ASYNCHRONOUS DESIGN IN DYNAMIC CMOS

by

Jamil Ahmed

A thesis submitted in conformity with the requirements
for the degree of Masters of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

© Copyright by Jamil Ahmed 1998

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-34126-7

ASYNCHRONOUS DESIGN IN DYNAMIC CMOS

Masters of Applied Science 1998
Jamil Ahmed
Department of Electrical and Computer Engineering
University of Toronto

ABSTRACT

This thesis proposes techniques that allow dynamic logic to be used in an asynchronous environment. In particular, the proposed techniques allow pipelined asynchronous systems to be built. The proposed circuit elements are variations on TSPC logic using Sutherland's micropipeline structure combined with dual rail logic to detect operation completion. The use of delay elements to provide circuit completion information is also examined. The resulting cell occupies more area than a TSPC cell, but has higher functionality because of the built-in data flow control mechanisms provided by the handshake signaling of the micropipeline structure.

For each of the techniques, a simple FIFO circuit and a more complex full adder circuit are constructed. Simulation results for each of the techniques are provided along with a discussion of the disadvantages and advantages of each.

ACKNOWLEDGMENTS

This work would not be possible if it were not for the help of many people. I would like to thank my supervisor, Prof. S. G. Zaky, for all the help and encouragement he has given me while working on this thesis. Also, his help throughout the rest of my studies here at U of T is greatly appreciated.

The ideas in the thesis were sparked by an interest in asynchronous design and dynamic logic systems. I would like to thank John Chappel and Prof. Zaky for introducing me to the interesting world of asynchronous design. One of the most rewarding experiences I have had here at U of T was taking the VLSI systems design course taught by Prof. D. M. Lewis. I would like thank Prof. Lewis for introducing me to the world of dynamic TSPC and high speed VLSI systems. The work in this thesis was only possible after so many hours spent finishing our DDFS project. :)

I would now like to take the opportunity to thank my partners involved in the DDFS project, Aris Balatsos and Silas Li. With their help and good humor I have learned a great deal and also have had lots fun along the way. I would also like to thank everyone in EECG and VRG for making my stay here so enjoyable.

Of course, I owe a great deal to my parents and family for the constant support and encouragement they given me while working on this thesis, and also during the rest of my studies.

TABLE OF CONTENTS

ABSTRACT.....	ii
ACKNOWLEDGMENTS.....	iii
TABLE OF CONTENTS.....	iv
LIST OF FIGURES.....	vii
 CHAPTER 1	
INTRODUCTION.....	1
1.1 Asynchronous vs. Synchronous Design.....	4
1.1.1 The Global Clock Signal.....	4
1.1.2 Power and Current Considerations	5
1.1.3 Performance Issues	6
1.1.4 Metastability - Interface to the Outside World	6
1.1.5 Design Complexity and CAD Tools	7
1.1.6 Conclusions.....	8
1.2 Summary of Design Techniques.....	9
1.3 Thesis Overview	9

CHAPTER 2	
BACKGROUND.....	11
2.1 Asynchronous Circuit Classes	12
2.1.1 Bounded-Delay Models	12
2.1.2 Delay-Insensitive Circuits.....	13
2.1.3 Speed-independent and Quasi-Delay Insensitive Circuits	14
2.2 General Background	15
2.2.1 Dynamic Logic - True Single Phase Clocked Logic	16
2.2.1.1 Basic Function Evaluation Cells.....	17
2.2.1.2 TSPC Latches.....	18
2.2.1.3 System Integration	19
2.2.1.4 Modification of TSPC P-Stage	20
2.2.2 Completion Detection	22
2.2.2.1 Dual-Rail Logic	22
2.2.2.2 Binary Decision Diagrams.....	24
2.2.2.3 Using Delays Elements In Bounded-Delay Circuits.....	25
2.2.3 Micropipelines	27
2.2.3.1 C-elements	27
2.2.3.2 Operation of the Sutherland Micropipeline	29
2.3 Current Asynchronous Pipeline Implementations	30
 CHAPTER 3	
DESIGN DETAILS	33
3.1 Design Overview	33
3.2 Method 1 - Basic Pipeline With One Computation per Cycle.....	35
3.2.1 Basic Cell Structure	35
3.2.2 System Level Integration	38
3.2.2.1 System Structure	38
3.2.2.2 System Initialization	40
3.2.2.3 Accepting New Input Data.....	41
3.2.2.4 Releasing Output Data.....	43
3.2.3 Extending the Basic Pipeline For Use in More Complex Systems.....	44
3.2.4 The C-element as a Clock Buffer and Multiple-Input C-elements	45
3.3 Method 2 - Enhanced Pipeline with Two Computations per Cycle.....	46

3.3.1 Basic Cell	47
3.3.2 System Integration	49
3.3.2.1 System Structure	50
3.3.2.2 System Initialization	51
3.3.2.3 Accepting New Input Data.....	51
3.3.2.4 Releasing Output Data	52
3.3.3 More Complex Systems	53
3.3.4 The Modified TSPC P-Stage	54
3.4 Method 3 - Using Delay Elements for Completion Detection.....	54
3.4.1 Basic Cell of the Bounded Delay System	54
3.4.2 System Integration - Bounded Delay Model	55
 CHAPTER 4	
IMPLEMENTATION AND TESTING	56
4.1 Basic Pipeline - Method 1	57
4.1.1 Basic FIFO	57
4.1.2 Full Adder	58
4.1.3 Testing and Performance	63
4.2 Enhanced Pipeline - Method 2	64
4.2.1 Implementation of the Basic FIFO Stage and Full Adder	65
4.2.2 Performance of Method 2	65
4.3 Bounded-Delay Technique - Method 3.....	66
4.3.1 Implementation of the Delay Element	66
4.3.2 Basic FIFO Stage and Full Adder Implementations	67
4.3.3 Performance	67
4.4 A Comparison of All the Design Techniques	68
 CHAPTER 5	
CONCLUSION	70
 REFERENCES.....	72

LIST OF FIGURES

FIGURE 1.	Differences between the synchronous and asynchronous systems	2
FIGURE 2.	Different timing models	14
FIGURE 3.	Basic TSPC N-stage and P-stage logic cells	17
FIGURE 4.	The TSPC latch stages - both N and P type.	19
FIGURE 5.	System level design of TSPC circuits	19
FIGURE 6.	Modified P-stage using NMOS transistors in the Logic Tree	21
FIGURE 7.	Completion detection using dual-rail logic	23
FIGURE 8.	Mapping a Logic Function onto a BDD	24
FIGURE 9.	Mapping the BDD onto a transistor implementation	25
FIGURE 10.	Delay Matching Techniques	26
FIGURE 11.	Delay element schematic	26
FIGURE 12.	C-element symbol and schematic	28
FIGURE 13.	The Sutherland Micropipeline	29

FIGURE 14. Basic Asynchronous Pipeline	31
FIGURE 15. TSPC block with completion detection	36
FIGURE 16. Timing Diagram Showing the Operation of a Basic Pipeline Cell	37
FIGURE 17. Integration of basic pipeline stages into a micropipeline structure.....	39
FIGURE 18. Initial Conditions of Micropipeline Structure.....	41
FIGURE 19. State of pipeline after first phase of operation	42
FIGURE 20. Operation of the micropipeline as it releases output data	43
FIGURE 21. The configurations for multiple input C-elements	45
FIGURE 22. Modified P-stage with completion detection.	47
FIGURE 23. Basic pipeline stage built using both N-stage and P-stage blocks	48
FIGURE 24. Micropipeline with two independent complete signals	50
FIGURE 25. Initial Conditions of Pipeline for Method 2.....	51
FIGURE 26. Input of new data into the micropipeline	52
FIGURE 27. Basic Pipeline Stage of the Bounded Delay System.....	55
FIGURE 28. Structure of multiple stage FIFO	58
FIGURE 29. Transistor Implementation of basic FIFO stage.....	59
FIGURE 30. Transistor implementation of SUM operation of full adder stage	61
FIGURE 31. C-element structure for full adder stage.....	62
FIGURE 32. 4-bit asynchronous adder schematic	63
FIGURE 33. Transistor Implementation of Delay Element.....	66

CHAPTER 1

INTRODUCTION

This thesis proposes new circuit elements and techniques that can be used to design pipelined asynchronous circuits. The proposed schemes are modifications on dynamic logic cells that enable them to operate in an asynchronous environment. Previously, most asynchronous design has been built using static logic. Concurrently, most dynamic logic design has been done synchronously. The design, implementation, and performance of the proposed schemes are discussed in the thesis.

Digital systems are classified as being either synchronous or asynchronous based on their synchronization schemes. The general structures of synchronous and asynchronous systems are shown in Figure 1. Synchronous systems use global timing control in the form of a single clock signal distributed to all parts of the system. All data is processed within a clock cycle and transferred between stages only on clock edges. In general, most asynchronous systems use handshaking signals to synchronize data transfers between the different stages of a system. A request (*req*) signal is sent to a stage when the data entering the stage is valid. The stage accepts the data and returns an acknowledge (*ack*) signal back to the sender, indicating the data have been received. The request/acknowledge cycle

repeats as new data become available. Of course there are several variations on the specific protocol that is used in actual implementations.

FIGURE 1. Differences between the synchronous and asynchronous systems

Most of today's digital systems use synchronous design techniques. In Figure 1, the *clock* signal is distributed to both blocks A and B. If there is a wire delay in the *clock* signal between the two blocks then the clock signal will not arrive at both blocks at the same time. This difference in clock arrival times is called the clock skew. As VLSI technologies get faster, system clock frequencies are increasing but the clock skew is not decreasing at the same rate. As this trend continues with faster and larger VLSI systems, the result is that the clock skew keeps increasing relative to the total clock period. Clock skew causes difficulties in clock distribution. In addition, the transistors in many synchronous systems switch at the same time across the whole system. Concurrent switching creates the problem of increased peak current requirements. With larger and faster VLSI chips the power distribution problem becomes non-trivial.

Self-timed asynchronous systems are not subject to the problems of clock skew and high peak currents. There are no globally distributed clocking signals. All of the control is localized through the use of handshake signals between adjacent logic blocks. Power usage is also distributed more evenly throughout the chip as all parts of the chip do not switch at the same time. Distributed switching reduces the peak current requirements.

In synchronous design, dynamic logic techniques are used to increase the operating speeds of digital systems. Circuits using dynamic logic implementations operate at higher speeds

than those using standard static CMOS logic. With dynamic logic, proper timing is essential to ensure the circuit operates correctly. Dynamic CMOS circuits depend on clock signals to repeatedly charge and discharge the capacitance associated with various circuit nodes. Circuit operation consists of alternating precharge and evaluate phases, controlled by a global clock signal.

In the absence of a clock signal, most previous work on asynchronous design used static CMOS circuits. Due to the speed advantages of the dynamic logic, the research in this thesis examines techniques that allow the integration of dynamic logic into an asynchronous framework. Dynamic logic relies on a global clock signal to synchronize incoming data with the logic cell. Also, the precharge and evaluate phases must be controlled properly. Alternatives to the clock signal are necessary to accomplish these tasks in the asynchronous environment.

Pipelining is widely used in the organization of modern digital systems. In such systems, it is possible to achieve higher throughput at the cost of increased latency. For systems with large amounts of data to process, this is an attractive option. The proposed asynchronous techniques can be used to design pipelined systems.

