/fall times. As an example, Figure 3.2 shows two implementations of XOR3 gates. The second implementation is preferred, even though it has several logic levels and may exhibit longer propagation delay. Since each gate has a shorter transistor chain (none has more than two NMOS or PMOS transistors in series), the resulting rise and fall times are likely to be much shorter. Note also that the simple-gate implementation uses a delay equalizer at the input, as needed for wave pipelining. The output is buffered, as a simple noninverting buffer usually shows shorter rise / fall times when driving heavy loads. If the load is light, the output buffer could be omitted. In addition, the simple-gate implementation exhibits more internal switching, thus, will likely consume much more power.

Because of the importance of short rise/fall times, it is also important to use a logic style which exhibits short rise/fall times. Since we will use CMOS technology, an obvious choice would be fully complementary CMOS (FCMOS), as it gives fast rise/fall times when loading is limited. Thus, unlike other works such as [10][27], we avoid novel or otherwise non-FCMOS logic styles, even though they often result in better delay [42].

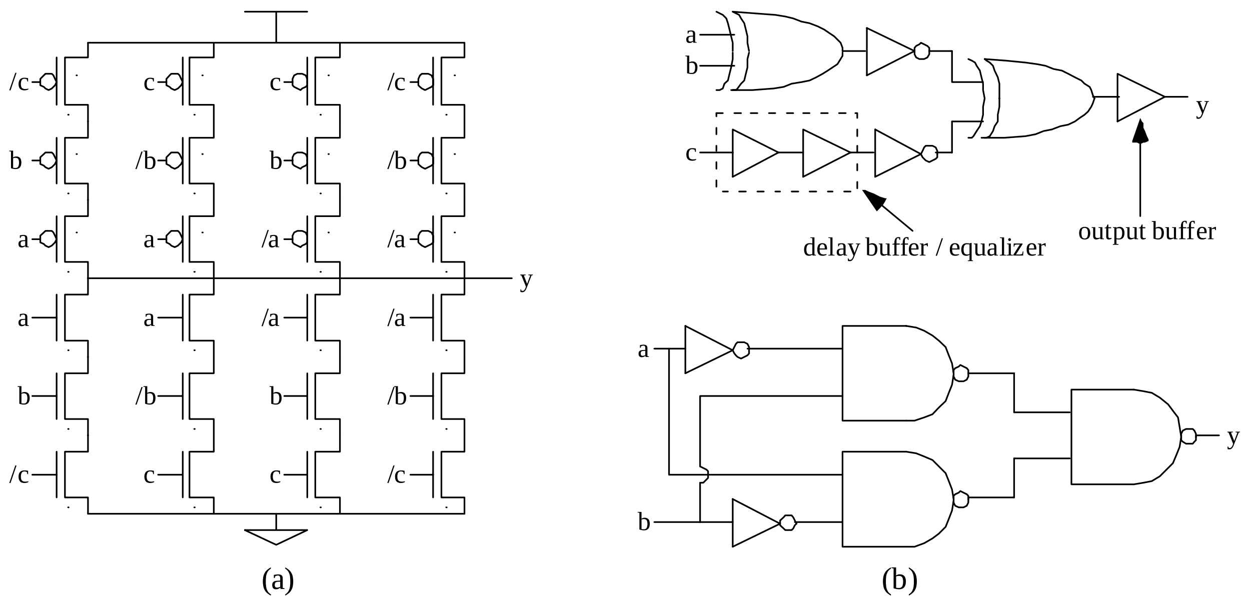


Figure 3.2. Two Possible Implementations of XOR3. (a) Complex Gate – inverters not shown. (b) Simple Gates: Top: overall circuit; Bottom: XOR2 used in the simple-gate implementation of XOR3. The implementation in (b) is preferred for HyPipe applications.

Further, wherever the highest speed attainable is desired, the rise and fall times of each gate, hence their *effort delay*, have to be limited; therefore, the load they drive must be limited as well. To achieve low load, some situations may force use of a combinational circuit with suboptimal D_{MAX}. If short rise/fall times and good delay balance are attained, suboptimal D_{MAX} could be compensated by the ability to attain high N values.

The HyPipe approach is best suited for arithmetic circuits, which could often be built from small, identical or similar building blocks. Arithmetic circuits may be handcrafted from a few building blocks, which can be tested exhaustively and replicated easily. The main problem remaining afterward would be placing and routing signals, which could be done with a placement-and-routing (PNR) tool, or even manually if the circuit has a regular structure.

Section 3.4 of this work gives an example of the application of the HyPipe concept toward the design of a multiplier.

3.2. Decomposition of Wide-NAND and Wide-NOR Gates: Theoretical Analysis

As mentioned in the previous section, one way to ensure short signal rise/fall times is by using logical gates with fewer inputs. However, various circuits potentially need wide logical gates. For instance, in address decoding, or in zero detection in arithmetic, frequently wide AND or NAND gates are required. It is possible to implement such a function with a single, wide NAND gate; such a solution will henceforth be termed “monolithic” solution in this work. However, while such a solution may result in least transistor count, it may not result in the fastest implementation. Also, it will result in the largest delay variation and rise/fall time possible, thus in very poor wave-pipelined performance. Therefore, a method to decompose such a wide gate into a circuit composed of logic gates of fewer inputs (“narrower”) is needed.

This section analyzes the decomposition of such a wide-NAND or wide-NOR gate, and attempts to find the best decomposition technique for wave pipelining. Instead of simply finding the implementation with least gate count or shortest delay, we will also attempt to find one with smallest delay difference. Note that although this section includes a theoretical analysis, the analysis is not expected to be quantitatively accurate. The analysis is only intended to be qualitatively accurate, to allow us to choose the best implementations. The implementation chosen will still have to be checked with a more accurate simulation tool, such as HSPICE, to determine its correct operating frequency.

In this section, we will also assume that the width ratio between NMOS and PMOS, k_w = W_p / W_n, will either be fixed, as apparently widely followed in the sea-of-gates and standard-cell design styles, or adjusted to attain approximately equal worst-case rise and fall delays.

The rest of this section will be organized as follows. In subsection 3.2.1, we will present the logic gates delay model to be followed in analyzing the alternatives. The delay model is based on the simplified RC delay model used in the LE method, extended to account for best-case delays. In 3.2.2, we will analyze the attained delay balance for the straightforward NAND-NOR and NOR-NAND decomposition, and show that the proposed decomposition method potentially results in rather poor delay balance when k_w is held constant. Here, we define “delay balance” solely based on delay variation; the smaller the delay variation is, the better the delay balance is. In 3.2.3, we will propose and analyze the NAND-gate only and NOR-gate only decomposition, and show that it potentially improves the delay balance for the constant k_w case, but increases worst-case delays. Thus, in 3.2.4, alternative decomposition techniques, namely the NAND-INV and NOR-INV decomposition, will be proposed and analyzed.

In 3.2.5, the monolithic implementation will also be analyzed, and shown to be a poor idea. This implementation is analyzed last as it is not useful for wave pipelining. Section 3.2.6 tabulates the calculation results for the alternative decomposition methods, and compares them.

3.2.1. Delay Model: Logical Effort with a Best Case

Here, we introduce a simplified model for CMOS gate delays. The delay calculation technique used here is similar to the one used in [49][50] and describes in section 2.5, except that here we also calculate best-case delays in addition to worst-case delays, since as previously, the performance attainable by a wave-pipelined circuit is limited by both best and worst case delays. Further, we do not normalize the delays and capacitances to those of the inverter.

While this model is not accurate, it should serve our purpose for simply qualitatively comparing between designs.

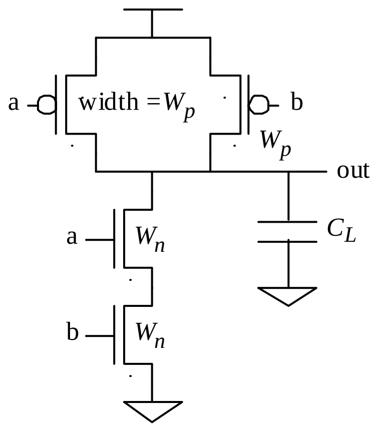


Figure 3.3. A 2-input NAND Gate with a Capacitive Load

As an example, for the NAND gate in figure 3.3, we calculate the delays as follows:

For rising outputs, the highest possible rise delay occurs when only one W_p-wide PMOS conduct; its resistance would be $R_{LH,MAX} = 1/W_p$ in that case. The lowest resistance will be attained if both PMOS transistor conducts; in that case, its resistance would be $R_{LH,MIN} = 1/2W_p$. However, the fall resistance would be approximately fixed at $R_{HL} = 2/W_n$, since the current to ground would have to pass both NMOS transistors. Actually, the inputs closest to output should be fastest. However, our very simplified model is not capable of showing this delay difference, since as was done in the LE method, it lumps the internal capacitances in the output pin only. Nonetheless, whenever the longest delay possible is desired, in a serial transistor chain, the input farthest away from the primary output should be chosen.

As in 2.5, the output capacitance is assumed to be lumped completely at the output pin, and proportional to the total sum of width of transistors at the output node; thus, $C_{out} = k_{out} \cdot (W_n + 2W_p)$. Likewise, the input capacitance for each input pin is assumed to be proportional to the sum of widths of the transistors attached to the pin. Thus, for each input pin, $C_{in} = k_{in} \cdot (W_n + W_p)$. Further, the delay is modeled as an RC delay. Thus, the load-free (or "parasitic") fall delay of the NAND gate is $p_{HL,NAND} = R_{HL} \cdot C_{out} = 2k_{out} \cdot (W_n + 2W_p)/W_n$.

To simplify calculations, we assume that $k_{in} = k_{out} = 1$. Since we only intend to make our model qualitatively accurate, the exact units are unimportant.

This simplified model will be used in the rest of this section. Here, we calculate the delays for inverters, NAND and NOR gates for two different cases, namely:

1. Constant PMOS-to-NMOS width ratio $W_p / W_n = k_w$.
2. Width ratio is adjusted such that worst-case rise delay is the same as worst-case fall delay.

Those two cases will be referred to as “case 1” and “case 2” henceforth.

Following (2.43), assuming long-channel MOSFET operation, the width ratio which gives the same rise and fall ratio for an inverter is $\gamma = \mu_n / \mu_p$ where μ_n and μ_p are the charge mobilities for the NMOS and the PMOS, respectively. For case 2, for an n_i-input NAND, $W_p / W_n = \gamma / n_i$, while for an n_i-input NOR, $W_p / W_n = n_i \gamma$.

It could be shown that in our model, the delay of a circuit remains unchanged if all transistor widths, including in the circuits directly driving it or directly driven by it, are scaled uniformly. Therefore, for delay calculations, we could assign any width which simplifies calculations. We will assume that $W_n + W_p = 1$. For instance, for an inverter, in case 1, $W_n = 1/(1 + k_w)$ and $W_p = k_w / (1 + k_w)$. Inverters, NAND gates, and NOR gates will exhibit unity input capacitance per pin ($C_{in} = 1$).

Table 3.1 gives the estimate of transistor width, input capacitance per pin, output capacitance, output resistance, and parasitic delays of inverters, NOR gates, and NAND gates for both cases. We do not have a good model for rise/fall times; however, a good estimate, found through SPICE simulations, is that the 10%-to-90% rise time (or 90%-to-10% fall time) of a logic gate is approximately 1.6 times gate delay. Therefore, since rise/fall times are generally longer than gate delay, they could easily become the bottleneck of a wave pipelined or fine-grained conventional pipelined design.

For case 2, the above table only differ cosmetically from the tables in p. 71 and p. 81 of [50], in that here the output resistance, and hence delay, is not normalized to the delay of an inverter; therefore, the results differ by a factor of $1 + k_w$ due to lack of normalizations. We chose not to normalize here, since we also evaluate case 1, where delays depend also on k_w, and such a normalization will not simplify further calculations here. Case 1 is not discussed in [50].

Table 3.1. Characteristics of Inverters, NANDs, and NORs for Both Cases

	INV		NAND		NOR	
	Case 1	**Case 2**	**Case 1**	**Case 2**	**Case 1**	**Case 2**
W_p	$\frac{k_w}{1+k_w}$	$\frac{\gamma}{1+\gamma}$	$\frac{k_w}{1+k_w}$	$\frac{\gamma}{n_i+\gamma}$	$\frac{k_w}{1+k_w}$	$\frac{n_i\gamma}{1+n_i\gamma}$
W_n	$\frac{1}{1+k_w}$	$\frac{1}{1+\gamma}$	$\frac{1}{1+k_w}$	$\frac{n_i}{n_i+\gamma}$	$\frac{1}{1+k_w}$	$\frac{1}{1+n_i\gamma}$
C_{in} per pin	1	1	1	1	1	1
C_{out}	1	1	$\frac{1+n_ik_w}{1+k_w}$	$\frac{n_i\cdot(1+\gamma)}{n_i+\gamma}$	$\frac{n_i+k_w}{1+k_w}$	$\frac{n_i\cdot(1+k_w)}{1+n_ik_w}$
G	-	1	-	n_i	-	n_i
$R_{LH,MIN}$	$\frac{\gamma}{k_w}(1+k_w)$	$1+\gamma$	$\frac{\gamma}{n_ik_w}(1+k_w)$	$\frac{n_i+\gamma}{n_i}$	$\frac{n_i\gamma}{k_w}(1+k_w)$	$1+n_i\gamma$
$R_{LH,MAX}$	$\frac{\gamma}{k_w}(1+k_w)$	$1+\gamma$	$\frac{\gamma}{k_w}(1+k_w)$	$n_i+\gamma$	$\frac{n_i\gamma}{k_w}(1+k_w)$	$1+n_i\gamma$
$R_{HL,MIN}$	$1+k_w$	$1+\gamma$	$n_i\cdot(1+k_w)$	$n_i+\gamma$	$\frac{1+k_w}{n_i}$	$\frac{1+n_i\gamma}{n_i}$
$R_{HL,MAX}$	$1+k_w$	$1+\gamma$	$n_i\cdot(1+k_w)$	$n_i+\gamma$	$1+k_w$	$1+n_i\gamma$
$p_{LH,MIN}$	$\frac{\gamma}{k_w}(1+k_w)$	$1+\gamma$	$\frac{\gamma}{n_ik_w}(1+n_ik_w)$	$1+\gamma$	$\frac{n_i\gamma}{k_w}(1+n_ik_w)$	$n_i\cdot(1+\gamma)$
$p_{LH,MAX}$	$\frac{\gamma}{k_w}(1+k_w)$	$1+\gamma$	$\frac{\gamma}{k_w}(1+n_ik_w)$	$n_i\cdot(1+\gamma)$	$\frac{n_i\gamma}{k_w}(1+n_ik_w)$	$n_i\cdot(1+\gamma)$
$p_{HL,MIN}$	$1+k_w$	$1+\gamma$	$n_i\cdot(1+n_ik_w)$	$n_i\cdot(1+\gamma)$	$\frac{n_i+k_w}{n_i}$	$1+\gamma$
$p_{HL,MAX}$	$1+k_w$	$1+\gamma$	$n_i\cdot(1+n_ik_w)$	$n_i\cdot(1+\gamma)$	n_i+k_w	$n_i\cdot(1+\gamma)$

As in section 2.5, the delay of a loaded logic gate is modeled as:

$$t_p = p + R\cdot C_L \qquad (3.5)$$

where C_L is the load capacitance it drives. As Table 3.1 suggests, even for the same logic gate, we will have several different p, R, and hence t_p values, depending on whether it is a rise or fall transition, and which inputs cause the transition. For conventional designs (not wave-pipelined), usually only worst case delays matter. For wave-pipelining, exact delays need to be known. We only consider best-case and worst-case delays, which is a pessimistic approach, since they may not necessarily both occur in a particular circuit.

However, this pessimistic approach is prudent, since our model is very highly simplified. Unlike [46] and [56], for instance, our model does not account for the dependence of delay on input rise/fall times, or the difference of delays between different input pins.

3.2.2. Straightforward NAND-NOR Decomposition

Here, we attempt to find the technique to decompose a wide NAND or wide NOR gate to attain the best delay balance, and hence the best wave-pipelining performance.

A wide-NAND or wide-NOR gate could be straightforwardly decomposed into a tree with alternating NAND and NOR levels, such as shown in Figure 3.4. Decomposition into an even number of NAND/NOR gate levels will result in NAND-NOR-NAND-NOR….. NAND-NOR-INV for NAND gates, and NOR-NAND-NOR-NAND….. NOR-NAND-INV for NOR gates; for an odd number of stages, the final inverter is not needed, and instead we have NAND-NOR-NAND-NOR….. NAND-NOR-NAND for wide NAND gates, and NOR-NAND-NOR-NAND….. NOR-NAND-NOR for wide NOR gates. In either case, all NAND gates have the same inversion parity, and so do all NOR gates, and the inversion parities of NAND and NOR gates will differ.

The terms "NAND-NOR decomposition" and "NOR-NAND decomposition" will be used interchangeably henceforth; it is assumed that it is clear which one is meant, based on the context. The terms "decomposition" and "implementation" will also be used interchangeably. The NAND-NOR decomposition method is considered as the "default method", as it is the most logically efficient of the decomposition methods which result in a reasonable delay balance, and is the likely result when we use an automatic synthesis tool to perform the decomposition, using a cell library of narrower logic gates.

Assuming that all gates have the same number of input n_i, the number of stages needed for an n_m-input wide-NAND or wide-NOR is:

$$n_s = \lceil \log_{n_i}(n_m) \rceil \tag{3.6}$$

which, in our work, is approximated as:

$$n_s \approx \log_{n_i}(n_m) \tag{3.7}$$

To simplify, let us assume that the number of stages n_s is even, or large enough that assuming it to be even will not result in large errors. Analysis for odd number of stages will not be performed here, as we only intend our analysis to be qualitatively accurate. With an even number of stages, we could estimate the delay and delay variation of a circuit by simply analyzing two consecutive gates, and multiplying the results by $n_s/2$.

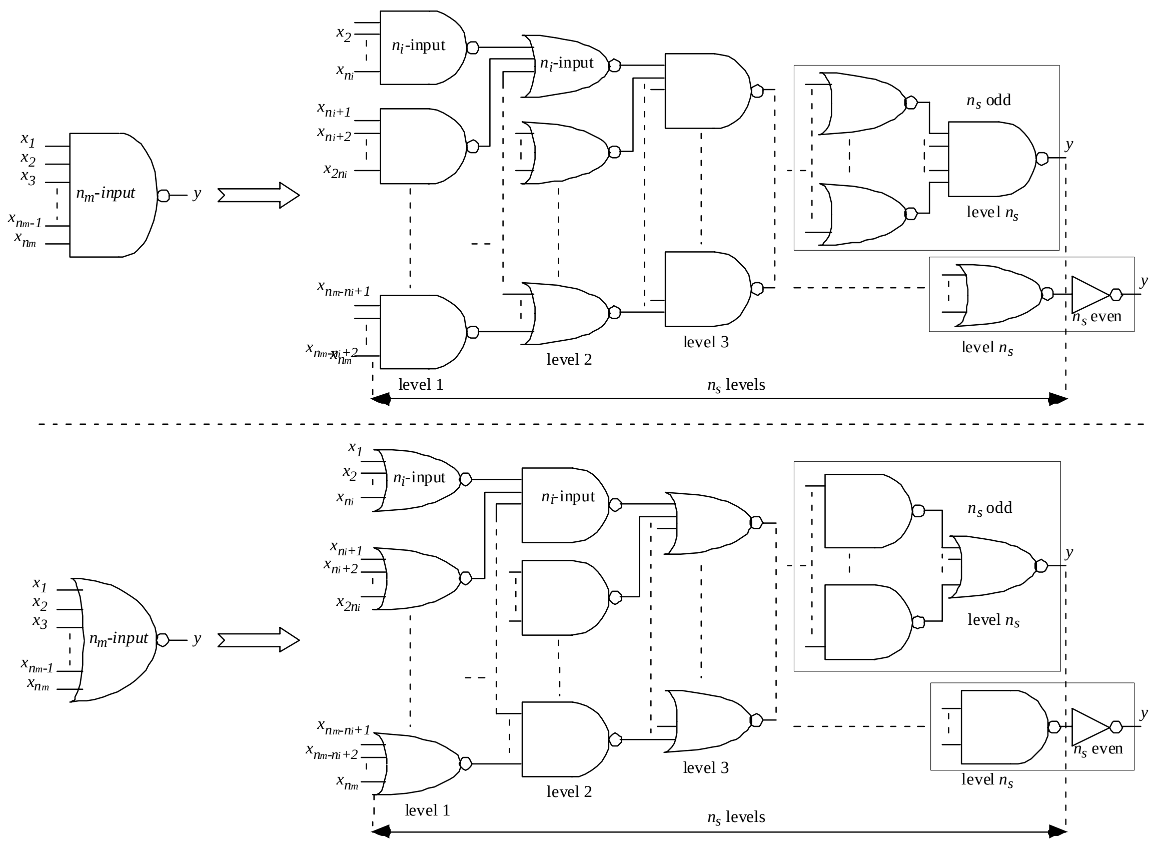


Figure 3.4. Top: NAND-NOR Decomposition of a Wide-NAND Gate. Bottom: NOR-NAND Decomposition of Wide-NOR gate.

