# A GASP OF FRESH AIR:

# A HIGH SPEED DISTRIBUTED FIFO SCHEME

# FOR MANAGING INTERCONNECT PARASITICS

By

RAY ROBERT RYDBERG, III

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 RAY ROBERT RYDBERG, III find it satisfactory and recommend that it be accepted.

Chair

## ACKNOWLEDGEMENT

I would like to acknowledge my advisor, Dr. Jabulani Nyathi, for his steadfast advice and long hours spent assisting me, when a thousand other projects needed his attention.

I would also like to acknowledge my roommates, without whom I would not have eaten nearly as well as I have during the writing process.

# A GASP OF FRESH AIR:
# A HIGH SPEED DISTRIBUTED FIFO SCHEME
# FOR MANAGING INTERCONNECT PARASITICS

## Abstract

by Ray Robert Rydberg, III, M.S.
Washington State University
May 2005

Chair: Jabulani Nyathi

Interconnect delays are increasingly becoming the dominant source of performance degradation in the nano-meter regime, largely because the wires do not scale as well as the transistors. Scaling trends allow for complete systems to be built on a single chip (SoC), but they require long interconnects for global signals and clock distribution networks. The parasitics of these global interconnects make efficient and high-performance operation difficult. On-chip communication shortly will require tens of clock cycles for signal propagation between communicating modules/components. This communication is becoming less reliable as feature size and power supply voltages decrease, thus increasing the effect of environmental and process variations. Currently, repeater insertion is widely used to improve global interconnect delays, but with very high latency. In this thesis, an asynchronous distributed first-in, first-out (FIFO) buffer is proposed to facilitate communication between modules of highly integrated SoCs. A synchronous FIFO is built for comparison. Using an asynchronous FIFO helps alleviate the clock skew, clock distribution and single clock synchronization problems associated with high-speed, synchronous digital design. The distributed FIFO buffer scheme also improves latency considerably. In addition, this asynchronous FIFO scheme has very good tolerance to voltage and temperature variations. The buffer control circuitry is self-timed and allows for ease of interfacing in multiple domain clock designs. The asynchronous FIFO allows a maximum data transfer rate of 1.67 GHz and 2.35 GHz in a $0.25\mu m$ and $0.18\mu m$ technology respectively.

# TABLE OF CONTENTS

# LIST OF TABLES

Page

# LIST OF FIGURES

## Dedication

This thesis is dedicated to my grandfathers, both of whom have passed beyond this world.

They have been my inspiration both professionally and personally.

# CHAPTER ONE

# INTRODUCTION

Global on chip interconnects are increasingly becoming a limiting performance factor in highly integrated systems such as system-on-chip (SoC). Some of the reasons leading to global interconnects being a limiting factor in system performance include: wires scaling less than transistors [7], power supply drop variations, process variations, single clock synchronization [7], large wires with unpredictable delays [36], and interconnect power dissipation [6]. These limitations also imply that it would be difficult to achieve correct functional and reliable operations while maintaining low energy consumption within the interacting modules or components of SoC [6]. Proposed solutions to alleviate these limitations include: separating the computation problem from the communication problem and introducing networks on chips [6], having communicating components operate asynchronously, while the computational blocks operate synchronously based on locally generated clocks [17] and optimizing repeater insertion for global interconnect [11], [28].

Our work is based on the premise that global wires that span a significant fraction of the chip will impose signal delays that exceed the clock period [7]. Additionally, synchronizing operations on components running at different clock speeds is becoming more difficult due to clock skew and distribution [6]. With wire delays that exceed the clock period, it becomes apparent that the interacting components in a SoC design will perform computations much faster than the results could be transferred between components (a process that now requires multiple clock cycles). This assertion can further be substantiated by the fact that transistor switching speeds are much faster than wire delays. It is stated that as the transistor switching speeds improve, the wire delays increase. A copper, low-k 35 nm process technology is reported to have device switching speeds of 2.5 ps, while interconnects of 1 mm in the same technology have delays of 250 ps (two orders of magnitude slower) [7].

A distributed first-in, first-out (FIFO) scheme is presented that enables communication between modules of a SoC with minimal influence of parasitic wire delay effects. Chapter 2 discusses the expected and predicted trends that occur as semiconductor technology scales in size, particularly into the nanometer regime. Chapter 3 highlights several architectural approaches to improve the performance in terms of throughput and power consumption. Chapter 4 discusses synchronous and asynchronous clocking schemes and distributed FIFOs. In Chapter 5, the performance of the asynchronous distributed FIFO. Chapter 6 mentions future directions of this work and concludes this thesis.

# CHAPTER TWO

# BACKGROUND ON SCALING TRENDS

Moore's Law (or Moore's observation) is commonly cited by people in reference to performance increases due to the scaling of technology. The observation, made in 1965 by Gordon Moore [24], noted that the number of transistors per square inch on integrated circuits had doubled every year since the integrated circuit was invented. Moore predicted that this trend would continue for the foreseeable future. However, performance, as related by Moore's Law, referred to device switching, since initially the delay of the wires was insignificant in comparison to the gate delay of integrated circuits. More recently, this observation has been perceived as describing the performance of digital system, rather than raw device count. This performance increase is caused by a combination of factors. First and most predominantly, is the shrinking of the size of transistors. Reducing the gate capacitances and channel length allows for decreased transistor switching delay. Second, and more subtle, is the growing size of dies, increasing the total allowable area for transistors and wires.

Currently, the 2004 International Technology Roadmap Specification [16] is predicting several billion devices per die. While this allows for significant integration of modules on a single die including multiple processing cores on a single die ( [30] and [8]), communication between these modules becomes a problem.

The performance of transistors versus interconnects as technology scales is discussed in Section 2.1. Communication issues in SoC design are discussed in Section 2.2. Section 2.3 discusses the paradigm change from computationally-bound design to communication-bound design. Section 2.4 concludes this chapter with a brief discussion of interfacing modules.

## 2.1 Switching vs. Interconnect Delay

One problem with scaling technology is the differences between parasitic device capacitance and global interconnect capacitance. It is shown in [7] that device parasitics will scale significantly faster than wire capacitance, creating a significant gap in performance in nanometer architectures.

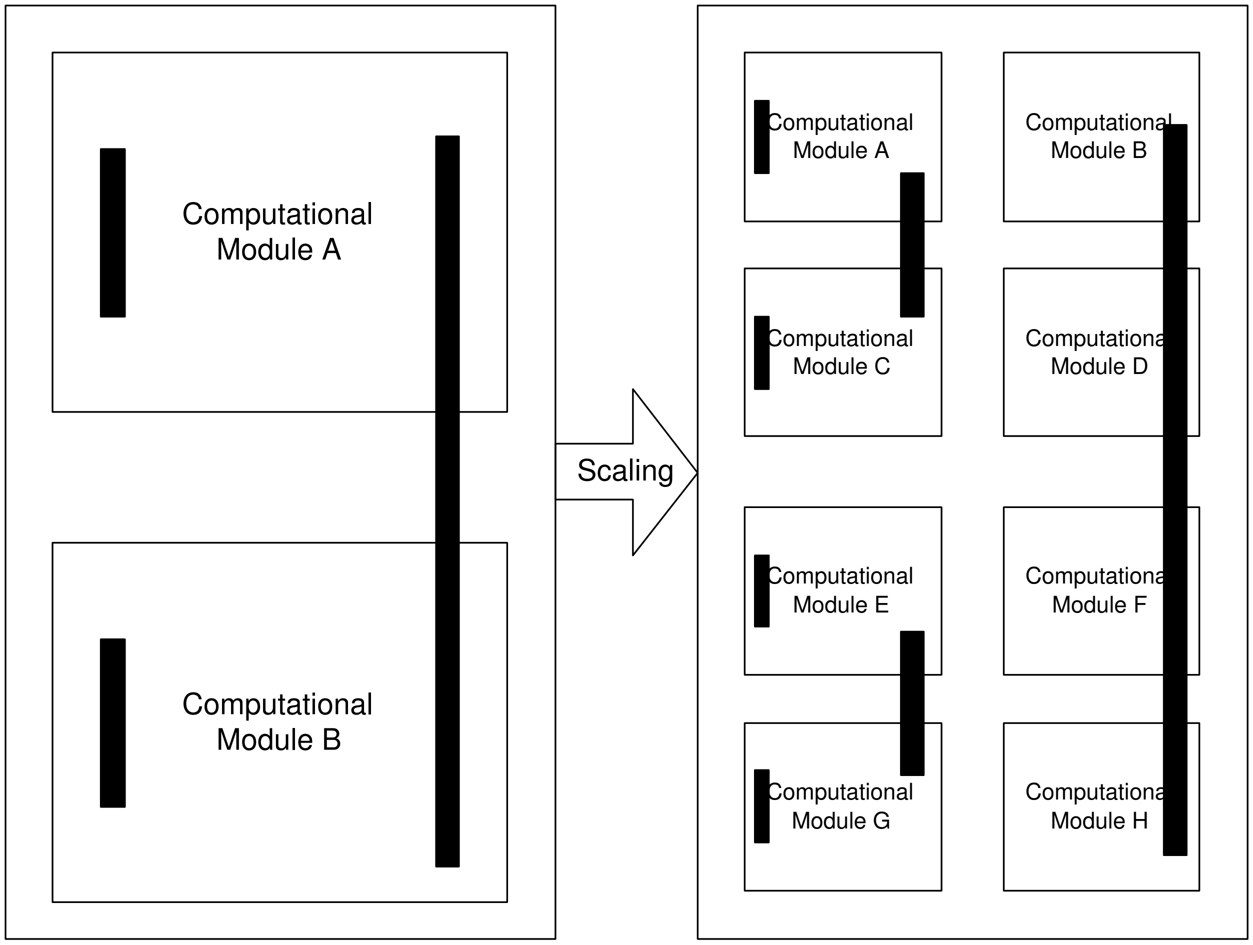


Figure 2.1: Demonstration of the Relative Effects of Scaling [7]

Figure 2.1 shows the difference in scaling between device size (parasitics) and interconnect size for two imaginary technology nodes. For the purposes of this figure, they are referred to as the larger or left node and the small or right node. On the left, the larger technology node allows for two computational modules on a chip. The chip using a smaller technology node contains four times as many modules in the same area. The local and adjacent interconnects shown on the larger node are comparable to the interconnects shown in the left column of modules of the smaller node. However, if module *B* requires communication with module *H*, the interconnect spans four computational modules. The length of the interconnect in the right column has not scaled with the transistor sizing.

However, most wires in a given digital design technology are short enough that the difference between device switching and the interconnect delay is negligible. These are wires that are used to connect transistors to create logic gates, and local routing inside logic blocks. Where difference becomes important is for medium to long interconnect wires; interconnect wires that connect beyond a modules immediate neighbors. These are wires that connect logic blocks inside a computational module, and most prominently, global interconnects that bridge computational modules in SoC design. It is possible that these longer wires, on a 1 $mm^2$ chip, could approach 10 cm in length, as the wire snakes through the chip to reach all of the modules that require the global signal. As it becomes possible to add several billion devices per chip [16], the distance between modules remains constant, while the size, and thus driving capabilities, of devices inside the modules continues to shrink.

Table 2.1: Wire Delay Comparison considering Scaling Factors

| | | Trans. Switching | | | Capacitance | | Delay per mm |
|---|---|---|---|---|---|---|---|
| Tech. Node | SF | Delay | SF | Wire Area | Value | SF | Time |
| 0.35 $\mu$m | 1 | 82.19 ps | 1 | $4\lambda \times 10,000\lambda$ | 27.2078 pF | 1 | 486 ns |
| 0.25 $\mu$m | 0.71 | 68.37 ps | 0.83 | $4\lambda \times 10,000\lambda$ | 21.4504 pF | 0.788 | 400 ns |
| 0.18 $\mu$m | 0.72 | 36.85 ps | 0.54 | $4\lambda \times 10,000\lambda$ | 9.5320 pF | 0.444 | 222 ns |

Table 2.1 shows an attempt to analyze wire delays at different technology nodes. However, the CAD tools that were available could not support the analysis. Table 2.1 was intended to show that transistor parasitics scale faster between technology nodes than wire parasitics. It seems unlikely

that the $21\%$ reduction in capacitance between $0.35\mu m$ and $0.25\mu m$ nodes resulted in a $41\%$ reduction in delay. The capacitance scaling factor between $0.35\mu m$ and $0.25\mu m$ seems reasonable, at a scaling factor of 0.788, given discussion in [31] and [16]. The switching speed of the devices was measured using a ring oscillator shown in Figure 2.2. For roughly balanced performance, $w_p = 2.5w_n$ with $w_n$ equal to the minimum device size for the given technology. The device switching speed was measured as the time it took a signal to travel through the ring twice (once high and once low) and then devided by the number of inverters the signal passed through (six). This ring behavior anaylsis provides a fair comparison between the technology nodes when the devices are not perfectly sized for balanced performance.

For comparison, the system that is explored in this thesis (in greater detail in Chapters 4 and 5) was designed in both $0.25\mu m$ and $0.18\mu m$ technology nodes. The throughput rate reduced from 600 ps to 425 ps, a scaling factor of 0.708. This scaling factor is noticeably higher than either the transistor switching delay or the wire capacitance, but equivalent to the scaling factor between technology nodes predicted by [16].

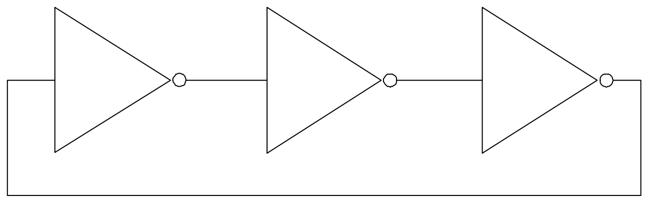


Figure 2.2: A Ring Oscillator

## 2.2 SoC Inter-Component Communication Issues

Global interconnects are becoming the dominant performance inhibitor in SoC designs [2]. Since more modules can be integrated onto a die, the number of global interconnects necessary to facilitate inter-module communication, as well as the relative length of these global interconnects, increases. This increase in the number of global interconnects contributes to the total increase of system power consumption in the following ways: increased total chip capacitance, operating frequency and device leakage [2]. Clearly, increasing the number of global interconnects will increase the total on-chip capacitance and thus the energy consumed during signal propagation down said interconnects. Additionally, the higher the frequency at which these global interconnects are switched, the more frequently parasitic capacitance is charged and discharged, increasing power consumption. Previously, power dissipation has been associated largely with computational blocks. It is shown in [23], that clocking contributes to over half of the power consumed by a high-speed digital system (see Figure 2.3). However, scaling has led to interconnects being an additional source of power dissipation.

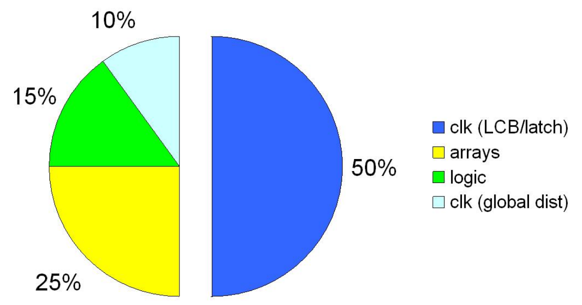


Figure 2.3: Power Consumption by Category in Modern Microprocessors

## 2.3 Separation of Computation and Communication

It is accepted that, in the near future, digital design will experience a paradigm shift from computationally bound design to communication bound design [7]. The on-chip latency of signal propagation is currently between 3 and 10 clock cycles in 0.18 $\mu$m technology and is expected to reach 10-30 clock cycles at the 50nm technology node. This latency is created by a combination of increasing clock frequency and an increase in die size. The clock frequencies are expected to reach

10 GHz for a 50nm process [16]. The die sizes have been increasing (to include even more devices than are added by scaling) as the manufacturing processes have improved, increasing yield.

These trends lend weight to a paradigm shift from computationally bound design to communication bound design. The research presented in this thesis was performed because of the increasing concern about problems with global interconnects [16]. If the communication and computation aspects of digital design are separated, then the most optimal solutions for each module or interconnect may be implemented without concern of the communication channel or the connected modules. This separation between computation and communication allows for power analysis to be performed on either a module or a communication channel. Since wires can no longer be considered equipotential, the performance of the computation modules can be optimized without being concerned with the delays now inherent with any communication scheme.

## 2.4 Interfacing Communicating Modules

With wire delays expected to dominate switching delays in smaller technology nodes, we examine methods that could best facilitate fast inter-module communication in SoC design. If the idea of separating computation from communication, presented in section 2.3, is adopted, then the next concern is the interface between modules for communication. By removing direct connections between modules, additional logic would become necessary to ensure that data is transmitted and received reliably and efficiently. This interfacing problem will only become more prevalent as SoCs integrate more modules.