In summary, the techniques developed in this thesis allow the use of dynamic logic in an asynchronous framework to implement pipelined systems. These systems do not have the clock skew and high peak current problems associated with synchronous systems. Asynchronous systems have the potential to operate faster when implemented using dynamic logic cells instead of static CMOS logic, which has been used traditionally.

The next section provides a more detailed comparison between synchronous and asynchronous systems. Advantages and disadvantages of both techniques are compared as they relate to the design of digital VLSI systems. Following that, the rest of the chapter

summarizes the design techniques proposed in the thesis, and as well summarizes the layout for the rest of the thesis.

1.1 Asynchronous vs. Synchronous Design

Although this thesis deals with digital asynchronous systems, most digital systems designed today are synchronous in nature. A short comparison is made between various aspects of synchronous and asynchronous systems. This comparison serves to illustrate the differences between the two approaches and provides motivation for research in asynchronous systems. In the discussion, various points pertinent to the design of digital VLSI systems are considered. Advantages and disadvantages of both the synchronous and asynchronous systems are outlined.

1.1.1 The Global Clock Signal

The fact that synchronous systems have a global clock signal and asynchronous systems do not is the main distinction between the two systems. As mentioned previously, clock skew can be a problem in synchronous systems, however, asynchronous systems are unaffected by clock skew as there is no global clock signal.

In larger dynamic VLSI circuits, the clock signal is expected to drive many precharge and evaluate transistors. The clock driver has to have enough strength to properly drive the gates of the clocked transistors. Meeting these requirement often results in using multiple buffer stages and H-tree clock distribution networks[13]. The cost of using these buffers is increased chip area and increased power consumption. As an example, in a particular synchronous processor, 10% of the area and 40% of the power is taken up by the clock buffer[14]. This problem will only get worse as improvements in technology allow more transistors on a chip operating at higher speeds.

If dynamic logic is clocked asynchronously, the need for a large clock buffer is eliminated. However, asynchronous logic needs additional control circuitry, which takes up different amounts of area and power depending on the application. Similarly, the area and power costs of clock distribution for synchronous VLSI systems vary quite a bit depending on the particular application. So the application will define which synchronization scheme is more efficient with respect to power and area.

1.1.2 Power and Current Considerations

Asynchronous systems have been known to have much lower total power dissipation than synchronous systems. In most synchronous systems, all parts of the system are active even if data is not being processed everywhere. The parts of the system that are active but not doing any useful work use up extra power unnecessarily. In asynchronous systems, only parts of the system that are actually processing data are active. So power is used only where and when useful work is being done. The power usage problem is magnified with the use of dynamic logic where internal nodes go through precharge and evaluate phases even if there is no change to the input data. Thus, on average, the asynchronous system uses much less power than its synchronous counterpart. Some modern synchronous systems deal with this problem by adding extra logic to allow certain parts of the synchronous system to be turned off. Of course the power savings depend on the application, but as much as 80% reduction in power has been demonstrated in asynchronous systems[4]. This type of power reduction is an increasingly important consideration given today's proliferation of portable applications.

In addition to average power requirement, peak current usage requirement can be troublesome. In high speed dynamic systems, such as TSPC[1], the clock switches concurrently across the whole VLSI chip, resulting in high current demands during these switching periods. This problem is aggravated by faster circuits with fast rise/fall times for the clock. Also, large dynamic logic systems have many transistor connected to the clock

and each transistor gate adds extra capacitance onto the clock node. Note that peak current is calculated as:

$$I = V_{dd} * (\text{Total Capacitance}) / (\text{Clock Slope}) [5]. \quad (\text{EQ 1})$$

High peak currents create problems in power distribution. The voltage or IR drop on the power line is directly proportional to the current. So it becomes difficult to get lower IR drop on the power rails. Usually it means making the power rail very wide as well as adding extra decoupling capacitors across the chip.

Asynchronous systems use local handshaking between adjacent blocks. So the switching is localized and thus the current requirements across a VLSI chip are more evenly distributed, reducing the problem of high peak currents and voltage bounce on the power lines.

1.1.3 Performance Issues

The clock speed of a synchronous circuit is determined by the worst-case delay through the circuit. Asynchronous designs work by sensing when the result is complete and latching the results of the computation at that point. In this manner, an asynchronous mechanism is effectively clocking the circuit in terms of the average case delay. However, asynchronous systems need extra control logic which has the potential to reduce performance. Asynchronous operation is advantageous in large systems where the existence of a large clock skew would reduce the performance of a synchronous systems.

1.1.4 Metastability - Interface to the Outside World

When an input to a circuit comes from an outside source, sometimes it is not possible to guarantee that the signal will satisfy the setup and hold times of the input registers. In a synchronous system, metastable behaviour may arise at the input and the circuit may not operate correctly. Asynchronous systems are designed to allow the arrival of inputs at non-

uniform intervals. In a certain class of asynchronous systems the arrival of the data signal is monitored explicitly using transition signaling or extra control wires. For these systems, inputs to the circuit can arrive any time and proper operation can be guaranteed. When the input signal is delayed, for some reason, the input circuitry will wait until valid data has arrived. Circuit correctness is gained at the cost of extra delay in the circuit. The ability to wait for valid data is useful for systems composed of sub-blocks that are spread out across a large chip. In a large synchronous system, a fraction of the clock period is used to guard against metastable behaviour.

1.1.5 Design Complexity and CAD Tools

Design complexity is one of the main disadvantages of asynchronous circuit design. Synchronous systems can be designed much more easily. Hazards occur in systems when a change in the input falsely changes the output intermittently and then settles back to the original value. Hazards can be avoided in synchronous systems. If there is a combinational logic block feeding into a register, then hazards that occur in the combinational logic can be prevented from being latched into the register as long as sufficient time is given for the circuit to settle down before latching the result. However, hazards can cause problems in asynchronous circuits. Since there is no global synchronization signal, events in the circuit are often realized using transition signalling methods, where changes in the output indicate the next phase of operation. A false transition at the output would result in incorrect operation. When designing these asynchronous circuits extra care has to be taken to avoid hazards through the use of extra circuitry and techniques such as redundant cover[6].

In dynamic logic systems, inputs to the current stage only change during the evaluate phase of the previous stage and are held constant during the evaluate phase of the current stage. In a sense, using dynamic logic makes the behaviour of asynchronous circuits

similar to that of synchronous circuits, thus alleviating the problem of hazards. The design of asynchronous systems, unfortunately, still remains complex.

Because of its relative simplicity, synchronous design has been the method of choice for most VLSI systems. As many more people use synchronous techniques, CAD and automation tools have been developed primarily for the design and implementation of synchronous VLSI systems. The lack of CAD tools for asynchronous design has been a disadvantage for designers wishing to use asynchronous approaches to implement VLSI systems. In addition, there are not too many verification tools that are available to the asynchronous designer. This trend, however, is slowly changing as more researchers explore asynchronous design[7].

1.1.6 Conclusions

The performance of large synchronous VLSI systems is limited by clock skew, power distribution, and voltage bounce during switching. These problems are especially prevalent in large systems built using dynamic logic. Asynchronous systems eliminate the clock skew problem by using localized control schemes and the peak current requirements are reduced because switching is distributed in time. In addition, asynchronous systems provide metastability protection and performance based on average case delays instead of worst-case delays. However, the design of asynchronous systems is more complex, and the design cycle is not as well defined. Continued research is required in these areas. The thesis will focus on the use dynamic logic in an asynchronous framework. The new techniques will allow dynamic logic to be used in a system without the classic problems of clock skew, power distribution, and high peak currents that are found in synchronous systems using dynamic logic.

1.2 Summary of Design Techniques.

This subsection will provide a brief overview of the specific logic elements and techniques that were developed during the course of the thesis. The overview summarizes the contributions of the thesis.

The research in the thesis began with a search for a technique that would allow dynamic logic to work in an asynchronous framework. The first step involved modifying a basic dynamic logic cell to work asynchronously. A dynamic logic family, known as True Single Phase Clocking(TSPC)[1] logic, was used as a starting point. Next, the system level operation of the proposed techniques was considered. Sutherland's micropipeline structure was chosen to provide the asynchronous handshaking control between adjacent logic cells. The Sutherland micropipeline[2] is a well known asynchronous method for building pipelined systems[12].

An important modification to the dynamic logic cells involved adding the ability to detect when a logic operation has been completed. Since there is no global synchronization, this completion information is essential in building asynchronous systems. Two methods for deriving the completion signals are used.

Three different variations of the proposed scheme are developed and tested. Each with different advantages and disadvantages. The first technique uses dynamic logic directly within the micropipeline control structure. The next technique enhances the operation of the pipeline. The final technique examines the use of a different method for deriving the completion signal.

1.3 Thesis Overview

An overview for the rest of the thesis is presented in this final section. Background information for the thesis will be presented in Chapter 2. An overview of currently used

techniques used in the design of asynchronous systems is presented, together with the background information necessary for the thesis. This information includes a description of the Sutherland Micropipeline, c-elements, dynamic TSPC, DCVSL trees, BDD algorithms, and delay matching techniques.

The design of the proposed techniques is given in Chapter 3. The discussion includes how TSPC is modified and how it is integrated into the micropipeline structure. The structure of each of the proposed methods is described in Chapter 3 along with details concerning the operation of these schemes.

More detailed implementation information is given next in Chapter 4. The chapter examines some of the details involved in taking the design ideas and implementing transistor level cells in a working system with maximized for performance. FIFO queue and full adder circuits are built to test circuit correctness and performance. Chapter 4 also covers the testing phase of the thesis. The simulation methods are presented, together with the final results for the various configurations.

Finally, Chapter 5 concludes the discussions in the thesis with a summary and evaluation of the techniques developed.

CHAPTER 2

BACKGROUND

Background information necessary to describe the techniques developed in the rest of the thesis is presented in this chapter. There are various types of asynchronous systems currently being researched. The first section examines various timing models that are used to classify these asynchronous systems.

During the development of the techniques proposed in the thesis, various ideas were combined together. The next section provides background information on some of these ideas which have already been developed. Information is presented about dynamic TSPC, C-elements, Dual-Rail Logic, Binary Decision Diagrams, delay matching circuit techniques, delay elements, and the Sutherland Micropipeline.

The final section of this chapter describes briefly how asynchronous pipelines are traditionally constructed, as described in the literature.

2.1 Asynchronous Circuit Classes

There are many asynchronous techniques that have been developed so far. Hauck's overview on asynchronous systems gives an informative introduction to this area[6]. The asynchronous field is varied and the breadth of research is quite large. The discussion in this section will cover a small portion of that research that is related to the thesis.

Asynchronous circuits are classified in the literature under certain categories, defined in terms of the timing models used. When describing the proposed circuit techniques it will be useful to have a background in the accepted and tested methods currently used for the design of asynchronous circuits. As well, the choice of different design methodology has various implications which will be discussed. The categories of asynchronous systems that are presented in the next section include bounded-delay models, delay-insensitive circuits, and speed-independent circuits.

2.1.1 Bounded-Delay Models

The asynchronous bounded-delay model is the one most closely matching the synchronous design philosophy. Bounded-delay models assume that all the delays in the gates and wires are known. The exact delay time might not be known, however, the delay is bounded. In synchronous systems, this bounded nature allows the designer to set the clock period and then run the system at a fixed clock rate.

Asynchronous systems using the bounded-delay model are the least complex asynchronous systems to design. Other than a few modifications, asynchronous circuits can be built in a fashion similar to synchronous ones. Combinational circuits are built exactly the same as in synchronous design. Sequential circuits use fixed delays in the circuit to match the bounded delays of the circuit elements. The more specific circuit types in this category include "Fundamental Mode Huffman Circuits", "Hollaar's extension to Non-Fundamental Mode", and "Burst-Mode Circuits"[8].