We also assume that the load of the wide NAND/NOR gate is the same as the input capacitance of one pin C_{in}. Further, we will simplify the problem by assuming that for all cells, $C_{load} = C_{in}$.

Let us denote the best (shortest) rising and falling delays for NAND and NOR gates as $t_{pLH,NAND,MIN}$, $t_{pHL,NAND,MIN}$, $t_{pLH,NOR,MIN}$, and $t_{pHL,NOR,MIN}$, respectively; while their maximum delays are $t_{pLH,NAND,MAX}$, $t_{pHL,NAND,MAX}$, $t_{pLH,NOR,MAX}$, and $t_{pHL,NOR,MAX}$. In principle, we need to check the following four values:

$$t_{p,NAND-NOR,MAX1} = \frac{n_s}{2}.\left(t_{pLH,NAND,\max} + t_{pHL,NOR,\max}\right) \tag{3.8}$$

$$t_{p,NAND-NOR,MAX2} = \frac{n_s}{2}.\left(t_{pLH,NOR,\max} + t_{pHL,NAND,\max}\right) \tag{3.9}$$

$$t_{p,NAND-NOR,MIN1} = \frac{n_s}{2}.\left(t_{pLH,NAND,\min} + t_{pHL,NOR,\min}\right) \tag{3.10}$$

$$t_{p,NAND-NOR,MIN2} = \frac{n_s}{2}.\left(t_{pLH,NOR,\min} + t_{pHL,NAND,\min}\right) \tag{3.11}$$

In the equations in the rest of this section, we treat $n_s/2$ as a common factor, so as to facilitate comparison.

Since NAND gates tend to be slowest when their outputs are falling, while NOR gates slowest when rising, we may conjecture that for reasonable k_w and realistic γ values, $t_{p,NAND-NOR,MIN} = t_{p,NAND-NOR,MIN1}$ and $t_{p,NAND-NOR,MAX} = t_{p,NAND-NOR,MAX2}$. It could be shown that for case 1, namely logic gates with constant PMOS-to-NMOS width ratio, the worst and best case delays, as well as the difference are:

$$t_{p,NAND-NOR,MAX} = \frac{n_s}{2}.n_i \cdot \left[\frac{\gamma}{k_w}\left(n_i + 1 + 2k_w\right) + 2 + \left(n_i + 1\right)\cdot k_w\right] \tag{3.12}$$

$$t_{p,NAND-NOR,MIN} = \frac{n_s}{2}.\frac{1}{n_i}\left[\frac{\gamma}{k_w}\left(2 + \left(n_i + 1\right)\cdot k_w\right) + \left(n_i + 1\right) + 2k_w\right] \tag{3.13}$$

$$\Delta t_{p,NAND-NOR} = \frac{n_s}{2}.\left[\left(n_i - \frac{\gamma}{n_i k_w}\right)\left(2 + \left(n_i + 1\right)\cdot k_w\right) + \left(\frac{n_i\gamma}{k_w} - \frac{1}{n_i}\right)\left(n_i + 1 + 2k_w\right)\right] \tag{3.14}$$

while for case 2, namely logic gates designed for equal worst-case rise and fall times:

$$t_{p,NAND-NOR,MAX} = \frac{n_s}{2}.\left(3n_i + 1\right)\cdot\left(1 + \gamma\right) \tag{3.15}$$

$$t_{p,NAND-NOR,MIN} = \frac{n_s}{2}.2\cdot\left(2 + \left(1 + \frac{1}{n_i}\right)\cdot\gamma\right) \tag{3.16}$$

$$\Delta t_{p,NAND-NOR} = \frac{n_s}{2}.\left[\left(3n_i - 3\right) + \left(3n_i - 1 - \frac{2}{n_i}\right)\cdot\gamma\right] \tag{3.17}$$

The problem with the alternating NAND-NOR decomposition is that while NAND gates tend to be fastest when their outputs are rising (with all input changing at the same time) and slowest when falling, NOR gates tend to be fastest when falling (again, with all inputs changing at the same time) and slowest when rising. Thus, in this decomposition, where any two consecutive stages will have opposite inversion parities, the input transition which causes a NAND gate to exhibit fastest possible rising output could also conceivably cause the NOR gate to reach its fastest possible falling output, and hence they combine to attain the shortest possible delay for the system. It is likewise possible that when the NAND exhibits the falling output while the NOR is, therefore, rising, they will attain their respective worst-case delays. *The straightforward NAND-NOR decomposition potentially maximizes the delay variation*, and reduces the wave pipelining performance.

A possible solution is decomposing the wide-NAND or wide-NOR gate into a structure in which the smaller NAND gates (or NOR) are used with both inversion parities, instead of using NANDs for one value of inversion parity and NOR for the other. One way is by decomposing the wide gate into a circuit consisting solely of NAND or NOR gates. It is not possible, however, to find an *efficient* decomposition of this form; "*efficient*" meaning that for any n_i-input gate within the structure, each the n_i inputs are actually connected to a distinct logic signal – thus, for instance, using a 2-input NOR gate as an inverter (whether by tying one input to ground, or by tying both input pins to the same input signal) is not considered efficient. Thus, the NAND-NOR decomposition is efficient, but NAND-only and NOR-only decompositions are not.

The NAND-only and NOR-only decompositions will have to use n_s-input NAND or NOR gates as inverters. They use twice as many gate stages ($2n_s$) as the NAND-NOR decomposition; they use n_s stages, each consisting of a NAND-NAND or NOR-NOR pair. Thus, even if this style of decomposition does improve delay balance, it potentially doubles the latency.

A possible compromise is using actual inverters, instead of using NAND or NOR gates, as inverters. Here all NAND or NOR gates will have the same inversion parities; thus, it may appear that it would not result in improvement. However, subsequent results will show that they in fact give best performance for case 2. There will be as many stages (n_s) of NAND (or NOR) gates as the total number of NAND and NOR stages in a NAND-NOR decomposition.

In the following subsections, we will analyze those alternative decompositions, in addition to simply using a single, wide ("monolithic") NAND or NOR gate. It will be shown that using wide, monolithic NAND or NOR will not give poor delay balance, and that the straightforward NAND-NOR decomposition gives better delay balance and best latency. However, NAND-only, and NOR-only decomposition may improve delay balance in some circumstances.

3.2.3. NAND-NAND and NOR-NOR Decompositions

This section analyzes NAND-NAND and NOR-NOR decompositions. Henceforth, the terms "NAND-NAND decomposition" and "NAND-only decomposition" will be considered synonymous, and so will "NOR-NOR decomposition" and "NOR-only decomposition" be.

The NAND-NAND decomposition is based on treating a wide-NAND gate as a wide-AND followed by an inverter. Then, the n_m-input wide-AND gate is decomposed into n_s stages smaller, n_i-input AND gates. Each small AND gate is further decomposed into two stages of NAND gates, in which the second stage is used simply as an inverter. Thus, as previously mentioned, this implementation consists of approximately $2n_s$ gate stages overall, or twice as many as the NAND-NOR decomposition, and potentially doubles the latency. However, it is expected that the delay variation is less, because we use NAND gates for both inversion parities, and the delay variations of NAND gates in any two consecutive stages should partly cancel each other: the transition which causes one stage to exhibit its longest possible delay should cause the next one to exhibit its shorter-than-average delays and vice-versa. Still, this implementation is not logically efficient, as in every other stage, we have n_i-input NAND gates which are functioned as inverters; thus, $n_i - 1$ inputs are not used for logic purposes.

Here we have chosen to use only one input pin for the inverter-functioned NAND. We could use several, but it would slow down the NAND gate which drives it, unless the inverter-functioned NAND is made smaller, which will slow it down unless its load – and all subsequent stages – is made smaller as well. If we use several inputs of the next stage of inverter-functioned NAND gate, it will have to be made even smaller. Eventually, in a multistage circuit, potentially the last stage will become very small, has very little load-driving ability, and exhibits excessive delay in the presence of load. Therefore, to keep the electrical effort of each gate at unity, only one input should be used in such NAND gates.

Similarly, a wide-NOR gate could be decomposed into a NOR-NOR circuit. Such a NOR-NOR decomposition is similarly inefficient in its use of NOR gates as inventers.

Alternatively, actual inverters could be used instead of inverter-functioned NAND or NOR gates. Here, we will have n_s stages of NAND or NOR gates, and n_s stages of inverters, which are faster than NAND or NOR gates. The resulting latency will be between those of NAND-NOR implementation and NAND-NAND / NOR-NOR implementation. Figure 3.5 gives an example for each of those four implementations: NAND-NAND, NOR-NOR, NAND-INV, and NOR-INV. In the picture, a 4-input NAND is decomposed into a circuit containing logic gates with no more than two inputs.

However, the treatment for NAND-INV and NOR-INV implementations will be left to the next section. This section focuses on NAND-NAND and NOR-NOR implementations. The worst and best case delays of the NAND-NAND decomposition are:

$$t_{p,NAND-NAND,MAX} = \frac{n_s}{2} . 2 \cdot \left(t_{pLH,NAND,MAX} + t_{pHL,NAND,MAX}\right) \tag{3.18}$$

$$t_{p,NAND-NAND,MIN} = \frac{n_s}{2} . 2 \cdot \left(t_{pLH,NAND,MIN} + t_{pHL,NAND,MIN}\right) \tag{3.19}$$

The factor of two appears since there are n_s stages of NAND-NAND pair, unlike in the NAND-NOR decomposition, which uses only $n_s/2$ stages of NAND-NOR pairs. The equations (3.18) and (3.19) are pessimistic, as the second stage of NAND is functioned as inverter, and it may not exhibit the shortest possible or longest possible delay. However, this approach is proper, for two reasons: First, as previously mentioned, our model is only qualitatively accurate; and second, for the NAND gate which is functioned as inverter, it is not clear which input pin will actually be used, as it may be influenced by layout considerations. However, it may be noted that this observation means that, since only one input of the inverter-functioned gate is used, according to simple RC-delay model we have followed, such stages could only exhibit one kind of rising delay and one kind of falling delay, which reduce their delay variation. Therefore, relative to other decomposition techniques, NAND-NAND and NOR-NOR decompositions should be more likely to perform better than our models would indicate.

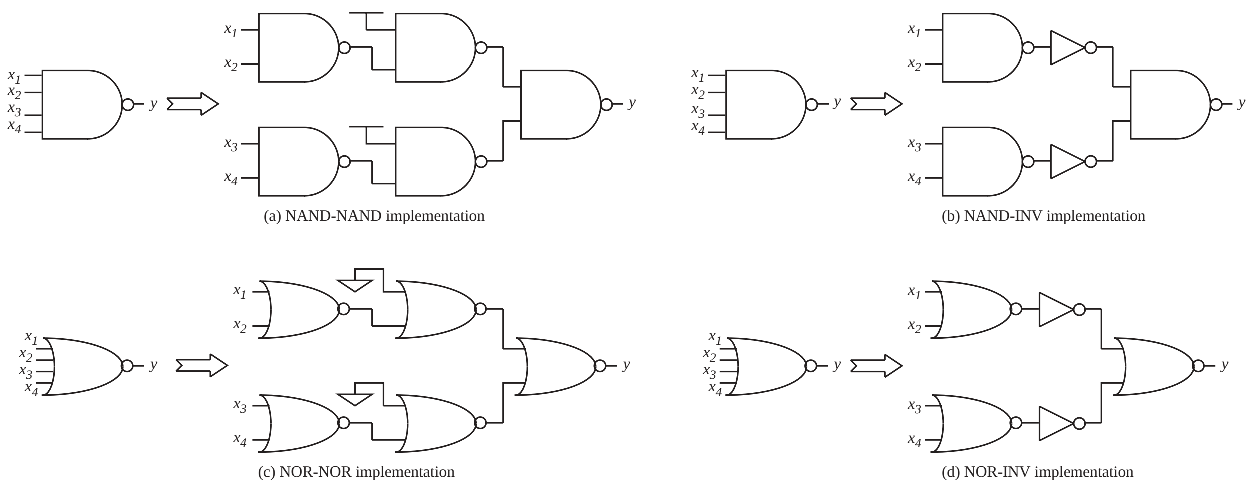

x_1
x_2
x_3
x_4
y
(a) NAND-NAND implementation
(b) NAND-INV implementation
(c) NOR-NOR implementation
(d) NOR-INV implementation

Figure 3.5. Examples of Four Different Decompositions of NAND4.

Similarly, for NOR-NOR decomposition, the worst and best case delays could be computed as:

$$t_{p,NOR-NOR,MAX} = \frac{n_s}{2} . 2 \cdot \left(t_{pLH,NOR,MAX} + t_{pHL,NOR,MAX}\right) \tag{3.20}$$

$$t_{p,NOR-NOR,MIN} = \frac{n_s}{2} . 2 \cdot \left(t_{pLH,NOR,MIN} + t_{pHL,NOR,MIN}\right) \tag{3.21}$$

Following the relation $t_p = p + R \cdot C_L$, and using the values from Table 3.1, the equations (3.18) – (3.21) could be expanded further, and Δt_p for both cases could be found. For case 1, for NAND-NAND decomposition, they become:

$$t_{p,NAND-NAND,MAX} = \frac{n_s}{2} . 2 \cdot \left[\left(n_i + \frac{\gamma}{k_w}\right)\left(2 + (n_i + 1) \cdot k_w\right)\right] \tag{3.22}$$

$$t_{p,NAND-NAND,MIN} = \frac{n_s}{2} . 2 \cdot \left[\left(n_i + \frac{\gamma}{n_i k_w}\right)\left(2 + (n_i + 1) \cdot k_w\right)\right] \tag{3.23}$$

$$\Delta t_{p,NAND-NAND} = \frac{n_s}{2} . 2 \cdot \left[\frac{\gamma}{k_w}\left(1 - \frac{1}{n_i}\right)\left(2 + (n_i + 1) \cdot k_w\right)\right] \tag{3.24}$$

While for NOR-NOR decomposition:

$$t_{p,NOR-NOR,MAX} = \frac{n_s}{2} . 2 \cdot \left[\left(\frac{n_i \gamma}{k_w} + 1\right)\left(n_i + 1 + 2k_w\right)\right] \tag{3.25}$$

$$t_{p,NOR-NOR,MIN} = \frac{n_s}{2} . 2 \cdot \left[\left(\frac{n_i \gamma}{k_w} + \frac{1}{n_i} \right) (n_i + 1 + 2k_w) \right] \tag{3.26}$$

$$\Delta t_{p,NOR-NOR} = \frac{n_s}{2} . 2 \cdot \left[\left(1 - \frac{1}{n_i} \right) (n_i + 1 + 2k_w) \right] \tag{3.27}$$

For case 2, the corresponding results for NAND-NAND decomposition are:

$$t_{p,NAND-NAND,MAX} = \frac{n_s}{2} . 4 \cdot [2n_i + (1 + n_i) \cdot \gamma] \tag{3.28}$$

$$t_{p,NAND-NAND,MIN} = \frac{n_s}{2} . 2 \cdot \left[(1 + n_i) \cdot \left(2 + \left(1 + \frac{1}{n_i} \right) \cdot \gamma \right) \right] \tag{3.29}$$

$$\Delta t_{p,NAND-NAND} = \frac{n_s}{2} . 2 \cdot \left[2n_i - 2 + \left(n_i - \frac{1}{n_i} \right) \cdot \gamma \right] \tag{3.30}$$

While for NOR-NOR decomposition:

$$t_{p,NOR-NOR,MAX} = \frac{n_s}{2} . 4 \cdot (n_i + 1 + 2n_i \gamma) \tag{3.31}$$

$$t_{p,NOR-NOR,MIN} = \frac{n_s}{2} . 2 . \left[\left(n_i + 2 + \frac{1}{n_i} \right) + 2(n_i + 1) \cdot \gamma \right] \tag{3.32}$$

$$\Delta t_{p,NOR-NOR} = \frac{n_s}{2} . 2 \cdot \left[n_i - \frac{1}{n_i} + 2(n_i - 1) \cdot \gamma \right] \tag{3.33}$$

For case 1, the delay balance of NAND-NAND and NOR-NOR structures should be better than it is for the NAND-NOR decomposition, since here Δt_p only depends linearly on n_i instead of quadratically. Defining:

$$A_{NAND-NAND} \equiv \Delta t_{p,NAND-NAND} - \Delta t_{p,NAND-NOR} \tag{3.34}$$

Or, using the symbol A as the improvement (or, equivalently, reduction) in delay variation attained by an implementation technique over the NAND-NOR implementation under the same case, we could show, that for case 1,

$$A_{NAND-NAND} = \frac{n_s}{2} . \left[\left(n_i - \frac{\gamma}{n_i k_w} \right) \cdot (2 + (n_i + 1) \cdot k_w) + \left((n_i - 2) \frac{\gamma}{k_w} + \frac{1}{n_i} \right) \cdot (n_i + 1 + 2k_w) \right] \tag{3.35}$$

For anything but inverters or buffers, $n_i \geq 2$ always. Further, $\left(n_i - \gamma / n_i k_w\right) \geq 0$ for reasonable values of k_w, since we will likely always have $k_w \geq \gamma / (n_i)^2$. Typically, k_w is in the 1–2.5 range, well over 0.75 needed to satisfy the inequality for $\gamma = 3$. Thus, $A_{NAND-NAND} > 0$, and we could expect the delay variation of NAND-NAND implementation to be much reduced compared with that for NAND-NOR decomposition. It could likewise be shown that:

$$A_{NOR-NOR} = \frac{n_s}{2} \cdot \left[\left(n_i - \frac{\gamma}{n_i k_w} \right) \cdot \left(2 + (n_i + 1) \cdot k_w\right) + \left(\frac{n_i \gamma}{k_w} - 2 + \frac{1}{n_i} \right) \cdot \left(n_i + 1 + 2k_w\right) \right] \tag{3.36}$$

It could likewise be shown that $A_{NOR-NOR} > 0$, for the same reason.

For case 2, it could be shown that:

$$A_{NAND-NAND} = \frac{n_s}{2} \cdot (n_i - 1) \cdot (\gamma - 1) \tag{3.37}$$

$$A_{NOR-NOR} = \frac{n_s}{2} \cdot \left(n_i - 3 + \frac{2}{n_i} \right) \cdot (1 - \gamma) \tag{3.38}$$

Thus, for case 2, $\Delta t_{p,NAND-NAND}$ is better (less) than $\Delta t_{p,NAND-NOR}$ by $(n_i - 1) \cdot (\gamma - 1)$, but $\Delta t_{p,NOR-NOR}$ is the same as $\Delta t_{p,NAND-NOR}$. However, for $n_i = 2$, the difference between $\Delta t_{p,NAND-NAND}$ and $\Delta t_{p,NAND-NOR}$ is only $\gamma - 1$. For instance, for $\gamma = 2$, $k_w = 2$ and $n_i = 2$, we obtain $\Delta t_{p,NAND-NOR} = 11$, $A_{NAND-NAND} = 1$, and $A_{NOR-NOR} = 0$. The equations will be evaluated in Section 3.2.6, where it will be suggested that, considering the simplifications used in our model, the 10% difference is insignificant enough to be ignored, and we have to resort to circuit-level simulations. Likewise, (3.38) could be discounted for the same reason. Therefore, in Chapter 4, we would compare their performances through SPICE simulations.

Comparison between equations (3.23), (3.26), (3.29) and (3.32) for the NAND-NAND and NOR-NOR decomposition, with (3.13) and (3.16) for NAND-NOR decomposition, suggest that the NAND-NAND / NOR-NOR decomposition approximately doubles the latency, represented by t_{pMAX}, over the NAND-NOR decomposition. Thus, we will also analyze the NAND-INV and NOR-INV decomposition as a compromise.

