HIGH-PERFORMANCE HYBRID WAVE-PIPELINE SCHEME AS IT APPLIES TO ADDER MICRO-ARCHITECTURES

By

JAMES E. LEVY

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

MAY, 2005

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of JAMES E. LEVY find it satisfactory and recommend that it be accepted.

_____
Chair

_____

_____

HIGH-PERFORMANCE HYBRID WAVE-PIPELINE SCHEME AS IT APPLIES TO ADDER MICRO-ARCHITECTURES

Abstract

by James E. Levy, M.S.
Washington State University
May, 2005

Chair: Jabulani Nyathi

Pipelining digital systems has been shown to provide significant performance gains over non-pipelined systems and remains a standard in microprocessor/digital design. The desire for increased performance has led to research on deeper pipelines and new pipelining architectures such as wave-pipelining and hybrid wave-pipelining. In this thesis a hybrid wave-pipelined parallel adder is presented and compared to conventional- and wave-pipelined parallel adders. The comparison shows that the hybrid wave-pipelined adder operates at frequencies 19% and 167% faster than wave-pipelining and conventional pipelining (when the same stage partitioning is used) respectively. A performance estimation shows that if a deep conventional pipelined adder is implemented the hybrid wave-pipelined adder still outperforms a super-pipelined adder by 42%. Performance is the main benefit of using hybrid wave-pipelining. Other benefits may include lessening the clock skew and clock distribution delays, the ability to sustain a greater number of data waves within the pipe and the ability to easily perform clock gating. This thesis also presents a novel hybrid ripple carry-/carry lookahead-adder (RCA/CLA) adder that uses a prediction scheme to calculate the carry. Simulation results have shown the prediction scheme outperforms a traditional RCA/CLA by 22%-67% with only a 1.5% increase in power. The scheme reduces the transistor count by 15% per CLA block.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

Dedication

To my parents for their love and support.

To my lovely wife Jamie Bellona for waiting for me.

And to Dr. Jabulani Nyathi for talking me into staying for my masters degree.

CHAPTER 1

Introduction

As technology scales the need to explore new architectures and re-evaluate old ones is increasingly important. New device physics and increased device speeds coupled with new wire models could mean that current architecture approaches will no longer operate at an optimum while older schemes might see an increase in their potential use. An architecture that worked well for one technology may perform poorly at another, concepts that where once thought obsolete might be better solutions. This becomes especially true as we approach the sub-90nm process.

One of the most important architectures in computer and digital designs is that of adders. Adders are used in many applications ranging from microprocessors to embedded systems. The adder being such an important digital component has been the focus of intense research for the last few decades. There have been many proposed architectures ranging from serial [39], [15] to parallel [27], [18], [3] and asynchronous [30] to synchronous [10]. These architectures make an attempt to optimize the three fundamentals of digital design, speed, power and area. The optimization of these circuits can be done at many levels including the architectural, logic and/or circuit levels.

Some of these architectures include ripple carry adders (RCA), carry lookahead adders (CLA), carry-skip adders, and carry select adders to name a few [39], [6], [20] and [24]. Each of these

basic adders then has numerous variations and optimizations to squeeze out as much performance as possible for a given circuit. Some optimize size, while others look toward high speed at low power.

As computing elements continue to grow in size and complexity the need for large data-paths requires adders to handle computations of 64-bits or more. As the number of inputs increases the delay associated with propagating the carry from the least significant bit to the most significant bit increases as well. The desire to eliminate this added delay has lead to parallel prefix adder implementations where the carry and sum are generated in parallel [3], [27]. In addition to the various adder architectures intended for high performance, there are techniques that can be employed to further enhance performance. Wave-Pipelining is one such technique which is used to enhance digital system design [30]. If applied in adder design this technique finds itself well suited to parallel structures because they have a more regular layout than other adder architectures. Finally Hybrid Wave-Pipelining seeks to further increase clock speed over wave pipelining by combining traditional pipelining techniques with wave-pipelining ideas [34], [11] and [35].

This thesis will explain the concepts, background and benefits of Hybrid Wave-Pipelining. It will also set up a fundamental understanding of adder architectures and their limitations. Hybrid Wave-Pipelining will then be applied to one of the parallel adder architectures in order to further elaborate on its advantages. The thesis will explore the constraints, limitations and performance enhancements that Hybrid Wave-Pipelining offers as compared to conventional and Wave-Pipelining techniques. In addition this thesis will briefly explore a newly proposed hybrid carry lookahead adder architecture for use in design technologies below 90 nm.

Chapter 2 will elaborate on the equations and basic concepts of traditional pipelining, Wave-Pipelining and Hybrid Wave-Pipelining. Chapter 3 will look at clock distribution with regard to pipelined systems. Chapter 4 will look at the problems associated with data dependencies in Wave-Pipelining and Hybrid Wave-Pipelining. Chapter 5 will outline the problems that latches and flip-flops can cause when implementing wave- and hybrid wave-pipelined systems. Chapter 6

contains the majority of the research in which adder architectures and our specific implementations are explored and results reported. Future work and research contributions are outlined in Chapter 7 and finally Chapter 8 will summarize the findings presented in a conclusion.

CHAPTER 2

Hybrid Wave-Pipelining

2.1 Introduction

Hybrid Wave-Pipelining is a technique which seeks to further reduce clock period by combining the techniques of Wave-Pipelining with conventional pipelining. In order to fully understand how hybrid wave-pipelining works sections 2 and 3 of this chapter will introduce the concepts of conventional pipelining and wave-pipelining. Sections 4 and 5 will develop the basic concepts of Hybrid Wave-Pipelining as well as compare and contrast these relationships to a typical Wave-Pipelining scheme. Section 6 will give concluding remarks on the three approaches, conventional pipelining, wave-pipelining and hybrid wave-pipelining

2.2 Conventional Pipelining

Pipelining has been used in a variety of applications the most prominent being high-speed central processing units(CPU's) [22], other digital systems in which pipelining is used include the design of multipliers [25] and [31], adders [38] and [10], as well as high speed memories [13].

In order to show the differences between a non-pipelined system and a conventional pipelined system we will use a multi-stage processor. In contrast to a conventional pipelined system, a non-pipelined system operates on one instruction at a time until completion. During this time no other

instructions can be executed or issued. Figure 2.1 shows the execution sequence for a non-pipelined system. This figure comes from [19].

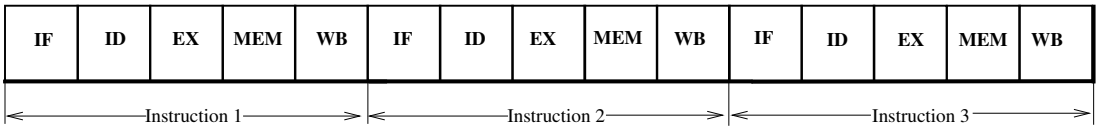

Figure 2.1: Execution pattern of three instructions in an un-pipelined machine.

In figure 2.1 a five stage processor has been used. The stages are instruction fetch (IF), instruction decode (ID), execute (EX), memory access (MEM), and write back (WB). The execution of three instructions is shown in the figure. In this arrangement the instructions must pass through all five stages before a new instruction can be issued. The output of each stage is the input to the following one. If an instruction is issued the instruction fetch brings the instruction from memory into the processor, this is then passed to instruction decode where the processor decodes the instruction type and registers to be used. During this time hardware used for the instruction fetch is idle. Also remaining idle is the hardware at stages EX, MEM, and WB. At any given time four out of the five stages are idle.

To make better use of the hardware conventional pipelining is used. Figure 2.2 shows how six instructions are overlapped in the conventional pipelining scheme. Notice that not only does conventional pipelining make better use of the hardware but also increases the system performance. Here we see that Instruction 1 will enter the pipeline first, once it has been fetched by the processor and passed to the instruction decode Instruction 2 will enter the pipe. While Instruction 1 is being decoded Instruction 2 is being fetched. When Instruction 1 is being executed Instruction 2 is being decoded and Instruction 3 enters the pipe. The stages of the pipe are operating simultaneously and given an equal amount of time to complete at each stage. This is accommodated by mandating that the frequency of the pipeline be limited by the stage that takes the longest amount of time to execute. Figure 2.2 illustrates how six separate instructions are executed in the conventional pipeline and as can be seen the only time hardware is idle is when the pipeline is not “full”.

However, if we consider the logic depth per stage the fact that all stages operate at the same rate of the slowest stage, we can show that the stage with the shortest computation time is under-utilized (remains idle for some fraction of the clock cycle).

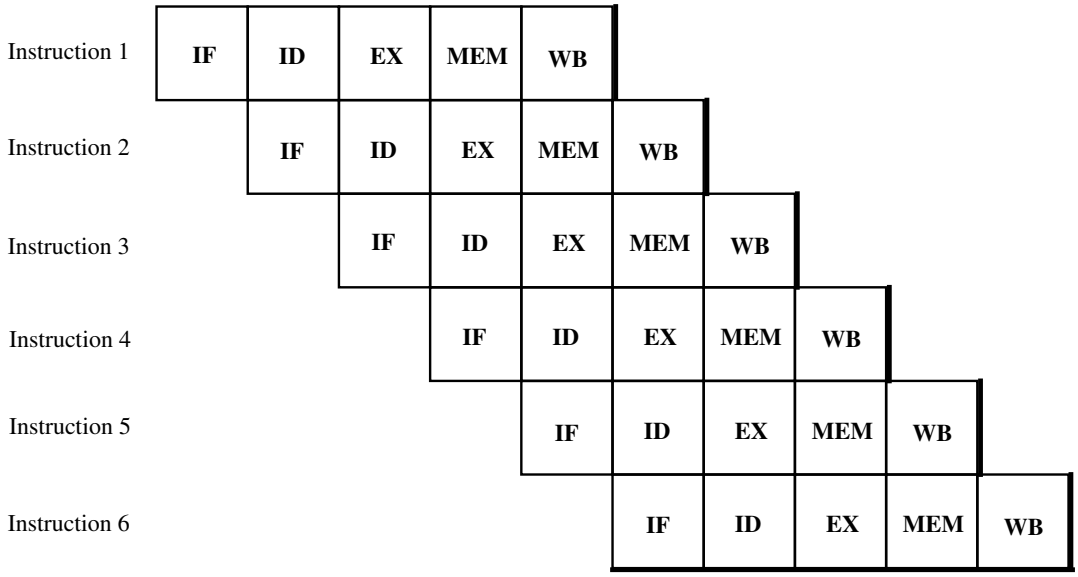

Figure 2.2: Execution of six instructions in a pipelined machine.

By using pipelining the throughput of the system is increased, however there are still problems involved with conventional pipelining. These problems include data hazards, structural hazards, and control hazards. From this brief overview of pipelined systems it is apparent that a pipelined system has many benefits over a non-pipelined one and is applicable to many different applications. In this research the use of pipelining in relation to adder architectures is of particular interest.

2.3 Wave Pipelining

Conventional pipelining utilizes latches or flip-flops (registers) to separate stages and guarantee that data is not transferred before it is supposed to be. In doing this pipelining requires the use of internal latches in addition to input and output registers. Figure 2.3 shows a block diagram of a pipelined system including internal registers. The internal registers ensure that when the clock edge arrives data is transferred from the previous stage to the next in a synchronous manner. With

such a system there is only one set of data between internal registers.

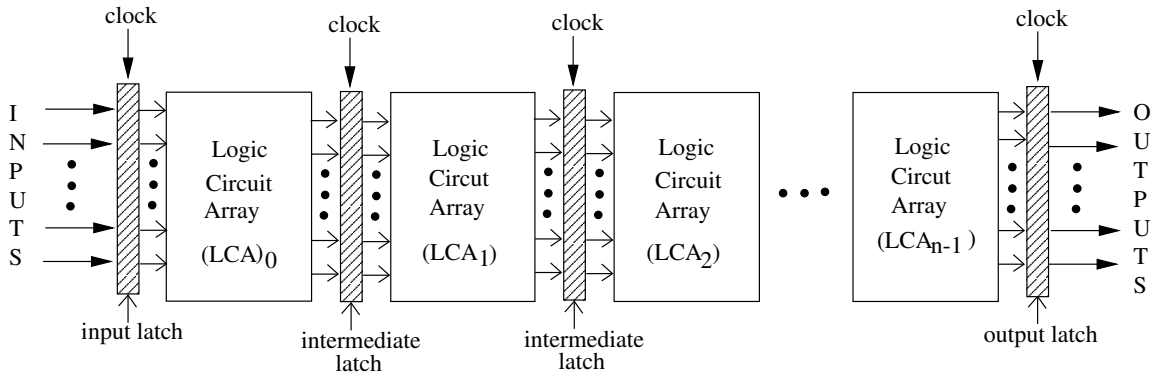

Figure 2.3: Block diagram of a digital system using conventional pipelining

Wave pipelining however implements a pipeline using the logic alone without the need for the internal registers [5]. Using this technique increases the clock frequency governing the digital system. Wave-pipelining allows for coherent waves to be sent trough the pipeline's logic blocks and allows for new data to be issued at the input register before the preceding wave has reached the output register. Multiple waves can be sustained within the pipeline.

Wave-pipelining reduces the area, power and load associated with the clock by reducing the number of intermediate register. The rate at which the pipeline can be run is now governed not by the slowest stage of the pipe but the difference between the longest and shortest data paths [5]. With this knowledge wave-pipelining requires that the data paths be balanced so that data issued at the input arrives at the output simultaneously regardless of the delay path taken. A block diagram of the wave-pipelined system is shown in Figure 2.4. The intermediate registers of Figure 2.3 have been removed and replaced by "intrinsic" latches. Also, the system clock is no longer distributed. Now, multiple coherent "waves" of data are available between storage elements.

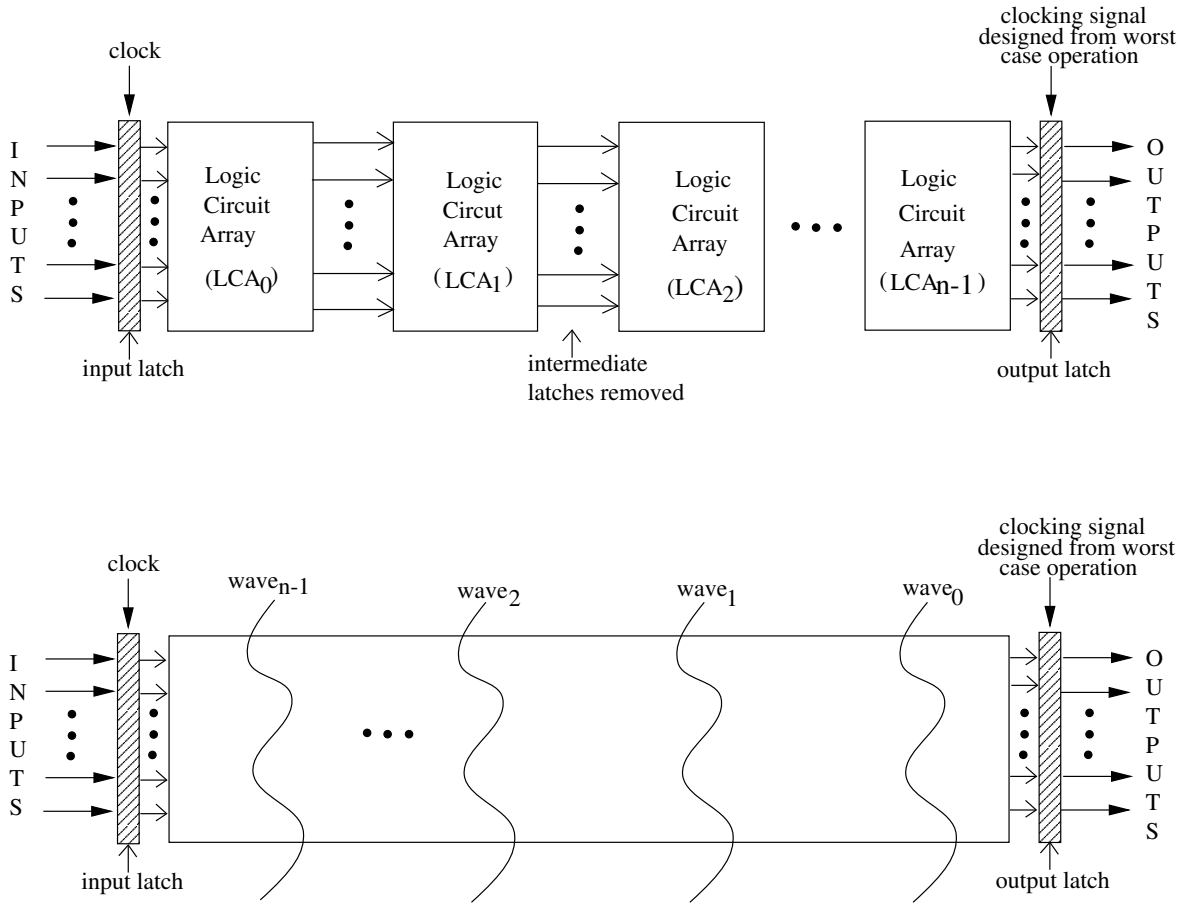

Figure 2.4: Block diagram of a wave-pipelined digital system.

2.4 Wave-Pipelining Requirements

Wave-pipelining requires studies across a variety of levels such as process, layout, circuit, logic, timing and architecture. Several research groups have explored some of these areas. Some of the challenges of designing wave-pipelined systems within any of the areas mentioned above include:

- *Preventing intermixing of unrelated data "waves."* There must be no data overrun in each circuit block, and it must be ensured that no over committing of the data path occurs. This is achieved by determining an appropriate range of clock frequencies at which to apply data at the input.

- *Designing dedicated control circuitry.* Control logic circuits must be designed to operate

synchronously with the circuitry of the pipeline stages. The number of these control logic circuits must be minimal in order to be area efficient.

- *Balancing delay paths.* Delay paths must be equalized to ensure that data waves applied to the input latch propagate through all the stages synchronously. This is achieved by inserting delay elements in the shortest paths within logic blocks to equalize their delays to those of the longest paths. This approach allows for the elimination of the intermediate latches or registers.

The requirements stated above are not the only ones. They represent some of the most important design issues in wave-pipelining.

2.5 Wave-Pipelining Modeling

In order to have a basic understanding of wave-pipelining, some of the underlying aspects of this approach are reviewed first. In this section some of the parameters of importance in wave-pipelining are discussed. These include delays within combinational logic, clock skew, and sampling time at output registers. Following an example from Wayne *et al* [5], a single combinational logic block is considered in order to come up with the terminology that is used to determine the timing requirements for wave-pipelining. The timing constraints derived from using this single block should hold for any design with more than a single stage of combinational logic.

2.5.1 Wave-Pipelining Formulation

To present the clock parameters and delays, a simple pipeline stage is shown in Figure 2.5.

Some important labels are:

- D_{min}; the minimum propagation delay through the combinational logic.
- D_{max}; the Maximum (worst case) delay in the combinational logic.
- T_{clk}; the clock period.

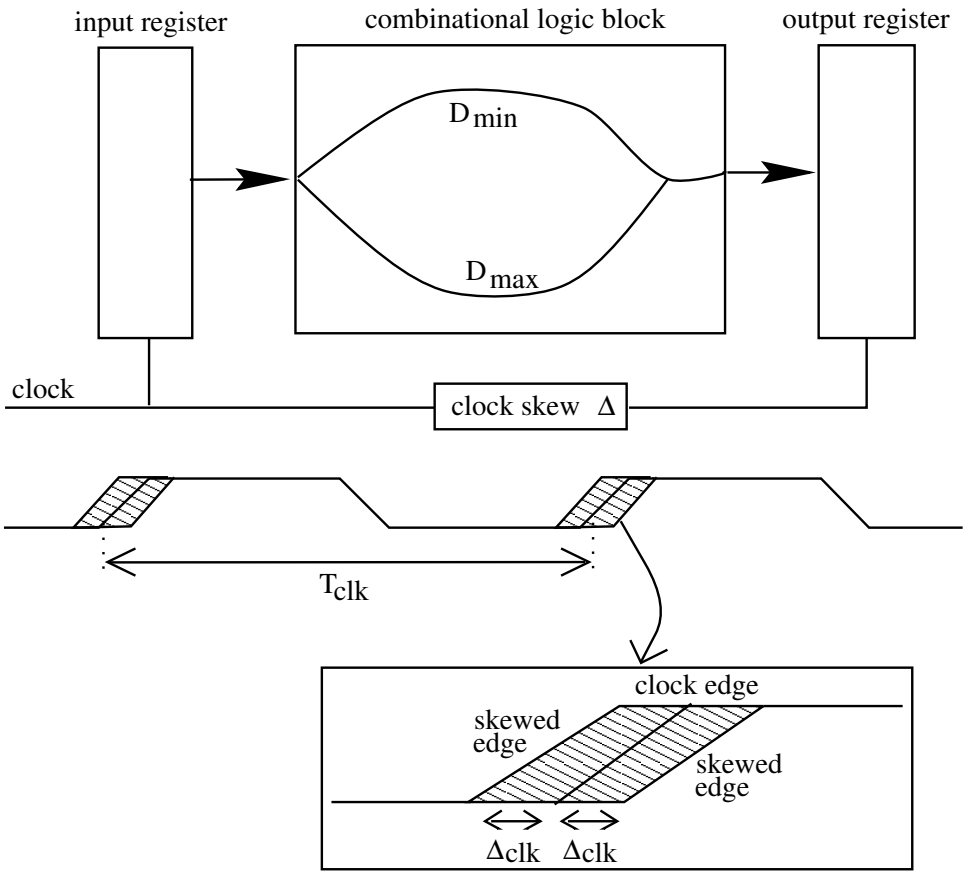

Figure 2.5: Longest and shortest path delays of a combinational logic block.