Good engineering practices emphasize modular design and, subsequently, the reuse of the modules in future designs. This allows for a module to be designed and characterized once, increase the reliability and the time needed for a design. It is common for design companies to sell these modules as intellectual property (IP) cores for use by others. Each of these IP cores are designed to operate in a specific range of frequencies. With enough of these IP cores on a single die, the likelihood of these designs to not operate at equivalent speeds is high. This would create multiple clock domains (MCD) on a die. Communication inside of each of these clock domains is not a problem, because each module in a clock domain is operating at the same frequency and generally with very little clock skew. Communication between these clock domains is tricky due to the varying frequencies of the clock domains and commonly solved with buffers at the transmitting and receiving modules. This thesis proposes using an asynchronous communication scheme to glue the various IP cores together with minimal concern for the potentially different clock rates in MCD. This scheme significantly reduces the concern of both interconnect delays and MCD.

## 2.5 Conclusion

This section has established some of the problems associated with shrinking the size of technology nodes. In particular, the problem with interconnect delays has been established. Also, concerns about global communication were raised. Chapter 3 discusses several techniques available to digital designers to tackle the problems associated with global interconnect parasitics.

# CHAPTER THREE

# SYSTEM-ON-CHIP COMMUNICATION

Whenever more than one module exists on a chip, it is necessary to connect these modules together. Scaling technology allows for modules that used to exist on multiple chips on a PCB to exist inside a single die. Instead of each chip having a particular, circuit-level task, entire systems can reside on a chip. This is referred to as System-on-a-Chip (SoC) design. It becomes necessary to be concerned not only with the logic of the computational modules, but also how these modules are interconnected. This chapter discusses three architectural methods of on-chip communication in SoC design: buffer insertion, bus architecture, network-on-chip.

## 3.1 Buffer Insertion

Buffer, or repeater, insertion (Figure 3.1) is the most common method for dealing with on-chip communication. It is the simplest scheme for reducing the delay of wires due to parasitics. There has been significant development, in recent years, investigating the most optimal method for buffer insertion.

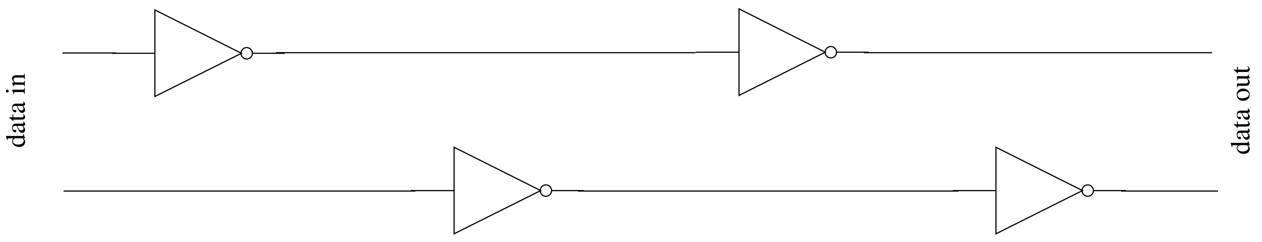


Figure 3.1: Buffer Insertion

Initially, designs focused solely on wire length, buffer size and signal propagation latency [28] and [2]. These devices were approaching 450 times greater than a minimum-sized inverter for optimized global interconnects. While this yielded a significant performance increase, the power consumption, among other problems with switching circuits, was not considered. A straightforward method for reducing power consumption is to lower the power supply rails; this method is commonly referred to as voltage scaling. This lowers the power consumption of the interconnect at the expense of increasing the propagation delay and lowering noise margins. It was found in [2] that delay is increased by $5\%$ and power consumption is reduced $10\%$ to $25\%$ for 180 nm and 50 nm technology nodes, respectively. Additionally, a paradigm shift from latency to throughput allowed [11] (and [12]) to report an operating frequency of 2 GHz using a scaled power supply voltage of 1V at 180 nm over 1 cm of wire using buffer insertion. This was accomplished by having multiple waves of data on the global interconnect. While this allows for much higher throughput, reliability becomes a concern.

In [19], the same concern of throughput and power is addressed, while also considering noise margins when optimizing buffer insertion. To improve delay beyond that of previous designs, low-swing signals were used in conjunction with a Schmitt trigger [33] at each buffer. This allowed for each buffer to read a rising transition at approximately $0.3V_{dd}$ and a falling transition at $0.7V_{dd}$, instead of the traditional transition at $0.5V_{dd}$. The clustering of buffers on a multi-bit data path creates several problems with regards to signal integrity. These problems are: power distribution on an interconnect, crosstalk and substrate noise. To address crosstalk and power distribution, [19] proposed staggering the buffers. This staggering breaks up long parallel wires to reduce crosstalk and distributes power consumption. Noise margins were improved by the use of differential signaling; a common trick in analog design to improve performance in noise sensitive designs.

Buffer insertion requires less design time and algorithms to optimize the process have been reported [2, 3, 11, 12, 19, 23, 28, 42]. The design is simple and the model of wire parasitics is well known. This makes the optimal placement of buffers on an interconnect to be a simple optimization problem in comparison to the optimization of bus or network-on-a-chip architectures. In terms of solving problems associated with global interconnects, buffer insertion is simplest and most ideal method for reducing wire parasitics to minimize the latency of global communication. However, buffer insertion does not provide memory to preserve data along an interconnect or allow for the connecting modules to operate at different frequencies. Also, buffer insertion is more susceptible to interconnect problems, such as: environmental and process variations, crosstalk, electromagnetic interference and radiation charge injection than other interconnect designs [7]. These problems cause the transmission to become unreliable and non-deterministic, requiring more elaborate schemes to resolve them.

## 3.2 Reduced Connectivity Architectures

Instead of a highly connected communication scheme where each module is directly connected with every module, by introducing a shared-channel communication architecture, it is possible to reduce the area required by the global communication scheme. Bus architectures and network-on-chip are two architectural design methods for improving the reliability of communication over increasingly problematic of global interconnects, while maintaining the high level of connectivity required by modules in a SoC.

### *3.2.1 Bus Architectures*

Bus architectures are borrowed from inter-chip communication schemes. They are commonly found at the system level for communicating between components that can afford to not be instantly connected to every device with which they need to possibly communicate. For example, microprocessors are commonly attached to several busses to communicate with main memory, hard drives, PCI cards and other external sources.

It is also necessary to optimize bus performance around several parameters to ensure the highest possible level of performance. These parameters include: the width of the signal wires, width of the shielding wires, wire spacing, gate (repeater) seize, length of the wire segment and the shielding period [23].

As bus architectures are currently in use for inter-chip communication, it is a reasonable logical step to consider busses for communication between modules in SoC design. Several bus architectures have been proposed to handle SoC communication. These architectures include: a static priority shared bus, a hierarchical bus, two-level time division multiplexing architecture (TDMA) and a ring network [20].

*Shared Bus*

A static priority shared bus, shown in Figure 3.2, is the simplest and most common on-chip bus architecture. This shared bus consists of a set of data and control lines shared by a set of master devices (M1 - M3) that prioritize access to all attached slave devices (S1 - S3). If more than one master exists, an arbiter is included to manage the possession of the bus between the master devices based on the assigned priority of each master. Burst-mode operation is possible with a static priority shared bus, enabling master devices to initiate transfer of multiple sets of data from a particular slave device without the overhead cost incurred for each set.

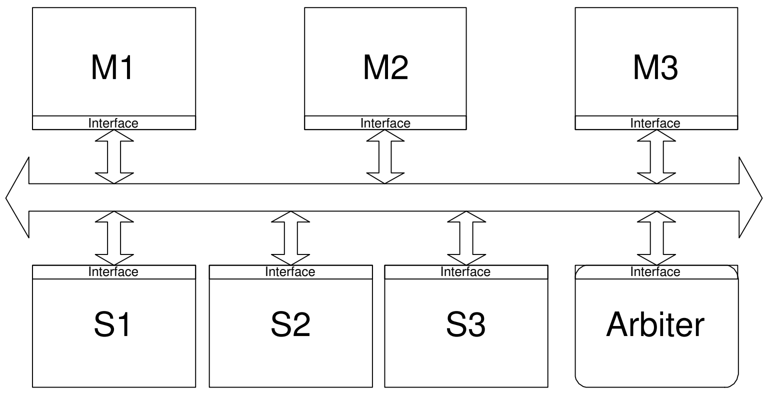


Figure 3.2: A Shared-Bus with 3 Masters, 3 Slaves and an Arbiter [20]

A shared bus architecture allows for a reduced number of global interconnects than buffer insertion. This reduction in the quantity of global interconnects, in combination with the additional hardware required for arbitration of, and connection to, the bus, could slightly reduce the power consumption of the design. However, a shared bus architecture greatly reduces the level of concurrency allowed in global communication; only two modules would be allowed to communicate at a time. A solution to this would be multiple shared busses, but this would further complicate computational modules by requiring those modules to determine which bus they are going to use for communication.

*Hierarchical Bus*

A hierarchical bus, depicted in Figure 3.4, is a split implementation of a static priority shared bus. Instead of one monolithic bus, that bus is split into *n* number of busses (Figure 3.3). Each of these busses independently act as a static priority shared bus. In addition to the normal set of master and slave devices on each of the shared busses, there is a master and slave connection to the other busses. This enables transmission to happen between busses. Also, since each bus is independent, each bus contains its own arbiter to handle bus communication. This design allows for greater concurrency among the individual busses as they can operate independently, until such a time as a module requests data that is from a module on a separate bus. The transmission between busses involves a significant amount of overhead compared to a standard transmission [20]. This design would be beneficial if modules predominantly require transmission to local modules and only occasional transmission to other busses.

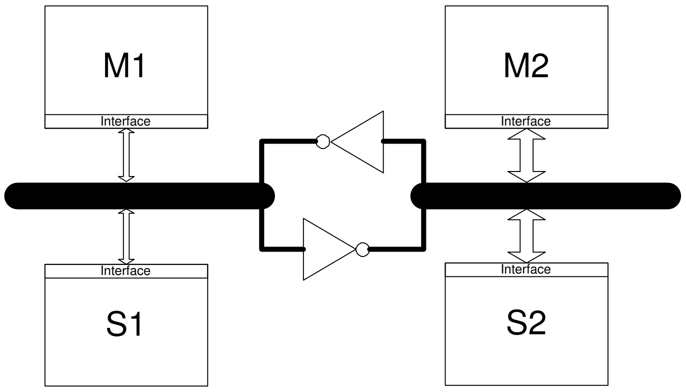


Figure 3.3: A Split-Bus Design [32]

A hierarchical bus architecture also reduces the number of global interconnects, and additionally allows for a greater level of concurrency amongst local modules, than a shared bus architecture.

However, in a hierarchical architecture, the penalty for communication across multiple busses is higher than that of a shared bus architecture. A hierarchical bus would be preferred over a shared bus if the associated SoC required predominantly local communication, rather than cross chip communication.

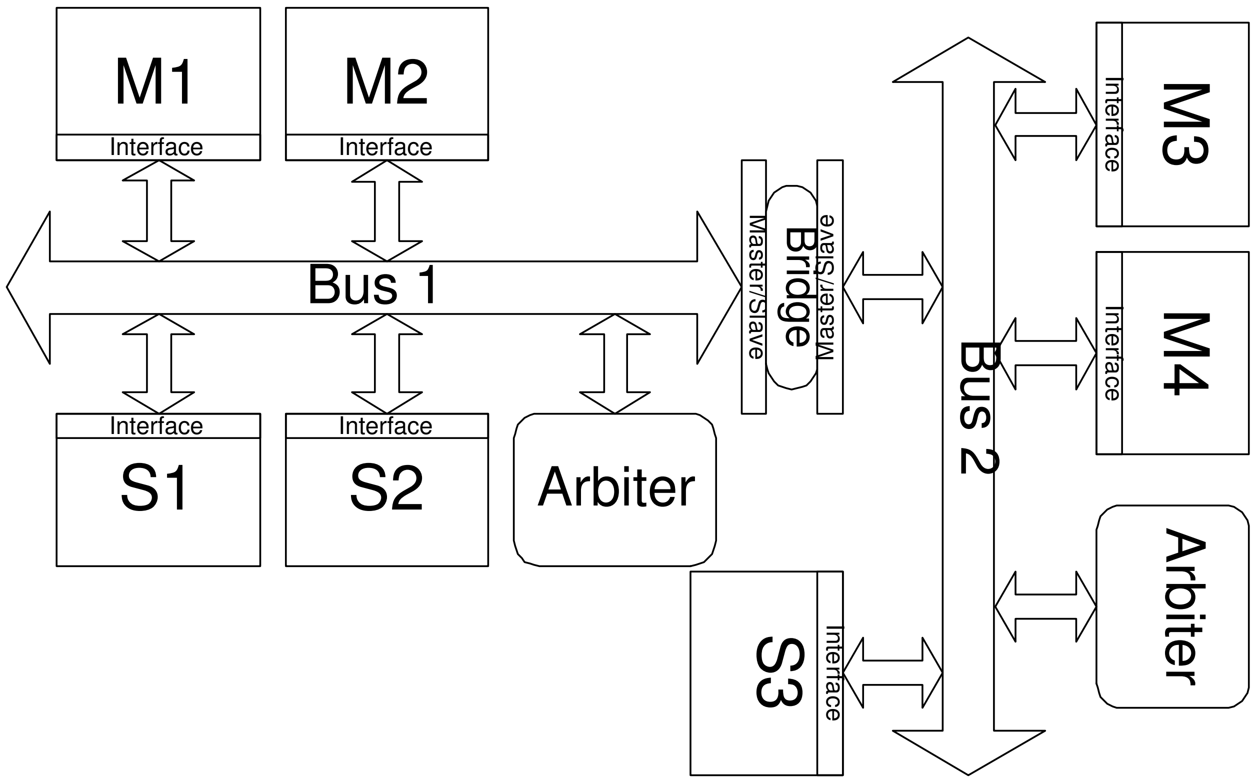


Figure 3.4: The Hierarchy Bus Architecture [20]

*Ring Bus*

A ring bus, shown in Figure 3.5, is similar to early designs in local area networks, specifically token ring topology [14]. The modules are connected in a unidirectional cyclic fashion. The modules only receive from one side and transmit to the other. Since each module only has one input source and one output destination, this allows the ring bus to operate at a significantly higher frequency than other bus designs. This model is considered a bus, rather than a network-on-chip, because all that is necessary for data transfer is the address of the destination; neither a network stack, nor other features of a network-on-chip are required for proper behavior. The major drawback is that for data to travel the opposite direction of the ring, the data must travel a majority of the ring before it arrives at its destination. As explained in [14], this architecture allows for a basic level of concurrency, since data can only travel in one direction and, at most, travels $N - 1$ nodes around a $N$ node ring.

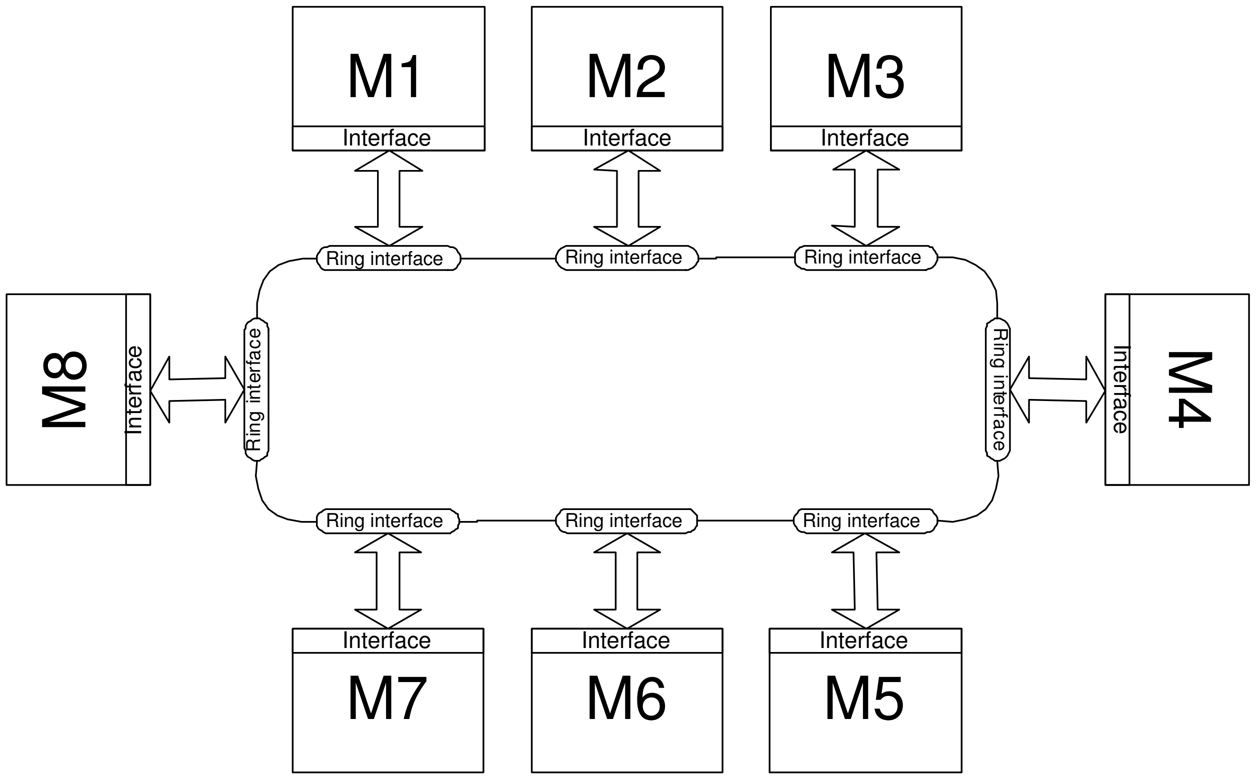


Figure 3.5: Ring Architecture

A ring bus eliminates cross-chip interconnect wires in favor of a locally connected ring. A module connected to a ring bus can perform one of three actions every clock cycle: i) read data and remove it from the ring, ii) propagate data to the next stage of the ring, and iii) generate data to be placed on the ring. If much data already exists on the ring, the intermediate ring nodes must be capable of delaying the transmission of data until an opening exists at their place in the ring. Since data travels in one direction round a ring bus, data that would normally be sent locally could be transmitted almost the entire length of the ring before arriving at a neighboring node.