The main problem when building circuits using the bounded-delay model is that the design must be based on worst-case conditions. The clock period in synchronous systems is derived in the same manner, based on the longest delay in the circuit path. Hence, the use of bounded-delay models involves a trade-off between design simplicity and reduced performance.

2.1.2 Delay-Insensitive Circuits

Delay-insensitive circuit design is based on the assumption that wire and logic cell delays are unbounded. The timing model is the opposite of the bounded-delay models. No assumptions can be made as to when a circuit has completed computations or when the data signals are valid. These extra restrictions on delay-insensitive circuits lead to additional circuit complexity. Extra circuitry that will enable some form of completion detection are needed to explicitly indicate that a circuit block has finished computation.

Because the wire delays can be unbounded, extra control circuitry are needed to determine when valid data have reached the end of a wire. This is done using some form of transition signaling, which allows the differentiation between a delayed signal and a signal that has the same logic value after successive cycles. Transition signalling can be implemented using an extra control signal for every data signal. A more relaxed constraint bundles the data, with one control signal used for multiple data bits along the same path.

Communication between various delay-insensitive blocks is achieved using handshaking signals. The state of valid data as well as availability of the logic cells for computation is communicated to adjacent blocks using transition signaling and completion signals.

Delay-insensitive models are good for filtering out hazards since it is explicitly known when new valid data is ready. These circuits can be used to build modular systems and also are good for datapath design[8]. Delay-insensitive circuits have the most rigid constraints of all asynchronous design methods. In these circuits correctness is independent of any

delays in the system. This feature can be useful for large VLSI systems that could potentially have long delays between circuit points. Of course, the guarantee of correctness comes at the cost of additional circuit complexity.

2.1.3 Speed-independent and Quasi-Delay Insensitive Circuits

Speed-independent and quasi-delay insensitive circuits are similar to delay-insensitive circuits with relaxed constraints for wire delays. Under the speed-independent timing model, wire delays are assumed to be bounded. In a practical speed-independent system the wire delays would be considered negligible in comparison to delays of logic gates and other components in a circuit. Quasi-delay-insensitive timing models do not assume bounded wire delays but that the difference in the delay between two wires is bounded. The diagram in Figure 2 summarizes the differences between all the timing models. Note that ' B ' represents a bounded delay, while '?' represents an unbounded delay.

FIGURE 2. Different timing models

In a practical circuit, the quasi-delay-insensitive model can be considered when implementing data busses. Bus signals can have large delays from input to output if they

are travelling across a large VLSI chip. However, the relative delays between individual bits on the bus are negligible.

Speed-independent and quasi-delay insensitive models are useful for the design of individual building blocks of a system. These local blocks cover a small area and the largest internal delay is much smaller than gate and component delays. There is no need to have extra circuitry in localized blocks that handle fully delay-insensitive conditions. However, when connecting these local building blocks into a larger system it is then useful to have fully delay-insensitive behaviour across a large VLSI system.

With relaxed constraints on wire delays in localized building blocks and the bounded nature of relative delays on bus signals, much of the extra transition signalling circuitry used in fully delay-insensitive models is not needed. The savings in circuitry cannot be practically realized in all technologies. For example, wire delays in programmable logic chips can be considerable. And there is no guarantee that localized logic blocks will not be spread out over large wire distances. Fortunately, implementations using custom VLSI approaches are suitable for the speed-independent and quasi-delay-insensitive models, because circuit layout is under control of the designer. In this thesis, the speed-independent and quasi-delay-insensitive models are used in the design of the basic building blocks.

2.2 General Background

The ideas presented in the thesis build on and combine several design techniques both from the synchronous and asynchronous areas. The following discussion presents these various techniques. The information in this section will be necessary in describing the design details presented in Chapter 3.

Background information in this section is provided in three key areas. A description of dynamic logic is given in the first section introducing the True Single Phase

Clocking(TSPC) logic family. TSPC logic served as a starting point for the Thesis. Classically TSPC logic design is done within an exclusively synchronous framework. The goal of the thesis involved modifying TSPC such that it could also be used in an asynchronous framework.

The next section will focus on the self-timed components. Dual cascoded voltage switch logic, a form of dual-rail logic will be described along with how the completion signal is obtained. Included in this section is an overview of binary decision diagrams which can be used to minimize the transistor cost of dual-rail logic. Completion detection is essential in the design of delay-insensitive circuits. Delay elements are introduced in the next section along with a description of how they will be used in an asynchronous systems. Delay elements are used in circuits that follow the bounded-delay timing model.

Finally, there will be a section describing the structure of the Sutherland micropipeline including a description of C-elements, the basic building block of the micropipeline. In addition the operation of the micropipeline will also be described. This pipeline provides the basic control structure for techniques proposed in the thesis.

2.2.1 Dynamic Logic - True Single Phase Clocked Logic

Dynamic logic is central to the techniques and circuits developed in the thesis. In this thesis, the True Single Phase Clocked (TSPC)[1] logic family was specifically chosen. There are two main advantages to using TSPC over other dynamic logic families. First, only a single phase of the clock is needed to control the precharge and evaluate phases of the logic, simplifying the control mechanisms that would be necessary in the asynchronous environment. Secondly, latches are an integral part of the TSPC circuit structure. These TSPC latches are useful for designing pipelined circuits.

2.2.1.1 Basic Function Evaluation Cells

Standard CMOS logic design involves using an NMOS transistor network for the pull-down path and a complementary PMOS network for the pull-up path. Dynamic logic eliminates the use of one of these networks and replaces it with a precharge transistor. In order to accommodate the new structure, dynamic logic is forced into a two phase operation. In the first phase an internal node is precharged to a set value and in the subsequent phase an evaluation occurs through the logic network.

TSPC behaves like the dynamic logic mentioned above with a precharge phase and an evaluate phase. There are two specifications of TSPC circuits based on whether or not the logic function tree is composed of NMOS or PMOS transistors. In the thesis, a TSPC cell built using a NMOS tree will be referred to as an N-stage cell. Similarly, a cell using a PMOS tree will be referred to as a P-stage cell. Figure 3 shows an N-stage TSPC cell on the left and a P-stage TSPC cell on the right. Both cells are designed to evaluate some logic function F .

FIGURE 3. Basic TSPC N-stage and P-stage logic cells

In the N-stage, the logic tree F_N is designed such that when function F is true a circuit path will be created through F_N . The behaviour of the P-stage is opposite. In the P-stage, the

logic tree F_P is designed such that when logic function F is false, a circuit path is created through F_P . Essentially F_N and F_P are dual complements of each other. That is, a regular CMOS gate that evaluates logic function F would be constructed using F_P for the pull-up path and F_N for the pull-down path.

The precharge phase of the N-stage occurs when the clock signal is low and the internal node x_{int} is pulled high. The node capacitance of x_{int} is used to store the precharge value. The evaluation phase occurs when the clock signal switches to a high value. If the logic function F_N evaluates to true, then x_{int} is pulled low and *out* becomes high. Otherwise, x_{int} stays high and *out* is set low.

The P-stage works in a similar manner with the polarities reversed. The precharge phase occurs when the clock is high and the internal node y_{int} is set to precharge low. The evaluate phase occurs when clock is low. If F_P evaluates true, then y_{int} is pulled high and *out* becomes low. Otherwise, y_{int} stays low and *out* is set high.

2.2.1.2 TSPC Latches

TSPC latches are essentially the same as the logic cells. The N-latch and P-latch are shown in Figure 4. Each latch is made using two clocked inverter stages. The operation is quite simple. The N-latch, shown on the left will latch new data when the *clk* signal is low. The P-latch, shown on the right, will latch new data when the *clk* signal is high. For each, during the other phase of *clk*, new data is prevented from going through to the output. The simplicity of these latches allows the design of highly pipelined systems with minimal transistor overhead.

FIGURE 4. The TSPC latch stages - both N and P type.

2.2.1.3 System Integration

The design of a TSPC pipelined system involves using N-stages and P-stages in an alternating manner. By always placing an N-stage cell next to only P-stage cells, it is guaranteed that while the N-stage is in the precharge phase all the adjacent stages will be in the evaluate phase. It is a requirement of dynamic logic that adjacent cells are always in opposite phases of operation. Otherwise, if two adjacent cells of the same type were placed one after the other, then data from the first stage would feed through the latches and corrupt operation of the next stage. The basic structure of a TSPC pipeline is shown in Figure 5. Note that logic evaluation cells and latch stages can be used interchangeably as long as they are of the same type.

FIGURE 5. System level design of TSPC circuits

In this pipeline configuration it is important for the clock signal to arrive at all the logic cells at exactly the same time. If there is clock skew in the system then it is possible that adjacent cells will temporarily be in the same phase of operation. Obviously this will cause errors, as mentioned above.

There are various ways to arrange the blocks in the pipeline. One method uses a logic evaluation cell in the N-stage followed by a P-latch. Minimizing the logic in the P-stage generally allows a circuit to run faster. The second method uses both N-stage and P-stage logic evaluation cells. This results in a system with two logic evaluations per clock cycle, but it is generally clocked slower than the previous one due to the logic in the slower P-stage. Recently, a method was developed that replaces the slower PMOS transistors in the P-stage logic tree with NMOS transistors as discussed in the next section.

2.2.1.4 Modification of TSPC P-Stage

In most CMOS processes, PMOS transistor are usually around 2 times slower than the NMOS transistors. In dynamic TSPC, the P-stage used a PMOS tree which became the bottleneck of the system. Subsequent to the development of the original TSPC technique, a new method was proposed that used NMOS transistors for the logic tree of the P-stages[16]. The modified P-stage is shown in Figure 6. One major difference is that the

PMOS tree F_P is replaced with a faster NMOS tree F_N . In addition an extra PMOS transistor, PI , is added to the cell.

FIGURE 6. Modified P-stage using NMOS transistors in the Logic Tree

Simply changing all the PMOS transistors in the logic tree to NMOS transistors results in an inversion of the final output value. This can be rectified by creating the complementary function with the NMOS transistors or by simply inverting the final output with a static inverter.

The purpose of the extra PMOS transistor PI is to help pull node int quickly up to VDD during the evaluate stage when function F_N is true. This is necessary because once the internal node int has reached $VDD/2$, it takes a long time for int to finish charging up to the full value of VDD . This problem at node int is caused by the body effect on transistors in F_N [16]. So transistor PI acts as a positive feedback mechanism that assists in bringing node int quickly up to VDD during the evaluate phase when function F_N is true. If node int could not pull up quickly then the $output$ node would of course have problems pulling low.

2.2.2 Completion Detection

Asynchronous systems do not use any global clock signals to synchronize data processing and transfers across a VLSI system. The processing and transfer of data is done locally in the logic block. Localized control is possible if the state of the logic cell is known. The state being whether or not the logic cell is still processing data or if it has completed its logic evaluation cycle. The state information for asynchronous systems is derived using completion detection techniques.

There are various methods used to derive completion information. The choice used depends on the timing models that are appropriate for the system. More specifically, in the thesis, the two extreme timing models are considered. The first method is based on the delay-insensitive protocol. The other is based on bounded delay models which somewhat mirror the behaviour of synchronous systems.

The first method is based on dual-rail logic. An algorithmic way of reducing the transistor cost of this logic family is presented too. The second technique is based on using delay elements to match delays in the circuit.