3.2.4. NAND-INV and NOR-INV Decompositions: Compromises

For the NAND-INV and NOR-INV decomposition, we note that the inverter has a faster worst-case delay than other gates for realistic γ and k_w values, but at the same time, it has a lower drive strength than the best-case drive strength of NAND or NOR gates, which occurs when the NAND/NOR gates' parallel transistors conduct simultaneously. Therefore, for the NAND-INV decomposition:

$$t_{p,NAND-INV,MAX} = \frac{n_s}{2} . 2 \cdot \left(t_{pLH,INV} + t_{pHL,NAND,MAX}\right) \tag{3.39}$$

$$t_{p,NAND-INV,MIN} = \frac{n_s}{2} . 2 \cdot \left(t_{pLH,NAND,MIN} + t_{pHL,INV}\right) \tag{3.40}$$

While for NOR-INV decomposition:

$$t_{p,NOR-INV,MAX} = \frac{n_s}{2} . 2 \cdot \left(t_{pLH,NOR,MAX} + t_{pHL,INV}\right) \tag{3.41}$$

$$t_{p,NOR-INV,MIN} = \frac{n_s}{2} . 2 \cdot \left(t_{pLH,INV} + t_{pHL,NOR,MIN}\right) \tag{3.42}$$

For case 1, for NAND-INV decomposition, the above equations expand to:

$$t_{p,NAND-INV,MAX} = \frac{n_s}{2} . 2 \cdot \left[\frac{2\gamma}{k_w}(1+k_w) + n_i\left(2+(n_i+1)\cdot k_w\right)\right] \tag{3.43}$$

$$t_{p,NAND-INV,MIN} = \frac{n_s}{2} . 2 \cdot \left[\frac{\gamma}{n_i k_w}\left(2+(n_i+1)\cdot k_w\right) + (2+2k_w)\right] \tag{3.44}$$

$$\Delta t_{p,NAND-INV} = \frac{n_s}{2} . 2 \cdot \left[\left(\frac{2\gamma}{k_w} - 2\right)(1+k_w) + \left(n_i - \frac{\gamma}{n_i k_w}\right)\left(2+(n_i+1)\cdot k_w\right)\right] \tag{3.45}$$

$$A_{NAND-INV} =$$

$$\frac{n_s}{2} . \left[\left(-n_i + \frac{\gamma}{n_i k_w}\right)\cdot\left(2+(n_i+1)\cdot k_w\right) + \left(\frac{n_i\gamma}{k_w} - \frac{1}{n_i}\right)\cdot\left(n_i+1+2k_w\right) + \left(\frac{2\gamma}{k_w} - 2\right)\cdot(1+k_w)\right] \tag{3.46}$$

while for NOR-INV decomposition they expand to:

$$t_{p,NOR-INV,MAX} = \frac{n_s}{2} . 2 \cdot \left[\frac{n_i\gamma}{k_w}\left(n_i+1+2k_w\right) + 2(1+k_w)\right] \tag{3.47}$$

$$t_{p,NOR-INV,MIN} = \frac{n_s}{2}.2\cdot\left[\frac{2\gamma}{k_w}(1+k_w)+\frac{1}{n_i}(n_i+1+2k_w)\right] \quad (3.48)$$

$$\Delta t_{p,NOR-INV} = \frac{n_s}{2}.2\cdot\left[\left(\frac{n_i\gamma}{k_w}-\frac{1}{n_i}\right)(n_i+1+2k_w)+\left(2-\frac{2\gamma}{k_w}\right)(1+k_w)\right] \quad (3.49)$$

$$A_{NOR-INV} =$$

$$\frac{n_s}{2}.\left[\left(n_i-\frac{\gamma}{n_ik_w}\right)\cdot(2+(n_i+1)\cdot k_w)+\left(\frac{1}{n_i}-\frac{n_i\gamma}{k_w}\right)\cdot(n_i+1+2k_w)+\left(\frac{2\gamma}{k_w}-2\right)\cdot(1+k_w)\right] \quad (3.50)$$

Evaluating (3.14), (3.46) and (3.50) for γ= 2, k_w= 2 and n_i = 2, we obtain $\Delta t_{p,NAND-NOR}$= 20.5, $A_{NAND-INV}$= -3.5, and $A_{NOR-INV}$ = -0.5. As mentioned before regarding case 2 for NAND-NAND and NOR-NOR decompositions, this difference (no more than 17% here) is too small to be considered significant, considering the crudeness of our model; therefore, SPICE simulations become mandatory. However, considering that both NAND-INV and NOR-INV decompositions are likely to considerably increase latency, the likely slight difference may not justify choosing them.

For case 2, the expressions for NAND-INV become:

$$t_{p,NAND-INV,MAX} = \frac{n_s}{2}.2\cdot(2n_i+2+(n_i+3)\cdot\gamma) \quad (3.51)$$

$$t_{p,NAND-INV,MIN} = \frac{n_s}{2}.2\cdot\left[4+\left(3+\frac{1}{n_i}\right)\cdot\gamma\right] \quad (3.52)$$

$$\Delta t_{p,NAND-INV} = \frac{n_s}{2}.2\cdot\left[2n_i-2+\left(n_i-\frac{1}{n_i}\right)\cdot\gamma\right] \quad (3.53)$$

$$A_{NAND-INV} = \frac{n_s}{2}.(n_i-1)\cdot(\gamma-1) \quad (3.54)$$

while for NOR-INV decomposition they become:

$$t_{p,NOR-INV,MAX} = \frac{n_s}{2}.2\cdot(n_i+3+(2n_i+2)\cdot\gamma) \quad (3.55)$$

$$t_{p,NOR-INV,MIN} = \frac{n_s}{2}.2\cdot\left(3+\frac{1}{n_i}+4\gamma\right) \quad (3.56)$$

$$\Delta t_{p,NOR-INV} = \frac{n_s}{2} . 2 \cdot \left[n_i - \frac{1}{n_i} + 2(n_i - 1) \cdot \gamma \right] \qquad (3.57)$$

$$A_{NOR-INV} = \frac{n_s}{2} . \left(n_i - 3 + \frac{2}{n_i} \right) \cdot (1 - \gamma) \qquad (3.58)$$

A comparison between equations (3.37) and (3.54), and between (3.38) and (3.58), show that for case 2, NAND-NAND and NOR-NOR decompositions result in identical delay variations as NAND-INV and NOR-INV decomposition, respectively. It must be remembered that our model is rather crude; however, if this prediction is correct, there will be no reason to use NAND-NAND and NOR-NOR decomposition for case 2, as they give longer worst-case delays than NAND-INV and NOR-INV decomposition.

In the next subsection, we will show an alternative not to take: single-gate (or “monolithic”) implementation of wide NAND / wide NOR. While this approach reduces transistor count and hence probably area, it attains a poor wave-pipelining performance.

3.2.5. An Obvious but Bad Alternative: Monolithic Implementation

One obvious question is: why do we have to decompose the wide gate into smaller ones at all? Why not just use a single wide gate?

Here, it will be shown that this “monolithic” alternative is a poor one. First, it increases the delay variation. Second, it also tends to increase the worst-case delay, despite using only one logic level. Finally, it also increases rise/fall time, since here the longest gate delay equals the longest delay. In contrast, when the wide gate is decomposed to smaller ones, the longest gate delay will be less than the delay of the structure, and the worst case rise/fall times will be shortened.

For reasonable ranges of k_w, for NAND gates, we have:

$$t_{p,NAND,MAX} = t_{pHL,NAND,MAX} = p_{HL,NAND,MAX} + R_{LH,NAND,MAX} \cdot C_L \qquad (3.59)$$

$$t_{p,NAND,MIN} = t_{pLH,NAND,MIN} = p_{LH,NAND,MIN} + R_{LH,NAND,MIN} \cdot C_L \qquad (3.60)$$

while for NOR gates,

$$t_{p,NOR,MAX} = t_{pHL,NOR,MAX} = p_{LH,NOR,MAX} + R_{LH,NOR,MAX} \cdot C_L \qquad (3.61)$$

$$t_{p,NOR,MIN} = t_{pHL,NOR,MIN} = p_{HL,NOR,MIN} + R_{HL,NOR,MIN} \cdot C_L \tag{3.62}$$

This is true for our wide-NAND and wide-NOR as well. As in the previous parts regarding other decompositions, we would express this as a product of $n_s/2$. Obviously, t_p = $(n_s/2)\cdot t_p/(n_s/2)$. Using the subscripts *monoNAND* and *monoNOR* for our wide single-gate NAND and NOR, and the symbol n_m to denote the number of input of our wide-NAND or wide-NOR gate, and using C_L = 1 as before, from table 3.1, for case 1, for the monolithic NAND gate, we obtain:

$$t_{p,monoNAND,MAX} = \frac{n_s}{2}\cdot\frac{1}{n_s/2}.n_m \cdot (2+(n_m+1)\cdot k_w) \tag{3.63}$$

$$t_{p,monoNAND,MIN} = \frac{n_s}{2}\cdot\frac{1}{n_s/2}\cdot\frac{\gamma}{n_m k_w}(2+(n_m+1)\cdot k_w) = t_{p,monoNAND,MAX}\cdot\frac{1}{(n_m)^2}\frac{\gamma}{k_w} \tag{3.64}$$

$$\Delta t_{p,monoNAND} = \frac{n_s}{2}\cdot\frac{1}{n_s/2}\cdot\left[\left(n_m - \frac{\gamma}{n_m k_w}\right)(2+(n_m+1)\cdot k_w)\right]$$

$$= t_{p,monoNAND,MAX}\cdot\left(1-\frac{1}{(n_m)^2}\frac{\gamma}{k_w}\right) \tag{3.65}$$

Let us compare the above results with what would have been attained by the NAND-NOR decomposition of this n_m-input gate into n_s stages of n_i-input gates. From eq. (3.7), $n_s \approx \log_{n_i}(n_m)$, or $n_m \approx (n_i)^{n_s}$. Thus, the above three equations could also be expressed as:

$$t_{p,monoNAND,MAX} = \frac{n_s}{2}\cdot\frac{1}{n_s/2}.(n_i)^{n_s}\left(2+\left((n_i)^{n_s}+1\right)\cdot k_w\right) \tag{3.66}$$

$$t_{p,monoNAND,MIN} = \frac{n_s}{2}\cdot\frac{1}{n_s/2}\cdot\frac{1}{(n_i)^{n_s}}\frac{\gamma}{k_w}\left[2+\left((n_i)^{n_s}+1\right)\cdot k_w\right] \tag{3.67}$$

$$\Delta t_{p,monoNAND} = \frac{n_s}{2}\cdot\frac{1}{n_s/2}\cdot\left[\left((n_i)^{n_s} - \frac{1}{(n_i)^{n_s}}\frac{\gamma}{k_w}\right)\left(2+\left((n_i)^{n_s}+1\right)\cdot k_w\right)\right] \tag{3.68}$$

As seen, for a monolithic implementation, both $t_{p,\max}$ and Δt_p grow exponentially with n_s – or, more precisely, they grow linearly with n_m. As a comparison, in the worst of the other decomposition techniques, they grow linearly with $(n_i)^2$ for each stage, while for the entire circuit, they grow linearly with $n_s \cdot (n_i)^2$, or linearly with $(n_i)^2 \cdot \log_{n_i}(n_m)$, at worst. Clearly, except for very small n_m values, the monolithic implementation gives worse delay balance than any decomposition technique. However, those very small n_m values will not require decomposition. In chapter 4, it will be shown through SPICE simulation that for n_i = 2, the monolithic implementation gives worse delay balance than other decomposition techniques even for n_m as low as 4.

For completeness, for monolithic wide NOR, the corresponding expressions are, for case 1:

$$t_{p,monoNOR,MAX} = \frac{n_s}{2} \cdot \frac{1}{n_s/2} \cdot \left[\frac{n_m \gamma}{k_w} (n_m + 1 + 2k_w) \right] \tag{3.69}$$

$$t_{p,monoNOR,MIN} = \frac{n_s}{2} \cdot \frac{1}{n_s/2} \cdot \left[\frac{1}{n_m} (n_m + 1 + 2k_w) \right] = \frac{t_{p,monoNOR,MAX}}{(n_m)^2 \cdot (\gamma/k_w)} \tag{3.70}$$

$$\Delta t_{p,monoNOR} = \frac{n_s}{2} \cdot \frac{1}{n_s/2} \cdot \left[\left(\frac{n_m \gamma}{k_w} - \frac{1}{n_m} \right) (n_m + 1 + 2k_w) \right] = t_{p,monoNOR,MAX} \cdot \left(1 - \frac{1}{(n_m)^2 \cdot (\gamma/k_w)} \right) \tag{3.71}$$

while for case 2:

$$t_{p,monoNAND,MAX} = \frac{n_s}{2} \cdot \frac{1}{n_s/2} \cdot [2n_m + (n_m + 1) \cdot \gamma] \tag{3.72}$$

$$t_{p,monoNAND,MIN} = \frac{n_s}{2} \cdot \frac{1}{n_s/2} \cdot \left[2 + \left(1 + \frac{1}{n_m} \right) \cdot \gamma \right] = \frac{t_{p,monoNAND,MAX}}{n_m} \tag{3.73}$$

$$\Delta t_{p,monoNAND} = \frac{n_s}{2} \cdot \frac{1}{n_s/2} \cdot \left[2n_m - 2 + \left(n_m - \frac{1}{n_m} \right) \cdot \gamma \right] = t_{p,monoNAND,MAX} \cdot \left(1 - \frac{1}{n_m} \right) \tag{3.74}$$

$$t_{p,monoNOR,MAX} = \frac{n_s}{2} \cdot \frac{1}{n_s/2} \cdot [n_m + 1 + 2n_m \gamma] \tag{3.75}$$

$$t_{p,monoNOR,MIN} = \frac{n_s}{2} \cdot \frac{1}{n_s/2} \cdot \left[1 + \frac{1}{n_m} + 2\gamma \right] = \frac{t_{p,monoNOR,MAX}}{n_m} \tag{3.76}$$

$$\Delta t_{p,monoNOR} = \frac{n_s}{2} \cdot \frac{1}{n_s/2} \cdot \left[n_m - \frac{1}{n_m} + (n_m - 1) \cdot \gamma \right] = t_{p,monoNOR,MAX} \cdot \left(1 - \frac{1}{n_m} \right) \tag{3.77}$$

The delays grow at lower rate (linear dependent on n_m instead of quadratically) compared with case 1. However, the delay variations, as rise / fall times, are still worse than when the wide

NAND/NOR gate is decomposed. Additionally, laying out a wide NAND or wide NOR (= large n_m) for case 2 may not be possible, since it may require unreasonably large transistors. For instance, a 32-input NOR gate so constructed will require W_p / W_n = 64 for γ = 2, thus requiring very wide PMOS transistors, and complicating the design of the circuit driving its input.

In 3.2.6, we will numerically evaluate the resulting formulae derived in Section 3.2 so far, and compare the performance attained with the four decomposition techniques.

3.2.6. Numerical Comparison amongst Implementations

Here, we will evaluate the already derived formulae to predict how well those schemes will likely work. Let us assume that we would like to implement a 16-input NAND or NOR, either as a single gate, or as a combination of inverters and several 2-input NAND or NOR gates. We will evaluate it for both case 1 and case 2. Let us use k_w = 2 for case 1, and let us evaluate it for γ = 2 and γ = 3. Using the formulae, we could estimate the worst-case delays and delay variation of the complete n_w-input gate, normalized with respect to $n_s/2$, as well as the wave number N_{MAX} theoretically attainable. It must be added that N_{MAX} in itself only suggests that a circuit is capable of operating under wave pipelining, but is not a useful measure of the wave pipelining performance of a circuit, however; even a circuit with large Δt_p could attain a high N_{MAX} if the worst-case delay is long, but while such a circuit will function properly under wave pipelining, it will exhibit poor performance both in terms of throughput and latency.

Table 3.2 shows the result for γ = 2, while table 3.3 shows the result for γ = 3. Here, it must be noted that it is not possible to wave-pipeline the monolithic implementation without the presence of other circuits, since for a single gate, output rise/fall times exceed propagation delay; thus, actually, N_{MAX} < 1 for wide gates. Thus, $N_{MAX} = t_{p,MAX} / \Delta t_p$ simply represents the wave number theoretically attainable by a circuit with those particular worst-case delay and delay variation.

Table 3.2. Worst Case Delays, Delay Variations, and Theoretical Maximum Wave Numbers for 16-Input Wide Gates, n_s = 4, for γ = 2

Decomposition Technique	Case 1			Case 2		
	$t_{p,MAX}$	Δt_p	N_{MAX}	$t_{p,MAX}$	Δt_p	N_{MAX}
Monolithic NAND	288	286.88	1.00	33	30.94	1.07
Monolithic NOR	168	167.34	1.00	40.5	37.96	1.07
NAND-NOR	30	20.5	1.46	21	11	1.91
NAND-NAND	48	8	6.00	40	10	4.00
NOR-NOR	42	7	6.00	44	11	4.00
NAND-INV	44	24	1.83	32	10	3.20
NOR-INV	40	21	1.91	38	11	3.46

Table 3.3. Worst Case Delays, Delay Variations, and Theoretical Maximum Wave Numbers for 16-Input Wide Gates, n_s = 4, for γ = 3

Decomposition Technique	Case 1			Case 2		
	$t_{p,MAX}$	Δt_p	N_{MAX}	$t_{p,MAX}$	Δt_p	N_{MAX}
Monolithic NAND	288	286.31	1.01	41.5	39.91	1.04
Monolithic NOR	252	251.34	1.00	56.5	52.97	1.07
NAND-NOR	37	27.5	1.35	28	15	1.87
NAND-NAND	56	8	7.00	52	13	4.00
NOR-NOR	56	7	8.00	60	15	4.00
NAND-INV	50	26	1.92	42	13	3.23
NOR-INV	54	29	1.86	52	15	3.47

Using the NAND-NOR implementation as the default implementation, Tables 3.4 and 3.5 show the same results again, but here we arbitrarily replace Δt_p and $t_{p,MAX}$ (for which smaller number means better performance) with normalized $1/t_{p,MAX} = t_{p,NAND-NOR,MAX} / t_{p,MAX}$ and normalized $1/\Delta t_p = \Delta t_{p,NAND-NOR} / \Delta t_p$, respectively. Thus, for all columns, now higher numbers translate to better performance. Here N_{MAX} is omitted, since as previously mentioned, high N_{MAX} values do not in themselves necessarily suggest high performance.

Table 3.4. Normalized Inverses of Worst Case Delays and Delay Variations for 16-Input Wide Gates, n_s = 4, for γ = 2

Decomposition Technique	Case 1		Case 2	
	$1/t_{p,MAX}$ [1]	$1/\Delta t_p$ [2]	$1/t_{p,MAX}$ [1]	$1/\Delta t_p$ [2]
Monolithic NAND	0.10	0.07	0.64	0.36
Monolithic NOR	0.18	0.12	0.52	0.29
NAND-NOR	1	1	1	1
NAND-NAND	0.63	2.56	0.53	1.10
NOR-NOR	0.71	2.93	0.48	1
NAND-INV	0.68	0.85	0.66	1.10
NOR-INV	0.75	1.02	0.55	1

Notes:

1. Normalized to $1/t_{p,NAND-NOR,MAX}$.
2. Normalized to $1/\Delta t_{p,NAND-NOR}$.

Table 3.5. Normalized Inverses of Worst Case Delays and Delay Variations for 16-Input Wide Gates, n_s = 4, for γ = 3

Decomposition Technique	Case 1		Case 2	
	$1/t_{p,MAX}$ [1]	$1/\Delta t_p$ [2]	$1/t_{p,MAX}$ [1]	$1/\Delta t_p$ [2]
Monolithic NAND	0.13	0.10	0.67	0.14
Monolithic NOR	0.15	0.11	0.50	0.28
NAND-NOR	1	1	1	1
NAND-NAND	0.66	3.44	0.54	1.15
NOR-NOR	0.66	5.93	0.47	1
NAND-INV	0.74	1.06	0.67	1.15
NOR-INV	0.69	1.05	0.54	1

Notes:

1. Normalized to $1/t_{p,NAND-NOR,MAX}$.
2. Normalized to $1/\Delta t_{p,NAND-NOR}$.

Obviously, the monolithic implementations of the wide NAND/NOR gates are unsuitable for wave pipelining. Also, for case 1, the NAND-NAND and NOR-NOR implementations result in improved delay balance over straightforward NAND-NOR implementations. For case 2, the situation is less clear; while NAND-NAND and NOR-NOR implementation may appear to give improved delay balance there as well, the improvement of 10%–15% predicted by our model should probably be discounted, since our model is only qualitatively accurate, and no clear conclusion could be drawn from such a modest difference. Thus, for case 2, we are forced to resort to SPICE simulations or similar circuit-level tools. At any case, the NAND-NOR implementation consistently gives the best (least) worst-case delays, as expected.

Also, in case 1, the NAND-INV and NOR-INV implementation appears a poor choice; it does not improve delay balance over NAND-NOR implementation, but increases worst-case delays considerably, and hence it should be avoided. On one hand, the predicted differences between $\Delta t_{p,NAND-INV}$ and $\Delta t_{p,NOR-INV}$ at one hand, and $\Delta t_{p,NAND-NOR}$ on the other, which did not exceed 20%, are insignificant enough to be ignored; thus, we cannot really confidently say that no improvement occurred at all. On the other hand, unlike in case 2, NAND-NAND and NOR-NOR decompositions are predicted to give very significant improvement in Δt_p – by a factor of 2½ or better – thus making the NAND-INV and NOR-INV decompositions irrelevant for case 1.

Finally, one more matter to note is the seemingly reasonable worst-case delay of monolithic NAND/NOR gates for case 2. While their worst-case delays appear reasonable, they may not in fact be conveniently implementable since for large number of inputs, they may require unreasonable transistor sizes. For instance, for charge mobility ratio γ = 2 and number of inputs n_m = 2, for a NOR gate, we need W_p/W_n = 4. However, for n_m = 3, we need W_p/W_n = 6, and for n_m = 16, W_p/W_n = 32. Even if we use minimum-sized NMOS, here we will need PMOS transistors 32 times minimum width. Unless the logic gates which drive such a transistor is made wide, and hence increasing energy consumption, such a design may incur excessive delays and signal rise/fall times.

3.3. Most Difficult K-Maps / Optimized Boolean Functions for Delay Equalization

An interesting question is: how bad can a Karnaugh map (K-map) or an optimized Boolean function get, as far as wave pipelining is concerned? How unbalanced can it get? Prior to answering such a question, however, we have to be able to find the worst K-map or optimized Boolean function to optimize, namely the one which, if implemented in a straightforward manner, will result in the circuit exhibiting the worst or largest difference between the longest delay and the shortest one.