### *3.2.2 Network on Chip*

An alternative to bus architectures for on-chip communication is to implement a full-blown network-on-chip. These networks-on-chip are a derivative of on- and off-board networks. On-board networks (like Philips IIC bus) can be implemented on-chip instead. Additionally, full 7-layer networks stacks have been considered for on-chip networks [32]. In addition to the traditional TCP\IP network stack, other protocols (e.g. USB [36]) have been used to implement a network on chip with success. It should be noted that the overhead involved in a 7-layer network is tremendous for on-chip communication. It is suggested in [32] that a 3-layer network stack is sufficient to handle any on-chip communication.

Network-on-chip provides the designer with several benefits. First, the communication network is robust. A 3-layer stack allows for error correction encoding to be used to ensure more reliable transmission of data over possibly error-prone global interconnects. Crosstalk, electromagnetic interference and radiation charge injection are other sources of errors for on-chip interconnects that cause the transmission to become unreliable and non-deterministic [7].

A true network-on-chip allows for an increase in concurrency. Due to the routing structure of a network, it is possible to have multiple outstanding transactions, increasing the reliability of transmission but reducing the throughput of a design. Mostly, the thoroughness of a network implementation on SoC is dependent on the specific application, and associated circuit and system connectivity, that is needed.

Designing with a Network-on-Chip provides a high level of reliability for global communication along interconnects and an improved level of concurrency that a bus structure for inter-module communication. However, a NoC requires a significant quantity of overhead to handle the transmission, routing and error correction. This extra overhead requires that the transmission devices

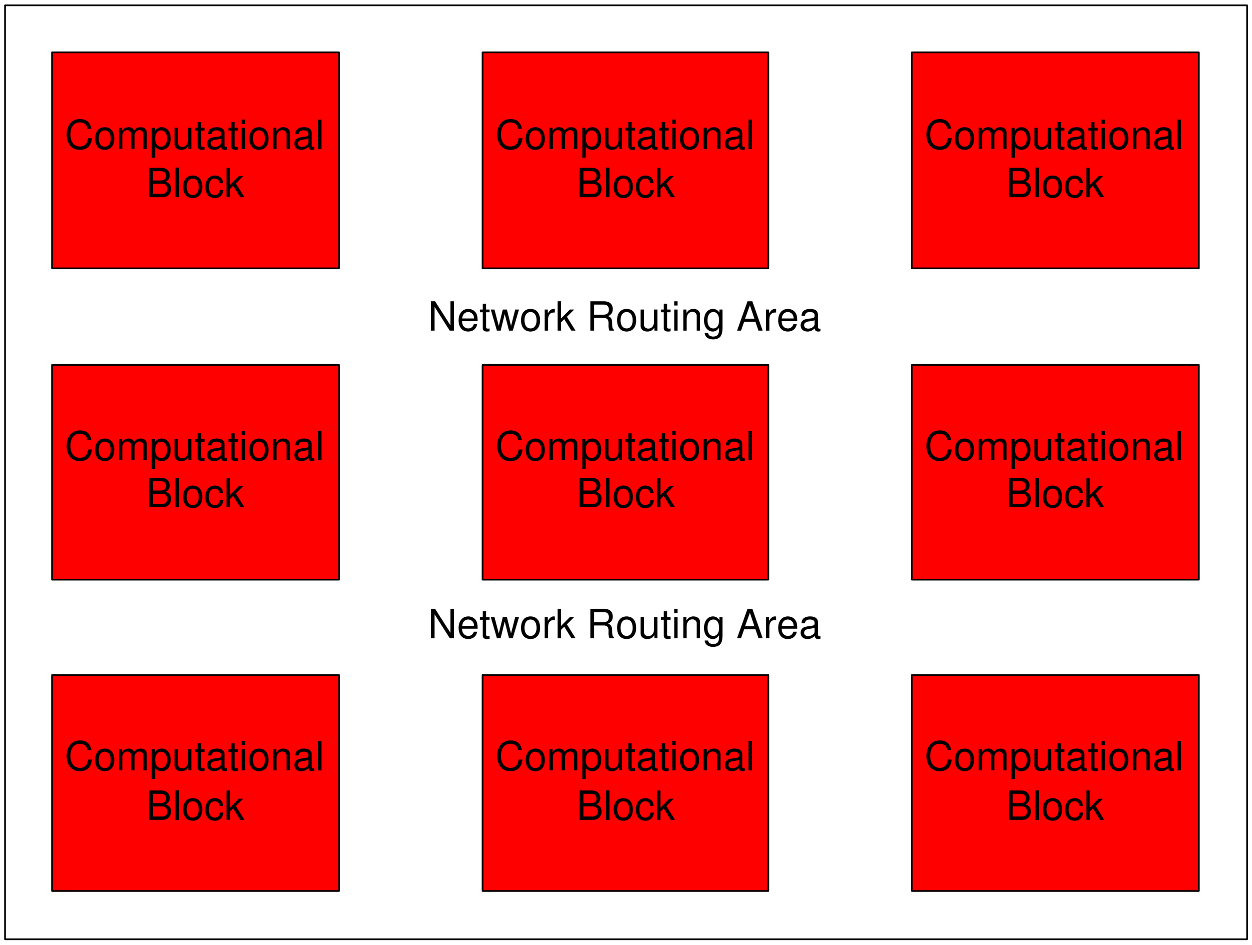


Figure 3.6: Possible Design for a Network-on-Chip Using Static Routes [32]

drive an entire interconnect wire, greatly increasing the size of driving devices. Using buffer insertion to reduce the device size requires doubling the number of connections because the interconnect wires could no longer be used for bi-directional communication. The routing tables require extra logic and a significant memory structure to handle the dynamically routing module communication. While a NoC improves the reliability and concurrency of inter-module communication, it requires a significant amount of overhead and thus occupy large amounts of area and increase power consumption. If error correction is not the most significant concern of the design, it would be best to consider other global communication architectures instead of NoC.

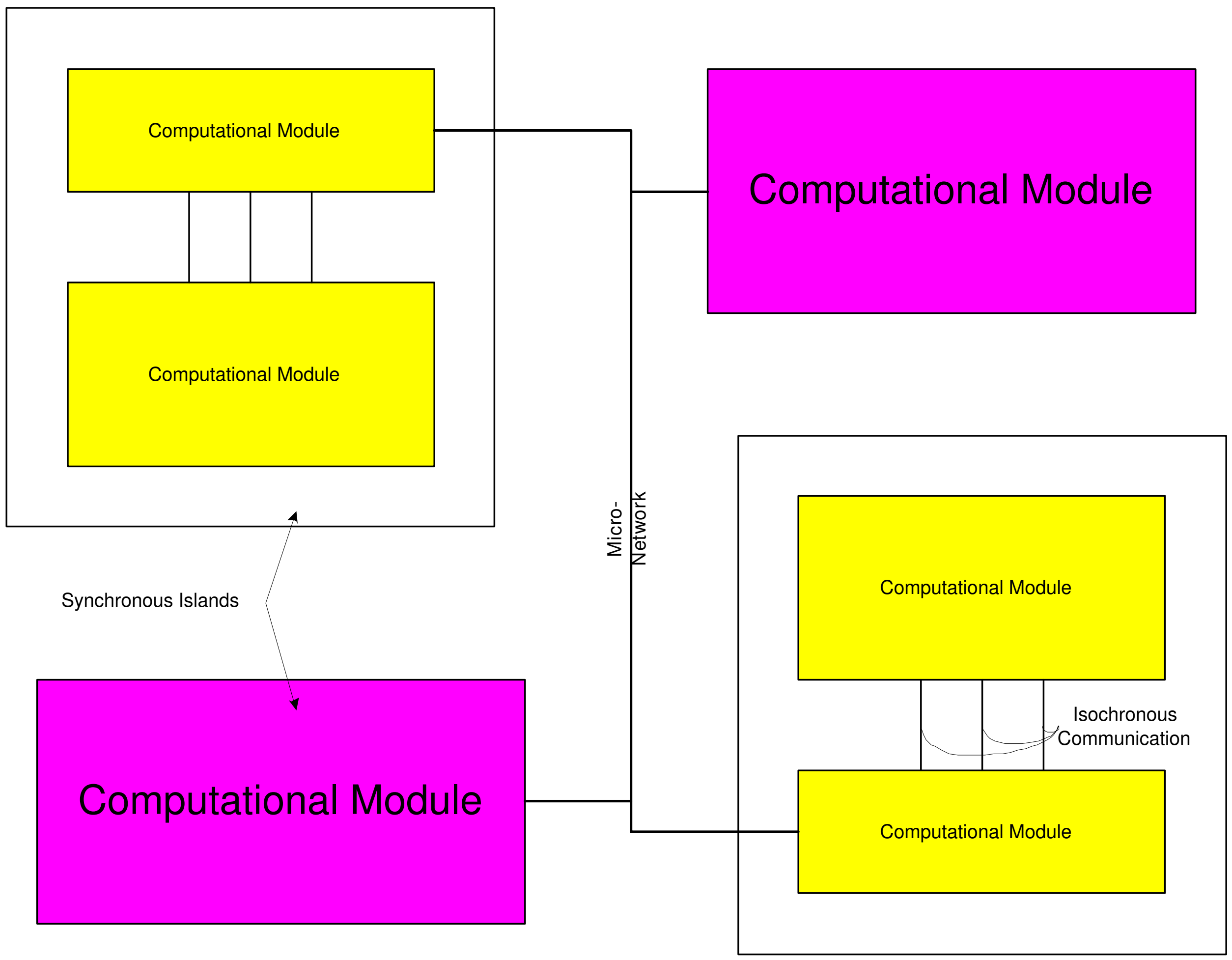


Figure 3.7: Possible Design for a Network-on-Chip Using a Router [36]

## 3.3 Wireless Communication

A final consideration for SoC inter-module communication is to use a wireless method of transmission. Radio frequency (RF) band communication is mentioned in [32] as a possible replacement to global clock and data distribution. On a similar note, [26] presents the potential of using optical interconnects on-chip for clock distribution at the GHz range. Both of these solutions avoid many of the process, temperature and voltage sensitivities of traditional metal interconnects. However, at both the sending and receiving end of the wireless communication, significant circuitry is required to convert the signal between a particular wireless domain and the metal interconnects. This circuitry is analog in nature. Thus, it needs to be properly shielded from environmental variations.

## 3.4 Conclusion

In this chapter, we have seen many of the ideas currently being investigated to assist digital designers to continue increasing the operating frequency, while considering the power consumption and reliability, of interconnects between modules. In Chapter 4, the main thrust of the research in this thesis is presented. This design focuses on improving the robustness, and possibly performance and power, of a buffer insertion design using a distributed FIFO in place of buffer insertion.

# CHAPTER FOUR

# ASYNCHRONOUS COMMUNICATION APPROACH

In modern processor design, with clocks greater than 1 GHz, much effort is spent in the generation and distribution of clock signals. Of concern in clock design is clock skew, clock jitter and duty cycle. These become important when distributing a clock signal from a single source, through meters of wire, to an entire SoC. A large number of digital systems are synchronous, and have to be designed to meet constraints such as: clock skew, clock jitter, duty cycle and power dissipation.

It is suggested in [6] and [7] that, because of the increasing disparity between computational performance and interconnect delays, a paradigm shift from computationally bound digital designs to communication bound digital designs is necessary to tackle the problems in shrinking technologies. The exploration of this thesis separated communication from computation and considered an asynchronous approach for communication. Asynchronous design methodology has not gained much of an audience, but has the capability to alleviate some of the problems of synchronous digital design. For the purposes of the research presented in this thesis, it is assumed that a system performs computations synchronously while global communication is done asynchronously.

In Section 4.1, four different types of clock classifications that depend on the delay assumptions of the communication channel, and their associated design problems, are presented. Section 4.2 introduces the idea of a distributed FIFO to help alleviate the problems associated with global interconnects and discusses the two implementations explored for this thesis.

## 4.1 Clock Classifications

In digital design, four types of clock schemes can be used: synchronous, mesochronous, plesiochronous and asynchronous [31]. Synchronous and mesochronous schemes refer to the phase relation between two modules using identical clock frequencies. Plesiochronous connections bridge two clock domains. Asynchronous designs are entirely without a global clock, but not without timing concerns. The first three clock classifications are discussed in sections 4.1.1, 4.1.2 and 4.1.3. Section 4.1.4 discusses the major problems in clocked designs. Finally, section 4.1.5 discusses the merits and faults of asynchronous design.

### *4.1.1 Synchronous Clocking*

A synchronous design has the global clock running at exactly the same frequency as the local clock with a known, fixed phase shift. The data flows through a synchronous circuit in a lockstep fashion with the clock. Since the data signals are synchronized with the clock, data can be sampled in a predictable and regular manner without any uncertainty. However, data cannot be sampled at any time. There exists a period of uncertainty or transition in synchronous design when data is being processed by a logic block. The maximum delay of a given logic block determines the upper limit at which the proceeding latch may sample data and thus an upper bound on the clock frequency of a system. The clock may transition faster than the slowest logic block that uses said clock. A simplistic equation for the maximum operating frequency of the system, shown in Figure 4.1, is stated in equation 4.1. Figure 4.1 illustrates how a synchronous digital design would be clocked. $\tau_q$ is the setup time of register $R_1$. $\tau_l$ is the maximum delay of the combinational logic. $\tau_s$ is the hold time of register $R_2$. $\tau$ is the maximum rate at which the clock can cycle. Figure 4.2 depicts the behavior of a synchronous clock with respect to equation 4.1.

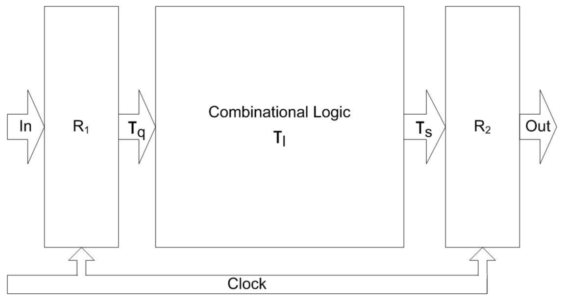


Figure 4.1: Synchronous Behavior of Digital Design [41]

$$\tau = \tau_q + \tau_l + \tau_s \tag{4.1}$$

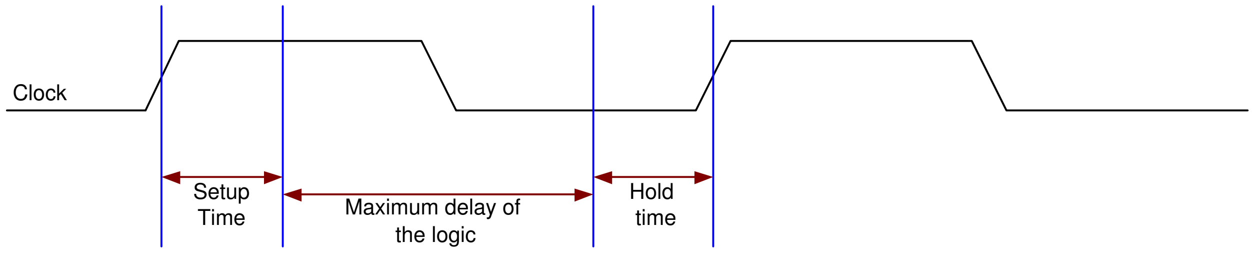


Figure 4.2: Synchronous Clocking

## *4.1.2 Mesochronous Clocking*

A local computational module that maintains the same frequency as the system clock but has an unknown phase shift is referred to as a mesochronous [31]. If a synchronous interconnect were bridging clock domains that were based on the same global clock, the receiving module would be unable to directly sample data on the interconnect. A synchronizing FIFO would be required to properly adjust the data to the phase of the receiving module. Instead of placing the synchronizing FIFO entirely at the receiving, the distributed FIFO scheme proposed in this thesis distributes the phase shift between the two modules across the interconnect. The difference in clock skew between the receiving module and the last stage of the FIFO would be negligible, allowing the receiving module to sample data at a predictable time. Figure 4.3 shows timing relationship a global clock (clock $A$) and a local clock (clock $B$). Notice that the delay between clocks $A$ and $B$ are constant between consecutive clock cycles.