2.2.2.1 Dual-Rail Logic

The technique that will be used to derive the completion information needs to integrate well into dynamic logic design. A dual-rail logic structure known as Dual Cascoded Voltage Switch Logic (DCVSL) is chosen for this task[9]. The basic structure is shown in Figure 7. As a modification to dynamic logic, DCVSL has two internal precharge nodes, labeled *out* and $\overline{out}$ in Figure 7. Node *out* is connected to an NMOS transistor network representing a function *F*. Conversely $\overline{out}$ is connected through a NMOS network

representing the inverse function $\bar{F}$. Finally the complete signal is obtained as the NAND function of out and $\overline{out}$.

FIGURE 7. Completion detection using dual-rail logic

The operation of the cell is controlled by the clk signal. When the clk signal is low the DCVSL cell is in the precharge phase and both out and $\overline{out}$ are precharged high. In this state the complete signal is set low. Then when the clk signal goes high the DCVSL cell enters the evaluate phase. Now since F and $\bar{F}$ are complementary functions only one of them will evaluate to true. This results in only one of either out or $\overline{out}$ always being pulled low and as soon as it does the *complete* signal is pulled high. This indicates to the next stage that valid data is ready. The dual-rail signals can be sent across a large VLSI a chip with large wire delays and still behave correctly, exhibiting the properties of delay-insensitive circuits.

In the basic cell, NMOS trees are shown explicitly as F and $\bar{F}$. Practical circuits do not necessarily have to be implemented in this exact manner. In many cases, much of the logic between F and $\bar{F}$ can be shared, minimizing total transistor overhead. The technique used to minimize the transistor cost in these DCVSL circuits is presented next.

2.2.2.2 Binary Decision Diagrams

Binary Decision Diagrams, or BDDs, can be used to map logic onto DCVSL gates[10]. BDDs map a function F and $\bar{F}$ onto a directed graph. The directed graph represents all paths from the ground node of a circuit to both the output nodes F and $\bar{F}$. An example of the carry function of a full adder mapped onto a BDD is shown in Figure 8.

FIGURE 8. Mapping a Logic Function onto a BDD

Each node represents an input variable. At each node, two cases are considered. Namely, what happens when the variable is high and what happens when the variable is low. Based on these cases two edges are drawn from the current node to other nodes. If there is enough information to know the outcome of function F , then the arrows are drawn to the leaf nodes $f=1$ and $f=0$, otherwise edges are drawn to the next variable. The edges based on high values of variables are shown as solid arrows, while the edges based on low values of variables are shown as dashed arrows. In the example of Figure 8, as long as any two variables are high, the path will lead to $f=1$. Otherwise all paths lead to $f=0$.

These diagrams can be mapped directly onto the DCVSL gates by making the node $f=0$ and $f=1$ the two precharge nodes. All the edges with solid lines are mapped to NMOS transistors with the variable as input. All the edges with dashed lines are mapped onto NMOS transistors with the variable complement at the input. An example is shown in Figure 9. A random truth table is mapped onto a BDD, which is then shown mapped as a DCVSL gate. Note that the two PMOS precharge transistors and the NMOS evaluate

transistor connected to ground are implicit in the mapping from BDD to DCVSL. Using BDDs to implement DCVSL reduces the transistor cost of using both functions F and $\bar{F}$ and is always preferable to having two explicit transistor networks for functions F and $\bar{F}$.

FIGURE 9. Mapping the BDD onto a transistor implementation.

2.2.2.3 Using Delays Elements In Bounded-Delay Circuits.

Delay-insensitive circuits have their advantages, however, extra transistors and area are needed for their implementation. Bounded-delay models will reduce the extra circuit overhead at the cost of possible reduced performance. In synchronous systems the period of the clock signal is matched with the largest delay in the circuit path. In asynchronous systems, this idea can be used similarly by using delay matching techniques to derive circuit completion information. A general system setup using delay matching is shown in Figure 10. The *request* signal is asserted when the *data_in* signal at the input to the logic

evaluation cell is valid. The *complete* signal is asserted after the given delay in the delay element, at which point the *data_out* value is valid.

FIGURE 10. Delay Matching Techniques

There are various methods of implementing delay elements in VLSI designs. One approach is shown in Figure 11. The structure is an inverter with a current limiting control transistor. Two inverters are used in series so that both the zero-to-one transition as well as the one-to-zero transition can be delayed using the current limiting transistors.

FIGURE 11. Delay element schematic

The V_{bias} signal is used to adjust the delay to a range of values. The maximum value of V_{bias} of VDD would result in the smallest delay. The delay could then be increased by adjusting V_{bias} down. Note that delay elements without a V_{bias} control and a fixed delay can also be used. However, the actual delay might have to be overcompensated to ensure proper circuit operation. With the use of V_{bias} and adjustable delays a more aggressive delay matching approach may be taken. The circuit could be designed to work as fast as

possible, but if for some reason extra delays arise in the logic computation path, then V_{bias} can always be adjusted and the circuit will operate correctly. This is similar to the situation in synchronous systems where, if the original clock speed is too fast for the circuit to operate correctly, the clock period can be adjusted down to accommodate and assure correct circuit operation.

2.2.3 Micropipelines

Micropipelines are asynchronous versions of synchronous pipelines. Input data to the pipeline may arrive in a non-uniform manner. Data will be allowed into the pipeline if the pipeline is not full and the data will eventually be sent through to the output if the output stage is ready to accept the data. If either the input or output stages of the pipeline are not ready, then the corresponding transfer process will block. The transfer of data will resume when the busy elements become available.

In a pipelined synchronous circuit each pipeline stage performs some type of logic operation on the input data. Similarly, each stage of the micropipeline can also be used to implement a logic operation. The structure of the micropipeline is still essentially the same except that now, along with handshaking control signals, there is data moving through the pipeline and having logical operation performed on them. With the ability to perform operations on the data as it moves along the asynchronous micropipeline, it is possible to create circuits with asynchronous datapaths. The Sutherland micropipeline structure was chosen as a framework for the asynchronous techniques developed in the thesis. The asynchronous pipeline is functionally similar to the synchronous pipeline except that the global synchronization is replaced with localized control.

2.2.3.1 C-elements

C-elements were developed by Muller as a method of synchronizing signals arriving at non-uniform intervals. The ability to synchronize generally asynchronous signals made

the C-element widely popular in many asynchronous schemes, including the micropipeline. In Figure 12, the C-element symbol, marked with the letter 'C', is shown on the left with the schematic of the transistor implementation shown on the right. The implementation shown is a dynamic one and the internal capacitance at node *X* is used to store state information for the C-element. The weak feedback inverter prevents charge leakage from reducing the voltage of node *X*, and incorrectly changing the value of node *OUT*.

FIGURE 12. C-element symbol and schematic

The operation of the C-element is quite simple. The output of the C-element will only change once both inputs have converged to the same value. Otherwise it will keep its current value. The structure of the C-element can be expanded to accommodate multiple inputs also. More information on multiple input C-elements are discussed in later sections of the thesis.

2.2.3.2 Operation of the Sutherland Micropipeline

Sutherland developed the micropipeline structure as a way of building asynchronous pipelined systems. A detailed description of the micropipeline and auxiliary circuit components is presented in Sutherland's paper[2]. Only the high-level handshaking mechanism was used for the thesis.

The basic control structure, which is built only using C-elements, is shown in Figure 13. Each C-element has inputs signals originating from both its neighbours and sends out its output signal to both its neighbours. The signals from a specific C-element to an adjacent C-element on the left is inverted before entering the destination C-element. The bubbles on the inputs of the C-elements are used to indicate these inversions. With additional logic, the micropipeline can be used to process data. Additional information about data processing is provided in a later section.

FIGURE 13. The Sutherland Micropipeline

Each C-element can be considered a single stage in the micropipeline. The operation of the micropipeline is based on adjacent stages communicating with each other through the use of a handshaking protocol.

Considered the initial state of the system with all the outputs of the C-elements and the *Rin* (input request) and *Aout* (output acknowledge) inputs set low. Setting *Rin* high forces the output of C-element C_1 high. This event will in sequence cause the output of C_2 high, followed by C_3 and finally C_4 set high. Eventually the signal *Rout* becomes set high. At the output side of the micropipeline, the outside world acknowledge receipt of the *Rout* signal by asserting *Aout* high. Similarly at the input end, once *Rin* has gone high, the micropipeline sets *Ain* high indicating it received the original *Rin*. The input side would then set *Rin* back low and the handshake cycle would repeat.

The operation of the micropipeline can be summarized by saying that in each stage a C-element copies the state of the previous C-element only when the states of the previous and next C-elements differ[2].

2.3 Current Asynchronous Pipeline Implementations

There has been quite a bit of discussion about asynchronous pipelines in area the of asynchronous design. This section describes briefly the operation of a basic asynchronous pipeline. Most asynchronous pipelines currently designed use Sutherlands's micropipeline as the underlying control structure.

The operation of the micropipeline has only been described in terms of its control signals. As mentioned previously, micropipelines can be used to process data as data flows though

the pipeline, enabling the design of asynchronous datapaths. The structure of the basic pipeline is shown in Figure 14.

FIGURE 14. Basic Asynchronous Pipeline

Note that the control structure shown in the pipeline is the same as the one described previously. Now a combination logic block has been added within the pipeline along with a delay element that matches the delay through the combinational block. Instead of the delay element, completion circuitry is sometimes used. Between each combinational block, there is a register that is controlled by the output of the C-element. Note that the C-element output flows through the register block. Ideally, the signal is delayed for as long as it takes the register to lock onto the input data. As is apparent, the structure shown is quite similar to synchronous pipelines which have combinational stages separated by registers. The difference of course is that the registers in the synchronous pipeline are controlled by a global clock, while the registers in the asynchronous pipeline are controlled by the micropipeline signalling protocol.

In the thesis, these combinational blocks are replaced with dynamic logic cells. There has been work done previously combining dynamic logic into an asynchronous

framework[18]. However, the system that was designed was useful only to implement FIFO structures. The ability to perform logic computations is not considered in that work.

In addition, instead of using the registers to only separate pipeline stages, the thesis examines how the register stage can also be used to perform logic operations. The use of completion detection instead of delay elements is also examined in the thesis.

All the background information necessary for the rest of the thesis has now been presented. The next chapter will bring these concepts together and develop the design ideas that define the techniques proposed in the thesis.

CHAPTER 3

DESIGN DETAILS

The design information for the basic cells and techniques that are developed for the thesis are presented in this chapter. The general schemes are discussed in this chapter while the details pertaining to particular test circuit implementations are deferred to Chapter 4. The first section provides a brief overview of all the techniques proposed in the thesis. The subsequent sections then provide more detailed descriptions for each technique.

3.1 Design Overview

The goal of the thesis was to develop techniques that enable dynamic logic to be used in an asynchronous pipelined systems. For each proposed technique, a new basic pipeline stage is defined, and the integration of the basic cell into a full system is discussed.

The basic logic cell is a modified version of the dynamic logic family True Single Phase Clocking (TSPC). TSPC was chosen because of the simplicity of requiring only one clocking signal to control the precharge and evaluate transistors. Also, the TSPC logic family provides latches that make it possible to implement the pipelined circuits easily. Classically, these TSPC circuits have been used exclusively in the synchronous framework

controlled using a single global clock signal. The clock signal is responsible for synchronization of data between TSPC cells and concurrently controls the precharge and evaluate phases of the TSPC cells.