It is assumed here, that the single most important factor in determining delay balance is difference of logic depths. Thus, a circuit with uniform delay balance will likely exhibit more balanced delay than one which exhibits different logic depth for each input.

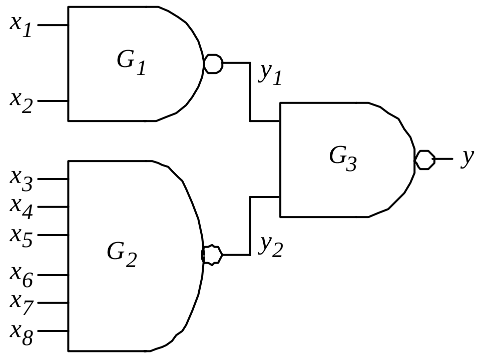


Figure 3.6. A Straightforward Two-Level Implementation of $y = x_1x_2 + x_3x_4x_5x_6x_7x_8$

Additionally, as the delay model used in section 3.2.1 shows, the delay of a logic gate depends on the number of its input. Thus, even if a circuit appears to exhibit uniform logic depth, it may still exhibit poor delay balance if it uses logic gates of widely varying number of inputs. For instance, suppose we have an eight-variable function $y = x_1x_2 + x_3x_4x_5x_6x_7x_8$, and we implement it straightforwardly as a two-level circuit of three monolithic logic gates, as shown in Figure 3.6. In the first level, we have a 2-input NAND gate G_1 and a 6-input NAND gate G_2, and in the second level, we have a 2-input NAND gate G_3. Thus, the circuit may appear to exhibit a uniform logic depth of two levels for each input. However, G_2 is likely to exhibit for greater worst-case delay and delay variation than G_1. Thus, y_2 will arrive at G_3 at a much later, and much more varying, time than y_1, resulting in a large variation in the $x_{1\ldots8} \rightarrow y$ delays. Thus, uniform logic depth does not prevent great delay variation if the logic gates used exhibit large variation in number of inputs. At any case, as previously mentioned, wide gates should also be avoided as they exhibit excessive delay variation and rise / fall times. Thus, a more precise assumption would be that *the most unbalanced K-map is one which, if implemented with wide logic gates decomposed into smaller ones, would exhibit the largest variation in logic depth*. Further, as (3.6) and (3.7) show, the wider a logic gate is, the deeper the decomposed circuit gets.

Thus, we could assume that the delay imbalance is proportional to the width difference; it could be assumed that rules of thumb:

1. The minimized two-level logic sum-of-product (SOP) expression with the worst delay balance is the one in which the difference between the number of literals in the longest product K and the number of literals in the shortest product L is the largest. The most unbalanced minimized product-of-sum (POS) expression, likewise, is the one where the difference between the number of literals in the longest and the shortest sums is the largest. Thus, the most unbalanced function is where $K-L$ is the largest.
2. The K-map with the worst delay balance is the one which, when fully minimized, would result in functions satisfying (1).

The K-map which exhibits the worst delay balance is not necessarily the worst one from a logic minimization standpoint. The worst n_i-variable K-maps to minimize are the chessboard patterned maps, which represent n_i-input XOR or XNOR function. However, such an XOR function could always be decomposed into a tree of 2-input XOR or XNOR gates, and when n_i = $2^{integer}$, the average logic depth of such a circuit will be similar for every input.

For the rest of this section, let us use the following notation:

$(x, \bar{x})$ = either x OR $\bar{x}$

$\sum x_i, \bar{x}_i$ = an OR function in which the inputs are either true or inverted;

$\prod x_i, \bar{x}_i$ = an AND function in which the inputs are either true or inverted;

As previously mentioned, in SOP form, a function with the most unbalanced delay would contain the products $\prod_{i=1}^{K} x_i, \bar{x}_i$ and $\prod_{j=1}^{L} x_j, \bar{x}_i$ with the difference in magnitude between K and L maximized. This form of function would guarantee that the difference in logic depth is maximized. Thus, we need to find the largest possible $|K-L|$. In the example given in Figure 3.6, $|K-L|$ = 4; it will turn out that it is not the largest value possible for n_i = 8.

It is obvious that $|K-L|$ < (n_i–1), as $|K-L|$ = (n_i–1) would imply that one of the two products consists of just one literal, in which case we could simplify the equation such that $|K-L|$ ≤ (n_i–2). For instance, $x_1 + \bar{x}_1 x_2 \bar{x}_3 x_4 = x_1 + x_2 \bar{x}_3 x_4$. Further, the expression $x_1 + x_2 \bar{x}_3 x_4$

cannot be simplified further. It could be shown that for any n_i, (n_i–2) is the largest depth difference possible. Further, this is true of POS expressions as well; for instance, $x_1 \cdot (\bar{x}_1 + x_2 + \bar{x}_3 + x_4) = x_1 \cdot (x_2 + \bar{x}_3 + x_4)$, which could not be further simplified.

There are also expressions of the form of (several two-literal-product)+(one n_i-literal product) and (two-literal sum1).(two-literal sum2)……. (one n_i-literal sum).

Thus, the list of "worst n-variable minimized functions" would include functions of one of the following forms.

$$y = x_1 + \prod_{i=2}^{n} x_i, \bar{x}_i \tag{3.78}$$

$$y = x_1 x_2 + \bar{x}_1 \bar{x}_2 \prod_{i=3}^{n} x_i, \bar{x}_i \tag{3.79}$$

$$y = x_1 x_2 + x_1 x_3 + x_1 x_4 + \ldots + x_1 x_{n-1} + x_1 x_n + \bar{x}_1 \bar{x}_2 \bar{x}_3 \ldots \bar{x}_{n-1} \bar{x}_n = \sum_{j=2}^{n} x_1 x_j + \prod_{i=1}^{n} \bar{x}_i \tag{3.80}$$

While in the POS form, they would be in the following three forms:

$$y = x_1 \cdot \sum_{i=2}^{n} x_i, \bar{x}_i \tag{3.81}$$

$$y = (x_1 + x_2) \cdot \left(\bar{x}_1 + \bar{x}_2 + \sum_{i=3}^{n} x_i, \bar{x}_i \right) \tag{3.82}$$

$$y = \left(\prod_{j=2}^{n} (x_1 + x_j) \right) \cdot \sum_{i=1}^{n} \bar{x}_i \tag{3.83}$$

It could be shown that the right hand side of (3.78) – (3.83) are irreducible. Thus, if they are composed in straightforward manner, not only do they maximize the difference in delay, but the difference in delay cannot be reduced by simply minimizing the logic.

Additionally, one more factor which may hinder the task of balancing delays is difference in inversion parity. The need to add inverters for certain inputs would only add to the logic depth seen by those inputs while not influencing others, and hence increasing delay imbalance. Thus, among functions of the forms given by (3.78) – (3.83), some may be more difficult to delay balance than others, depending on how many true and complemented literals appear in the right-hand side expression. However, if latency is unimportant, it is possible to rectify this problem by creating "fork" circuits, or buffering circuits which splits the inputs into their true and complemented values and attempts to equalize the arrival time of the true and complemented copies of the input. It is believed that the construction of such a driving circuit is essentially a solved problem, and therefore will not be discussed here. Readers are referred to the chapter about forks in [50] for more details. In any case, in this work, the effect of differing inversion parities of inputs on delay balance is not explored.

For large number of variables, the use of K-maps may not be helpful. Nonetheless, should they be used, the presence of "worst balance" equations is easily recognizable. For instance, with a 4-variable K-map, the following K-maps show the equations in SOP form:

x_1x_3 =	00	01	11	10
x_2x_4 = 00	0	0	1	1
01	0	1	1	1
11	0	0	1	1
10	0	0	1	1

Figure 3.7. K-map for $y = x_1 + \bar{x}_2x_3x_4$

Note that here the placement / ordering of the variables are not as usual, where we would have had x_1x_2 in one side and x_3x_4 in the other. This unusual placement is chosen to allow the variables to use similar index numbers as in (3.69). At any case, half of the map is filled with ones, and then there is a lone one in the other half. The map could have been of more or fewer variables instead. In that case, to obtain this type of expression, we will still have a contiguous block of ones filling half of the map, then a lone one elsewhere.

x_1x_3 =	00	01	11	10
x_2x_4 = 00	0	0	0	0
01	0	1	0	0
11	0	0	1	1
10	0	0	1	1

Figure 3.8. K-map for $y = x_1x_2 + \bar{x}_1\bar{x}_2x_3x_4$

Here, we have only a quarter of the map filled with ones, and there is a lone one which is not adjacent with the larger block of ones. Note that the larger block could have been a row or column instead of a quadrant.

$x_1x_3 =$	00	01	11	10
$x_2x_4 = 00$	0	0	1	1
01	0	1	0	1
11	0	0	1	1
10	0	0	1	1

Figure 3.9. K-map for

$$y = x_1x_2 + x_1\bar{x}_3 + x_1\bar{x}_4 + \bar{x}_1\bar{x}_2x_3x_4$$

There, the map is similar to the one for $x_1 + \bar{x}_2x_3x_4$, but here the block of ones has a hole in just the place which prevents the lone one from having a friend there. Since the name of variables are arbitrary, here x_3 and x_4 in (3.70) are replaced by $\bar{x}_3$ and $\bar{x}_4$, respectively.

For the POS form, the maps are identical with ones and zeros are interchanged.

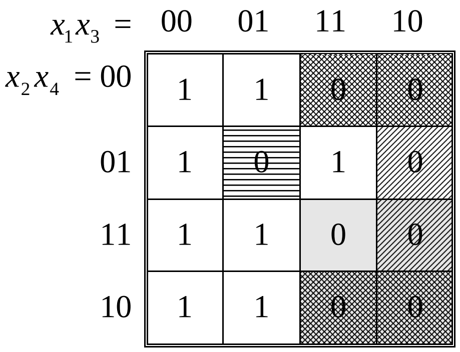
$x_1x_3 =$	00	01	11	10
$x_2x_4 = 00$	1	1	0	0
01	1	0	0	0
11	1	1	0	0
10	1	1	1	1

Figure 3.10. K-map for $y = \bar{x}_1 \cdot (x_2 + \bar{x}_3 + \bar{x}_4)$

$x_1x_3 =$	00	01	11	10
$x_2x_4 = 00$	1	1	1	1
01	0	0	1	1
11	1	1	0	0
10	1	1	0	0

Figure 3.11. K-map for

$$y = (\bar{x}_1 + \bar{x}_2) \cdot (x_1 + x_2 + \bar{x}_3 + \bar{x}_4)$$



$x_1x_3 =$ / $x_2x_4 =$	00	01	11	10
00	1	1	0	0
01	1	0	1	0
11	1	1	0	0
10	1	1	0	0

Figure 3.12. K-map for y = $(\bar{x}_1+\bar{x}_2)\cdot(\bar{x}_1+x_3)\cdot(\bar{x}_1+x_4)\cdot(x_1+x_2+\bar{x}_3+\bar{x}_4)$

The above K-maps should allow quick recognition of the "worst balanced" functions, should they be encountered in practice. However, for large number of variables, where they may not be useful and a logic minimization tool is used instead, the equations (3.69) – (3.74) should still be easily recognizable.

In section 4.2, the four decomposition techniques will be used for the worst-case K-map, in addition to for wide NAND / wide NOR gates. The wide NAND/NOR gates are the best cases as far as delay equalization is concerned, because all inputs will see identical logic depths. Thus, by simulating both wide-NAND/wide-NOR and "worst case" K-maps, we would already perform comparison for both best and worst case Boolean functions.

3.4. Example Case – Design of a Multiplier

Our past HyPipe design was a 4-bit, non-saturating, signed multiply and accumulate (MAC) block, which was designed and simulated at the netlist level in the TSMC 0.18 μm technology [45], and was found to attain 5 billion multiply-add/sec at V_{DD} = 1.8 V through HSPICE simulations. Later, a smaller 4-bit×4-bit signed multiplier was laid out in the same technology [48]. In postlayout HSPICE simulations, the multiplier attained a throughput of 4.17 billion multiplications/sec at 1.8 V and 2.33 billion mul/sec at 1.2 V.

The design of the multiplier is presented here. This section is organized as follows. We start by describing the building blocks, starting with the full adders and the flip-flops in subsection 3.4.1. Then, in subsection 3.4.2, other building blocks, which are built by simplifying the adders for some specific cases, are also described. Even though it is possible to use only a single combinational building block [45], such a practice is likely to waste chip area, and more importantly, waste energy. Therefore, several different combinational building blocks were used. Then, in 3.4.3, the design of the 4-bit signed multiplier is described. Finally, the layout technique is described in section 3.4.4.

3.4.1. Flip-flops and Full Adders

The flip-flop and the full adder circuit used for constructing the multiplier are shown in Figure 3.13. The design goals of the circuits are good delay balance and short rise/fall times, both of which are necessary to attain high operating speed for wave pipelining.

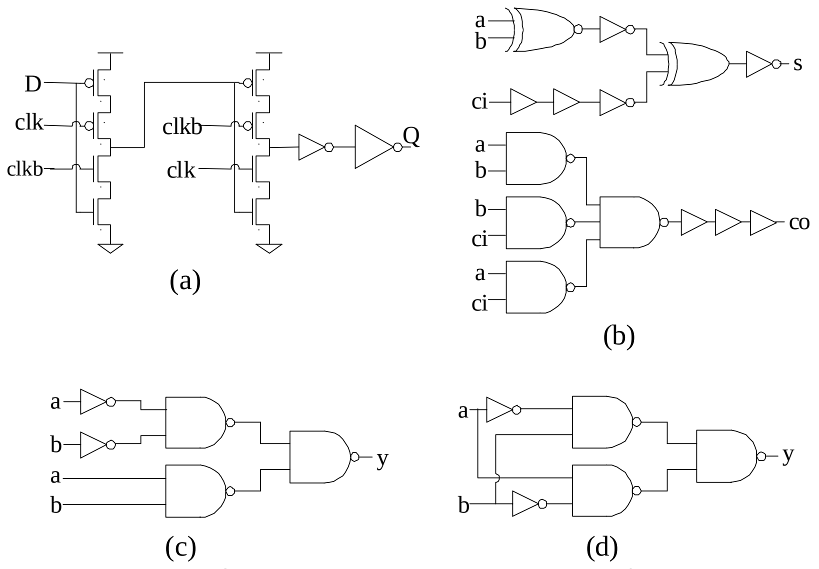


Figure 3.13. Circuit Diagrams of the Building Blocks: (a) Flip-flop with output buffer (b) Full Adder (c) 2-input XNOR gate (d) 2-input XOR gate

C^2MOS flip-flops are used due to its better immunity to the clock overlap rather than the simple dynamic flip-flops in our previous design [47], as well as its high toggle rate. The electrical effort of the flip-flop is defined as the ratio of the load capacitance to its per-pin input capacitance [49], excluding the buffer. In our case, the electrical effort is 1.2. However, a full adder poses three gate loads for the *ci* input and four for the *a* and *b* inputs. Therefore, a tapered buffer, i.e., a cascade of two inverters, is added at the output of the flip-flop. Since the inverter has a better worst-case driving strength than any other logic gates, we assign a higher electrical effort (or tapering ratio) of 2.5 to the smaller inverter. Ignoring wire loads, the output inverter will only face an electrical effort of only 4/3. This number will be higher if wireloads, which depends on high-level layout, are considered. We attempt to limit its electrical effort to no higher than about 2, even after considering wireloads. Even though the tapered buffer increases the delay of the flip-flop, it actually increases its maximum toggle frequency, as the rise/fall times of the output are reduced. Also, since the rise and fall delays of the simple buffer could be equalized easily, the buffer's presence will not decrease the potentially attainable wave number.

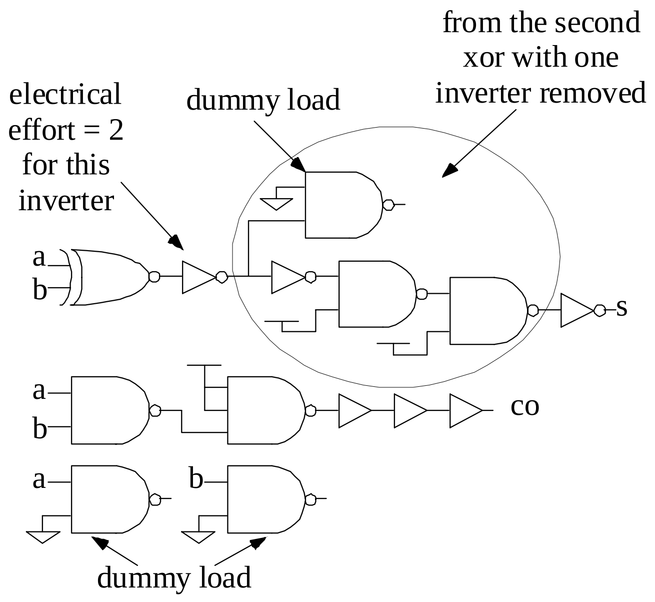
The electrical effort of all other gates is limited to one to reduce rise/fall times. Those modifications allowed us to increase the wave number of the full adder (including flip-flops at the input and output of the adder) from $N = 2$ in our previous design [47] to $N = 3$.

The XOR and XNOR gates of full adders are built using NAND gates and inverters instead of complex gates as shown in Figure 3.13 (c) and (d), since the complex gate has long output rise/fall times.

3.4.2. Other Building Blocks

In addition to flip-flops and full adders, five other combinational building blocks are needed. Those five building blocks are a normal half adder (henceforth referred to simply as "half adder"), a modified half-adder for carry-in = 1 always (hence only *a* and *b* vary), a partial product generator (i.e., AND gate), a noninverting one-stage delay line, and an inverting one-stage delay line.

Although it is possible to create a completely new circuit for each building block, this may incur additional delay variation. In our previous work [45], we sidestepped this problem by using only full adders for all those functions; hence, all CLBs in our circuit exhibited the same delay range. However, this solution incurred unnecessary circuit area and, more importantly, power penalties. For instance, when a full adder functions as a partial product generator, only a small part of the adder is actually used. The *ci* input is held to 0, the *co* output is taken as the partial product, and the sum-generating circuit is not utilized. Even so, the sum-generating circuitry continues to occupy silicon area and dissipate power, despite performing no useful function.


from the second xor with one inverter removed
electrical effort = 2 for this inverter
dummy load
a
b
s
a
b
co
a
b
dummy load
Figure 3.14. The Half-Adder

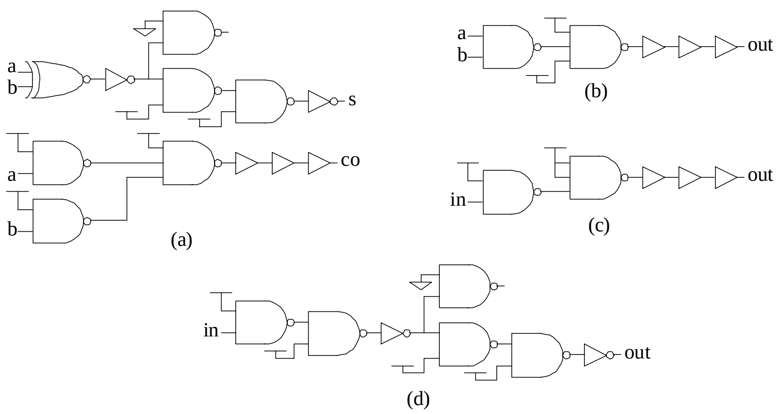


Figure 3.15. Other Building Blocks: (a) Adder for the (Carry-in = 1 always) case (b) Partial product Generator (c) Noninverting Δ Delay (d) Inverting Δ Delay

This work attempts to keep the delay variation of the CLB within the range of the delay of the full adder, without incurring the excessive area penalty of using full adders for all CLBs. This is attained by deriving the building blocks from full adders and removing unnecessary parts. For instance, the half adder is built by removing the parts which receive the *ci* input. Figure 3.14 shows a half-adder which is built by removing parts from the full adder in Figure 3.13. Since the delay of CMOS gates depends heavily on loading, the unused NAND2 is kept as a dummy load. This maintains the loading condition seen by the indicated inverter the same as in the full adder.

The second input of the dummy load is set to the controlling value, so that no additional short-circuit power is incurred. When used in a multiplier, the dummy loads at the primary inputs allow the register driving the half adder to see the same load as it does with the full adder. Figure 3.15 shows the other four building blocks, which were constructed by following the same method. Note that the dummy loads at the primary inputs are not shown; in actual use, dummy loads were added at the primary inputs such that the input capacitance of each pin equals that of four small inverters.

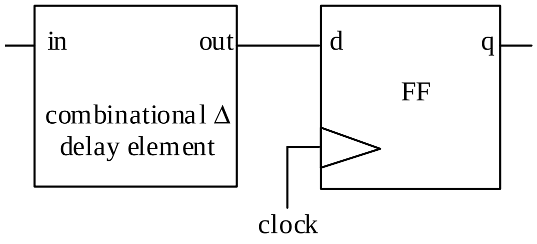


Figure 3.16. A Δ Delay Register, with Its Output Flip-Flop

The combinational Δ delay elements used in our design, which are shown in Figure 3.5. (c) and (d), merit a further note. A combinational Δ delay element is to be used in delay Δ registers with an output flip-flop as shown in Figure 3.16. The Δ delay register gives a delay of one HyPipe stage delay, namely N clock cycles, where N is the wave number. Thus, the combinational Δ delay element delays the signal by $(N - 1)$ clock cycles. In a conventional pipelined circuit, where $N = 1$, the combinational Δ delay element is not needed. For a HyPipe circuit, it is possible to use an $(N - 1)$-stage shift register instead of the combinational Δ delay elements. The shift register would increase clock power and fix N, which is undesirable for testing purposes. For instance, we could use $N = 3$ during normal operation and $N = 1$ during low-speed functional verification or simulation.