## *4.1.3 Plesiochronous Clocking*

A Plesiochronous interconnect is an interconnect that bridges two clocks of slightly different frequencies [31]. This phenomenon is more prevalent in PCB design than in IC design at the moment, due to different chips being driven by different clock crystals. However, with ICs containing over

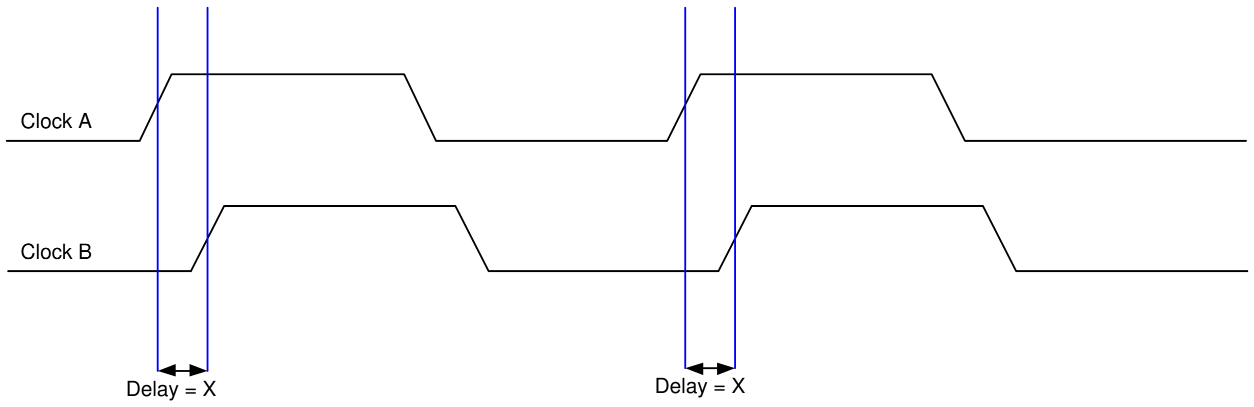


Figure 4.3: Mesochronous Clocking

one billion devices per chip, IP modules operating at different clock frequencies will become more prevalent, leading to complications in distributing a single clock amongst every module. With clock frequencies near each other, the phase between clocks will shift over time. Due to this dynamic phase shift, the data must be buffered, most commonly with a FIFO, at the receiving module. In addition to the FIFO storage at the receiving module, the interconnect must be occasionally reset because of overflow conditions.

### *4.1.4 Challenges of Synchronous Design*

One of the most challenging aspects of digital design is the distribution of the clock signal to every device that requires the clock. This is a particularly daunting task as potentially tens of millions of devices would need to be connected to the clock that operates above 1GHz [34]. Additionally, the clock distribution network consumes more than $50\%$ of the total power on a gigascale microprocessor [23]. Challenges include: managing clock skew, minimizing clock dissipation due to clock networks, clock distribution, and the synchronization of different modules running at different frequencies.

### *Clock Skew*

While clock skew is occasionally beneficial to synchronous designers, it is not useful without careful consideration. Clock skew refers to the spatial variation in arrival time of a clock transition. It is measured as the difference in the arrival time of a clock transition at two points on an IC. Skew can be either positive or negative depending on the location of the clock source and the routing direction. By definition, skew is constant between clock cycles [31]. This means that clock skew is constant and does not contribute to clock cycle variation, and is only a phase shift in the local clock signal. Skew is caused by mismatches in the clock distribution path and differences in the capacitive load of the individual distribution paths. Figure 4.4 shows two clocks (*A* and *B*) in skew when the clocks have an equal duty cycle. Notice that the delay between the rising and falling edges of clock *A* and clock *B* is constant over every clock cycle.

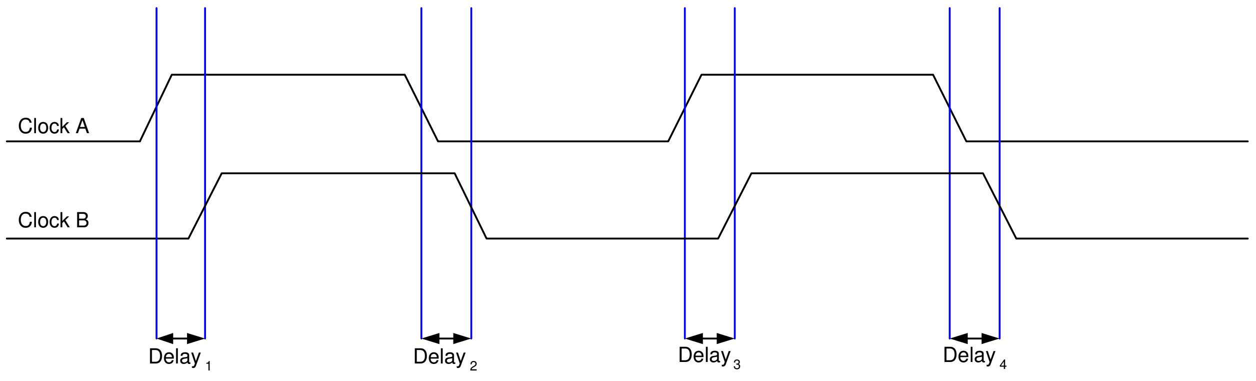


Figure 4.4: Clock Skew

Of concern when designing logic blocks and the flow of data are race conditions. A race condition in synchronous design occurs when latch separating two logic blocks is triggered before the second block can transmit its data. If the clock skew is large enough, the data in the second logic block could be over-written before the outbound latch is opened, resulting in data loss. Negative clock skew arrives against the flow of data and positive skew is distributed in the direction of data flow. Negative clock skew will never result in a race condition because with the clock arriving against the data, the latches that receive the clock first are latest in the data path. While never causing a race condition, negative clock skew does slow the overall throughput of the system. Designing systems with feedback loops, common in pipelined systems, would render negative skew inapplicable. Positive clock skew, in small amounts, can actually speed up the performance of any given system. This is because data can be let into the next logic block before that block has potentially finished the previous set of data. This is one of the principles behind wave-pipelining [13]. The problem with this occurs if the new data takes a path through the next logic block that is faster than the clock skew, because the new data can overwrite the previous set of calculations currently completed in the logic block before they are latched into the next stage. This results in an undetectable loss of data.

Since logic can flow in both directions down a particular path, the skew could be alternating negative and positive for the same set of devices. Thus, the designer must consider a worst-case scenario for clock skew. It is not feasible for the clock to be routed such that only negative skew would exist. Due to this, it is essential to design low-skew clock networks. The amount of clock skew in a global network approaches 70 ps (or $7\%$ of the total clock) in a 1 GHz design [34]. Without actively working to minimize skew, it could easily approach $25\%$, significantly increasing the delay penalty used to calculate the maximum operating freqeuncy, thus reducing the maximum operating frequency, of a globally synchronous design.

*Clock Jitter*

Clock jitter is the uncertainty of the arrival of the clock on a cycle-by-cycle basis. Clock jitter is measured as the difference between the expected and the actual arrival of the clock at a single point in an IC. For example, if the clock jitter at a certain point is $\tau$, then one rising transition could be delayed by $\tau$ and the following falling transition could be increased by $\tau$, causing a $2\tau$ decrease in the pulse width of a particular clock cycle and possibly changing the duty cycle of the clock. Clock jitter is never beneficial to synchronous designers. This jitter directly and negatively impacts the maximum performance of a synchronous system. Figure 4.5 shows the jitter in clock arrival times between two clock signals with no skew. $\text{Delay}_1$ and $\text{Delay}_4$ show positive jitter between clock *A* and clock *B*. At $\text{Delay}_2$, no jitter is experienced. $\text{Delay}_3$ demonstrates that jitter can also be negative. Notice that jitter is not a regular shift in the clock and can change between negative and positive shifts at any time.

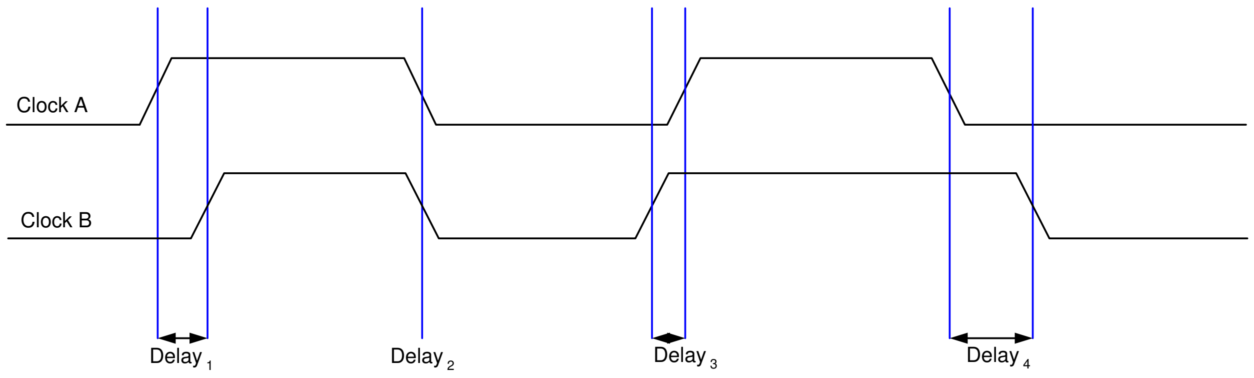


Figure 4.5: Clock Jitter

*Causes of Clock Uncertainty*

Skew and jitter are caused by several factors: (i) clock signal generation, (ii) process variations, (iii) dimensional interconnect variations, (iv) environmental variations, and (v) capacitive coupling.

Clock signal generation creates clock jitter. The generation circuits are analog circuits, like phase-locked loops or voltage controlled oscillators, that up-sample an external clock frequency from a crystal and create the higher, on-chip clock signal. These circuits are more sensitive to substrate noise generated by the surrounding digital circuits [18]. This substrate noise creates variations in the pulse width from cycle to cycle.

The delays of signals through different branches of the clock tree are matched to reduce clock skew. Process variations create differences in device delay on a graduated basis across the die. These static differences cause skew in the clock signal between clock tree branches. The causes of these manufacturing flaws include: dopant variation, lateral dimension variation, wafer-level and die-level variation of polysilicon and oxide variation [31].

Interconnect variation is the dimensional variation of the wire interconnects. Since the parasitic parameters (resistance, capacitance and inductance) of wires are based on the physical dimensions of the interconnects, these variations create a static imbalance of delays in different sections of the die, contributing to increased clock skew. The causes of these interconnect variations include inter-layer dielectric thickness variation and deviations in wire width and line spacing.

Environmental variations are significant contributors to skew and jitter. The most dominant environmental variations are temperature and power supply voltage. Both of these effects are directly related to the switching activity of a particular area on a chip. Part of the energy consumed in each device switching is lost to heat [1]. Thus, the more active a particular area is on a chip, the higher the local temperature. This phenomenon causes these areas of high activity to be at higher temperatures than areas of lower activity; this is especially noticeable in circuits that apply clock gating, where an area that has the clock disengaged is significantly cooler than an active area [1]. Upon initial consideration, the temperature variation would cause jitter as the devices and wires switch. However, the change in temperature can be measured in the microsecond domain. So, while it is a dynamic delay, for the purposes of digital design temperature is considered to cause a fixed delay.

Power supply variation is a significant source of jitter. The delay of buffers in a clock distribution network is strongly related to the voltage of the supply rails. When large quantities of devices switch simultaneously, the supply rails might not provide enough current at the nominal voltage. This causes a dip in the supply voltage until more energy can be drawn from the on-chip supply nets or from the off-chip power supply. Clock gating highlights this particular problem. When a clock is re-enabled, an entire module is suddenly added to the clock network, increasing load and drawing a significant amount of power to initialize.

Capacitive coupling, or crosstalk, is the final significant contributor to clock uncertainty. Coupling capacitance between the clock wires and the neighboring signal wires is a function of the current value of the clock and signals. Since the signals can transition at arbitrary times, the wire coupling capacitance can change between clock cycles, contributing to jitter. In a similar light, transistors have a significant quantity of internal capacitance that is dependent on the voltage level of the gate, drain, source and bulk of the device. These capacitances change in a highly non-linear fashion, and, being level sensitive, can change between clock cycles.

### *4.1.5 Asynchronous Clocking*

Asynchronous interconnects are logic driven and completely separate from either a local or global clock. Asynchronous design has been heralded as a solution to many of the problems plaguing synchronous design, particularly in high performance designs. The initially perceived value of asynchronous looks quite promising. Asynchronous design could potentially:

- operate at average case performance through relaxed timing constraints;
- lower power by only operating when data is present;
- significantly ease global timing problems from clock distribution to general communication constraints;
- better adapted for technology migration;
- automatically adapt to environment and process variations.

These claims have been made about asynchronous design since its inception, before the formal design methodologies were established by Huffman and Muller [27].

The principle behind asynchronous design is that instead of using a clock to determine the flow of data (e.g. latch and flip-flop control), asynchronous designs are event driven. To ensure proper behavior, asynchronous designs use handshaking to ensure data is transmitted successfully. This handshaking allows asynchronous designs to be active only when necessary and to be idle when there is no data to process, providing a potential for power savings at no additional hardware cost.

There is a price to pay for removing the clock. In digital design, area has recently been used as a metric for power consumption. The additional logic needed in asynchronous design to handle the handshaking control increases the size of asynchronous design an average of $10\%$ to $20\%$ greater than comparable synchronous design [15]. It is yet to be evaluated if the percentage increase in area overshadows the benefits of aynchronous design. Without a synchronizing clock, every logic block and communication channel needs to be more finely tuned to avoid race conditions and other logical hazards. With the size of digital designs increasing dramatically, the formal verification is challenging on an increasingly non-deterministic system. Logic synthesis and verification thus becomes an NP-hard problem [27]. Currently, CAD tool support of asynchronous synthesis is still in the experimental stages.

It is possible to use asynchronous interconnects to bridge clocked or clock-less modules. One interconnect methodology that has been much considered recently is a globally asynchronous, locally synchronous (GALS) clocking scheme. This scheme, first presented in [9], is concerned with a system that is no longer entirely synchronous, but has pockets or modules of synchronous behavior that are interconnected using asynchronous logic [22]. Each of the synchronous modules has its own phase-locked loop to ensure a regular local clock. This collection of independent synchronous modules is referred to as a multiple clock domain (MCD) design. Since each of these devices could potentially operate at differing frequencies, this could be handled in two manners. A plesiochronous system (a bus or network-on-chip architecture) incorporating a sufficiently large FIFO at either the sending or receiving end of communicating modules would solve the problem. Alternately, and possibly more simply, asynchronous interconnects could be used.

To incorporate an asynchronous interconnect into synchronous circuits, buffers are needed at both the sending and receiving circuits. The state of these buffers would enable the connecting modules to know when data could be sent or has been received. The assumption of regular data arrival is gone. To read data, modules that connect asynchronously require noticeably more overhead than modules that use a synchronous methodology for connection. Instead of being concerned about the timing characteristics of the global clock, asynchronous design could be used for medium and long interconnects [9, 15].