- T_s and T_h; the register setup and hold times.
- Δ; the constructive clock skew.
- Δ_{clk}; the register's worst case uncontrolled clock skew.
- D_R; the register's propagation delay.

For this discussion it will be assumed that the combinational logic block has two inputs, the setup and hold times of the input and output registers are equal and the propagation delay through the register is the same for both the input and output registers. From the diagram of Figure 2.5 the propagation delay of the signals through the combinational logic can be related to the input and output clocks. Figure 2.6 shows the maximum and minimum delays in relation to logic depth.

The shaded region in the figure represents a period during which data is being processed within the combinational logic block. Figure 2.6 can be extended to represent several data sets being processed as time progresses.

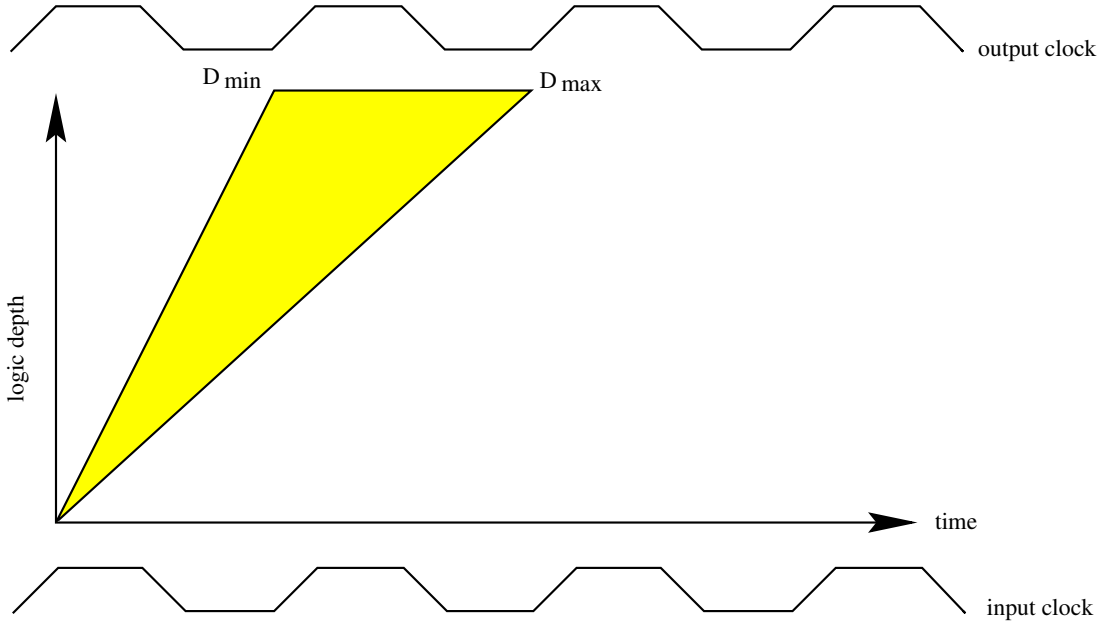

Figure 2.6: Relating the delay differences to logic depth.

Figure 2.7 has labels that are used to describe the timing constraints of wave-pipelining. A few more terms need to be defined from this figure. They are T_L, the time at which data at the output register can be sampled, and T_{ax}, the temporal separation between the waves at an intermediate node x. These definitions appear in [5] and the same variables are used here for ease of comparison with the equations that are derived in the next subsection.

For wave-pipelining to operate properly, the system clocking must be such that the output data is clocked after the latest data has arrived at the output, and the earliest data from the next clock cycle arrives at the output [5]. The time at which data can be sampled at the output register T_L is given by:

$$T_L = N\, T_{clk} + \Delta \tag{2.1}$$

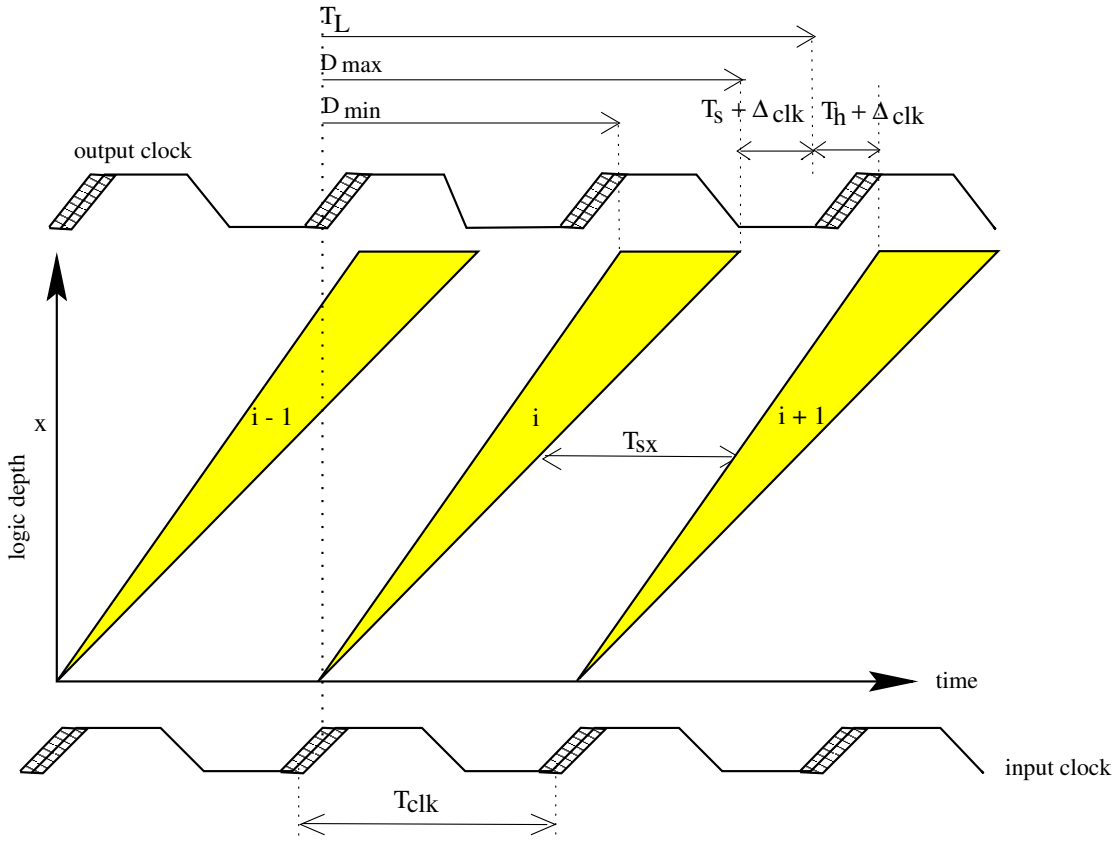

Figure 2.7: Temporal/spatial diagram of a wave-pipelined system.

where N is the number of clock cycles required to propagate the input through the combinational logic block before it could be latched at the output register.

Latching the latest data at the output register requires that the latest possible signal arrives early enough to be clocked by this register during the N^{th} clock cycle. Thus, the lower bound of T_L is:

$$T_L > D_R + D_{max} + T_s + \Delta_{clk} \tag{2.2}$$

To latch the same data requires that the arrival of the next wave not interfere with the latching of the current wave output. The clock skew Δ_{clk} must also be accounted for, resulting in the upper bound of T_L being:

$$T_L < T_{clk} + D_R + D_{min} - (\Delta_{clk} + T_h) \tag{2.3}$$

The difference of these two equations results in a value of T_{clk} given by:

$$T_{clk} > (D_{max} - D_{min}) + T_s + T_h + 2\Delta_{clk} \tag{2.4}$$

From Equation 2.4 it is apparent that the minimum clock period is limited by the difference in path delays ($D_{max} - D_{min}$) and the clocking overhead $T_s + T_h + 2\Delta_{clk}$. This overhead occurs as a result of the inclusion of the input and output registers and the clock skew. Internal nodes within the system need to be considered in this analysis. It is important to ensure that there is no data overlap within the system. Therefore, the next earliest possible set of data should not arrive at a node until the latest possible wave has propagated through. If an output x is the output of an internal node within a logic network at a point on the logic depth axis of Figure 2.7, its longest and shortest delays from its inputs would be represented by the variables d_{max} and d_{min}, respectively. The equation that describes this internal node's constraints is:

$$T_{clk} \geq d_{max(x)} - d_{min(x)} + T_{sx} + \Delta_{clk} \tag{2.5}$$

where T_{sx} is the minimum time that node x must be stable in order to correctly propagate a signal through the gate.

2.5.2 Minimizing the Clock Period

In order to minimize the clock period, both register constraints and internal node constraints must be taken into account. According to [29] the feasibility region of valid clock period T_{clk} is not continuous. It is composed of a finite set of disjoint regions. Equations 2.2 and 2.3 are revisited in order to derive a two sided constraint on the clock period. It was established that the register latching time is given by $T_L = NT_{clk} + \Delta$ and if this is applied to Equations 2.2 and 2.3 to obtain the intermediate values between the lower and upper bound of the register latching time, the

following equation results:

$$D_R + D_{max} + T_s + \Delta_{clk} < NT_{clk} + \Delta < T_{clk} + D_R + D_{min} + D_{max} - (\Delta_{clk} + T_h) \quad (2.6)$$

To avoid writing long expressions two variables T_{max} and T_{min} are introduced.

- T_{max}; maximum delay through the logic including clocking overhead and clock skews.
- T_{min}; minimum delay through the logic including clocking overhead and clock skews.

$$T_{max} = D_R + D_{max} + T_s + \Delta_{clk} - \Delta \quad (2.7)$$

and

$$T_{min} = D_R + D_{min} - (\Delta_{clk} - T_h) - \Delta \quad (2.8)$$

Having established these new variables Equation 2.6 can be simplified to read:

$$\frac{T_{max}}{N} < T_{clk} < \frac{T_{min} + T_{clk}}{N} \quad (2.9)$$

The inequality to the right can be simplified as follows:

$$NT_{clk} < T_{min} + T_{clk}$$

$$NT_{clk} - T_{clk} < T_{min}$$

$$(N-1)T_{clk} < T_{min}$$

Equation 2.9 becomes:

$$\frac{T_{max}}{N} < T_{clk} < \frac{T_{min}}{N-1} \quad (2.10)$$

The above analysis shows that for $N = 1$, the clock period is not continuous and is only bounded below by T_{max}.

2.5.3 *Hybrid Wave-Pipelining*

Having presented the timing constraints of wave-pipelining, attention is now turned toward describing these timing constraints as they apply to a hybrid wave-pipelined approach. Equations to describe the timing constraints for this approach are derived and compared to those of the previous subsection. In many computer/digital systems each stage has a significantly different function and circuitry, therefore, wide variations in delays (D_{min} and D_{max}) may not be tolerated. Figure 2.8 shows an example of a wave-pipeline system. The inputs to the system are passed in a synchronous manner by means of an external clock. Assuming three variables need be computed in this stage and passed on to the following stage, it follows that for a given set of inputs, these variables would have delays associated with each one of them. These delays are denoted as d_A, d_B, and d_C, for inputs A, B, and C respectively.

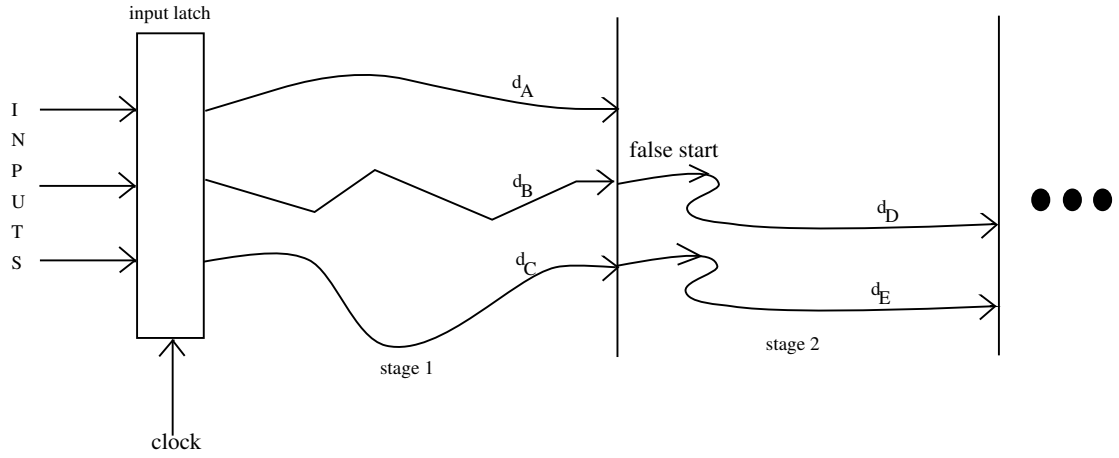

Figure 2.8: Example of delays associated with pipeline stages.

The difference in delay may cause stage 2 (Figure 2.8) to start in a different computation path than what is expected. This in turn produces a false start. This false start creates unnecessary changes in stage 2 as well as additional delays, that need not have occurred. These delays d_A, d_B and d_C would depend on the gates associated with each path, and also on the set of input values.

These problems are avoided using a hybrid wave-pipelining approach. In order to provide some

insight into the problem definition and solution, a brief summary is provided below. This summary can be drawn from Figures 2.10 and 2.11. A common engineering practice is to consider the worst case delay (D_{max}), to ensure that the system runs properly. D_{max} plays a very important role in the system's performance and safe regions of operation. D_{min} (the shortest delay path), on the other hand imposes a restriction in the valid input window. Getting D_{min} closer to D_{max} could increase this window; in other words it could decrease clock cycle time. Figures 2.10 and 2.11 show D_{min} and D_{max} for both wave-pipelining approaches.

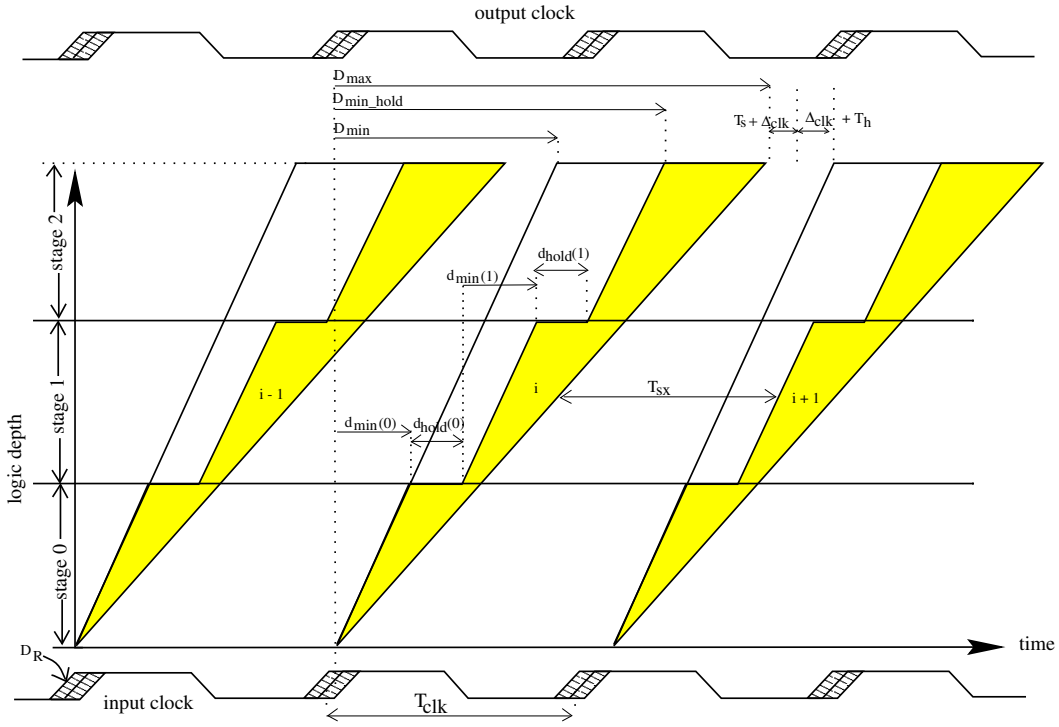

Figure 2.9: Temporal/Spatial diagram of a hybrid wave-pipelined system.

The equations derived for the hybrid wave-pipelining are denoted by the subscript h to differentiate them from those of the wave-pipelining time constraints. The definitions for the variables presented in the previous subsection still hold. To derive the equations that describe the time constraints for the hybrid wave-pipeline, the temporal/spatial diagram representing this scheme is presented first. The shaded regions of Figure 2.9 indicate that data is not stable; therefore, register

outputs cannot be sampled. The cones in this diagram have been arranged to represent each stage within the design. Some variables need to be defined. They are:

- $d_{min}(n)$; the minimum delay encountered in propagating data within a single stage n.
- D_{min_hold}; the overall minimum delay of all the stages and it includes the intrinsic registers' hold times.

For hybrid wave-pipelining, T_L's lower bound is described as follows:

$$T_{Lh} > D_R + Dmax + T_s + \Delta_{clk} \tag{2.11}$$

The upper bound of T_{Lh} is

$$T_{Lh} < T_{clk} + D_R + D_{min\_hold} - (\Delta_{clk} + T_h) \tag{2.12}$$

where $D_{min\_hold} = d_{min}(0) + d_{hold}(0) + d_{min}(1) + d_{hold}(1) + d_{min}(2) + d_{hold}(2)$

This equation takes into consideration the intermediate stages of the design. The minimum delays and the hold times of each stage are considered. From the above equation it can be determined that:

$$D_{min\_hold} \geq D_{min} \tag{2.13}$$

This implies that this delay difference is less than $D_{max} - D_{min}$ for the wave-pipelining scheme. If further derivations are carried out, the clock period for the hybrid approach is determined to be:

$$T_{clk(h)} > (D_{max} - D_{min\_hold}) + T_s + T_h + 2\Delta_{clk} \tag{2.14}$$

Comparing Equations 2.4 and 2.14 and having $D_{min\_hold} \geq D_{min}$, a conclusion that $T_{clkh} \leq T_{clk}$ can be drawn. This implies that the clock period for the hybrid wave-pipelined approach

allows for the clock signal period to be reduced, hence an increase in performance. A complete analysis of the hybrid wave-pipelining scheme must include clock cycle minimization, taking into consideration the constraints of the internal nodes of the system and the register constraints. Based on the analysis of Equation 2.6, the minimum delay of the hybrid approach can be re-written to include the stage hold times as follows:

$$T_{minh} = D_R + D_{min\_hold} - \Delta_{clk} - T_h - \Delta \tag{2.15}$$

The maximum delay through the logic including the overhead and clock skews T_{max}, remains unchanged for the hybrid scheme. From Equation 2.15, and by taking into consideration the fact that $D_{min\_hold} \geq D_{min}$, it is determined that: $T_{min} < T_{minh}$. Also from Figure 2.9 it can be noticed that the region in which data is not stable, i.e. the difference between $D_{max} - D_{min\_hold}$, is short. It can then be safely stated that $D_{max} \approx D_{min\_hold}$. The signal latching time, Equation 2.6 now becomes:

$$D_R + D_{min\_hold} + T_s + \Delta_{clk} - \Delta < NT_{clk} < T_{clk} + D_R + D_{min\_hold} - (\Delta_{clk} + T_h) - \Delta \tag{2.16}$$

This analysis is graphically presented in Figures 2.10 and 2.11. Figure 2.10 shows that there is room to make the clock cycle smaller, since the distance between labels $window_h$ and $window_w$ can be reduced. Figure 2.11 shows how effectively this could affect the clock period, reducing it from T_{clk} to T'_{clk}.

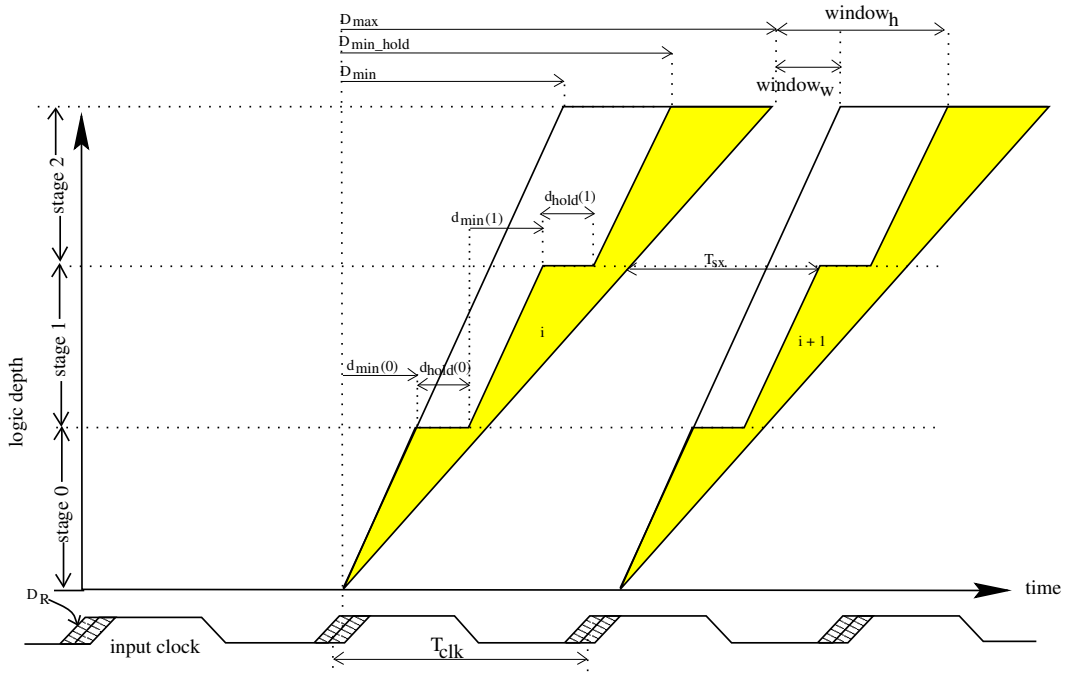

Figure 2.10: Temporal/spatial diagram before clock period reduction.