3.4.3. Complete Multiplier

This section presents the design of the 4×4 signed multiplier, which adopts a bit-level pipelined HyPipe architecture. The logic function evaluated by the proposed architecture is described as follows.

Consider two n-bit 2’s complement numbers, *A* and *B*:

$$A = -2^{n-1} \cdot a_{n-1} + 2^{n-2} \cdot a_{n-2} + ... + 2^1 \cdot a_1 + 2^0 \cdot a_0 = -S_A + L_A$$
$$B = -2^{n-1} \cdot b_{n-1} + 2^{n-2} \cdot b_{n-2} + ... + 2^1 \cdot b_1 + 2^0 \cdot b_0 = -S_B + L_B$$

In the aforementioned equations, S is the signed part of a number, e.g., $S_A = 2^{n-1}a_{n-1}$, while L is the unsigned part of the number. Then,

$$A \times B = (S_A \times S_B) + (L_A \times L_B) - (S_A \times L_B) - (S_B \times L_A) \tag{3.84}$$

Note that $(S_A \times S_B) = 2^{2n-2}a_{n-1}b_{n-1}$ and $-X = \overline{X} + 1$ for a 2's-complement number X. Hence, with $X = (S_A \times L_B) + (S_B \times L_A)$, (3.75) becomes:

$$A \times B = (a_{n-1} \cdot b_{n-1} << (2n-2)) + (L_A \times L_B) + \left(\left(\overline{(S_A \times L_B) + (S_B \times L_A)}\right) + 1\right) \tag{3.85}$$

where << is the left shift operator. Since $\overline{X + Y} = \overline{X} + \overline{Y} + 1$ for 2's complement numbers X and Y, (3.85) can be expressed as:

$$A \times B = (a_{n-1} \cdot b_{n-1} << (2n-2)) + (L_A \times L_B) + \left(\overline{(S_A \times L_B)} + \overline{(S_B \times L_A)} + 1\right) + 1 \tag{3.86}$$

The two terms, $S_A \times L_B$ and $S_B \times L_A$, have $(n - 2)$ significant bits and two leading 0s each. So, the two leading 0s have to be complemented, just the significant bits. It should be noted that Baugh-Wooley's 2's complement multiplication algorithm essentially relies on the above expression [36].

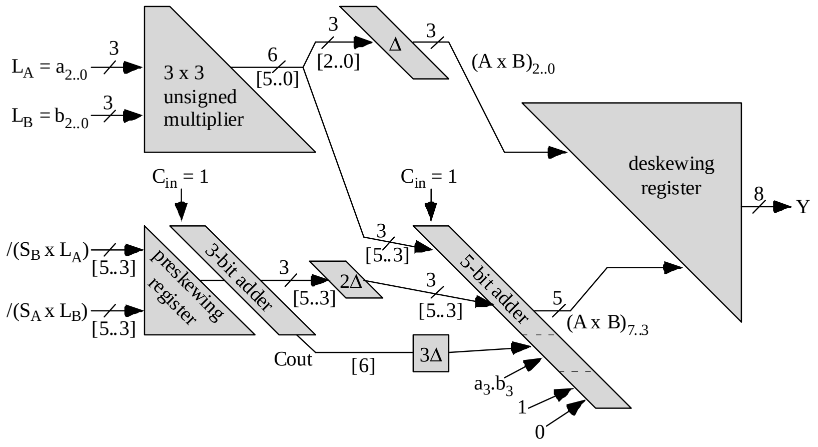


Figure 3.17. Block Diagram of the 4-bit Signed Multiplier

The architecture given in Figure 3.17 evaluates (3.86). It is composed of two ripple-carry adders, one array multiplier, one preskewing register, and three delay elements. The 3×3 multiplier computes $L_A \times L_B$, and the 3-bit adder computes the third term in (3.86). The first and the last terms are taken care of by the 5-bit adder. One MSB input of the 5-bit adder is set to 1, as the resultant MSB of the third term in (5) is 1.

Calculation of $\overline{(S_A \times L_B)}$ does not require a multiplier. Since S_A has only one possible nonzero bit, NAND gates with shifted bit positions suffice. The NAND gates are constructed with an AND stage followed by an inverting stage, using the building blocks shown in Figure 3.5.

3.4.4. Layout Technique

The proposed multiplier was laid out in the TSMC 0.18 μm technology following MOSIS SCN6M_DEEP design rules [32]. Since wave pipelining relies on tight delay balancing, the lengths of inter-building-block signal wires were limited to avoid additional delay variations due to wire loads. Further, since the highest possible throughput may require fractional wave numbers, every register stage may have to be given a separate clock. Therefore, the clock for each register stage has to be separate and easily accessible.

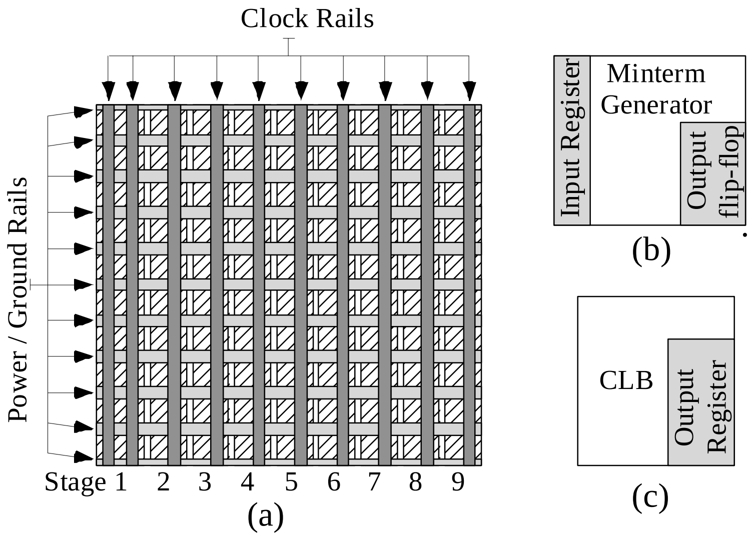


Figure 3.18. Layout Floorplans (not to scale) (a) Floorplan of the multiplier (b) Floorplan of the Partial product Generator (c) Floorplan of other building blocks

To accommodate those requirements, a levelized layout organization places each HyPipe stage in a separate column as shown in Figure 3.18. This organization follows a modified multirow standard cell-like structure with a vertical clock rail for each stage, in addition to the customary horizontal power and ground rails. Further, there is a space between any two consecutive stages to facilitate inter-stage signal routing. The building blocks within a column are placed to minimize the length of signal wires inside the multiplier. Since every CLB is laid out in a multirow standard cell style, it is sandwiched between a pair of registers. Therefore, the building block is laid out as in Figure 3.18 (b) and (c), in which a CLB is paired with an output register. The exception is the partial product generator, which is always placed in the first stage, so it is paired with both input and output registers.

Figure 3.19 shows the layout. The total layout area was 120,092.544 μm^2 (210.6 $\mu m \times$ 570.24 μm). 53,796.26 μm^2 (44.80 %) is occupied by the building blocks, while the rest is occupied by filler cells. The filler cells keep the layout shape rectangular and allow for contiguous power / ground rails, clock rails, and wells.

It is apparent that the layout method wastes a large amount of space. Additionally, the resulting layout shape is far from square with an aspect ratio of 2.71, so it may be unfavorable for some applications. However, in absence of a better layout approach, this approach was followed here. The multiplier has 10 clock rails, 64 standard cell rows, 32 V_{DD} rails, and 33 ground rails.

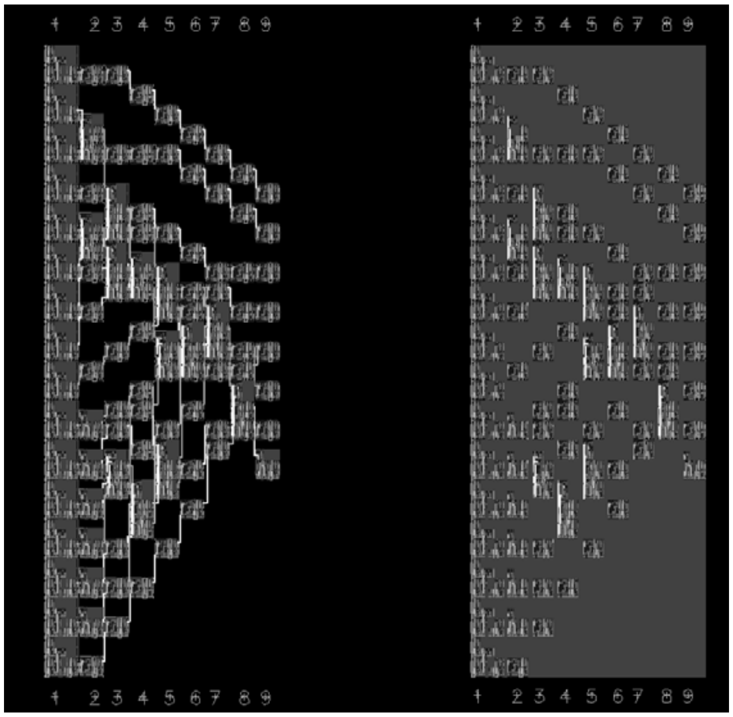


Figure 3.19. Layout of the Multiplier. Left: Without fillers. Right: With fillers

3.5. Simulation Technique and Design of Testbench

While this section focuses primarily on the design of the testbench for the multiplier in Section 3.4, here it stands as a separate section, as much of its contents also apply to simulation of any HyPipe designs in general.

The circuits designed in 3.4 – both the multipliers and the building blocks – were simulated using HSPICE in both schematic and final layout levels. For the most part, the testbenches use standard HSPICE language constructs (such as the **.measure** statement) to estimate delays and power consumptions. This section describes some general ideas related to the design of the testbenches which may not be obvious at first glance. The following two subsections discuss the design of the testbenches for the building blocks and for the complete multiplier, respectively.

While this section only discuses the simulation of the multiplier, much of the techniques followed here have to be followed during the simulation of any HyPipe based circuit.

3.5.1. Design of Testbench for the Building Blocks

Here, we will discuss the design of the testbench for the HyPipe building blocks. Specifically, for the following four matters:

- circuit arrangement for delay determination;
- clock rise/fall times;
- input patterns;
- flip-flop simulations;

Some less obvious sides of the testbench design are discussed. Both the circuit arrangement and clock rise/fall times used here are also used in the next subsection, where the complete multiplier is simulated.

Estimates for the highest attainable wave number and the corresponding allowable frequencies have to be available before the multiplier is constructed. These are obtained from simulating the building blocks. For this purpose, the minimum and maximum path delays exhibited by each type of building block have to be determined.

One way to determine the path delay D is by separately determining flip-flop setup times, clock-to-q delays, and combinational logic block delays, then summing up the three numbers.

The clock-to-q delay and the setup time sum to R. In our case, the combinational building blocks are always sandwiched between two register levels, so we use a different method as follows.

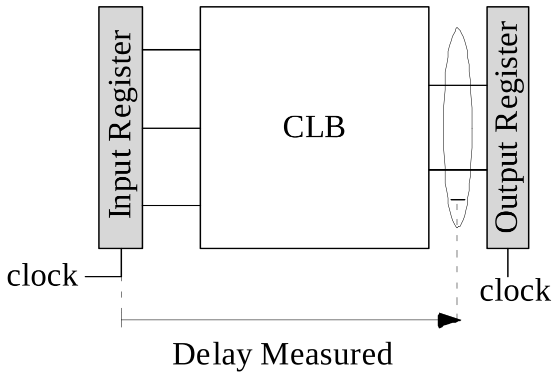


Figure 3.20. Circuit Setup for Clock-to-Combinational Output Delay Measurement. For building blocks other than partial product generator, the input register is added manually in the SPICE netlist.

First, the setup time of the flip-flop was obtained separately. Second, instead of separately finding the flip-flop's clock-to-q delay and the combinational building block delay, we found the clock-to-combinational-output delay, which is the sum of the two, as shown in Figure 3.20. This method should be more accurate for two reasons. First, it implicitly considers the effects of the input register's output waveform shape on the delay of CLB. Second, it implicitly includes the loads driven by the CLB. At the layout level, the CLB load includes wire loads, and the partial product generator load includes the wires driven by the input register. Note that the delay of a CMOS circuit is heavily influenced by the load it drives as well as by the actual input waveform shape [46]. The path length D is the sum of the clock-to-combinational-output delay and the setup time of the flip-flop. For all simulations in this work, the clock rise/fall times used were 70, 110, and 150 ps for the V_{DD} values of 1.8, 1.5, and 1.2 V, as those values approximate a lightly loaded buffer. Some circuits may drive heavier loads, but here it is assumed that the clock distribution system is designed with high drive strength to attain sharp edges and maximum speeds.

Since we have six different combinational building blocks, we have six different minimum and maximum path lengths. D_{MIN} denotes the shortest of the six minimum path lengths, and D_{MAX} denotes the longest of the six maximum path lengths.

Next, we must determine the input patterns for obtaining the delays. Since an *n*-bit signal could have only 2^n distinct values, it may be tempting to think that simply applying all the 2^n input patterns would determine the D_{MIN} and D_{MAX} of an *n*-input circuit. However, *this is not correct*, since this procedure determines only functional correctness, and not delay behavior. The delay of a CMOS circuit depends not only on the steady-state value of the input, but also on the specific input transition. Consider a 2-input NAND gate. An input pattern of 11 always causes the output to reach a steady-state value of 0, regardless of the preceding input. However, the input transitions of $00 \rightarrow 11$ and $01 \rightarrow 11$ will give different delays, even though both input transitions result in $1 \rightarrow 0$ output transition. Most likely, $00 \rightarrow 11$ will cause a longer delay, since it will necessitate both NMOS transistors in series to turn on simultaneously. The simultaneous switching of both transistors results in larger ground$\leftrightarrow$output resistance at the beginning of the switching compared with $01 \rightarrow 11$, where one of the NMOS is already ON before the input transition. Therefore, in general, to determine delays, it is not enough to just consider all possible input patterns. Instead, all input transitions may have to be considered. In the absence of *a priori* knowledge of the input transitions that cause D_{MIN} or D_{MAX}, all input transitions which cause any output changes must be used. In fact, even if a certain input transition does not change the *steady-state* value of the output, it must still be used if it causes glitches; the time it takes the output to return to the correct state has to be considered as our D.

With the aforementioned considerations in mind, all possible input transitions were used, and it is left to future work to determine the procedure for weeding out unnecessary input transitions. For an *n*-input block, we have 2^n distinct input values, each of which could be followed by $2^n - 1$ different next-input values. Hence, a series of up to $2^n \cdot (2^n - 1)$ distinct input transitions will have to be simulated. For the three-input full adder, the test vector must contain 56 distinct transitions, hence requiring a sequence of 57–112 3-bit input patterns. One advantage of the HyPipe approach is that since the circuit is broken up to small blocks, even such an exhaustive simulation for a building block could be performed with a reasonable number of test patterns.

For flip-flop testing, in addition to finding setup time, it is important to determine the *highest toggle frequency* f_{TMAX} the flip-flop could actually support. No matter how good the delay balance is, the circuit cannot operate faster than the toggle frequency. The f_{TMAX} may have to be found through iterative simulations, and it is not necessarily the same as $1/R_{MAX}$, which may serve as an initial guess. For our flip-flop, $f_{TMAX} > 1/R_{MAX}$ since the buffer at the C^2MOS flip-flop output does not change the flip-flop's ability to toggle, although it does lengthen the clock$\rightarrow$output delay.

The same toggle frequency limits apply for combinational logic gates as well. However, since our combinational gates are relatively simple, f_{TMAX} is not likely to be a concern. More complicated gates will be subject to the toggle frequency limits.

3.5.2. Design of Testbench for the Complete Multiplier

There are two issues to address in the simulation of the multiplier:

– Determination of D_{MIN} and D_{MAX} of the multiplier

– Circuit setup for power measurement

Prior to simulating the circuit with wave-pipelined operation, ***the wave number it could attain, and hence*** D_{MIN} and D_{MAX}, first have to be determined. The D_{MAX} of the multiplier is larger than the D_{MAX} of the building blocks, due to wire loads. However, the wires are relatively short, so the difference was not observed to be very significant. Hence, building block values could be used as starting values in the iterative simulations. Simulations obtain D_{MAX}, since D_{MIN} cannot be lower than the values obtained in the previous section. As the presence of inter-block wire loads increases circuit delays, there is no need to try to find the actual D_{MIN}. If the actual D_{MIN} is higher than that obtained in the previous section, the performance is better, since higher D_{MIN} will result in better delay balance, higher N, and higher throughput. We cannot go wrong by assuming the old D_{MIN}, since at worst, it will result in a lower N than the circuit's actual capabilities.

Due to the large number of distinct test patterns and possible input transitions, exhaustive simulations are impractical; therefore, only fewer test patterns were used. After D_{MIN} and D_{MAX} were determined, N_{MAX} and the related working frequency are determined. Then, simulation was performed under a desired N for the complete multiplier.

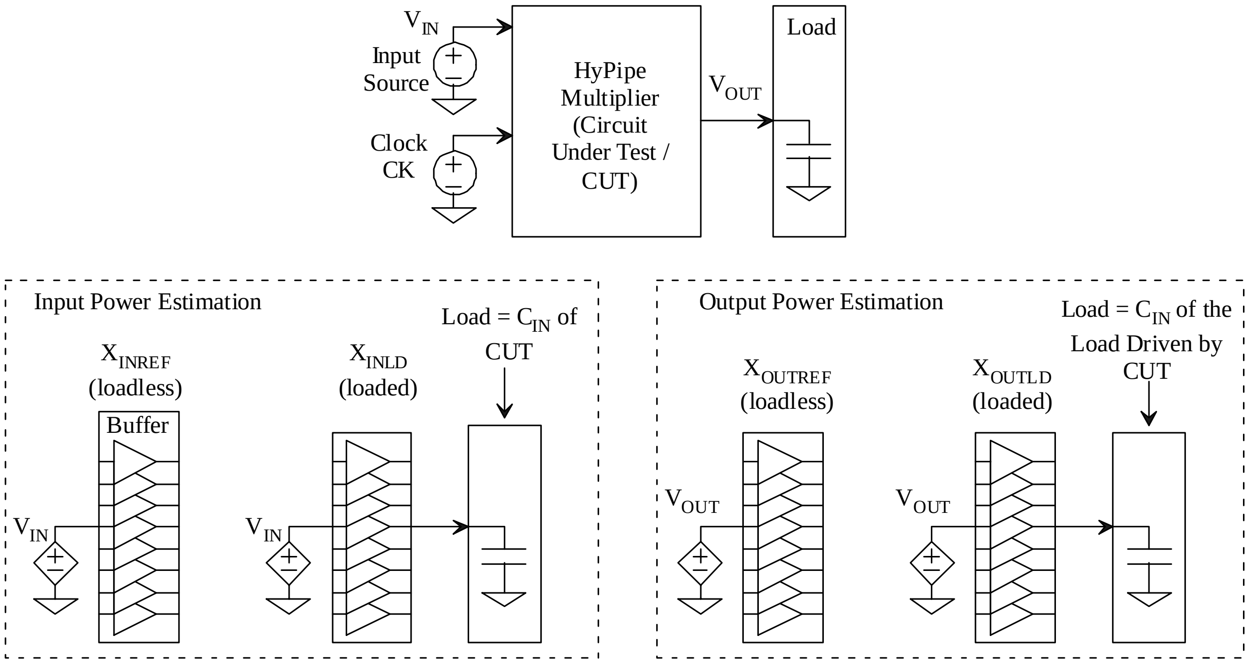


Figure 3.21. Circuit Setup for Power Measurement.

For ***power measurements***, the circuit setup is shown in Figure 3.21, and this is a more detailed view of the circuit as used in D_{MIN} and D_{MAX} determination. The basic idea is similar to the technique proposed by Stojanovic and Oklobdžija [44]. If we simply use the **.measure** statement of HSPICE to measure the energy consumption of the multiplier, the result will be inaccurate because of the following two reasons:

1. The .measure statement measures all resistive power dissipated by all transistors [2] inside the multiplier; hence, it will also measure the power of driving the load ($(1/2) \cdot CV_{DD}^2 f_{output}$). This energy should be ignored, as any circuit – not just the multiplier – driving that load will dissipate energy just because of the presence of the load. The load cannot be removed since it is necessary for simulating actual circuit conditions.
2. The .measure statement does not include the switching power at the input port of the multiplier (including clock power) as those are capacitive rather then resistive. They would appear as resistive power to the circuits which drive the multiplier (including the clock distribution circuit), if they have any output resistance. Their power dissipation will be included in the multiplier's operating power, so it should be added in.