## 4.2 Distributed FIFO

A distributed FIFO approach is proposed to alleviate the problems encountered with long interconnect communication. This is similar to the idea of buffer insertion, using stages to break up wire delay. However, a distributed FIFO has the distinct advantage of: storing data at each stage, being placed in idle state without the use of additional hardware, and requiring no additional clock cycles to push data through multiple stages of the interconnect in the event that no further processing is required. This is in stark contrast with clock gating, where the clock must run until data is flushed from the computing module. Thus, beyond communicating with neighboring stages or modules, the FIFO is removed from the communication cost of global round-trip signals. The sender no longer needs to wait for its latest computation to travel completely down the FIFO to the receiving module. The sender only needs to wait until the first stage of the distributed FIFO has transmitted the data to the second stage before sending the next wave of data. In theory, this would allow the sender to produce data at a much higher rate controlled data transfer, with no concerns of data over-run in the event of pipe stalls, and no need for times (lock-step) transfers.

Figure 4.6 shows the block diagram of a distributed FIFO with $n$ stages between a sending and receiving module, both wave-pipelined. The sender's wave-pipelined clock signals the first FIFO control stage when data is ready to transmit. If empty (noted by the buffer_state line), the first FIFO control cell reads the data at the output stage of the sending module into the first column of buffer cells. The data is then transmitted down the distributed FIFO until the data arrives at stage $n$. FIFO control cell $n$ holds the data until the receiver signals the final control cell that the data has been read.

The proposed distributed FIFO scheme is driven by a hybrid wave-pipelined clock on both the input and output ends, making it easy to bridge multiple clock domains in a multiple domain

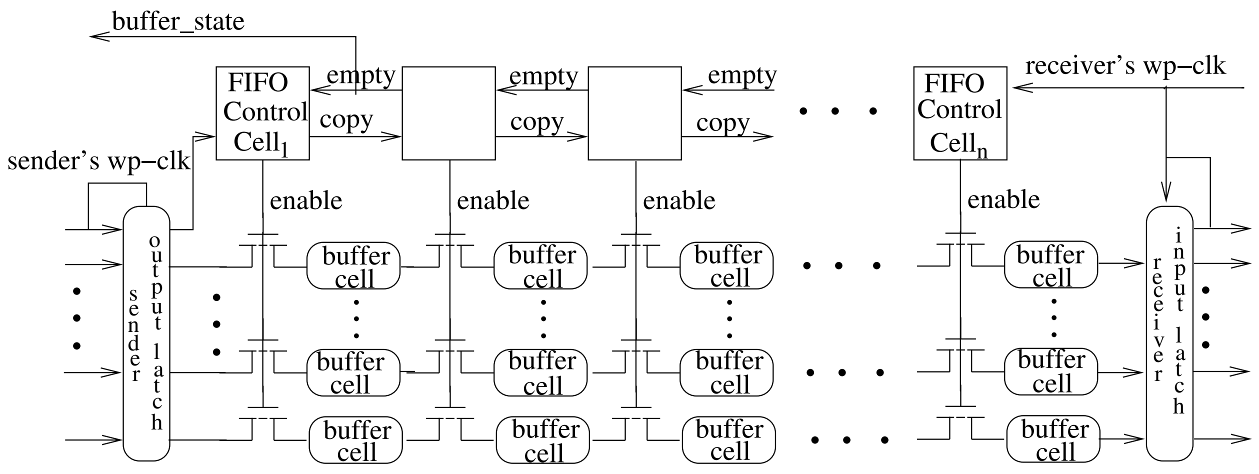


Figure 4.6: A Distributed FIFO Architecture

SoC. The pipelined clocks of the communicating modules are not necessarily synchronized. The FIFO buffers are data dependent, staying true to the concept of wave-pipelining by allowing data to asynchronously propagate through the buffer stages. Wave-pipelining allows local computations to be performed and propagated down the pipe asynchronously, while the system has the appearance of synchronous operation.

### *4.2.1 Asynchronous FIFO*

Asynchronous design was initially considered because throughput was our main concern [11]. When transmitting data between modules, it is already accepted that the delay is several clock cycles [7].

#### *Control Logic Behavior*

The design in Figure 4.7 uses a self-resetting NAND gate formed by transistors $N_2$, $N_3$ and $P_2$, along with inverters *R* and *S*. The operation of the NAND gate's pull-down network depends upon the status of the current latch's neighboring latches. The read and write signals driving transistors $P_3$ and $N_1$ are the same signals labeled as sender's wp-clk and receiver's wp-clk in the block diagram of Figure 4.6. These signals must be active in order for the NAND gate to generate the enable pulse. A successful generation of the enable signal occurs only when both transistors $N_2$ and $N_3$ are turned ON. The self-resetting NAND gate would set its output to logic 1 while its inputs are reset to logic 0 after a predefined delay (based on the load to be driven), the appropriate nodes are reset. The FIFO control circuit has inverters designed to maintain any given state at nodes *A* and *C* (the keepers in the figure). This permits for proper operation, i.e., in the absence of any activity at either the sending component or the receiving component these nodes would hold logic levels that ensure that the output of the NAND gate is at logic 1. In the event that the sending component has results while the receiving component is servicing other components or is inaccessible, we can fill the FIFO buffer and hold the data until the component it is intended for is ready to read the stored values.

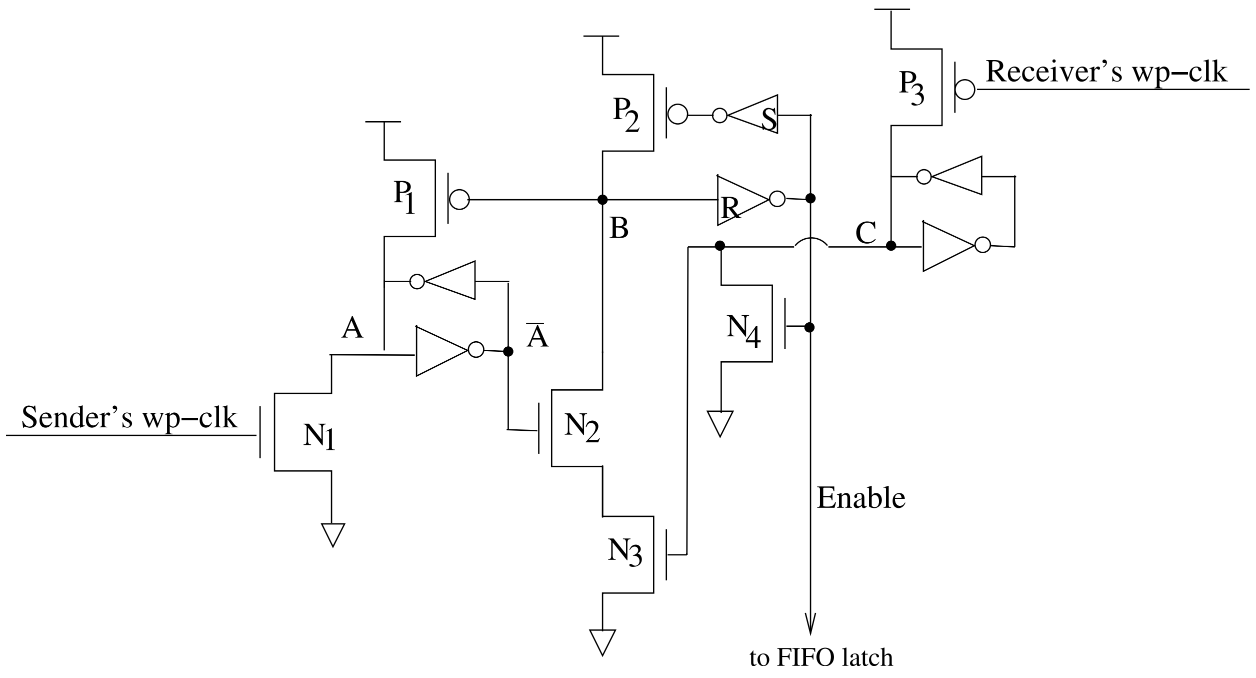


Figure 4.7: Improved Control Logic for the Asynchronous Distributed FIFO

Our design of the control circuit has a master reset that initializes nodes *A*, *B* and *C* to logic 1 at start-up. Initializing the design in this manner allows transistors $N_2$ and $N_4$ to be OFF while transistor $N_3$ is ON. This status indicates that the receiving module is ready or there are unfilled intermediate states. Data can be transferred from one communicating component to another. The sending component produces a pulse (the wave-pipelined clock), indicating that there is data to be transferred. For proper operation, the write signal must be at logic 1 for an inverter delay. Keeping the write pulse longer than one inverter delay would result in short circuit current being drawn since transistors $P_1$ and $N_1$ would eventually be conducting simultaneously. This would exacerbate power dissipation. Cascading the basic cell of Figure 4.7 would enable the progressive transfer of data along the data line. Our model requires that the receiving component indicate its readiness to accept new data. Then, the sending component would set the write signal high, triggering a situation in which the enable signal of each stage is sequentially set to logic 1, after sufficient delays to prevent data overrun. This action enables transistors on the data line (turns them ON), thereby passing the data from one stage to the next. The sending component is always apprised of the status of the FIFO buffer and the receiving component via the empty signal (Figure 4.6) of the first entry of the FIFO controller. Thus, we have a way to determine if the buffer is full or empty. If the receiving component stalls, the sending component is made aware via this empty signal. In Figure 4.10, a schematic of the circuit used on the data line to break the interconnect delay is shown. The circuit is a simple modification of the two inverter repeater with two additional pass transistors. The gates of the added transistors are driven by the enable signal and its inverse, generated by the control circuitry.

The design shown in Figure 4.7 is the final modified design. It was noticed during simulation that the initial design (Figure 4.8) had a couple of problems. First, and easiest to solve, are redundant devices. Three inverters ($R_1 - R_3$) where conveniently labeled to show that they each perform the same operation. This leads to a potential problem with race conditions, as well as additional area and power consumption. The most probable race condition would occur when $R_3$ would be loaded by a large bus. Since the capacitive load on $R_1$ and $R_2$ is independent of the bus width and the distance between stages, the FIFO control logic could perform the handshaking required for data transmission with the transmission actually occurring. Both of these problems are solved by combining $R_1$, $R_2$, and $R_3$ together into inverter *R* in Figure 4.7. Also, the state conductors are replaced with a feedback inverter from $\overline{A}$ to $A$. This weak feedback inverter replaces the function of the state conductor, while allowing the designer to cut one inverter per state node (*A* or *C*).

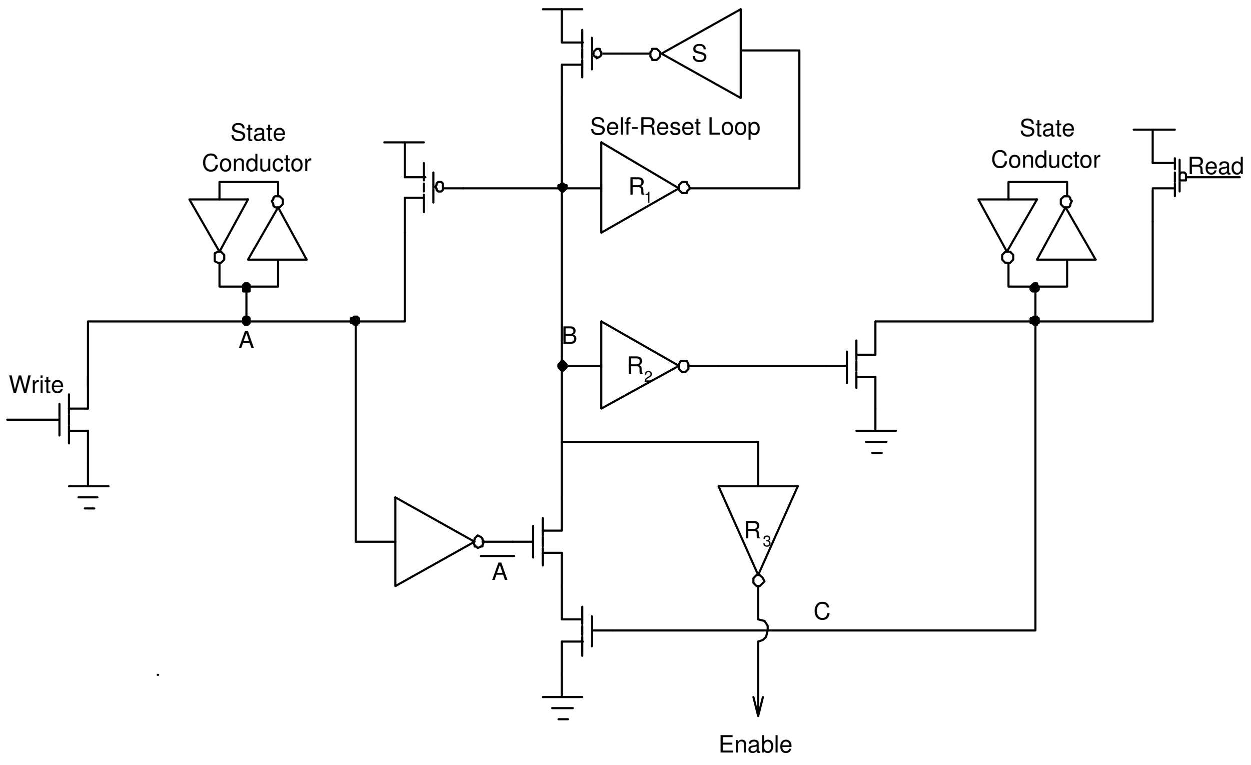


Figure 4.8: Original Control Logic for the Asynchronous Distributed FIFO (GasP) [38]

Another concern of the design for the asynchronous FIFO is the ability to interface with modules operating at any frequency below the maximum operating frequency of the asynchronous FIFO. It was discovered during simulation that if the write line is held high, the GasP controller will sample data as quickly as the control logic can reset. Because the sending module should not have frequency limitations placed on it (keeping with the asynchronous ideology), a method was developed to restrict the GasP to sample data only once per rising edge. This method uses three inverters and a pass transistor at both the sending and receiving modules. The inverter chain (Figure 4.9) is driven by the wave-pipelined clocks and is used to switch off the pass transistor after a sufficient amount of time (3 inverter delays) has elapsed. This is long enough for the signal to be passed to the first stage of the asynchronous distributed FIFO, because the state conductor preserves the new level on line $A$ after two inverter delays. This is a simple solution that has been tested at $0.25\mu m$ and $0.18\mu m$ and is expected to operate appropriately at further scaled nodes.

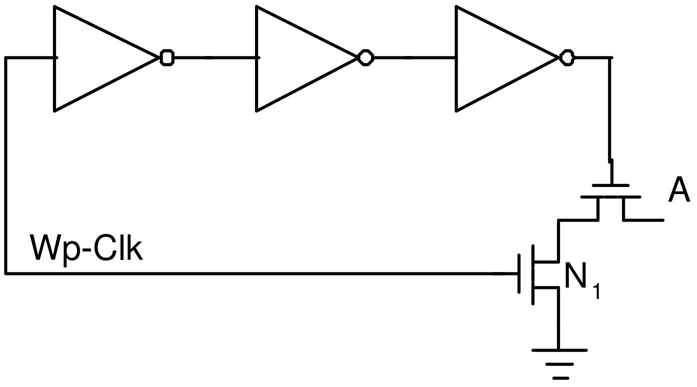
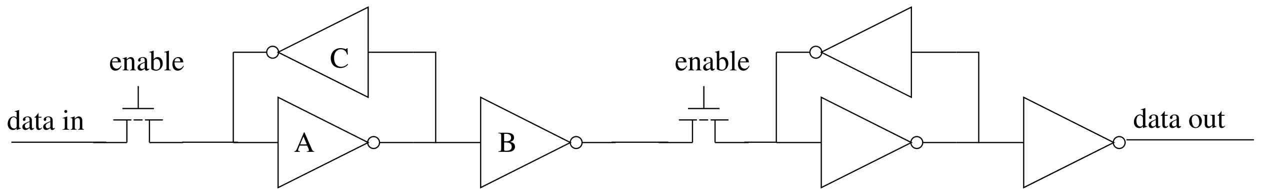


Figure 4.9: Three Inverter Chain with Pass Transistor

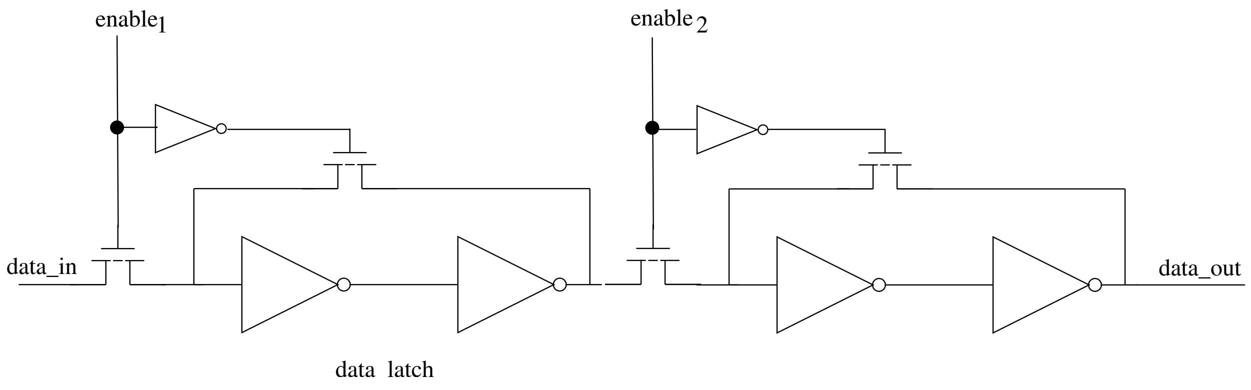
*Data Path Behavior*

The data path for the asynchronous distributed FIFO is simple in nature, especially compared to the control logic. The originally proposed data path (Figure 4.10(a)) used the *enable* signal from the control logic (Figure 4.7) to control a pass transistor that fed a 1-bit memory block [38]. The memory block is composed of a feed-forward and a feedback inverter. The feedback inverter is sized significantly smaller than the feed-forward device to prevent the incoming data from being overwritten before the feed-forward inverter can sample the new bit of data.