In asynchronous systems, there is no longer a global clock signal that can be used for synchronization control. Each basic cell needs a way to indicate when processing of input data has finished. This is accomplished through the use of completion signals which are derived in one of two ways. Dual-Rail Logic is used in some of the new circuit elements to derive completion information for the logic block. The other method for deriving completion signals involved using a delay matching mechanism, where the computation time of the logic cell is matched with a delay before a completion signal is asserted.

At a higher system level, the Sutherland micropipeline structure is used to provide the asynchronous handshaking control between adjacent logic cells. In this structure, handshake signals replace the global clock of a synchronous system.

Three techniques that integrate the dynamic logic cells into an asynchronous framework are presented. The first technique is a modified version of the original Sutherland micropipeline with one logic evaluation in each basic cell every cycle. Here the cycle time refers to the time for the dynamic logic cell to go through a complete precharge and evaluate phase. This is analogous to the clock cycle in a synchronous system. In the next technique, the structure is modified to allow one computation every half cycle or two logic evaluations per cycle in each basic cell. These first two versions use DCVSL logic to derive the completion information.

Method 3 uses delay matching to derive the completion signals. The use of delay elements allows a reduction in the overhead needed in the DCVSL approach. Method 3 most resembles the synchronous timing model of using bounded delays.

3.2 Method 1 - Basic Pipeline With One Computation per Cycle

Method 1 is similar to the basic structure of the existing asynchronous micropipeline with the modification of using dynamic logic. It is necessary to integrate the TSPC cells within the micropipeline structure such that the precharge and evaluate phases of the dynamic logic function correctly. Since there is no clock, the precharge and evaluate phases must be controlled using local signals derived from the micropipeline protocol.

3.2.1 Basic Cell Structure

Each pipeline stage in the original micropipeline has a combinational block for logic evaluation and a register to store the results of the evaluation. A description of the original micropipeline structure was provided at the end of Chapter 2. The new basic cells that are developed for the proposed techniques preserve the external behaviour of the original micropipeline stages.

The new basic cells are based on a pipeline stage of synchronous TSPC. Recall that synchronous TSPC cells consists of one N-stage and one P-stage. One common configuration when designing synchronous pipelined systems is to use only the N-stage for logic evaluation and the P-stage as a latch. This configuration results in one logic evaluation per clock cycle and is often used since logic evaluation in the P-stages is slow relative to the N-stages. In addition note that the original micropipeline structure also has only one logic evaluation per stage followed by a latch stage. For these reasons the basic cell of the first technique performs one logic evaluation per cycle.

The TSPC N-stage was modified so that it can be used in an asynchronous framework. This modification involves using dual-rail logic evaluation with completion detection built into the basic cell. The dual-rail logic is referred to in the literature as Dual Cascoded Voltage Switch Logic (DCVSL). The completion signal is derived from the dual-rail

signals through a NAND gate. The P-latch stage remains the same as in the original TSPC implementation.

Another property of the new cell involves using localized clocking. A C-element is used to drive the transistors that were originally driven by a global clock signal. The C-element is used in each basic cell to synchronize the handshaking signals arriving from adjacent stages. The new basic cell is shown in Figure 15. This cell constitutes a single pipeline stage in the new micropipeline structure of the first proposed technique.

FIGURE 15. TSPC block with completion detection

Similar to TSPC logic the new cell in Figure 15 has two phases of operation, where the phase is determined by the value of the local *ctrl* signal which is derived from the C-element. More detailed information about system level handshaking control is provided in a later section. For now it is sufficient to know that the *ctrl* signal is set low when either

the input or output to the cell is waiting for some event, and *ctrl* is high once the input data is ready and the output stage is ready to accept new data. Note that both the N-logic and P-latch of the cell are controlled by the same *ctrl* signal.

The operation of the basic cell is quite simple. When the C-element is waiting for the proper handshaking signals, *ctrl* is low. In this state, the N-logic is precharging both the *out* and $\overline{out}$ high and the P-latch is latching the value just computed in the N-stage. The *complete* signal is low because both precharge nodes at the input of the NAND gate are precharged high. Thus when the output of the C-element is low, eventually the *complete* signal also becomes low. The timing diagram of this operation is shown in Figure 16.

FIGURE 16. Timing Diagram Showing the Operation of a Basic Pipeline Cell

Once the C-element is activated by the handshaking signals, the *ctrl* signal becomes high. The high value of *ctrl* indicates to the logic stage that the input data is valid. The evaluation phase of the N-logic is started while the P-latch shuts off. As evaluation takes place, only one of the internal precharge nodes, *out* or $\overline{out}$, will evaluate to low and set the *complete* signal high. Again note that the value of the *complete* signal follows the value of

the *ctrl* signal. However, the value of *complete* changes only after the logic evaluation or precharge in the N-stage has completed.

The *complete* signal is generated from the dual-rail signals in the N-stage. However, the *complete* signal is used to indicate the status for the entire cell which includes completion of the P-latch stage. In other words, this cell is not delay-insensitive between the N-stage and P-stage. For practical circuits the time necessary to complete the latch operation is much less than the time a logic evaluation and completion detection takes. Also note that the final half latch of the N-stage completes in parallel with the operation of the NAND gate to determine the completion signal. For this basic cell the quasi-delay-insensitive model best describes the timing within the cell where actual delays may be unbounded but the relative delays between two components within the cell are bounded.

3.2.2 System Level Integration

As mentioned before, the system level control of the circuits developed in the thesis is provided using the Sutherland micropipeline. Sutherland's idea was to insert logic computation blocks within an asynchronous FIFO structure, thus accommodating asynchronous data path circuits. A modification on the approach is to use dynamic logic to implement the computational blocks. The dynamic pipeline stage, described in the previous section, is integrated into the micropipeline structure resulting in a methodology to design asynchronous systems and circuits with dynamic logic implementations.

3.2.2.1 System Structure

The basic pipeline stage of Figure 15 consists of a N-stage, P-latch, C-element and NAND gate. A micropipeline structure is formed by connecting the C-elements of each of the stages in the configuration shown in Figure 17. The block *basic cell*, which contains the rest of the circuit elements constitutes the data path section of each pipeline stage.

One input to the basic cell is the *ctrl* signal originating from the C-element along with new input data from the previous stage. The output of each cell is the completion signal and the output data. The data moves from left to right through the pipeline. The inputs to each C-element are the *complete* signals from the adjacent basic cells.

Also, the complete signal of the input stage becomes the input acknowledge(*in_ack*) and that of the output stage becomes the output request(*out_req*). The structure shown in Figure 17 is the general case for a simple circuit and will be useful in describing the operation. More complex circuits are examined in a later section.

FIGURE 17. Integration of basic pipeline stages into a micropipeline structure.

The global handshaking signals are labeled on the input and output sides of the micropipeline shown in Figure 17. On the input side, the *in_req* signal is used to present new data to the pipeline, and *in_ack* acknowledges that data has been received. On the output side, *out_req* signals that output data is ready and acts as a request to the next circuit stage to take the data. The signal *out_ack* is returned by the next circuit stage to indicate that the data has been accepted.

Local handshaking signals between successive stages within the micropipeline behave like their global counterparts. As mentioned before, the inputs of the C-element of a given stage are the output completion signal from adjacent stages. Completion signals from a subsequent stage behave like the *out_ack* signal. Similarly, the completion signals from preceding stages behave like the *in_req* signal.

There is one minor difference between the original micropipeline described earlier in Chapter 2 and the micropipeline implemented in the thesis. The difference is in the polarity of the signals. The basic cell consists of a N-stage and a P-latch with the completion signal output from within the N-stage. The complete signal goes high once the data has been processed in the N-stage. However, the data is actually ready to be used by a subsequent stage only after it has also been latched into the P-latch. So a low value of the *complete* signal indicates that there is valid data at the output of the P-latch. For this reason, the polarity of all the signals in the micropipeline are reversed and all the handshaking control signals are actually active low. The change does not affect the functionality of the pipeline. The main reason for the change was to keep the interaction between signal *ctrl* and *complete* consistent with the original micropipeline.

There is one more point to mention before describing the operation of the micropipeline. The *in_req* and *out_req* signals are shown as single bit signals to simplify the explanation. In real circuits these request signals can be several bits wide. Multiple bits are necessary if the input data is coming from different sources that may have unmatched delays. Similarly the output signals might also be sent to different sources. In such cases, multiple input C-elements would be needed.

3.2.2.2 System Initialization

In the steady state of an idle pipeline, successive stages will be alternately in the precharge and evaluate phases of their operation. The output value of *ctrl* from the C-elements are

therefore alternating low and high for the respective pipeline stages. In the steady state, this results in the complete signals of the corresponding stages to also have low and high values, respectively. The *in_req* signal is high, indicating that there is no data to process and the *out_ack* is also high to indicate the subsequent stage is not ready to accept more data. These initial conditions are labeled in Figure 18.

FIGURE 18. Initial Conditions of Micropipeline Structure

3.2.2.3 Accepting New Input Data

This section examines what happens as new data becomes available at the input to the system that is in the initial condition described above. Starting with the initial state, the pipeline will start operation as new data becomes available at the input. Note that the *complete1* signal of the first stage is low indicating a request for new data. Valid data at the input is signalled when *in_req* is asserted low causing the *ctrl1* signal of the first stage to go high, putting that stage into the evaluate phase. Once the evaluation has taken place the *complete1* signal of the first stage goes high leading to the next phase of operation.

Once the *complete1* signal goes high, it triggers new events on the pipeline. The diagram in Figure 19 shows final resting state of the pipeline. Note that *ctrl2* is now low and subsequently *complete2* has also gone low. Basic cell 2 is now in the precharge phase while basic cell 3 has gone into the evaluate phase. Since the micropipeline is just being filled with data, the data inside basic cell 2 and 3 are not yet meaningful.

FIGURE 19. State of pipeline after first phase of operation

The *complete1* signal in the first stage is also the *in_ack* signal for the whole pipeline. When *in_ack* has been toggled high the external circuitry needs to respond by toggling *in_req*. This communication is part of the handshaking mechanism and the pipeline only continues operation once the external circuitry has responded. Note that the exact same behaviour is required at the output of the C-element as well.

Once *in_req* is high, the micropipeline enters the next phase of operation. Now the inputs to the C-element of the first stage are both low and *ctrl1* becomes low. Note that *in_req* is inverted before entering the C-element. With the low *ctrl1* signal the N-stage is precharge phase and the P-stage is in the evaluate phase. Since the P-stage consists of just a latch, the value that was evaluated during the last cycle in the N-stage now gets stored into the latch in the P-stage. Once these operations are complete, *complete1* goes low signalling to the next stage that there is new valid data available. In addition, a low *complete1* signal indicates to the previous stage that it may send more data to the micropipeline. The operations discussed so far describe a full cycle of micropipeline operation. The sequence of events described is the same between any two adjacent stages within the micropipeline.

3.2.2.4 Releasing Output Data

The handshake signals continue along until they reach the output. This section examines how the output of the pipeline works. The operation is similar in nature to the input operation. Consider the state of the micropipeline when it is waiting for new data as shown in Figure 20.

FIGURE 20. Operation of the micropipeline as it releases output data

Signal *out_ack* is set high indicating that the external output circuit is busy. In addition the N-stage of the last basic cell has just completed evaluation in the N-stage. Once the external output circuitry is available, *out_ack* is asserted low. Signal *ctrl3* becomes low and the N-stage of the last basic cell enters the precharge phase and the P-stage latches the value that was just computed in the N-stage. Subsequently *complete3* is asserted low, followed by *out_req* pulling low. A low value of *out_req* signals to the external circuitry that the output data of the micropipeline is now ready.