2.6 Concluding Remarks

In this chapter, the background material on wave-pipelining has been presented and compared to that of a hybrid wave-pipeline system. It is determined from the equations derived that the hybrid wave-pipeline can reduce further the clock cycle period. This chapter provides the basis for the studies undertaken in the subsequent chapters.

It has been shown in this chapter how efforts to improve performance have progressed starting with conventional pipelining where new instructions/data can be fed into a pipeline before the preceding instructions have been processed to completion. Wave-pipelining extends this pipelining scheme, and introduces the ability to remove intermediate latches within the pipeline, hence reducing the delays associated with these intermediate latches. Wave-pipelining further provides the ability to reduce clock cycle time. By carefully studying the timing constraints of wave-pipelining, a method termed hybrid wave-pipelining is introduced. Hybrid wave-pipelining further reduces the clock period by making the minimum delay (D_{min}) at each stage of the system approach the

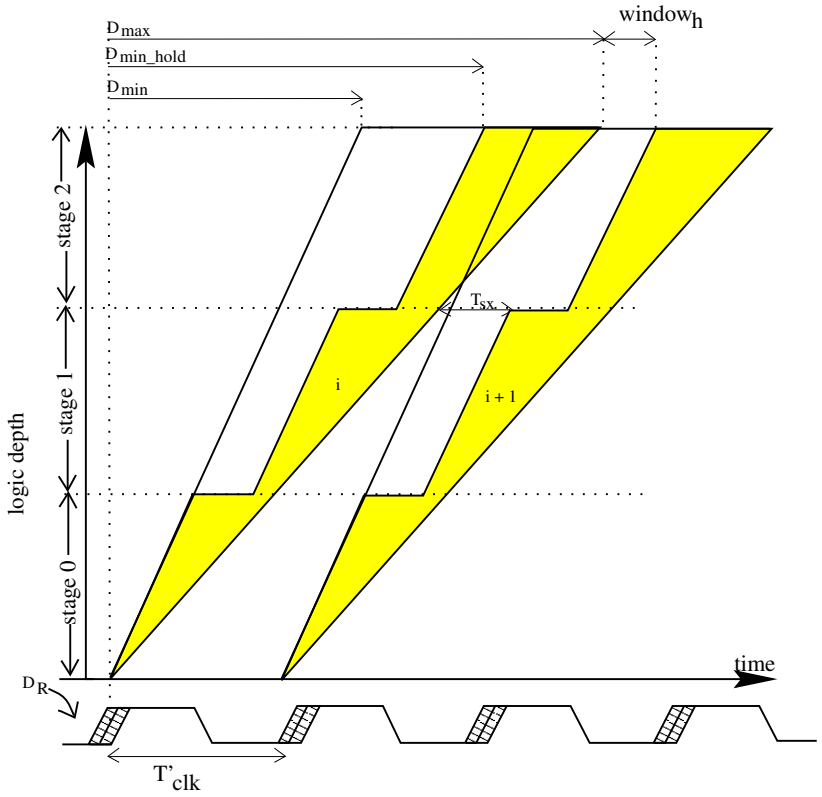

Figure 2.11: Temporal/spatial diagram after clock period reduction.

maximum delay (D_{max}). This in turn reduces the delay path difference and enables the reduction of T_{sx}, the separation between data "waves" at intermediate nodes. The results of these improvements have a bearing on the clock period. It can be made shorter and still enable data to propagate in it's own wave.

CHAPTER 3

Clock Distribution for Hybrid Wave Pipelining

3.1 Introduction

Clock distribution has become a significant challenge in the design of digital systems. The need for large clock trees and the ability to drive signals across chip make this challenge very cumbersome and tedious. In order to alleviate this problem while enhancing speed, Hybrid Wave-Pipelining reduces the number of latches needed in a conventional pipeline and thus reduces the size of the clock tree. In this chapter we will explore various clock distribution techniques as motivated by Hybrid Wave-Pipelining and adder designs. Section 3.2 will look at Clock Trees and Matched RC Trees. Section 3.3 will evaluate the technique of computing the clock. Section 3.4 will look at our current approach a modified matched RC Tree, while section 3.5 will close with some concluding remarks regarding clocking techniques as applied to Hybrid Wave-Pipelining.

3.2 Clock Trees and Matched RC Trees

We will discuss clock distribution in the context of pipelined systems where there is a need to clock all the latches within the pipeline simultaneously. In previous technology nodes (2μm, 1.5μm, 1μm

to 0.5μm) it has been sufficient to consider an interconnect of a given length as being equipotential. The rise of dominant wire delays have changed this view. Figure 3.1 below shows a pipeline scheme that would allow the latches to clock with minimal skew in previous technologies.

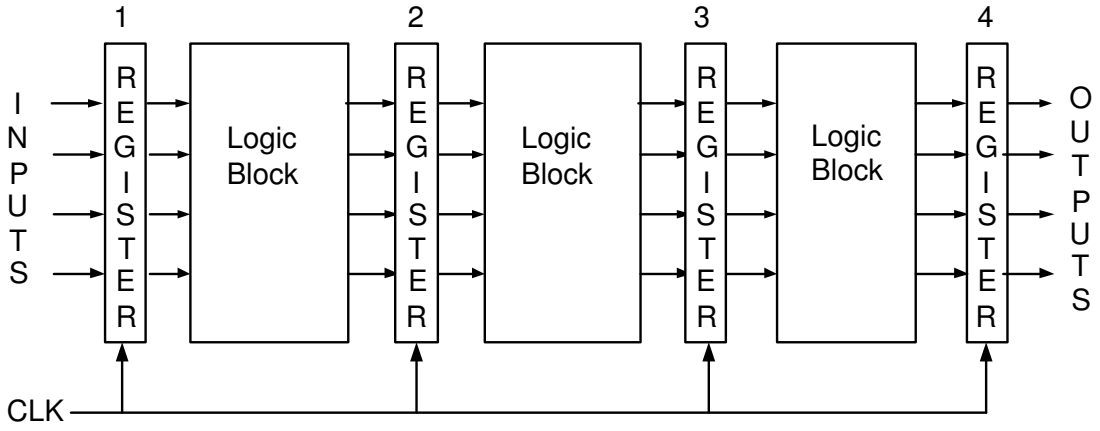

Figure 3.1: Typical Clock Distribution

In current technologies it will be difficult to have latches 1 and 4 receiving the clock signal at the same time due to the dominate wire delays. Several mature methods of addressing the clock distribution issue include: H-Tree, Grid Structures, matched RC Trees, Spines, etc [39], [36]. An example of a distributed RC Tree is shown below for the pipelined system in Figure 3.2.

The typical buffer insertion for a matched RC Tree is to start with a small inverter and continually double the next inverters size until it has reached the load it is driving. These buffers are distributed evenly throughout the clock signal wires to guarantee the optimal performance.

At the end of the buffer insertion if the output load is still too large, clock trees are used to break the load an individual gate is driving. A clock tree can fan-out to any number of nodes and is typically dependent on the design and loads driving. Figure 3.2 shows what a simple clock tree for an arbitrary circuit may look like. These clock trees are created using inverters since they are the smallest logic gate next to a single transistor and can easily be sized to provide appropriate drive strength, and provide equal rise and fall times. Many models have been suggested to model optimal buffer insertion and wire delays [33], [1] and [12].

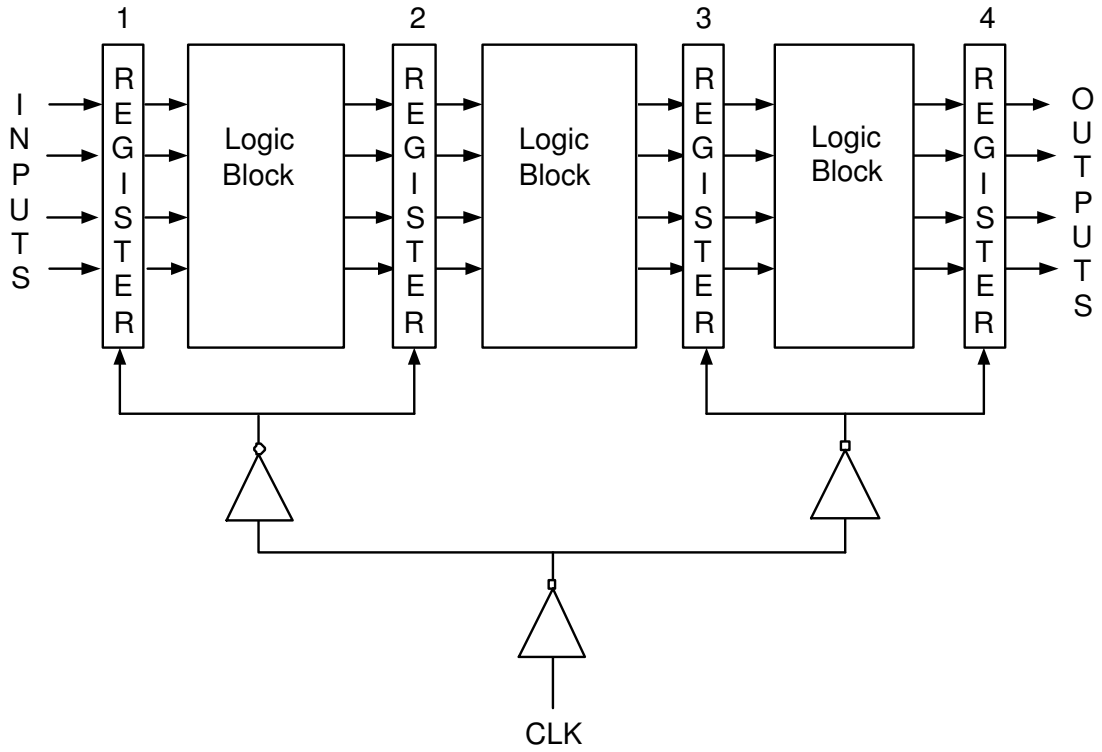

Figure 3.2: Distributed RC Tree

3.3 Clock Computation

The Hybrid Wave-Pipelining Scheme described in Chapter 2 Section 2.5.3 offers reduced clock cycle times, hence improved performance. The approach also provides a means by which clock skew and clock distribution can be managed. The scheme permits data to "travel" with its associated clock pulse. This is achieved by designing the clock signal to experience the same delays as the data. In this subsection we show how the clock circuitry is designed to mimic the delay of the data path thus alleviating the clock distribution issue. Figure 3.3 below is a block diagram of a general hybrid wave-pipelined clock system. This approach eliminates the need to design a matched RC Tree, Grid Structure, etc. since local clocks are generated at each stage.

In clock computation the idea is that the clock itself can be delayed using the same delays the logic experiences. If the same components are used the clock should in theory experience the same

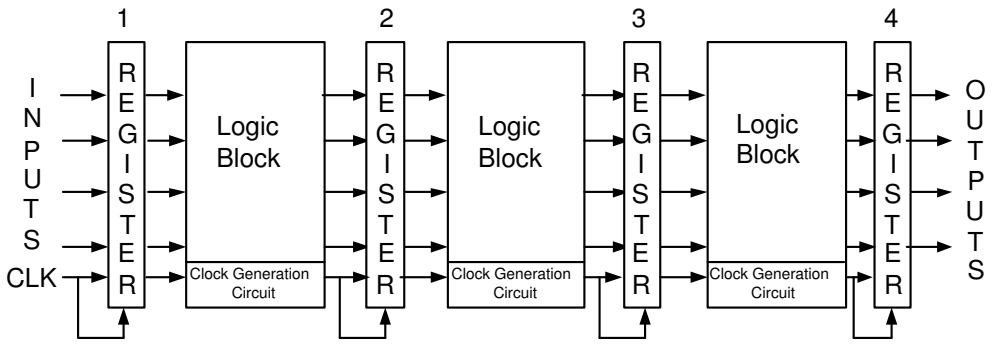

Figure 3.3: General Method for Pipelining the CLK

delay. An example of this is illustrated in Figure 3.4 below. In this figure the data experiences a delay through a series of biased-NAND gates (to be discussed in Chapter 4). If the clock is to follow the same delay the NAND gate can have one input tied to logic 1 which effectively creates an inverter. Now passing the clock through these inverter type NAND gates will force the clock to experience the same delay as the logic. It should be noted that in this figure the final NAND gate in the clock signal path will experience a much larger output load than those NAND gates within the data path. In order to guarantee the arrival of the clock in conjunction with the data signals the additional loading on the clock signal at the output registers must be accounted for.

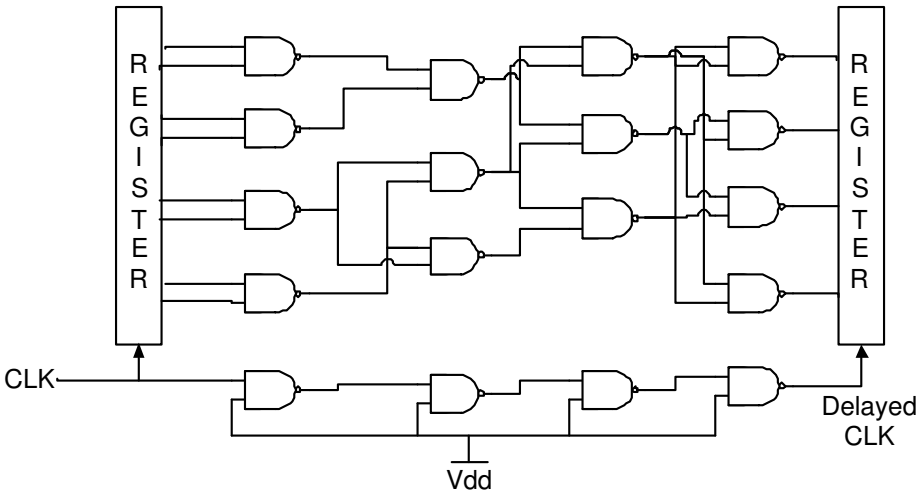

Figure 3.4: Clock Compuation by delaying clock to match data path

The example given above is a simple case. It should be noted that some gates can not be configured as easily to accommodate the clock. In these cases a special circuit may need to be designed to match the delay path of the logic. These circuits may not be easy to implement and may take up valuable design time. These components if done correctly should match the delay of the logic, but may come at the cost of additional hardware in the clock signal path. Regardless of what type of gates are used it is imperative that the clock signal be driven "strong" from the beginning of the delay path to the output where it will be used by the next stage. Weak clock signals drastically effect the performance of the circuit in a negative manner. Figure 3.5 shows the clock signal when biased NAND gates are used. Note the lowest voltage the clock signal reaches is just below 1V. In this example four biased NAND gates are used in the clock path. The lack of drive strength makes this particular approach unacceptable for the Hybrid Wave-Pipelined adder presented later in this thesis.

3.4 Matched RC Tree

In this thesis we report a clocking scheme that has elements of the matched RC Tree as well as the data matching delays of Hybrid Wave-Pipelining. Inverters are used to match the delay of the logic as well as matching the loading per stage. Figure 3.6 below shows the clocking approach used. Here each clock's branch is limited to a fan-out of 2 (FO2) in anticipation of skew issues with further scaling, and to support high frequencies greater than 1GHz.

The RC clock tree is included as part of the delay matching. If extra delay is needed to match the clock signal to the data, inverters can be added before the matched RC Tree to provide the necessary delay. Figure 3.7 shows how the clock propagates with its associated data. In this figure the output of an XOR is propagated down the pipe along with its associated clock. The final data to the output registers in Figure 3.7 is represented by the signal f31_2 and the output clock to the registers is shown as signal clk28a. The signals shown in the figure are those of a high performance Hybrid Wave-Pipelined adder (to be discussed in detail in section 6.6).

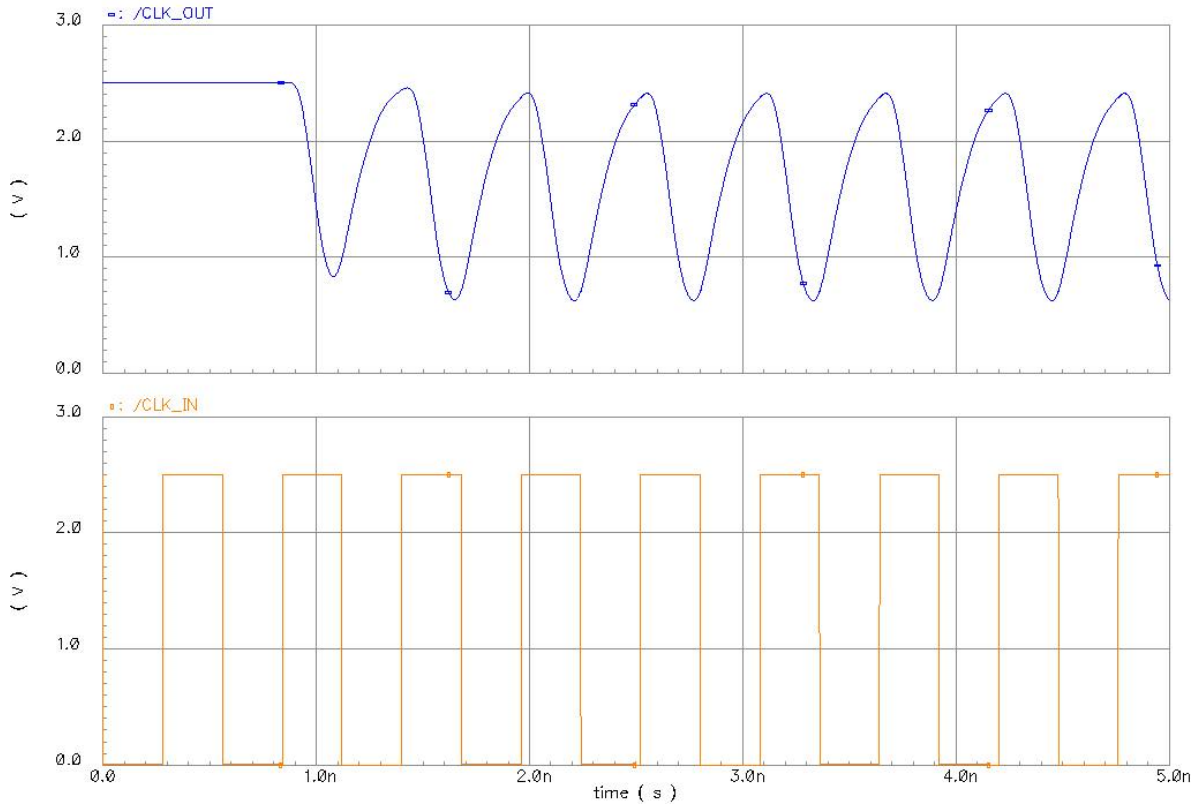

Figure 3.5: Clock Signal when using Biased NAND gates to match data path.

3.5 Conclusion

In this chapter three separate architectures have been presented for clock distribution. In relation to Hybrid Wave-Pipelining conventional clock distribution is not sufficient because the clock does not propagate with individual data waves. Clock computation is a viable alternative but consumes extra area and it can be tedious to design delay matching circuits. A modified RC Tree provides strong signal drive and the delay to match the data waves can easily be accommodated by the sizing and addition of inverters in the clock signal path.