During experimentation involving testing the distance between distributed FIFO stages, it became apparent that the design for the feedback transistor from original data path was creating a problem. The feedback transistor was stronger than the previous feed-forward inverter when the feed-forward inverter was more than 1 mm away from the pass transistor. To eliminate this problem, the data path was modified slightly by using a pass transistor to provide the feedback path instead of a small inverter (Figure 4.10(b)). The feedback pass transistor, also driven by the enable



(a) Original



(b) Modified

Figure 4.10: Data paths for the Asynchronous FIFO

signal, only creates a feedback path when the data path pass transistor is closed. Thus, the memory device only stores data when the enable signal is low. When the enable signal is high, the feedback path is cut off while new data is read from the previous stage.

The new latch design could potentially experience a race condition if the enable signals were controlled by a clock instead of the associated control logic driving the enable. During normal operation of the asynchronous FIFO, it is possible for the same bit of data to exist in adjacent stages. By the nature of the handshaking of the asynchronous protocol, a stage requires the next stage to be EMPTY and the previous stage to be FULL for data propagation to occur. When the transfer occurs, momentarily both stages are FULL with the same bit of data. Shortly thereafter, the previous stage will become EMPTY again. This means that while more than one stage can have the same bit of data, this can only occur in the forward direction of propagation. It is not possible, by design, for a latch to raise the *enable* signal twice before the *enable* signal of the next stage is raised. On a related note, the operating frequency could be measured by the delay from one stage's *enable* signal to its predecessor's *enable* signal.

*Control delay vs data path delay*

To achieve ideal performance from the distributed FIFO, the delay of the control path should not be larger than that of the data path, thereby becoming a bottleneck in data transmission. Ideally, the control circuit's delays should be such that they match those of the wost case data path. To strive for this goal, the GasP was designed as an aggressive asynchronous FIFO control [4, 39], moving away from a truly delay-independent asynchronous design. The GasP, however, falls short of the ideal timing characteristics of a distributed FIFO. The GasP control logic requires 6 inverter delays between data sets (4 forward and 2 backward), while the data has a pass transistor and 2 inverters per stage. With some additional wave-control circuitry, it was shown in [21] that it is possible to reduce the forward latency of the GasP. However, this design was not shown to increase the throughput of the channel, it just shifts the balance between the forward and backward communication paths.

### *4.2.2 Synchronous FIFO*

A synchronous distributed FIFO was designed for a more fair comparison to the asynchronous design. The synchronous FIFO is based on a distributed shift register (DSR) approach shown in Figure 4.11. The DSR design uses a clock signal to drive the latches. There is no feedback in this design to create a memory device. This could possibly hinder the performance of the DSR under extreme conditions. This design was chosen for its simplicity, as it was primarily intended as a reference for comparison. It should be noted that the design for the synchronous FIFO is not the most optimal design in terms of robustness or performance. Unlike the data path for the asynchronous FIFO, the synchronous FIFO requires pass transistors to be of alternating type (or operate on opposite levels of the clock). If every pass transistor was nmos, then the DSR would become just a global interconnect with interspersed buffers. The current DSR design does not have true memory devices; it relies on the speed of the clock and the parasitic capacitances of the devices and wires to maintain data.

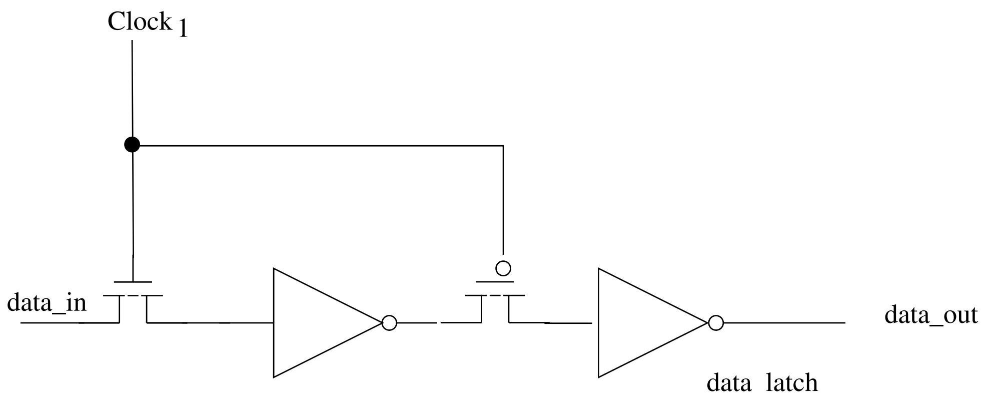


Figure 4.11: Synchronous Distributed FIFO - Shift Register

## 4.3 Conclusion

This chapter presented several classifications of clock timing relationships present in all digital designs. In addition to the communication schemes discussed in Chapter 3, both synchronous and asynchronous distributed FIFOs were presented. The control and data path logic for the asynchronous distributed FIFO was explained. Chapter 5 presents the results of the simulations that were conducted for performance analysis under ideal and extreme operating conditions.

# CHAPTER FIVE

# GASP FIFO PERFORMANCE

Hypothesis and theory are all well and good, but at the end of the day, simulation and measurement are necessary for a quantitative analysis of design work. In this chapter, the performance of asynchronous and synchronous distributed FIFOs are compared. Where relevant, buffer insertion is also discussed. All simulations were conducted post-layout, using Taiwan Semiconductor Manufacturing Company (TSMC) technology processes with the Cadence CAD package.

## 5.1 Basic Operation

Figure 5.1 shows plots of the nodes of interest within a unit cell of the control logic (Figure 4.7). These cells were duplicated at appropriate intervals (3 mm for our experiment in a $0.25\mu m$ technology) to form a distributed FIFO buffer for a typical SoC design. A single stage of both the control and the data line circuits are analyzed here and the associated delays are recorded, which can be read from the charts of Figures 5.1 and 5.2. At the onset of a write request 192.4 ps elapse before the latch is enabled for data copying to occur. This is remarkably fast at the $0.25\mu m$ technology node. It takes 320.27 ps to generate the enable pulse after a transfer request is issued, 29.67 ps to reset the NAND gate, 159.9 ps to turn OFF transistor $N_2$ after a successful transfer, 53.34 ps to turn OFF transistor $N_4$ after a successful transfer, 759.35 ps to perform a succeeding transfer. These delays provide some sense of the overhead incurred from having such control circuitry to control bundled data buses. It must be noted that the reported delays take into account enabling a 32-bit bus. This implies that a single control stage is designed to gate signals on 32 data lines. The recorded values are based on the required delays for proper operation. The generated enable signal has a maximum frequency of 1.67 (600ps) GHz, in a modest $0.25\mu m$ technology, and this is limited by the node capacitance of the self-resetting NAND gate (labeled as node *B* in Figure 4.7). This node has the largest load amongst all the nodes of the circuit designed to generate control of the data line FIFO.

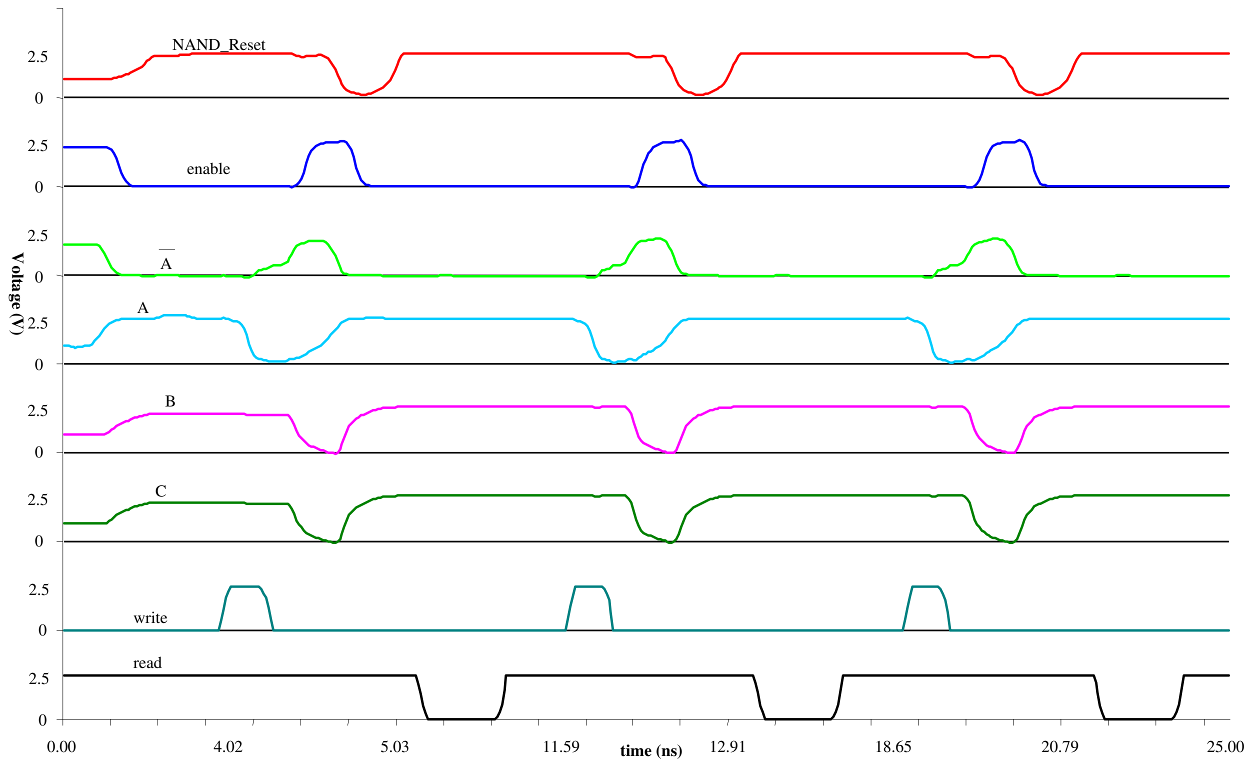


Figure 5.1: Normal operation of the asynchronous FIFO

The simulation results for burst mode operation appear in Figure 5.2. The simulated case is one whereby the initial transfer has sparse data bits and no transfers are required for some time and eventually several data bits need to be transferred in rapid succession. Figure 5.2 depicts a case in which the computing component signals the FIFO controller to indicate that it is ready to accept data. There is only a single 32-bit word to be transferred when the FIFO is signaled, thus the wave-pipelined clock stays at logic 0 after the initial transfer. The FIFO buffer propagates the data to the receiving module in 8 ns and returns to its normal state (all nodes corresponding to B and C set to logic 1). There is no activity for 20 ns and a burst of data arrives at time 21 ns and is transfered.

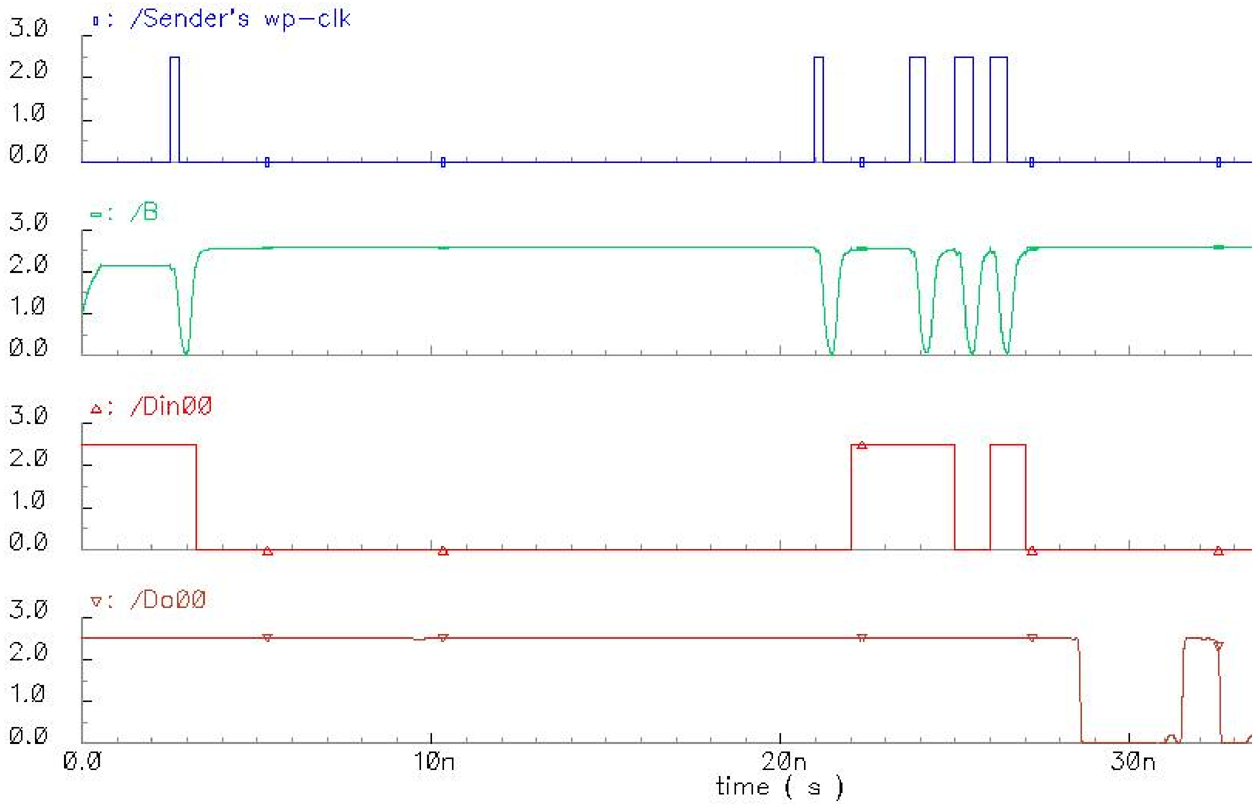


Figure 5.2: Burst mode operation of the asynchronous FIFO

## 5.2 Power/Area/Performance/Energy

The argument for using the distributed buffer centers around the idea that buffer insertion will neither be the fastest solution for interconnect delays, nor the solution that consumes the least power. The departure from the classic, two-inverter insertion scheme for delay reduction leads to an increase in transistor count and additional switching activity. Comparing power dissipation and area to the buffer insertion scheme would result in a comparison biased against the asynchronous distributed FIFO. Thus, the DSR circuitry is used to make comparisons. Power for the two designs is computed in idle, burst and normal modes. The normal mode refers to an operation that can occur at the highest frequency achievable for a continuous data stream. In idle mode, the clock driving the shift register is gated.

Table 5.1: Speed and Power Measurements in $0.25\mu m$ over 16 Stages and a 32-bit Bus

| Network - Speed | Power | Latency | PDP |
|---|---|---|---|
| FIFO - 1.67 GHz | 111.25 mW | 8 ns | 890 pJ |
| FIFO - 750 MHz | 87.25 mW | 8 ns | 698 pJ |
| FIFO - Burst | 13.50 mW | 8 ns | 108 pJ |
| FIFO - Idle | 192 $\mu W$ | N/A | N/A |
| DSR - 750 MHz | 80.75 mW | 24 ns | 1938 pJ |
| DSR - Burst | 28.75 mW | 24 ns | 690 pJ |
| DSR - Idle | 100 $\mu W$ | N/A | N/A |

Table 5.1 has the related delays, power and energy figures relating to each mode of operation. The FIFO network consumes $40\%$ more power than the DSR when in idle mode. During normal operation, the FIFO consumes $7\%$ more power than the DSR at the same operating frequency. This power difference could be explained by the amount of leakage current lost by each network. The FIFO network consumes more area, and requires larger devices, than the DSR network. The burst mode simulation shows the most promise for the FIFO network in terms of power dissipation. For data sets that are intermittent, the FIFO network maintains the same latency while consuming $53\%$ less power. The maximum speed for the FIFO is more than double that of the DSR (1.67 GHz compared to 750 Mhz). A 16-bit stage FIFO design has 584 transistors compared to 510 for the DSR. This is a $12.6\%$ increase in device count. For such a small penalty in area, the FIFO buffer provides a significant $55\%$ performance improvement.