The input and output sides of the micropipeline are independent to a certain extent. Of course, the input data can arrive at non-uniform intervals and the output data can be taken, in a similar manner, at non-uniform intervals. Internally the handshaking takes place and the precharge and evaluate phases occur. However, the throughput of the micropipeline is determined by the slower of the input and output stages. As in a synchronous system, each pipeline stage contain data. So if the external output circuitry can not take the data, it has to be kept in the micropipeline preventing the input from sending more new data into the

pipeline. Similarly, if the input circuitry does not provide data, obviously the output circuitry has to wait.

3.2.3 Extending the Basic Pipeline For Use in More Complex Systems.

In the previous example, there was only one completion signal output from each stage and the *in_req* and *out_req* signals were one bit wide signals. In a general system, there may be multiple completion and request signals active in the micropipeline. For a single pipeline stage, there might be input data that is arriving from two different external logic units. Obviously, in an asynchronous system, the timing between the two external blocks will not necessarily be concurrent. In this case two *in_req* signals would be required and a multiple input C-element would have to be used. Similarly, there might be multiple external output blocks, which would require the use of multiple *out_req* signals.

In general, each input or output bit can have associated with it a separate control signal. In such systems, no assumptions are made about delays between logic cells. This is a delay-insensitive approach which is guaranteed to always be correct independent of delays in the wires, but has the disadvantage of extra circuitry. In many cases the delay-insensitive approach is too rigorous in a practical system. Using a speed-independent or quasi-delay-insensitive model will result in much less overhead in the handshaking circuitry. In particular, the number of inputs entering the C-element can be greatly reduced. C-elements with fewer inputs operate at a much faster speed which in turn allows the whole system to operate faster. The quasi-delay-insensitive approach can be used if there are many bits coming from the same source, so only one *in_req* signal can be used for all of them. Similarly if the data is arriving from an adjacent cell that is very close then the speed-independent model can be used to save extra overhead.

3.2.4 The C-element as a Clock Buffer and Multiple-Input C-elements

As noted earlier, the outputs of the C-elements in each basic cell replace the clock in synchronous TSPC. This enables a very good form of localized clock buffering. The output inverter in the C-element can be appropriately sized to accommodate the load on the clock signal. If necessary, more buffering can be added to a given cell after the C-element without affecting the correctness of the circuit. Local buffering of the clock in a synchronous circuit would introduce unacceptable clock skew unless the delay is matched at every other clock output.

FIGURE 21. The configurations for multiple input C-elements

The C-element introduces a critical delay which determines the maximum operating speed of the circuit. In order to achieve the best performance, it is better to reduce the number of inputs entering the C-element. However, when multiple inputs are necessary, we can use either a tree of 2-input C-elements or a single multi-input element as shown in Figure 21. The tree structure is preferable because for the same latency and drive strength, the layout area of the tree structure is smaller. At the same time, the input gate load for the tree structure can be made smaller when sizing these input transistors, helping speed up the circuit. With the tree structure, the complete signals from adjacent cells have a much smaller gate load that has to be driven.

3.3 Method 2 - Enhanced Pipeline with Two Computations per Cycle

Each stage in the original micropipeline structure can perform only one logic computation per cycle. The technique presented in the section allows each pipeline stage to perform two logic computations per cycle. The ability to perform two logic computations per cycle is a significant modification to the micropipeline structure as all previous implementations of micropipelines only contain one logic evaluation per stage. The ability to double the number of computations is because of the inherent structure of dynamic TSPC logic.

Micropipelines have a storage element in each stage. The technique developed in this section is possible by noting that the storage element can concurrently be used to also perform logic evaluations when TSPC logic is used. One nice property of TSPC is that it is possible to have logic evaluation cells in the P-stage with the latching mechanism built into the logic cell. For each basic cell, now both the N-stage and P-stage contain logic evaluation circuitry. Functionally, the difference between Method 1 and Method 2 is that the number of pipeline stages needed to implement any given asynchronous circuit is cut in half when using Method 2. From a performance perspective, Method 2 has half the latency of Method 1.

In addition, there was an assumption made for the internal timing of Method 1 based on a quasi-delay-insensitive timing model. The total delay of the N-stage and the completion detection circuitry would be slower than the delay through the single P-latch stage. The technique in this section eliminates this assumption as both the N-stage and P-Stage each has its own independent timing and completion circuitry. The timing model of Method 2 is based on a delay-insensitive protocol where the delay through each stage can be unbounded and independent of each other while still maintaining circuit correctness.

In order to develop the new method, initially the P-stage of the basic cell was defined. The new P-stage is basically a dual of the N-stage with built in completion detection circuitry.

The N-stage and new P-stage are then combined into a new basic cell which becomes the building block of the new micropipeline structure. The design and operation of these basic components are presented in the following subsections.

3.3.1 Basic Cell

The basic cell for Method 1 and Method 2 are similar. The N-stage remains the same. However, the TSPC P-latch is modified in the same manner that the original TSPC N-stage was modified. The P-latch is converted into a logic stage which evaluates a given logic function F with built in completion detection. This modified P-stage, shown in Figure 22, is essentially the logic dual of the N-stage shown in Figure 15.,

FIGURE 22. Modified P-stage with completion detection.

There are minor differences between the N-stage and the P-stage. The P-stage evaluates a given logic function F_p , where the transistors in F_p are all PMOS. When using PMOS transistors it is necessary to use the complement function $\overline{F_p}$ to determine the final result of the logic evaluation. As shown in the Figure 22, $\overline{out}$ is connected to the final latch stage.

Like the N-stage, there are two precharge nodes, but in this case, both internal nodes, *out* and $\overline{out}$, will precharge low when the *ctrl* signal is set high. The evaluate phase takes place when *ctrl* becomes low and only one of *out* or $\overline{out}$ is pulled up high through F_p or $\overline{F}_p$, respectively. Once one node has been pulled up, the completion signal is set high indicating completion of the evaluate phase. Note that the NAND gate that was used in the N-stage is replaced with a OR gate in the P-stage.

With both the N-stage and P-stage defined, the new basic pipeline cell for Method 2 can be constructed as shown in Figure 23. In the new implementation, the N-stage and P-stage has their own independent completion circuitry and completion signals *Complete_N* and *Complete_P*, respectively. However, each basic cell still has only one C-element and consequently also one *ctrl* signal. Also output data can be taken from either the output of the N-stage or P-stage.

FIGURE 23. Basic pipeline stage built using both N-stage and P-stage blocks

With one *ctrl* signal, the N-stage and P-stage of a given basic cell are synchronized to each other which means that no extra mechanism is necessary to guarantee that the *internal data* of a basic cell is transmitted correctly between the N-stage and P-stage. Of course

data entering the P-stage from other basic cells will have to have certain conditions met. These conditions are discussed in a later section.

Another change made to the basic cell is that now the C-element does not need to have its inputs from the previous pipeline stage inverted. The extra inversion can be eliminated because the completion signals *Complete_N* and *Complete_P* behave logically opposite of each other. That is when ctrl is high, *Complete_N* eventually goes high and *Complete_P* eventually becomes low. Conversely when ctrl is low, *Complete_N* goes low and *Complete_P* eventually goes high.

From an external point of view, the new basic cell behaves in exactly the same manner as the basic cell used in Method 1 in the case when the computation time for both the N-stage and P-stage is the same. When the computation time differs, the completion signals will be asserted independent of each other. Though the external behaviour is different for the new basic cell, it still integrates well into the micropipeline structure. The next section discusses how the new basic cell is used as a building block to form a larger system.

3.3.2 System Integration

With the new basic cell, additional functionality is introduced into the micropipeline. The main improvement, of course, is that now both the N-stage and P-stage can be used to perform a given logic operation on the input data. The result is that the level of pipelining has basically doubled. Two logic operations can be completed over the course a complete handshake cycle. Except for this additional functionality, the operation of the new micropipeline structure is very similar to the micropipeline of Method 1.

This section describes the structure and operation of the new micropipeline. Initially, just the basic operation of a simple structure is described, followed by a discussion of considerations that need to be made for more complex systems.

3.3.2.1 System Structure

The new micropipeline structure is constructed out of the new basic cells as shown in Figure 24. Instead of forking one completion signal out to both the previous and next stage, there are now two separate completion signals. Signal $complete_N$ indicates the status of the N-stage and is routed to the preceding basic cell. Similarly, signal $complete_P$ indicates the status of the P-stage and is routed to the next cell. Previously the completion signal arriving from a preceding stage had to be inverted before entering the C-element. In the current implementation, this extra inversion can be eliminated since $complete_P$ is already inverted with respect to $complete_N$. In addition, the fact that $complete_P$ is inverted allows the use of active high signals for the handshaking signals. Recall that in Method 1 these signals were active low.

FIGURE 24. Micropipeline with two independent complete signals

The operation of the structure shown in Figure 24 follows quite nicely from the operation of the micropipeline of Method 1. As noted earlier, when the delay in the N-stage matches the delay in the P-stage, then signals $complete_N$ and $complete_P$ are synchronized, just

opposite in polarity. In this ideal case, the operation of the micropipeline is similar to the micropipeline of Method 1. A minor difference being that the input *in_req*, and *out_req* signals have been inverted. We will now examine the case when *complete_N* and *complete_p* change state at different times.

3.3.2.2 System Initialization

The initial conditions for the idle pipeline are similar to that of the micropipeline structure described in the previous section. The initial values of all the signals of the pipeline structure for Method 2 is shown below in Figure 25. Recall that the initial condition of an idle pipeline was to have alternating pipeline stages in the precharge and evaluate phases as indicated by the alternating values of the *ctrl_n* signals. In addition both *in_req* and *out_ack* are also low to indicate that the external circuitry is busy. As indicated before, there are two completion signals for each stage which will have opposite values in steady state conditions.

FIGURE 25. Initial Conditions of Pipeline for Method 2.

3.3.2.3 Accepting New Input Data

Once new data is ready at the *data_in* input, *in_req* is toggled high. From the initial conditions, the change in *in_req* results in a high value for *ctrl1*. Now the first pipeline stage enters a new phase of operation with the N-stage in the precharge phase, and the P-stage in the evaluate phase. The *complete_N* and *complete_P* signals can change

independently and are toggled once their respective stages finishes. Figure 26 shows the steady state values off all the signals in the pipeline once in_req is high.

FIGURE 26. Input of new data into the micropipeline

Two independent operations are now taking place. Once $complete_N$ is high, a high in_ack is sent to the external circuitry at the input. To continue operation the signal in_req needs to be set low. On the other side, once $complete_P$ of the first stage goes low it triggers the second stage to go into its next phase of operation. These two events occur in parallel but independently. Once in_req is set low and the next stage has finished operation, the first stage of the pipeline goes into the next phase of operation.

Signal $ctrl1$ goes low and the N-stage enters the precharge phase and the P-stage does an evaluation operation. Once again, $complete_N$ goes low when the N-stage precharge is done and $complete_P$ is pulled high once the P-stage logic evaluation is done. At this point in_ack is also low. The first stage has now completed its first full cycle of operation and is now ready to accept new data. The operation of the basic cell is similar in all subsequent cells and data eventually gets to the last basic cell in the micropipeline.

3.3.2.4 Releasing Output Data

From our initial conditions out_ack was set low indicating that the external output circuitry was not ready to accept the output data of the micropipeline. In the steady state,

signal *ctrl3* would be high and the N-stage should have completed an evaluation. After *out_ack* is high, the value of *ctrl3* is toggled low, and the P-stage does its evaluation. Once completed, *complete_p* goes high and the external output circuitry is free to take the micropipeline result.