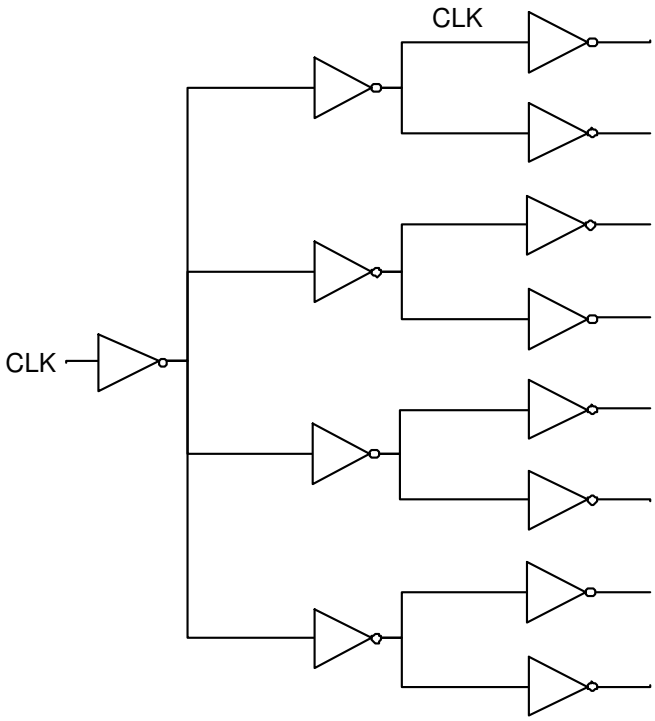

Figure 3.6: Matched RC Clock Tree Approach

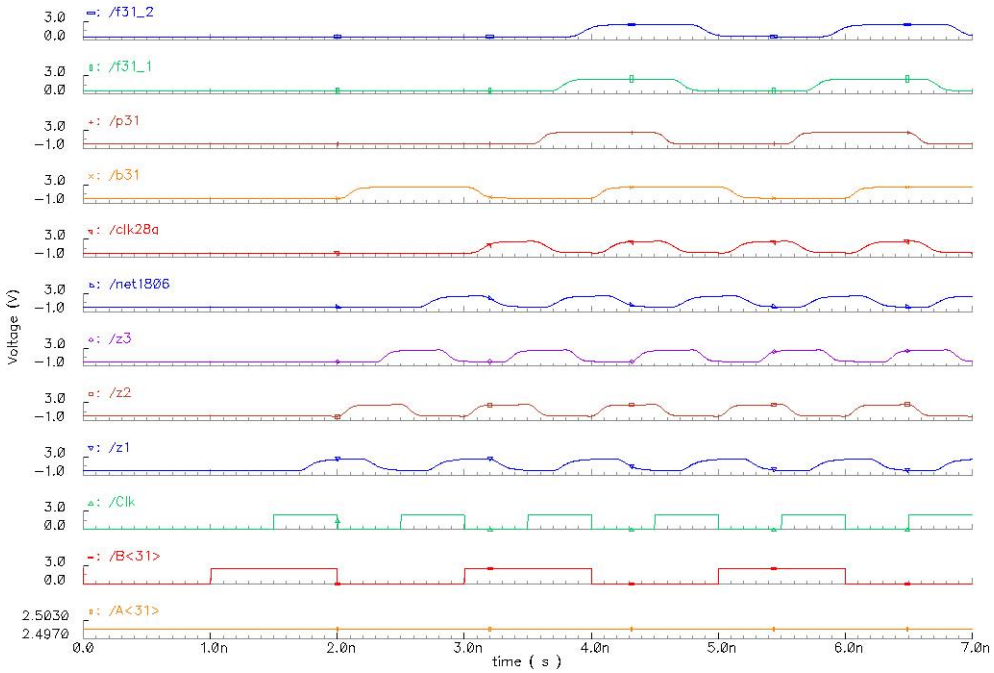

Figure 3.7: Clock Signal Traveling with Data Wave

CHAPTER 4

Data Dispersion

4.1 Introduction

Both wave-pipelining and Hybrid Wave-pipelining require that delay paths be balanced as much as possible. Introducing latches as in Hybrid Wave-Pipelining, permits for a less strict balancing process, since latches re-synchronize the data. If the longest path cannot be minimized to share the same delay as the shortest path, then the shortest path must be increased to match the delay of the longer. There is a risk of making the shortest path longer than the worst case path when balancing. There are many factors involved in why dispersion occurs and the three directly related to design include:

- *Data Dependencies.* Different input patterns to a given circuit or even gate can generate different response times at the output
- *Fan-in and Fan-out.* As the fan-in and fan-out increases so does the delay associated with the output of that gate. As the fan-out increases the capacitance the gate needs to drive decreases the response time.
- *Logic Paths.* Even if individual gates have been balanced for data dependencies different gates can have different delay paths.

We will look at an example of the three above occurrences of data dispersion in more detail, but it should be noted that there are many other factors that can affect data dispersion. Some of these include temperature variations, power supply noise, cross-talk [30] as well as process variations. Each of these factors have techniques associated with them to reduce their effect, but they will not be discussed in this thesis.

4.2 Data Dependencies

Data dependencies are the result of different input patterns causing different delay paths. This can easily be observed in a simple CMOS NAND gate. Figure 4.1 shows a 2 input CMOS NAND gate as it is typically designed. Figure 4.1 also shows the three different delay paths. Of the four combinations of input patterns, 00, 01, 10, and 11, there exist three different delay paths. A 00 case turns on both p-mos devices driving a one at the output. If a 11 case is applied at the input then both n-mos devices are active and a zero is seen at the output. Finally if either a 01 or a 10 is applied at the input one of the two p-devices is active and again a logic 1 appears at the output. Depending on the number of p-mos devices driving the output node the delay from input to output will change. If both p-type devices are on then the delay from input to output will be less than if only one device was on [30].

Even if care is taken to size the transistors appropriate to match rise and fall times a problem still occurs. The 01 and 10 case will always be slower than the 11 case in this circuit. The change in B resulting from 00 -> 01 or 00 -> 10 causes the delay to increase. A 11 case also can have a negative effect on delay if the series device closest to teh output is turned on before the device closest to ground. A trace of this problem is shown in figure 4.2. Because of this problem standard CMOS gates are not the best solution for solving the data dependency issue. Other balanced gates or biased gates provide better results at reducing the delay between different input patterns.

To further illustrate the problems with data dependencies biased and standard CMOS AND and XOR gates where simulated. The results for these four gates have been tabulated and reported in

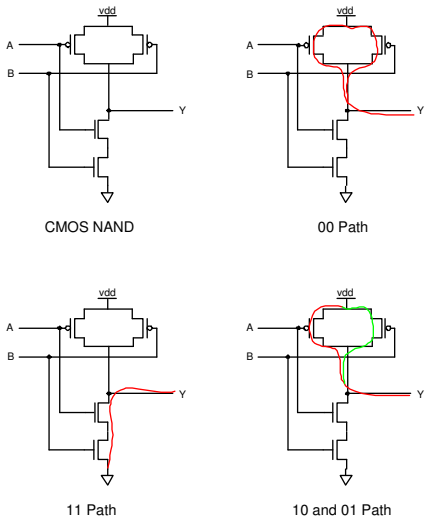

Figure 4.1: Data Dependencies of a CMOS NAND gate.

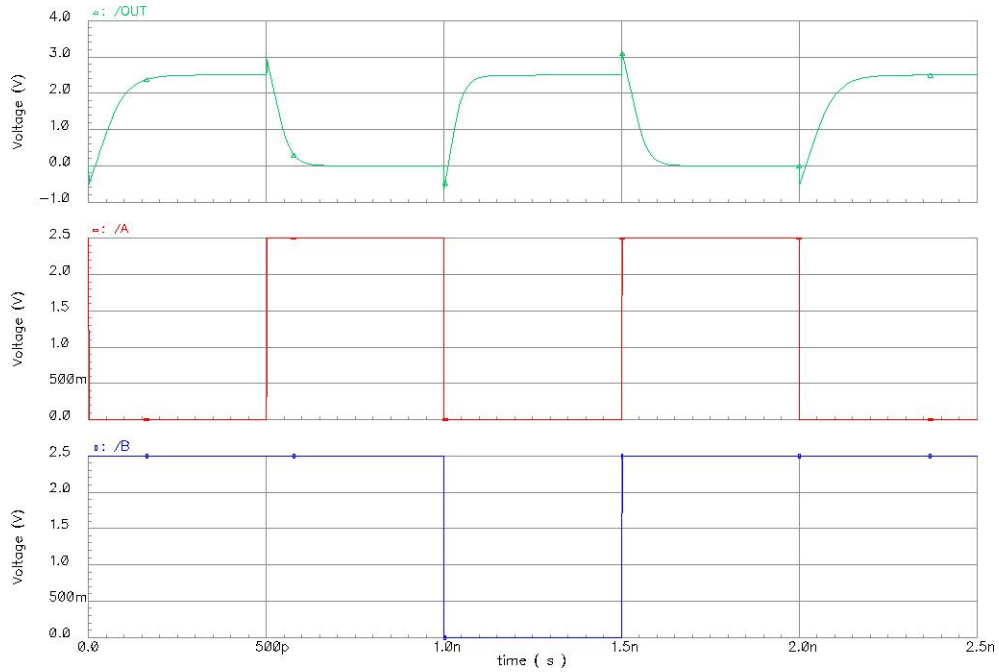

Figure 4.2: Wave Diagram of Data Dependencies of a CMOS NAND gate.

Figure 4.3 and Figure 4.4.

The CMOS XOR gate (shown in Figure 4.6) is one of the few CMOS gates other than the inverter that is fairly insensitive to data dependencies compared to biased gates. This is because the implementation of the CMOS XOR has the same number of pull up and pull down devices ON

	Delay	
Cases (A,B)	**CMOS**	**Biased**
01->11	94ps	71ps
11->00	64ps	88ps
11->01	101ps	85ps
00->11	101ps	73ps
10->11	93ps	71ps
11->10	94ps	85ps

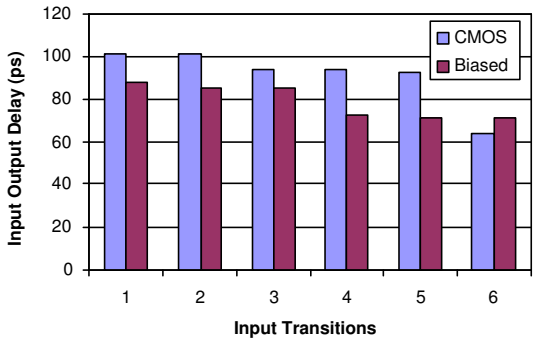

Figure 4.3: Input Output Delays (Standard CMOS and Biased AND)

	Delay	
Cases (A,B)	**CMOS**	**Biased**
00->10	39ps	86ps
00->01	68ps	86ps
11->01	39ps	85ps
11->10	67ps	85ps
10->11	68ps	118ps
01->11	45ps	118ps
10->00	48ps	108ps
01->00	69ps	108ps

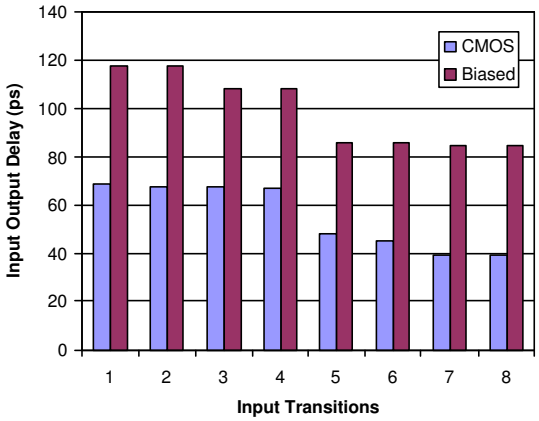

Figure 4.4: Input Output Delays (Standard CMOS and Biased XOR)

at the same time. These devices can be sized accordingly and as can be seen the data dependency occurs when one device has already charged its output node before the other turns on. Unlike the NAND gate there is no way of biasing the XOR that alleviates this problem.

Table 4.1 reports the average power dissipation of each gate for completeness. The biased logic gates dissipate more power due to the short circuit path whenever the series n-type devices are ON at the same time.

Table 4.1: Power Consumption (Data Rate 150 ps)

Gate	Average Power (μW)
Biased NAND	229.9
CMOS NAND	188.8
Biased XOR	290
CMOS XOR	164.2

4.3 Fan-in and Fan-out

Figure 4.5 shows the dispersion of data associated with different fan-outs. Even if gates are designed to be tolerant of data dependencies there can still be problems with data dispersion. In Figure 4.5 three different NAND gate outputs are represented, each with a different load. It can be seen that the outputs have a widely varying range in terms of time. When dealing with these cases one must keep in mind what each gate will be driving. Two possible solutions exist to solve the problem. One is to attempt to balance the gates by either loading all gates the same the other is to increase the drive capability of those gates with a higher fan-out. Obviously the second is the preferable of the two approaches because it will enhance system performance where the first option may add additional hardware or capacitance and will surely slow the system down. Sizing presents problems since input capacitance is increased by increasing the transistor size, therefore careful device sizing must be performed.

It should be noted here that loading affects the operation of all pipelined systems. However, it is especially detrimental to Wave-Pipelining and Hybrid Wave-Pipelining. This is because the speed at which the system can operate is governed by the difference between the fastest and slowest data-paths. Loading can significantly increase this difference. With conventional pipelining this is not a major problem for two reasons, the first being there is only one wave in the pipe at a time and the second is that the speed of operation is limited by the longest delay path to begin with.

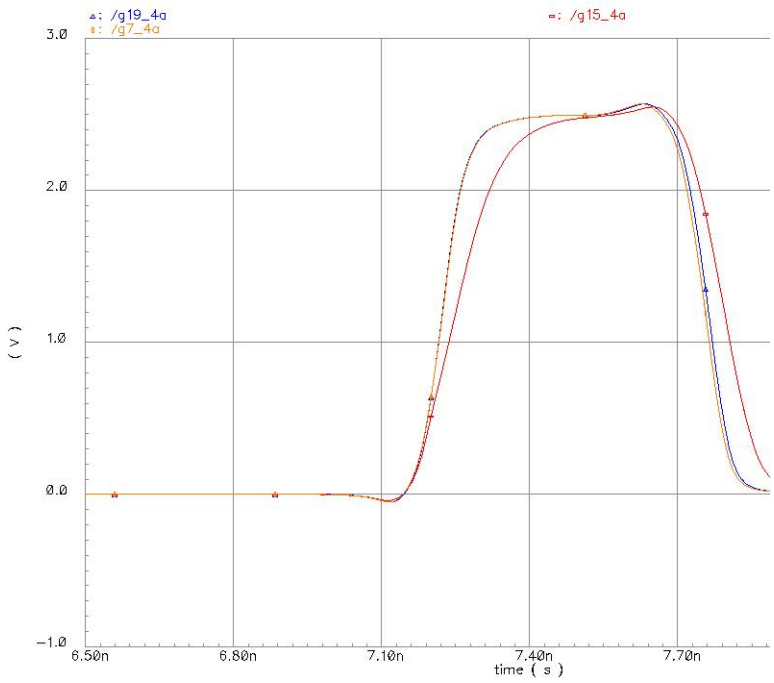

Figure 4.5: Wave Diagram of Data Dispersion due to Loading.

4.4 Circuit Paths

The last factor affecting data dispersion is the circuit path the data takes from input to output. This can best be illustrated by looking at the differences between two simple gates. For this example we will use an XOR gate and a NAND gate. Even if care has been taken to minimize the data dispersion due to loading and data dependencies it can be seen that there still exists data dispersion. This is easily explained as the NAND and XOR gate having different delays. Even if some cases between the two are balanced it may be impossible to balance all the possibilities.

Figure 4.6 shows a Biased-NAND gate as well as an XOR gate. Both have been balanced in terms of data dependencies and loading. We assume that both the inverted and non-inverted input signals arrive at the same time, a circuit that can be used to generate such signal arrival is shown in Figure 4.7. If we assume the inputs are a = 1 and b = 0 then the output of the NAND gate will arrive faster than the output of the XOR. Although this delay difference is very small in most cases if this is propagated through many gates it can grow quite rapidly. Figure 4.8 shows the resulting

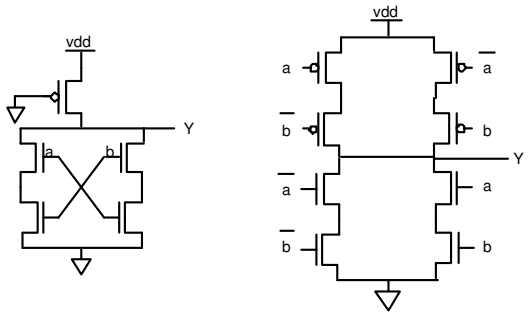

Figure 4.6: Biased NAND and CMOS XOR Gate

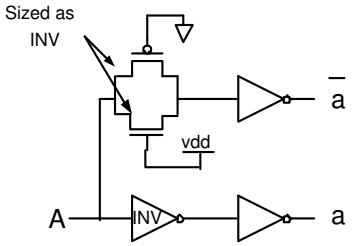

Figure 4.7: Circuit to match arrival of inputs a and $\bar{a}$.

waveform of such an event. Here the data has propagated through many different circuit paths and the dispersion is quite large. It should also be noted here that while most of this is due to the path the data has taken other factors have contributed as well including data dependencies, local interconnects as well as loading. This example comes from the a Hybrid Wave-Pipelined adder discussed in section 6.6.

This type of data dispersion is probably one of the most difficult to fix. If the two gates appear at the same level in the schematic it may be impossible to match the outputs perfectly. However if the delay is large one may be able to increase the delay of the slowest path by adding inverters or adding additional load to the output to force slower operation.

4.5 Conclusion

Data dispersion can be a problem for any pipelined system. It is especially a problem for Wave-Pipelined and Hybrid Wave-Pipelined systems. Data dependencies that results from changing input

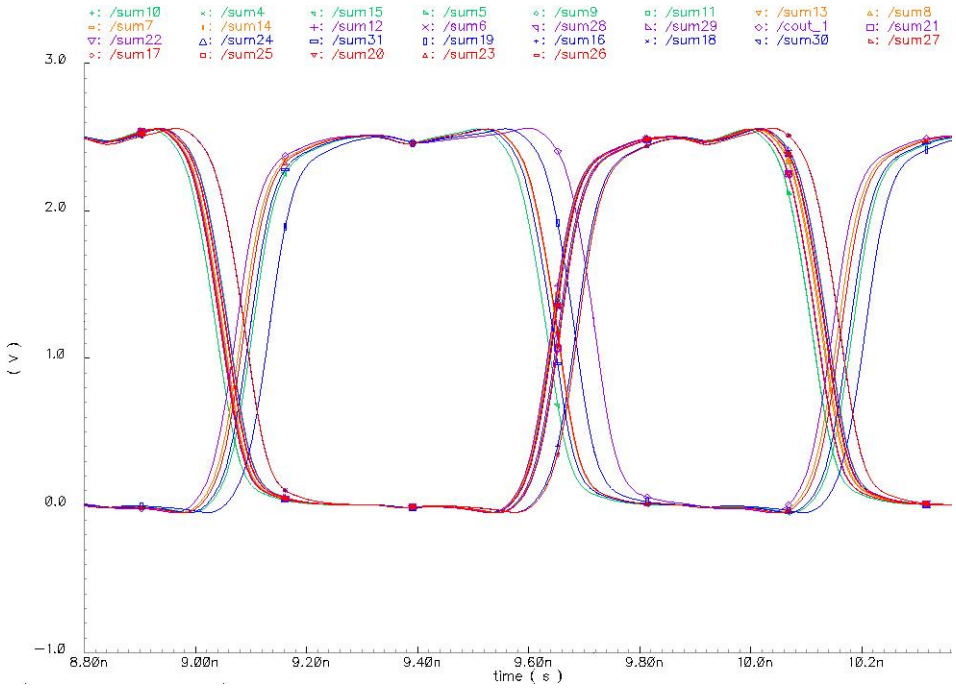

Figure 4.8: Accumulated results of Data Dispersion due to Circuit Paths

patterns can be eliminated by using special data independent circuits. Data dispersion due to fan-out or loading can be reduced by improving the drive strength of gates with a large fan-out or by adding additional load to gates with a smaller-fanout. The data dependencies that arrive from data taking multiple circuit paths is one of the most difficult to evaluate and solve. The delay can be reduced by designing gates and circuits that all have the same delay or by adding delay to slower gates and data paths in the design. Resolving these issues is a critical part of Hybrid Wave-Pipelining. Minimizing the data dispersion will greatly increase the speed at which the circuit will operate thus improving performance.

CHAPTER 5

Latches and D-Flip Flops

5.1 Introduction

Hybrid Wave-Pipelining like conventional pipelining requires the use of intermediate latches or registers. Unlike conventional pipelining however there are fewer registers, but the requirements on the registers are far more demanding. In the rest of this chapter we will look at static versus dynamic latches, edge triggered versus level sensitive latches and the overhead associated with registers as it pertains to Hybrid Wave-Pipelining. Section 5.2 will look at dynamic latches and flip-flops compared to static ones. Section 5.3 will outline edge triggered devices versus level sensitive ones. Section 5.4 will explain the consequences of the overhead associated with these devices and section 5.5 will comment on the use of these devices in regards to Hybrid-Wave Pipelining.

5.2 Dynamic versus Static

The difference between dynamic and static circuits is associated with memory. A dynamic device has no memory and relies on the parasitic capacitance at a node to store any charge until it is changed or "refreshed". A static device has some memory device that will maintain the voltage level until it is changed or until power is turned off.

Dynamic devices usually have a smaller area overhead but do not do well when trying to drive

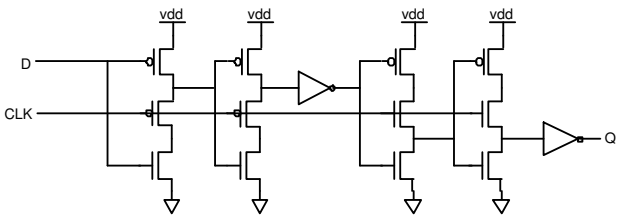

Figure 5.1: Dynamic Edge Triggered DFF

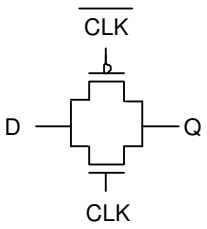

Figure 5.2: Dynamic Level Sensitive Latch

large capacitances at high frequencies. This is because there is a limited amount of time to charge and discharge the capacitance. If the period between discharging is large, the output signal may degrade causing problems in the circuitry. With static latches or flip-flops the area overhead may be larger but there is no worry about how long the output will hold charge assuming there is power to the device. If the time between data transitions is long, the memory associated with the device will maintain the previous voltage level without loss of information or degradation to the output signal.

Another problem with dynamic devices is their sensitivity to cross-talk. If nearby data signals are changed it is possible that capacitance between the lines may cause the output line of the dynamic device to vary. This variation may not effect the operation of the circuit or it could be so severe that the output actually changes states from a 0 -> 1 or from a 1 -> 0.

There are two issues of interest when using a dynamic device in Hybrid Wave-Pipelining. The first is that if the clock is pushing the limits of the technology it is very difficult to allow enough time for the dynamic devices to charge or discharge before the next wave arrives. Especially when one must consider the time needed to account for any clock skew in the system. The second concern is that the data must be propagated with the clock in an extremely efficient manner. Figures 5.1 and 5.2 show two different dynamic latches that could be used. Figure 5.1 shows a two stage true single phase latch presented by [36] and Figure 5.2 is a simple transmission gate.