The power consumption of the multiplier can therefore be expressed as:

$$P_{MUL} = P_{MUL\_RAW} + P_{IN} + P_{CK} - P_{OUT} \tag{3.87}$$

where P_{MUL_RAW} is the power of the multiplier as reported by HSPICE .measure statement, P_{IN} is the input power (excluding clock power), P_{CK} is the clock power, and P_{OUT} is the output power. Further, P_{IN} and P_{OUT} are calculated as follows:

$$P_{IN} = P_{XINLD} - P_{XINREF} \tag{3.88}$$

$$P_{OUT} = P_{XOUTLD} - P_{XOUTREF} \tag{3.89}$$

$$P_{CK} = \frac{\left|E_{CK\_RISE}\right| + \left|E_{CK\_FALL}\right|}{T_{CK}} \tag{3.90}$$

where E_{CK_RISE} and E_{CK_FALL} are the energy (rather than power) dissipated by rising and falling clock sources. Absolute values are used, since the clock sources are impedanceless energy sources, and HSPICE may report negative energy dissipation for a rising clock.

The above equations raise three more questions:

- First, why do we not treat P_{CK} the same as P_{IN} and P_{OUT}?
- Second, why does XINLD drive dummy loads, instead of another instance of the multiplier?
- And third, why do XOUTREF / XOUTLD consist of only buffers, and not another instance of the multiplier?

The reasoning goes as follows. First, P_{CK} is treated differently, since, except for finite rise/fall times, we assume an ideal symmetrical (skew free, 50% duty cycle always, equal rise and fall times), and repetitive clock signal. A clock waveform of such a shape is difficult to create (even in simulations) if actual clock buffers are used. Therefore, voltage sources are used instead.

For the second question, it is true that XINLD could have directly driven the multiplier. However, the use of the buffer makes it harder to time the input, as the delay of the buffer would have to be carefully calculated. Therefore, VIN would have to be timed earlier and more carefully. It is much simpler to use independent voltage sources. Also, it is difficult to design a buffer which exhibits exactly equal rise and fall delays, and this inequality exacerbates the problem. Further, simply measuring the energy expended by VIN is more difficult, since the exact timing depends on the timing of the actual input, which is neither constant nor repetitive. Also, XINLD does not drive an additional instance of the multiplier, as this would increase the size of the simulated circuit and result in impractical simulation times. Therefore, a dummy load of approximately the same input capacitance as the multiplier is used.

Third, the same reasoning applies for XOUT as well. Instead of duplicating the multiplier, only the output register needs to be duplicated. Further, only the buffer of the output registers directly drives the output signals, so only the buffer needs to be duplicated.

The power consumption of the multiplier presented in the next section thus includes all resistive power associated with typical operating conditions.

Chapter 4

Simulation Results and Discussions

Here, simulation results are presented and discussed. This chapter is divided into three parts. Section 4.1 presents and discusses the results of simulations on the alternative wide-NAND/wide-NOR decomposition techniques. It will be shown that the proposed NAND-NAND/NOR-NOR decomposition could improve delay balance compared with the straightforward NAND-NOR decomposition for the case where the width ratio of the transistors is held constant. Afterward, section 4.2 presents the results of those same decomposition techniques applied to two eight-variable "worst-case" functions. It will be shown that most of the observations made in the wide-NAND/wide-NOR decomposition experiment apply here as well.

Finally, section 4.3 presents and discusses the simulation results for the HyPipe multiplier presented in Section 3.4. The performance of the multiplier, as well as the factors limiting its performance, will be discussed. Additionally, the multiplier's performance will also be compared with the performance of the multipliers surveyed in Section 2.4. It will be shown that our HyPipe multiplier significantly outperformed other known multipliers in terms of throughput, but it lags in latency and energy efficiency.

4.1. Wide-NAND and Wide-NOR Decomposition

We decomposed wide-NAND and wide-NOR gates of four different widths, namely 3, 4, 8, and 32 inputs, following the three different decomposition techniques presented (NAND-NOR, plus NAND-NAND and NAND-INV for wide-NAND gates and NOR-NOR and NOR-INV for wide NOR gates). We also simulated monolithic implementations of 3-, 4-, and 8-input gates, and found that even with just eight inputs monolithic gates were too slow and unbalanced. In fact, its delay variation was always the largest even for four inputs.

In section 3.2.6, sample calculations were done for 16-input gates. However, since the analysis method focused on the delays and delay variations of a consecutive pair of gate stages, and by multiplying the results by the number of such two-gate stages in the decomposition, the analysis will apply to other word widths as well. The exception to this rule is the monolithic implementation, which is unimportant since, as will be shown here, it attains a poor performance.

As in Chapter 3, we use the term “case 1” for the case where we have constant W_p/W_n = 2, and “case 2” for W_p/W_n adjusted to equalize the worst-case rise and fall delays. We use the TSMC 0.18 μm technology, with all transistors having minimal length. Also, we do not make any assumptions on γ; instead, for case 2, each logic gate is designed to exhibit equal worst-case rise- and fall-delays under a unity electrical effort, which is the condition encountered in this experiment. Further, we keep $W_p + W_n$ = 1.35 μm = 15λ whenever possible (λ = 0.09 μm for our 0.18 μme technology, which requires that width are multiples of half-λ). This restriction is also followed by most of the logic gates used later in the multiplier, and it help guaranteeing that, as we assumed in section 3.2, that all logic gates used have the same C_{in}, while using only realistic transistor sizes. However, since the technology has a minimal transistor width of 0.27 μm, this requirement cannot always be satisfied. For instance, for case 2, for NOR gates where W_p/W_n = $n_i\gamma$, this width requirement cannot be satisfied for more than two inputs. In such cases, we used the narrowest transistors possible (0.27 μm) for the smallest transistors. Table 4.1 shows the resulting W_p/W_n for case 2.

Table 4.1. Transistor Widths in Multiples of λ and Width Ratios Used in Case 2

Gate Type	W_p	W_n	W_p/W_n
Inverter to be used with NOR4	12.5	4.5	2.78
Inverter to be used with NOR8	17	6.5	2.61
Inverter for Other Purposes	11	4	2.75
NAND2	10.5	4.5	1.0
NAND4	9.5	5.5	1.73
NAND8	8	7	1.14
NOR2 for use with NOR4	12.5	4.5	2.78
NOR2 for use with NOR8	17	6.5	2.61
NOR2 for other purposes	11	4	2.75
NOR4	14	3	4.67
NOR8	20.5	3	6.83

Table 4.2. Maximum Delay (in ps), Delay Variation, and Theoretical Maximum Wave Numbers for Results of Various Wide-NAND/Wide-NOR Decomposition Techniques

Gate/Decomposition technique	**Case 1**			**Case 2**		
	$t_{p,MAX}$	Δt_p	N_{MAX}	$t_{p,MAX}$	Δt_p	N_{MAX}
NAND3						
Monolithic	100	74	1.35	100	74	1.35
NAND-NOR	180	92	1.96	183	95	1.93
NAND-NAND	183	56	3.27	189	61	3.10
NAND-INV	164	69	2.38	169	71	2.38
NAND4						
Monolithic	146	121	1.21	136	110	1.24
NAND-NOR	192	110	1.75	195	113	1.73
NAND-NAND	186	70	2.66	191	73	2.61
NAND-INV	168	81	2.07	172	82	2.10
NAND8						
Monolithic	409	387	1.06	311	287	1.08
NAND-NOR	217	142	1.53	218	142	1.54
NAND-NAND	308	106	2.91	317	109	2.91
NAND-INV	271	128	2.12	279	128	2.18
NAND32						
NAND-NOR	371	251	1.48	369	245	1.51
NAND-NAND	553	176	3.14	565	177	3.19
NAND-INV	479	220	2.18	487	214	2.28
NOR3						
Monolithic	145	126	1.15	122	98	1.25
NOR-NAND	196	111	1.77	191	98	1.95
NOR-NOR	214	81	2.64	210	75	2.80
NOR-INV	192	107	1.79	185	95	1.95
NOR4						
Monolithic	222	205	1.08	177	154	1.15
NOR-NAND	195	117	1.67	189	104	1.82
NOR-NOR	215	92	2.34	211	88	2.40
NOR-INV	198	120	1.65	191	109	1.75
NOR8						
Monolithic	666	653	1.02	459	440	1.04
NOR-NAND	245	175	1.40	236	162	1.46
NOR-NOR	350	126	2.78	347	126	2.75
NOR-INV	316	183	1.73	307	170	1.81
NOR32						
NOR-NAND	402	287	1.40	390	267	1.46
NOR-NOR	615	189	3.25	617	199	3.10
NOR-INV	551	308	1.79	542	295	1.84

Table 4.2 shows the results for wide NAND gates, decomposed into circuits consisting only of inverters two-input NAND and NOR gates. For simplicity, we assume that $N_{MAX} = t_{p,MAX} / \Delta t_p$, as in 3.2.6; thus, the estimated N_{MAX} is extremely optimistic, since signal rise and fall times are ignored, and the internal node constraint of (2.22) is also ignored. Actually it is not possible to even attain $N = 1$ in a single stage of monolithic logic gate. Thus, the given N_{MAX} is simply a measure of how little the delay varies, relative to the longest delay.

At any case, as predicted in Section 3.2.6, the monolithic implementation results in worst (largest) delay variation. Additionally, it also results in the largest worst-case delay for NOR4, NAND8, or wider gates. Further, the rise/fall times reached 580 ps for NAND8 and 1.5 ns for NOR8 under case 1 (740 ps for case 2), which preclude the use of such a wide gate for GHz range. Therefore, the use of eight-input or wider NAND/NOR gates appears ill advised.

Further, for case 1, NAND-NAND decomposition of wide NAND gates, as well as NOR-NOR decomposition of NOR gates, reduces delay variation and increases N_{MAX} compared with the straightforward NAND-NOR decomposition. However, for gate widths where the decomposition becomes absolutely necessary for GHz applications (more than 4 inputs), the larger number of logical stages also increases the worst-case delay. Further, NAND-NAND and NOR-NOR decompositions give significant improvement in delay balance compared with NAND-NOR decomposition; Δt_p was shown to be reduced by a factor of ranging from 1.27 for NOR4 to 1.64 for NAND3, corresponding to reductions by 21% to 39%, respectively. However, the improvement is much less than what is predicted in Tables 3.2 and 3.3, where a Δt_p reduction by a factor of 2.6 – 3.9 was predicted. However, this difference may be simply due to the simplifications used in our model.

Also, as predicted, for case 1, where W_p / W_n is held constant, NAND-INV and NOR-INV decompositions result in as large a delay variation as NAND-NOR while increasing the worst-case delay, and hence there is no reason to employ those two techniques. So far, the predictions of section 3.2.6 were completely proven for case 1. This special case is important because it is often used; it is followed by the sea-of-gates layout styles and, frequently, also in standard-cell libraries, where the cells of the same drive strength usually have the same W_p / W_n ([51][52][53][54]).

For case 2, as predicted, all “alternative” decomposition techniques increase worst-case delays compared with the “default” NAND-NOR decomposition, just as for case 1. However, a rather surprising result was attained, namely that NAND-NAND and NOR-NOR decompositions also give the best delay balance for case 2, where our theoretical prediction was inconclusive. This surprise may reflect the inaccuracy caused by the many simplifications made in Section 3.2. However, the worst-case delays of the NAND-NOR decompositions are generally shorter than that of other decomposition techniques, particularly for wider gates. There the better latency of NAND-NOR decomposed gates may compensate for its marginally inferior delay balance.

In general, NAND-NOR decomposition do not perform better in case 2 than in case 1, relative to NAND-NAND and NAND-NOR decomposition. This is rather surprising, since theoretically the 2-input NAND and NOR gates used in case 2 are already marginally exhibiting the least delay variation, as the ranges of their rise and fall delays marginally overlap. As the formulae in section 3.2.1, Table 3.1 indicate, we could as a first-order approximation assume that for case 1, the slower transition, namely the falling transition for NAND gates and rising for NOR gates (due to serial chain of NMOS transistors of the NAND gate and serial PMOS transistors of the NOR gate) could be assumed as constant. Therefore, only the shorter delay (due to the parallel PMOS transistors of the NAND gate and parallel NMOS of the NOR gate) varies. However, for case 1, even the longest “short” delay is shortest than the shortest “long” delay. For example, for a NAND gate, even its longest rising delay than its falling delay (the opposite for NOR gates). Thus, the constant delay of the slower transition is completely outside the range of the shorter delay. Denoting the delay of the serial and parallel branches as t_{serial} and $t_{parallel,}$, for case 1, we have:

$$t_{parallel,MIN} < t_{parallel,MAX} < t_{serial,MIN} = t_{serial,MAX} \qquad (4.1)$$

Thus, the ranges [$t_{serial,MIN}$, $t_{serial,MAX}$] and [$t_{parallel,MIN}$, $t_{parallel,MAX}$] are disjoint, maximizing the delay variation which is $t_{serial,MAX} - t_{parallel,MIN}$ here. In case 2, however, the constant delay of the serial NMOS chain of the NAND (PMOS of the NOR) is placed at the upper endpoint of the variable delay of the parallel PMOS (NMOS of the NOR), hence assuring that the variable delay of the parallel transistors is, theoretically, the only source of delay variation, or:

$$t_{parallel,MIN} < t_{parallel,MAX} = t_{serial,MIN} = t_{serial,MAX} \qquad (4.2)$$

Thus, the delay variation shrinks to $\Delta t_{parallel} = t_{parallel,MAX} - t_{parallel,MIN}$ instead of $\Delta t_{parallel} + \Delta t_{serial} + t_{sep}$, where t_{sep} is the separation between the two ranges.

Therefore, in case 2, the NAND-NAND or NOR-NOR decomposition should not do better than NAND-NOR, since the delay variations per gate is already minimized. If we had used NAND-NAND or NOR-NOR decomposition, we would end up with n_s stages of NAND or NOR gates which are not functioned as inverters, all of which have the same inversion parity, and thus incur a total delay variation of $n_s \cdot \Delta t_{parallel}$. Likewise, if we had used NAND-NOR decomposition instead, we would have $n_s/2$ stages of each, and thus we end up with a delay variation of $(n_s \Delta t_{parallel}/2) + (n_s \Delta t_{parallel}/2) = n_s \cdot \Delta t_{parallel}$ as well, assuming equal delay ranges for NAND and NOR gates. Thus, for case 2, the straightforward NAND-NOR decomposition should have performed better than in case 1.

As observed, however, the NAND-NAND and NOR-NOR decomposition techniques appear to give less delay variation than the NAND-NOR decomposition for both cases. We think that this apparent anomaly is caused by the pessimistic assumption followed when deriving the estimated delay variations that for any two gate stages, there exist some transitions in the primary input which cause both stages to exhibit shortest possible delays in the opposite direction (shortest possible rising delay and shortest possible falling delay), and there also exist some transitions of primary inputs which cause both stages to exhibit longest possible delays in both direction. While the correctness of this assumption has not been formally proven for any of the decomposition methods, this is least likely to be true for NAND-NAND and NOR-NOR decompositions, since every other stage is functioned as an inverter by using only one input. Therefore, according to simple RC-delay model we have followed, such stages could only exhibit one kind of rising delay and one kind of falling delay, which reduce their delay variation. Therefore, relative to other decomposition techniques, NAND-NAND and NOR-NOR decompositions should be more likely to perform better than our models would indicate.

In the next section, we will discuss the performance of the decomposed gate if used for implementing a “bad” function as discussed in Section 3.3. In general, the results closely follow the results given here.

4.2. Worst-case K-map Decomposition

The wide-NAND and wide-NOR gates could be considered the best-case functions for the purpose of delay equalization, as they present the same logic depths for all inputs. On the opposite end of the delay balance, section 3.3 presented minimized functions and K-maps which result in greatest difference in logic depths between inputs. This section presents the simulation results of the decomposed circuits of two such worst-case functions of eight variables. Thus, it may be expected that the results closely resemble those for 8-input NAND and NOR gates from the previous section. It will be shown, however, that while the results obtained are generally similar, there are also some differences, as the inputs to the decomposed wide gates arrive at slightly differing times. Also, the last-level gate (final NAND2 for SOP functions, final NOR2 for POS function) gives further delay variation.

The functions simulated here are:

SOP form: $y = x_7 x_6 + \bar{x}_7 \bar{x}_6 \bar{x}_5 x_4 x_3 x_2 x_1 x_0$ (4.3)

POS form: $y = (x_7 + x_6) \cdot (\bar{x}_7 + \bar{x}_6 + \bar{x}_5 + x_4 + x_3 + x_2 + x_1 + x_0)$ (4.4)

Figure 4.1 shows the implementation of the circuit. We did not use any delay-equalized forks to generate the inverted inputs; thus, the inverted inputs to the wide NAND/NOR $\bar{x}_7$, $\bar{x}_6$ and $\bar{x}_5$ arrive at the wide NAND/NOR gate are late by a one-load inverter delay, or approximately 40 ps. It will be found that this variation in input arrival time hardly affects the performance and delay balance of the decomposed wide gate, and the best decomposition technique according to section 4.1 is also the best here.

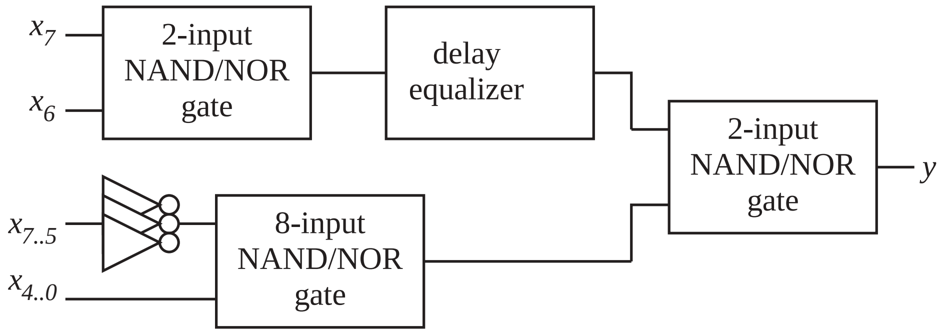


Figure 4.1. The SOP/POS Circuit.

Table 4.3. Maximum Delay (in ps), Delay Variation, and Theoretical Maximum Wave Numbers for Results of Various Decomposition Techniques for the SOP/POS Circuits

Gate/Decomposition technique	**Case 1**			**Case 2**		
	$t_{p,MAX}$	Δt_p	N_{MAX}	$t_{p,MAX}$	Δt_p	N_{MAX}
SOP Function						
1.8 V						
Monolithic	577	461	1.25	421	321	1.31
NAND-NOR	291	147	1.98	287	137	2.09
NAND-NAND	375	100	3.75	394	103	3.83
NAND-INV	339	128	2.65	341	117	2.91
1.5 V						
Monolithic	733	594	1.23	533	411	1.30
NAND-NOR	366	190	1.93	362	177	2.05
NAND-NAND	468	129	3.63	492	131	3.76
NAND-INV	423	164	2.58	425	151	2.81
1.2 V						
Monolithic	1080	891	1.21	782	617	1.27
NAND-NOR	529	283	1.87	521	267	1.95
NAND-NAND	667	193	3.46	702	192	3.66
NAND-INV	604	243	2.49	607	228	2.66
POS Function						
1.8 V						
Monolithic	727	638	1.14	569	477	1.19
NOR-NAND	294	132	2.23	297	136	2.18
NOR-NOR	407	74	5.36	452	123	3.67
NOR-INV	370	143	2.59	373	150	2.49
1.5 V						
Monolithic	948	838	1.13	727	613	1.19
NOR-NAND	370	171	2.16	370	171	2.16
NOR-NOR	512	92	5.57	569	156	3.65
NOR-INV	466	187	2.49	467	191	2.45
1.2 V						
Monolithic	1428	1272	1.23	1077	912	1.18
NOR-NAND	533	256	2.08	529	250	2.12
NOR-NOR	736	130	5.66	816	227	3.59
NOR-INV	677	284	2.38	670	281	2.38

The 8-input NAND and NOR gates were decomposed into circuits comprised by inverters and 2-input NAND or NOR gates (thus, n_i = 2 and n_m = 3). Afterward, the delays of the two branches of the circuit (the 2-input NAND/NOR and the decomposed 8-input NAND/NOR gate) were equalized by duplicating parts of the longest path of the decomposed 8-input gate and placing the 2-input NAND/NOR gate at the beginning of the path, which is the same method used in creating the building blocks of the multiplier. For the monolithic implementation, equalization is done by using 8-input gates instead of 2-input ones, and just use two inputs; thus, the delay equalizer became unnecessary. The reasoning was that for fairness, for all implementations, we have to use the same delay equalization method. For other implementations, the delays of the faster, narrower product (or sum) is done by duplicating parts of the path for the slower, wider product (sum) and using that duplicated path as the narrower product or sum. If the 8-input gate is monolithic, this can be done only by using an 8-input gate for the 2-input branch as well.

At any case, Table 4.3 gives the results. The simulations were done in three different V_{DD} values, namely 1.8, 1.5, and 1.2 V. As apparent, the NAND-NAND/NOR-NOR decompositions give the least delay variation, as was also the case in section 4.1, although for case 2, POS function, the difference (10%) is less marked than in the 8-input wide-NOR decomposition (29%). Also, the NOR-NAND decomposition of the POS function tends to perform best for case 2, POS circuit as well.