3.3.3 More Complex Systems

By having two processing stages in every basic cell, the number of possible ways of making forked connection increases and proper synchronization must be maintained. For example when data is sent from the output of an N-stage of a basic cell to the input of the P-stage in another basic cell, it is necessary to ensure there is proper synchronization. One way to achieve proper synchronization is to have one C-element for both basic cells. Having one C-element results in using the same ctrl signal between all adjacent N-stages and P-stages that share data. If this is too restrictive each block may still use separate C-elements but extra control signals need to be used to indicate when data values are valid to ensure correct circuit operation.

Recall that in the previous micropipeline implementation, input data from different sources are synchronized with multiple input C-elements. This idea has to be extended to allow for the case described above with the N-stage and P-stage residing in different blocks. Basically the shared C-element would have to accept more control signals from outside blocks. If a fully delay-insensitive system is desired, then there is no choice but to add this extra hardware. However, in practical circuits, a speed independent approach of using single control signals for multiple data bits with similar delays can also be used. This speed independent approach uses less hardware, however, the strict delay-insensitivity of the circuit can not be guaranteed.

3.3.4 The Modified TSPC P-Stage

It is known that PMOS circuitry is slower than NMOS circuits[5]. In order to optimize the technique described in this section, a new P-stage is defined. The new faster PMOS structure, known as All-N-Logic TSPC[16], was described in the background chapter. The operation of the basic cell remains essentially the same from an external point of view. The only difference is that now the PMOS tree used in the P-stage of a basic cell is replaced with a faster NMOS logic tree. By using this approach some improvement in performance can be achieved.

3.4 Method 3 - Using Delay Elements for Completion Detection

The two methods presented above are good for designing either delay-insensitive or speed-independent systems, but require extra circuitry to perform the completion detection operation. To reduce hardware, a bounded delay timing model may be used. As mentioned before, bounded delay models match most closely the synchronous style of design where the clock period is defined by the longest delay in the circuit. The concept of matched delays has been used in previous asynchronous designs[8]. It useful, however, to apply the delay matching concept to the techniques developed in this thesis to determine how much overhead can be saved by removing the dual-rail structure. The basic cell and system level integration for designing with the bounded delay model are discussed in the next two sections.

3.4.1 Basic Cell of the Bounded Delay System

The basic pipeline stage is similar to the other pipeline stages presented with the modification of replacing the DCVSL logic and completion detection circuitry with a delay element. The cell structure is shown in Figure 27. In the same way it was possible to have completion circuitry for both the N-stage and P-stage logic blocks, separate delay elements can be used for the N-stage and P-stage. That is, if only one stage is used for

processing, a single delay element can be used. However, when both stages are used to process data, two delay elements are needed to generate *complete_N* and *complete_P*. With the overhead for completion detection removed from the TSPC cells, the TSPC logic that is used in the basic cell is the same as TSPC used in synchronous designs.

FIGURE 27. Basic Pipeline Stage of the Bounded Delay System

3.4.2 System Integration - Bounded Delay Model

The architecture of the micropipeline structure does not change when delay elements are used instead of completion detection circuitry. So both pipeline structures described for Method 1 and Method 2 can be implemented using delay elements. The main difference is in the design and implementation of the basic cell. Choosing the delay element properly could mean that extra delay is added to the circuit as a safety margin to guarantee correct operation of the circuit.

Recall that when dealing with complex systems that have multiple forks and joins, extra control signals and larger C-elements are needed. One option to reduce the extra synchronization overhead would be to use delay elements with the same delay in all basic cells. This could mean that certain cells would finish a bit before the delay element outputs a completion signal, however, correct operation could be maintained without adding too much extra hardware. Synchronous systems use this philosophy. Even though each basic cell has the same delay through it, the system as whole would still remain asynchronous with respect to the flow of input and output data. The critical design question is assuring that the delay element does indeed have the proper delay.

CHAPTER 4

IMPLEMENTATION AND TESTING

This chapter presents implementation details for the proposed asynchronous design techniques presented in earlier sections of the thesis. Schematics for the basic circuit components are presented along with any extra circuit level details. In addition, the testing methodology for each of these circuits is presented together with final performance results obtained through simulations.

Circuits in the thesis were designed using NORTEL's 0.8um BiCMOS technology. All designs were carried out using CADENCE software and the resulting circuits were simulated using HSpice.

In the next few sections the three design techniques proposed in the thesis are discussed in sequence. Two implementations are discussed for each technique. At first a simple implementation is chosen to illustrate general properties of the circuit. A second more complex implementation is included to discuss issues arising from more complex circuits that generally have multiple forks and joins between basic cells. For the simple circuit, an asynchronous FIFO is built using the design ideas of the given technique. For the more

complex circuit, a full adder is built and tested. In addition performance results are provided for each of the design techniques.

The asynchronous FIFO results for all the techniques were obtained using 4-stage FIFOs. The performance results of the full adder circuit are based on a 4-bit implementation. The basic FIFO was also built using 20 pipeline stages with correct operation observed and with the same measured throughput rate as the 4-stage FIFO. For practical reasons, 4-bit implementations were chosen for the comparisons between the different techniques since there would be little difference in the performance results from larger systems except for latency. The latency, however, is directly related to the number of pipeline stages as was observed with the larger 20 stage FIFO that was implemented.

The final section of this chapter presents the relative performance results of the different proposed techniques. In addition, some comments are given about the differences in practical aspects of implementing each of the techniques.

4.1 Basic Pipeline - Method 1

An implementation using Method 1 was used as starting point for all methods. The other methods are either enhancements or alterations on Method 1. Details of a basic FIFO and 4-bit adder circuit for Method 1 are now presented. In subsequent sections only unique characteristics or issues pertaining to other methods are included.

4.1.1 Basic FIFO

The basic FIFO was constructed initially as the first actual implementation of the asynchronous design ideas, because it is the most basic configuration for any type of pipelined system. In addition, the performance results of the FIFO determine the upper bound performance for the system in general.

The schematic of a basic FIFO stage is shown in Figure 29. The transistors sizes shown resulted in the maximum performance of the FIFO. Note that the C-element shown as a box is the first half of the C-element described in the background chapter, with the final inverter stage shown explicitly. The inputs of the C-element are the *in_req* and *out_ack* signals indicating when input data is valid and when the output stage is ready respectively. The output of the C-element is the *ctrl* signal which drives all the precharge and evaluate transistors as shown in Figure 29. The input data is labeled *A* with a static inverter used to obtain *notA*. Also the *complete* signal is shown as an output of the static NAND gate. The operation of this FIFO stage is the same as was described earlier in Chapter 3. The structure used to make a full FIFO is shown in Figure 28.

FIGURE 28. Structure of multiple stage FIFO

4.1.2 Full Adder

The second circuit tested was a 4-bit adder, which was used to test both the correctness and performance of the proposed structure for more complex systems. The 1-bit full adder has two data inputs and one *carry-in* input. In most cases, all three inputs originate from different sources. Similarly there is a *sum* output and a *carry-out* output which, in most cases, will be sent to different destinations. Input data arriving at different times and output data reaching their final destination at different times are considered using the adder circuit. This input/output structure of the adder cell is useful since it is general enough to represent a larger class of circuits, enabling the design methodology to be tested properly.

FIGURE 29. Transistor Implementation of basic FIFO stage

The transistor level implementation of the SUM operation is shown in Figure 30. The transistors sizes shown resulted in the maximum performance of the 4-bit adder circuit. The schematic shows only the N-stage of the pipeline stage. The P-stage is a latch as in the case of the FIFO.

One detail worth mentioning in the schematic is the use of internal precharge transistors which are shown circled in Figure 30. These extra precharge transistors are required to eliminate the charge sharing problem found in dynamic circuits[17]. Charge sharing can cause the voltage of internal nodes to change levels and ultimately cause an error in circuit operation. Consider when all internal nodes, particularly I_X , have been discharged low and the circuit enters the evaluate phase. If the situation arises where the logic function connected to node P_X evaluates false and the NMOS transistor between nodes P_X and I_X is on then charge sharing occurs. The charge stored in the capacitance of precharge node P_X will be dumped into the capacitance of internal node I_X . The voltage on node P_X could drop below an acceptable level and cause an error at the output. The use of the circled precharge transistors prevents such a situation.

Another point to note is that Binary Decision Diagrams (BDDs), which were described in the background chapter, can be used to minimize the number of extra transistors required to convert a normal TSPC stage into a dual-rail structure. From Figure 30, only 3 extra transistors were required for the SUM operation of a full adder cell. This includes 1 extra PMOS precharge transistor and 2 extra transistors in the NMOS logic tree. The extra transistors are shown in Figure 30 with a box around them.

FIGURE 30. Transistor implementation of SUM operation of full adder stage

The final component used in the 4-bit adder is the C-element structure as shown in Figure 31. The first half of the C-element is contained in the box labeled *c-element1*, while the inverter making up the other half is shown explicitly. The transistor sizes shown in the diagram were used for the final performance values. Note that the 4-input C-element is structured as a tree as opposed to a single 4-input C-element. The tree structure has the advantage of being faster and also makes it easier to buffer up the final drive inverter to properly drive the next stage.

FIGURE 31. C-element structure for full adder stage

These sub-blocks are all combined to implement the 4-bit adder circuit under test. The top level schematic of the 4-bit adder is shown in Figure 32. This structure can be used as a building block to a larger asynchronous system. The figure is useful for visualizing how the data flows through the circuit, in particular the carry signal. Each full adder cell is labeled *Adder_N* while all the other cells are just one stage FIFOs. The carry signal propagates diagonally from *Adder1* through to *Adder4*. The FIFO stages are required to keep each of the input data bits and each of the output data bits synchronized with the carry signal.

FIGURE 32. 4-bit asynchronous adder schematic

4.1.3 Testing and Performance

The adder circuit was first tested by applying requests that arrived at random intervals. Testing the random arrival of inputs confirmed the proper operation of the design in an asynchronous framework. Next the circuit was tested for maximum speed by toggling the input request lines as fast as possible in a synchronous fashion. Also, all inputs were sent as alternating high and low values to ensure that full swing was obtained at each stage in the circuit.

Two versions of the full adder were tested. The first made no assumptions and was fully asynchronous. The second took into account that the basic 1-bit adder cell is slower than the basic FIFO stage, hence the completion signals could be obtained from just the adder cells. This simplification meant that instead of using a 4-input c-element, only a 2-bit version was needed, and this improved the performance of the system. Note that this second version is only valid if the wire delays between components is much smaller than the delay through the logic cell. In other words, this technique uses a speed-independent timing model.

Simulation results for the FIFO and the two versions of the adder are given in Table 1. The optimized adder has 2-input c-elements while the other adder is using 4-input C-elements. The *operations/second* indicate the number of logic evaluations done per second in each basic cell. Cycle time is the fastest time at which the pipeline can send out successive data values. Latency is the time it takes for data to go from the input of the circuit until it is finally ready at the output. Pipelined systems work on the principle of reducing the cycle time at the cost of increased latency. The next section examines how an enhancement to the system can reduce the latency even further.