The power and speed analysis, summarized in Table 5.1, is for a 16-stage, 32-bit bus in $0.25\mu m$ technology. The distributed FIFO designs were tested at four rates of operation: maximum throughput, highest equal throughput (with the asynchronous FIFO slowed to match the speed of the synchronous FIFO), burst mode, and idle. The asynchronous FIFO was found to perform at a maximum rate of 1.67 GHz with a latency of 500 ps per stage and 8 ns of propagation delay across the entire system. The DSR was found to have a maximum clocking rate of 750

MHz, thus a delay of 1.5 ns per stage and 24 ns for the entire system. To implement burst mode operation, the DSR required additional hardware to perform clock gating after the final block of data left the DSR. The asynchronous FIFO required no additional logic for burst mode operation. For power analysis, the two systems were tested over the course of 30 ns of continuous operation, excluding burst mode, which was three sets of data transmitted inside of 30 ns. The equal frequency was 750MHz, at this node, the asynchronous FIFO consumed 86.25 mW of power, while the DSR consumed 80.75 mW. However, the power-delay product for the asynchronous FIFO (890 pJ) is less than half that of the DSR (1938 pJ), because of the significant difference in propagation delay. It should be noted that the propagation delay of the asynchronous FIFO is independent of the clocking frequency. If the inputs were slowed to 1 Hz, the propagation delay would continue to be 8 ns. For the synchronous design, the propagation delay is a factor of the clock speed ($Propagation_{delay} = clk_{rate} \times \#stages$). In burst mode, the asynchronous FIFO begins to shine in terms of power. By its nature, the asynchronous design is idle unless there is data to move. In contrast, the DSR, even with clock gating, must wait for the data to finish propagation before the clock may be disengaged. When both systems are idle, no data and no clock, the DSR consumes nearly half that of the asynchronous FIFO ($192\mu W$). This could be due to the additional leakage current of the extra devices in the control logic of the asynchronous FIFO. The values recorded in Table 5.1 are post layout results. The asynchronous FIFO was also designed and tested at a $0.18\mu m$ technology node. The maximum operating frequency of the asynchronous FIFO is 2.35 GHz in $0.18\mu m$ technology.

Table 5.2 shows the results of a test between the asynchronous FIFO and buffer insertion scheme for wire lengths broken into 1 mm sections. The experiment was conducted using a $0.18\mu m$ TSMC process. The buffers and the control logic of the asynchronous FIFO were sized appropriately to handle the wire parasitics of 1 mm. Wires longer than 1 mm were broken into 1 mm sections. Thus, a 2 mm wire would have 3 stages, a 5 mm wire would have 6 stages, etc. The asynchronous FIFO was tested with a 16-bit bus attached. The buffer insertion model tested using a single bit and then the result was multiplied by 16. This model was chosen because of the nature of the interconnect for buffer insertion.

For Table 5.2, the throughput was measured as the highest rate at which data could be input to the interconnect. For buffer insertion, the throughput and latency were measured as the time required for a signal to reach the end of the wire. This is the highest level of reliability for buffer insertion because there is only one set of data in the channel at a time. For the asynchronous FIFO, the rate of transfer is the time it takes the first stage to transmit data to the second stage, the latency is the time is takes the data to arrive at the final stage. The energy calculation, or Power-Delay Product, was computed by multiplying the average current, the throughput, and the 1.8V power supply voltage. Notice that, while buffer insertion has considerably less latency (1 inverter per stage vs. a 6 inverter path in the asynchronous FIFO), the asynchronous FIFO shows a significant improvement in terms of energy consumption and a constant throughput, regardless of wire length. The other setup considered for this experiment was the duplication of the interconnect 16 times to create a 16-bit interconnect. Testing a 1 bit interconnect and multiplying the results (for power) by 16 was considered an appropriate estimation and equivalent to testing a 16-bit, buffer insertion styled interconnect. A 16-bit bus was chosen because the asynchronous FIFO was ported to $0.18\mu m$ to integrate with an 8x8-bit multiplier (i.e. 16-bit result).

## 5.3 Parameter Variation Tolerance

Power supply voltage and temperature fluctuations could lead to unreliable operations. The increased device densities imply that a large number of gates switch, resulting in increased dynamic activity. This could result in power supply voltage variations. These changes in supply voltage can result in data corruption or increased delays. Circuits need to be able to tolerate these changes to maintain reliable performance during operation. Therefore, it becomes necessary to ensure that a design can tolerate these changes. The asynchronous and synchronous distributed FIFO circuits were tested over a large range of power supply voltage variations and temperature variations with

Table 5.2: Energy vs Wire Length in $0.18\mu m$ for a 16-bit Bus

| Method | Distance | Avg Current (mA) | Throughput (ps) | Latency (ps) | Energy (pJ) |
|---|---|---|---|---|---|
| GasP | 0 mm | 2.497 | 500 | 500 | 2.247 |
| GasP | 1 mm | 4.209 | 500 | 850 | 3.788 |
| GasP | 2 mm | 5.129 | 500 | 1200 | 4.616 |
| GasP | 3 mm | 6.114 | 500 | 1550 | 5.503 |
| GasP | 4 mm | 7.095 | 500 | 1900 | 6.386 |
| GasP | 5 mm | 8.070 | 500 | 2250 | 7.263 |
| Inv | 0 mm | 2.497 | 100 | 60 | 0.63818 |
| Inv | 1 mm | 10.62 | 300 | 100 | 5.7348 |
| Inv | 2 mm | 14.81 | 400 | 174 | 10.6632 |
| Inv | 3 mm | 17.59 | 500 | 265 | 15.831 |
| Inv | 4 mm | 16.78 | 700 | 350 | 21.1428 |
| Inv | 5 mm | 16.72 | 870 | 435 | 26.1835 |

$0.25\mu m$ TSMC process.

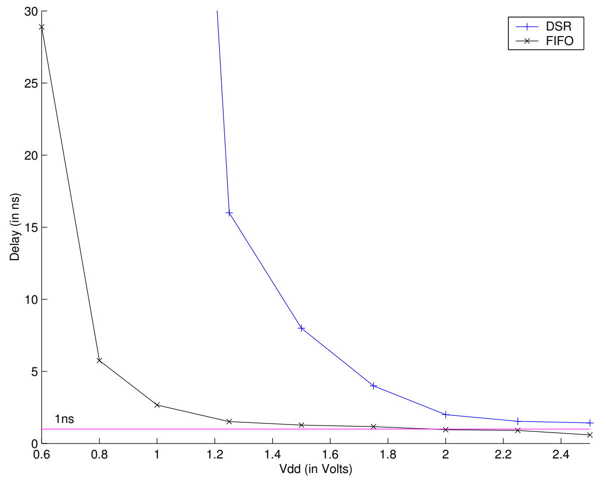


Figure 5.3: Effect of supply voltage variation on FIFO performance

### *5.3.1 Voltage Variation*

Figure 5.3 shows the designs' operation delays as the power supply voltage is reduced. The supply voltage was reduced from 2.5V to 0.6V. At 0.6V (barely above $2V_{th}$), it was determined that the

FIFO slowed to 34.6 MHz, a degradation in operating speed. The latches of the channel held and propagated valid data even at this low voltage. Simulation results show that the power supply voltage can only drop to 1.3V before the operation is incorrect for the DSR and the classical inverter approach. With the power supply reduced to 1.3V, the distributed FIFO is $94\%$ faster than the DSR.

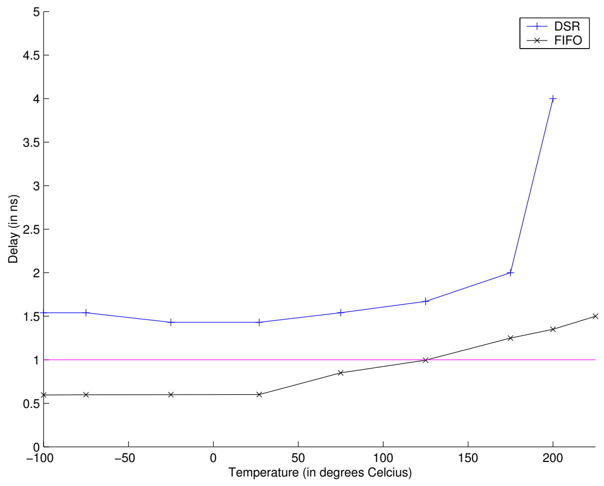


Figure 5.4: Effect of temperature variation on FIFO performance

### *5.3.2 Temperature Variation*

Temperature was varied from -100 $^{o}$C to 225 $^{o}$C to evaluate the designs response against temperature changes. Performance (shown in Figure 5.4) degrades by $60\%$ at 225 $^{o}$C for the FIFO. The DSR, however, takes a significant performance hit above 175 $^{o}$C and ceases to function beyond 200 $^{o}$C.

These simulation results indicate that the distributed FIFO scheme, though sensitive to environmental variations, is able to maintain correct functional operations. The distributed FIFO also shows very good tolerance to temperature changes as shown in Figure 5.4.

## 5.4 Interface with Clocked Systems

In addition to performance comparisons, it was also determined that FIFO should be attached to other computation modules as a proof-of-concept for interfacing with other modules. The module that was chosen was an 8x8-bit hybrid wave-pipelined multiplier [40]. The multiplier was slowed to 450 ps because of the fragile nature of the operating frequency of the multiplier. The only additional logic required to interface with the multiplier was a set of latches (Figure 5.5) driven by the output wave-pipelined clock of the multiplier. Figure 5.5 shows the general connectivity of the multiplier with the asynchronous FIFO. The multiplier receives two 8-bit inputs, *X* and *Y*, in addition to a clock signal. The input clock and input data transition in lockstep and the multiplication begins. Since the multiplier is hybrid wave-pipelined, the input clock travels with the product through the multiplier. Once the result in finished for each input wave, the result is placed on the output, synchronized with the rising edge of the wave-pipelined clock. The result is captured by a set of latches, driven by the wave-pipelined clock, to ensure that the data is ready when the FIFO control logic enables a data capture on the 16-bit bus by raising the enable signal. Then the wave-pipelined clock is used to begin the transmission of data through the asynchronous FIFO.

The addition of the output latches was made after careful analysis of the multiplier, paying particular attention to the delay between the wave-pipelined clock and the arrival of the multiplication results (Figure 5.6(a)). Figure 5.6(b) shows the operation of the hybrid wave-pipelined clock, the enable signals of two FIFO stages, and the data path of bit 13 from the result of the multiplier. While this example does not present absolute proof that the FIFO could be integrated with any clocked system, the author feels that it would not be complicated or time-consuming to integrate the asynchronous FIFO with any clocked system once the relationship between the clock and the arrival of the data is known. Most likely, all that would be needed is the same set of latches that were required for the multiplier.

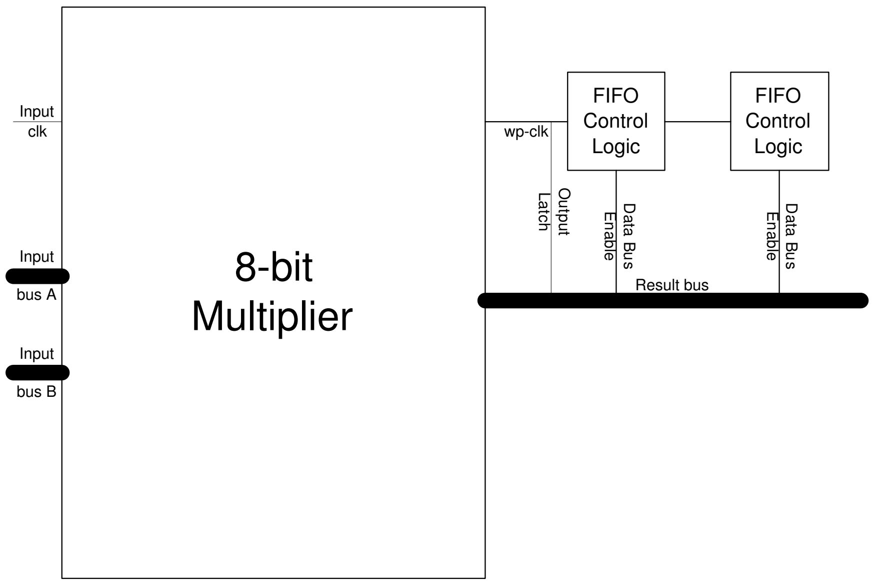
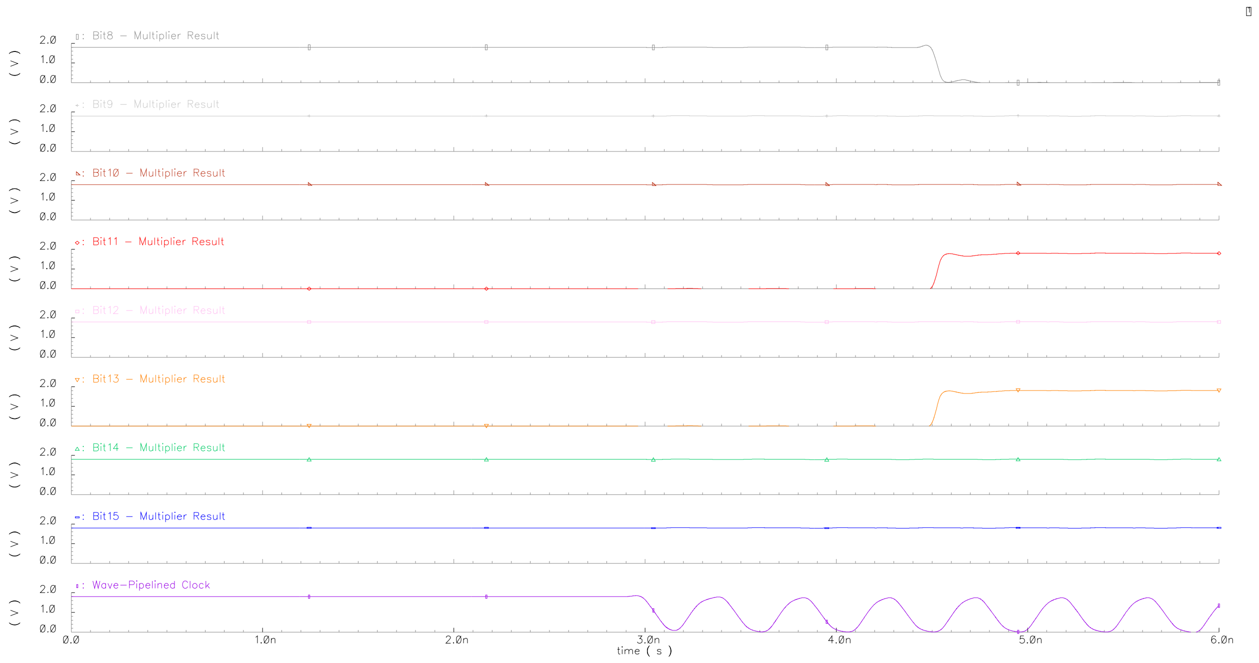


Figure 5.5: Block diagram of the multiplier, FIFO & latches

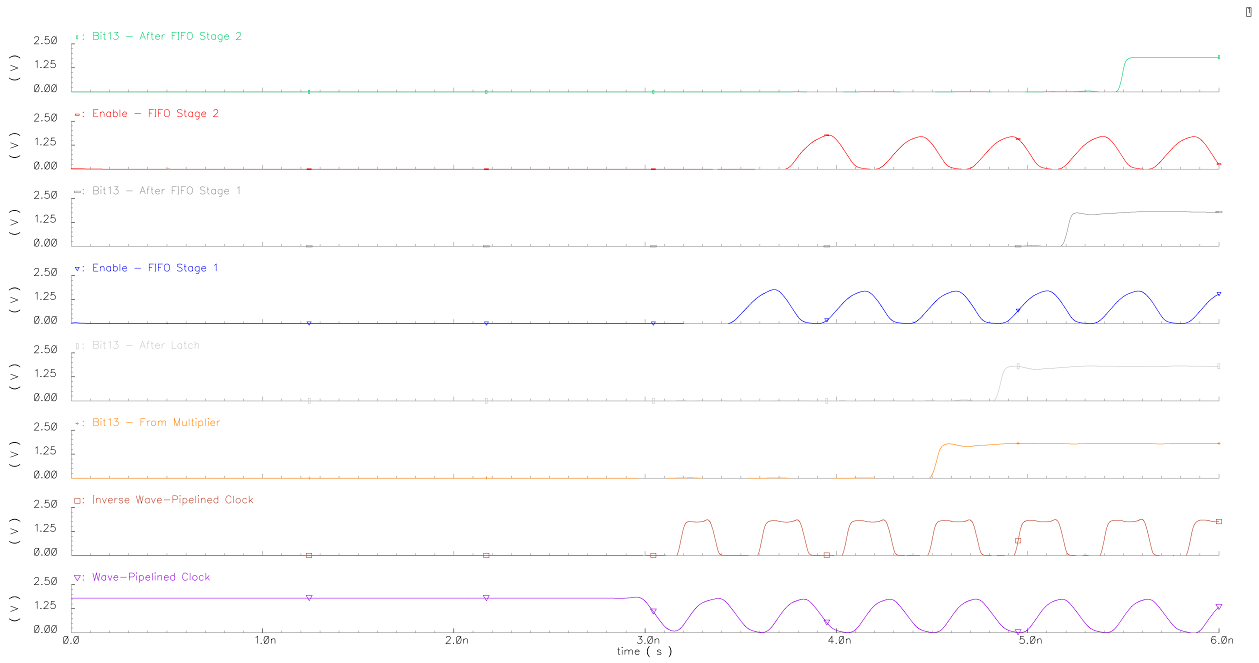
The wave-pipelined clock is delayed in Figures 5.6(a) and 5.6(b) because of the load attached by the set of 16 output latches. The strength of the wave-pipelined clock was improved until performance requirements were met. However, the wave-pipelined clock in these two figures arrives slightly behind the multiplier result.