To check the possibility that some decomposition techniques may have been more heavily influenced by the input-inverter delays than others, the experiment was repeated with $\bar{x}_7$, $\bar{x}_6$ and $\bar{x}_5$ generated at the same time as x_7, x_6 and x_5 using dependent voltage sources, instead of being late by an inverter delay; in the actual circuit design, delay equalized forks as proposed by Harris, Sutherland and Sproull [50] may have to be utilized to attain this effect. Table 4.4 gives the result. It was found that the same observations hold here as well.

We conclude that while the NAND-NOR gives the shortest latency, the NAND-NAND and NOR-NOR decompositions give the least delay variation. Therefore, the choice between the two depends on the application. For wave pipelining, the NAND-NAND and NOR-NOR decomposed gates' superior delay balance, and hence potentially superior wave pipelined throughput, may compensate for their longer latencies.

Table 4.4. Maximum Delay (in ps), Delay Variation, and Theoretical Maximum Wave Numbers for Results of Various Decomposition Techniques for SOP / POS Circuits without Input Inverters

Gate/Decomposition technique	Case 1			Case 2		
	$t_{p,MAX}$	Δt_p	N_{MAX}	$t_{p,MAX}$	Δt_p	N_{MAX}
SOP Function						
1.8 V						
Monolithic	575	467	1.23	419	332	1.26
NAND-NOR	264	134	1.97	263	130	2.02
NAND-NAND	359	102	3.52	365	100	3.65
NAND-INV	324	125	2.59	328	120	2.73
1.5 V						
Monolithic	735	609	1.21	530	424	1.25
NAND-NOR	334	174	1.92	332	167	1.99
NAND-NAND	449	130	3.45	457	127	3.60
NAND-INV	404	160	2.53	410	154	2.66
1.2 V						
Monolithic	1092	923	1.18	784	641	1.22
NAND-NOR	484	260	1.86	481	251	1.92
NAND-NAND	639	188	3.40	652	186	3.51
NAND-INV	576	236	2.44	585	230	2.54
POS Function						
1.8 V						
Monolithic	480	479	1.00	564	484	1.17
NOR-NAND	291	148	1.97	289	147	1.97
NOR-NOR	399	100	3.99	404	112	3.61
NOR-INV	367	160	2.29	366	161	2.27
1.5 V						
Monolithic	766	764	1.00	723	624	1.16
NOR-NAND	368	191	1.93	364	189	1.93
NOR-NOR	502	119	4.22	503	135	3.73
NOR-INV	464	206	2.25	459	207	2.22
1.2 V						
Monolithic	1244	1242	1.00	1072	933	1.15
NOR-NAND	540	295	1.83	529	287	1.84
NOR-NOR	723	168	4.30	714	184	3.88
NOR-INV	677	316	2.14	661	308	2.15

4.3. HyPipe Multiplier

Here, the results of the simulations of the multiplier are presented and discussed. Additionally, some comparison with the previously built multipliers surveyed in Chapter 2 will be made.

The simulations were performed at three V_{DD} values, namely 1.8, 1.5, and 1.2 Volts. As stated in Section 3.3 regarding testbench design, the first simulation performed was the simulation of the building blocks. The simulations were performed for netlists generated from both schematics and layout; however, only results from layout simulations will be presented here. Table 4.5 tabulates the setup times of the flip-flops for various supply voltages. Table 4.6 tabulates the sum of flip-flops clock$\rightarrow$q delays and combinational delays. The shortest T_{CK} possible is calculated using the data and equation (2.19), while the frequency range for $N = 3$ is obtained with (2.26). Table 4.7 shows the same tables in a condensed form. Here, the frequency ranges shown are for $N = 3$; such an operation appears feasible.

Table 4.5. Setup Times of the Flip-Flop under Various Supply Voltages

Supply Voltage (V)	**Setup Time (ps)**	
	Minimum	**Maximum**
1.8	29	39
1.5	43	51
1.2	73	75

The row on "T_{CKMIN}**, ps, ignoring** $T_{rf,MAX}$" is simply an illustration on the effect of signal rise/fall times on the attainable clock speed; there, $T_{rf,MAX}$ is ignored in using (2.19). As apparent, the effects of rise/fall times are substantial; if the rise/fall times were negligible, the maximum clock frequency attained could be increased by 40%–52%. Evidently, the signal rise/fall times cannot be ignored.

Table 4.6. K (=C+$T_{CK\rightarrow Q}$) and Estimated Allowable Clock Periods for N=3 for Each Building Block

Supply Voltage (V)	*K* Range (ps)	Maximum T_{rf} (ps)	Shortest T_{CK} possible (ps)	*N*=3 Clock Period Range (ps)
Full Adder (FA)				
1.8	557-639	63	184	226-293
1.5	695-799	73	228	283-369
1.2	987-1137	89	314	404-530
Half Adder				
1.8	568-628	60	159	222-299
1.5	708-785	71	189	279-376
1.2	1004-1114	95	280	396-539
Adder, C_{in} = 1 always				
1.8	582-632	61	150	224-306
1.5	726-790	71	186	280-385
1.2	1030-1123	96	264	399-552
Partial Product Generator				
1.8	556-615	51	149	218-293
1.5	694-768	61	189	273-369
1.2	985-1090	83	263	388-529
Noninverting Delay				
1.8	569-593	52	118	211-299
1.5	710-742	59	142	264-377
1.2	1011-1053	79	196	376-542
Inverting Delay				
1.8	573-584	47	97	208-301
1.5	714-729	57	123	260-379
1.2	1016-1041	90	190	372-545

Table 4.7. D_{MAX} and f_{CK} Estimates from Building Block Simulations

V_{DD}	**1.8 V**	**1.5 V**	**1.2 V**
$D_{MIN/MAX}$ **, ps**	585-678	737-850	1058-1212
$T_{rf,MAX}$ **, ps**	63	73	96
T_{CKMIN} **, ps**	184	228	314
T_{CKMIN} **, ps, ignoring** $T_{rf,MAX}$	121	155	225
$T_{CK}\big\|_{N=3}$ **, ps**	216-293	283-369	404-529
Freq. Range, *N* = 3 (GHz)	3.41-4.63	2.71-3.53	1.89-2.47

As mentioned earlier, the actual attainment of such high working frequencies as predicted in Table 4.7 is contingent upon the flip-flops being actually capable of at least toggling at that frequency. Table 4.8 lists the f_{TMAX} of the flip-flop at those four frequencies. The toggle period is obtained from directly simulating the flip-flop itself, and not from its setup time or clock-to-output delay.

Table 4.8. Maximum Toggle Rate of the Flip-Flop

V_{DD}	**1.8 V**	**1.5 V**	**1.2 V**
Minimal Toggle Period, ps	212	268	395
f_{TMAX} **(GHz)**	4.72	3.73	2.53

Table 4.6 suggests that the register and the full adder pair could operate in a reasonably broad range of clock frequency for each supply voltage at $N = 3$, while Table 4.5 shows that the flip-flop actually functions properly at those particular frequencies. Since the flip-flop is barely fast enough to function properly at the upper frequency limits for $N = 3$, no simulations for higher wave numbers were attempted.

It is interesting to note that for all three V_{DD} values, f_{CK} is limited not by path delay variation, but by flip-flop toggle speeds. It was found that the reason for this limitation was the slow signal transitions in the internal nodes inside the flip-flop. Figure 4.2 shows the master-slave structure of the C^{2}MOS flip-flop. The output node of the master latch appears to be the speed-limiting node. Figure 4.3 shows the snapshot of simulation waveforms of some of the nodes when the flip-flop toggles at the highest frequency possible.

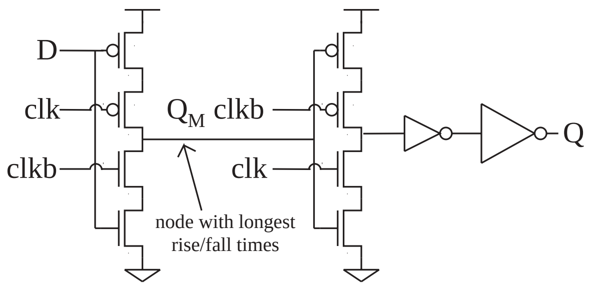


Figure 4.2 The C^{2}MOS Flip-Flop, Showing Master Latch Output Q_M.

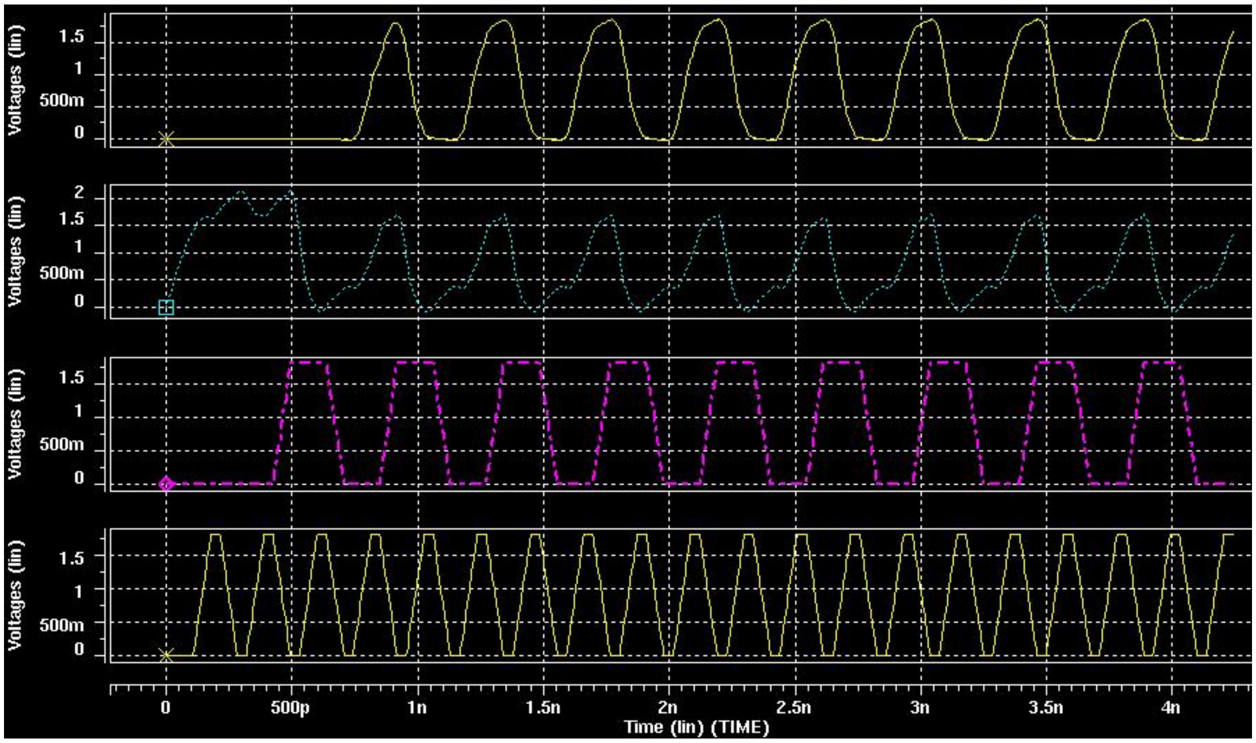


Figure 4.3 From Top to Bottom: Flip-flop Output Q; Master Latch Output Q_M, Input D, and Clock.

It is apparent that the rise time of Q_M limits the speed of the C^2MOS flip-flop. It is not clear how the effect of this rise time toward operation speed is to be accounted for, or even how to define rise time or fall time. This T_{rf} of Q_M cannot be accounted for as T_{rf} in (2.19) or (2.22), because it is inside the flip-flop. It may not be meaningful to define T_{rf} as 10%-to-90% or any other fixed percentage, because $V(Q_M)$ does not swing rail-to-rail, thus 10% and 90% of V_{DD} are not 10% and 90% of its swing. However, if we define rise time, which is longer than fall time, as the shortest duration possible between the minimum and maximum voltages (trough-to-summit rise time), Table 4.9 is obtained. As seen, the flip-flop is capable of running properly at shorter clock period than the T_r of Q_M! Thus, the rise time of Q_M could exceed T_{CK}, and Q_M is allowed to not have completed its rise when the next cycle comes, as long as the new clock cycle is long enough to allow it to both complete that rise transition and complete a new falling transition as well. This phenomenon is quite interesting, but apart from suggesting the reason for the limited speed of the flip-flop, a detailed analysis is beyond the scope of our work.

Table 4.9. Risetime of Node Q_M and Maximum Toggle Rate of the Flip-Flop

V_{DD}	**1.8 V**	**1.5 V**	**1.2 V**
T_r of Q_M **, ps**	308	400	590
T_{CKMIN} **, ps**	212	268	395

Afterward, simulations were performed for the multiplier. The simulations, which were performed for the same three V_{DD} values as the building block simulations, were performed in two steps. In the first step, it was run with $N = 1$, both to verify its functional correctness and to estimate its D_{MAX}. It was found to function properly by simulating it at significantly lower clock speed than its maximum attainable clock frequency f_{MAX} for $N = 1$; clock frequencies of 1.25 GHz, 833 MHz, and 500 MHz were used at 1.8, 1.5, and 1.2 Volts, respectively. The simulation for D_{MAX} needed iterations, and the relation $D = 1/ f_{MAX}\big|_{N=1}$, where $f_{MAX}\big|_{N=1}$ is assumed to be the highest frequency where the multiplier is found to function properly at $N = 1$, is assumed. This new D_{MAX} replaces the D_{MAX} estimate obtained from simulation of the building blocks. Table 4.10 gives the revised estimates of operable frequencies for $N = 3$.

Table 4.10. D_{MAX} **and** f_{CK} **Estimates from Multiplier Simulations**

V_{DD}	**1.8 V**	**1.5 V**	**1.2 V**
D_{MAX} **, ps**	684	854	1218
$T_{CK}\big\|_{N=3}$ **, ps**	228-293	285-369	406-529
Freq. Range, $N = 3$ (GHz)	3.41-4.39	2.71-3.51	1.89-2.46

As apparent, the circuit is theoretically still capable of operating at $N = 3$. It is apparent also that although the maximum operable frequency does decrease slightly compared with the building block simulation results, the differences are not very significant (5.2%, 0.6%, and 0.4% respectively, at the three V_{DD} values), which perhaps signify that the inter-building-block wires posed surprisingly little load. It must be noted, however, that the multiplier simulation was not exhaustive, and the input sequence used might not have caused the worst case delay to occur.

The second step of the multiplier simulation was the actual main simulation, at $N = 3$. It was decided that, instead of trying to find the actual $f_{MAX}\big|_{N=3}$, an operating frequency was instead chosen somewhat arbitrarily for each V_{DD}; the chosen f_{CK} were deliberately chosen closer to $f_{MAX}\big|_{N=3}$ instead of to $f_{MIN}\big|_{N=3}$, to improve the chance that it was perhaps the highest frequency attainable. The chosen operating frequencies were 4.167, 3.33, and 2.33 GHz, respectively. It was found that at those frequencies, the multiplier functions correctly. Table 4.11 tabulates the simulation results. For the calculation of EDP and ED^2P, the corresponding clock periods (240, 300, and 429 ps) are taken as our D_{MAX}. Here, theoretical $f_{CK,MAX}$ is attained from T_{CKMIN} in Table 4.7.

Table 4.11. Performance of the Multiplier at *N* = 3

V_{DD}	**1.8 V**	**1.5 V**	**1.2 V**
f_{CK} **, GHz**	4.17	3.33	2.33
Percentage of Theoretical $f_{CK,MAX}$	76.7	76.0	73.2
D_{MAX} **,** $\times FO4$ **delays**	2.52	2.56	2.63
Power, mW	76.17	41.08	18.18
EDP **,** $\times 10^{-21}$ **J.s**	4.39	3.70	3.35
ED^2P **,** $\times 10^{-30}$ **J.s^2**	1.05	1.11	1.44
Latency, ns	6.72	8.40	12.00

As apparent, the multiplier reaches a lower D_{MAX} than any of the multipliers reviewed in Section 2.4; it reaches a throughput of 2.52 – 2.56 $FO4$ delays, compared with 4.61 $FO4$ delays attained by the highest-throughput design [22]. Furthermore, neither the EDP nor the ED^2P of the proposed multiplier is independent of V_{DD}. As previously mentioned, the ED^2P has been shown to be independent of V_{DD} for a circuit under any $V_{DD} > \max\left(V_{tn}, \left|V_{tp}\right|\right)$ assuming long-channel CMOS model [31]. However, it is perhaps not surprising that it is in fact V_{DD} dependent, since at 0.18 μm short-channel models have to be assumed. Unfortunately, to the author's knowledge, there is not yet any comparable metrics for short-channel technologies in the open literature as of now.

Also, it is interesting that the throughput of the multiplier reaches about three-fourth of the theoretical maximum for our circuit. While similar figure is not available for other designs, we surmise that this percentage may be the highest achieved so far.

To see what those figures really mean, we compare those figures with those attained in previously published work, as surveyed in Section 2.4. Table 4.12 shows the comparison of the minimum clock period of the HyPipe multiplier with the top five multipliers previously reported. Clearly, the HyPipe multiplier enjoys a significant advantage.

Table 4.12. Comparison of Throughputs – HyPipe vs. Earlier Works

Ref.	Year	Pipeline Type	$T_{CK,MIN}$ (in FO4 delays)	$f_{CLK,MAX}$ (MHz)	Feature Size (μm)	Estimated FO4 Delay (ps)	Input Word Length
Our Work	2005	HyPipe	**2.52**	4166.7	0.18	95	varies[1]/4
[22]	1996	conventional	**4.61**	482.5	1	450	4
[40]	2001	wave	**4.92**	1612.9	0.35	126	8
[17]	2001	conventional	**5.3**	715	1	264	8
[10]	1995	wave	**5.56**	625	0.8	288	8
[21]	1994	wave	**6.35**	350	1	450	16

Note: 1. Design in [22] accepts varying input widths.

Table 4.13. Latencies and Latency Rankings of the Highest-Throughput Multipliers

Ref.	Year	Pipelines	Input Word Length	$f_{CLK,MAX}$ (MHz)	$T_{CK,MIN}$ (in FO4 delays)	Normalized Latency (FO4)	Latency Rank
Our Work	2005	HyPipe	**4**	4166.7	**2.52**	**70.65**	22
[22]	1996	conventional	varies[2]/4	482.5	**4.61**	**29.97**	16
[40]	2001	wave	8	1612.9	**4.92**	**20.63**	13
[17]	2001	conventional	8	715	**5.3**	**38.64**	19
[10]	1995	wave	8	625	**5.56**	unknown	unknown
[21]	1994	wave	16	350	**6.35**	**12.7**	5
[55]	2000	conventional	8	625	**7.41**	**62.96**	21
[7]	2002	conventional	8	1000	**7.94**	**59.52**	20
[1]	1995	conventional	16	50	**15.38**	**153.85**	23
[19]	2002	conventional	54	1538.5	**15.7**	**19.08**	12
[35]	2001	conventional	32	444.4	**17.86**	**14.29**	6
[30]	1996	conventional	56	285.7	**19.44**	**30.23**	17

Notes:

1. If the size is not 4-bit×4-bit and not pipelined, its latency D is normalized further according to $D \propto m + n$ for array multipliers and $D \propto \log_2(m \times n)$ for tree multipliers.
2. Design in [22] accepts varying input widths.

Table 4.13 shows the same table, but with the latencies of those multipliers shown as well, as in Table 2.3. While our multiplier leads in throughput, it is inferior in latency.

Table 4.14. Latencies of the HyPipe Multiplier and Other Multipliers Surveyed

Ref.	**Yr.**	**Pipe-line**	**Input Size (bits)**	**Feature Size (μm)**	**Estimated FO4 delay (ps)**	**Laten cy (ns)**	**Latency (FO4)**	**Normalized Latency (FO4)**
[34]	1996	none	24	1	450	10.5	23.33	**10.18**
[12]	1998	conv.	54	0.25	90	2.7	30	**10.43**
[58]	1990	none	16	0.5	180	3.8	21.11	**10.56**
[37]	1993	conv.	16	1	450	11	24.44	**12.22**
[21]	1994	wave	16	1	450	11.4	25.4	**12.7**
[35]	2001	conv.	32	0.35	126	4.5	35.71	**14.29**
[25]	2001	unkn.	64	0.25	90	4	44.44	**14.81**
[12]	1997	none	54	0.25	90	4.1	45.56	**15.83**
[20]	2001	conv.	54	0.18	65	3.16	48.62	**16.9**
[29]	1996	none	54	0.5	180	8.8	48.89	**16.99**
[33]	1995	unkn.	54	0.25	90	4.4	48.89	**16.99**
[19]	2002	conv.	54	0.13	41.4	2.27	54.9	**19.08**
[40]	2001	wave	8	0.35	126	3.9	30.95	**20.63**
[24]	2002	none	54	0.25	90	6.9	76.67	**26.64**
[23]	2001	none	16	0.25	90	5.14	57.11	**28.56**
[22]	1996	conv.	vary[1]/4	1	450	13.47	29.95	**29.95**
[30]	1996	conv.	56	0.5	180	15.8	87.78	**30.23**
[39]	2002	conv.	59	0.13	41.4	4	96.62	**32.85**
[17]	2001	conv.	8	1	264	15.3	57.95	**38.64**
[7]	2002	conv.	8	0.35	126	15	119.05	**59.52**
[55]	2000	conv.	8	0.6	216	27.2	125.93	**62.96**
Our Work	2005	HyPipe	4	0.18	95	6.72	70.65	**70.65**
[1]	1995	conv.	16	1.2	1300	400	307.69	**153.85**

Note: *Pipeline types:* unkn. = unknown, conv. = conventional pipeline, wave = wave pipeline. Design in [22] accepts varying input widths. If the size is not 4-bit×4-bit and not pipelined, its latency D is normalized further according to $D \propto m + n$ for array multipliers and $D \propto \log_2(m \times n)$ for tree multipliers

Table 4.14 gives a similar comparison for latency. Unlike before, all other multipliers are shown as well. As appears here, latency wise, the HyPipe multiplier is quite poor – hence suggesting that the great throughput was essentially obtained at the expense of poor latency. Although in theory it may appear that the latency should not be so great, as the number of register stages (9) is no more than what would be exhibited by a conventional-pipelined 4-bit multiplier, the design of the CLB may not be optimal for latency. The reason is that for wave pipelining to attain best throughput, it is imperative that the best delay balance be achieved. Although the CLB achieves good delay balance, it apparently also exhibits poor latency.