When implementing Hybrid Wave-Pipelined systems it is best to use static devices. In using

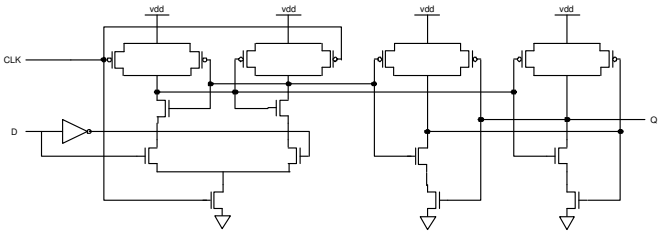

Figure 5.3: Static Edge Triggered DFF

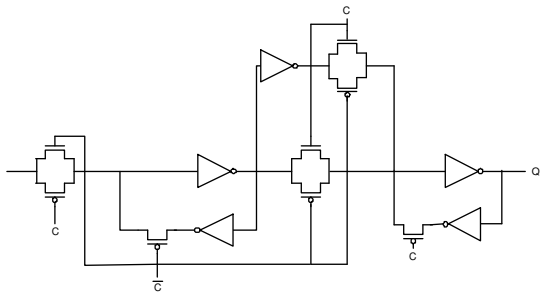

Figure 5.4: PPI Static Edge Triggered DFF

static devices the worry over the time to charge or discharge the output node is eliminated. Cross-talk is also less of an issue as long as power is supplied to the circuit. Figures 5.3 and 5.4 shows a selection of static type latches. Where Figure 5.3 is the one currently being used for our Hybrid Wave-Pipelining circuitry and is modified from [8] and Figure 5.4 is a Push-Pull Isolation (PPI) DFF [26].

5.3 Edge Triggered Versus Level Sensitive

The difference between level sensitive and edge triggered devices relates to what part of a clock signal data is transferred from the input to the output. With an edge triggered device the input is transferred to the output during the rising or falling edge of the clock signal. Depending on the edge data is transferred at, these devices are called “rising edge triggered” or “falling edge triggered”. A level sensitive device on the other hand transfers data between the input and the output as long as the clock is at a logic one or a logic zero. A device that transfers data while the clock is at a logic one is called “level sensitive high” and one that transfers data while the clock is at a logic zero is referred to as “level sensitive low”. For a level sensitive high device if the clock is high and data changes the change will be seen at the output. The converse is true for a level sensitive low device.

One problem with level sensitive devices is the “transparency” when the device is transferring data. If the data and clock are not synchronized and occur at the same time the output may latch in erroneous data. If the clock arrives late then the data may change and the wrong value will be

latched at the output. If the clock is too early then the data may arrive late and the transition from input to output will be missed altogether. It was also seen that as the speeds of the clock increases there simply may not be enough time to transfer all the data required if the registers/latches are dynamic ones. Such a device would be a simple transmission gate.

5.4 Overhead

Another concern is the overhead associated with the latch or flip-flop. The overhead is the time it takes from a clock transition to see the result at the output. The amount of time for the transition to occur will add to the entire latency of the system. In the case of conventional pipelining or wave-pipelining this delay must be accounted for. In regards to conventional pipelining this delay will affect the overall clock frequency the pipeline can be run at, while Wave-Pipelining must account for this extra overhead as extra time and circuitry in terms of delay matching. Figure 5.5 shows the time it takes for a DFF to propagate the input to the output of a "rising edge triggered" static DFF.

Another issue is the setup-time required by the device. The setup time required for an edge triggered device is the time the data must be stable before the triggering edge of the clock arrives. If this time is larger than the delay of the logic between registers it becomes the bottleneck of the system, this is especially true when the system is being run at very high frequencies.

When choosing whether or not to use edge triggered or level sensitive devices it is best to use edge triggered. This is because Hybrid Wave-Pipelining can operate at very high frequencies. As the clock frequency increases the ability to perfectly match the clock signal with the data decreases. For our system we found that even with careful balancing techniques there was still enough data dispersion that level sensitive devices would latch in wrong data. When trying to use simple level sensitive dynamic devices such as transmission gates the problem was only amplified. The need to have CLK and $\overline{CLK}$ added additional complexity in creating the two signals. Even with careful design generating CLK and $\overline{CLK}$ still resulted in clock skew that was above acceptable level for latching in strong signals.

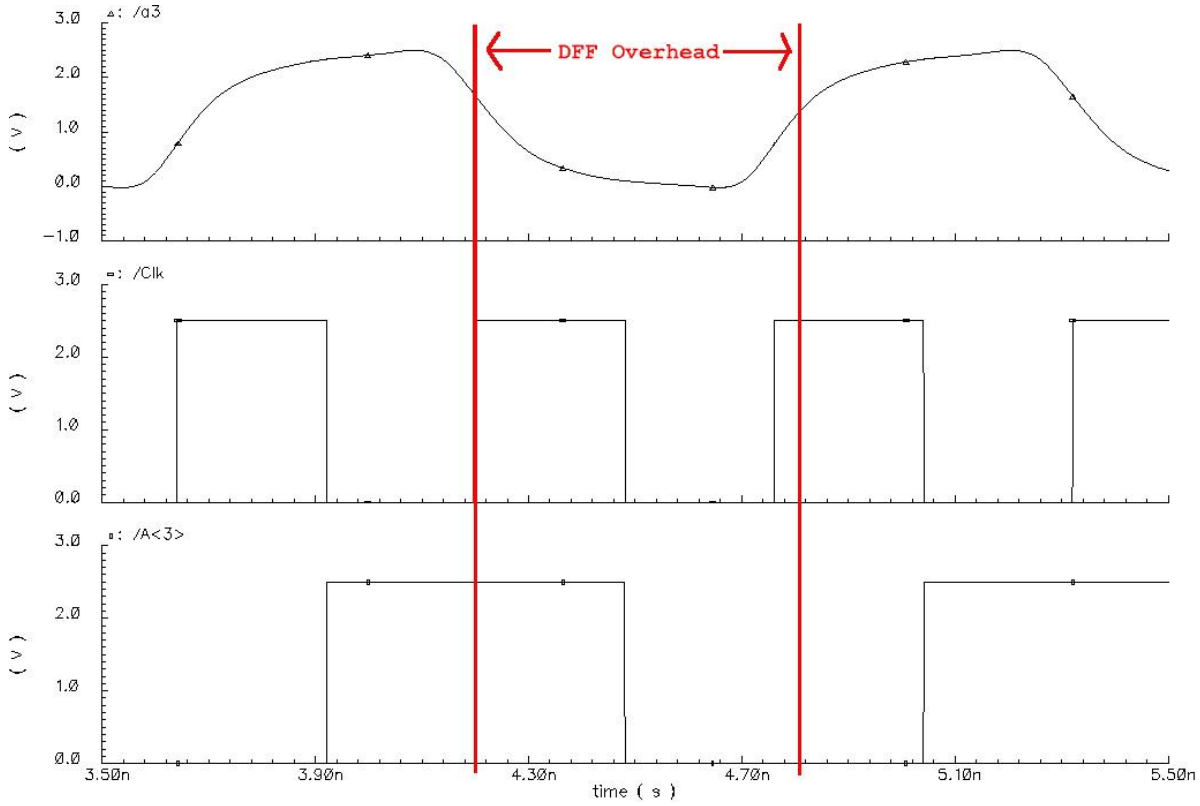

Figure 5.5: Overhead Associated with DFF from Fig. 5.3

The overhead is always a problem and the less there is the better. If the overhead of the latch can be reduced the circuitry needed to match the delay can be reduced as well. Also the amount of time it takes to propagate one wave through the system decreases. The overhead will only affect the throughput of the system if the latch itself is the bottleneck of the system, meaning that the latch overhead is greater than the logic's delay within a given stage.

5.5 Conclusions

For Hybrid Wave-Pipelining the latch is one of the key components. In regards to static versus dynamic devices Hybrid Wave-Pipelining can use either as long as the clock rate accommodates the charge and discharge time for dynamic devices. It is best if possible to use static devices. As was hinted to above the generation of CLK and $\overline{CLK}$ increases the area of the overall circuit and requires additional design time to create the two CLK's. If possible it is best to use a latch that does

not require both CLK and $\overline{CLK}$. If the delayed clock is well synchronized with the data waves then either "edge triggered" or "level sensitive" register can be used. If there is a problem synchronizing the clock with the data waves, "edge triggered" devices may provide relaxed requirements in regard to matching the clock with the data wave. The delays associated with latch overhead must be minimized in order for Wave-Pipelined and Hybrid Wave-Pipelined circuits to perform at there optimal potential.

CHAPTER 6

Adder Architectures

6.1 Introduction

In this chapter we will look at and review two groups of adder architectures. The two categories of adders that will be discussed are serial adders and parallel adders. Although many implementations of both categories exist we will only focus on a couple. The Ripple Carry Adder (RCA) will be presented in section 6.2, a newly proposed hybrid adder is discussed in section 6.3. The newly proposed hybrid adder is a combination of a carry lookahead (CLA) in conjunction with a RCA scheme. A parallel adder as implemented by Brent and Rung [3] will be outlined in section 6.4. Sections 6.5 and 6.6 will discuss the implementation of a parallel adder implemented using Wave-Pipelining and Hybrid Wave-Pipelining techniques respectively. A comparison is made between conventional pipelining, Wave-Pipelining and Hybrid Wave-Pipelining adders in section 6.7 and concluding remarks are found in section 6.8.

6.2 Ripple Carry Adder

Figure 6.1 shows the block diagram for a basic ripple-carry adder. The carry must propagate from the input (C_{in}) from left to right to the output (C_{out}). While the sum is generated from top to bottom as long as the carry into each block is valid.

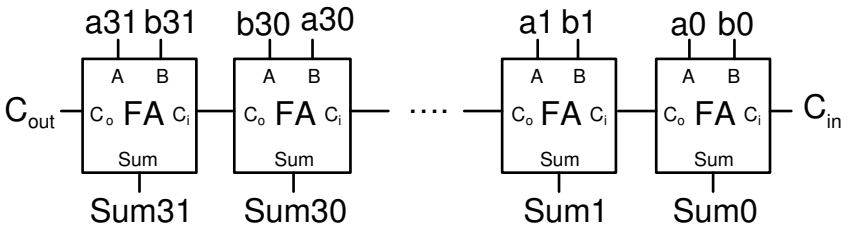

Figure 6.1: Block Diagram of a 32-bit Ripple Carry Adder

The equations for the simple full adders (FA) used are as follows:

$$sum_i = a_i \oplus b_i \oplus c_{in} \tag{6.1}$$

$$C_{out} = a_i b_i + b_i c_{in} + a_i c_{in} \tag{6.2}$$

The ripple carry adder is limited by the need to propagate the carry from the input to the output. As the number of inputs increases, the delay associated with this path increases as well. The delay dictates the speed at which the adder can operate. The delay of the adder (t_{adder}) can be expressed as follows:

$$t_{adder} \approx (N-1)t_{carry} + t_{sum} \tag{6.3}$$

Where N is the number of input bits, t_{carry} is the time to propagate C_i to C_o and t_{sum} is the time to propagate the inputs to the sum. Much effort has been put into reducing the delay associated with ripple carry adders [4] and [15].

6.3 Hybrid RCA/CLA

In order to minimize the delay a newly proposed architecture that combines the feature of a CLA with prediction in a RCA format is proposed. Although the carry still needs to propagate along the adder because of the carry lookahead features the delay is reduced considerably. First we will

present the basic architecture for a carry lookahead adder. Followed by an explanation of the new CLA scheme with prediction.

6.3.1 Carry Lookahead

In this section we discuss the fixed group-4 propagate and generate carry lookahead adder on which we base our prediction scheme and the new CLA with carry prediction. The logic equations describing the carry out signals generated per bit position show that these signals are produced in parallel [39], however, there is a ripple effect when the sums of the subsequent blocks have to be generated. This is so because the carry-out generated by the 4^{th} bit position of a 4-bit block needs to be propagated to become part of the inputs to the next block. The cascading effect of the carry-out leads to considerable delays. The standard carry lookahead adder equations that dictate if the carry will be generated or propagated culminating in the sum being generated are reproduced here just for convenience. We use $g_i = a_i \bullet b_i$ and $p_i = a_i \oplus b_i$, where g_i and p_i are the generate and propagate signals respectively with inputs at bit position *i*. The sum is given by:

$$s_i = c_{i-1} \oplus a_i \oplus b_i \tag{6.4}$$

while the carry of the i^{th} bit stage is:

$$c_i = g_i + p_i \bullet c_{i-1} \tag{6.5}$$

Equation 6.5 when expanded becomes:

$$c_i = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + ... + p_i ... p_1 c_0 \tag{6.6}$$

It is common knowledge that the size and fan-in of the gates needed to implement the carry-lookahead scheme increases as the number of input bits grows. Limiting the carry lookahead to

four bit blocks allows for breaking the resultant delay in carry generation. This is the approach taken in this study. Expanding Equation 6.5 makes it apparent that each carry bit can be generated independent of the previous carry (6.6), however, each sum bit depends on the value of the previous carry bit. This observation has led to layered implementations of carry lookahead adders with the first layer producing the g_i and p_i terms, the second layer producing the carry bits and the third layer producing the sum. Arranging the adder in this form lends itself well to pipelining, however, as aforementioned an increase in the number of inputs leads to a long delay due to an increase in the number of logic gates required to generate the carry and the dependency of the sum on the carry of the previous bit positions. The fourth carry (c_3) for instance will require four AND gates, with one having a fan-in of 4 and one 5-input OR gate. The fifth carry (c_4) will have 6 AND gates (one having a fan-in of 6). The description above does not necessarily dictate that the hardware to implement these logic terms must include the number of gates cited. Many innovative ways of breaking this delay and enhancing the performance of adder circuits have been developed and used over the years. Performance has further been improved owing to the capability to scale transistors [7], [9].

6.3.2 Ripple Carry Lookahead

The basic implementation of carry lookahead adders has involved limiting the carry generation circuitry to only produce up to the 4^{th} carry bit per block (fixed group-4). Doing this reduces the number of gates needed to generate the carry as explained in section 6.3.1. This scheme is shown in Figure 6.2. Blocks are therefore, cascaded together in a ripple carry adder fashion such that the 4^{th} carry bit of each block is the carry input to the succeeding block. Therefore, the 4^{th} carry bit of the 1^{st} block will be the carry input to the 2^{nd} block while the 8^{th} carry bit of the 2^{nd} block will be the carry input to the 3^{rd} block, and so on. This approach results in a ripple effect. The inputs of all the blocks are presented at the same time, enabling each block to commence addition in parallel, however block 1 has to wait for block 0 to produce a carry, similarly block 2 waits on block 1,

resulting in a ripple effect as the most significant carry signals of each block must be available as input to the next stage. The focus of many researchers has been to reduce this carry propagation delay in order to improve performance as discussed in [6], [20], [14].

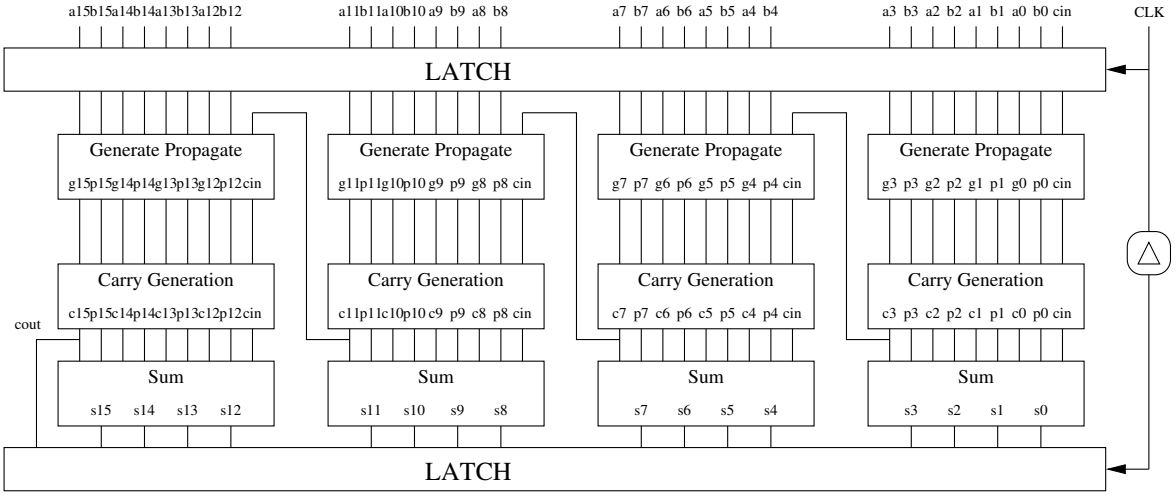

Figure 6.2: Three level block diagram of 16-bit CLA without prediction

6.3.3 Carry Prediction Scheme

We have designed a carry lookahead adder that attempts to break the ripple effect of the carry signal by predicting the value of the carry-out each block has to propagate to its nearest neighbor. The prediction scheme accepts the inputs at the same layer or gate level as the propagate and generate circuitry and determines if the next block will receive a carry-in of one or zero and provides this carry long before the carry generating block can evaluate this condition. This permits the next block to start adding the received inputs without having to wait for its previous neighbor to produce its most significant carry-out. Our prediction scheme is achieved by considering the upper three bits of both addends at each 4-bit block. Their logic values enable for early detection of what the carry-in of the next block will be. The expression derived to perform this detection is provided in equation 6.7 and is somewhat similar to the logic used for propagate and kill signals for the Manchester carry lookahead adder [9]. The representative logic expression for the prediction logic is given by:

$$C_{out} = a_3b_3 + a_2b_2 \bullet (a_3 + b_3) + a_1b_1 \bullet (a_2a_3 + b_2b_3 + a_2b_3 + b_2a_3) \tag{6.7}$$

This expression obviously has many ANDed terms and requires a maximum fan-in of 4; however, the carry prediction still takes place much earlier than the computed carry at the second layer of the CLA. It should be noted that one advantage to this scheme is that prediction occurs at the same time the generate and propagate signals (which are used to produce the carry) are computed. The fan-in of the OR gate also gets large. We show a block diagram of this scheme in Figure 6.3. The figure shows that the predicted carry is multiplexed with the 1^{st} carry bit instead of the 4^{th} carry bit. This results from the observation that whenever a prediction cannot be made the resulting 4^{th} carry-out bit has the same logic level as the 1st carry-out bit. Implementing the prediction scheme in this manner results in the elimination of circuitry required to generate the 4^{th} carry. The 1^{st} carry bit requires less circuitry to generate and can thus be propagated very fast to the next block.

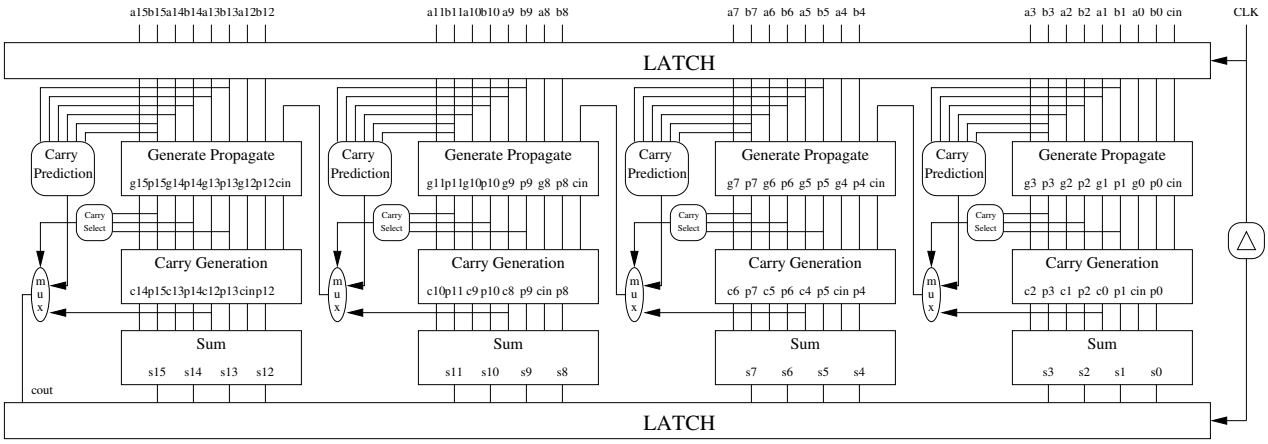

Figure 6.3: Three level block diagram of CLA with carry-out prediction based on three upper bits

Using the three upper bit pairs result in a situation in which only 8 out of the 64 possible cases cannot predict the carry-out required for the next block. These cases are shown in Table 6.1. In the event that the carry-out cannot be predicted, the scheme relies on p_1, p_2 and p_3 to notify the system

that prediction could not be made in which case the next block is made to wait for the computed carry-out, thus the worst case operation is encountered. This worst case delay is better than the one encountered if the approach of Figure 6.2 is used because the 1^{st} carry bit is propagated instead of the 4^{th}. As stated earlier the 1^{st} carry requires less circuitry to generate and is thus available much earlier than the 4^{th} carry.

Table 6.1: Input patterns that result in no prediction

a3	b3	a2	b2	a1	b1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	0	1	0
1	0	0	1	0	1
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	0	1	0

If the current block (the one whose carry-in could not be predicted) can make a prediction of its most significant carry out, the next block can perform its computations without being affected by its preceding neighbor. This scenario still has some considerable advantage in terms of performance. The worst case operation occurs when no CLA block can make a prediction therefore propagating the carry-out of the adder to the next block. In contrast, in the event that prediction can be made, any CLA block whose carry-input was predicted from the previous block will execute in parallel. Thus, the best case operation is one in which all blocks can predict and the delay is approximately equivalent to that of one 4-bit CLA block. If we consider that the input patterns have an equal probability to occur we can show that only 12.5% of the time prediction would not be achieved and we default to the worse case. We thus have 87.5% of operation with the capability to make prediction. This leads to considerable performance improvement. The prediction circuitry dictates that there be a multiplexer in the design in order to enable for a selection between the predicted

carry and the computed carry in the event that prediction cannot be made. The circuitry designed to control the multiplexer is at the level of the carry-out generation circuitry, but can be designed using a single 3-input NAND gate enabling this control signal to be generated much earlier than the carry-out can be computed.