## 5.5 Conclusion

This chapter explored the performance of the asynchronous distributed FIFO in comparison to buffer insertion for various wire distances and a synchronous distributed FIFO for analysis of power consumption and environmental parameter variation. The asynchronous distributed FIFO was shown to have superior performance over buffer insertion for interconnect wires of sufficient length and over the synchronous distributed FIFO when considering the energy consumption per transaction. However, the asynchronous distributed FIFO consumes more power than the synchronous version and takes considerably more design time that either the synchronous FIFO or buffer insertion.



(a) The Output Wave-Pipelined Clock and Resultant from the Multiplier



(b) The Data Path for Bit 13 from the Multiplier through the FIFO

# CHAPTER SIX

# RESEARCH CONTRIBUTIONS, FUTURE WORK

Now that the work is done, it remains necessary to discuss where this research is useful and what future avenues of research exist. This chapter discusses the target audience of this work, the research contribution of the thesis, and possible future directions of research on this topic.

## 6.1 What is the Target Audience for this Design

The target audience of this research are digital designers that are working with a significant number of IP cores. If the additional area could be afforded in the design space, the ease of integration would outweigh the redesign time, if redesign is possible, of the IP cores being used. On a similar note, the asynchronous FIFO has been shown to handle environmental, and theoretically, process variations that plague small technologies, with grace and reliability. Full-custom ASIC designs that utilize MCD or delay independent communication would benefit from the additional reliability provided by the memory and dynamic reaction to changing environmental variations. The research for the asynchronous FIFO was performed using $0.25\mu m$ and $0.18\mu m$ technology nodes because smaller technology nodes were not available for design and simulation. With interconnect delays becoming more dominant in smaller technologies, it is felt that the asynchronous FIFO would be increasingly useful below $0.13\mu m$. Further scaling would result in less current drive and thus require interconnect lengths to be reduced drastically. Distributed asynchronous FIFOs could be a viable solution to this problem.

In comparison to the other architectural designs presented in Chapter 3, the distributed FIFO offers some distinct advantages, but not without shortfalls of its own. The distributed FIFO architecture, similar to buffer insertion, provides a significantly higher data throughput per line in point-to-point communication than the bus architectures or networks-on-chip. The distributed FIFO makes global interconnects operate at the speed of local interconnects, while providing a distributed memory structure to ensure that data integrity is maintained for the entire length of the interconnect, an improvement over buffer insertion.

The studied distributed asynchronous FIFO would require additional hardware, on the order of doubling the device count, to provide bi-directional communication. This could possibly slow the maximum rate of communication by introducing pass transistors into the flow of the wave-pipelined clock in the control logic. While network routers can fork data by duplicating packets at the router, forking data with the asynchronous FIFO requires waiting on both forks to be empty before transmission can occur. In terms of reliability, while the asynchronous FIFO provides good tolerance to environmental parameters, with the interconnects themselves becoming unreliable, the error correction available to networks-on-chip provide the highest level of reliability for data transmission.

## 6.2 Research Contributions

- **Alleviate clock distribution**: By using an asynchronous global design, the global clock distribution requirement is eliminated. The only concern is the frequency of the global clock, the skew and jitter can be mostly ignored because a PLL or other clocking logic will recreate a clean clock signal at the local level.

- **Provide storage that is otherwise needed on the modules**: When interfacing between clock domains, or transitioning between synchronous and asynchronous logic, buffers are needed to shift the data movement to synchronize with the local transitions. A distributed FIFO creates this data storage on the interconnect itself while still providing a means to synchronize the data to the sending and receiving modules.

- **Idle mode is achievable at no cost**: Since the proposed distributed FIFO is asynchronous, the control logic is entirely event driven. There is activity only when data is present. This is one of the inherent advantages of an asynchronous design over a synchronous one. For a synchronous design to reduce power consumption during idle states, additional clock gating logic, and timing, are required.

- **Alleviate the need for multiple clock cycles for data transmission**: On the proposed distributed FIFO, consider the first two stages and assume they start empty. When the sender has data, the first stage is notified and the data is read. Then the second stage is checked and since it is also empty, the data is transferred to the second stage. The first stage is now empty and ready to receive data. This pattern continues down the entire distributed FIFO, regardless of length. Thus, while the interconnect could be of any length, data can be read at a fast and predictable rate. This is important as interconnect delays increase and computational delays decrease. Even if the asynchronous FIFO slightly slows the computational module, due to the 6 inverter delay handshake, this slowing would be preferred to the tens of clock cycle delay of the interconnect.

- **Tolerate voltage and temperature variations**: The operating environment has a significant impact on the operation of integrated circuits. The asynchronous FIFO has been shown to perform well under significant environmental variation while maintaining data integrity.

- **Provide a means to deal with the multiple clock domain problem**: In a world of scaling devices, including multiple processor SoC designs, the likelihood of devices operating at different frequencies, or areas operating on different phases of a clock are high. The asynchronous distributed FIFO would provide a method to quickly and reliably transfer information between clock domains without concern for the frequency or phase discrepancy between the the domains.

## 6.3 Future Work

The use of asynchronous FIFOs have the most promise in the design of GALS systems. With the increasing problems of global clock distribution, researchers are turning to asynchronous design (or at least borrowing ideas from asynchronous design) to help ease the timing problems that arise in multi-gigahertz digital designs. It also might be of interest to explore the need of global interconnects schemes in the sub-threshold design space. Operating in sub-threshold greatly exacerbates the problems with interconnect delay (and size) against the drive strength of logic blocks. As with all research, there are still unanswered questions to further explore. These questions include:

- Can asynchronous design compete with the well established synchronous methodologies? For the last 40 years, there has been significant development of CAD tools to greatly simplify the design process for synchronous problems. If a sufficient amount of effort were spent on automating asynchronous design, would the result be competitive with the current synchronous designs?

- Investigate the uses of asynchronous communication modules for seamless MCD integration. Is asynchronous a good solution to the problems communicating between multiple clock domains?

- What are the area/power/performance trade-offs between asynchronous and synchronous design? If asynchronous is deemed better by the area/power/preformance metric, does it warrant the additional design effort?

## 6.4 Concluding Remarks

In general, the goals of this research have been met. The asynchronous FIFO operates in the gigahertz range despite large, expected swings of environmental variations in temperature and power supply voltage. Because of its loosely coupled, self-timed nature, the asynchronous FIFO is better adapted to the variation in the ratio between device parameters as technology scales, requiring less redesign than comparable synchronous designs. However, where this design looses to synchronous models or buffer insertion is in terms of power consumption and design-time. The most promising use of an asynchronous FIFO is in GALS architectures, where the asynchronous FIFO is used as a global communication channel instead of synchronous communication channels.

# BIBLIOGRAPHY

[1] K. Banerjee, A. Mehrotra, A. Sangiovanni-Vincentelli and H. Chenming, "On Thermal Effects in Deep Sub-Micron VLSI Interconnects," *Proc. Design Automation Conference,* 21-25 June 1999, pp. 885-891.

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

[3] R. Bashirullah, W. Liu, and R. K. Cavin III, "Current-Mode Signaling in Deep Submicrometer Global Interconnects," *IEEE Trans. on VLSI Systems,* Vol. 11, Issue 3, June 2003, pp. 406-417.

[4] P. A. Beerel, "Asynchronous Circuits: An Increasingly Practical Design Solution," *Proc. International Symposium on Quality Electronic Design,* 18-21 March 2002, pp. 367-372.

[5] V. Beiu, et al., "The Vanishing Majority Gate: Trading Power and Speed for Reliability," *ASAP.* Samos, Greece, July 23-25, 2005.

[6] L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm," *IEEE Computer,* Jan. 2002, pp. 70-78.

[7] L. P. Carloni and A. L. Sangiovani-Vincentelli, "Coping with Latency in SoC Design," *IEEE Micro,* October 2002, pp. 24-35.

[8] K. Chang et al., "Clocking and Circuit Design for a Parallel I/O on a First-Generation CELL Processor," *ISSCC Technical Digest,* Feb. 2005.

[9] D. M. Chapiro. *Globally-Aynchronous Locally-Synchronous Systems.* PhD thesis, Standford University, October 1984.

[10] B. Curran, et al., “A 1.1 GHz First 64B Generation Z900 Microprocessor,” *ISSCC Technical Digest,* February 2001, pp 238-239.

[11] V. V. Deodhar and J. A. Davis, “Voltage Scaling and Repeater Insertion for High-Throughput Low-Power Interconnects,” *Proc. ISCS,* Vol. 5, May 25-28, 2003, pp. 349-352.

[12] V. V. Deodhar and J. A. Davis, “Optimization of Throughput Performance for Low-Power VLSI Interconnects,” *IEEE Trans. on VLSI Systems,* March 2005, Vol. 13, Num. 3, pp. 308-318.

[13] C. T. Gray, W. Liu and R. K. Calvin, III, *Wave Pipelining: Theory and CMOS Implementation*, United States: Kluwer Academic Publishers, 1994.

[14] R. L. Gordon, ”The Application of Token Rings to Local Networks of Personal Computers,” *Symp on Small Systems, Proc. SIGSMALL Symp and SIGPC Symp on Small Systems,* Palo Alto, CA, 1980, pp. 21 - 22.

[15] S. Hauck, “Asynchronous design methodologies: an overview,” *Proc. IEEE,* Vol. 83, Issue 1, Jan. 1995, pp. 69-93.

[16] *International Technology Roadmap for Semiconductors,* ITRS, 2004. Available: http://public.itrs.net/

[17] A. Iyer and D. Marculescu, “Power and Performance Evaluation of Globally Asynchronous Locally Synchronous Processors,” *Proc. International Symposium on Computer Architecture,* May 25-29, 2002, pp. 158-168.

[18] D. A. Johns and K. Martin, *Analog Integrated Circuit Design*, United States: John Wiley & Sons, Inc., 1997.

[19] H. Kaul and D. Sylvester, "Low-Power On-Chip Communication Based on Transition-Aware Global Signaling (TAGS)," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* Vol. 12, Num. 5, May 2004, pp. 464-476.

[20] K. Lahiri, A. Raghunathan and S. Dey, "Evaluation of the Traffic-Performance Characteristics of System-on-Chip Communication Architectures," *Fourteenth International Conference on VLSI Design,* January 3-7, 2001, pp. 29-35.

[21] J. G. Lee, et. al., "Handshake-Wave Combined Approach with Runtime Reconfiguration for Designing a Low Latency Asynchronous FIFO," *IEEE Asia-Pacific Conference on Advanced System Integrated Circuits,* 4-5 Aug 2004, pp.188-191.

[22] J. J. Little and J. Kam, "A Smart Buffer for Tracking Using Motion Data," *Proceedings from 1993 Computer Architectures for Machine Perception,* 15-17 Dec. 1993, pp. 257-266.

[23] E. Malley, A. Salinas, K. Ismail and L. Pileggi, "Power Comparison of Throughput Optimized IC Busses," *Proceedings of the IEEE Computer Society Annual Symposium on VLSI,* 20-21 Feb. 2003, pp. 35-44.

[24] G. Moore, "Cramming more Components onto Integrated Circuits," *Electronics,* Vol. 38, Num. 8, April 19, 1965.

[25] M. L. Mui, K. Banerjee and A. Mehrotra, "Global Interconnect Optimization Scheme for Nanometer Scale Dissipation," *IEEE Transactions on Electron Devices,* Vol. 51, February 2004, pp. 195-203.

[26] A. V. Mule, E. N. Glytsis, T. K. Gaylord and J. D. Meindl, " Electrical and Optical Clock Distribution Networks for Gigascale Microprocessors," *IEEE Trans. on VLSI Systems,* Vol. 10, Num. 5, Oct 2002, pp 582-594.

[27] C. J. Myers, *Asynchronous Circuit Design*, New York, NY, John Wiley & Sons, Inc., 2001.

[28] A. Nalamalpu and W. Burleson, “A Practical Approach to DSM Repeater Insertion: Satisfying Delay Constraints While Minimizing Area and Power,” *Proc. International Conference on ASIC/SoC,* September 12-15, 2001, pp. 152-156.

[29] J. Nyathi and J. G. Delgado-Frias, “A Hybrid Wave-Pipelined Network Router,” *IEEE Transactions on Circuits and Systems: Fundamental Theory and Applications,* Vol. 49, Num. 12, December 2002, pp. 1764-1772.

[30] S. Parikh and T. E. Martinez, “Dual Processors, Hyper-Threading Technology and Multi Core Systems,” *Intel Corp,* http://www.intel.com/cd/ids/developer/asmo-na/eng/200677.htm.

[31] J. Rabey, A. Chandrakasan and B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, Upper Saddle River, NJ: Prentice Hall, 2003.

[32] V. Raghunathan, M. B. Srivastava and R. K. Gupta, “A Survey of Techniques for Energy Efficient On-Chip Communication,” *Proc Design Automation Conference,* June 2-6, 2003, pp. 900-905.

[33] M. H. Rachid, Microelectronic Circuits: Analysis and Design, Boston, MA: PWS Publishing Company, 1999.

[34] P. J. Restle, T. G. McNamara, et. al, “A Clock Distribution Network for Microprocessors,” *IEEE Journal of Solid-State Circuits,* Vol. 36, Num. 5, May 2001, pp. 792-799.

[35] R. Rydberg, J. Nyathi and J. G. Delgado-Frias, “A Distributed FIFO Scheme for System on Chip Inter-Component Communication,” *Proc. International Conference on VLSI,* Las Vegas, NV, USA, June 2004, pp. 536-540.

[36] R. Seigmund and D. Muller, “Efficient Modeling and Synthesis of On-Chip Communication Protocols for Network-On-Chip Design,” *Proc. International Symposium on Circuits and Systems,* Vol 5, May 25-28, 2003, pp. 81-84.

[37] H. Soeleman, K. Roy and B. C. Paul, "Robust Subthreshold Logic for Ultra-Low Power Operation," *IEEE Trans. VLSI Systems,* Vol. 9, Num. 1, February 2001, pp. 90-99.

[38] I. Sutherland and S. Fairbanks, "GasP: A Minimal FIFO Control," *Proc. of International Symposium on Advanced Research in Asynchronous Circuits and Systems,* 2001, pp. 46-53.

[39] I. E. Sutherland and J. K. Lexau, "Designing Fast Asynchronous Circuits," *Proc. International Symposium on Asynchronous Circuits and Systems,* 11-14 March 2001, pp. 184-193.

[40] S. Tatapudi and J. Delgado-Frias, "A High Performance Hybrid-Wave Pipelined Multiplier," *IEEE Computer Society Annual Symposium on VLSI,* May 2005.

[41] N. H. E. Weste and K. Eshraghan, *Principles of CMOS VLSI Design: A Systems Perspective*, Reading, MA: Addison Wesley Longman, 1993.

[42] A. Youssef, M. Anis and M. Elmasry, "POMR: A Power-Aware Interconnect Optimization Methodology," *IEEE Trans. on VLSI Systems,* March 2005, Vol. 13, Num. 3, pp. 297-307.