Table 4.15. Energy Efficiencies and Power Dissipations of HyPipe Multiplier and Other Multipliers Surveyed

Ref.	Year	Type / Pip'd	Max Clock Speed (MHz)	ED^2P, $\times10^{-30}$ J.s.	EDP, $\times10^{-21}$ J.s.	Power (mW)	At Clock Speed (MHz)	Input Word Length	Feature Size (μm)
[8]	2003	Tree, Y	1200	**0.04**	0.09	16.81	1000	13	0.35
[22]	1996	Serial, Y	482.5	**0.12**	0.31	69	482.5	4[5]	1
[55]	2000	Array, Y	625	**0.27**	0.57	52.4	300	8	0.6
[35]	2001	Tree, Y	1612.9	**0.35**	1.08	150	1500	8	0.35
[21]	1994	Tree, Y	350	**0.44**	0.85	1360	300	16	1
[7]	2002	Tree, Y	1000	**0.90**	1.76	100.52	1000	8	0.35
Our Work	2005	Array, Y	4166.7	**1.05**	4.39	76.17	4166.7	4	0.18
[29]	1996	Tree, N	114	**2.72**	1.94	540	100	54	0.5
[11]	1997	Tree, N	244	**7.84**	5.99	2.23	1	54	0.25
[24]	2002	Tree, N	145	**11.06**	5.01	111	100	54	0.25
[30]	1996	Tree, Y	285.7	**13.49**	10.71	5.1	1	56×8	0.5

Notes:

1. EDP and ED^2P are normalized to the estimated values for an 0.18 μm implementation, according to formulae (2.3)-(2.11)
2. If the size is not 4-bit×4-bit, energy per operation is normalized down further to 4-bit×4-bit by assuming that for an *m*×*n*-bit multiplier, $E \propto m \times n$
3. If the size is not 4-bit×4-bit and not pipelined, its clock period D is normalized further according to $D \propto m + n$ for array multipliers and $D \propto \log_2(m \times n)$ for tree multipliers
4. [31] multiplies a 32-bit pixel data with a 32-bit FP, using a 56×8 multiplier (Booth recoded, possibly tree topology)
5. Design in [22] accepts varying input widths.

Table 4.15 compares the EDP and ED^2P of the HyPipe multiplier with those achieved by other multipliers as normalized to the same word with and technology, ranked according to their ED^2P. For the HyPipe multiplier, only the figures for 1.8 V are used, under the assumption that most multipliers will likely be operated under their highest rated V_{DD}. As apparent, the energy efficiency of the HyPipe multiplier is poorer than that of most other pipelined multipliers announced in open literature. Obviously, energy efficiency wise, the HyPipe multiplier's high operating speed does not compensate for the very high power consumption. It may well be that the large transistor count associated with static CMOS technology, as well as heavy use of dummy loads in the CLBs, have caused large number of switching and high capacitive loads.

It is also possible that the use of large number of delay / deskewing elements, and the use of combinational circuits (i.e., delay buffers) as the deskewing elements, may have contributed significantly to the high power consumption. While work in [38], for instance, also employs preskewing and deskewing registers, they are no more than flip-flops, with no combinational elements between successive flip-flops. The HyPipe multiplier proposed, due to the potential desirability of operation at two different N (1 for functional testing and 3 for normal operation) uses combinational circuits in addition. However, the presence of those combinational logic gates caused additional power consumption.

In conclusion, HyPipe in current implementation results in much superior throughput over other techniques, but the resulting latency is inferior, and its energy efficiency is also poor compared to conventional pipelining, although still better than non-pipelined designs. Therefore, it is left to future work to determine whether it is possible to improve either of those two shortcomings. At least, we could already suggest that its main goal of superior throughput is attained.

Chapter 5

Conclusions

Pipelining is a method for increasing the throughput of a circuit by dividing its critical path into shorter sections. Two synchronous methods are known from open literature, namely conventional pipelining and wave pipelining. We introduced a third, hybrid method, called HyPipe, which attempts to combines the two to attain the best throughput possible. Additionally, since delay balance is necessary for wave-pipelined circuits to attain high performance, we attempted to find a method to decompose a class of circuit, namely wide-NAND and wide-NOR gates, into the circuit with the least delay variation possible.

In the main part of our work, it was shown that HyPipe pipelining resulted in much superior throughput compared with other methods. However, it also resulted in poor latency and energy efficiency. Whether the throughput and, more importantly, energy efficiency of HyPipe circuit could still be improved is left to future works. At least, its main aim of the highest throughput possible appears attained.

We also proposed alternative methods to decompose wide-NAND and wide-NOR gates, which were expected to be better than the straightforward NAND-NOR decomposition, and showed that they attain a better delay balance than the straightforward decomposition method, particularly when the width ratio of PMOS to NMOS transistors is held constant. Therefore, they should facilitate the use of an automatic tool to construct a HyPipe circuit from sea-of-gates layout or a standard cell library. We leave such a possibility to future works and researchers.

References:

[1] W. Amendola Jr., H. R. Srinivas, and K. K. Parhi, “A 16-bit×16-bit 1.2 μ CMOS Multiplier with Low Latency Vector Merging”, *Proceedings of the 8th International Conference on VLSI Design*, Jan 1995, pp. 398 – 402

[2] Avant! Corp, *Star-HSpice Manual, Release 2000.2*, Chapters 20 – 21, Avant! Corp, Fremont, CA, May 2002

[3] P. A. Beerel, “Asynchronous Circuits: An Increasingly Practical Solution”**,** *Proceedings of the 2002 International Symposium on Quality Electronic Design*, Mar 18-21, 2002, pp. 367–372

[4] W. P. Burleson, M. Ciesielski, F. Klass, and W. Liu, “Wave Pipelining: A Tutorial and Research Survey”, *IEEE Transactions on VLSI*, vol. 34 No. 3, September 1998, pp. 464 – 474

[5] W. P. Burleson, L. W. Cotten, F. Klass, and M. Ciesielski, “Wave-pipelining: Is It Practical?”, Proceedings of the IEEE International Symposium on Circuits and Systems, vol. 4, pp. 163 – 166, 1994

[6] A. P. Chandrakasan and R. W. Brodersen, “Minimizing Power Consumption in Digital CMOS Circuits”, *Proceedings of the IEEE*, vol. 83 issue 4, Apr 1995, pp. 498–523

[7] H. –C. Chow and I-C. Wey, “A 3.3V 1GHz High Speed Pipelined Booth Multiplier**”,** Proceedings of the 2002 IEEE International Symposium on Circuits and Systems, vol. 1, pp. 457 – 460

[8] H. –C. Chow and I-C. Wey, “A 3.3V 1 GHz Low-Latency Pipelined Booth Multiplier with New Manchester Carry-Pass Adder**”,** Proceedings of the 2003 IEEE International Symposium on Circuits and Systems, vol. 5, pp. V-121 – V-124, 2003

[9] V. V. Deodhar and J. A. Davis, “Optimization of Throughput Performance for Low-Power VLSI Interconnects”, *IEEE Transactions on VLSI*, vol. 13 issue 3, Mar 2005, pp. 308 – 318

[10] D. Ghosh and S. K. Nandy, “Design and Realization of High-Performance Wave-Pipelined 8×8 b Multiplier in CMOS Technology”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 3 issue 1, Mar 1995, pp. 36 –48

[11] G. Goto, A. Inoue, R. Ohe, S. Kashiwakura, S. Mitarai, T. Tsuru, and T. Izawa, “A 4.1-ns Compact 54×54-b Multiplier Utilizing Sign-Select Booth Encoders”, *IEEE Journal of Solid-State Circuits*, vol. 32 issue 11, Nov 1997, pp. 1676 –1682

[12] Y. Hagihara, S. Inui, A. Yoshikawa, S. Nakazato, S. Iriki, R. Ikeda, Y. Shibue, T. Inaba, M. Kagamihara, and M. Yamashina, “A 2.7 ns 0.25 μm CMOS 54×54 b Multiplier”, *Digest of Technical Papers, 45th IEEE International Solid-State Circuits Conference*, 5-7 Feb 1998, pp. 296 –297

[13] S. Heo and K. Asanovic, “Power-Optimal Pipelining in Deep Submicron Technology”**,** *Proceedings of the 2004 International Symposium on Low-Power Electronics and Design (ISLPED ’04)*, Aug 9-11, 2004, pp. 218-223

[14] M. Horowitz, “Life After Silicon: An Oxymoron?”, http://www.cs.wisc.edu/~arch/www/ISCA-2000-panel/Mark.Horowitz.isca2000.ppt, Stanford University, 1999

[15] M. Horowitz, R. Ho, and K. Mai, “Wires: A User’s Guide”, http://www.stanford.edu/papers/rh_srcmacro_99.pdf, Stanford University, 1999

[16] M. Horowitz, T. Indermaur, and R. Gonzalez, “Low-Power Digital Design”, Digest of Technical Papers, IEEE Symposium of Low-Power Electronics, pp. 8 – 11, 1994

[17] G. N. Hoyer, G. Yee, and C. Sechen, “Locally Clocked Pipelines and Dynamic Logic”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10 issue 1, Feb 2002, pp. 58 –62

[18] M. J. Flynn and S. F. Oberman, *Advanced Computer Arithmetic Design*, ISBN 0-471-41209-0, New York, NY: John Wiley and Sons, Inc., 2001, Chapter 4.

[19] Intel Corp, *Intel Pentium 4 Optimization Guide*, Intel Corp, Santa Clara, CA, 2002

[20] N. Itoh, Y. Naemura, H. Makino, Y. Nakase, T. Yoshihara, Y. Horiba, “A 600-MHz 54×54-bit Multiplier with Rectangular-Styled Wallace Tree”, *IEEE Journal of Solid-State Circuits*, vol. 36 issue 2, Feb 2001, pp. 249 –257

[21] F. Klass, M. J. Flynn, and A. J. van de Goor, “A 16×16-bit Static CMOS Wave-Pipelined Multiplier”**,** *Proceedings of the 1994 IEEE International Symposium on Circuits and Systems*, vol. 4, 30 May-2 Jun 1994, pp. 143 -146

[22] P. Larsson-Edefors, “A 965-Mb/s 1.0-µm Standard CMOS Twin-Pipe Serial/Parallel Multiplier**”,** *IEEE Journal of Solid-State Circuits*, vol. 31 issue 2, Feb 1996, pp. 230 –239

[23] M. M. –O. Lee and B. L. Cho, “Ultra-High Speed Parallel Multiplier with New First Partial Product Addition Algorithm”, *Proceedings, 4th International Conference on ASIC*, 2001, pp. 592 –595

[24] S. H. Lee, S. I. Bae, and H. J. Park, “A Compact Radix-64 54 × 54 CMOS Redundant Binary Parallel Multiplier”, *IEICE Transactions on Electronics*, vol.E85-C No.6, June 2002, pp. 1342 – 1350

[25] R. Lin, “A Regularly Structured Parallel Multiplier with Low-Power Non-Binary-Logic Counter Circuits**”,** *VLSI Design*, vol. 12 no. 3, 2001, pp. 377 –390

[26] F. Liu and K. T. Lau, “Pass Transistor Adiabatic Logic with NMOS Pull-down Configuration”, *Electronics Letters*, vol. 34 issue 8, 16 Apr 1998, pp. 739–741

[27] W. Liu, C. T. Gray, D. Fan, W. J. Farlow, T. A. Hughes, and R. K. Cavin, “A 250-MHz Wave Pipelined Adder in 2-µm CMOS**”,** *IEEE Journal of Solid-State Circuits*, vol. 29 No. 9, September 1994, pp. 1117 – 1128

[28] E. Macii, “RT and Algorithmic-Level Optimization for Low Power”, in W. Nebel and J. Mermet, eds., *Low Power Design in Deep Submicron Electronics*, Dordrecht, the Netherlands: Kluwer Academic Publishers, 1999

[29] H. Makino, Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara, and K. Mashiko, “An 8.8-ns 54×54-bit Multiplier with High Speed Redundant Binary Architecture”, *IEEE Journal of Solid-State Circuits*, vol. 31 issue 6, Jun 1996, pp. 773 –783

[30] H. Makino, H. Suzuki, H. Morinaka, Y. Nakase, K. Mashiko, and T. Sumi, “A 286 MHz 64-b Floating Point Multiplier with Enhanced CG Operation”**,** *IEEE Journal of Solid-State Circuits*, vol. 31 issue 4, Apr 1996, pp. 504 –513

[31] A. J. Martin, M. Nyström, and P. I. Pénzes, “ET^2: A Metric for Time and Energy Efficiency of Computation”, in R. Graybill and R. Melhem, eds., *Power Aware Computing*, New York, NY: Kluwer Academics / Plenum Publishers, 2002.

[32] MOSIS, Inc., http://www.mosis.org/Technical/Designrules/scmos/scmos-main.html

[33] N. Ohkubo, M. Suzuki, T. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, Y. Nakagome, “A 4.4 ns CMOS 54×54-b Multiplier Using Pass-Transistor Multiplexer**”,** *IEEE Journal of Solid-State Circuits*, vol. 30 issue 3, Mar 1995, pp. 251 –257

[34] V. G. Oklobdžija, D. Villeger, and S. S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using An Algorithmic Approach", *IEEE Transactions on Computers*, vol. 45 issue 3, Mar 1996, pp. 294 –306

[35] M. Olivieri, "Design of Synchronous and Asynchronous Variable-Latency Pipelined Multipliers", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9 issue 2, Apr 2001, pp. 365 –376

[36] B. Parhami, *Computer Arithmetic – Algorithms and Hardware Designs*, 1st ed. New York, NY: Oxford University Press, 2000, Chapter 11.

[37] M. C. Park, B. W. Lee, G. M. Kim, and D. H. Kim, "Compact and Fast Multiplier Using Dual Array Tree Structure", *Proceedings of IEEE International Symposium on Circuits and Systems*, 3-6 May 1993, vol. 3, pp. 1817 – 1820

[38] R. S. Parthasarathy and R. Sridhar, "Double Pass Transistor Logic for High Performance Wave Pipeline Circuits", *Proceedings of 1998 Eleventh International Conference on VLSI Design*, 4-7 Jan 1998, pp. 495 – 500

[39] R. Rogenmoser, L. O'Donnell, and S. Nishimoto, "A Dual-issue Floating-Point Coprocessor with SIMD Architecture and Fast 3D Functions", *Digest of Technical Papers, 2002 IEEE International Solid-State Circuits Conference*, 2002, vol. 1, pp. 414 – 415

[40] T. Säntti and J. Isoaho, "Modified SRCMOS Cell for High-Throughput Wave-Pipelined Arithmetic Units"**,** *Proceedings of the 2001 IEEE International Symposium on Circuits and Systems*, vol. 4, May 2001, pp. 194 –197

[41] N. R. Scott, *Computer Number Systems & Arithmetic*, ISBN 0-13-164211-1 01, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1993, Chapter 4.

[42] A. M. Shams, T. K. Darwish, and M. A. Bayoumi, "Performance Analysis of Low-Power 1-bit CMOS Full Adder Cells", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10 issue 1, Feb 2002, pp. 20 – 29

[43] A. Srivastava and J. A. Davis, "Minimizing Total Power by Simultaneous V_{dd}/V_{th} Assignment", *IEEE Transactions on Computer Aided Design for Integrated Circuits and Systems* , vol. 23 issue 5, May 2004, pp. 665 – 677

[44] V. Stojanovic and V. G. Oklobdžija, "Comparative Analysis of Master-Slave Latches and Flip-flops for High-Performance and Low-Power Systems", *IEEE Journal on Solid-State Circuits*, vol. 34 issue 4, April 1999, pp. 536 – 548

[45] J. B. Sulistyo and D. S. Ha, "5 GHz Pipelined Multiplier and MAC in 0.18μm Complementary Static CMOS", Proceedings of the 2003 IEEE International Symposium on Circuits and Systems, vol. 5, pp. V-117 – V-120, 2003

[46] J. B. Sulistyo, and D. S. Ha, "A New Characterization Method for Delay and Power Dissipation of Standard Library Cells", *VLSI Design*, vol. 16 no. 3, Nov 2002, pp. 667 – 678

[47] J. B. Sulistyo and D. S. Ha, "HyPipe: A New Approach for High Speed Circuit Design", Proceedings of the 15th Annual IEEE International ASIC/SOC Conference, pp. 203 – 207, 2002

[48] J. B. Sulistyo and D. S. Ha, "4.17 GHz Pipelined Signed Multiplier in 0.18μm Complementary Static CMOS", To be submitted to *IEEE Transactions on VLSI*, 2005

[49] I. E. Sutherland and R. F. Sproull, "Logical effort: designing for speed on the back of an envelope," in *Proc. Conf. Adv. Res. VLSI*, Nov. 1991.

[50] I. E. Sutherland, R. F. Sproull and D. Harris, *Logical Effort: Designing Fast CMOS Circuits*, ISBN 1-55860-557-6, 1st ed. San Francisco, CA: Morgan Kaufmann Publishers, Inc, 1999.

[51] Tanner Corporation, "Digital Low-Power Standard Cell Library for MOSIS TSMC CMOS 0.25 Process Deep Sub-Micron Technology, mTSMd 0.25µ", Tanner Research, Inc., 1999, http://www.mosis.org/cell-libraries/scn025-std-cells/mtsmd025dl.pdf

[52] Tanner Corporation, "MTSMS035DL Digital Low-Power Standard Cell Library for TSMC CMOS 0.35µ Sub-Micron Process, Revision A, mTSMs035", Tanner Research, Inc., 1999, http://www.mosis.org/cell-libraries/scn035-std-cells/mtsms035dl.pdf

[53] Tanner Corporation, "MAMIS035DL Digital Low-Power Standard Cell Library for MOSIS AMI 0.5µ Sub-Micron Process, Revision A, mAMIs0.5µ", Tanner Research, Inc., 1999, http://www.mosis.org/cell-libraries/scn05-std-cells/mAMIs05DLs.pdf

[54] Tanner Corporation, "Digital Low-Power Standard Cell Library for MOSIS HP AMOS14TB Process Sub-Micron Technology, mHPs 0.5µ", Tanner Research, Inc., 1999, http://www.mosis.org/cell-libraries/scn05-std-cells/mhps05dls.pdf

[55] J. –S. Wang, P. –H. Yang, and D. Sheng, "Design of a 3-V 300-MHz Low-Power 8-b×8-b Pipelined Multiplier Using Pulse-Triggered TSPC Flip-Flops", *IEEE Journal of Solid-State Circuits*, vol. 35 issue 4, Apr 2000, pp. 583 – 592

[56] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design, A System Perspective*, 1st ed. Santa Clara, CA: Addison-Wesley Publishing Company, 1993.

[57] D. C. Wong, G. De Micheli, and M. J. Flynn, "Designing High-Performance Digital Circuits Using Wave Pipelining: Algorithms and Practical Experiences", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12 no. 1, January 1993, pp. 25 – 46

[58] K. Yano, T. Yamanaka, T. Nishida, M. Saito, K. Shimohigashi, and A. Shimizu, "A 3.8-ns CMOS 16×16-b Multiplier Using Complementary Pass-Transistor Logic", *IEEE Journal of Solid-State Circuits*, vol. 25 issue 2, Apr 1990, pp. 388 – 395

[59] G. K. Yeap, *Practical Low Power Digital VLSI Design*, 1st ed. Boston, MA: Kluwer Academic Publishers, 1998

Vita

Jos Budi Sulistyo was born in Jember, Indonesia, in May 11, 1970. He completed his Bachelor of Science degree in Electrical Engineering from Texas A&M University in 1993 and his Master of Science degree, also in Electrical Engineering, from Virginia Polytechnic Institute and State University, under the guidance of Professor Dong S. Ha, in 2000.

He is a staff member at the Indonesian National Atomic Agency, where he has worked since 1988.