In the event that prediction cannot be made based on the evaluation of eq 6.7, the obvious course of action would be to default to the standard method of generating the 4^{th} carry-out bit based on:

$$c_3 = g_3 + g_2p_3 + g_1p_3p_2 + g_0p_3p_2p_1 + p_3p_2p_1p_0c_{in} \tag{6.8}$$

This would lead to degradation in performance whenever prediction cannot be made. We note that whenever prediction fails, the logic value of the 1^{st} carry-out bit is the same as the logic value of the 4^{th} carry-out bit. This therefore means that we can use the 1^{st} carry-out bit as part of the inputs of the succeeding block. If our simple delay estimation holds, we can show that c_0 is computed long before c_1, c_2 or c_3. This can be verified by comparing the equations defining these carry-out values, represented in this paper by the general expression of eq 6.6. The following analysis will prove that whenever prediction cannot be made $c_3 = c_0$. The condition that results in a failed prediction occurs when:

$$p_1 = p_2 = p_3 = 1 \tag{6.9}$$

$$c_0 = g_0 + p_0c_{in} \tag{6.10}$$

We examine eq 6.8 for the condition satisfying eq 6.9 and note that we have:

$$g_1 = g_2 = g_3 = 0 \tag{6.11}$$

For this condition eq 6.8 reduces to:

$$c_3 = g_0 p_3 p_2 p_1 + p_3 p_2 p_1 p_0 c_{in} \tag{6.12}$$

Since, $p_1 = p_2 = p_3 = 1$ we have that

$$c_3 = g_0 + p_0 c_{in} = c_0 \tag{6.13}$$

We thus can propagate c_0 whenever a condition in which prediction cannot be done occurs.

Simulations of a 16-bit carry lookahead adder have been performed with the carry prediction scheme being implemented in pseudo NMOS. Figure 6.4 shows the circuit used to realize the prediction equation. Further optimization of this circuit is possible, but is not depicted in this figure. Several transistors that can be eliminated are duplicated in the circuit for clarity.

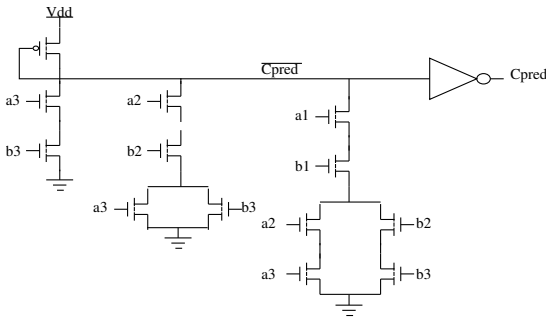

Figure 6.4: Circuit used in Prediction

Performance analysis involved evaluating all possible cases that could result in performance degradation. The simulation results were compared to those of a carry look-ahead adder without a prediction scheme. Figure 6.5 shows the delays measured from the adder without the carry-out prediction logic. After the inputs change it takes 3.083 ns for the sum to be computed.

If the CLA with a prediction scheme is considered, a delay of 2.382 ns is recorded. These are worse case delays when the full 16-bit addition is performed. The prediction scheme shows a

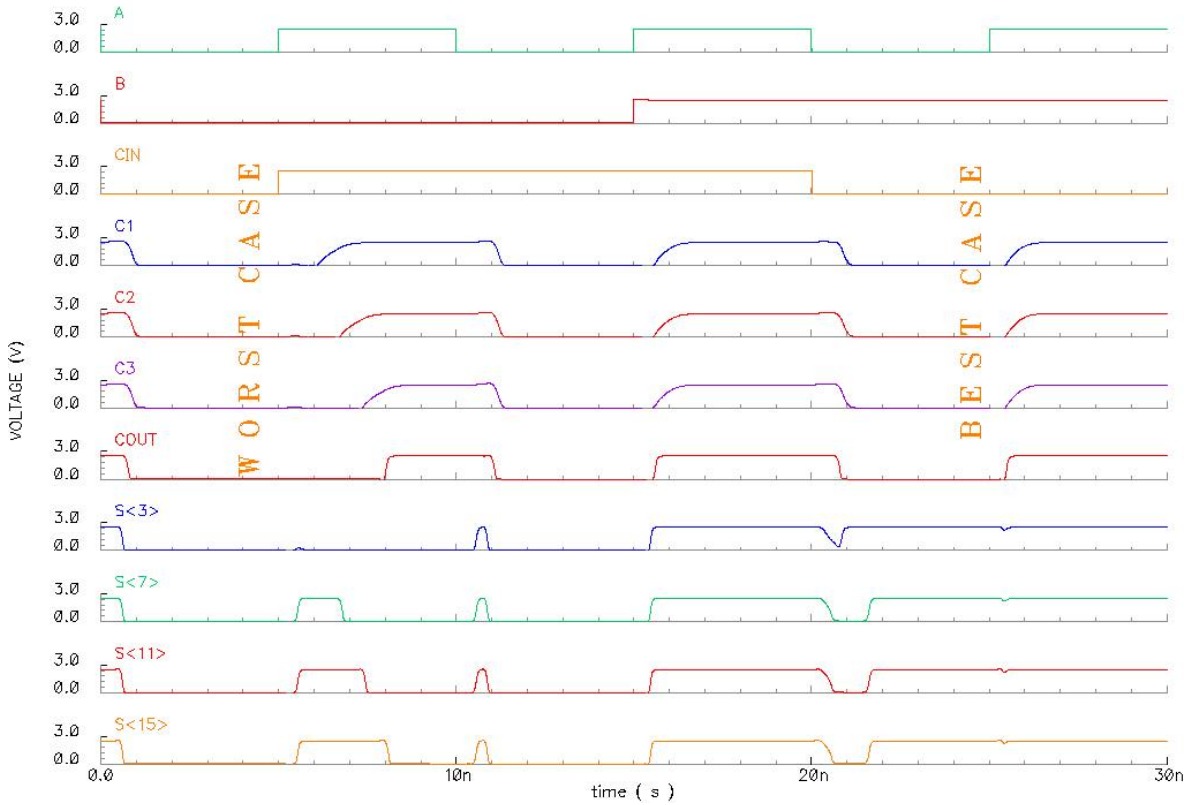

Figure 6.5: Simulation Results of Standard CLA Adder.

22% improvement over the case when no prediction is used. The best case of the carry prediction approach yields 67% performance improvement and occurs when prediction is successful at all blocks of the adder. Figures 6.5 and 6.6 are traces showing the reported values graphically. To obtain the traces on Figure 6.6, a case in which the first, third and fourth blocks successfully predict the carry-out signals while the second block fails to predict.

We have also measured power dissipation and determined that our CLA implementation dissipates 1.5% more power than the adder without the prediction scheme. We attribute this small increase in power dissipation to the use of pseudo-NMOS design style. The ability to propagate the 1^{st} carry-out to the next block results in reduction of transistor count since the circuitry required to generate the 4^{th} carry-out is no longer required. Predicting the carry saves 15% per carry look-ahead block in transistor count and provides performance gains of 22-67% with only 1.5%

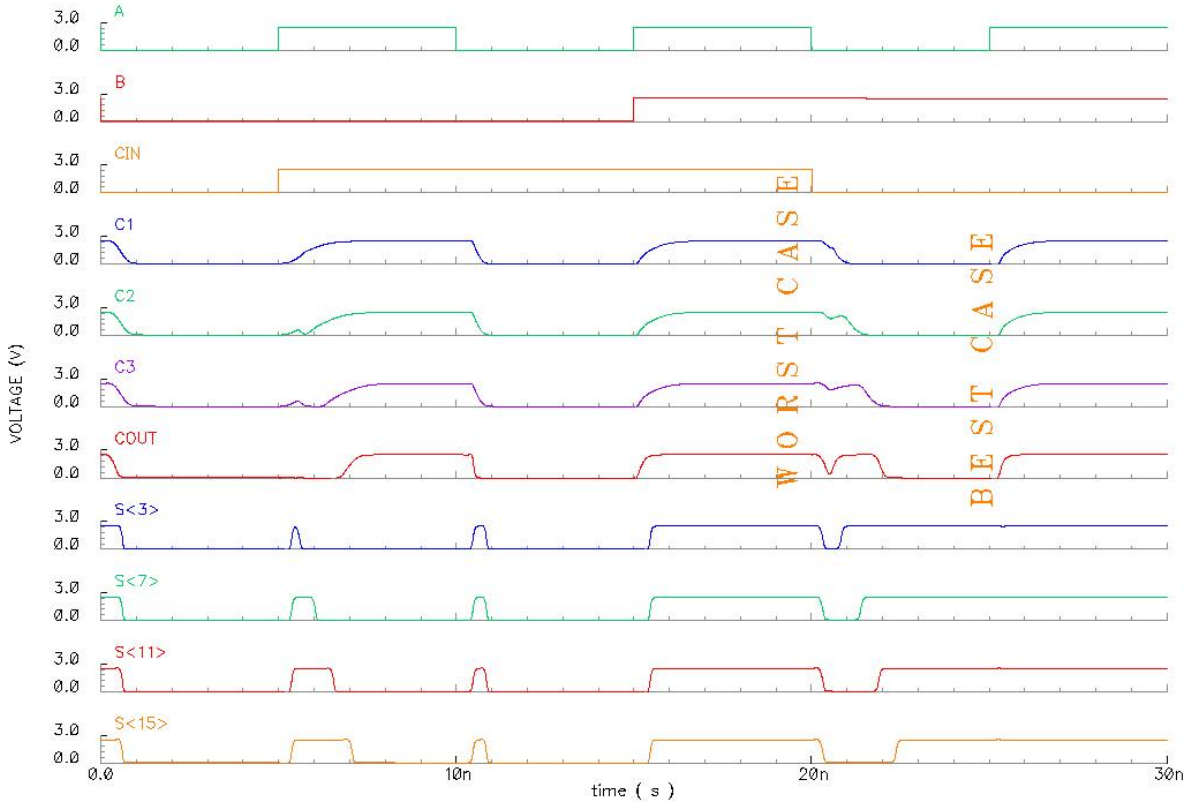

Figure 6.6: Simulation Results of Standard CLA Adder with Prediction.

increase in power dissipation. This scheme may lend itself usefull to pipelining but requires an extensive amount of additional registers in order to compensate for the adder propagating data in two directions, both horizontally and vertically such an adder has been presented by [38]

6.4 Parallel Adders

6.4.1 Introduction

Parallel adders in current CMOS technologies provide very fast addition and we thus have to consider them. In smaller CMOS technologies parallel adders fan-in and fan-outs might become unmanageable leaving serial adders as a viable option. In order to minimize the delay associated with rippling a carry, parallel-prefix adders attempt to reduce the need to propagate the carry by calculating it in parallel with the sum. Although there exists many parallel prefix adders, Brent

and Kung [3], Kogge and Stone [27] and [18], this thesis will mainly focus on the implementation originally presented by Brent and Kung and how it can be applied to Wave-Pipelining and Hybrid Wave-Pipelining. We will first look at the generic concept and architecture and finally focus on the Brent and Kung implementation as it pertains to Wave-Pipelining [30] and Hybrid Wave-Pipelining. In section 2 we will look at the general equations and layout for a parallel prefix adder. Section 3 will present a pipelined and Wave-Pipelined adder and Section 4 will look at our new approach using Hybrid Wave-Pipelining.

6.4.2 Background

The block diagram for a parallel prefix adder is shown in figure 6.7. The diagram is broken up into three main portions. The generate and propagate generation block, the carry generation block and finally the sum block. The generate propagate block (labeled (g, p) generator) takes the inputs a_i and b_i and creates the p_is and g_is. The p_is and g_is are then inputs to the carry generator block which computes the carries (c_is). The final block or sum block then takes the generated c_is and p_is and computes the sum through an XOR operation

The parallel adder propagates from the bottom up with all calculations being done in parallel. The adder is also very regular reusing the same components throughout the carry generation block. If the number of inputs is increased the delay increases only if the number of layers must be increased. The number of layers is dependent on the number of inputs. A new layer is required whenever the inputs reach a new power of 2^x where x becomes the number of layers needed in the carry block.

In order to fully understand the proposed tree structure presented in [3] a set of recurrence equations are defined over bits a_i and b_i for all *i* as follows:

$$g_i = a_i \wedge b_i \tag{6.14}$$

$$p_i = a_i \oplus b_i \tag{6.15}$$

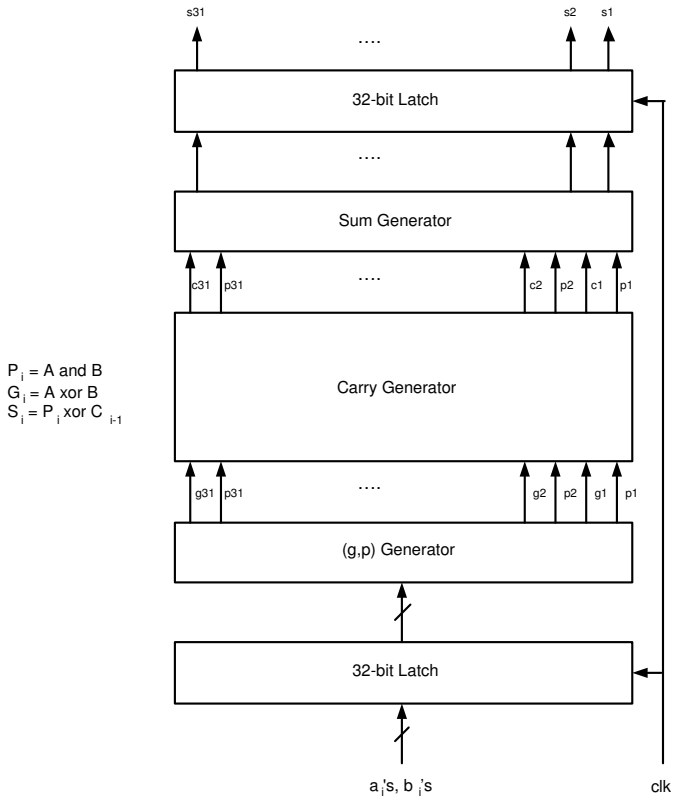

Figure 6.7: General Block Diagram for a Parallel Adder.

$$c_i = G_i \text{ for } i = 1,2, ...,n \tag{6.16}$$

where

$$(G_i, P_i) = (g_1, p_1) \text{ for } i = 1 \tag{6.17}$$

$$(G_i, P_i) = (g_i, p_i) \circ (G_{i-1}, P_{i-1}) \text{ for } i > 1 \tag{6.18}$$

the $\circ$ is a concatenation operator defined as:

$$(g_y, p_y) \circ (g_x, p_x) = (g_y \vee p_y \wedge g_x, p_y \wedge p_x). \tag{6.19}$$

When all the carry bits c_i have been generated the sum bits s_i are computed with:

$$s_i = p_i \oplus c_{i-1} \text{ for } i > 1 \tag{6.20}$$

$$s_1 = p_1 \text{ for } i = 1 \tag{6.21}$$

With this set of recurrence relations defined it is now possible to explore the use of parallel adders when a wave-pipelined or hybrid wave-pipelined scheme is applied

6.5 Wave-Pipelined Parallel Adder

The regular structure of the adder presented by Brent and Kung make it very applicable to Wave-Pipelining. The complexity of this adder from a architectural stand point is centered around the computations of the carry block. Figure 6.8 shows the carry block for a modified Brent and Kung carry adder as done in [30]. In this diagram the block has half of the computational depth of the Brent and Kung adder at the addition of larger drivers because of the increased fan out. It also has four basic computational components rather than two.

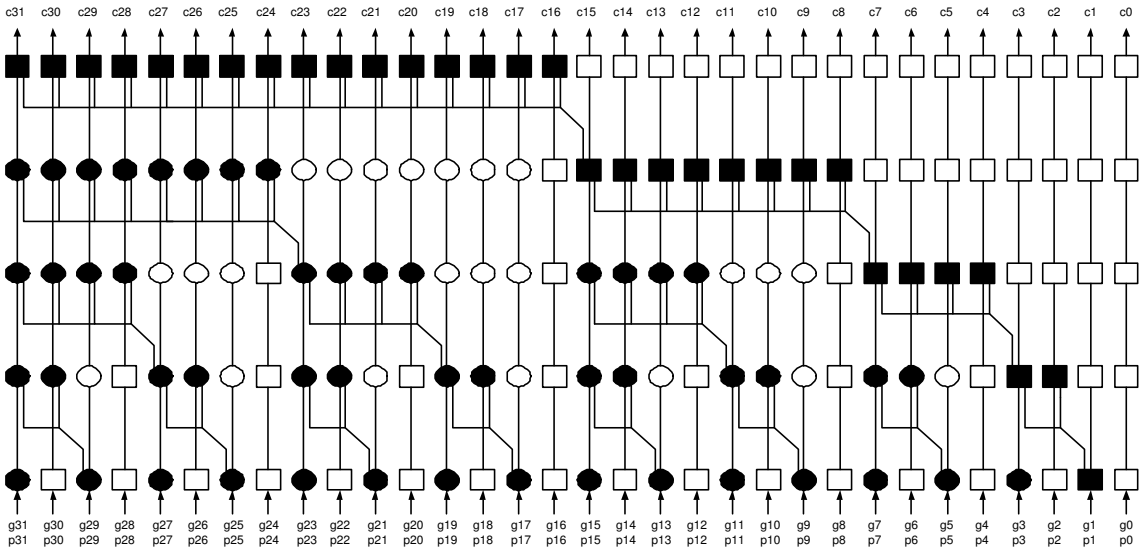

Figure 6.8: Modified Carry Block in expanded Tree Form.

In order to balance the delay paths special computational cells are used in the carry block. The

squares and circles of the carry generation are elaborated in figure 6.9 and are a notation from the paper by Brent and Kung [3]. The shaded blocks perform the required computations as defined in equation 6.19 while the white or un-shaded blocks are used as "padding elements" in order to generate the necessary delay for wave-pipelining.

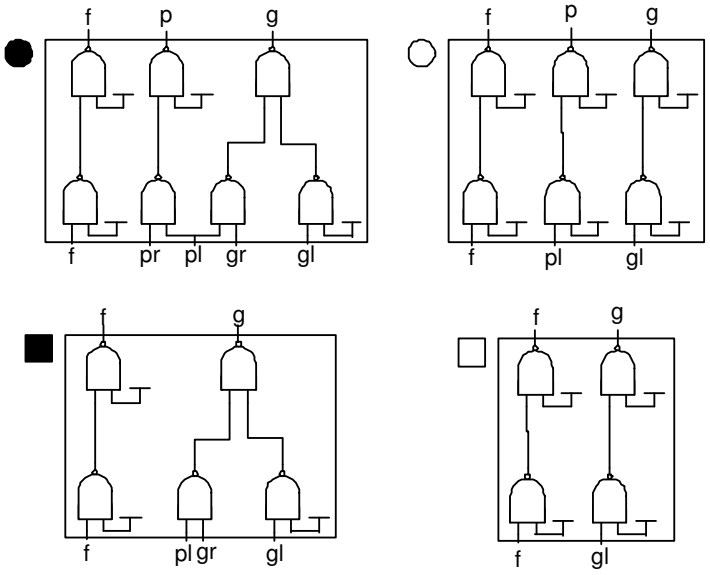

Figure 6.9: Blocks Used in Computation of Carries.

The gate shown in figure 6.10 is a biased NAND gate used to implement the objects shown in figure 6.9. This architecture comes from [30] and was used as a stepping stone to create the hybrid wave-pipelined adder which will be presented in the next section. The structure of the adder remains unchanged except that biased NAND gates are used in order to create balanced delay paths. The XOR gate used to create the propagate signals (g_i) and the final sum computation is shown in Figure 6.11. The additional circuitry in the figure is needed to guarantee that a, $\overline{a}$, b and $\overline{b}$ all arrive at the gate with the same delay.

The entire adder architecture including expanded carry block is presented in Figure 6.12, this adder was optimized so gates needing to drive large loads could do so in approximately the same amount of time as gates driving smaller ones. All of the equations outlined in chapter 2 dealing

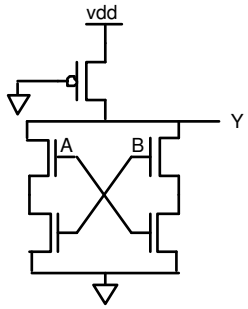

Figure 6.10: 2 Input Biased NAND Gate.

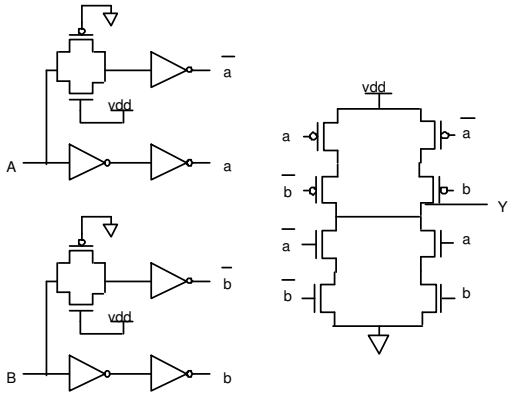

Figure 6.11: CMOS Xor with circuitry to balance inputs.

with wave-pipelining still apply.

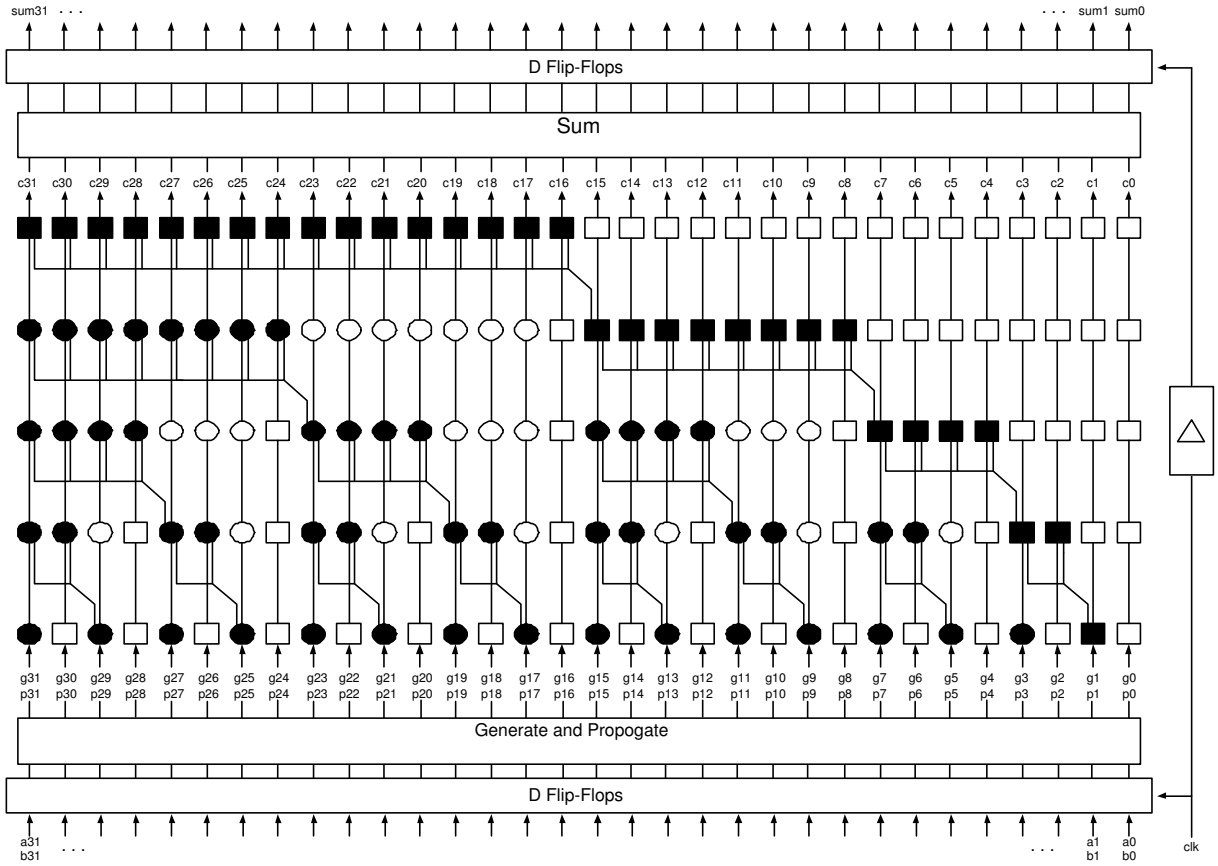

Figure 6.12: Wave-Pipelined Adder with Expanded Carry Block

The Wave-Pipelined adder was implemented using Cadence schematic tools in TSMC .25μm technology. The fastest operation of the adder is recorded at 670ps or 1.49 GHz. The adder was not optimized as aggressively as reported in [30]. Wave-Pipelining as mentioned early operates in regions of disjoint points. The upper end of this implementation can operate in a range between 780ps and 670ps before failure. The period in which operation cannot be sustained for the upper end is 835ps to 790ps. The maximum sustainable number of waves of this Wave-Pipelined adder is three with one wave leaving the system as another wave enters. Figure 6.13 shows the resultant output waveform for the Wave-Pipelined Adder running at its maximum operating frequency.

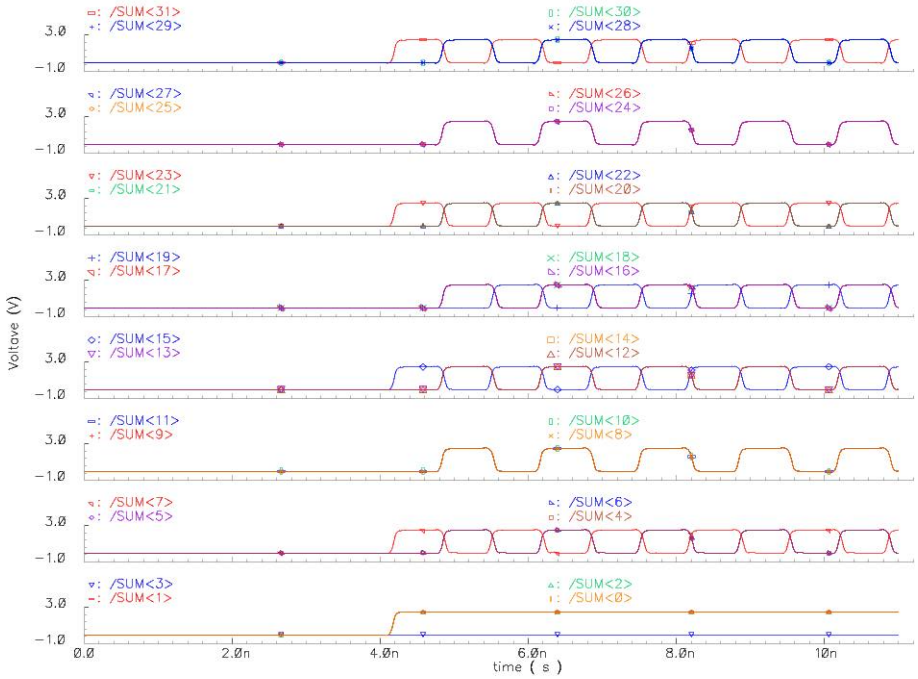

Figure 6.13: Simulation Results of Wave-Pipelined Adder

6.6 Hybrid Wave-Pipelined Parallel Adder

The hybrid wave-pipelined adder is presented in figure 6.14. As can be seen in the figure internal registers are reinstated into the design in order to synchronize the data at given points. Again biased NAND gates where used to create the cells in the carry-block. The design is broken apart

into three stages each having its own delay associated with it. Unlike the wave-pipelined scheme the clock is delayed to match the data as it propagates through the circuit. The delay of each stage is labeled as Δ_1, Δ_2 and Δ_3 in the clock path. For this design the delays where generated using inverters in a matched RC Tree formation as they provide strong signal strength and are easy to modify in order to balance delay and load.

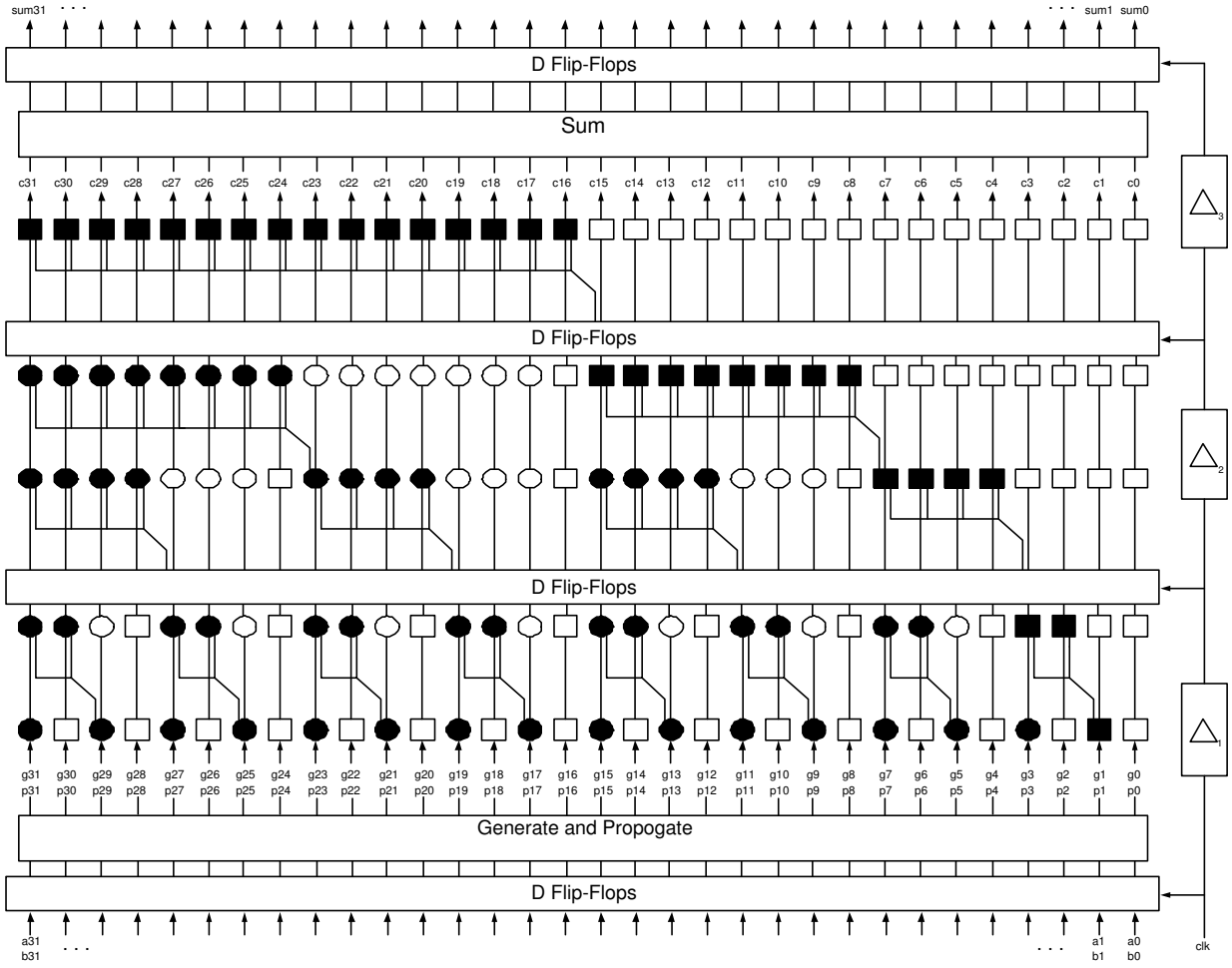

Figure 6.14: Hybrid Wave-Pipelined Adder with Expanded Carry Block

The maximum operating rate for the Hybrid Wave-Pipelined adder is 560ps. This was obtained using Cadence schematic tools and simulations in a TSMC .25μm technology. Unlike the Wave-Pipelined adder design there is no bound on how slow the clock rate can be run at. The number of

sustainable waves is reported per stage; stage 1 can maintain three waves, stage 2 can sustain two waves and stage 3 can again sustain three waves. Combining the stages shows the total maximum number of waves the system can maintain is eight waves. Figure 6.15 shows a wave form diagram of the final output of the Hybrid Wave-Pipelined adder running at its maximum speed.

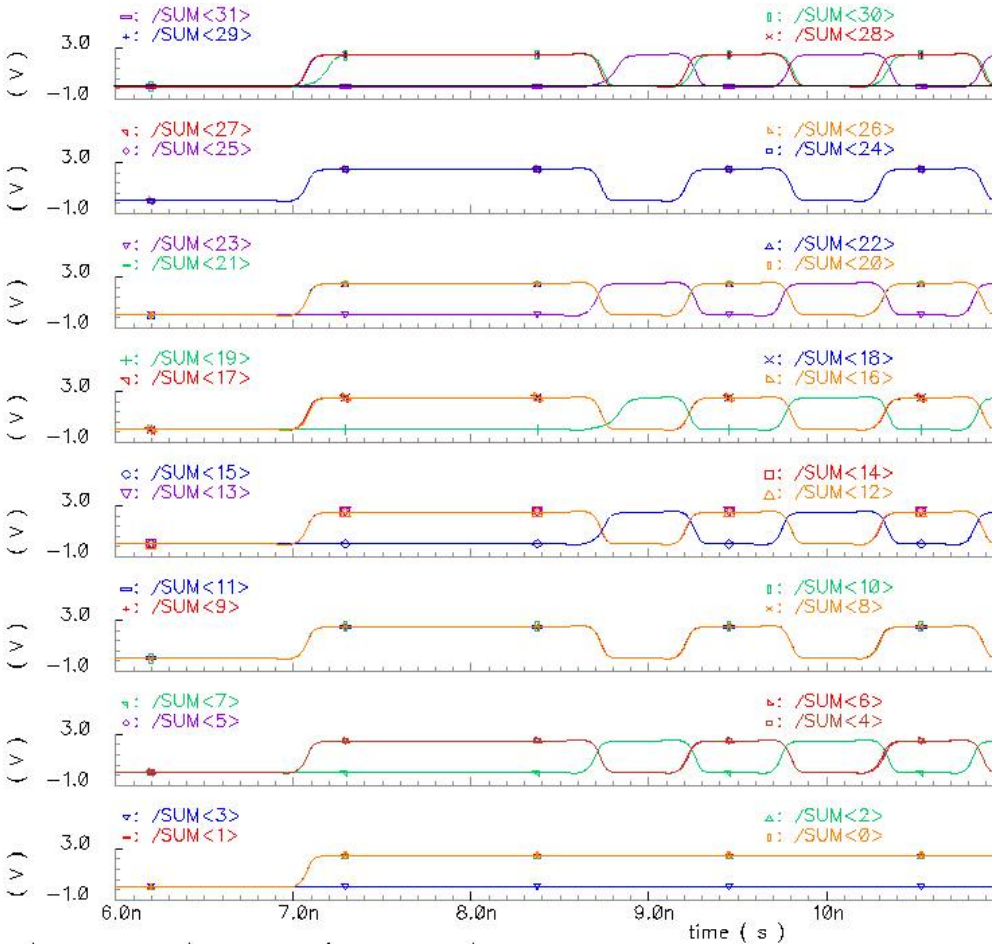

Figure 6.15: Simulation Results of Hybrid Wave-Pipelined Adder

The data delay through each stage dictates the overall speed the adder can be run at. It also determines how much the clock itself should be delayed. The maximum and minimum delays per stage are reported in table 6.2. These delays are for the logic path between registers only and does not account for register overhead.

We have used the same architecture to implement the conventional pipelined adder. The delay

Table 6.2: Maximum and Minimum Data Delays per Stage.

Stage	Minimum Delay	Maximum Delay	Difference
1	788ps	900ps	112ps
2	346ps	412ps	66ps
3	822ps	986ps	164ps

matching circuitry is eliminated. It also must be noted that for both Hybrid Wave-Pipelined and conventional pipelined adders it would be ideal to partition the stages such that the internal registers become the bottleneck of the system instead of the logic between the stages. This could result in faster stages. Figures 6.16 shows the waveform for the conventional pipelined adder described above.

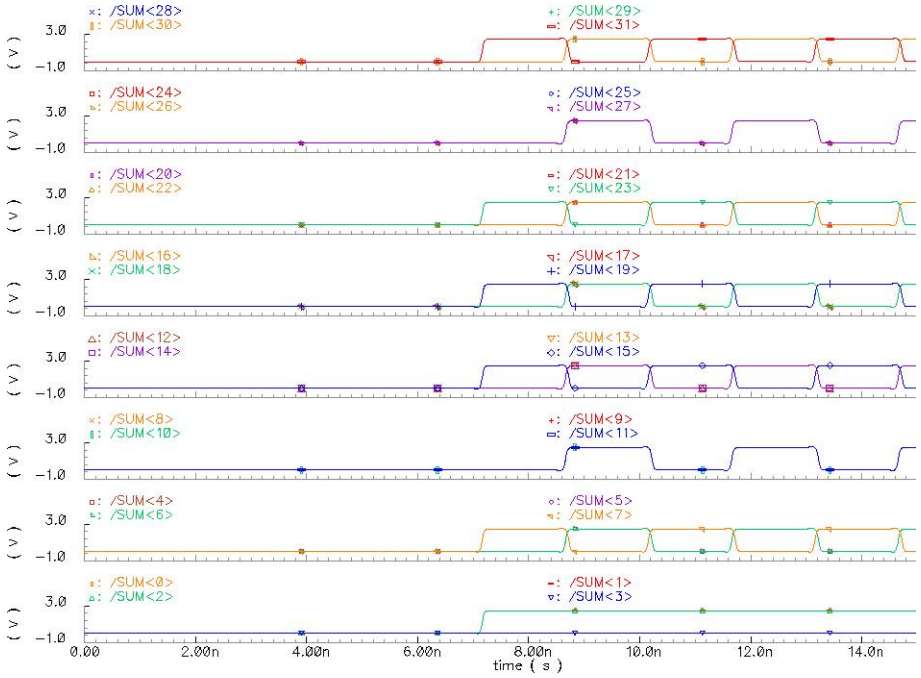

Figure 6.16: Simulation Results of Conventional Pipelined Adder

6.7 Comparison

In this section we will compare the results presented above for the Wave-Pipelined, Hybrid Wave-Pipelined and where applicable conventional pipelined adders. Simulations were done using Cadence Spectre Schematic tools on the three pipelined adders (Conventional, Wave and Hybrid-Wave). The results of these simulations are shown in table 6.3. The conventional pipelined system was implemented by clocking the Hybrid Wave-pipelined adder with no clock delay. We note here that the performance of the conventional pipelined adder could be improved by adding additional registers creating a deeper pipe. This would increase the number of registers needed as well as add additional complexity and loading to the clock signal not to mention an increase in power. Speculative performance results of a deep conventional adder pipeline are shown in Table 6.3. It is assumed here that the SUM block (XOR logic gate) will present the bottle-neck and thus the worst case delay. The results show that the Wave-Pipelined Adder and Hybrid Wave-Pipelined Adder systems perform far better than the conventional pipelined adder. The results for both the wave-pipelined system and the conventional pipeline system where extracted using perfect clocks with no clock distribution network. The Hybrid Wave-Pipelined adder uses a perfect clock coming into the system and then is distributed to all register accordingly. If a clock network where to be applied to the other two systems it is expected that their performance would decrease.

Table 6.3: Adder Clock Cycle Times

Design	**Max Speed**	**Frequency**
Hybrid Wave-Pipeline	560ps	1.78 GHz
Wave-Pipeline	670ps	1.49 GHz
Conventional Pipeline	1.5ns	666 MHz
Deep Conventional Pipeline	800 ps	1.25 GHz

The number of waves that the Wave-Pipelined Adder and Hybrid Wave-Pipelined adder can sustain is shown in Table 6.4. Conventional pipelining cannot sustain multiple waves of data

between a set of two registers. It should also be noted that the Hybrid-Wave Pipelined system has three stages where wave-pipelining only has one. The number of waves sustainable per stage of the Hybrid-Wave pipelined adder are also included in the table.

Table 6.4: Number of Sustainable Waves Per Stage

Design	**Stage1**	**Stage2**	**Stage3**	**TOTAL**
Hybrid Wave-Pipeline	3	2	3	8
Wave-Pipeline	3	-	-	3
Conventional Pipeline	1	1	1	3

Another advantage to using Hybrid Wave-Pipelining is that the clocks at the input and output register need not be synchronized. This is in contrast to the other two approaches which expect the same clock edge with minimal skew at both the input and output registers. Figure 6.17 illustrates how the input clock CLK edge is not synchronized with the output clock edge $clk4c$. This clocking scheme also makes it very applicable to clock gating. If the clock is shut off the clock will propagate to the output with the data. There is no need to continue clocking to flush the pipeline as in conventional pipelining and wave-pipelining. In this figure the clock initially stays low because it has not yet propagated to the final output with data. This is effectively the latency of the system.

The throughputs of the systems are reported in table 6.5. The throughput of a conventional pipelined system is comparable to the number of stages in the pipeline. In order to compare this to Wave-Pipelining and Hybrid Wave-Pipelining this is compared to the number of sustainable waves within these systems. Using this metric is misleading however, because the clock frequency for both the Wave-Pipelined and Hybrid Wave-Pipelined adder is much greater than the conventional pipelined adder. The Wave-Pipelined and conventional pipelined system both can sustain three separate instructions within the system but the Wave-Pipelined systems clock frequency runs 124% faster than that of the conventional pipelined adder.

Power is always a concern when designing digital circuits. The adders presented above have

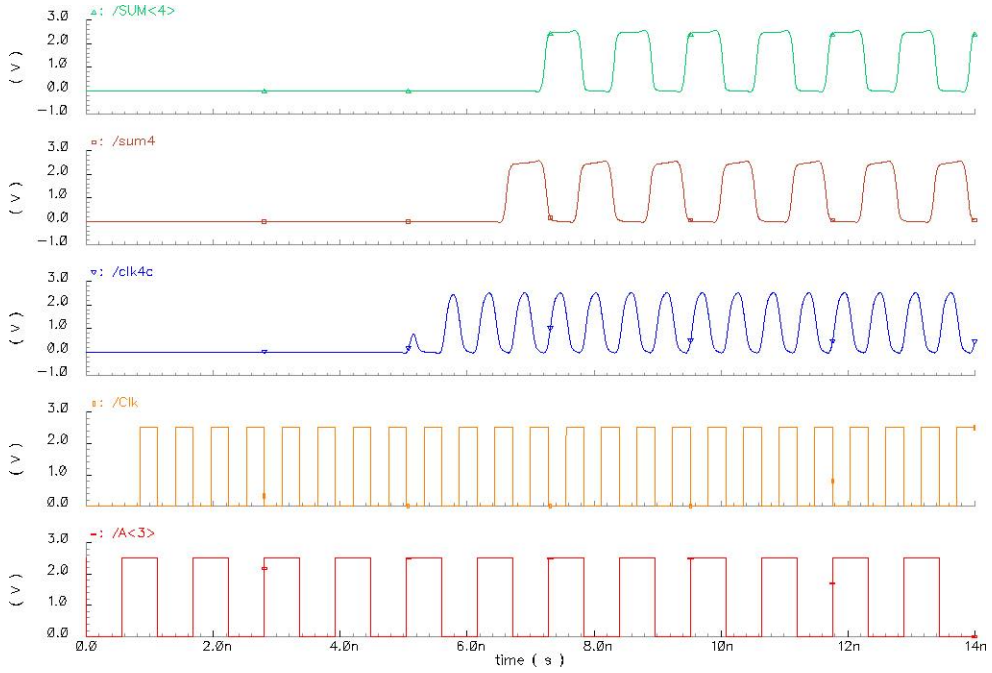

Figure 6.17: Illustration of the lack of synchronization between Input and Output Clocks.

Table 6.5: Throughput of Pipelined Systems

Design	**Throughput**
Hybrid Wave-Pipeline	8
Wave-Pipeline	3
Conventional Pipeline	3

been optimized for speed rather than power. Research and possible solutions regarding issues dealing with power dissipation have been noted and are currently being looked into. In Table 6.6 the average power dissipation of the three pipelined adders is presented. The average power was measured at the maximum operating frequency of each adder as well as at a common operating frequency. It should be noted that both the wave-pipelined and conventional adders do not have associated clock trees, the power dissipation would increase if these circuits where added. Most of the power consumption is attributed to the use of the biased NAND gates in the carry block of the adders.

Table 6.6: Average Power Consumption

	Average Power (W)		
Design	**@ 666 MHz**	**@ Max Speed**	**Max Speed**
HWP	1.348	1.76	1.78 GHz
WP	.8384	0.902	1.49 GHz
CP	.943	.943	666 MHz

6.8 Conclusion

The adder is a very important and extremely complex circuit used in digital design. As shown many implementations and variants exist. In this chapter we have outlined a basic understanding of adder circuits as they pertain to the implementations presented in this thesis. An understanding, implementation as well as comparison between conventional, wave, and hybrid wave-pipelined adders has been given. In contrast to these parallel adder designs, a hybrid ripple-carry adder was also presented and evaluated.

CHAPTER 7

Research Contributions

7.1 Introduction

This chapter will summarize the research contributions, future work and exploration regarding Hybrid Wave-Pipelining and finish with a section of concluding remarks. In working with Hybrid Wave-Pipelining many discoveries where made relating to its limitations. These results as well as the positive contributions will be highlighted in Section 7.2. Section 7.3 will present the contributions and uses of the hybrid RCA/CLA adder presented in 6.3. Section 7.4 will outline the future work and research that should be completed and section 7.5 will end in a summary of our findings.

7.2 Failed Approaches and Contributions to Hybrid Wave Pipelining

Hybrid Wave-Pipelining increases the performance of a pipelined system. It achieves this by reducing the clock cycle time and increasing the number of sustainable waves within the pipeline. The scheme can be implemented by semi-custom design methods once a standard library of gates has been developed that decrease the issues associated with data dependencies. Hybrid Wave-Pipelining delays the clock along with the data making this approach very applicable to clock-gating. If the clock is turned OFF there is no need to worry about flushing the pipeline because subsequent clock signal will travel with the data until it reaches the output.

In implementing the Hybrid Wave-Pipelined adder biased-NAND gates where used because of their tolerance to data dependencies. When trying to use these same devices in the clock signal path to match delays serious problems where encountered. The functionality of the biased-NAND results in a weak logic 0. The lack of drive strength that is associated with this "weak" signal is not sufficient to drive the output latches, this was only amplified at higher speeds where the signal would degrade further due to decreased discharge time. This could be improved by applying a bias voltage to the gate of the p-mos device. The bias voltage should be such that it turns the device OFF for any input condition that results in a path to ground.

When implementing the internal latches of the Hybrid-Wave Pipelined adder many different approaches where taken. The first approach was to use edge triggered DFF's as internal registers, the drawback to this first approach was that the flip-flops required both CLK and $\overline{CLK}$. The skew between these two clock signals was enough to significantly reduce the operation of the adder. Initially it was thought that the capacitance that each signal was driving was to blame. To reduce the capacitance the clock signal had to drive transmission gate latches where used. This move created even larger problems, the clock signals where still skewed but not as significantly. However, the clocks themselves could not be matched to the data at high frequencies when the clock ceases to look as a square wave and essential becomes sinusoidal. Because of this data at the internal latches was not transfered correctly. Data was consistently missed because the time the clock signal was active enabling the transmission gate to pass data was not significant enough to drive the loads at the output. This was only amplified when the effects of the clock skew between CLK and $\overline{CLK}$ where considered. It was found that using edge-triggered devices that required only the CLK signal provided the best results and thus where used. The clock signal load was offset by the use of matched RC Trees to distribute the load.

The current implementation of the "padding" elements within the carry block is to connect one input of a biased NAND gate to V_{dd} thus creating an inverter. These biased NAND gates use five transistors in their implementation and it was thought initially that this could be reduced. An

attempt was made to match the worse case delay of the biased NAND gate using only inverters or modified inverters which would reduce the number of gates used. The modified inverter uses two n-mos devices in series, one tied to v_{dd}, to match the delay of the biased NAND gate. However, it was found that because of the unbalanced loads seen between the two gates this approach did not perform as well as using the original biased NAND gate design.

7.3 Hybrid CLA

The Hybrid Carry Lookahead adder presented in section 6.3 runs at a much slower operating frequency than the parallel prefix adders studied in this thesis. However, as technologies scale below 35nm it has been seen that the delay is no longer associated with the switching of devices but occurs in the wires [2]. New models must include the inductance of the lines which until this point has been greatly ignored [37]. In light of this it would be desirable to design circuits with very short interconnecting wires. Parallel adders unfortunately have very long wire lengths [21], in order to move data to appropriate nodes as shown in Figure 7.1. Power supply voltages will go below 1 volt making it difficult to drive the large loads of parallel adders.

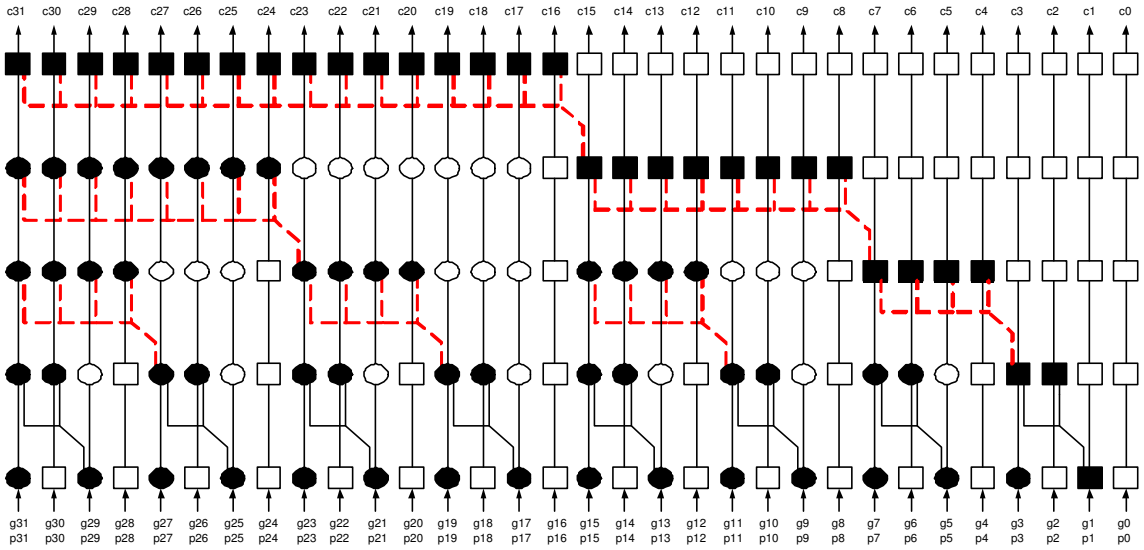

Figure 7.1: Long wire routes of Parallel Adder.

Ripple carry adders and our hybrid CLA scheme have very short wire interconnects as the output of one stage is directly adjacent to the input of the next stage as shown in Figure 7.2. Depending on the length of wire the parallel adder must drive it is foreseeable that at some point the hybrid CLA delay may become comparable to that of the parallel design. This occurring at a much lower cost in area, extrapolated from this reduction of area it is foreseeable as well that a great reduction in power consumption may be possible.

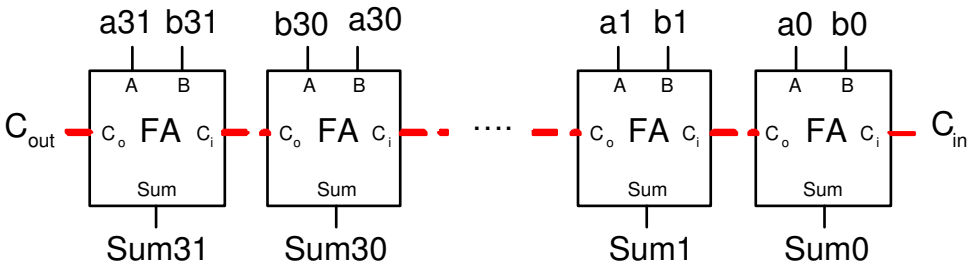

Figure 7.2: Short Wire routes of RCA.

7.4 Future Work

7.4.1 Introduction

There is still much work to be done in regards to Hybrid Wave-Pipelining and its association with adder designs. The following subsections outline the possibilities for future exploration.

7.4.2 Power Dissipation Due to Clock Network

One aspect of particular interest would be to evaluate the power dissipation associated with the clock distribution network. It would be valuable to identify the difference in power consumption between a Hybrid Wave-Pipelined network versus the power consumed by a traditional clock network driving a conventional pipeline.

7.4.3 Limited Fan-Out

Experiments involving a limited fan-out should be conducted to see the trade-offs between design time, area, and performance. Limiting the fan-out will increase the hardware but would reduce the

design complexity by decreasing the need to balance multiple loads. There would be an increase in area associated with this but it may be outweighed by the performance gains.

7.4.4 Algorithm for Optimal Insertion of Internal Registers

Developing an algorithm for optimal insertion of internal registers would be of great use to this design. The places that internal registers where inserted in this design was not done by any formal process. To develop an actual model for the optimal insertion point should lead to an increase in performance as well as decrease in design time and effort.

7.4.5 Internal Register Implementation

Finally other DFF implementations should be explored. In the work presented the overhead and unbalanced nature of the DFF was a major limitation. A DFF with low overhead and very little variation due to data dependencies, i.e. transitions between logic 0 and 1, would greatly increase the performance of any Hybrid Wave-Pipelined system.

7.5 Conclusion

Hybrid Wave-Pipelining provides many advantages as shown in this thesis. However, there is still extensive work to be done on the subject. As technologies scale and the need for increased performance continues to dominate alternatives to the traditional digital design techniques currently in use will arise. Hybrid Wave-Pipelining may be one of these new approaches, as shown in this chapter there are still many hurdles to overcome.

CHAPTER 8

Concluding Remarks

This thesis has provided the background information as well as reasons for implementing digital systems using Hybrid Wave-Pipelining. It has provided a comparison of conventional pipelining, Wave-Pipelining and Hybrid Wave-Pipelining by implementing all three as applied to a parallel prefix adder. The necessary conditions and equations for these adders have also been presented and explored. The work shows that the critical design issues when dealing with Hybrid Wave-Pipelining include; balancing delay paths, developing an appropriate clock delay scheme and reducing the overhead and latency associated with intermediate latches.

The Hybrid Wave-Pipelined adder performs 167% better than the conventional pipelined adder and has a speed up of 19.6% over the Wave-Pipelined adder. The number of sustainable waves the Hybrid Wave-Pipelined adder can sustain is eight as compared to three for Wave-Pipelining.

The use of Hybrid Wave-Pipelining can significantly increase the speed of a digital system. It is well suited for regular structures where the delay paths can be easily balanced. Hybrid Wave-Pipelining increases the design time and complexity compared to conventional pipelined systems but this increase is offset by the resulting benefits.

The hybrid RCA/CLA performs 22-67% better than a standard RCA with only a 1.5% increase in power. The size of the hybrid RCA/CLA is 15% smaller per carry look-ahead block used in

terms of transistor count. As technolgies scale large wire interconnects will dominate the critical delay path of digital circuits. The hybrid RCA/CLA with prediction presented in this thesis may be a solution to elliminate such problems.

REFERENCES

[1] K. Banerjee and A. Mehrotra, "A Power-Optimal Repeater Insertion Methodology for Global Interconnects in Nanometer Designs," *IEEE Transactions on Electron Devices,* Vol. 49, Issue 11, November 2002, pp. 2001-2007.

[2] L. Benini and G. De Micheli, "Networks on Chips: A New SOC Paradigm," *IEEE Computers,* Vol. 35, Issue 1, January 2002. pp. 70-78.

[3] R. P. Brent and H.T. Kung, "A regular Layout for Parallel Adders," *IEEE Transactions on Computers,* C-31, March 1982, pp. 260-264.

[4] A. Burg, F. K. Gürkaynak, H. Kaeslin and W. Fichtner, "Variable Delay Ripple Carry Adder With Carry Chain Interrupt Detection," *IEEE International Symposium on Circuits and Systems,* Vol. 5, 25-29 May, 2003. pp. V-112 - V-116.

[5] W. P. Burleson, M. Ciesielski, F. Klass and W. Liu, "Wave-Pipelining: A Tutorial and Research Survey," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems,* Vol. 6, No. 3, pp. 464-474, September 1998.

[6] P. K. Chan, M. D. F. Schlag, C. D. Thomborson, and V. G. Oklobdzija, *"Delay Optimization of Carry-Skip Adders and Block Carry-Lookahead Adders,"* 10th IEEE Proceedings on Computer Arithmetic, June 26-28, 1991, pp.154164.

[7] K-H. Cheng; W-S. Lee and Y-C. Huang, "A 1.2V 500 MHz 32-bit Carry-Lookahead Adder," *8th IEEE International Conference on Electronics, Circuits and Systems,* Vol. 2, September 2-5, 2001, pp. 765768.

[8] E. Y. Chou, J. C. Huang, M. C. Hsieh and A. Y. Hsu, "Baud-Rate Channel Equalization in Nanometer Technologies," *IEEE Transactions on Very Large Scale (VLSI) Systems,* Vol. 12, No. 11, November 2004, pp. 1174-1181.

[9] P. Corsonello, S. Perri and G. Cocorullo, "Hybrid carry-select statistical carry lookahead adder," *Electronics Letters,* Vol. 35, Issue 7, April 1, 1999, pp. 549-551

[10] L. Dadda and V. Piuri, "Pipelined Adders," *IEEE Transactions on Computers,* Vol. 45. NO. 3, March 1996, pp. 348-356.

[11] J. G. Delgado and J. Nyathi "A Wave-Pipelined Network Router," *IEEE Computer Society Workshop on VLSI 2001,* April 19-20, 2001, pp. 165-170.

[12] V. V. Deodhar and J. A. Davis, "Voltage Scaling and Repeater Insertion for High-Throughput Low-Power Interconnects," *Proceedings of the 2003 International Symposium on Circuits and Systems,* Vol 5, May 25-28, 2003, pp. 349-352.

[13] A.G. Dickinson and C. J. Nicol, "A Systolic Architecture for High Speed Pipelined Memories," *IEEE International Conference on Computer Design: VLSI in Computers and Processors,* October 3-6, 1993, pp. 406-409.

[14] H. Eriksson, P. Larsson-Edefors, and A. Alvandpour, "A 2.8 ns 30 uW/MHz area-efficient 32-b Manchester carry-bypass adder," *IEEE International Symposium on Circuits and Systems,* Vol. 4, May 6-9, 2001, pp. 84-87.

[15] C. Fang, C. Huang, J. Wang and C. Yeh, "Fast and Compact Dynamic Ripple Carry Adder Design," *IEEE Asia-Pasific Conference on ASIC,* 6-8 Aug, 2002. pp. 25-28.

[16] C. T. Gray, W. Liu, R. K. Cavin, III, *Wave Pipelining: Theory and CMOS Implementation,* Kluwer Academic Publisher 1994.

[17] C. T. Gray, W. Liu, and R. K. Cavin, III, "Timing Constraints for Wave-Pipelined Systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* Vol. 13, No. 8, pp. 987-1004, August 1994.

[18] T. Han and D. A. Carlson, "Fast Area-Efficient VLSI Adders," 8^{th} *Symposium on Computer Arithmetic,* May 1987.

[19] J. L. Hennessy and D. A. Patterson, *Computer Architecture A Quantitative Approach, Second Edition,* Morgan Kaufmann 1995.

[20] C-H. Huang, J-S. Wang, C. Yeh and C-J. Fang, "The CMOS Carry-Forward Adders," *IEEE Journal of Solid-State Circuits,* Vol. 39, NO. 2, February 2004, pp. 327-336.

[21] Z. Huang and M. D. Escegovac, "Effect of Wire Delay on the Design of Prefix Adders in Deep-Submicron Technology," *Conference Record on the Thirty-Fourth Asilomar Conference on Signals, Systems and Computers,* Vol. 2, 29 October - 1 November, 2000. pp. 1713 - 1717.

[22] Ivanov, Lubomir, "Modeling and Verification of a Pipelined CPU," *The 2002* 45^{th} *Midwest Symposium on Circuits and Systems,"* Vol. 3, August 4-7, pp. III-417 - III-420.

[23] D. A. Joy and M. J. Ciesielski, "Clock Period Minimization with Wave Pipelining," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* Vol. 12 No. 4, pp. 461-462, April 1993.

[24] Y. Kim and L-S Kim, "64-bit carry-select adder with reduced area," *Electronics Letters,* Vol. 37, Issue 10, May 10, 2001, pp. 614615.

[25] F. Klass, M. J. Flynn and Ad J. van de Goor, "A 16x16-bit Static CMOS Wave-Pipelined Multiplier," *IEEE International Symposium on Circuits and Systems,* Vol. 4, 30 May - 2 June, 1994. pp. 143-146.

[26] U. Ko and P. T. Balsara, "High-Performance Energy-Efficient D-Flip-Flop Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* Vol. 8, No. 1, February 2000, pp. 94-98.

[27] P. Kogge, H. Stone. "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions Computers,* C-22(8), August 1973, pp. 786-793.

[28] J. B. Kuo, H. J. Liao, and H. P. Chen, "A BiCMOS Dynamic Carry Lookahead Adder Circuit for VLSI Implementation of High-Speed Arithmetic Unit," $IEEE\ Journal\ of\ Solid-State\ Circuits$, Vol. 28, No. 3, pp. 375-378, March 1993.

[29] W. K. C. Lam, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Valid Clock Frequencies and Their Computation in Wave-pipelined Circuits," *IEEE Trans. on Computer-Aided Design,* vol. 15, no. 7, pp. 791-807, July 1996.

[30] 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-um CMOS," *IEEE Journal of Solid-State Circuits,* Vol. 29, NO. 9, September 1994, pp. 1117-1128.:

[31] W. Luk, "Regular Pipelined Multipliers," *Electronic Letters,* Vol. 25, Issue 20, September 28, 1989, pp. 1405-1407.

[32] T. Lynch and E. E. Swartzlander, "A Spanning Tree Carry Lookahead Adder," $IEEE\ Transactions\ on\ Computers$, Vol. 41, No. 8, pp. 931-939, August 1992.

[33] A. Nalamalpu and W. Burleson, "A Practical Approach to DSM Repeater Insertion: Satisfying Delay Constraints While Minimizing Area and Power," *Proceeding of the* 14^{th} *Annual IEEE International Conference on ASIC/SoC,* September 12-15, 2001, pp. 152-156.

[34] J. Nyathi and J. G. Delgado-Frias, "Hybrid Wave-Pipelined Network Router," *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications,* December 2002, pp. 1764 - 1772.

[35] J. Nyathi, J. G. Delgado and J. Lowe, "A High Performance, Hybrid Wave-pipelined Linear Feedback Shift Register with Skew Tolerant Clocks" *46th IEEE International Midwest Symposium On Circuits and Systems,* Cairo, Egypt, December 27-30, 2003.

[36] J. M. Rabaey, A. Chandrakasan and B. Nikolić, *Digital Integrated Circuits: A Design Perspective,* 2^{nd} *ed.,* Pearson Education, Inc., NJ. 2003.

[37] S. Simon Wong and others, "On-Chip Interconnect Inductance - Friend or Foe," *Fourth International Symposium on Quality Electronic Design, 2003. Proceedings,* 24 - 26 March, 2003, pp. 389-394.

[38] Sukumar, Vinesh., and others, "Design of A Pipelined Adder Using Skew Tolerant Domino Logic in A 0.35um TSMC Process." *IEEE Workshop on Microelectronics and Electron Devices,* 2004, pp. 55-59.

[39] N. West and K. Eshraghian, *Principles of CMOS VLSI Design: A System Perspective, 2nd ed.,* Addison Wesley, NY. 1992, pp. 513-536.

[40] D. C. Wong, G. De Micheli, and M. J. Flynn, "Designing High-Performance Digital Circuits Using Wave-Pipelining: Algorithms and Practical Experiences." *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* Vol. 12, No. 1, pp. 25-48, January 1993.