	Operations/s	Cycle Time	Latency
Basic FIFO	$220 * 10^6$	4.5 ns	18.0 ns
Basic Adder	$130 * 10^6$	7.5 ns	30.0 ns
Optimized Adder	$180 * 10^6$	5.5 ns	22.0 ns

TABLE 1. Final Performance Results for Method 1

4.2 Enhanced Pipeline - Method 2

The original TSPC methodology allowed logic evaluations to occur in both the N-stage and the P-stage. With some minor modifications the asynchronous pipeline can also be designed to allow this behaviour as described in Chapter 3. In this case two computations are done in each stage. The main advantage of Method 2 over Method 1 is that the latency

requirements are reduced essential by half, for similar cycle times. In Method 1, the P-stage is very simple, and hence can be assumed to be faster than the N-stage. In Method 2 this assumption can not be made, and the final cell is truly asynchronous. Much of the implementations of Method 1 and Method 2 are similar, but some of the differences are discussed in this section.

4.2.1 Implementation of the Basic FIFO Stage and Full Adder

In Method 2, a single c-element is still required to drive the precharge/evaluate control transistors for both the N-stage and the P-stage. As the P-stage will now have extra logic evaluations, extra transistors will be connected to all the precharge nodes. To accommodate, the PMOS precharge transistors have to made larger. To drive these PMOS transistors, the C-element output transistors also have to be larger. On the other hand, there are now two completion signals in each basic cell. The load that the completion signal has to drive is now shared between these two signals. By sharing the output load, the output transistors of the NAND gate the creates the completion signals can be made smaller.

4.2.2 Performance of Method 2

As would be expected more operations are performed in each stage per second using Method 2. However there is a slight increase in the cycle time due to the slower P-stage. By increasing the number of operations in each basic cell the latency of Method 2 is much less than in Method 1. Table 2 presents the results obtained for Method 2. Recall from the discussion about Method 1 that the optimized adder takes into account that the delay through and adder cell is longer than the delay through a FIFO stage.

	Operations/s	Cycle Time	Latency
Basic FIFO	$400 * 10^6$	5.0 ns	10.0 ns
Basic Adder	$240 * 10^6$	8.25 ns	16.5 ns
Optimized Adder	$320 * 10^6$	6.25 ns	12.5 ns

TABLE 2. Final Performance Results for Method 2

4.3 Bounded-Delay Technique - Method 3

The bounded-delay technique has simpler logic cells, however the implementation of actual circuitry has to be done more carefully. Extra work is involved in ensuring that the delay element has enough delay for correct operation. At the same time, it is important to keep any added delay to a minimum. The next sections discuss issues pertaining to the delay element followed by a section on the FIFO and full adder implementation.

4.3.1 Implementation of the Delay Element

The delay element should be implemented such that it matches the delay in the computational circuitry as closely as possible. A schematic of the delay element is shown in Figure 33. The V_{bias} in the delay element can be used to adjust the amount of delay in the circuit. The extra pair of static inverters at the output are useful to filter out the slow rise times of the signals internal to the delay element.

FIGURE 33. Transistor Implementation of Delay Element

When design the circuits of Method 3, one approach to take is to design all circuits assuming the minimum delay in the delay elements, or that V_{bias} is fully on. The same

V_{bias} signal would be distributed to all the cells on a VLSI chip. Once the physical design comes back, process variations may cause delays to become unmatched. The V_{bias} signal can be adjusted down to increase the delay of the delay elements, giving more time for the computations to take place. With this type of control the circuit can still be made to operate correctly, though at a slower speed. This is similar to synchronous designs that are implemented to run at a certain frequencies, but process variations or other problems force the design to be run at a slower clock rate.

The output of the delay element is essentially the *complete* signal for the basic cell. These completion signals drive the inputs of C-elements in adjacent cells. One nice property of using delay elements is that they can easily be used as buffers to drive many C-elements, which can be useful for circuits that have many inputs or outputs in each basic cell.

4.3.2 Basic FIFO Stage and Full Adder Implementations

The structure of Method 3 still retains the underlying micropipeline control structure. To test performance of the bounded-delay technique, Method 3 was modeled after Method 1 which has logic computations only in the N-stage, with the P-stage used only as a latch. Also for the adder there is no need to have separate delay elements in each basic cell. Since the delay through the full adder is longer than the latch stage, only one delay element is needed per pipeline stage across all 4 bits of the adder. This reduces the overhead in the circuit substantially. As with Method 1, the performance of the *Basic Adder* is based on using 4-input C-elements, while the *Optimized Adder* only has 2-input C-elements.

4.3.3 Performance

The performance of Method 3 can be directly compared to Method 1. The final results of Method 3 are shown in Table 3. The performance improvements of Method 3 over Method 1 are quite good. Some of the reasons for performance improvements include elimination

of the static NAND gate and reduced internal capacitance. In addition to the performance gain, one would use Method 3 to reduce the extra circuitry required for dual-rail logic and completion detection.

	Operations/s	Cycle Time	Latency
Basic FIFO	$330 * 10^6$ ^a	3.0 ns	12.0 ns
Basic Adder	$180 * 10^6$	5.5 ns	22.0 ns
Optimized Adder	$250 * 10^6$	4.0 ns	16.0 ns

TABLE 3. Final Performance Results for Method 3

4.4 A Comparison of All the Design Techniques

Three design techniques were implemented and tested. Method 1 is fully asynchronous using dual-rail logic with completion detection. Computation is done only in the N-stage, with the P-stage used as a latch. Method 2 adds the ability to do computation in the P-stage as well. Method 3 replaces the completion detection mechanism used in the first two methods with delay elements. A comparison of all the methods is given in Table 4 for the case of the Optimized Adder. The performance of the synchronous adder is also given as a reference point.

	Throughput	Cycle Time	Latency
Method 1	180 MHz	5.5 ns	22.0 ns
Method 2	160 MHz	6.25 ns	12.5 ns
Method 3	250 MHz	4.0 ns	16.0 ns
Synchronous Adder	660 MHz	1.5 ns	6.0 ns

TABLE 4. A comparison of the different techniques

The first two methods are useful to design fully asynchronous systems without having to worry about matching delays, however, there is extra circuit overhead for the first two methods. The choice between the first two methods depends on the application. If higher throughput is desired and longer latency can be tolerated then the first method is better. However if latency is of concern then the second technique is better. The first two methods

have the disadvantage of extra circuitry overhead. To reduce the circuit overhead, delay matching techniques, shown in Method 3, are preferable. Of course, extra care has to be taken when using delay matching schemes.

Compared to fully synchronous systems, these asynchronous systems do not have as high a performance. The overhead of the micropipeline structure is the main reason for slower performance of asynchronous systems. Asynchronous systems trade off performance for added functionality. Another difference is that asynchronous systems are not affected by clock skew problems. The performance results of the synchronous system do not account for any clock skew that might be present in a real circuit.

CHAPTER 5

CONCLUSION

Dynamic logic has been traditionally used in a synchronous framework. The goal of this thesis was to develop techniques that would allow dynamic logic to operate in an asynchronous environment. The dynamic logic family known as True Single Phase Clocked (TSPC) logic, which is well suited to high speed pipelined systems, has been chosen for this purpose. Asynchronous circuits have traditionally been implemented with static logic. Combining dynamic TSPC together with asynchronous methods results in techniques that take advantage of both ideas.

Three techniques have been developed in the thesis. The first two used dual-rail logic for completion detection. The first method has been shown to be useful for higher throughput and longer latency applications. The second method has lower throughput but latency is reduced by about 50%. The third method developed in the thesis replaces the overhead of dual-rail logic by using delay elements whose delay is matched to the delay in the logic computation section. The third technique requires much less logic to extract the completion signals.

For each technique developed in the thesis, example circuits were implemented and tested. The performance results were obtained through HSpice simulations. If performance of small circuits is considered independently, then the techniques developed in the thesis have lower performance when compared to fully synchronous systems. The main advantage of the asynchronous approach is that it is free from clock skew limitations, as all clocking signals are generated locally. Asynchronous circuits also have the potential for low power dissipation, because no switching takes place when the circuit is idle[11].

The operating speeds of the circuits tested in the thesis were limited by the speed of the micropipeline structure. The relative delay of the dynamic logic evaluation cells was small compared to delay of the asynchronous control structure. Since most of the delay was due to the asynchronous control scheme, the use of dynamic logic minimizes the cost of doing logic evaluations in an asynchronous environment.

Future work in this area should address the issue of circuit complexity. Practical asynchronous circuits are still much harder to implement compared to their synchronous counterparts. In order to make the design of asynchronous systems more attractive, CAD tools that automate much of the design process need to be developed. Of course, the final challenge with respect to the design of asynchronous designs is to increase performance such that it is comparable to synchronous designs.

In conclusion, the original goal of developing techniques to operate dynamic logic in an asynchronous framework has been achieved.

REFERENCES

- [1] J. Yuan and C. Svensson, "High-Speed CMOS Circuit Technique", IEEE Journal of Solid-State Circuits, Vol. 24, No. 1, February 1989.
- [2] I. E. Sutherland, "Micropipelines", Communications of the ACM, Vol. 32, Number 6, pp. 720-738, June 1989.
- [3] T. H. Meng, Synchronization Design for Digital Systems, Kluwer Academic Publishers, 1991.
- [4] K. van Berkel et al, "Asynchronous Circuits for Low Power: a DCC error corrector", IEEE Design & Test, vol. II, pp. 22-32, 1994.
- [5] N. Weste & K. Eshraghian, Principles of CMOS VLSI Design, Addison-Wesley, 1995
- [6] Unger, Stephen, The Essence of Logic Circuits, Englewood Cliffs N.J.: Prentice Hall, 1989.
- [7] Brzozowski, J. A. and C. H. Seger, Asynchronous Circuits. New York: Springer-Verlag, 1994.

- [8] Hauck, Scott, "Asynchronous Design Methodologies: An Overview.", *Proceedings of the IEEE*, Vol. 83, No. 1, pp. 69-93, January 1995.
- [9] Meng, Teresa H., Synchronization Design for Digital Systems. Boston: Kluwer Academic Publishers, 1991.
- [10] Cortadella, Jordi, "Mapping BDDs into DCVSL gates", *UPC/DAC Technical Report*, No. RR95/04, 1995.
- [11] Mead, Carver and Lynn Conway, *Introduction to VLSI Systems*. Don Mills, Ontario: Addison-Wesley, 1980.
- [12] Furber, Steve, "Computing without Clocks: Micropipelining the ARM Processor", in *Asynchronous Digital Circuit Design*, G Birtwistle and A Davis, Editors, pp. 211-262, New York: Springer, 1995.
- [13] Bakoglu, H. B., "Circuits, Interconnections, and Packaging for VLSI", Addison-Wesley, 1990.
- [14] Dobberpuhl, D. W., et al., "A 200 MHz 64-bit Dual-issue CMOS Microprocessor. *Digital Technical Journal*, 4(4):35-50, 1993.
- [15] Al Davis and Steven Norwick, "Asynchronous Circuit Design: Motivation, Background, & Methods", Asynchronous Digital Circuit Design, G Birtwistle and Al Davis, Editors, pp. 1-49, New York: Springer, 1995.
- [16] Richard X. Gu and Mohamed I. Elmasry, "All-N-Logic High Speed True-Single-Phase Dynamic CMOS Logic", IEEE Journal of Solid-State Circuits, Vol.31, No. 2, February 1996..
- [17] Martin, K, "Digital Design", Course Notes ECE 342S, 1994
- [18] Anthony J. McAuley, "Dynamic Asynchronous Logic for High-Speed CMOS Systems", *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 3, March 1992.

IMAGE EVALUATION TEST TARGET (QA-3)

APPLIED IMAGE, Inc.
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved