IN THE NAME OF **ALLAH**;
MOST GRACIOUS, MOST MERCIFUL!

FPGA–Compliant Micropipeline Based Asynchronous Systems

by

YOUSAF ZAFAR

A dissertation submitted to M.A.J.U. in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Department of Electronic Engineering

Faculty of Engineering and Sciences

MOHAMMAD ALI JINNAH UNIVERSITY

2005

My work is dedicated to the brave Muslims

who seek martyrdom to protect Islam.

Acknowledgements

I am grateful to ALLAH, Who permits me to live and accomplish tasks including the research work being presented in this thesis.

Dr. M. Mansoor Ahmad (CEng, FIEE), my Ph.D. Supervisor and HoD(EE) Mohammad Ali Jinnah University (M.A.J.U) is the person who was always there to help me during my entire graduate program. I would have been no where without him. I am very proud to have work under his supervision.

I must also thank the M.A.J.U administration for providing me an excellent environment, perfect for conducting research and Higher Education Commission, Govt. of Pakistan for financially supporting me in my academics through its indigenous PhD program.

My current research work is the fruit of a tree of practical experience planted and nurtured by my beloved parent department, Pakistan Atomic Energy Commission. I gratefully acknowledge all that my boss and my teacher Mr. Anwar Ali, Member (Tech.), PAEC did to polish my skills in computer engineering.

It is because of Dr. Richard Sandige, then Associate Professor at the University of Wyoming, USA; my advisor during the undergrad studies; that I am conducting research in the field of Digital Electronics. He was the one who introduced me to asynchrony and reconfigurable devices back in 1990.

I must also thank my teachers at ICB, G-6/3, Islamabad, from where I did my high school, for inducing a love of religion and country in me. I collectively thank them by addressing my vice-principal Mr. Subhanullah. It is because of them that I am conducting research in Pakistan in an effort to elevate the status of Muslims.

How can I ever find words to thank my parents, who made me what I am today. It is they who engraved the love of knowledge in my soul.

I am grateful to my beloved wife and sons, Salahuddin (*Salah*) and Jalaluddin (*Hamza*) for supporting me during this extremely exhaustive period of research. I would not have achieved the goals without their sincere cooperation and love.

I am also using this opportunity to thank all my students who prayed for me and encouraged me through their love and sincerity.

May Allah bless all these wonderful people.

Yousaf Zafar
September 23, 2005

List of Publications and Submissions

[1] Y. Zafar and M. M. Ahmad, "A Novel FPGA Compliant Micropipeline", IEEE Transactions on Circuits & Systems II, vol. 52(9) 2005, pp. 611-615.

[2] Y. Zafar and M. M. Ahmad, "Adaptive On–chip Oscillator for FPGA based Synchronous Designs", Proceedings of IEEE International Conference on Emerging Technologies ICET'05, September 2005.

[3] Y. Zafar and M. M. Ahmad, "Globally Asynchronous Locally Synchronous Micropipelined Processor Implementation in FPGA", Proceedings of IEEE International Conference on Emerging Technologies ICET'05, September 2005.

[4] Y. Zafar, N.D. Gohar and M. A. Uppal, "Self–Controlling Asynchronous Processor (SCAP)–A System on Programmable Chip (SOPC)", Proceedings of IEEE INMIC'02, December 2002.

[5] Y. Zafar, and N. D. Gohar, "Reconfigurable Computing – Where it stands now", Proceedings of IEEE INMIC'02, December 2002.

[6] Y. Zafar and M. M. Ahmad, "A Micropipelined Processor Implementation in a Reconfigurable Medium", Electronics and Telecommunications Research Institute (ETRI) Journal. (submitted).

[7] Y. Zafar and M. M. Ahmad, "Adaptive On–chip Ring Oscillator for FPGAs Based Clocked Circuits", IEE Electronic Letters, (submitted).

ABSTRACT

Latest trends in digital electronics require designs with increased functionality, contained in the smallest of spaces, performing at the highest of speeds, consuming and dissipating the minimum of power and generating the least of electromagnetic interference. The merger of all these properties in a single synchronous design is becoming increasingly difficult, as one property contradicts the other, whereas asynchronous systems exhibit technology independence, power efficiency, average case computational capability and electromagnetic compatibility. Above all, the problem of clock skews does not exist in asynchronous systems because of the absence of a common clock.

Reconfigurable mediums such as Field Programmable Gate Arrays (FPGAs) associate lower costs and lesser turn around time associated with prototyping. FPGA based designs perform like ASICs while retaining the flexibility of General Purpose IC.

Because of the very construct of FPGAs and their programming environments, asynchronous systems find their place in the full custom domain while reconfigurable mediums are associated with synchronous designs. In the presented research, two areas are combined to come up with techniques through which an asynchronous system like a 4-phase micropipeline has been implemented in a reconfigurable and traditionally synchronous medium of FPGAs.

The implementation of a FPGA–compliant micropipeline required development or change in fundamental building blocks of standard micropipeline. Therefore, a technology independent and customizable delay element with dynamic calibration capability was developed for FPGAs. Special Event Controlled Register (ECR) for

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

Introduction

"He who finds a thought that enables him to obtain a slightly deeper glimpse into the eternal secrets of nature has been given great grace."
Albert Einstein.

1.1 Motivation

Increased functionality, reduced real-estate, high performance, power efficiency and Electromagnetic Compatibility (EMC) have lead researchers into the arena of asynchronous systems. Another important issue related to today's mobile world is reconfigurability of a digital system to make it more versatile for the varying user needs. A lot of quality research work has already been done in the field of asynchronous processing while staying in the full custom domain. But implementation of asynchronous designs in conventional reconfigurable devices such as Field Programmable Gate Arrays (FPGAs), manufactured to suit synchronous designs, has attracted the attention of a few, because of some very fundamental reasons related to the architecture of such devices. Therefore, asynchronous systems till now reside in the full custom domain, while the reconfigurable mediums support synchronous designs. In this thesis a methodology is presented that imports

asynchronous system like micropipeline to the reconfigurable medium of FPGAs. The next few pages of the introduction chapter highlight the following issues:

- Justification of asynchrony,
- Importance of reconfigurability,
- Issues related to the implementation of micropipeline based asynchronous systems in FPGAs,
- Thesis layout.

1.2 Synchronous vs. Asynchronous Systems

A synchronous system by definition is the one, whose subsystems are synchronized by a common clock. The fundamental building blocks for a synchronous system i.e. the flip flops are characterized by their setup and hold time requirements [RSS90]. An input to the synchronous system is incorrectly manipulated if these requirements are not met. These characteristic requirements of sequential elements are dependent on the associated wire and propagation delays [RFT91]. Technology involved in the fabrication of these elements and the physical properties of metal interconnects /tracks define the values of these delays [MJS97] in case of both, single IC based or larger, synchronous systems. Conventional CISC (Complex Instruction Set Computer) and RISC (Reduced Instruction Set Computer) processors [HJM86] are typical examples of large synchronous systems.

A pipeline generally associated with RISC processors as shown in Figure 1.1, is a synchronous system where the inter–stage latches, using common clock, synchronize its various stages [DAP03]. The size of functional units associated with each stage in a pipeline and the total number of pipeline stages, define its real estate [JBT99]. As

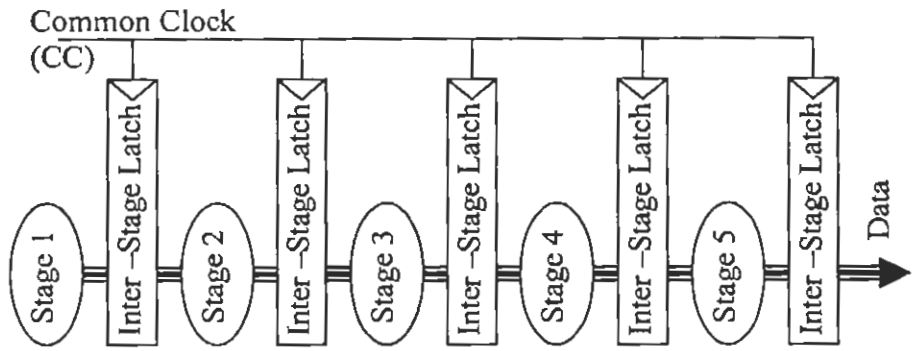


Figure 1.1: A 5-stage synchronous pipeline with processing, driven by common clock (CC).

the real estate of pipeline increases a problem that begins to emerge is that of clock skews. The synchronizing edges of clock reach the various sections of synchronous system with significant variation in time, although synchronous systems are based on assumption that time is discrete [SHK95]. Significant variation means time period comparable to the clock pulse width as explained by Figure 1.1.

In Figure 1.1a, a large synchronous system is driven by a common clock. Clock path to sub section B is much longer than path to sub section A. So the clock edge to A reaches earlier than B. The result is a system that cannot truly be called synchronous because all the subsections are not being triggered simultaneously. In case of a high frequency common clock (*CC2*) an entire clock cycle can be missed before the previous pulse reaches the farther sub section as shown in Figure 1.1b. Therefore, the problem of clock skews is aggravated by clock speed [SMN97]. The maximum clock speed is limited by the setup and hold time requirements of sequential elements, being driven by it. Enhanced technology incorporating reduced feature size also affects the sequential elements thus, permitting the use of high speed clocks. Therefore, as the speed of a synchronous system is improved, the problem of clock skews becomes more and more evident, forcing designers to modify designs [SMS96].

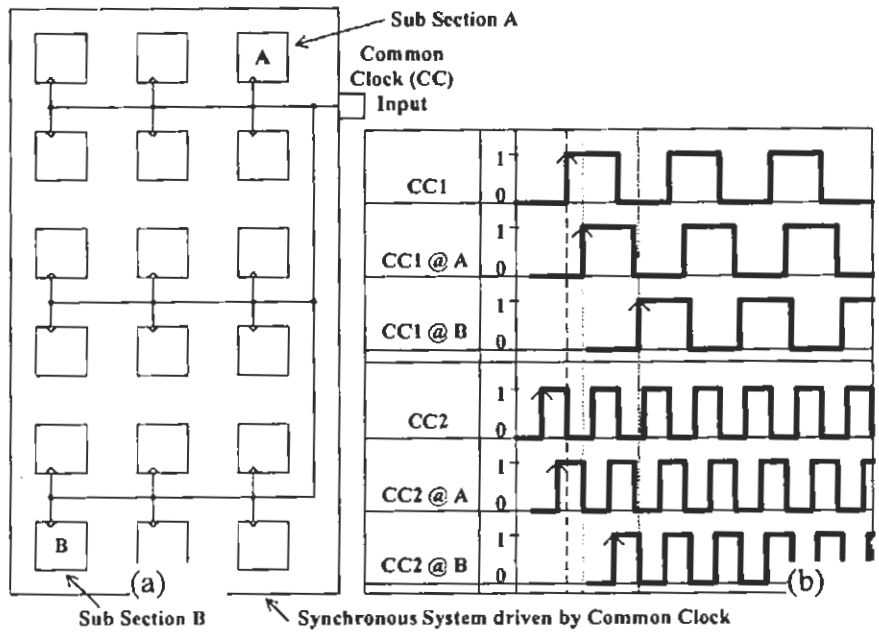


Figure 1.2: Describes the phenomenon of clock skew: (a) Large synchronous system having clock skews because of variation in common clock (CC) path length to sections *A* and *B*, (b) Problem of clock skew aggravated in case of faster common clock CC2, where new pulse arrives at *A* when *B* is still waiting for the previous pulse.

Another important aspect of enhanced technology is the issue of interconnects. Interconnects consume major portion of total area and the rate at which interconnect metal technology develops is three times slower than the rate at which the semiconductor technology develops [DAP03, DAM00]. This causes a rapid growth in gap between transistor density and interconnect density on a large synchronous circuit [JDM01]. Clock skews become more evident as a result of this difference in densities.

Increase in transistor density, on its part creates another kind of problem associated with power. Increase in the number of transistors switching per unit area and the frequency with which they switch leads to an overall growth in power consumption [DAP03].

Additionally, in case of synchronous systems the spectral components that approach or exceed the limits specified in the relevant (Electromagnetic Compatibility) EMC standard are those produced by clock oscillators or their harmonics [DLR00]. The high speed clock on Printed Circuit Board (PCB) traces is a major cause of Electromagnetic Interference (EMI), especially at the orthogonal trace corners [HWO88, CRP92]. As frequencies and edge rates continue to rise, corners introduce excess capacitance and cause a small change in characteristic impedance [MIM96]. This becomes disastrous at high frequencies (e.g. 100 MHz) when electrons virtually fly off the sharp corners of the bend [MIM96]. Therefore, an external clock source, by virtue of being off–chip (on–board) is a source of EMI, difficult to suppress.

In the light of the above mentioned issues, it can safely be stated that a race in time to make synchronous systems like pipelined microprocessors faster with more real estate for enhanced on–chip functionality and higher density for reduced cost, results in serious issues to be dealt with. In order to accommodate these issues like clock skew, electromagnetic interference and power inefficiency [JBT99], designers spend exhaustive and expensive man hours in modifying the synchronous design thus increasing the cost of the product [SHK95]. The pressure to launch an upgrade in the minimum of time while maintaining cost in a highly competitive market has pushed synchronous design engineers to their limits.

As a technical field moves closer to saturation, researchers look for alternate solutions that are radically opposed to the fundamentals of the previous one. High end CISC architectures like VAX (Virtual Address eXtension), triggered alternate approach of RISC in the form of Berkeley RISC I [JBT99]. Similarly, for some time now, researchers have been experimenting with asynchronous processor architectures as

apposed to the prevalent synchronous ones [SMN99]. The problems associated with large and high speed synchronous designs are inherently taken care of in case of asynchronous designs.

A completely different approach generally requires development of all its fundamental building blocks. Therefore, for a competitive asynchronous RISC microprocessor to find its place in the market, basic structures like an asynchronous pipeline dependent on request /grant protocol instead of clock, delay model, event controlled registers etc. had to be developed first. This lead to the development of micropipeline [IAS89]: an event–based elastic pipeline with or without processing. Details of the asynchronous microprocessor architectures based on various models of micropipeline are presented in the following chapter. The following chapter also discusses the salient features, classification, design and characteristics of fundamental building blocks for the asynchronous systems.

1.3 Reconfigurability

Another very important issue effecting the overall cost and turn around time of a product is prototyping. FPGAs have played a vital role in prototyping because of their reprogrammable nature. A design can be implemented in an FPGA, modified and re–implemented in a fraction of time and cost associated with full custom prototyping. More recently, because of the incorporation of Run Time Reconfiguration (RTR) capability [BLH95], FPGAs have found their way into the market as end products providing platform to hardware designs requiring reconfiguration at runtime.

Application specific devices are efficient in power and performance but are limited to specific type of applications. On the other hand, general purpose devices that are not

limited to specific applications are less efficient in terms of power and performance as they have real estate overhead to take care of all types of applications. All the resources of such devices, therefore, are not fully utilized or are not available to the extent required by a specific application. The real estate overhead adversely effects power calculations, whereas lack of relevant resources, effects performance. Both these factors are not acceptable in modern designs. Latest technology trends require fast designs that are small in size to be contained in mobile devices. Mobile devices with on-board power source are generally based on power efficient designs to avoid frequent recharging. These factors pave way for reconfigurable devices such as FPGAs that at run time can reconfigure to suit the requirement of a specific application, while maintaining their real estate. Reconfigurable mediums such as FPGAs are therefore expected to play a dominant role in tomorrow's computing. Hence, it was essential to introduce micropipeline based asynchronous systems to the reconfigurable mediums.

1.4 Contribution: Merger of Asynchrony and Reconfigurability

FPGAs are extensively used today for the implementation of synchronous systems. In fact, the very structure of FPGAs and associated programming environments support synchronous designs, whereas asynchronous designs are dependent on their delay model. For an asynchronous design to be technology independent, delays in the design must have dynamic calibration capability. A delay element with these characteristics is considered to be implausible in case of FPGAs. Isochronic forks [AJM89] in the delay model also require full custom implementation. The problem is aggravated by the fact that design implementations in FPGAs are not necessarily

repeatable, in various programming environments because of the stochastic processes involved and the variety in reduction, placement and routing algorithms [SHK95].

As a result, asynchronous systems find their place in full custom domain while reconfigurable computing is associated with synchronous designs. Alternate architectures for FPGAs, facilitating asynchronous design implementation have also been proposed [REP96, JTL04, CGW03]. But implementation of asynchronous designs in conventional reconfigurable devices, manufactured to suit synchronous designs, has attracted the attention of a few.

In the presented research, the two fields are combined to come up with a technique through which an asynchronous system like a 4-phase micropipeline; an event based pipeline with or without processing; has been implemented in a reconfigurable medium of FPGAs.

Available models for the fundamental building blocks of full–custom micropipeline, such as the delay elements, Event Controlled Registers (ECRs) and hand shaking protocols could not be used, rather new concepts and techniques had to be developed for their FPGA based implementation.

Therefore, the presented methodology can import asynchronous designs along with their benefits to the traditionally synchronous environment of FPGAs. The merger of asynchrony and reconfigurability may reshape computer architecture in future.

1.5 Thesis Layout

Following the introduction chapter, a detailed description of asynchronous systems, their merits over synchronous systems, characteristics of their basic building blocks

and associated protocols is presented in chapter 2. Evolution of reconfigurable mediums justifying their importance and making them unique platform for the implementation of hardware designs is also presented in the same chapter.

Chapter 3 is dedicated to the discussion of Single Inverter Ring Oscillator (SIRO) implementation in FPGAs. SIRO's behavior as technology independent element, its characteristics as on-chip oscillator triggering fellow synchronous circuits and its use in the generation of delay elements with dynamic calibration capability in reconfigurable media is also discussed.

Chapter 4 presents the instruction set architecture of simple RISC, created to test the concepts presented in this thesis. The same architecture is implemented in various FPGAs in chapter 5, while triggered externally by a clock source and then by coexisting SIRO based on-chip oscillator to show the advantages of SIRO based designs and their optimal performance regardless of FPGA technology.

Chapter 6 gives the detailed description of the developed FPGA compliant micropipeline. All the building blocks of this FPGA–based asynchronous system and the associated protocols are also discussed in this chapter. In order to verify the proposed concept, a Reconfigurable Micropipelined Processor (RMP) based on the architecture of Simple RISC, is presented in chapter 7. Power calculations for the applications to satisfy the power efficiency requirement of asynchronous designs are discussed in chapter 8, while the conclusion and possible future extensions are given in chapter 9.

2

Background

Clearly, there is potential for use of asynchronous machines. In fact, it is predictable that designers will become more familiar with this type of machine, that asynchronous design techniques will improve, and that asynchronous FSM methods will play an important role in the design of future super high-speed microprocessors and computers. It is the judgment of many digital designers that synchronous IC system design (in general) has been pushed to its practical limit and that new approaches to digital design must be developed if future expectations are to be realized.

Richard F. Tinder

2.1 Asynchrony

Being digital in nature, asynchronous circuits assume the signals to be binary, but the other important assumption of time being discrete, is ignored in their designing. Asynchronous systems exhibit the following characteristics, giving them an edge over their synchronous counterparts:

2.1.1 No clock skew

The absence of a common clock triggering every section of the design eliminates the problem of clock skews. However, if the system is Globally Asynchronous and

Locally Synchronous (GALS) [DMC84, AIR02] where subsystems driven by various internal or external clocks communicate asynchronously to form the complete system as shown in Figure 2.1. The synchronous subsystems may have clock skews especially if they are large or improperly designed.

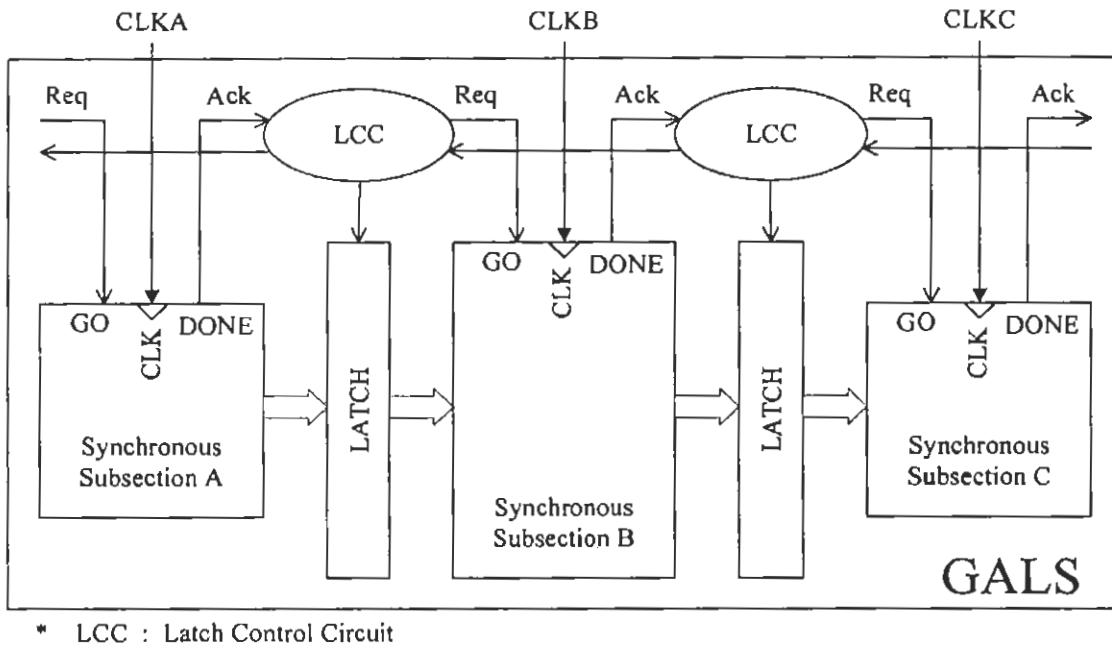


Figure 2.1: Globally Asynchronous Locally Synchronous (GALS) system showing synchronous subsections driven by different clocks.

2.1.2 Average-case performance

In synchronous systems all the subsections are driven by a common clock. Some sections are more sluggish than the other ones. Therefore, clock speed is chosen, taking into consideration the slowest subsection, exhibiting an overall worst case performance. On the other hand, in case of asynchronous systems absence of a common clock and freedom of subsections to operate at their own pace yields average case performance.

Prone to worst case scenario, synchronous systems, have all sub portions including rarely used sections, carefully optimized to achieve the highest clock rate. Since asynchronous systems operate at the speed of the circuit path currently in operation, rarely used portions of the circuit can be left un-optimized without adversely affecting system performance. This in turn facilitates designing process and reduces turn around time.

2.1.3 Power efficiency

The continuously toggling clock lines reach all subsections of a synchronous system whether used in the current computation or not. For example, if an instruction stream contains only integer manipulations, the floating point unit in the synchronous processor will still be operated by the clock, despite being unused. On the other hand, asynchronous systems have transitions only in areas involved in the current computation and the remaining regions stay in the sleep mode thus, conserving power.

2.1.4 Technology independence

A system may migrate from gate arrays, to semi-custom to custom IC in its developmental process, or may face technology upgrade for enhanced performance as discussed in the Chapter No. 1. Asynchronous designs exhibit technology independence as design modifications are not required upon migration from one technology to another. In many asynchronous systems, modification in only the more critical system components can improve system performance on average, since performance is dependent on only the currently active path. The asynchronous systems where computation completion is sensed, components with different delays

can be substituted into a system without altering other elements or structures [SHK95].

2.1.5 Adaptation to physical environment

Delays in the system can change with variations in fabrication, temperature, and power-supply voltage. The frequency of the common clock in case of synchronous circuits must be selected in such a manner that it incorporates even the worst combination of factors adversely effecting system delays. This results in poor performance, when most of the time, the situation is not critically worst. Asynchronous systems automatically adapt to the changing environment, always giving optimal performance under all circumstances. Or in other words, asynchronous systems are characterized by system delays that have dynamic calibration capability.

2.1.6 Electromagnetic Compatibility

Because of the absence of common clock, the greatest single source of electromagnetic interference in a system, and wide spread clock traces / interconnects, asynchronous systems show Electromagnetic Compatibility (EMC). These systems generate less EMI, making them ideal for applications where the communication section is to be placed close to the processing section. The example of such an application is the use of asynchronous implementation of 80C51 microcontroller [HVG98] in communication equipment.

2.2 Classification of Asynchronous Systems

Asynchronous systems can be hierarchically classified into the following five categories:

2.2.1 Delay Insensitive (DI) Designs

A DI circuit is the one that is designed to operate correctly regardless of the delays on its gates and wires i.e., an unbounded gate and wire delay model is assumed [SMN97]. In bounded-delay model, if an input is applied to a circuit, it settles in a finite and known amount of time so that a new input can safely be sent to it again. In case of a delay–insensitive model, even an infinite amount of time does not guarantee that the input is received properly. So before the sender sends another piece of information it must wait for a (transaction) completion or acknowledge signal from recipient, signifying that it has received the previously sent information, completely. For the generation of such a signal in the receiver, completion detection circuitry is added to it.

As an example consider a simple 4-bit serial adder whose inputs and outputs are latched, as shown in Figure 2.2. The latched result (R[3:0]) is to be used by the following stage, when the latched inputs to the adder are : A[3:0] = 0111 and B[3:0] = 0001. For simplicity, it is assumed that the inputs are certainly known to have reached stable state at the input latches and that wire delays associated with a[3:0] and b[3:0] are equal. Before the following stage uses the latched value, the R must be stable, which is dependent on lot of factors, i.e. length of r_0, r_1, r_2 and r_3 defined by the wire delays, gate delays associated with the adders and the design of adder which is serial in nature as every higher result bit is generated after a stable carry is generated by the previous bit manipulation. The adder must not be allowed to perform the following manipulation unless and until the present result is utilized by the following stage or the loss of data will occur.

In case of the current manipulation, least significant bit of the result will get stable first, also generating c_0, that will cause r_1 to be generated and finally the most significant bit r_3 will get stable at the end. The output latch must not be allowed to operate, till all the result bits r[3:0] are stable, or possibility of incorrect latching of information will prevail. The gate delays associated with adders and the wire delays associated with c[2:0], r[3:0] also contribute to the time that must be elapsed before output latch is allowed to perform its function.

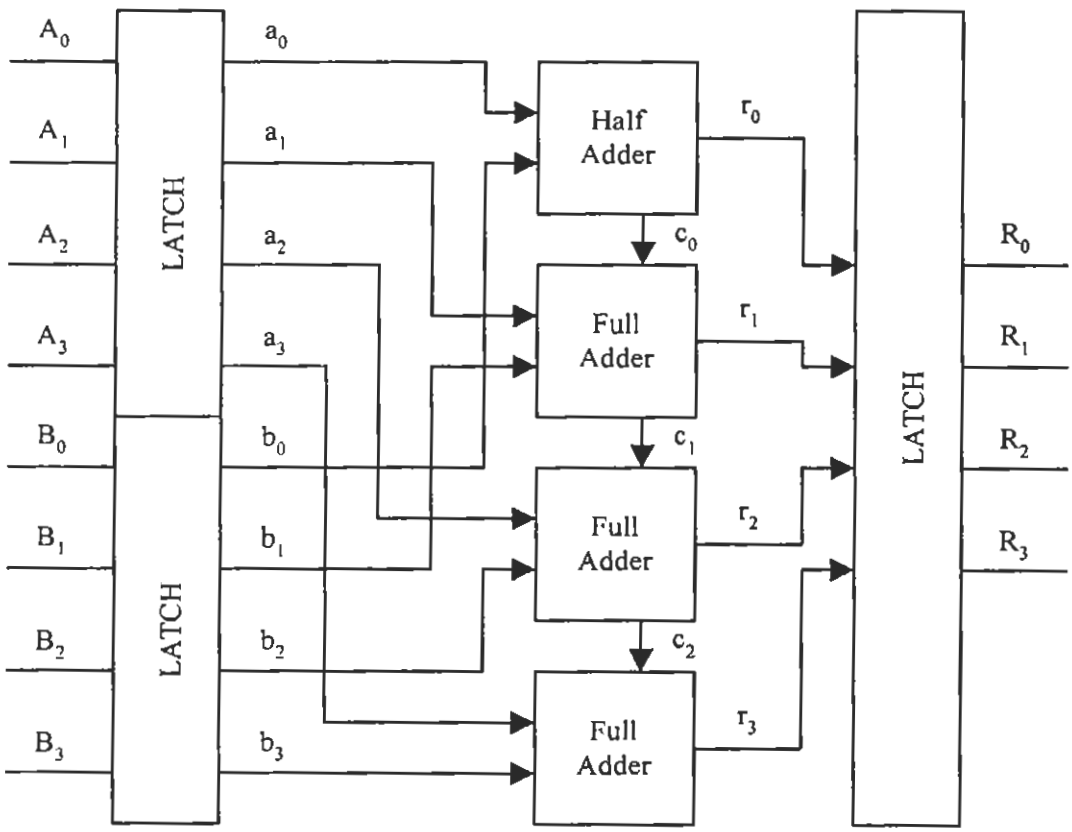


Figure 2.2: A simple 4-bit serial adder with latched inputs and outputs.

If the circuit is converted to a synchronous design, frequency of a common clock triggering all the latches of the system is selected in such a manner that it incorporates the associated gate and wire delays, so that the active edges activate the latches upon the completion of delay requirements by the stage (adder). In case of more than one

stages triggered by common clock the stage with the longest delay defines cycle time, thus associating worst case scenario with synchronous designs.

The other possibility is that the gate delays of adders in Figure 2.2 are known, so the time required in the generation of carry bits is known in each case, which in turn can define the amount of time lapse between the generation of successive result bits (c[2:0] is predefined). The wire delays associated with r[3:0] can be adjusted in such a manner that they cancel out the effect of generation of result bits in variable time. Therefore, in this case all the results bits will reach the output latch at same instant and at that moment, the latch can be permitted to perform its duty of latching. This is a bounded delay model, where gate and wire delays are previously known.

One can imagine the scenario when no limits are imposed on the gate and wire delays associated with inputs, outputs and processing unit in Figure 2.2, as is the case with technology independent designs. Such is the unbounded delay model for wire and gate delays, defining a DI system. In DI circuits, there is no guarantee that a wire will reach its proper value at any specific time, since some prior element may be delaying the output. Therefore, for the data to transit between the sender and the receiver in a DI system, there must exist a handshaking protocol of request and acknowledge between the two (i.e. one element informing the next that it can proceed).

2.2.1.1 Handshaking Protocol

Upon sending data, a request transition is sent from the sender to the receiver, and a response transition is sent back by the completion detection logic, upon complete reception. This forms a 2-*phase handshaking*, where both the positive and negative edges are considered. On the other hand, a 4-phase handshaking protocol is the one

where a second set of request and response (acknowledge) transitions are sent in order to return the connecting wires to their original (inert) values. 2-phase and 4-phase handshaking protocols are shown in Figure 2.3.

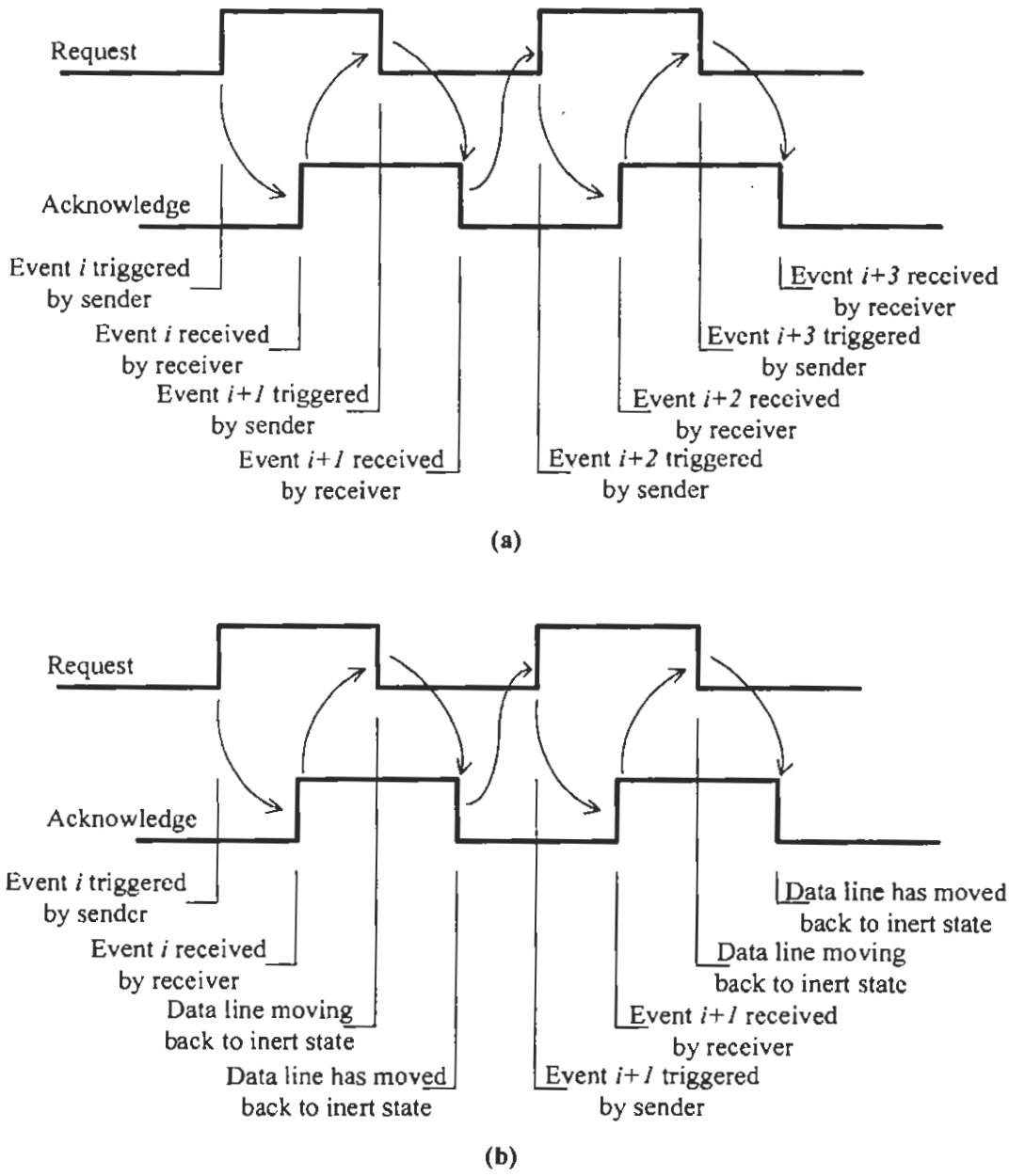


Figure 2.3: Shows (a) 2-phase handshaking protocol. Every edge in the request / acknowledge protocol is an active edge. (b) 4-phase handshaking protocol. Only positive edges are active edges in the request /acknowledge protocol. The other set of edges is used to bring the communication line back to the inert state.

In case of 4-phase protocol only rising transitions are considered active, for data transmission and reception completion notification. Although the 4-phase handshaking appears to require twice as much time because twice as many transitions are sent, in most cases computation time dominates communication time, making 4-phase delays competitive. Also, since only a rising edge initiates communication, the 4-phase circuits can be simpler than their 2-phase counterparts.

2.2.1.2 Bit Encoding

Another important aspect that needs to be discussed here is the completion detection logic. How can the receiver know whether information that the sender sent is completely received or not prior to generating an acknowledge signal. In the example of Figure 2.2, r[3:0] pass through the following values before the result is stable: 0111 → 0110 → 0100 → 0000 → 1000. An event is triggered, even at the transition of a single bit, therefore the receiver has no way of knowing when the data is stable. In case of DI systems, delays are unpredictable making the issue a complex one. The data in this case is said to be unbundled. Bit encoding is used to take care of this problem, where each data bit is encoded to two wires. Let the two wires be labeled as *I0* and *I1* (Figure 2.4), with a transition on *I0* indicating the data bit is a 0, and a transition on *I1* indicating the data bit is 1.

As an example, Lets consider the adder shown in Figure 2.2 to have R=0100. After manipulation, the new R=1100. Only the most significant bit changed. But the receiver did not know whether to expect change on a single bit or on all the bits, because there is no way of differentiating between 0100 → 1100 transition and 0100 → 1101 transition. Bit encoding can be used to ensure that all bits change their status

upon a manipulation, whether their status changes in the actual manipulation or not. Figure 2.4 shows the basic unbundled data handled by bit encoding and the related completion detection circuitry.

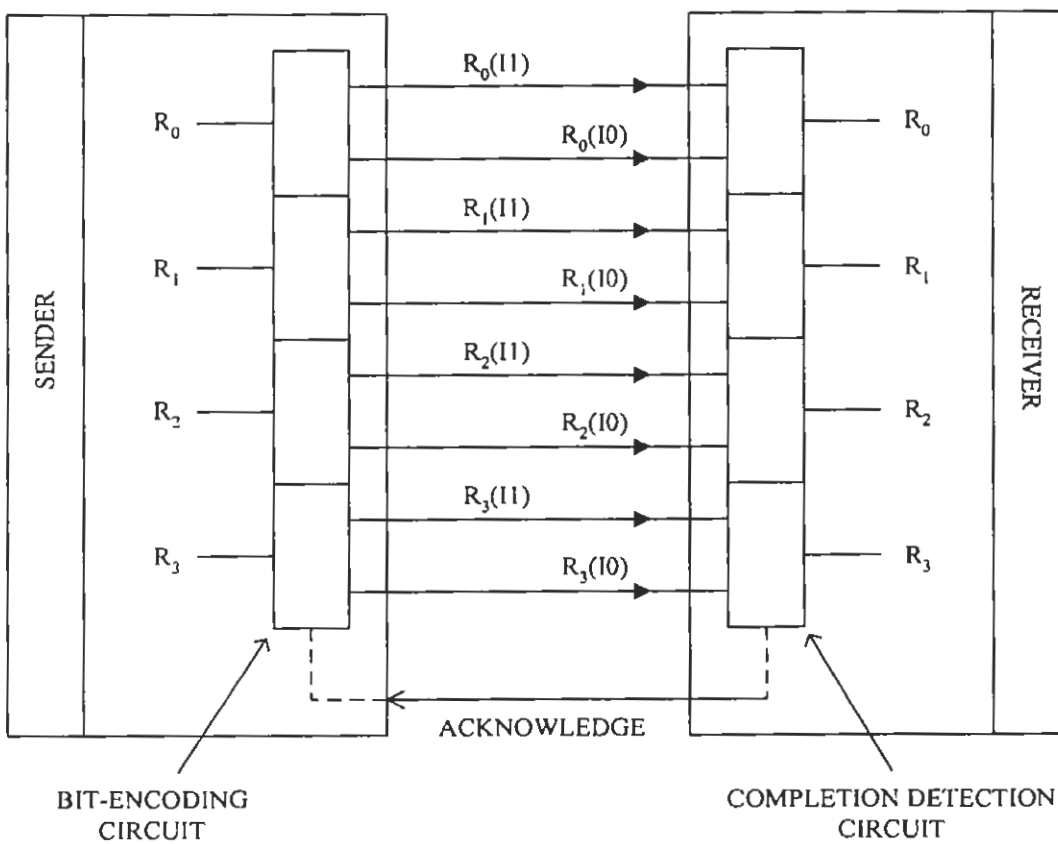


Figure 2.4: Bit-encoding and completion detection circuitry associated with unbundled data strategy.

Table 2.1 shows the bit encoding with Return To Zero (RTZ) scheme. Note that both 2-phase and 4-phase protocols can be implemented to incorporate this scheme, where prior to the following transition, the encoded data wires are forced to an inert state (zero in RTZ).

TABLE 2.1

Bit Encoding with Return to Zero scheme for unbundled data strategy.

Data Manipulation # 1	**Data**	R_0		R_1		R_2		R_3	
		0		1		0		0	
	Encoded Data	$R_0(I1)$	$R_0(I0)$	$R_1(I1)$	$R_1(I0)$	$R_2(I1)$	$R_2(I0)$	$R_3(I1)$	$R_3(I0)$
		0	1	1	0	0	1	1	0
Return To Zero (RTZ) *Inert state		$R_0(I1)$	$R_0(I0)$	$R_1(I1)$	$R_1(I0)$	$R_2(I1)$	$R_2(I0)$	$R_3(I1)$	$R_3(I0)$
		0	0	0	0	0	0	0	0
Data Manipulation # 2	**Data**	R_0		R_1		R_2		R_3	
		1		1		0		0	
	Encoded Data	$R_0(I1)$	$R_0(I0)$	$R_1(I1)$	$R_1(I0)$	$R_0(I1)$	$R_0(I0)$	$R_1(I1)$	$R_1(I0)$
		1	0	1	0	0	1	1	0
Return To Zero (RTZ) *Inert state		$R_0(I1)$	$R_0(I0)$	$R_1(I1)$	$R_1(I0)$	$R_2(I1)$	$R_2(I0)$	$R_3(I1)$	$R_3(I0)$
		0	0	0	0	0	0	0	0
Data Manipulation # 3	**Data**	R_0		R_1		R_2		R_3	
		1		1		0		1	
	Encoded Data	$R_0(I1)$	$R_0(I0)$	$R_1(I1)$	$R_1(I0)$	$R_2(I1)$	$R_2(I0)$	$R_3(I1)$	$R_3(I0)$
		1	0	1	0	0	1	1	0
Return To Zero (RTZ) *Inert state		$R_0(I1)$	$R_0(I0)$	$R_1(I1)$	$R_1(I0)$	$R_2(I1)$	$R_2(I0)$	$R_3(I1)$	$R_3(I0)$
		0	0	0	0	0	0	0	0

2.2.1.3 Bundled Data Strategy

The problem associated with unbundled data strategy is real estate overheads, as twice the number of data wires are required in the datapath implementation. So an alternate approach called Bundled data strategy was developed [SHK95, SMN97, IES89], that allows fewer wires to be used, but violates the delay-insensitive model. It allows a single wire for each data bit, and one extra control (request) line for each bundled data word as shown in Figure 2.5.

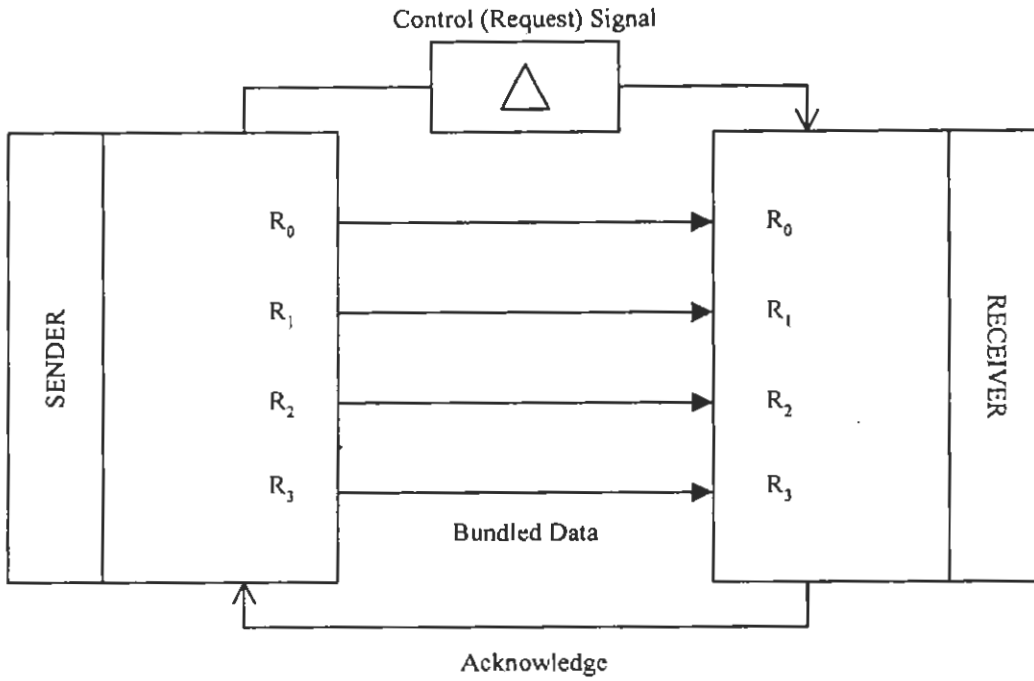
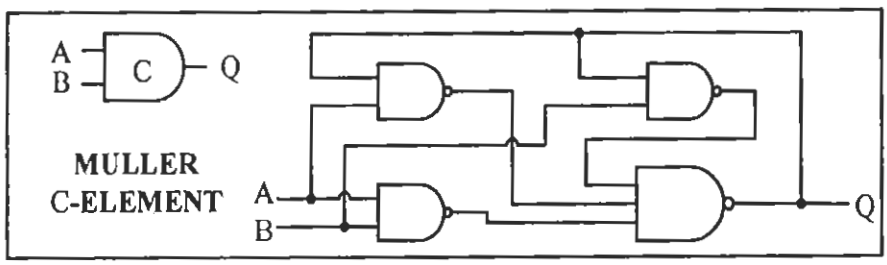


Figure 2.5: Bundled data strategy showing delay in request and acknowledge signals greater than the delay in datapath.

It is assumed that the delay in the extra control wire is guaranteed to be longer than the delay in each of the data wires. Thus, the control signal arrives at the receiver end after all the data bits have reached it having the desired values on them. When the receiver sees the transition on the control wire it knows the values on the data lines have already arrived.

2.2.1.4 Muller C–Element

DI circuits evolved as a result of work done on Macromodules [WAC67] in 1960s and 70s by Clark and Molnar. The DI system was formalized by Udding [JTU86]. C–element [REM65] shown in Figure 2.6 is an example of delay–insensitive circuit. C–Element is extensively used while implementing handshaking in larger asynchronous systems.



A	B	Q
0	0	0
0	1	NC
1	0	NC
1	1	1

* NC: No Change (previous state is maintained)

Figure 2.6: Schematic view of a delay insensitive circuit called Muller C–element. If binary value is same on both inputs, it appears as output else a transition on a single input retains previous output state.

2.2.2 Quasi–Delay Insensitive (QDI) Designs

A Quasi Delay Insensitive (QDI) circuit is a Delay Insensitive circuit except that the wire delays are not unbounded, rather they are arranged as isochronic forks with bounded skews. Isochronic forks are forking wires where the difference in delays between destinations is negligible [AJM89]. In case of DI circuits, the delays on the fork wires are completely independent of each other and may vary considerably. Martin and Van Berkel have extensively described the advantages and disadvantages of QDI circuits [AMJ90, KVB92].

2.2.3 Speed–Independent (SI) Designs

A speed independent circuit operates correctly regardless of gate delays while the wire delays are so negligible that they are ignored. SI circuits were introduced by David Muller in 1950s [REM65].

2.2.4 Self–Timed Designs

Speed independent elements i.e., circuits with negligible delays or circuits with well bounded wire delays communicate delay insensitively with each other to form Self–

Timed systems. Each element is considered to be in an equipotential region. Self-timed circuits are described by Seitz [CMD80].

2.2.5 Micropipeline

Micropipelines were introduced in Ivan Sutherland [IES89] primarily as an asynchronous alternative to synchronous pipelines. A micropipeline defined as an event based elastic pipeline with or without processing is shown in Figure 2.7.

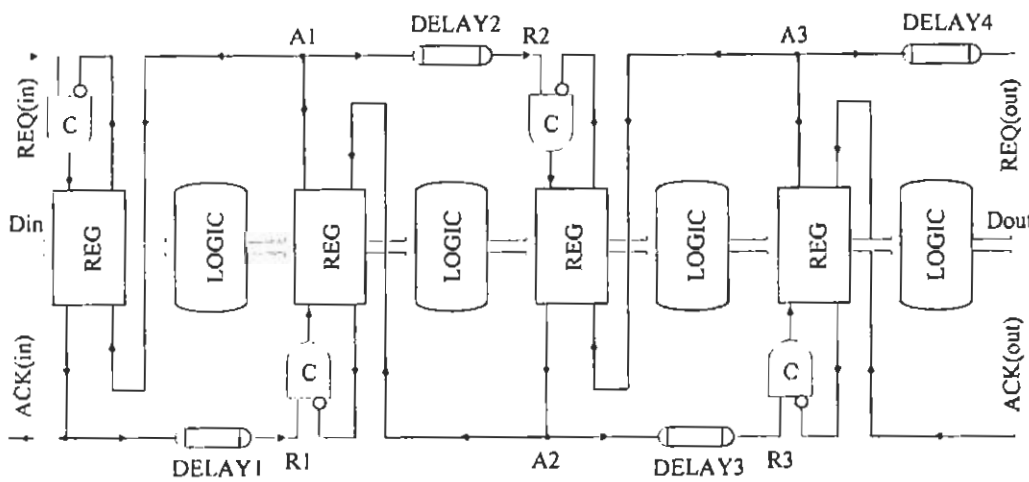


Fig. 2.7. Shows the basic concept of micropipeline with processing [IAS89].

Micropipeline is categorized as a delay-insensitive methodology with bundled data. The timing constraints in this system are not simply bundled data, but timing of all computation elements including the communication interfaces need be considered. That is the reason why micropipeline implementation is associated with full custom designs to conserve timing constraints. Since a micropipeline shares features of both delay-insensitive and bounded-delay circuits, it can be considered as a class of its own in the asynchronous hierarchy. The following features are characteristic to Micropipelines:

- Signaling protocol
- Bundled Datapath
- Delay Elements

There are two basic signaling protocols fundamental to micropipeline i.e. 2-phase and 4-phase [IAS89, TEW91, CSC01, OAP97]. Sutherland's micropipeline is an example of 2-phase micropipeline, where special event–controlled registers are used for inter–stage latching and Muller C-elements are used for the micropipeline control (as discussed earlier, C-elements are DI circuits). Delay pads equivalent to the time taken by the respective logic unit to generate the desired result on the datapath are inserted to bundle the data. In fact, these delay pads define the request (control) cycle at each stage and the time difference between the edges of the request and the acknowledge signals.

The micropipeline model used by Furber in the implementation of Amulet2e [SBF97] and upgrades is a 4-phase model [SBF96], characterized by design simplicity and reduced real-estate of control circuitry resulting in power efficiency.

2.3 Evolution of Asynchrony

Asynchronous processing has always been an integral part of computing. Asynchronous designs have been successfully applied to control oriented applications (such as chip interfaces [AYV95, LLO93], bus controllers [KYY95], cache controllers [SMN93], and network communication controllers [WSC93, MBJ92]), datapath components (such as adders [AJM92, JDG93], multipliers [ADA94], and dividers [TEW91a], as well as general purpose microprocessors.

The dawn of event-based computing saw small asynchronous modules available for integration with other modules to form a computing system. Asynchronous building blocks such as registers, adders, memories, and control devices, called macromodules [WAC67, WAC73], were designed and individually tested to construct arbitrarily large and complex systems during 1960s and 70s at the Washington University.

Data Driven Machine (DDM) [ALD78] consisting of wire-wrapped boards and back plane was the world's first operational dataflow machine designed at the University of Utah during 1978 and 1982. Its self-timed operation used Huffman- style state machines [REM65] for control.

The Caltech asynchronous processor [AMN89] developed during 1988 and 1989 was the first asynchronous processor from the VLSI era. It was not meant to be innovative. Rather it was supposed to provide a proof–of–concept for Caltech asynchronous design style. Caltech Asynchronous Processor was a simple 16-bit processor with two-stage pipeline. It was constructed using 1.6-micron CMOS technology and had 20,000 transistors on board a 6.6x4.6mm die. It executed 18 million instructions per second, a very high performance for a processor of that era.

The concept of micropipelines was introduced by Sutherland [IES89] in 1989 that led to various micropipeline based asynchronous processing applications.

Research in asynchronous systems was driven by industry requiring speed and power conservation in the 1990s that continues to the day. Some of the famous research projects are STRiP of Stanford University [MED92], NSR by the University of Utah [EBD93], Counterflow Pipeline Processor by Sun Microsystems [RFS94], Amulet(-1, -2, -3) of the University of Manchester [SBF94, SBF97, SBF98, SBF02], Fred of

the University of Utah [WFR96], Titac(- 1, -2) of the Tokyo Institute of Technology [ATA97][TYA99], MiniMIPs by Caltech [AJM97], Async 80C51 by Philips [HVG98], Async DDMP: Joint venture of Sharp Corporation, Osaka University and the Kochi University, Japan [HTA99] and FLEETzero Asynchronous Project by Sun Laboratories [CEM95, CEM97, IES01, WSC01].

The Amulet series of microprocessors was developed at the University of Manchester to demonstrate the feasibility, desirability and practicability of employing asynchronous techniques in embedded applications. The main goal was to reduce the power consumption. Asynchronous architecture was used as a means to that end. Amulet series had a micropipelined organization. The research continues to this day.

Sun Microsystems developed a proof-of-concept chip called FLEETzero [WSC01], with a radically new architecture. The conventional synchronous processors are designed in terms of operations such as addition, division and I/O. This operations-centric view was correct when cost of logic was more than communication, not only in financial terms, but also in terms of delay, power consumption and volume. Recently, the cost of logic has plunged, leaving communications as the high cost task. Today the task of getting two numbers to an adder takes more chip area, consumes more energy, and takes longer than doing the addition. But the operation-centric design remains dominant, in conventional designs.

The communication-centric design focus of the FLEETzero chip concentrates on how the data moves through the circuit. The arrival of data triggers the operations. In the FLEETzero chip, each processing element, called a "ship", performs its own function at its own pace. Binary code routes data from ship to ship. This lets the programmer

specify a sequence of data movements, so that operations become side effects of where the program sends data. Consequently, only the portions of the circuit that are required for computation are actually active and the rest stay in sleep mode to conserve power.

2.4 Reconfigurable Mediums

As the size of applications developed on the general–purpose computing machines grew, speed became a major issue. Various endeavors to enhance speed included the development of Application Specific Integrated Circuits (ASICs), performing specific tasks with greater efficiency [MJS97]. These ICs were especially popular with distributed computing setups, as they in conjunction with each other, resulted in better performance.

With the advent of mobile computing [MSN97], real estate also became an issue of significant importance. Independently efficient ASICs, integrated to form efficient setups took too much of space to be part of hand-held mobile devices. This gave birth to a new field called Reconfigurable Computing [KCN02], in which the ASICs had the capability of getting reconfigured to suit the requirements of multiple environments without affecting the real–estate. Although programmable devices such as Programmable Array Logic (PAL) and Programmable Logic Array (PLA) existed since the 70s, they were only used to eliminate the hard-wired glue logic and the timing mismatches associated with unequal fork legs in its implementation [MJS97]. With the passage of time, the programmable devices started increasing in size and functionality and today we live in the era where microprocessors are sold as open cores, implementable in reconfigurable devices. Another very important issue

effecting the overall cost and turn around time of a product, is prototyping. Reconfigurable mediums have played a vital role in prototyping because of their reprogrammable nature. As shown in Figure 2.8, reconfigurable devices give the performance of ASICs with the flexibility of General Purpose ICs (GPICs) [SHK98].

In order to fully understand the characteristics of reconfigurable devices, a hierarchy of devices [MJS97] used in various modes of computing, must be kept in mind. Such a hierarchy chart is presented in Figure 2.9. Some important features of today's more complex reconfigurable devices like FPGAs can be traced back to the Simple Programmable Logic Devices (SPLDs). The internal structure a generic PAL9V6 is shown in Figure 2.10 for demonstration.

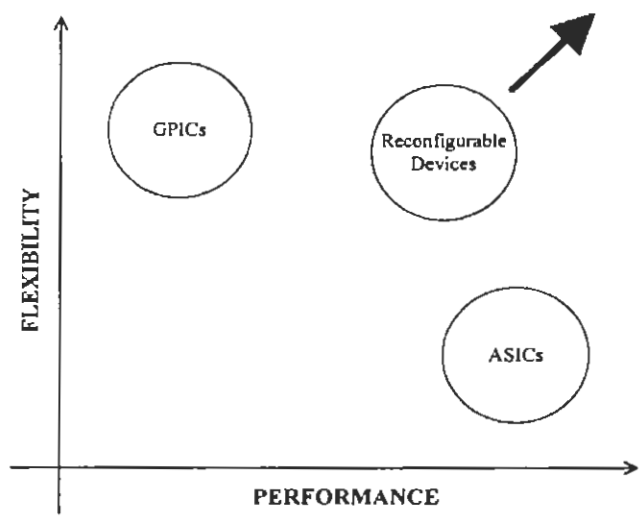


Figure 2.8: Comparison of GPICs, ASICs and reconfigurable Devices on the performance vs. flexibility scale.

Any logic circuit can be written in Sum-Of-Products (SOP) form. Left hand side of the truth table describing the circuit is nothing but an implementation of minterms with AND gates. The right side of the truth table represents the summing of these

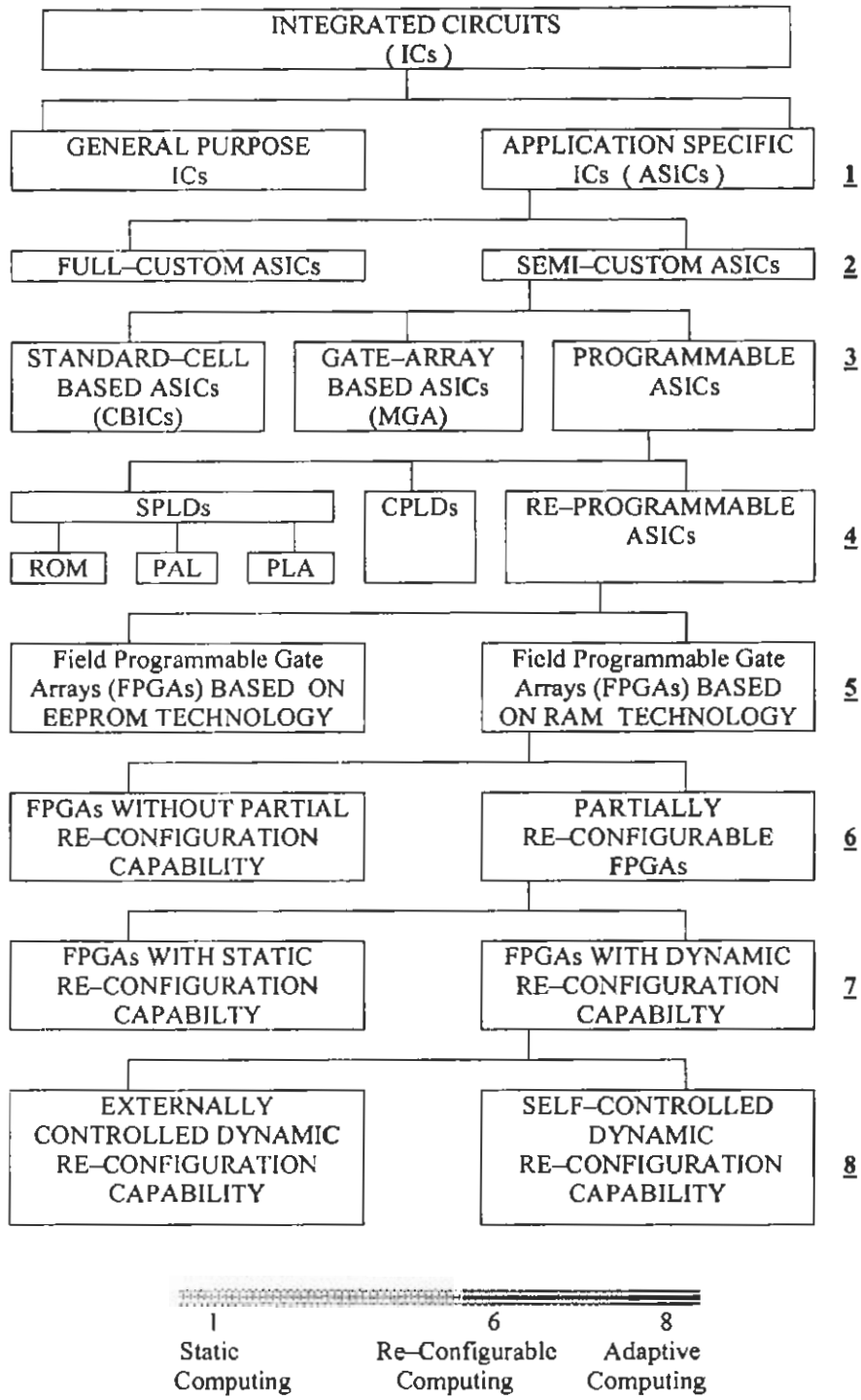


Figure 2.9: Hierarchy describing devices used in various modes of computing.

terms with OR gate. Programmable Array Logic (PAL) type devices are characterized by a fixed OR–array and a programmable AND–array for the implementation of the circuit equation in SOP form. The output can, however, be manipulated. If the

resulting output moves out of the devices without changing the logic level, then such a PAL is H-type device. If the resulting output is inverted in logic before being ejected from the device, then PAL is L-type.

However, both L and H type PALs can only be used for the implementation of combinatorial logic. A D-type latch can be placed at the output within the device to make it compatible to the sequential logic. Such a PAL would then be called an R-type PAL. The Evolution of R-type PAL meant that counters could be implemented in it. But the counter equations consist of feedbacks. For an R-type PAL output to be fed back to the input, wires external to the device had to be used, that were inefficient. Therefore, R-type PALs were designed to have local feedbacks within the device, for the implementation of efficient feedbacks.

Requirement of different type of PALs for the implementation of combinatorial (positive / negative logic) and sequential logic has the potential to increase the real estate of a design, by possibility of having unused resources in the different types. This problem can be eliminated by using a V-type PAL, whose outputs can be configured with the help of fuses to deliver in the L, H or R mode as per requirement. The output is manipulated in an additional section of the device called OLMC (Output Logic Mega Cell), as shown in Figure 2.10. Important thing to note is that result is generated in both combinatorial and sequential forms by letting it move through combinatorial and sequential paths. A multiplexer decides which result moves to the output pad. The multiplexer select signal is controlled by a fuse.

PALs are single shot devices based on fuse or anti-fuse technology [MAJ97]. They are field programmable but not reprogrammable. A UV-PAL can be used to introduce the element of re-programmability, as it can be erased by UV exposure.

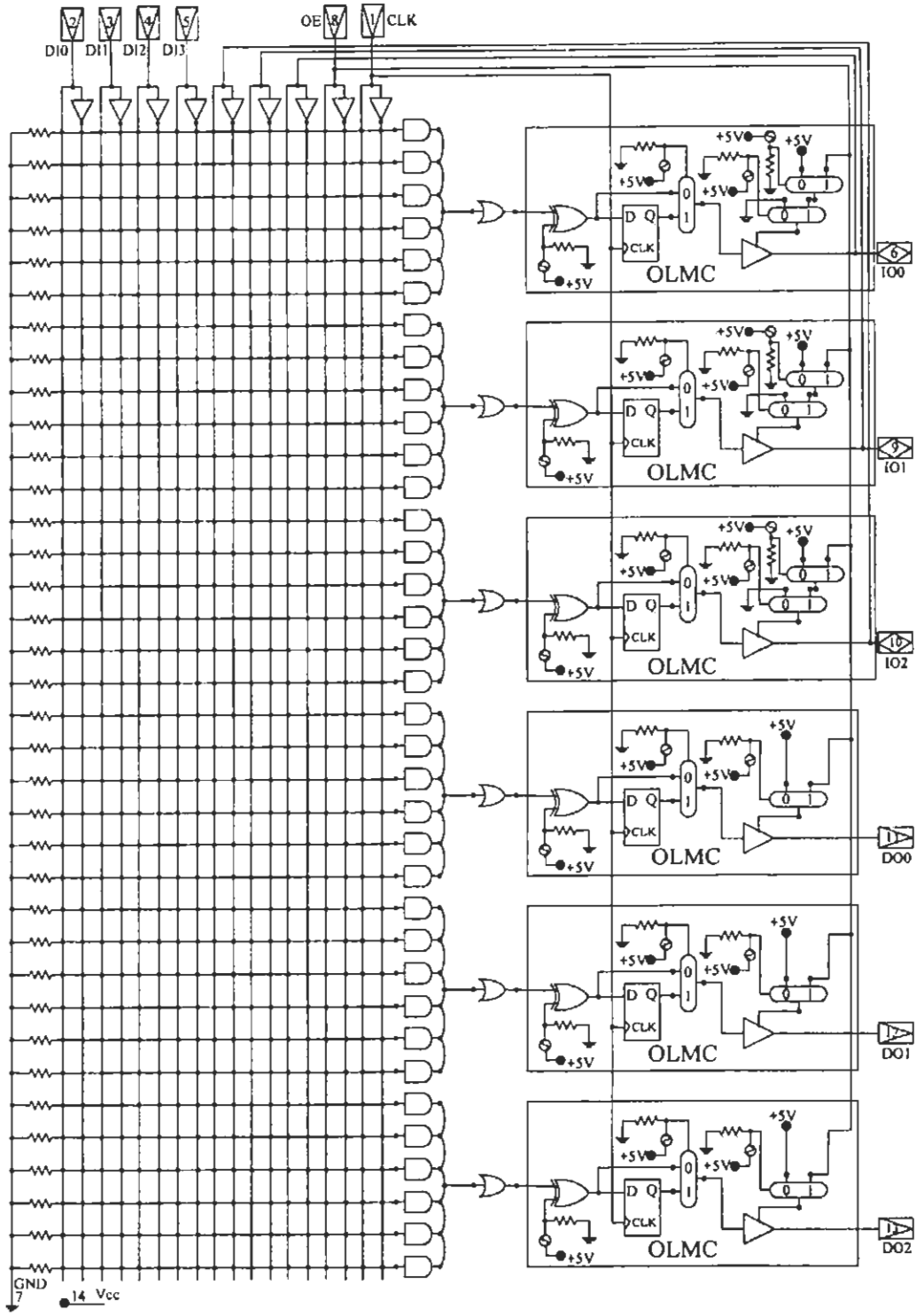


Figure 2.10: Internal structure of generic PAL9V6

However, these devices cannot be labeled as reconfigurable, as they need to be pulled out of the circuit to be erased and require programming voltages higher than the operational voltages. GAL is a variant of PAL that is non-volatile and has the capability of in-circuit reconfiguration at operational voltage.

The circuit implementation capacity of PALs / GALs is dependent of their IOs. The implementation of a large circuit therefore, requires cascading of these devices. The circuit has to be broken into smaller circuits suitable for the PAL size and then after implementation, these PALs can be cascaded to get the result. For different circuits, the IOs between different PALs / GALs will vary in the cascaded circuit. So if the cascaded structure is moved to a single IC platform it will require interconnects to be reconfigurable. Such a device with cascaded SPLDs, communicating with each other through reconfigurable interconnect structure called Programmable Switch Matrix (PSM) the Complex Programmable Logic Device (CPLD). The feedbacks within SPLDs constituting a CPLD act as local interconnects, while interconnects associated with PSM can be labeled as global interconnects of the device. A Generic structure of a CPLD is shown in Figure 2.11.

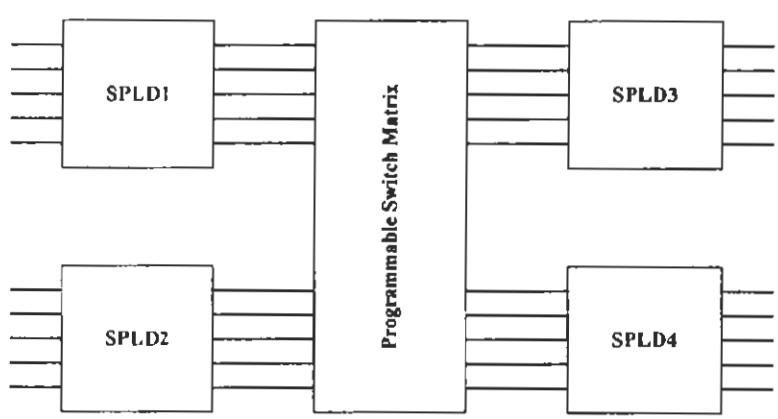


Figure 2.11: A generic structure of CPLD showing SPLDs cascaded through Programmable Switch Matrix (PSM).

FPGAs constitute the next higher level of reconfigurable devices. FPGA families are generally SRAM based as shown in Figure 2.9 i.e., their configuration memory is composed of SRAM cells and are, therefore, volatile. The two major FPGA structures can categorized as MUX based like the FPGA families by Actel and LUT (Look-Up Table) based like Xilinx and Altera FPGA families [MJS97]. Logic is implemented in the FPGAs in Logic Elements (LEs), Configurable Logic Blocks (CLBs) and modules for Altera, Xilinx and Actel, respectively. A logic equation can be implemented in an Actel module with the help of Shannon's Expansion theorem and binary trees, while the output of a truth table can directly be considered an LUT in case of Altera and Xilinx. Figure 2.12 explains the difference between design implementation style of MUX and LUT based FPGAs.

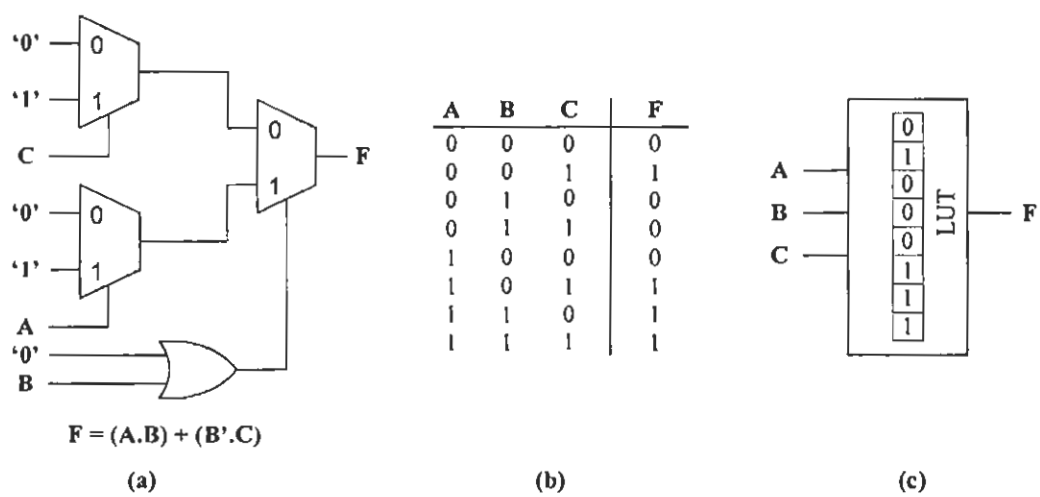


Figure 2.12: (a) Implementation of logic in a MUX–based FPGA, (b) Truth Table of logic to be implemented, (c) Implementation of same logic in LUT–based FPGA.

In the presented research LUT based FPGAs are used therefore Altera and Xilinx FPGAs need further explanation. Function *F* in Figure 2.12, after emerging from the LUT based function generator moves through the combinatorial and sequential paths

as in SPLDs, before one of the results is selected to leave LE in case of Altera and CLB in case of Xilinx. Eight LEs constitute a Logic Array Block (LAB) [FLX10]. Within a LAB, local interconnects exist that are fast and are used to cascade fragmented logic in the LEs. The same local interconnects are used for the implementation of feedbacks also. Xilinx FPGAs have the CLBs arranged in matrix formation [XXC30, VTX25]. Direct interconnects (like the local interconnects in Altera) exist between neighboring CLBs in Xilinx. Other global interconnects are spread around FPGA to connect different parts of the circuit. Extensive discussion of the Sequential and combinatorial paths of logic elements and the local interconnects will be presented in Chapter No. 3, to explain the behavior of technology independent implementation of SIRO.

FPGAs are associated with programming environments that hold the information on these FPGAs in their libraries. The main responsibility of these environments is to convert a design available in Hardware Descriptive Language (HDL), Schematic or any other acceptable form, to a configuration file that can be loaded into the configuration memory of the device. The environments make use of stochastic processes and various algorithms to reduce the entered design, map it to the device resources by effective placement and routing of sub circuits. Sometime the resulting scheme may not satisfy the design constraints, in which case an altered design or manual adjusted in placement and routing is necessary to achieve the desired results. Simulations are necessary each step of the ways to know which step requires amendment. Simulations are also important to verify compatibility with user-defined constraints before the downloading of configuration into the device. Two main categories of simulation are the functional simulation done in the beginning of the

process to ensure the functional correctness of the entered design. This kind of simulation is device independent as it only checks functionality with the help of test vectors to the equations. The more important kind of simulation done, just before the configuration is downloaded into the device is called post-layout simulation, which is device dependent and incorporates the delays of the device, found in the library database, for the manipulation of results, to verify the timing constraints. Figure 2.13 shows the various steps associated with FPGA programming environments.

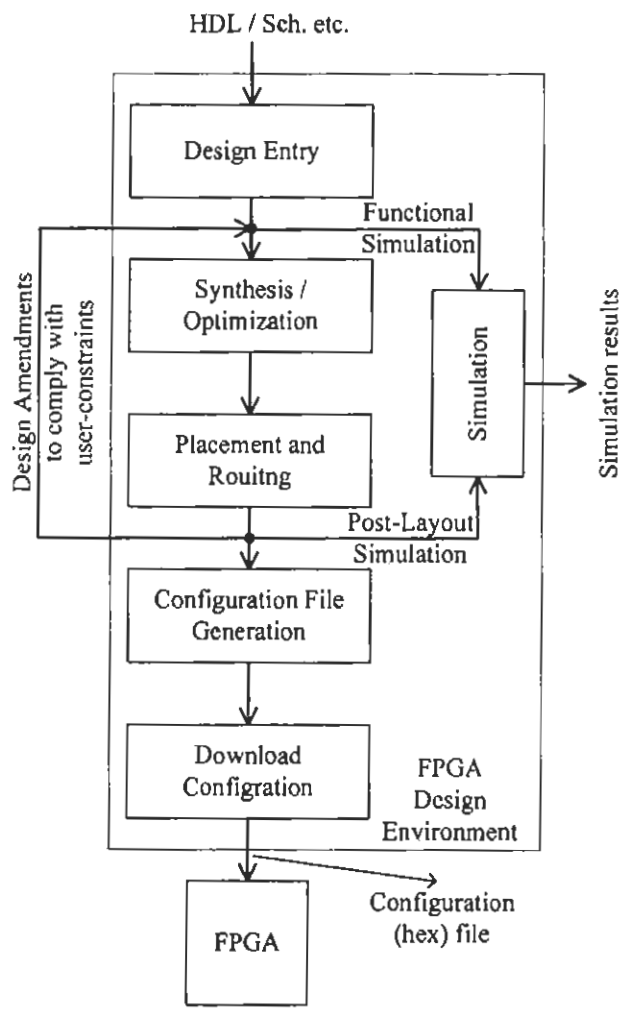


Figure 2.13: FPGA design environment and the associated design flow.

A FPGA can be visualized as having two layers, a real one and a virtual one. The real layer consists of the FPGA resource layout, while the virtual layer consists of SRAM cells used for configuration [XXC30]. Systems implemented with FPGAs can make use of their reconfigurability in one of two ways: Compile-Time Reconfiguration (CTR) or Run-Time Reconfiguration (RTR) [BLH95]. CTR systems do not change the FPGA's configuration for the life-time of the application. RTR systems change the FPGA configuration during the course of operation, either by full reconfiguration [JMD00, DRS93] or partial reconfiguration [JDH95, SMM00]. Dynamic partial reconfiguration allows an FPGA to implement multiple functions and to change those functions while the system is running.

Therefore, the reprogrammable nature of FPGAs in CTR mode makes them ideal for low cost and fast prototyping, while the RTR makes them suitable hardware implementations designed to support applications performing variety of tasks in real estate and power controlled environment.

3

Single Inverter Ring Oscillator

A standard ring oscillator consists of an odd number of inverters connected in a feedback loop. The odd count results in a self–oscillating circuit. A simple Single Inverter Ring Oscillator (SIRO) as shown in Figure 3.1(a) is an inverter whose input and output are connected, so that the gate and wire delays of this circuit account for the cycle time of oscillator.

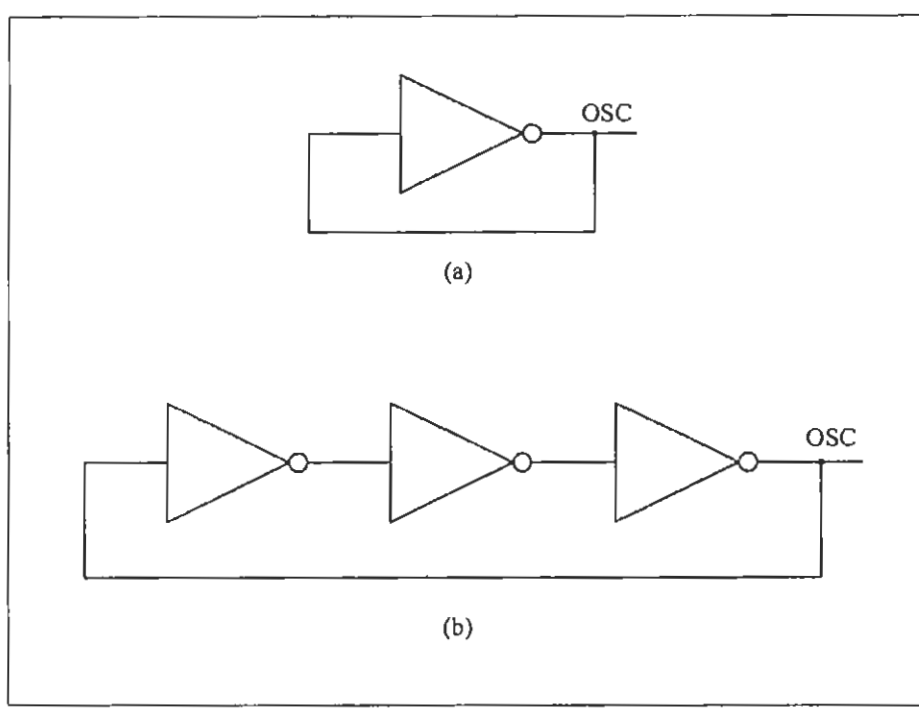


Figure 3.1: (a) single inverter ring oscillator, (b) tri inverter ring oscillator

Any odd number of inverters can be placed in series following the same pattern to implement oscillators with lesser frequencies. Very commonly used is Tri Inverter Ring Oscillator (TIRO), as shown in Figure 3.1(b).

Due to its integrated nature, a ring oscillator can be used in a variety of applications, such as clock recovery circuits, on-chip clock distribution schemes [TMI04, RFR98, RWR87, DLLS2] and as sensor [SLB02, NDJ00] where the frequency of oscillator varies with the parameter to be sensed. These applications of ring oscillator are full–custom. Ring oscillator application does exist in FPGAs within the Delay–Locked Loop (DLL) [DLLS2] but this is also a full–custom implementation.

Schematic diagram of a GaAs–based Direct Coupled FET Logic (DCFL) TIRO is shown in Figure 3.2. Simulation shown in Figure 3.3 was done in AIM-Spice ver.3.08f, using Shur's unified extrinsic model for uniformly doped N–channel MESFET [KLE93]. According to the simulation results this full–custom ring oscillator operates at 41.5GHz.

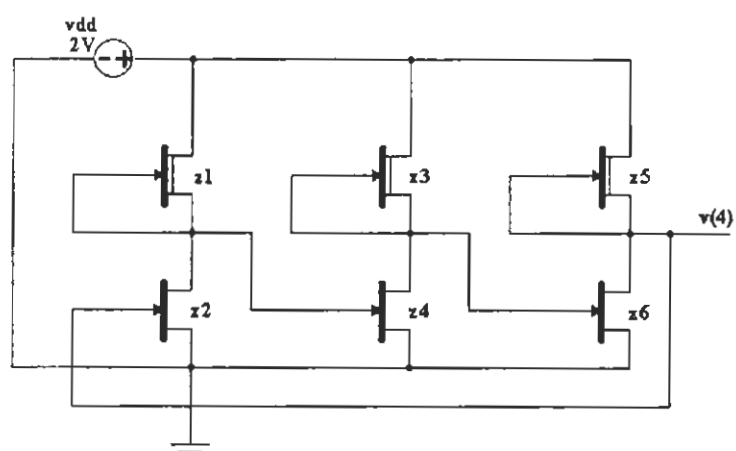


Figure 3.2: Schematic diagram of GaAs based DCFL tri inverter ring oscillator.

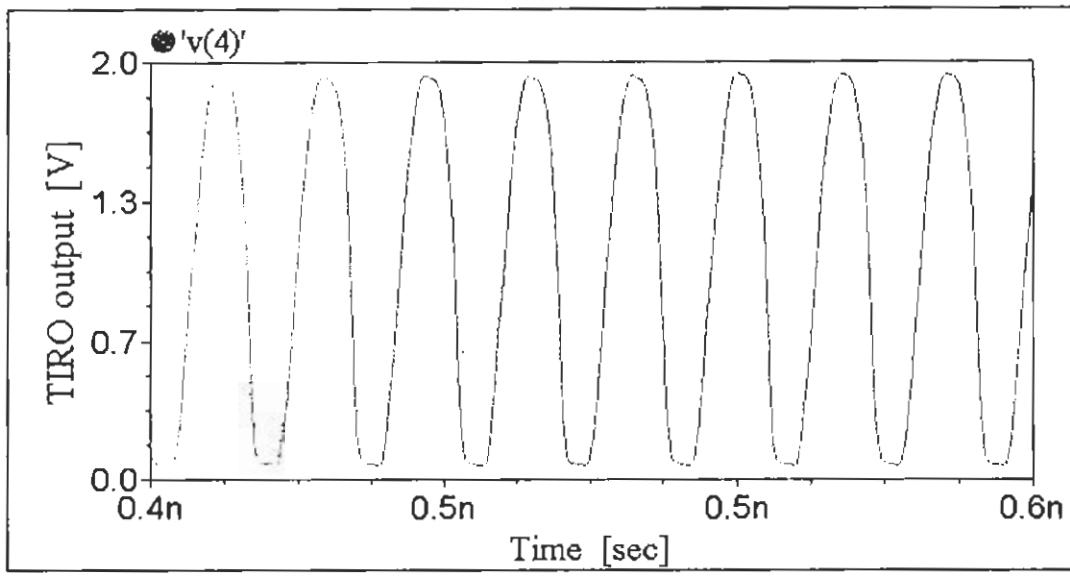


Figure 3.3: Simulation of TIRO done in AIM-Spice.

Now consider implementation of a ring oscillator in a reconfigurable medium such as FPGA using HDL. A SIRO model in Verilog HDL, fails to produce functional simulation results as $A = \sim A$; makes no sense, yet when the same code is synthesized and implemented in an FPGA, one can observe a stable oscillatory signal in its post-layout simulation. SIRO implementation presented in this thesis, consists of a simple 2–input Nand gate, as shown in Figure 3.4(a).

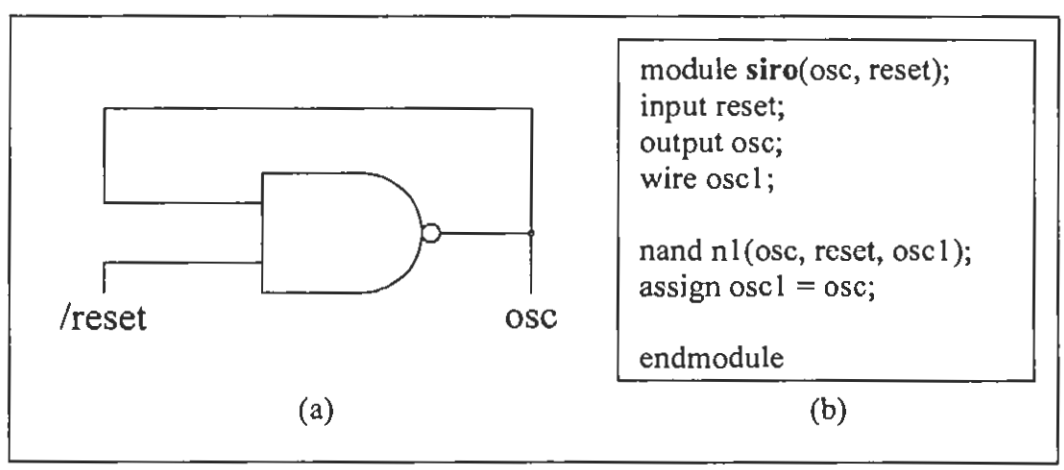


Figure 3.4: represents (a) Schematic diagram of SIRO, (b) Verilog model of SIRO

Figure 3.4(b) shows the Verilog model for SIRO, where the Nand gate, has its output '*osc*' looping back to one of its inputs. The other input of the Nand gate is the global, negative logic 'reset'. As a result, '*osc*' remains high when '*reset*' is low and toggles when it is high.

The result is an oscillator with frequency far exceeding the maximum frequency specifications of the FPGA used to implement the design. The maximum frequency of the FPGA, specified in its data sheet [FLX10, VTX25] is associated with the setup and hold time requirements of its sequential elements. The reason why SIRO frequency by-passes the maximum operational frequency, is based on the internal structure of FPGA [FLX10, VTX25, XXC30]. A typical structure of logic element in an LUT–based FPGA is shown in Figure 3.5. The inputs to the LE are manipulated by the LUT to generate an output, that passes through two paths; sequential containing D-flip flop and combinatorial. A multiplexer controlled by an SRAM configuration bit decides the path that can proceed out of the LE. The HDL code is responsible for the setting or resetting of the Multiplexer related SRAM bit.

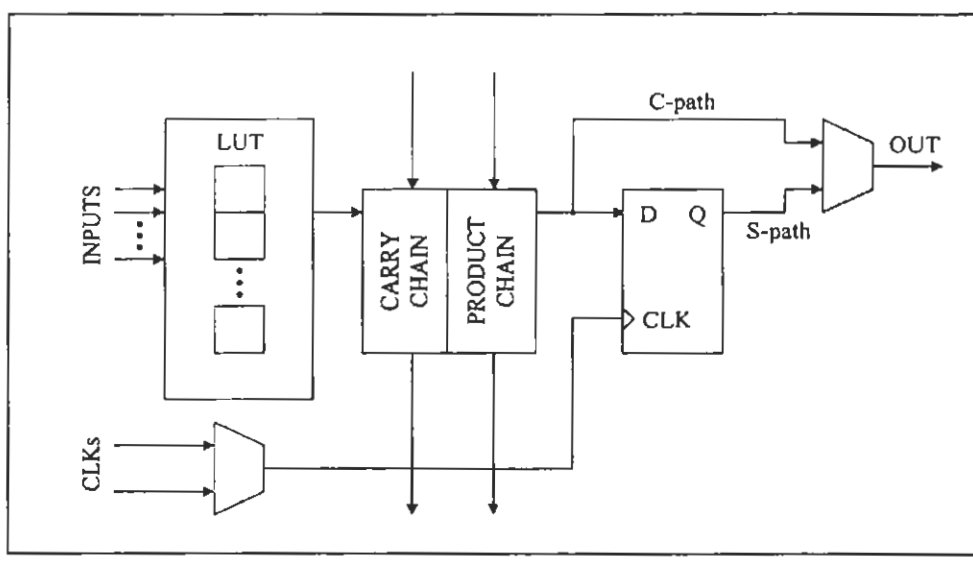


Figure 4.5: A typical Logic Element (LE) of an LUT–based FPGA.

The self–starting SIRO presented in Figure 3.4, fits into a single LE following the combinatorial path shown as c-path in Figure 3.5. Figure 3.6 shows the fast interconnects of FPGAs. Altera model is presented in Figure 3.6a, where group of LEs form a Logic Array Block (LAB) and the output of each LE can move to the global as well as local interconnects that establish fast feedback to the inputs of other LEs in the LAB. Figure 3.6b shows the Xilinx model, where the output of CLBs can connect directly to the global, as well as direct interconnects that provide a fast and direct link to the neighboring CLBs. There are also feedback loops with LEs and CLBs that are direct and fast [FLX10, XXC30].

The fast interconnects (direct / local) establish the feedback ring of SIRO. As a result delay in the combinatorial path plus wire delay associate with fast local interconnects of LE separate *A* from *~A*, producing a high frequency oscillator, with frequency greater than the setup and hold time requirement of the latch found in the sequential path presented as s-path in Figure 3.5. As SIRO, independent of the FPGA programming environment, always consumes a single LE, this frequency is always optimal.

For SIRO to meet the setup and hold time requirements FPGAs' sequential elements, Latch Synchronizing Circuit is added to the oscillator circuit to generate a Latch Synchronized Oscillator (LSO) called Asynchronous Clock (*ACLK*) in Figure 3.7. In the LSO Verilog model, as shown in Fig. 3.7(b), '*osc*' is prevented from triggering the LE latch, till its present state '*a1*' is opposite to next state '*a2*', as the output of latch '*ACLK*' is fed back to its input after inversion, to generate a T-flip flop.

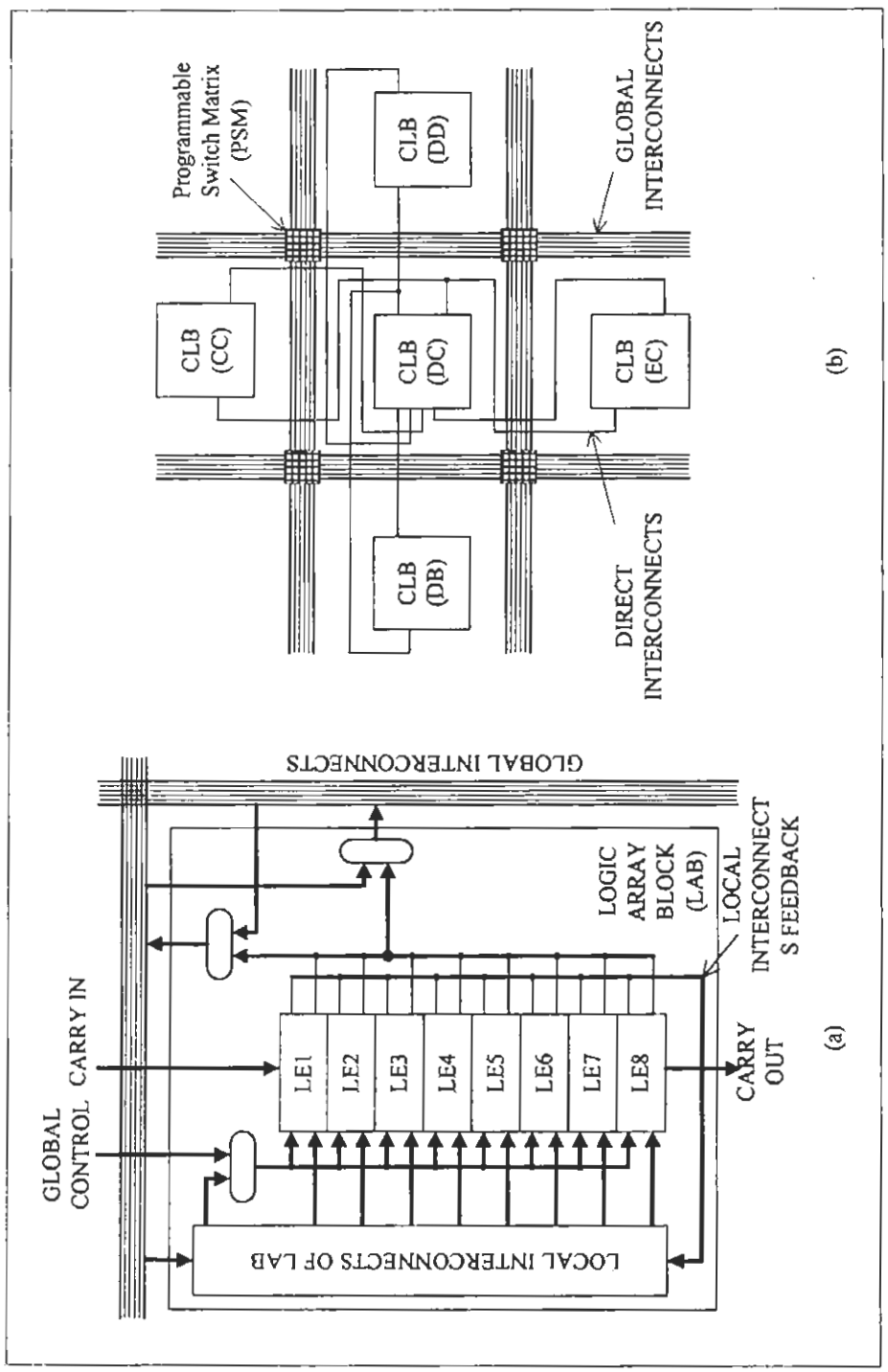
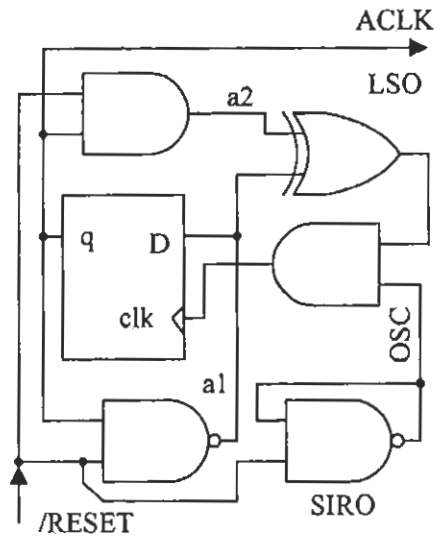


Figure 3.6: shows LUT based FPGAs: (a) local interconnects (Altera), (b) direct interconnects (Xilinx)



(a)

```
module lso(osc, aclk, reset);
input reset;   output aclk, osc;
reg aclk1, a1, ack;  wire osc, a2;

siro s1(osc, reset);

always @ (posedge osc or negedge reset)
begin
      a1 = ~aclk;
      if (osc == 1)
      begin
            if (reset == 0)
            begin
                  aclk1 = 0;
                  a1 = 1;
                  ack = 0;
            end
            if (ack == 1) aclk1 = ~aclk;
            if (a1 != a2)       ack = 1;
            else  ack = 0;
      end
end
assign a2 = aclk1;
assign aclk = aclk1;
endmodule
```

(b)

Figure 3.7: (a) Functionally equivalent schematic diagram of LSO, (b) Verilog code for LSC.

As in case of T-flip flop, the present state and next state only complement upon the satisfaction of setup and hold time requirement, the '*ack*' signal permits triggering only when $a1 \neq a2$. This automatically generates a toggling '*ACLK*' signal, that complies with setup and hold time requirement of LE latch. As the technology is uniform over the entire FPGA, SIRO adjusted to Latch Synchronized Oscillator (LSO) for one latch, is compatible to the sequential elements over the entire chip. '*ACLK*' may further be divided as per the requirement of circuit. Figure 3.8 shows *SIRO* (OSC), *LSO* (ACLK) and LSO divided by a factor of two (*DIV2*) and eight (*DIV8*).

3.1 SIRO as on–chip clock source

The property of SIRO circuit that it adapts to the FPGA technology by varying its oscillation frequency is considered while using it as an on-chip clock source to drive co-existing synchronous circuits. Same SIRO Verilog code, without any modification was synthesized and implemented in various FPGAs, to demonstrate self–adjustment of the circuit proportionate to the maximum frequency specification of the device. In fact LSO frequency was found to be the same as the maximum specified value. Table 3.1 presents the observed values of SIRO, LSO, DIV2 and DIV8 for XC2S400E-7FG456 and XCV200E-8FG456 devices, emphasizing the concept of technology independence through dynamic calibration or adaptation.

The SIRO circuit is stable enough so that the FPGAs can be identified with their SIRO number; an entity measured on the time scale, directly proportional to the maximum frequency factor of the device.

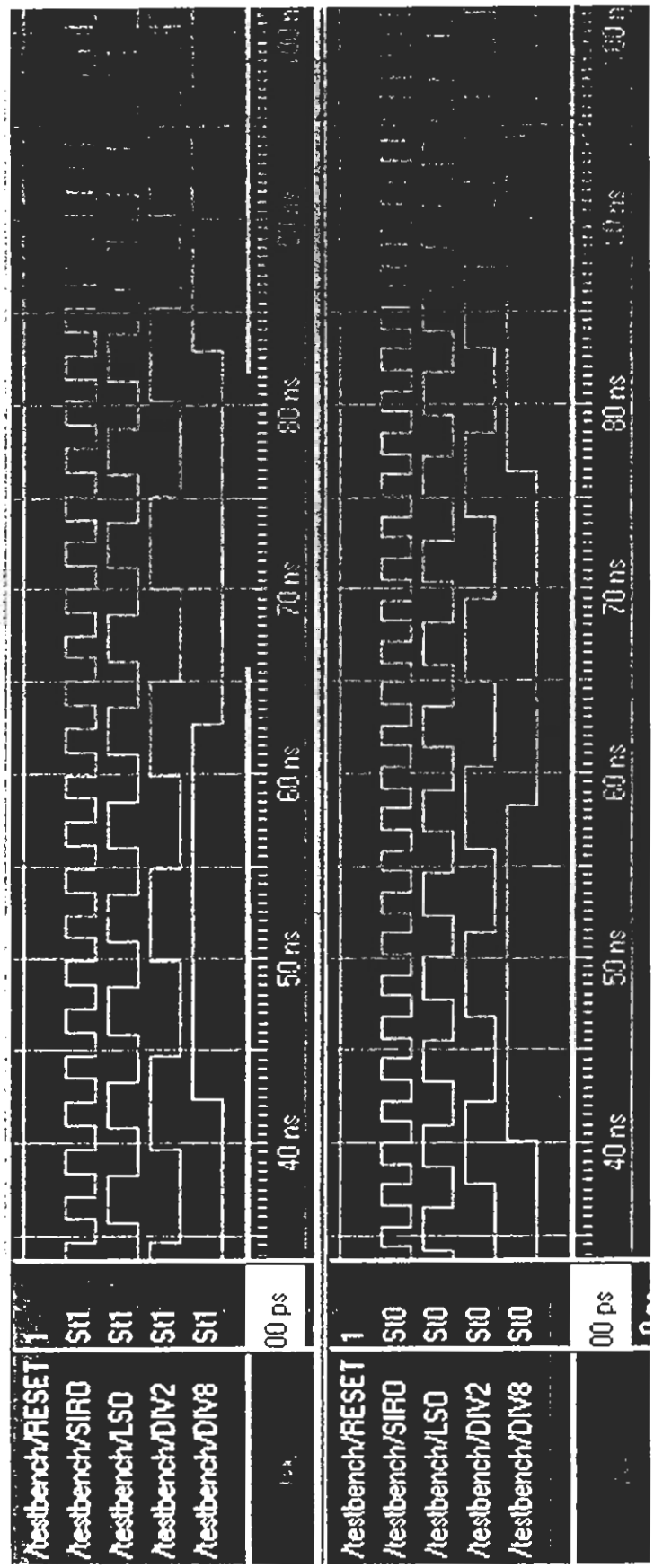


Figure 3.8: SIRO, LSO, DIV2 and DIV8 implementation in FPGAs: (a) XC2S400E-7FG456, (b) XCV200E-8FG456

TABLE 3.1

SIRO, LSO, DIV2 (LSO divided by 2) and DIV8 (LSO divided by 8) values for different devices showing technology independence of circuits.

DEVICE		SIRO (nS)	LSO (nS)	DIV2 (nS)	DIV8 (nS)
(a)	XCV200E-8FG456	2.26	4.52	9.04	36.16
(b)	XC2S400E-7FG456	2.53	5.06	10.12	40.48

Another set of readings was taken after letting the clock signals (external and internal) ride the global clock–tree via Delay–Locked Loop (DLL) to enhance the fanout and reduce the skews in the clock signal [DLLS2, DLLVE]. CLKDLL macro provided by Xilinx was used to implement DLL. In order to connect internal clock to the '*clkin*' input of CLKDLL, a windows environment variable 'XIL_MAP_ALLOW_ANY_ DLL_INPUT' was set to '1'. It was necessary as the environment in normal mode only lets external clock signal riding the clock tree to use DLL macro. Whereas, SIRO acting as the clock source, is implemented on the same FPGA and is not required to utilize any of the FPGA IOs. Therefore, CLKDLL was forced to accept an internal signal at the input by setting the environment variable.

It was observed that the maximum external frequency locked by the DLL in an FPGA is approximately the same as the SIRO parameter for that device. This frequency is much greater than the maximum frequency specification for that device and if used to drive a synchronous circuit, causes setup and hold time violations. It was also observed that an external frequency, same as the LSO, when manipulated by CLKDLL, effectively drives a synchronous circuit without timing violations. This

frequency is the same as maximum frequency parameter specified in the datasheet for the device.

3.2 SIRO based Delay Element

SIRO can be used as a delay element with dynamic calibration capability. The adaptive nature of SIRO circuit already discussed in section 3.1 can be utilized in the generation of such an element, as Figure 3.9 explains. In this scheme, SIRO is controlled by LSC, which is used to drive an N-bit counter where the depth of the counter *N* is parameterized.

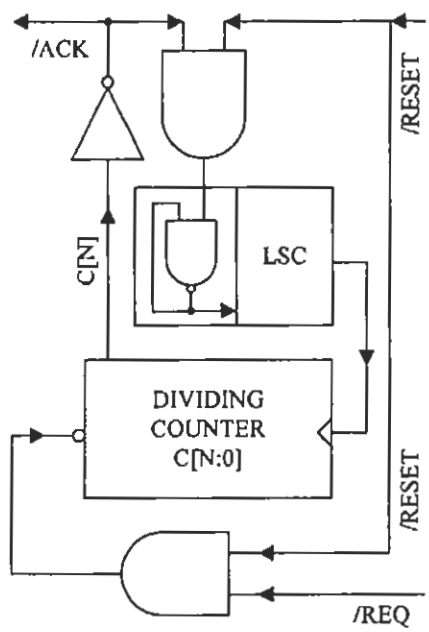


Figure 3.9: SIRO based delay element with dynamic calibration capability.

The N–bit counter basically divides the frequency of LSO to introduce the desired delay anywhere in the circuit. Setup and hold time violations do not exist in the counter circuit because of it being driven by LSO. The cleared counter is initially

triggered by a request signal and the Nth bit of the counter acts as an acknowledge signal. Therefore, the depth of the counter defines the delay between request and acknowledge while the circuit acts as a delay element.

As asynchronous systems including micropipelines depend critically on their delay model, the SIRO based technology independent delay element is widely used in this thesis to introduce desired delays. The delay element can be used in different stages of the micropipeline to insert required delay. A simple change in the parameter *N* of the N–bit counter, changes the value of delay to suit the requirements of any stage. Post–Layout simulation of the SIRO output (*OSC*), LSO (*ACLK*) and the 8–bit counter (*N*=3), for Altera's Flex10K and Xilinx XCV50e-8 is shown in Figure 3.10. It can be observed that '*reset*' triggers SIRO based '*osc*', that is adjusted by LSC to produce '*aclk*'. '*aclk*' is divided by 2 without timing violations to produce '*aclk2*'. '*count*' is the output of 3-bit counter driven by SIRO based oscillator. In Figure 3.10a&b, post-layout simulations in both Altera and Xilinx devices is presented to show the technology independent behavior of the developed delay element for FPGAs. The delay element adapts to the FPGA environment due to the frequency variation in SIRO i.e., the same counter driven by variable environment based frequency sets bit *N* after variable delay.

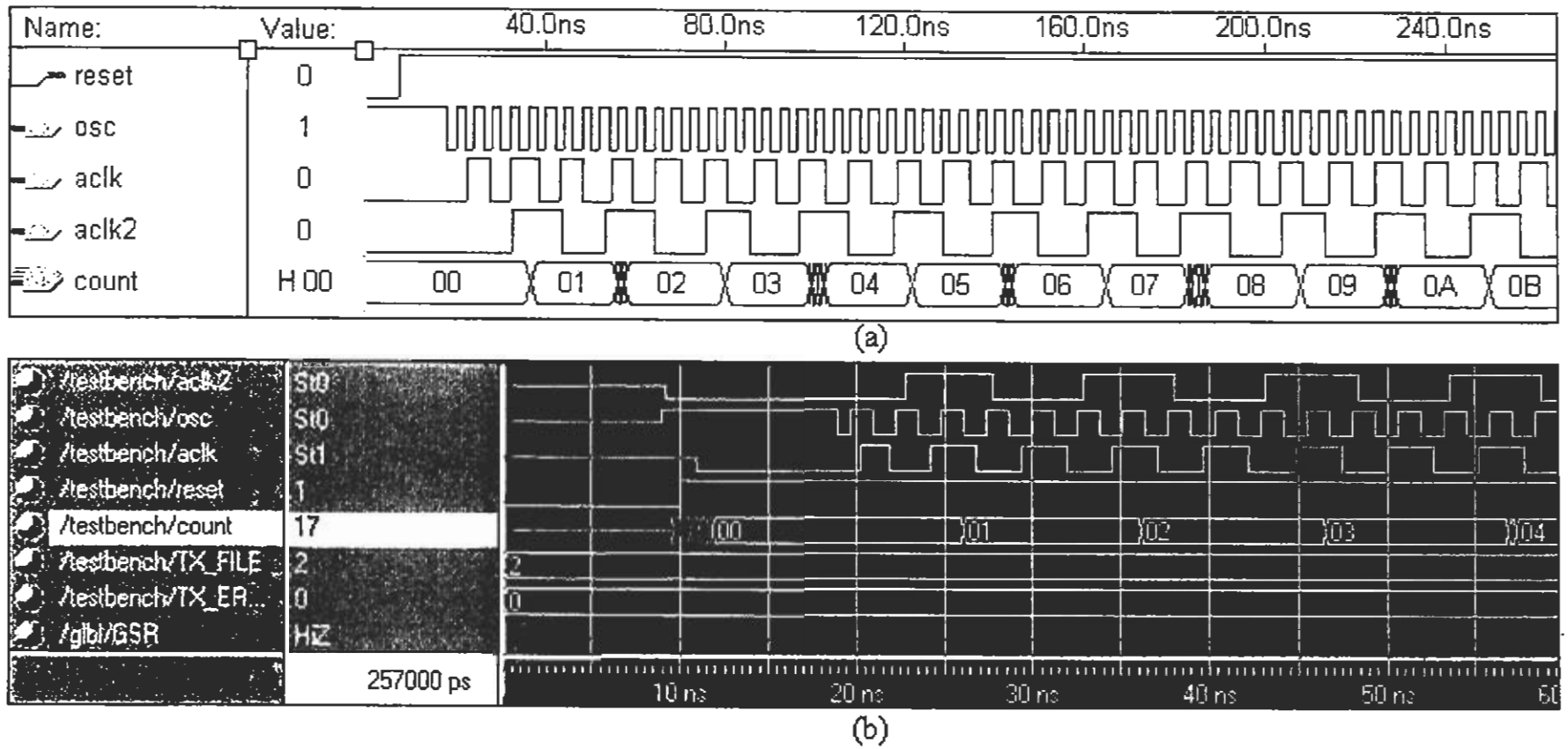


Figure 3.10: Post-layout simulation of SIRO output (OSC), LSO (ACLK) and an 8-bit counter, when implemented in:

(a) Altera Flex10K, (b) Xilinx XCV50E–8

4

Instruction Set Architecture

Instruction Set Architecture (ISA) for a simple 32-bit Load / Store RISC machine was developed, so that its various microarchitectures [DAP03, JBT99] are based on the techniques presented in this thesis. As the main focus of this thesis is not the development of a new kind of architecture, rather it is used as an application to support the presented concepts, therefore the ISA has intentionally been kept minimal, but complete [DAP03, JBT99], as shown in Figure 4.1.

The architecture defines an Integer Unit and a 16*32 bit register file. R0 is fixed at zero. Instruction and Data Memories consume the on–chip memory resources of the FPGA used in its implementation. Data manipulations are restricted to 32-bit only. It has four types of instructions, arranged in two groups (Group 1 and Group 2), details of which are presented in the following sections.

4.1 Instruction Set

Instructions belonging to Groups 1 & 2 are shown in Table 4.1. Being a tri-operand architecture, opcode of ADD has intentionally been kept zero, so that a 32-bit 'zero' instruction provides a no operation function (NOP: R0←R0+R0). Note that R0 is not writable in the register file.

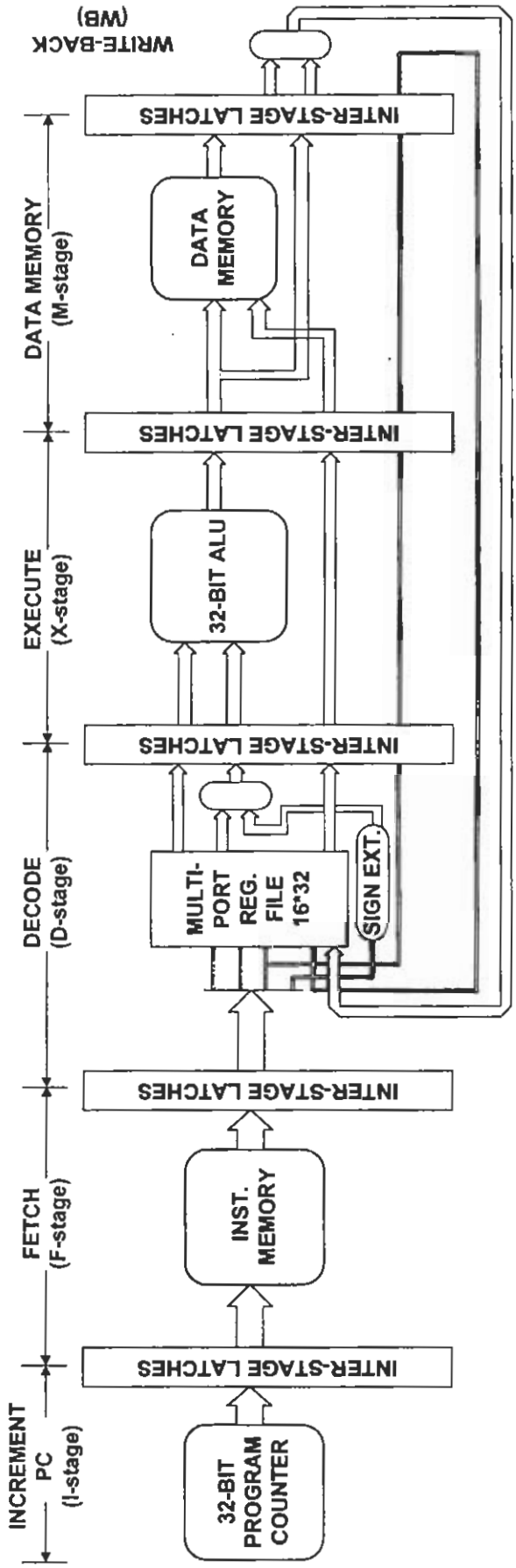


Figure 4.1: Pipeline of simple 32-bit RISC processor.

TABLE 4.1
Simple RISC Instruction Set

INSTRUCTION	FUNCTION	OPCODE	GROUP
ADD / ADDI	Add / Add Immediate	0000	1
ST / STI	Store / Store Immediate	0001	1
LD / LDI	Load / Load Immediate	0010	1
SUB / SUBI	Subtract /Subtract Immediate	0011	1
AND / ANDI	AND / AND Immediate	0100	1
OR / ORI	OR / OR Immediate	0101	1
XOR / XORI	XOR / XOR Immediate	0110	1
SHR / SHRI	Shift Right / Shift Right by Immediate	1000	1
SHL / SHLI	Shift Left / Shift Left by Immediate	1001	1
BR	Un-Cond. Branch	-	2
BNEZ	Branch if not zero	-	2
BEZ	Branch if zero	-	2

4.2 Instruction Types

Table 4.2 shows different instruction types in RMP. Register–Type (R-type) and Immediate–Type (I-type) instructions (Group 1) are characterized by bit 31 = 0. '0' at bit 26 for group 1 represents R-type, while '1' at this location for the same group represents I-type instruction. Conditional–Control (CC-type) and Direct–Control (DC-type) instructions (Group 2) have bit 31 = 1. '0' at bit 30 for group 2 represents CC-type while '1' represents DC-type.

TABLE 4.2
Simple RISC Instruction Types

REGISTER–TYPE INSTRUCTION						
0	Opcode	0	Rd	Rs1	Rs2	Unused
31	30 ... 27	26	25..21	20..16	15..11	10 ... 0
Format: ADD R1, R2, R3 RTL: R1 ← [R2] + [R3]						

IMMEDIATE–TYPE INSTRUCTION					
0	Opcode	1	Rd	Rs1	Immediate
31	30 ... 27	26	25..21	20..16	15 ... 0
Format: ADDI R1, R2, 0300 RTL: R1 ← [R2] + 0300					

CONDITIONAL–CONTROL INSTRUCTION					
1	0	T	unused	Rc	Offset
31	30	29	28..26	25..21	20 ... 0
Format: BNEZ R4, Loop RTL: If (R4 != 0) PC = PC + 1 + offset					

DIRECT–CONTROL INSTRUCTION		
1	1	Offset
31	30	29 ... 0
Format: BR Loop RTL: PC = PC + 1 + offset		

'Rd' stands for the destinations register; 'Rs1' for source register 1; 'Rs2' for source register 2; 'Rc' for condition register to be tested for validity of condition and 'T' for the type of condition; '0' for BEZ and '1' for BNEZ.

As simple RISC is a Load /Store machine supporting register and immediate modes, the ALU instructions must have at least one source coming from the register file (Rs1). The other source can be from the register file (Rs2) or can be an immediate value (IMM). The manipulated result is stored in the register file at location specified by Rd. In case of load instruction, Rs1 + (Rs2 or IMM), defines the effective data memory address, from where the data is loaded into Rd. In case of store instruction, Rs1 + (Rs2 or IMM), defines the effective data memory address, to which the contents of Rd are copied.

When the condition is true for CC-type instruction or in case of unconditional branch, the offset specificd by the control instruction is added to the Program Counter (PC) value and then loaded back into the PC. As the offsets and immediate values in casc of fixed instruction length architectures are always limited, so these values must always be sign extended prior to addition.

4.3 Pipeline

The program counter resets to zero which is the reset address and when the load control signal is set high upon the execution of a control instruction, it loads a new value else it increments.

Multi-port register file with read after write capability in the same stage is implemented to avoid structural hazards, because of the Write Back (WB) operation.

Decode stage, receives two destination register (Rd) values simultaneously, one from the current instruction reaching this stage and other, from a previous ALU or load instruction. A point to be noted here is that out–of–order execution is not supported in this simple RISC.

The Read After Write (RAW) type data hazards were avoided through forwarding using multiplexers at ALU input whereas control hazards were avoided by insertion of stall.

A branch, conditional or otherwise, is deciphered in the decode stage, by which time PC increments as if branch was not taken and points to the flowing instruction in the memory. If a branch is to be taken, load control signal originating from the decode stage orders PC to load branch offset and at the same time masks the unwanted fetched instruction to zero before it enters the decode stage. A 32 bit zero instruction in the decode stage is interpreted as a NOP, which disables the PC load control signal, active during the last cycle to load branch offset. If in case of conditional control instruction, condition is not met, load control signal does not become active and the instruction following the branch instruction is normally executed without stall.

5

Externally Clockless RISC

In this chapter a SIRO implementation in FPGAs, serving the purpose of on–chip oscillator driving co–existing clocked circuits is presented. The property of SIRO to adapt to the FPGA technology by automatic frequency adjustment always ensures optimal performance of synchronous circuit. Externally clocked circuits upon FPGA technology variation require a change in oscillating source for optimal performance, whereas SIRO–based on–chip oscillator provides platform independence, without power overhead. In addition, a clocked circuit driven by the on–chip oscillator has greater Electromagnetic compatibility (EMC), as compared to the externally clocked one. To support the claim, simple RISC presented in Chapter No. 4, was implemented in various FPGAs and was driven by external clock source. Then the same RISC was modified to incorporate on–chip oscillator as shown in Figure 5.1 and was driven by it. In case of external frequency source, oscillator had to be changed for optimal performance, while in case of on-chip SIRO–based design, optimal frequency was automatically achieved.

5.1 Implementation

Presented is the clocked (external oscillator) and clockless (SIRO–based on–chip oscillator) implementations of simple RISC in Xilinx XC2S400E-7FG456 and

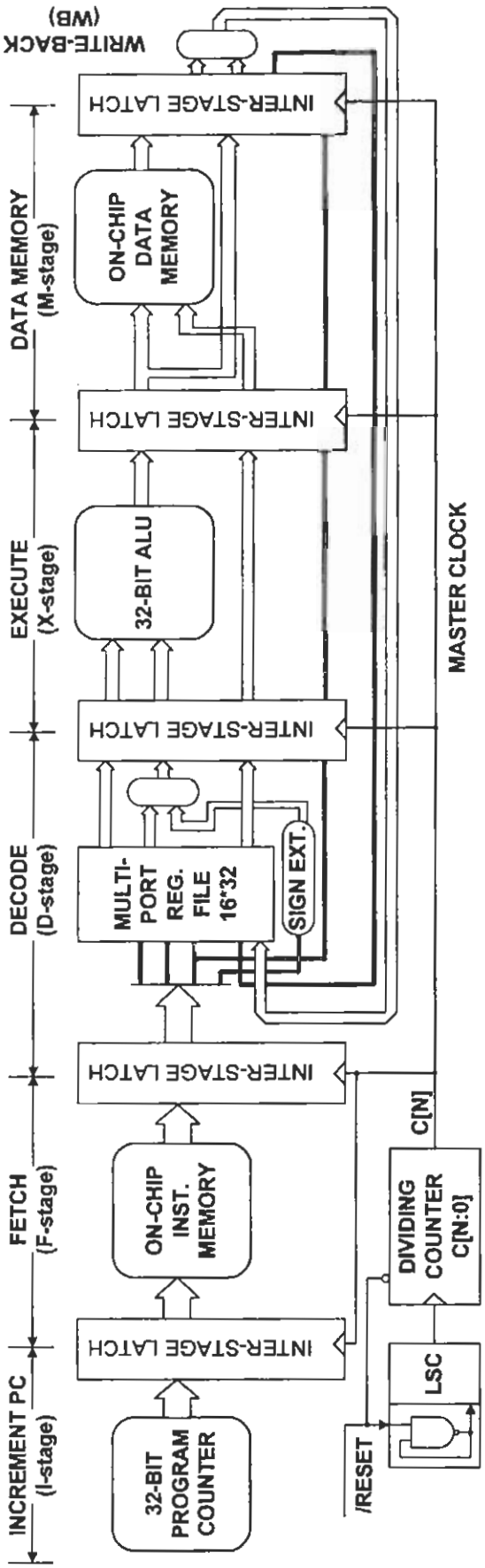


Figure 5.1: Block diagram of extcrnally clockless simple RISC machine.

XCV200E-8FG456 FPGAs. Test program given in Table 5.1, was executed in case of all these implementations as shown in Figure 5.2 and Figure 5.3.

TABLE 5.1
Test program for externally clockless RISC machine

Address (hex)	Data Memory (hex)	Instruction Memory (hex)	Instruction
0	00000000	00000000	**NOP**
1	00000000	14200003	**LDI** R1, R0, #3
2	00000000	14400004	**LDI** R2, R0, #4
3	00000003	14600005	**LDI** R3, R0, #5
4	00000006	14800006	**LDI** R4, R0, #6
5	00000002	14A10007	**LDI** R5, R1, #7
6	00000001	10C11000	**LD** R6, R1, R2
7	00000000	14E10002	**LDI** R7, R1, #2
8	00000000	00432000	**ADD** R2, R3, R4
9	00000008	00642800	**ADD** R3, R4, R5
A	00000005	00853000	**ADD** R4, R5, R6
B	00000000	08253000	**ST** R1, R5, R6
C	00000000	08413000	**ST** R2, R1, R6
D	00000000	08620800	**ST** R3, R1, R2
E	00000000	00000000	**NOP**
F	00000000	1480000D	**LDI** R4, R0, #0D

It was observed that the clocked and clockless versions consumed, approximately, the same amount of power. The excess power consumed by the on–chip oscillator in case of the clockless version was comparable to the power consumed by external clock

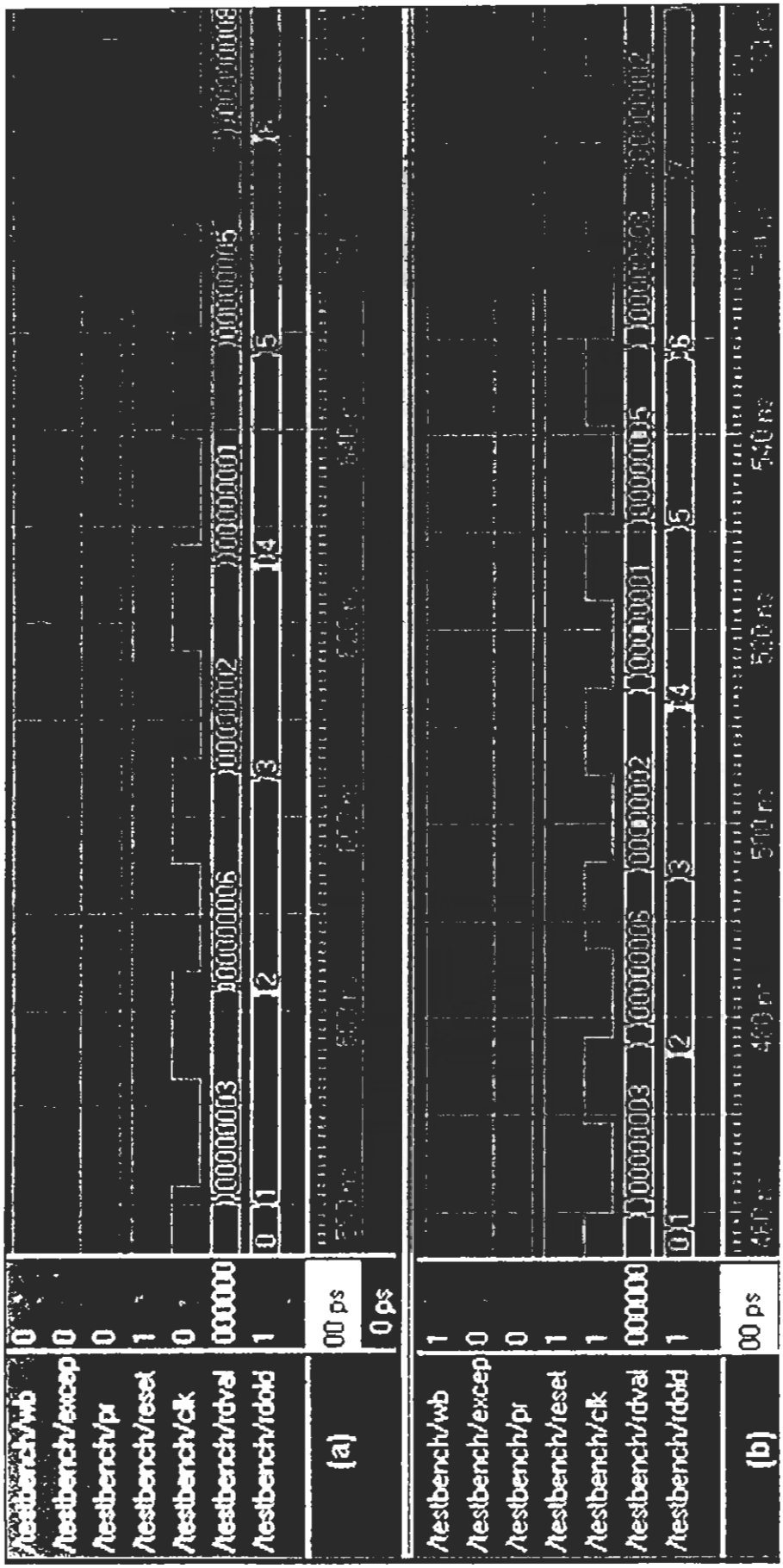


Figure 5.2: Externally clocked RISC implementation in FPGAs:
(a) XC2S400E-7FG456 : clock = 22nS, (b) XCV200E-8FG456 : clock = 18nS.

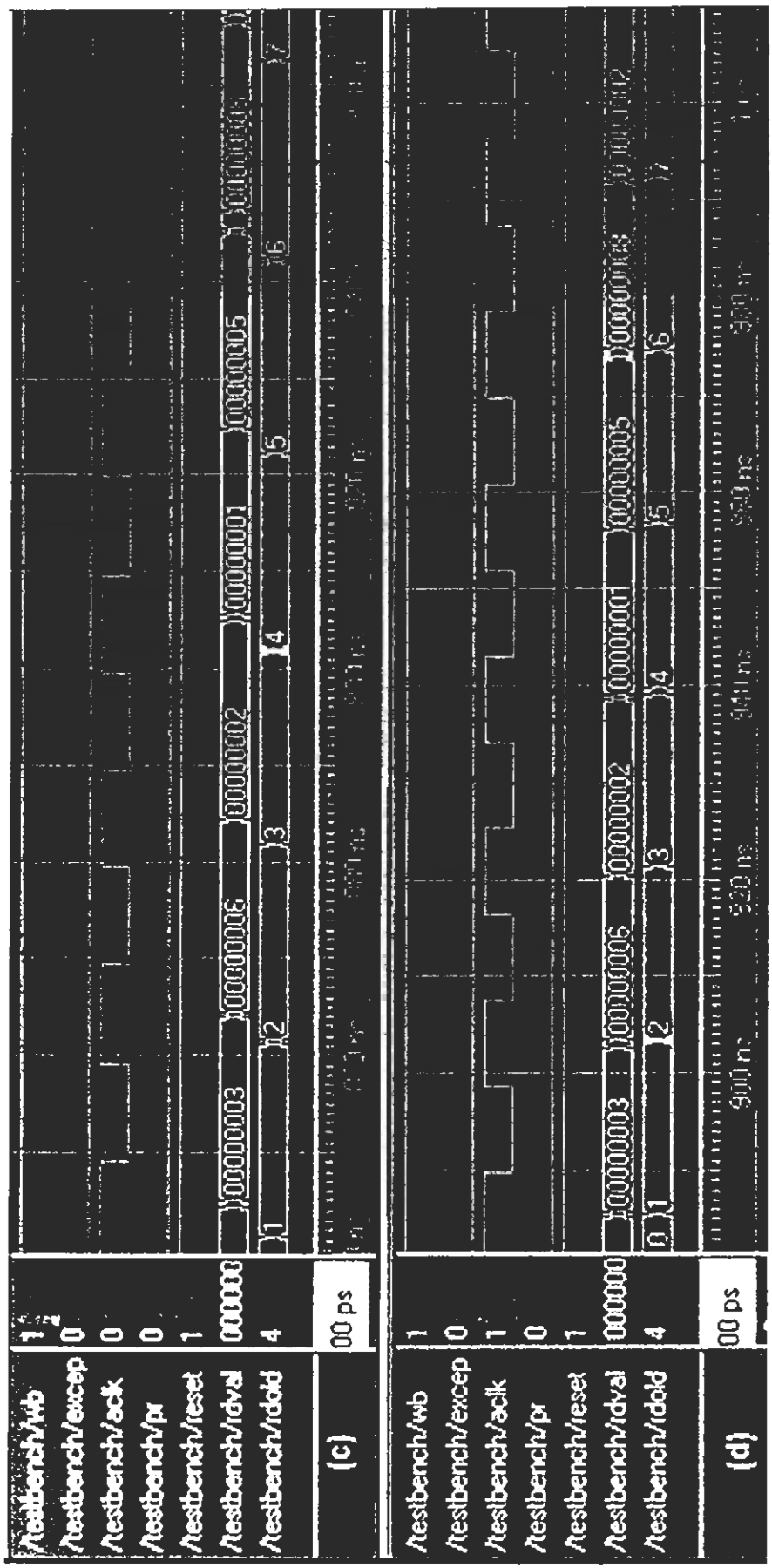


Figure 5.3: Externally clockless RISC implementation in FPGAs:
(c) XC2S400E-7FG456: self-generated clock = 20.2nS, (d) XCV200E-8FG456: self-generated clock = 17.8nS

input in case of the clocked version. The results as presented in Table 5.2 were verified using Xpower and Xilinx power estimator worksheets [POWS2, XXPOW]. Additionally, clocked version of simple RISC when implemented in XCV200E-8FG456 worked well with 55.6MHz oscillator, whereas, the same core when implemented in XC2S400E-7FG456, failed to function properly and required a replacement of external oscillator by the one with lower frequency for proper functioning. Post–layout simulation shown in Figure 5.2 presents this fact. On the other hand, as shown in Figure 5.3, SIRO–based on–chip oscillator automatically adapted itself to the technology change by reducing its frequency in case of XC2S400E-7FG456, optimally triggering simple RISC core in both the devices, thus, producing correct results without the need of any circuit modification.

TABLE 5.2
Comparison of various parameters of the Clocked and Clockless versions of RISC

Design	Device	Ext. Clock /On-chip Osc. 'nS'	Freq. 'MHz'	Power 'mW'	
				C	T
Externally Clocked RISC	XC2S400E	22	45.4	27.046	648.59
				29.488	653.81
	XCV200E	18	55.6	34.164	655.14
				34.532	657.03
Externally Clockless RISC	XC2S400E	20.2	49.5	0.0	654.55
	XCV200E	17.8	56.2	0.0	656.28

C = Clock Input, **T** = Total power consumption including the Quiescent power of 547mW,
☐ =Calculations @ clockless freq. for comparison.

In case of larger sequential circuits, where the on–chip oscillator is required to ride the clock tree within the traditionally synchronous FPGA mediums, CLKDLL and BUFG (Buffer for Global) modules [DLLS2, DLLVE] are incorporated to enhance

the fanout of the signal and to reduce the skews. But when the LSO was connected to BUFG, in order to use CLKDLL module, the following error message appeared:

"*When the CLKIN pin or the CLKFB pin of a CLKDLL is being driven by a BUFG, the BUFG must also be driven by a CLKDLL. To by-pass this error, set environment variable XIL_MAP_ALLOW_ANY_DLL_INPUT.* "

Therefore, as explained in Chapter No. 3, the environment variable XIL_MAP_ALLOW_ANY_DLL_INPUT was set to 1, in order to let the LSO use delay-locked loop.

Because of being on–chip rather than on–board, the SIRO–based oscillator driven synchronous designs have better EMC (a property generally associated with asynchronous circuits), with an option of further improvement through IC shielding, impossible in case of externally triggered systems. This makes SIRO–based oscillator driven circuits, more suitable for use with communication equipment.

6

FPGA Compliant Micropipeline

As already explained in Chapter No. 2, a micropipeline is an event driven elastic pipeline with or without processing [IAS89]. It can exist in a 2-phase or in a 4-phase form [IAS89, TEW91, CSC01, OAP97] and is characterized by bundled data strategy. In Sutherland's 2-phase micropipeline [IAS89], special event-controlled registers are used for inter stage latching and Muller C-elements are used for the micropipeline control. Delay elements equivalent to the time taken by the respective logic unit to generate the desired results on the datapath are inserted to bundle the data. In fact, these delay elements define the request cycle at each stage and the time difference between the edges of the request and the acknowledge signals. This approach works very well with full custom designs but fails in case of FPGAs because of difficulty in implementing delay elements in FPGAs. Therefore, a modified FPGA compliant micropipeline model is presented in this chapter.

The concept of micropipeline which is native to full custom design is imported to a traditionally synchronous environment of FPGAs by introducing SIRO–based delay elements, customized with the help of dividing counters to suit the requirements of various stages of micropipeline. To implement FPGA–compliant micropipeline

unbundled data strategy has been adopted by incorporating special event controlled registers as opposed to traditional bundled data approach of micropipeline.

Micropipelines are not implemented in reconfigurable mediums because FPGAs have predefined building blocks such as LE or Configurable Logic Blocks (CLBs) and interconnects. These FPGAs are reconfigurable but their layout is fixed and different for various devices. Secondly, variety of synthesizers, optimizers and routers follow different schemes for the implementation of a given design. This makes it virtually impossible to implement technology independent delay model in commercially available FPGAs. Manual intervention can eliminate this problem, but then the time factor becomes the same as that for a full custom implementation making full custom solution preferable for its higher efficiency.

The difficulty in case of FPGAs is the unpredictability of on–chip delays. To be technology independent a predefined delay requires a circuit that supports dynamic calibration. It has been shown that on–chip SIRO is a circuit with this property.

Another critical problem is that of datapath implementation in FPGAs. Reduction in real estate of a design requires bundled data strategy. To bundle the data in an asynchronous design generally means to place accurate delay elements parallel to the datapath [IAS89]. Now even if a technology independent delay pad is placed in the circuit, various design tools may implement the same datapath in different styles ruining the ratio between delay element and the propagation delays in various fork legs of datapath [TEW91, CSC01]. Therefore, in this study, the unbundled data strategy with bit–encoding is adopted for the implementation of micropipeline in FPGAs.

6.1 SIRO–based Delay Element

Design of a SIRO–based delay element has already been discussed in detail in Chapter No. 3. It has also been proven that this circuit is technology independent. Therefore, these delay elements are used to develop the delay model for the FPGA–compliant micropipelinc, and thus eliminate the first problem associated with such an implementation. By changing the parameterized depth of the dividing counter, the delay elements can be placed in various stages of micropipeline to insert the required delay. The technology independent nature of these delay elements ensures dynamic calibration of delay to maintain timing constraints for the micropipeline upon technology variation. Technically, this should be enough to bundle the data, but the uncertainty associated with environments' implementation styles force the use of unbundled data strategy despite the use of technology independent delay elements. Figure 6.1 shows the implementation of delay elements in Xilinx XC2s400e-6fg456 and Altera EPF10K30ETC144-1 devices. ISE Foundation series 5.2i with ModelSim XE ver. 5.6e was used for the simulation of Xilinx device, while Max Plus+2 ver.10.2 was used for Altera device. Rising edge of a request signal triggers the SIRO that drives the dividing counter. Upon reaching a preset count value, the SIRO stops and acknowledge signal is generated to show that the stage has completcd its task. Another request to the stage, forces acknowledge signal to go low while the SIRO is retriggered to run the dividing counter. Figure 6.1a&b show post-layout simulation results for the same delay i.e. N=2 in Xilinx and Altera devices, while Figure 6.1c shows simulation results for the same code in Altera device when the delay required parameter N to be 3.

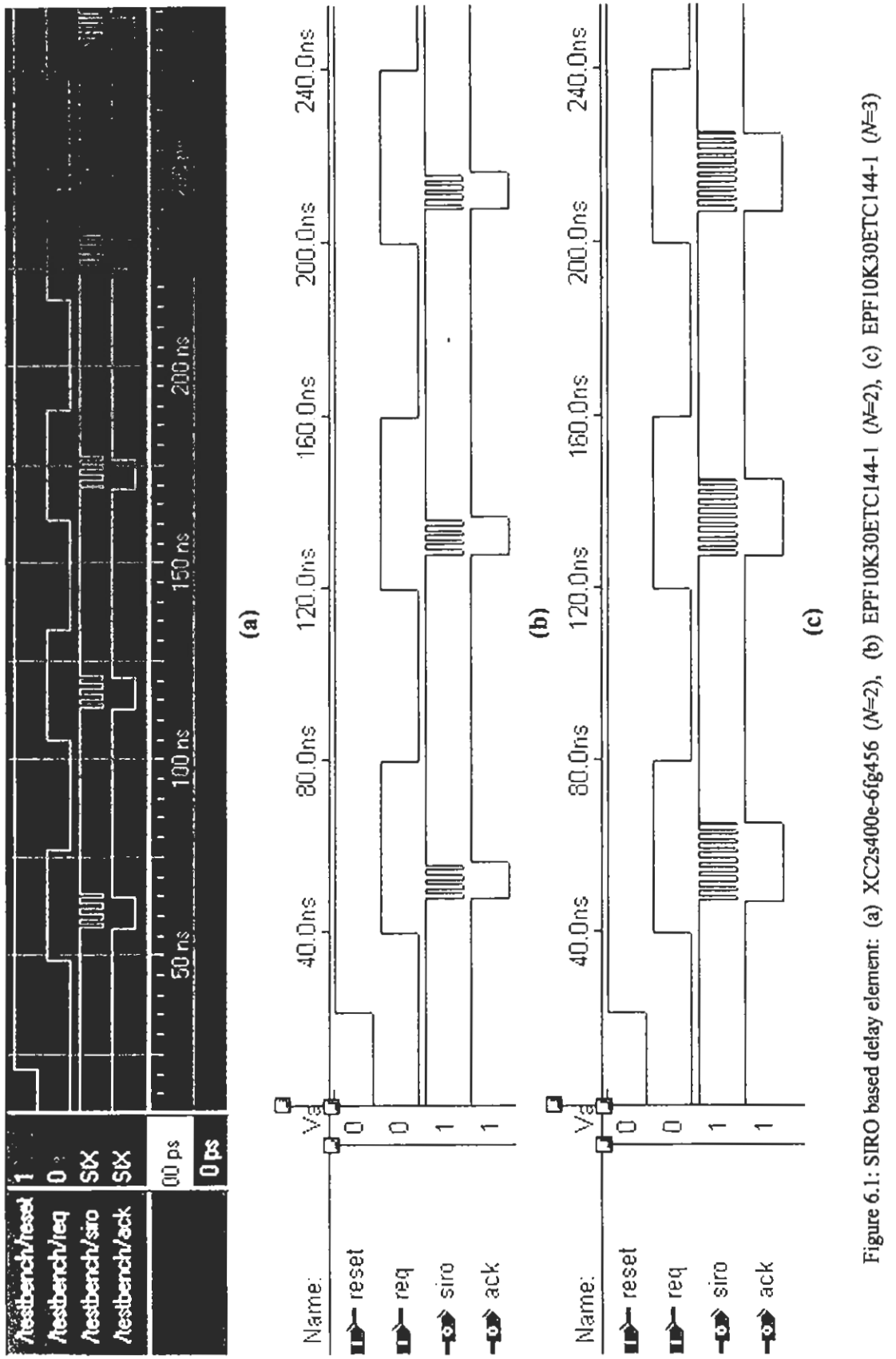


Figure 6.1: SIRO based delay element: (a) XC2s400e-6fg456 (N=2), (b) EPF10K30ETC144-1 (N=2), (c) EPF10K30ETC144-1 (N=3)

6.2 Unbundled Datapath

To overcome the unpredictability of the datapath layout in FPGAs special event controlled inter stage registers have been designed which incorporate bit encoding and return-to-zero strategies to handle the unbundled datapath [WJB01]. A 2–bit Event Controlled Register (ECR) is shown in Figure 6.2.

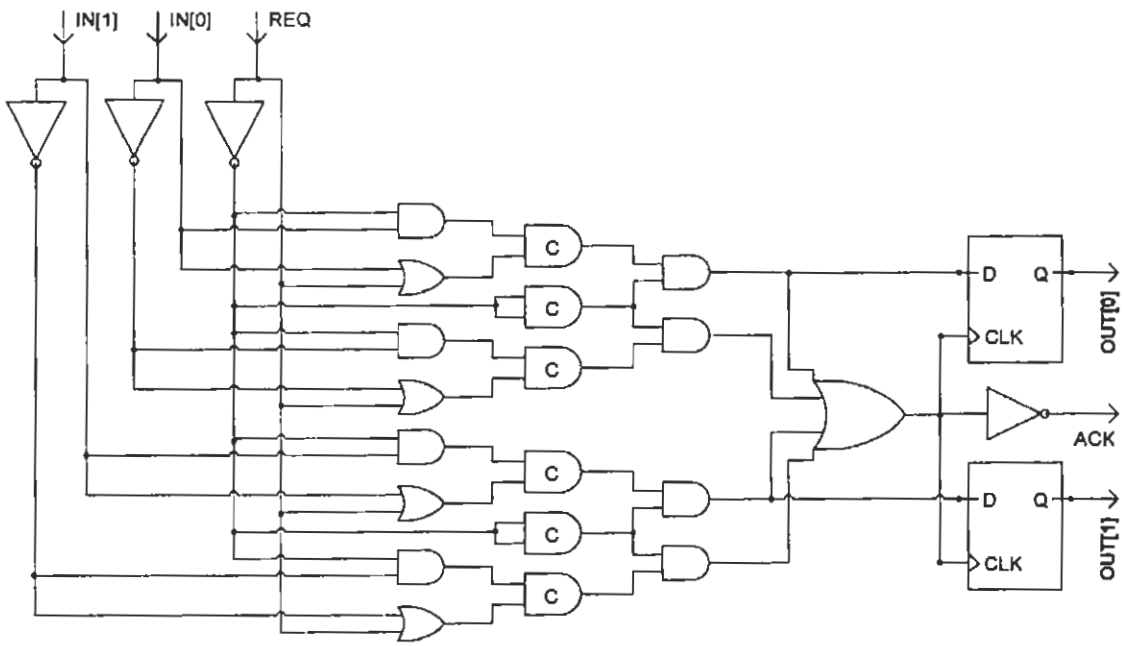


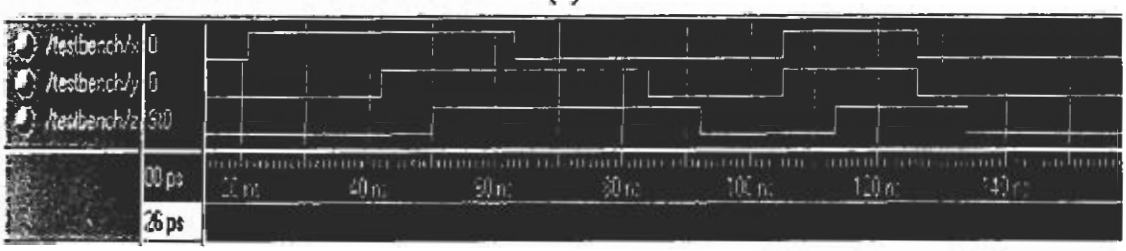
Figure 6.2: Functional equivalent of Verilog model for a 2-bit event controlled register with bit encoding.

Each input data bit is encoded to '10' or '01' for its state of '1' or '0' respectively. At the falling edge of the request signal, all the encoded bits return-to-zero so that even if a similar event occurs twice, the two may be distinguishable. At the rising edge of the request to the ECR the input bits get encoded generating a '1' at the 'clk' input of LE latch. Whereas on the falling edge of the request, stable value of the event is latched and an inverted acknowledge (/ACK) signal is triggered when all the encoded bits return-to-zero.

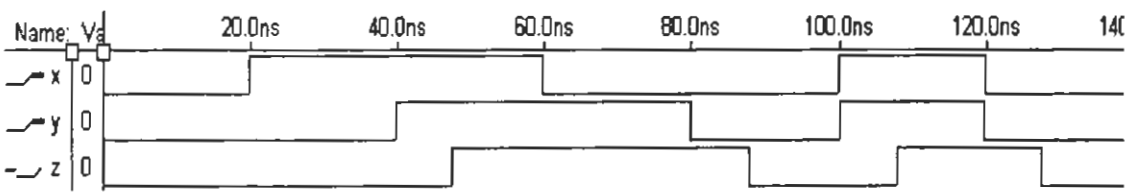
Muller C-element implementation in FPGA is used for completion detection. Verilog model of C-element implemented and the post–layout simulation results for XC2S44E-6fg456 and EPF10K30ETC144-1devices are shown in Figure 6.3.

```
module celement(z,x,y);
input x,y;
output z;
wire z1, z2, z3;

nand u1(z1, x, y);
nand u2(z2, z, y);
nand u3(z3, z, x);
nand u4(z, z1, z2, z3);

endmodule
```

(a)



(b)



(c)

Figure 6.3: (a) Verilog code for Muller C-element, (b) post–layout simulation results for Xilinx XC2S44E-6fg456 and (c) Altera EPF10K30ETC144-1 devices.

The ECR receives a request from the associated state. After receiving all the bits of data, it generates an acknowledge signal to the associated state that computes the next result and goes into sleep mode but does not deliver this new result to the ECR till the

following stage receives the previously latched result from ECR. Once the ECR delivers the result to the following stage, it removes it's acknowledge signal, that lets the associated stage latch the pre-computed result in it. The cycle is then repeated. This pre-charging feature resembles the William's PSO pipeline [TEW91]. Post layout simulation of a 32–bit ECR for (a) XC2s400e-6fg456 and (b) EPF10K30ETC144-1 devices is shown in Figure 6.4.

6.3 Micropipeline for FPGAs

Figure 6.5 shows a FPGA–compliant model of a 4-phase micropipeline [YZR04]. In this figure special ECRs provide inter stage latching. Each stage consists of a delay pad with its own adaptive SIRO, latch synchronizing circuit and a counter with reconfigurable depth *N*. The depth of the counter is parameterized to customize the delay element of each stage. A stage after completing its task stops the SIRO and generates a request signal to the following ECR. The ECR does not process this request till the following stage completes its task and generates a request to its relevant ECR for the saving of the results. Once the results are successfully latched, the acknowledge signal from the ECR of stage 1 re-triggers the SIRO for another task to be performed by stage 1.

This request / grant protocol is repeated in every stage of the micropipeline. The only difference being in case of the final stage where the request to the following ECR comes from stage 1, as in case of Williams PSO pipeline [TEW91]. As a result of this strategy, a stage completes its task; its SIRO goes into the sleep mode to conserve power and does not wakeup till the following stage requests the results of the first stage to be passed onto it through the ECR.

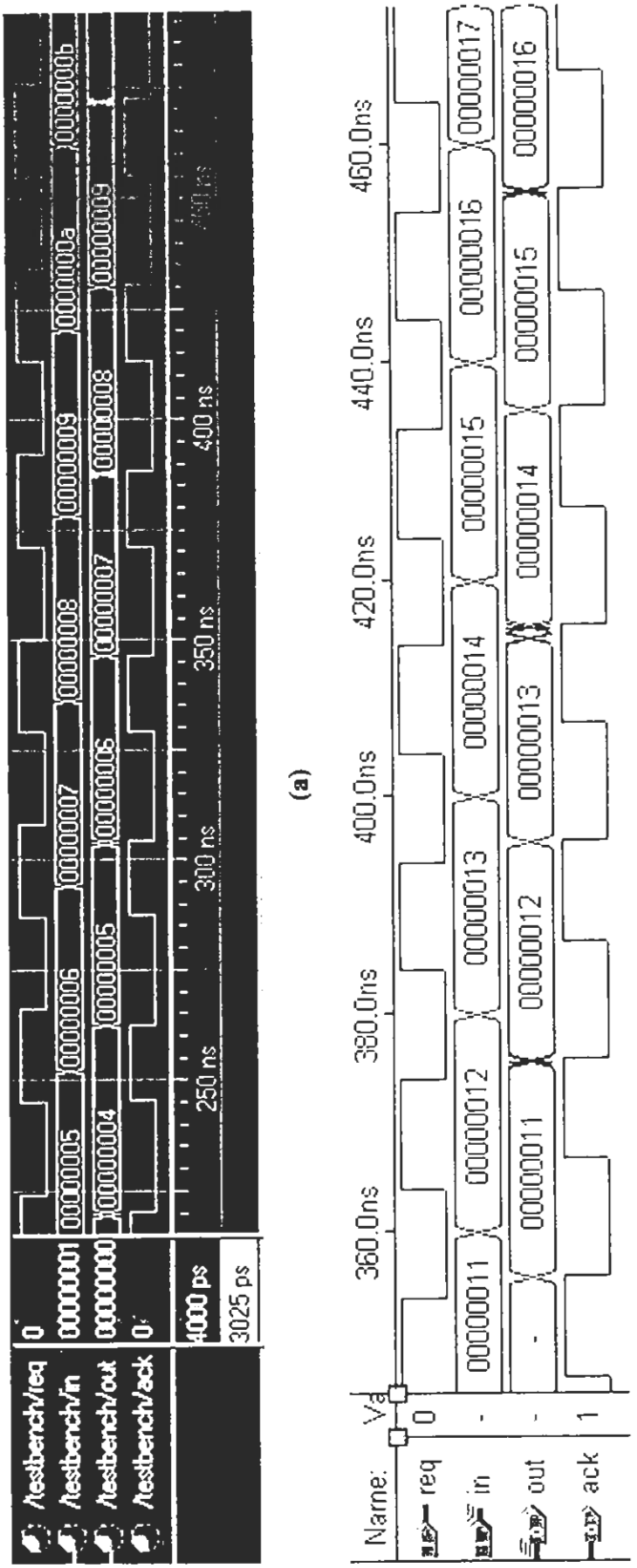


Figure 6.4: Unbundled datapath for FPGA–compliant micropipeline: (a) XC2s400e-6fg456, (b) EPF10K30ETC144-1

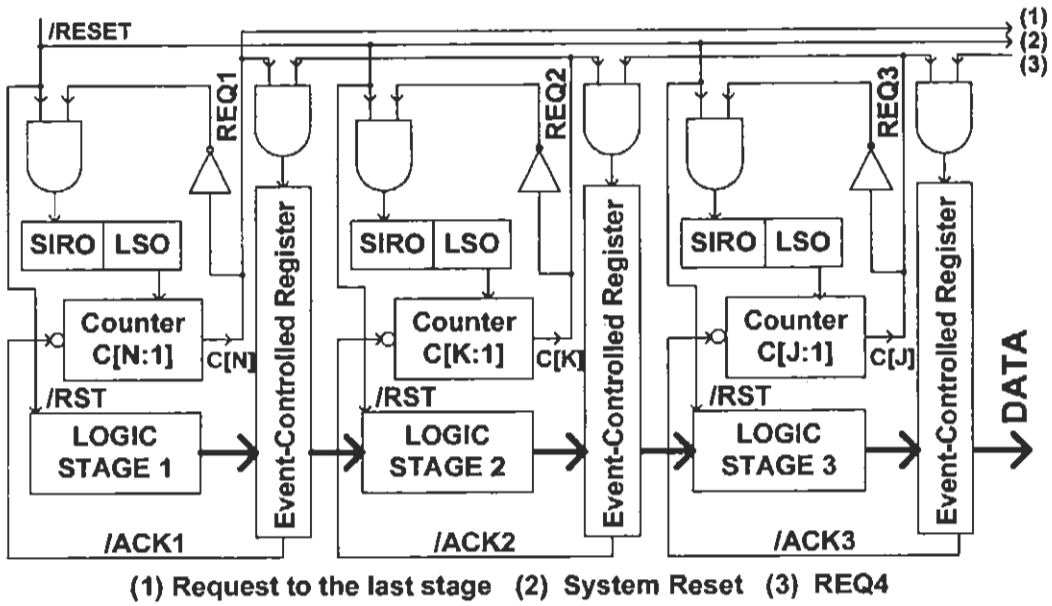


Figure 6.5: Illustrates a micropipeline model implemented in FPGA.

Same Verilog code for a 5-stage FPGA–compliant micropipeline was implemented in various FPGAs, to confirm the technology and environment independence of the design. Figures 6.6 and 6.7 show the post–layout simulation results when the design was implemented in Xilinx XC2S400E-6456FG and Altera EPF10K30ETC144-1 devices using ISE Foundation series + ModelSim XE and MaxPlus+2 environments, respectively. Different stages of the micropipeline have different values of *N*, to implement the desired delay. Stage one has a 32-bit counter as the functional unit associated with it. This counter is also triggered by the same SIRO that is used in the implementation of delay element, thus eliminating the need of external clock source. In the same manner other stages can also have processing units that may use the on-chip oscillator associated with that stage, giving micropipeline an over all GALS look [YZR5a], for the stages communicate asynchronously. The count of stage 1 proceeds through the other stages with different delays and emerges from the final stage,

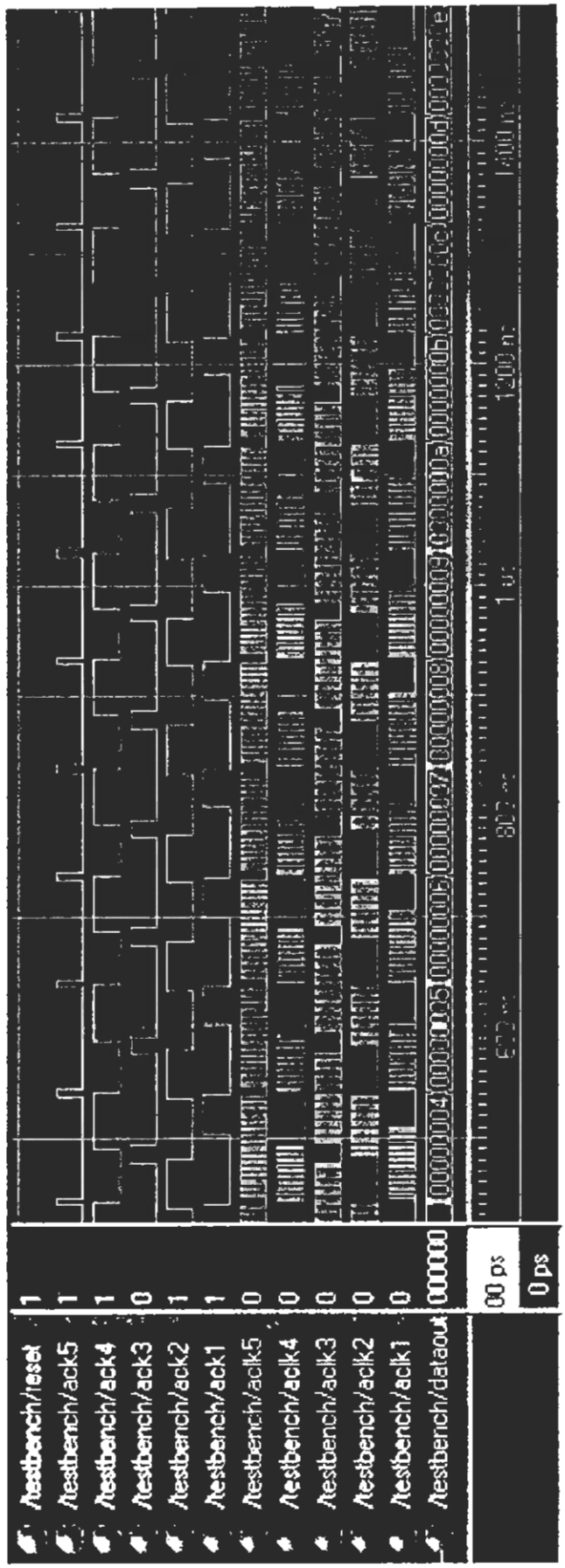


Figure 6.6: FPGA–compliant micropipeline implemented in XC2s400e-6fg456 using ISE Foundation Series and ModelSim XE.

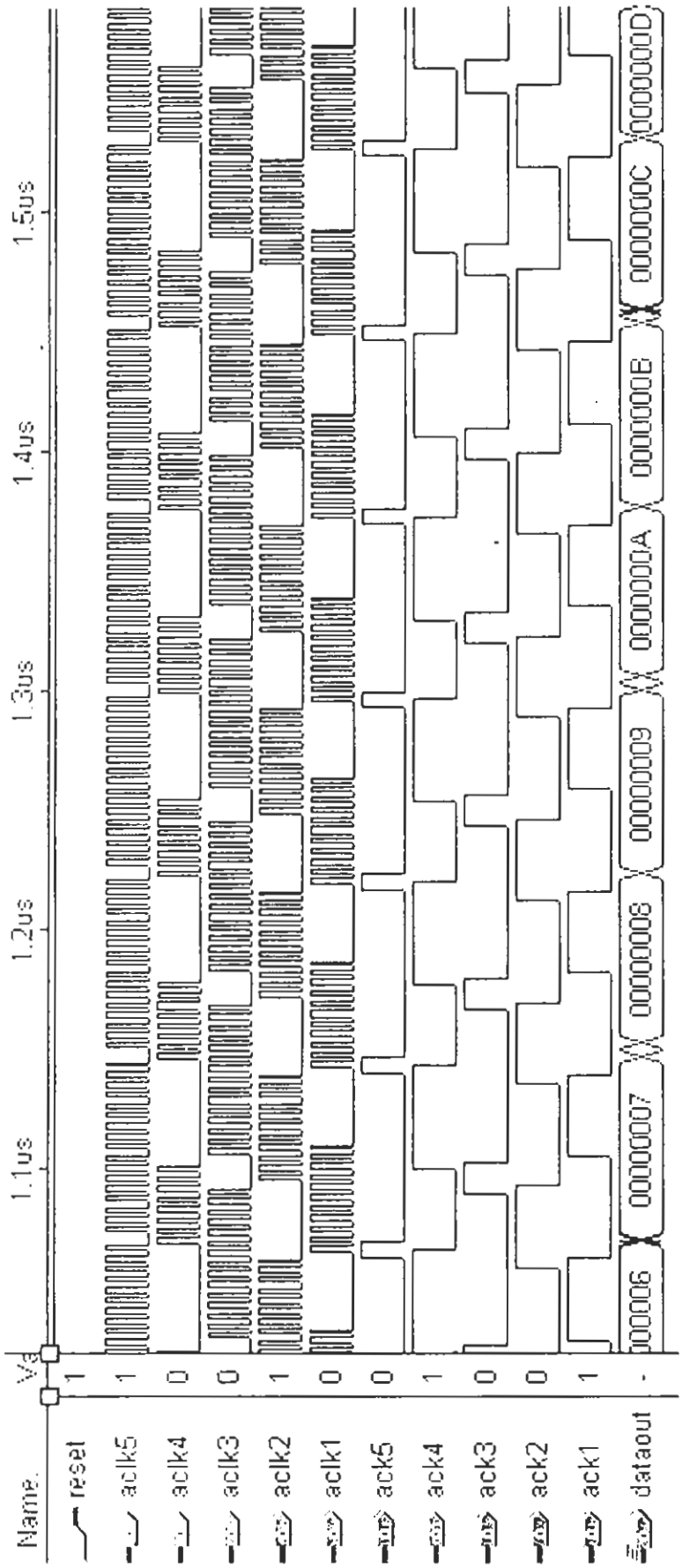


Figure 6.7: FPGA–compliant micropipeline implemented in EPF10K30ETC144-1 using Altera MaxPlus+2.

proving the correctness of design. Variation in performance of the same design upon implementation in different devices, emphasizing technology independence, is reported in Table 6.1. The simple RISC presented in Chapter No. 4 was implemented using this micropipeline model, as an application to support the claims. The following chapter describes in detail the implementation Reconfigurable Micropipelined Processor (RMP) and the associated power issues are discussed in Chapter No. 8.

TABLE 6.1
Micropipeline performance in Altera and Xilinx devices.

Device	SIRO 'nS'	LSO 'nS'	Avg. Data output 'nS'	Throughput '10^6 Data items / S'
Altera EPF10K30ETC144-1	1.8	3.6	70	14.3
Xilinx XC2s400e-6fg456	2.4	4.8	94.6	10.6

7

Reconfigurable Micropipelined Processor

As an application of FPGA–compliant micropipeline, simple RISC presented in Chapter No. 4 was implemented as RMP in different FPGAs such as Altera EPF10K70RC240-2 and Xilinx XC2s400e-6fg456. In the implementation, on–chip memory resources of FPGA were configured as instruction and data memories. Multi-port register file with read after write capability in the same stage was implemented to avoid structural hazards. Read After Write (RAW) type data hazards were avoided through forwarding using multiplexers at ALU input. Control hazards were avoided by insertion of stall.

This chapter presents two different implementations of RMP. First one is its implementation in Altera EPF10K70RC240-2 device using MaxPlus+2 environment. This implementation is simple without human interference. The second implementation is in Xilinx XC2s400e-6fg456 device using ISE Foundation series with ModelSim XE. In this implementation, timing constraints are provided through a customized user constraint file (ucf) and manual editing is done in placement and routing. Additionally, in both cases all the stages of the micropipelined processor are individually tested to observe their performance and are then merged in a top level

module to test the entire asynchronous processor. It is found that the design confirms the properties of micropipeline in the reconfigurable medium of FPGAs.

As the purpose of implementing RMP was to confirm the asynchronous properties of FPGA–compliant micropipeline and not to develop a new kind of architecture with advanced features, the architecture has been kept minimal but complete to verify the presented concepts. Firstly, the Altera implementation with its details is presented, while the later sections present Xilinx implementation.

Figure 7.1 shows FPGA compliant model of a 4-phase 5–stage micropipeline used in the implementation of RMP in Altera device. In this figure special ECRs provide inter stage latching. Each stage consists of a delay element with its own adaptive SIRO, LSC and a dividing counter with reconfigurable depth *N*. The depth of the counter is parameterized to customize the delay element in each stage.

7.1 Increment PC Stage

The program counter resets to zero which is the reset address and when the load input is set high upon the execution of a control instruction, it loads a new value else it increments. SIRO of this stage is synchronized with the setup & hold time of flip flops through LSC, to produce '*ACLK*'. As shown in Figure 7.2, *N* of the dividing counter for this stage is 3 (2:0). The second Most Significant Bit (MSB) of the counter i.e. counter[1], is used to drive the PC, while the MSB produces the acknowledge signal. Upon successful delivery of PC value to the ECR associated with stage 1, request to this stage clears, resulting in the halt of SIRO, forcing the stage to go into sleep mode. Post–layout simulation of the functional unit (32-bit resetable / loadable

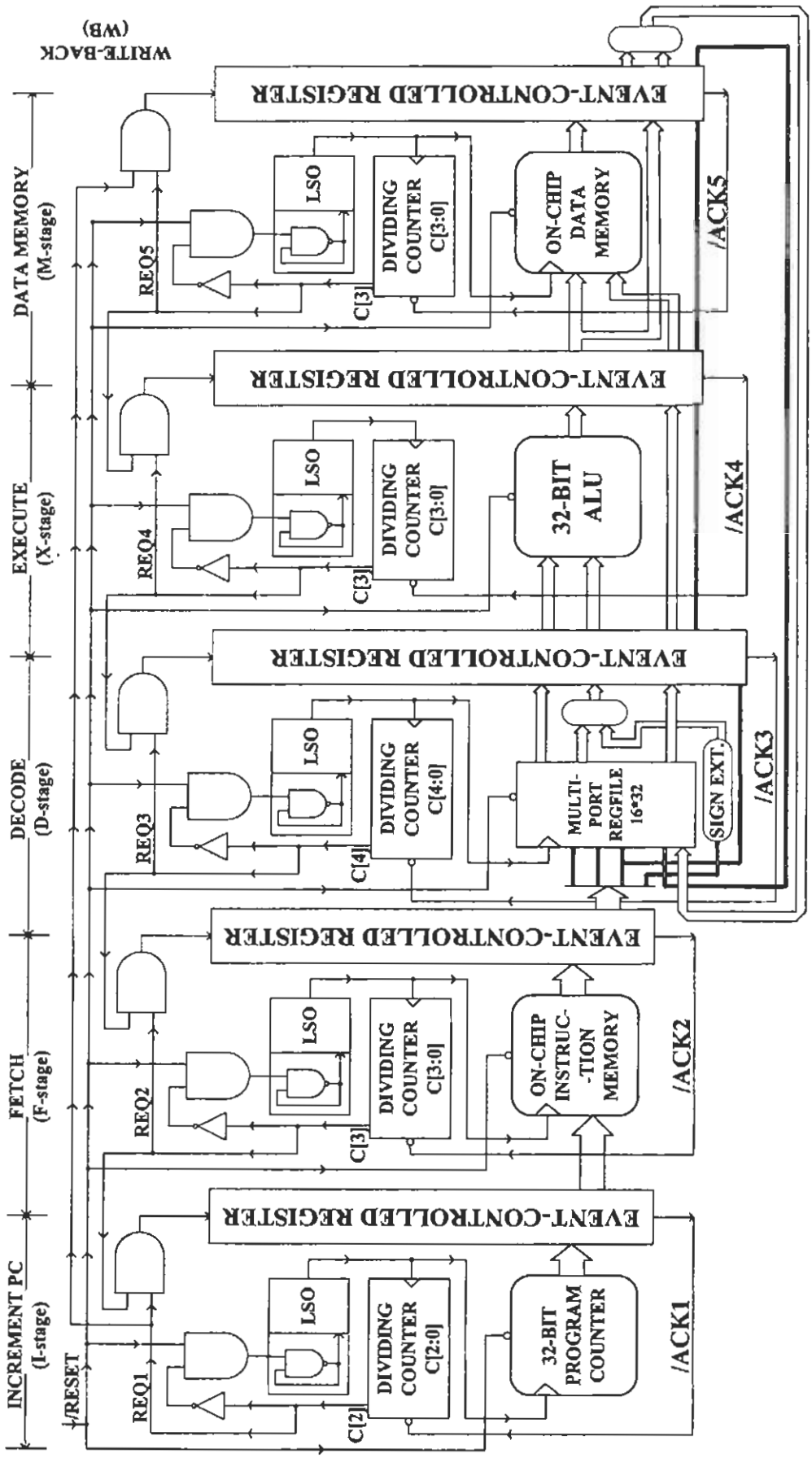


Figure 7.1: Illustrates micropipeline used in the implementation of RMP.

PC) associated with stage 1 (Increment PC) of our micropipeline is presented in Figure 7.3.

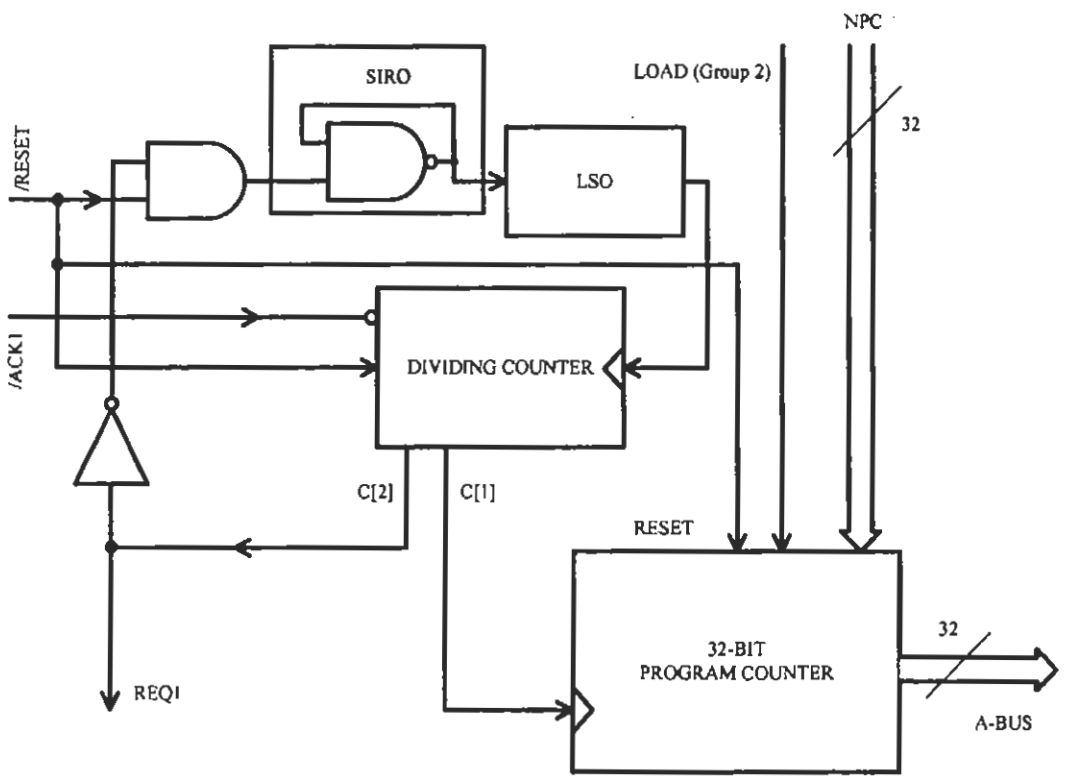


Figure 7.2: Block diagram Increment PC stage.

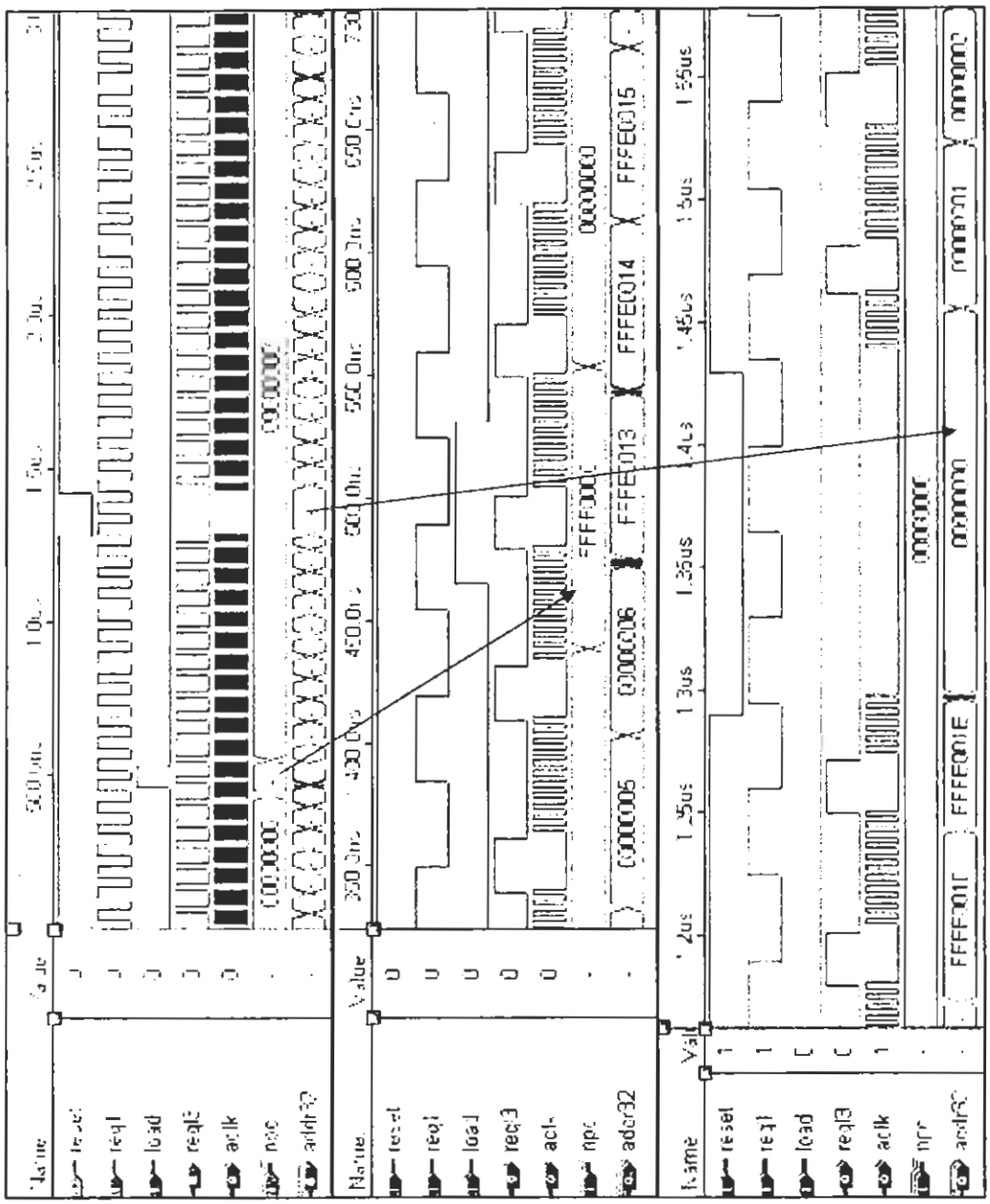


Figure 7.3: Post-layout simulation of 32-bit resetable / loadable PC.

7.2 Fetch Stage

Fetch receives PC value from the stage 1 via ECR to address the instruction memory, implemented using embedded array resources of the FPGA, as shown in Figure 7.4. The embedded array resources of the FPGA are configured as 32-bit wide words in lieu of RMP's fixed length instruction size. The *N* of dividing counter for this stage is 4(3:0). The second MSB of the counter i.e. counter[2], is used to fetch instruction from the memory, while the most significant bit produces the acknowledge signal.

7.3 Decode Stage

As maximum on–chip memory resources are reserved for the implementation of instruction and data memories, the functional unit associated with decode stage, i.e. the multi-port register file is implemented using latch resources of the LEs. Therefore, this stage consumes the maximum space in our design and is the slowest. *N* for this stage is 5 with MSB of the dividing counter producing the acknowledge signal. Write strobe (/counter[4] & /counter[3] & counter[2]) and read strobe (counter[3]) implement read after write functionality in the register file.

Another important feature associated with the decode stage is the Write Back (WB) operation., Decode stage, therefore, receives two destination register (Rd) values simultaneously, one from the current instruction reaching this stage and other, from a previous ALU or load instruction, As mentioned earlier, out–of–order execution is not supported in this version of RMP. Figure 7.5 shows post–layout simulation of the multi-port register file with read after write capability in the decode stage.

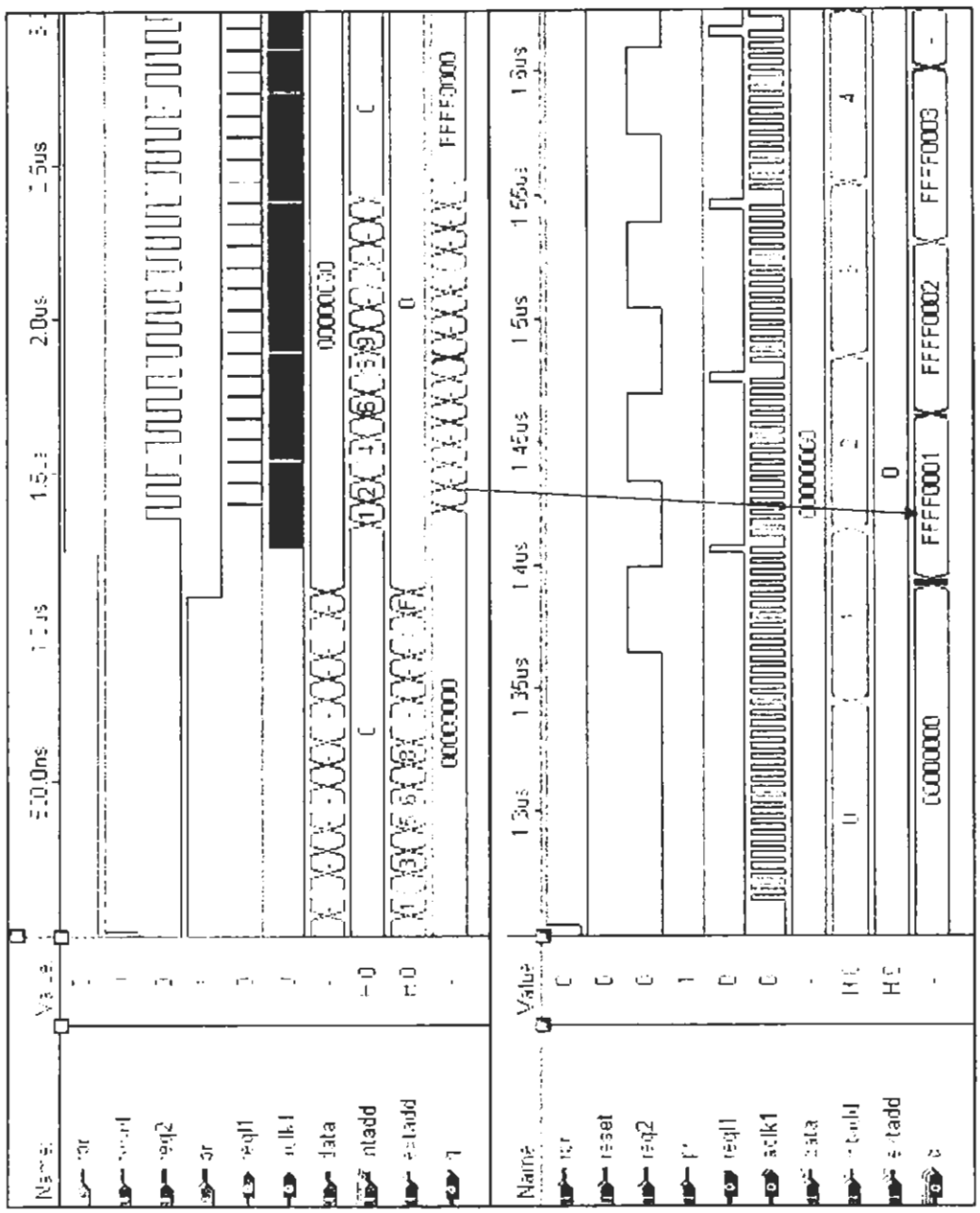


Figure 7.4: Post-layout simulation of Fetch stage associated with instruction memory, implemented using the embedded array.

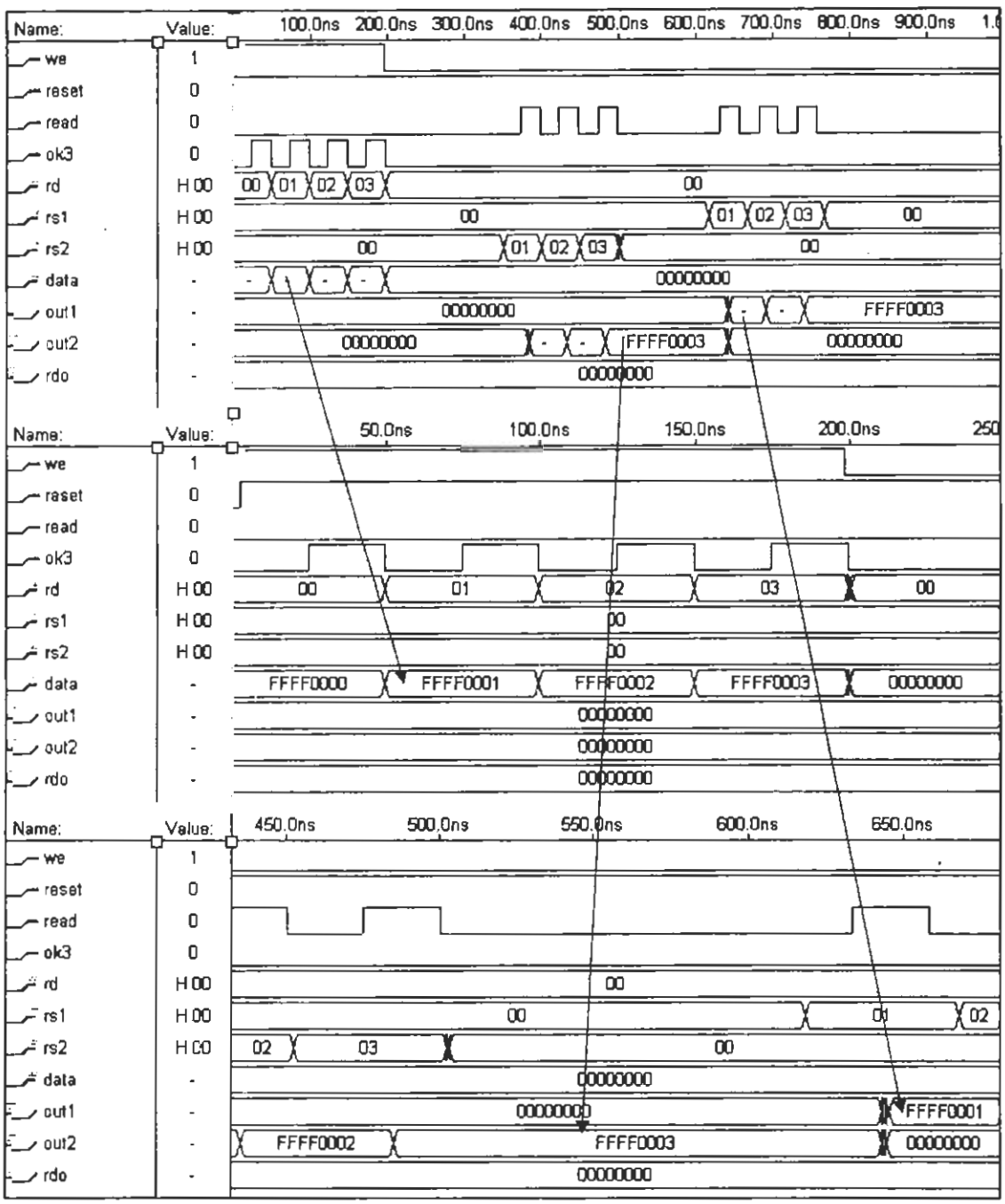


Figure 7.5: Post-layout simulation of the multi-port register file with read after write capability in the decode stage.

7.4 Execute Stage

Execute stage is associated with a 32-bit ALU. It is used to calculate effective address in case of load / store instructions, while in case of ALU instructions it is used to perform arithmetic and logic operations on the data in registers. The value of *N* for this stage is 4(3:0). The MSB of the counter produces the acknowledge signal. The post–layout simulation of ALU is shown in Figure 7.6, where *cbus* = 0 means addition operation, while *cbus* = 1 means subtraction.

7.5 Memory Stage

The data memory is implemented using embedded array of the FPGA. Its structure resembles the instruction memory, the only difference being that it can be read as well as written into at an address generated by the ALU in case of load or store instruction. The contents to be written, in case of store instruction, originate in the decode stage using the 'Rd' field. They pass by the successive stages unaltered using ECRs. In case of load instruction, the data memory address is generated in the ALU with the help of 'Rs1' and 'Rs2' (or 'IMM') fields and the data that gets ejected by the memory, is written into a register in the register file, pointed by the 'Rd' field. The 'Rd' field is therefore, carried along the various stages for write back purposes.

N of the dividing counter for this stage is 4(3:0). The second MSB of the counter i.e. counter[2], is used to read or write from /to the data memory, while the most significant bit produces the acknowledge signal. The post layout simulation of data memory implemented using the embedded array resources is shown in Figure 7.7.

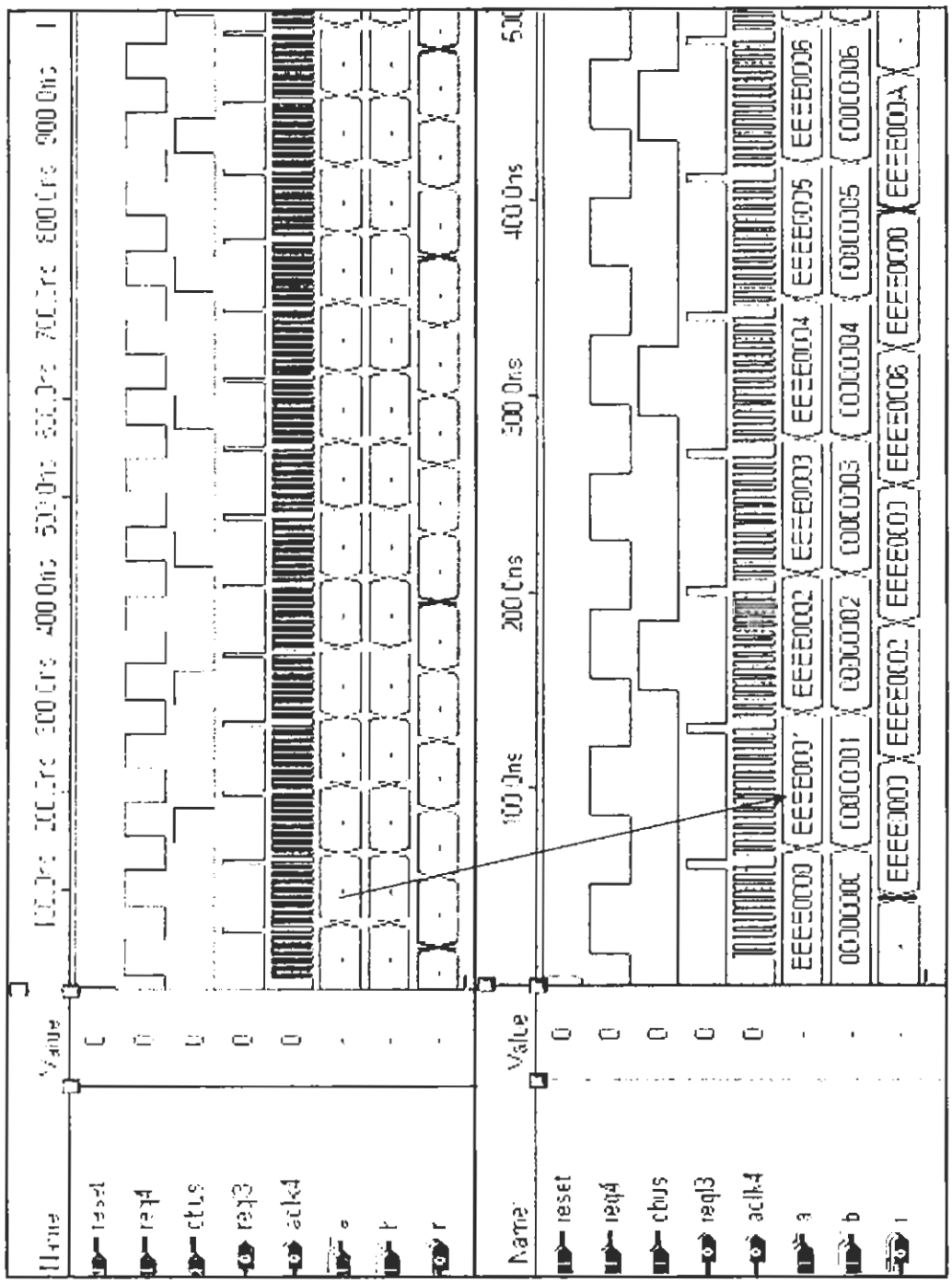


Figure 7.6: Post-layout simulation of the Execute stage of RMP associated with 32-bit ALU.

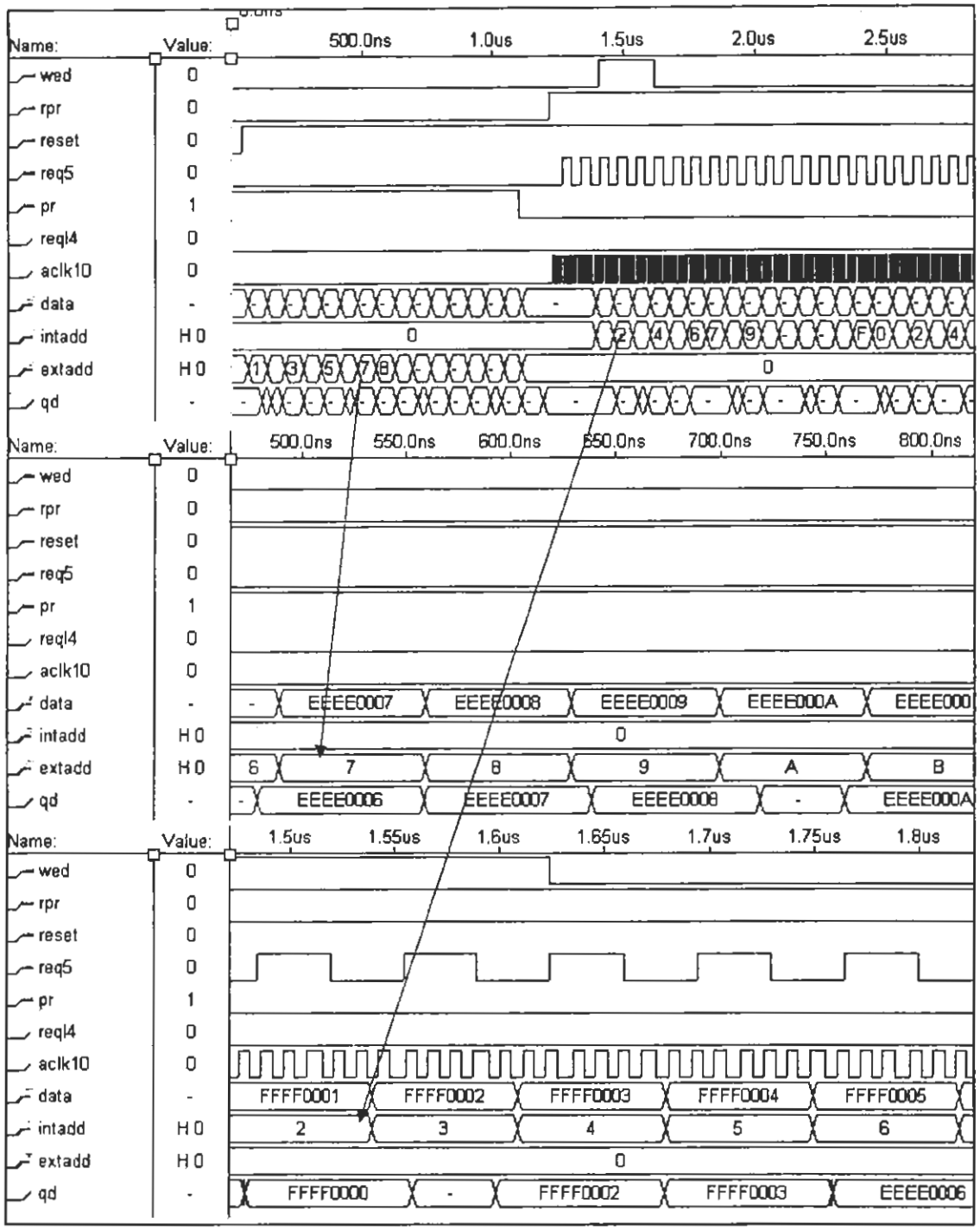


Figure 7.7: Post–layout simulation of the Memory stage of RMP associated with data memory, implemented using the embedded array of FPGA.

7.6 Reconfigurable Micropipelined Processor

All the stages, separated by the ECRs, are instantiated in a top level module. Each stage has its own SIRO based delay element customized by its dividing counter, *N* of which is provided by Figure 7.1 for Altera implementation.

A sample program shown in Table 7.1 is loaded into the memory to validate the results. Figure 7.8 shows a few signals in the post layout simulation of RMP. Important thing to note is that because of varying delay elements each stage operates at its own pace. The SIRO of a particular stage goes into sleep mode after completing its task till it is requested to wake by the following stage. So the events of varying length are observed including the power saving sleep-mode durations in each stage, a property unique to asynchronous processing. Of particular importance is the result 'r' generated by ALU in stage4, which is transported by different stages at their own leisure to finally reach register file for writing as 'res3'. In case of an ALU instruction 'res3' is the output of ALU to be stored in the register file, whereas in case of a LD (Load) instruction it is the data memory content moving towards the register file.

TABLE 7.1
Test program loaded into RMP to validate results

Addr.	INSTRUCTION						
65H	0	0010	1	00001	00000	0000000000000001	
	G	Opcode	I	Rd	RS1	Immediate	
	LDI R1, R0, #1						
66H	0	0011	0	00010	00011	00000	00000000000
	G	Opcode	I	Rd	RS1	RS2	Unused
	SUB R2, R3, R0						
67H	0	0001	0	00011	00001	00000	00000000010
	ST R3, R1, R0						
68H	0	0010	0	00010	00001	00000	00000000001
	LD R2, R1, R0						

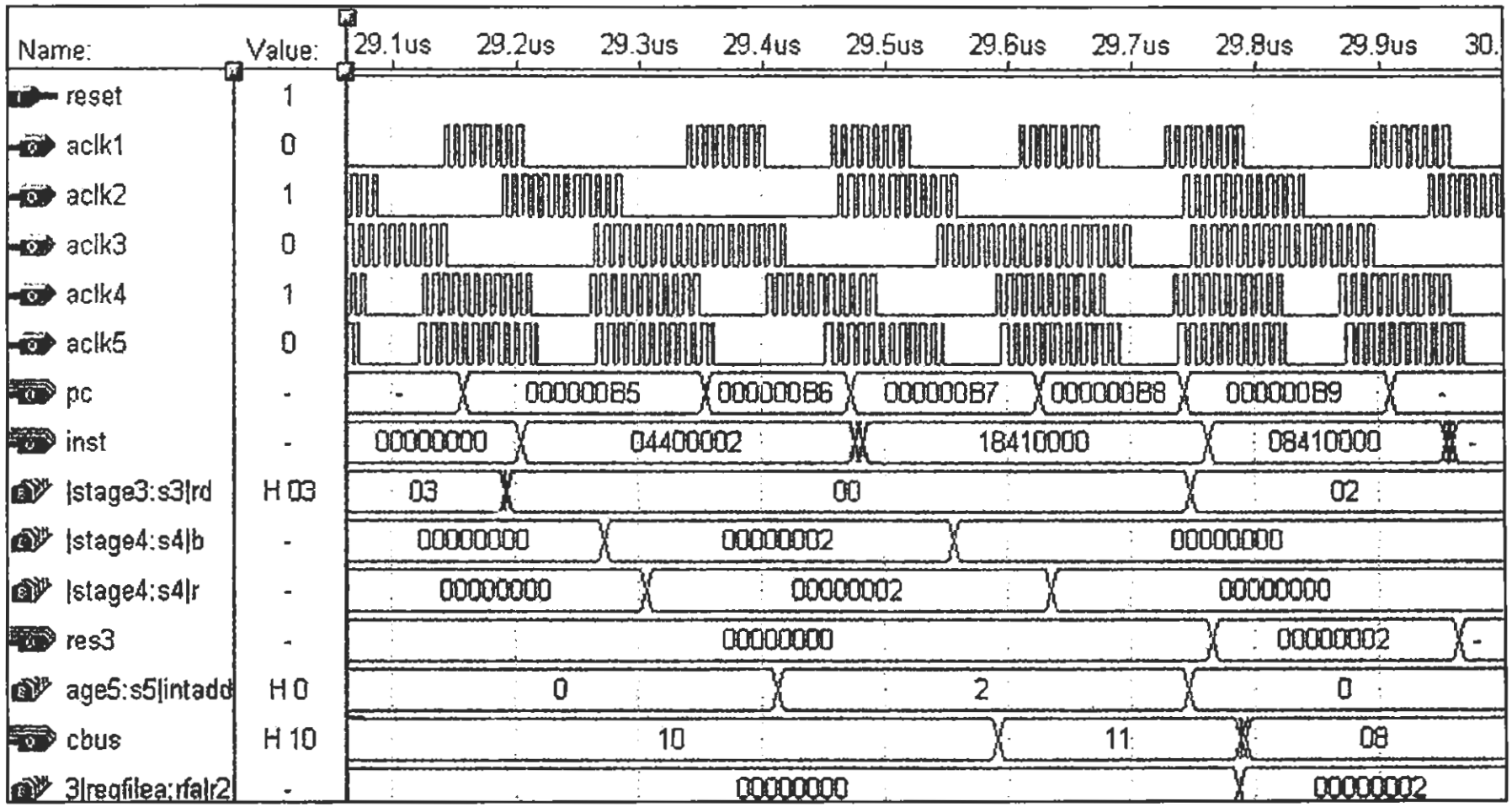
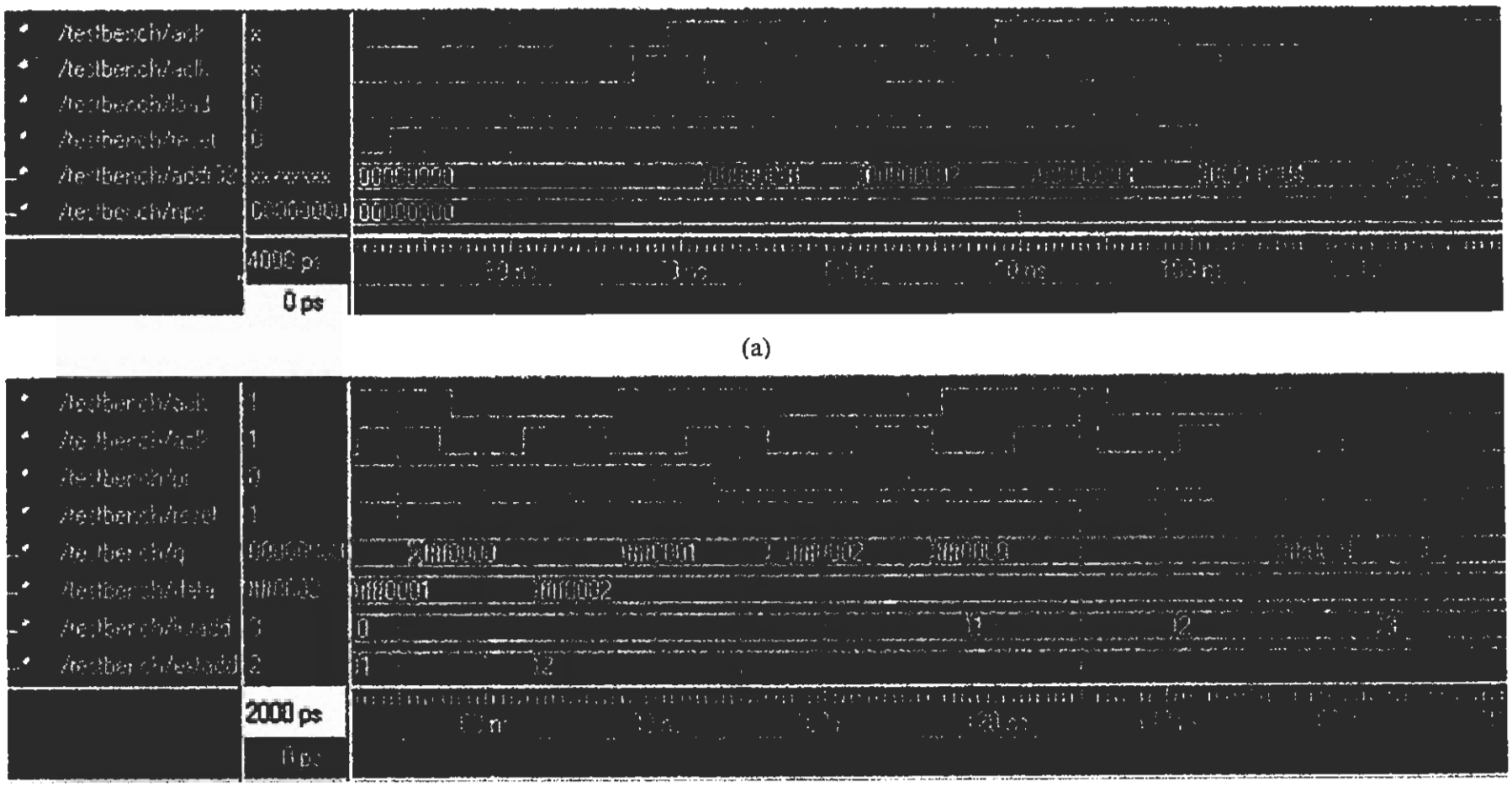


Figure 7.8: Post-layout simulation of RMP. Signals ACLK1-5 show both sleep and active modes of the latched synchronized oscillator. 'pc' is the program counter output to the instruction memory which releases the instruction "inst'. '|stage3:s3|rd' is the destination register, while '|stage4:s4|b' is the 2nd input to ALU. '|stage4:s4|r' is the result produced by ALU in stage4)

7.7 Edited Implementation in a Xilinx Device

As explained earlier, an alternate optimized implementation of RMP in a Xilinx XC2S400E-6FG456 device using ISE Foundation Series and ModelSim with user constraints and manual editing in placement and routing is also being presented. Figure 7.9a&b show the post-layout simulation of 32-bit program counter and instruction memory associated with Increment PC and Fetch stages, respectively. Instruction Memory is loaded when the program/ run (*pr*) signal is high. In the run mode preloaded instructions are read by the internal PC generated address, to be decoded in the following stage. Figure 7.10a&b present the post–layout simulation of RMP's multi-port regfile and ALU associated with Decode and Execute Stages. In order to test the read after write and multi-port capability of the register file, it is loaded with the contents of R1, R2, R3 and R4 shown in Table 7.2 and the instructions presented in the same table are provided to the decode stage to be decoded. The instructions generate the expected signals on the control bus and deliver the updated results at the appropriate ports of the register file. Post-layout simulation of the execute stage is done while using the contents provided in Table 7.3. It can be observed that the control signals generated by the decode stage, in context of the corresponding ALU instructions, perform the desired operations in the execute stage, thus verifying the correctness of this stage.

A feature to be noted in all these simulation results is that the request signal to a stage triggers the operation and the completion of the task for a particular stage is represented by the generation of acknowledge signal by that stage. As already explained in Chapter No. 6 the request signal turns–on the SIRO of this



(a)

(b)

Figure 7.9: Post–layout simulation of RMP using ModelSim XE for Xilinx device:
(a) Increment PC stage associated with 32–bit PC, (b) Fetch Stage associated with instruction memory.

stage and after the required dividing counter value, acknowledge signal is generated that also stops the SIRO and ensures the completion of task by the relevant stage.

Table 7.2
Register File contents and instructions decoded in Decode Stage

R1	00000003
R2	00000004
R3	00000005
R4	00000006
ADD R2, R3, R4	00432000
AND R2, R3, R4	20432000
SUB R6, R2, R1	18C20800
STI R4, R3, #7	0C830007
JMP #C	C000000C

Table 7.3
ALU instructions to test the Execute Stage

ALU Instruction	Control Bus *cbus*	Operand 1 *a* (hex)	Operand 2 *b* (hex)	Result *r* (hex)
ADD	000	00000002	00000001	00000003
SUB	001	00000002	00000001	00000001
AND	100	00000004	00000003	00000000
OR	101	00000003	00000004	00000007
SHL	110	00000007	00000004	00000070
SHR	111	00000007	00000001	00000003

Figure 7.11 presents the post–layout simulation of data memory associated with Memory Stage for the duration of 0~365nSec. The difference between the instruction and data memories is that data memory can be read as well as written into. However, it does not support read after write feature like the register file.

Figure 7.12 shows the post–layout simulation of the integrated RMP, where each stage runs at its own pace, because of its customized delay element.

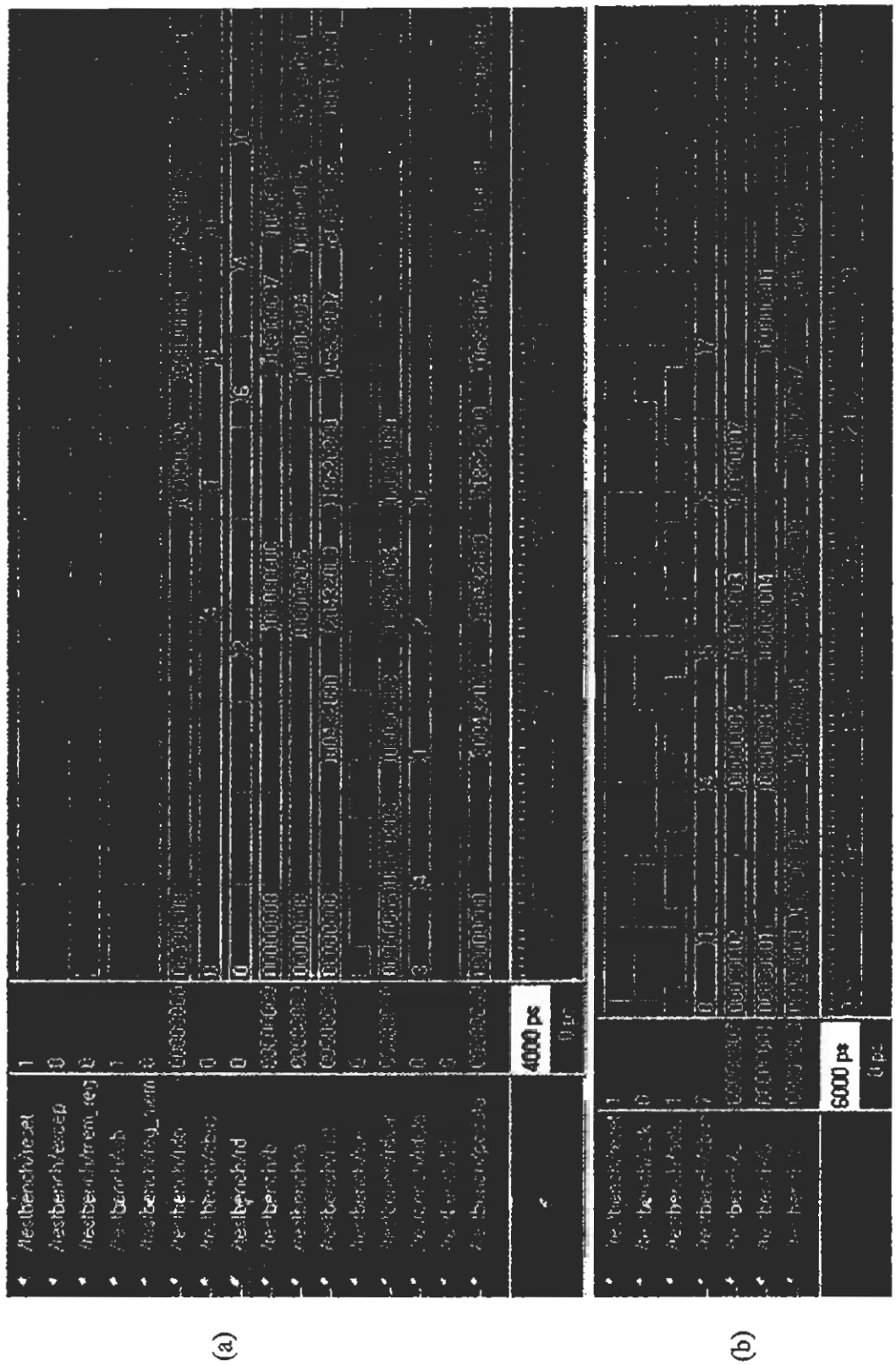


Figure 7.10: Post-layout simulation of RMP using ModelSim XE: (a) Decode Stage with Regfile, (b) Execute Stage with ALU.

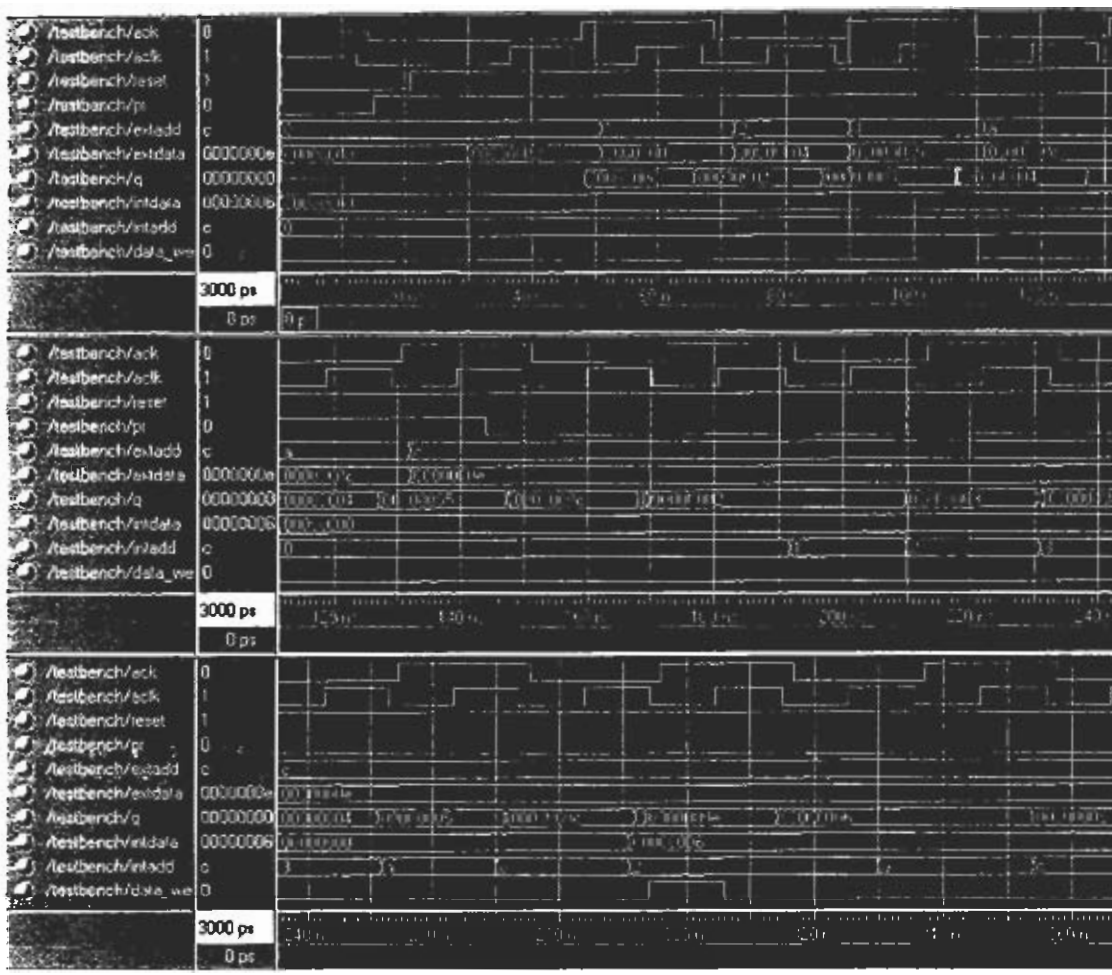


Figure 7.11: Memory stage associated with data memory (both read and write functions).

Stages go into sleep mode to conserve power upon completion of their assigned task, while the sluggish states are busy performing their duty.

As explained in Chapter No. 2, if any of the sub-circuits in an asynchronous design is optimized, it can directly be replaced without changing the entire circuit. In the edited implementation of RMP, it can be observed that some of the stages are optimized but the over all design remains the same thus, complying with the properties of full custom asynchronous implementations.

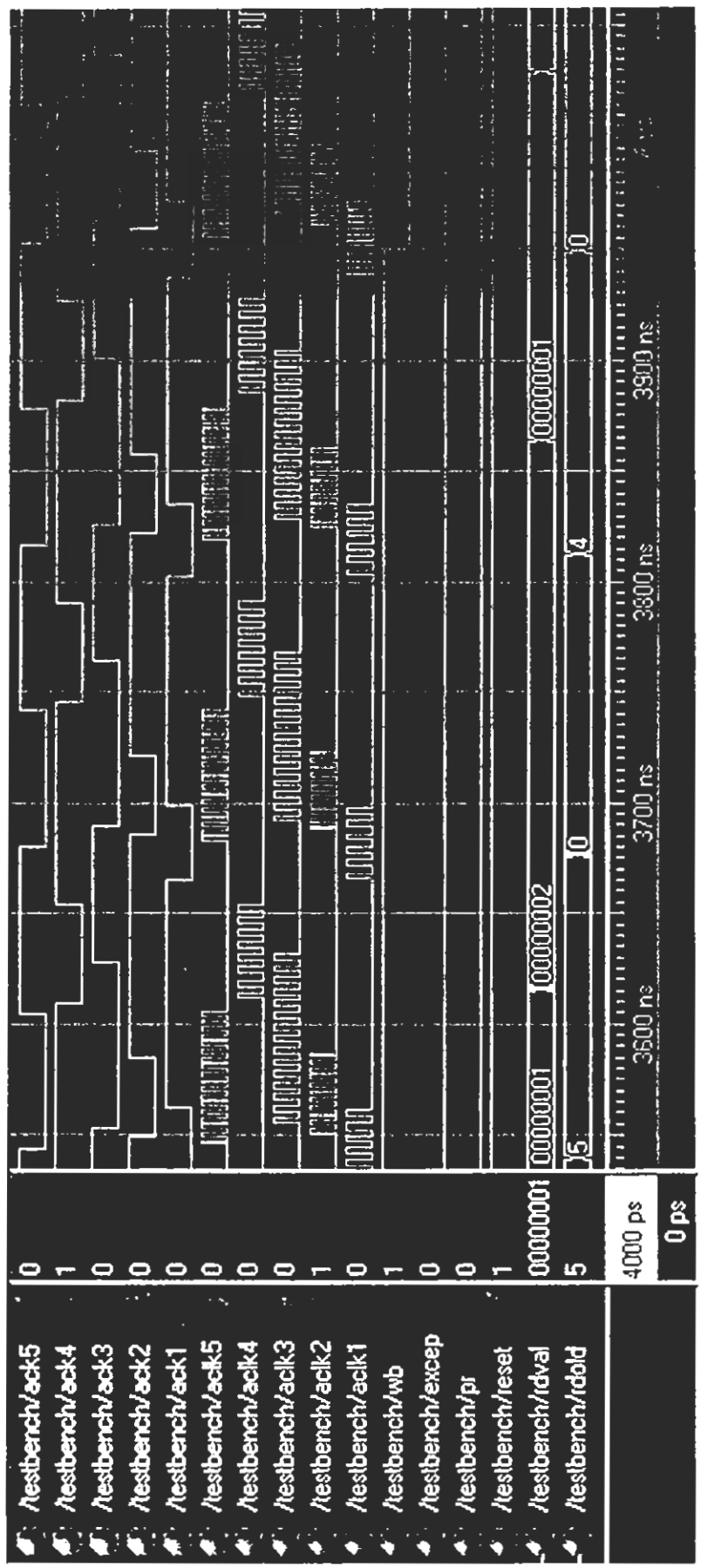


Figure 7.12: Post-layout simulation of RMP using ModelSim XE.

The micropipeline model presented in Chapter 6 was used to implement RMP therefore, the performance of the pipeline is the performance of the processor with the exception that branch instructions are associated with a stall. A comparison table for the RMP performance in Altera and Xilinx devices, assuming 1% branch instructions in Test program is presented in Table 7.4.

TABLE 7.4
RMP performance in Altera and Xilinx devices.

Device	SIRO 'nS'	LSO 'nS'	Avg. Data output 'nS' (no branch)	Throughput @ 1% branch instr. '10^6 Data items / S'
Altera EPF10K70RC240-2	4.0	9.8	193	5.13
Xilinx XC2s400e-6fg456	2.4	4.8	76 (*optimized implementation*)	13.03

8

Power Analysis

One of the major advantages of asynchronous circuits is power efficiency. It has been shown that the proposed methodology for the implementation of asynchronous circuits in reconfigurable medium of FPGAs, complies with all the properties of asynchronous circuits, making it a viable solution. The only topic that was not touched yet was the power analysis of the proposed model. Therefore, this chapter is dedicated to the power issues.

A lot of factors affect the power efficiency of a design. Majority of these factors are associated with the technology involved in the fabrication of an integrated circuit. However, if the technology is kept constant, different designs can then be evaluated for their power efficiency. This methodology can very easily be adopted in case of FPGAs, for they permit implementation of a variety of designs, while retaining consistency in technology. So a lot of parameters used in the power calculations are eliminated, and the calculations become solely dependent on the designs properties. In the current discussion, power analysis for three designs is presented for similar FPGA (in order to keep the technology constant). The first one is the simple externally clocked RISC, the second is the Simple RISC driven by on-chip SIRO based oscillator, presented in Chapter No. 5 and the third one is RMP.

It must be kept in mind that the platform remains the same in all these implementations, therefore the factors effecting the calculations, are mainly the real estate, frequency of the oscillator source and average toggle rate of the design. A brief discussion of the above mentioned parameters is necessary to establish their relationship with power calculations.

8.1 Factors Affecting Power Calculations

Toggling means a transition from low to high state or vice versa. In case of complementary MOS or CMOS technology, for a signal to toggle one of the paths i.e., the source or sink must be opened. N-type and P-type transistors gate these paths. For demonstration, lets consider a CMOS inverter, as shown in Figure 8.1.

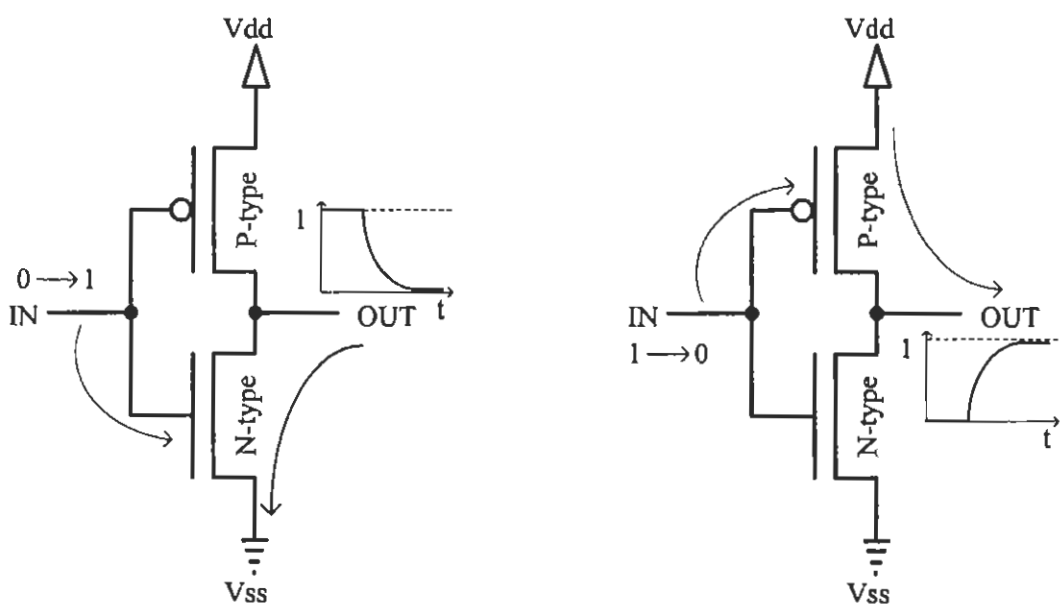


Figure 8.1: CMOS inverter representing toggling (a) 0 → 1, (b) 1 → 0.

A zero at the output means the N-type transistor opens the path to sink, while a 0 → 1 transition at the output means, the P-type transistor opens the path to the source. In both cases a transistor has to be activated to open the path. Now lets consider switching 'ON' of a transistor. Figures 8.2a&b show the transistor switching

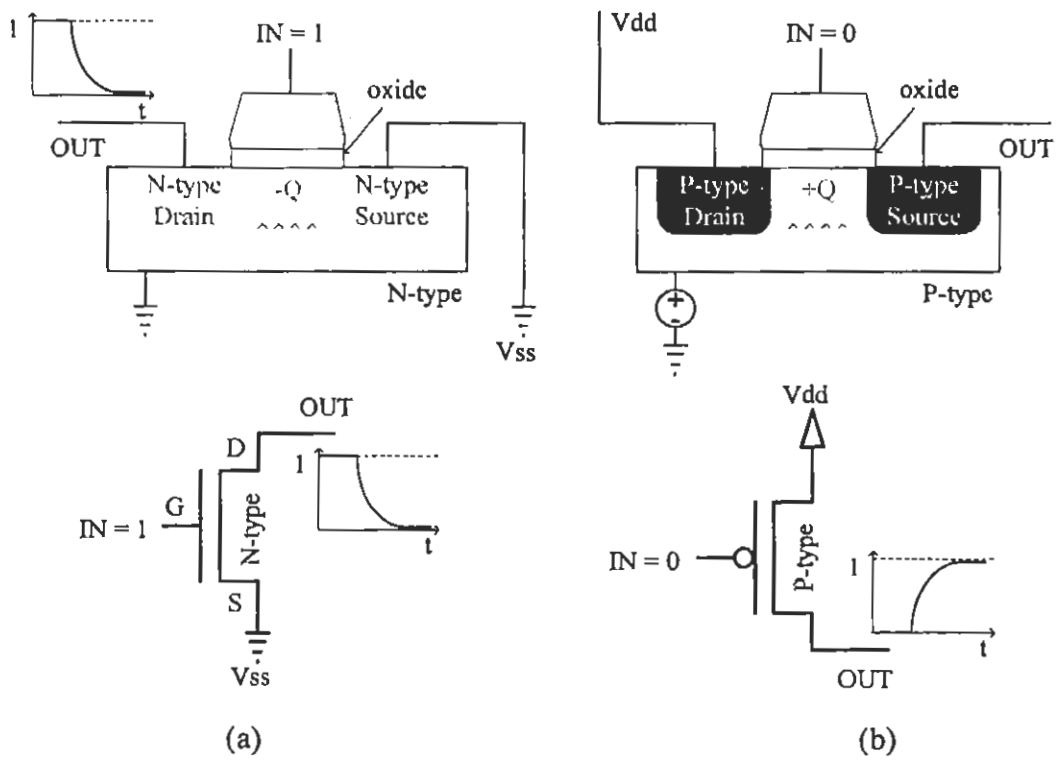


Figure 8.2: Switching of CMOS transistors used in the inverter design (a) N-type, (b) P-type.

in case of N-type and P-type transistors. In both the cases, the gate pulls the relevant carriers towards it, to form a channel that acts as a path between drain and source. The pulling of carriers by the gate and the forming of channel requires electric field, which is generated by consuming power. Larger the area of gate, greater is the size of channel, which is created by the pulling of greater number of carriers resulting in the consumption of larger amount of power. If the FPGA technology is kept constant by implementing various designs in the same FPGA, all the physical factors remain the

same, but different designs have different real estate. Larger real estate engages larger number of transistors. So technically a larger design must consume more power, but all the transistors might not be toggling all the time. For example, the least significant bit of a 2-bit counter made out of T-flip flops toggles at every clock edge, whereas the most significant bit toggles at half the rate. The rate of toggling for higher order bits in larger counter, is even less. So, if a design contains a 32-bit counter, and another one contains three 2-bit counters, the first one is larger in real estate, but has low average toggle rate, whereas, the second design, though small in real estate has greater average toggle rate, for its active sections at any given time are more than the first case. So it is very possible that the smaller design might consume more power than the larger one. Another issue is the frequency at which the design toggles. A 2-bit counter running at 10MHz will have its transistors switching ON for toggling at twice the speed as compared to the same run at 5MHz. Therefore, the frequency also plays a important role in determining the amount of power consumed by a design.

In short, if the technology is kept constant, the three factors that affect the power calculations are: a) the real estate of the design, b) average toggle rate and c) the operating frequency.

In case of synchronous designs power consumption is high, because all the sections of such a system, whether required in the current manipulation or not, are always driven by the master clock. So they consume power all the time, whereas in case of asynchronous systems, during a processing only the relevant portions are active whilst other portions are in sleep mode, resulting in reduced power consumption.

In FPGAs, the real estate of a design is calculated on the basis of CLBs it consumes. For a given FPGA, its CLB may consist of one or more similar slices and the real estate associated with slices is known. The other factor i.e., the average toggle rate is calculated by separating various sections of the design. As per their design characteristics an average toggle rate value is assigned to each section, as suggested by the FPGA vendor [POWS2]. The operational frequency in case of synchronous designs is also known.

All these parameters are entered in the power estimator sheet for the FPGA [POWS2] containing the design. These sheets are available from the vendor and are FPGA family specific, as they contain fundamental information such as the device quiescent power dependent on the FPGA technology. Quiescent power and the estimated power for the design, amount to the overall power consumption of a synchronous design. Xilinx Xpower plug-in to the ISE Foundation series that calculates the power for the entered design also performs power calculations using these estimator sheets.

There are however, few other factors that effect the power estimation. For example, IOs, clock tree and the DLLs, they all consume power. Therefore, use of any of these elements also effects calculations in the estimator sheets.

The standard power calculating techniques, in the FPGA environments are customized towards synchronous designs. Therefore, in order to calculate power for asynchronous designs that may have more than one or no clock sources, a different technique must be improvised to use the available power calculations resources.

Figure 8.3 shows various parameters enlisted in Power Estimator Sheet for Xilinx Spartan 2S [POW2S], that that are requires for the calculation of expected power

Device Quiescent Power										
										VCCint Subtotal (mW)
CLB Logic Power										
Name	Frequency (MHz)	Total Number of CLB Slices	Total Number of Flip/Flop or Latches		Total Number of Shift Register LUTs	Total Number of Select RAM LUTs	Average Toggle Rate %		Amount of Routing Used	VCCint Subtotal (mW)
Block SelectRAM Power										
Name	Total Number of RAMB4 Cells	Port A Frequency (MHz)	Port A Width	Port A Read Rate %	Port A Write Rate %	Port B Frequency (MHz)	Port B Width	Port B Read Rate %	Port B Write Rate	VCCint Subtotal (mW)
Clock Delay Locked Loop Power										
Name	Clock Input Frequency (MHz)	DLL Frequency Type								VCCint Subtotal (mW)
Input/Output Power										
Name	Frequency (MHz)	IO Standard Type	Total Number of Inputs		Total Number of Outputs	Average IOB Toggle Rate %	Average Output Load (pF)		VCCint Subtotal (mW)	VCCo Subtotal (mW)

Figure 8.3: Various parametcrs contributing towards the power calculations in Xilinx Power Estimator Sheet [POWS2].

consumption of a given design when implemented in Spartan 2S family FPGA. Most of the required parameters are obtained during the implementation process of a design in ISE Foundation Series upon automatic generation of mapper report file (.mrp). The remaining fields are directly filled with the help of comments associated with blank enteries in the excel sheet. Scripts from the sample mapper report file are presented in Figure 8.4 for reference.

```
 Total Number Slice Registers:    191 out of 9,600   1%
  Number used as Flip Flops:          181
  Number used as Latches:             10
 Number of 4 input LUTs:          101 out of 9,600   1%
Logic Distribution:
  Number of occupied Slices:                141 out of 4,800   2%
  Number of Slices containing only related logic:   141 out of   141 100%
  Number of Slices containing unrelated logic:      0 out of   141   0%
    *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:        119 out of 9,600   1%
    Number used as logic:               101
    Number used as a route-thru:         18
  Number of bonded IOBs:          43 out of   325   13%
    IOB Flip Flops:                   32

Total equivalent gate count for design: 2,564
Additional JTAG gate count for IOBs: 2,064
Peak Memory Usage: 64 MB
```

Figure 8.4: Scripts from the automatically generated .mrp file for estimator sheet fields.

Power was estimated with the help of estimator sheets for externally clocked RISC, on-chip SIRO triggered RISC and RMP, for mutual comparison.

It was observed that the externally clocked and on-chip SIRO triggered synchronous versions of simple RISC consumed, approximately, the same amount of power. The excess power consumed by the on–chip oscillator in case of the externally clockless

version was comparable to the power consumed by external clock input in case of the clocked version, when implemented in XC2S400E-6FG456 device. The FPGA compliant micropipeline based RMP was found to be 20% more power efficient than its synchronous counterpart. In case of RMP, relative area occupation of the functional units, datapath and delay elements, was found to be 39.5%, 57% and 3.5% respectively. A few parameters of power estimator sheet are presented in Figure 8.5.

Name	Frequency (MHz)	Total Number of CLB Slices	Total Number of Flip/Flop or Latches	Average Toggle Rate %	Amount of Routing Used	VCCint Subtotal (mW)	% real-estate of total
stage1_Async	208	72	37	4%	Medium	8	12.19512
datapath_stage1	208	63	1	1%	Medium	1	
stage2_Async	208	23	6	6%	Medium	6	13.4598
datapath_stage2	208	126	2	1%	Medium	2	
stage3_Async	208	315	170	8%	High	75	45.52846
datapath_stage3	208	189	3	1%	Medium	3	
stage4_Async	208	39	6	6%	Medium	5	14.90515
datapath_stage4	208	126	2	1%	Medium	2	
stage5_Async	208	28	11	7%	Medium	9	13.91147
datapath_stage5	208	126	2	1%	Medium	2	
Total		**1107**				**113**	**100**

(a)

stage1_sync	208	65	33	10%	Medium	15	14.84018
stage2_sync	208	22	7	12%	Medium	7	5.022831
stage3_sync	208	300	150	13%	High	108	68.49315
stage4_sync	208	32	0	10%	Medium	6	7.305936
stage5_sync	208	19	0	12%	Medium	7	4.3379
Total		**438**				**143**	**100**

(b)

design (overall %age)	datapath overhead (overall %age)	siro overhead (overall %age)	power async **'Pa'** (mW)	power sync **'Ps'** (mW)	rel power % = (Ps - Pa) * 100 /Ps
39.5664	56.91057	3.523035	113	143	20.97902

(c)

Figure 8.5: Power calculations for (a) RMP, (b) Synchronous RISC, (c) mutual comparison.

It is worth mentioning that in order to use standard estimator sheet developed to suit synchronous designs, special methodology was adopted for power estimation of asynchronous RMP. As our micropipeline is a Globally Asynchronous, Locally Synchronous system, a particular frequency is not associated with complete RMP. Therefore, using power estimator, power consumption of each stage was calculated individually, by using SIRO frequency for XC2S400E-6FG456. It was observed in case of asynchronous version that the reduction in 'average toggle rate' because of the sleep-mode in various stages as shown in Figures 7.8 and 7.12, reduced power consumption much more than the increase caused by real-estate overhead due to delay-elements, 4-phase handshaking circuitry and unbundled datapath. The performance of the developed system directly depends on the host FPGA, as the latch synchronizing circuit adjusts the SIRO output to match the maximum permissible frequency of the device.

9

Conclusion and Future Research Plans

Latest trends in digital electronics require designs with increased functionality, contained in the smallest of spaces, performing at the highest of speeds, consuming and dissipating the minimum of power and generating the least of electromagnetic interference. The merger of all these properties in a single synchronous design is becoming increasingly difficult, as one property contradicts the other. Consequential issues such as clock skews, power and real estate overheads, and design modifications associated with upgrades, resulting in slow turn around time in a highly competitive market, have forced researchers to look for alternate solutions like asynchronous designs. Asynchronous systems exhibit technology independence, power efficiency, average case computational capability and electromagnetic compatibility. Above all, the problem of clock skews does not exist in asynchronous systems because of the absence of a common clock.

Reconfigurable mediums such as Field Programmable Gate Arrays (FPGAs) have become increasingly popular since the 90s because of lower costs and lesser turn around time associated with prototyping. Additionally, reconfigurable computing has made it possible for a design to perform like ASIC while retaining the flexibility of

general purpose IC. Reconfigurable mediums are power efficient for applications that require diverse resources, as they maintain real estate, while re-allocating existing resources optimally.

FPGAs are extensively used today for the implementation of synchronous systems. In fact, the very structure of FPGAs and associated programming environments support synchronous designs, whereas asynchronous designs are dependent on their delay model. For an asynchronous design to be technology independent, delays in the design must have dynamic calibration capability. A delay element with these characteristics is considered to be implausible in case of FPGAs. Isochronic forks in the delay model also require full custom implementation. The problem is aggravated by the fact that design implementations in FPGAs are not necessarily repeatable, in various programming environments because of the stochastic processes involved and the variety in reduction, placement and routing algorithms.

As a result, asynchronous systems find their place in full custom domain while reconfigurable computing is associated with synchronous designs. In the presented research, the two fields are combined to come up with a technique through which an asynchronous system like a 4-phase micropipeline; an event based pipeline with or without processing; has been implemented in a reconfigurable medium of FPGAs.

Available models for the fundamental building blocks of full–custom micropipeline, such as the delay elements, Event Controlled Registers (ECRs) and hand shaking protocols could not be used, rather new concepts and techniques had to be developed for their FPGA based implementation.

It was observed that a Single Inverter Ring Oscillator 'SIRO' (A = ~A), that cannot be functionally simulated, can be implemented in an FPGA and its behavior is observable in post–layout simulation. It was also observed that SIRO adapts to the FPGA technology by altering its frequency thus, exhibiting technology independence. SIRO always consumes a single Logic Element (LE) of a LUT–based FPGA, as it consists of a single primitive and its feedback loop always uses the local / direct interconnects, irrespective of the FPGA or its associated environment. Technically, SIRO is a combinatorial element, thus it follows the combinatorial path within LE. On the other hand, the maximum operational frequency defined for an FPGA, depends on the setup and hold time requirements of its sequential path. Therefore, the SIRO frequency is too high for the sequential elements to function. For this reason, a special Latch Synchronizing Circuit (LSC) was developed that automatically adjusts SIRO frequency to meet the requirements of sequential path. It was also observed that the maximum frequency locked by Delay–Locked Loop (DLL) in a FPGA, is the same as the SIRO frequency for that FPGA. Output of LSC called Latch Synchronized Oscillator (LSO) can drive co–existing sequential circuits such as counters. If FPGA technology permits higher frequency operations, SIRO adapts to the same technology by enhancing its frequency. Thus, the entire circuit becomes technology independent.

A simple 32-bit RISC processor was driven externally by clock while being implemented in various FPGAs and was then driven by co-existing on-chip SIRO based oscillator. It was observed that the externally clocked RISC required a change in clock source upon FPGA technology variation, while the on-chip SIRO based oscillator, automatically adjusted its frequency along with the circuit driven by it, to give optimal performance without the requirement of any design modification. The

technology independent SIRO driven RISC was found to consume the same amount of power as its externally clocked counterpart. However, the SIRO based circuit has better Electromagnetic Compatibility (EMC) because of being off–board and on–chip. The EMC is a real issue for synchronous systems operating beyond 100MHz.

The adaptive nature of SIRO was used to drive counters with parameterized depth *N*. As counters divide the frequency of triggering source, these counters with their own SIRO and LSC were placed as delay elements in FPGA compliant micropipeline stages. By changing the value of *N*, delay for different stages was controlled as per their processing requirements. As discussed earlier, sequential circuits driven by SIRO based oscillator exhibit technology independence, so the SIRO based delay elements in the micropipeline model for reconfigurable medium, possess the same feature as exhibited by their full custom counterparts.

The uncertainty associated with repeatability in datapath implementation in various FPGAs, was covered by developing special ECRs consisting of Muller's C-elements. The datapath in FPGA–compliant micropipeline follows unbundled data strategy incorporating bit encoding and return to zero schemes as apposed to bundled data strategy in full custom micropipeline implementations.

The handshaking protocol in the presented micropipeline model also involves additional signals between SIRO–based delay element of the stage and it's ECR, to force the stage into sleep mode upon completion of its task to conserve power.

As an application, to validate the FPGA–compliant micropipeline model, a Reconfigurable Micropipelined Processor (RMP) based on the simple RISC architecture was implemented. It was observed that each stage in the RMP functioned

at its own pace driven by its SIRO and LSO. Furthermore, the sluggish stages went into sleep mode while waiting for faster stages to accept or deliver data. The design was portable to various FPGAs and related environments without the requirement of any modification thus, exhibiting technology independence. The RMP was estimated to be 20% more power efficient than its synchronous counterpart.

Therefore, the presented methodology can import asynchronous designs along with their benefits to the traditionally synchronous environment of FPGAs. The merger of asynchrony and reconfigurability may reshape computer architecture in future.

The inspiration in the development of RMP architecture comes from DLX architecture [DAP96]. DLX is a synchronous pipelined architecture, which has a floating point unit and is further expanded to incorporate features like dynamic scheduling. Development of RMP is the first step in formalizing technique that imports such architecture to an asynchronous domain implementable in a reconfigurable medium. In this thesis, the emphasis was on the development of methodology and its verification by actual implementation. For this reason, the RMP architecture was kept minimal, though complete.

Now that the technique has been formalized, the next step is to develop a full scale micropipelined processor, implementable in a reconfigurable medium that can compete with its counterparts in performance and related issues. So the plans are to concentrate the research efforts on the following issues:

- ⇨ Addition of a Floating Point Unit in RMP
- ⇨ Incorporation of Dynamic Scheduling with Scoreboard
- ⇨ Incorporation of Dynamic Scheduling using Tomasulo Algorithm

⇨ Conversion of RMP into a Reconfigurable Multiple–Issue Processor in both forms i.e. Superscalar and VLIW (Very Long Instruction Word).

References

[AIR02] A. Iyer, D. Marculescu, "Power and performance evaluation of globally asynchronous locally synchronous processors", Proceedings of the 29th annual international symposium on Computer architecture, Alaska, 2002, pp. 158-168.

[ADA94] A. de Angel and E. Swartzlander Jr., "A new asynchronous multiplier using enable/disable CMOS differential logic", In Proceedings of International Conference on Computer Design (ICCD), IEEE Computer Society Press, October 1994.

[AJM89] A. J. Martin, "Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits", in *UT Year of Programming Institute on Concurrent Programming,* C. A. R. Hoare, Ed. MA: Addison-Wesley, 1989.

[AJM90] A.J. Martin, "The limitation to delay-insensitivity in asynchronous circuits", In W.J. Dally, editor, Advanced Research in VLSI: Proceedings of the Sixth MIT Conference,. MIT Press, Cambridge, MA, 1990, pp 263-278.

[AJM89] Alain J. Martin, Steven M. Burns, T. K. Lee, Drazen Borkovic, and Pieter J. Hazewindus, "The design of an asynchronous microprocessor", In Advanced Research in VLSI (Charles L. Seitz: editor): Proceedings of the Decennial Caltech Conference on VLSI, MIT Press, 1989, pp. 351-373.

[AJM92] A. J. Martin, "Asynchronous datapaths and the design of an asynchronous adder", Formal Methods in System Design, vol. 1, no. 1, July 1992, pp. 119-137.

[AJM97] A.J. Martin, A. Lines, R. Manohar, M. Nystrom, P. Penzes, R. Southworth, U. Cummings and T.K. Lee, "The Design of an Asynchronous MIPS R3000 Microprocessor", Proceedings of 17th Conference on Advanced Research in VLSI, September 1997.

[ALD78] A.L. Davis. "The architecture and system method for DDM1: A recursively structured data-driven machine". 5th Annual Symposium on Computer Architecture, April 1978.

[ATA97] A. Takamura, M. Kuwako, M. Imai, T. Fujii, M. Ozaw, I. Fukasaku, Y. Ueno and T. Nanya, "TITAC2: An Asynchronous 32-Bit Microprocessor Based on Scalable-Delay Insensitive Model", Proceedings of ICCD'97, October 1997, pp. 288-294.

[AYV95] A. Yakovlev, V. Varshavsky, V. Marakhovsky, and A. Semenov, "Designing an asynchronous pipeline token ring interface", In Asynchronous Design Methodologies, IEEE Computer Society Press, May 1995, pp. 32-41.

[BLH95] B. L. Hutchings and M. J. Wirthlin, "Implementation approaches for reconfigurable logic applications", in Field-Programmable Logic and Applications (FPL'1995) (editors: W. Moore and W. Luk), (Oxford, England), Springer-Verlag, Berlin, August 1995, pp. 419-428.

[CEM95] Charles E. Molnar and Huub Schols, "The Design Problem SCPP-A Technical Report: TR-95-49", Sun Microsystems Laboratories, USA, December 1995.

[CEM97] C. E. Molnar, I. W. Jones, W. S. Coates, J. K. Lexau, S. M. Fairbanks, and I. E. Sutherland, "Two FIFO Ring Performance Experiments", 3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC '97), Netherlands, April, 1997.

[CGW03] C. G. Wong, A. J. Martin and P. Thomas, An Architecture for Asynchronous FPGAs, Technical Report, IEEE Computer Society Press, 2003.

[CMD80] C. Mead and L. Conway. Introduction to VLSI Systems, chapter 7. Addison-Wesley, Reading, MA, 1980. C.L. Seitz, System Timing.

[CRP92] C. R. Paul, "Introduction to Electromagnetic Compatibility", John Wiley & Sons, 1992.

[CSC01] C-sing Choy, J. Butas, J. Povazanec and C-fat Chan, "A New Control Circuit for Asynchronous Micropipelines", IEEE Transactions on Computers, vol. 50, no. 10, October 2001, pp. 992-997.

[DAM00] David A. Miller, "Rationale and Challenges for Optical Interconnects to Electronic Chips", Invited Paper, Proceedings of the IEEE, vol. 88, no. 6, June 2000.

[DAP96] D. A. Patterson and J. L. Hennessy, "Computer Architecture–A Quantitative Approach", 2nd Ed., Morgan Kaufmann Publishers, 1996.

[DAP03] D. A. Patterson and J. L. Hennessy, "Computer Architecture–A Quantitative Approach", 3rd Ed., Morgan Kaufmann Publishers, 2003.

[DLLS2] Using Delay-Locked Loops in Spartan-II FPGAs, Application Note: Spartan-II FPGAs, www.xilinx.com/bvdocs/appnotes/xapp174.pdf.

[DLLVE] Virtex-E DLL Verilog and VHDL example files, ftp://ftp.xilinx.com/pub/applications /xapp/xapp132.zip.

[DLR00] D. Lauder and J. Moritz, Investigation into possible effects resulting from dithered clock oscillators on EMC measurements and interference to radio transmission systems, Final Report prepared for the Radio Communications Agency (AY3377 (510001891)), Univ. of Hertfordshire, March, 2000.

[DMC84] Daniel M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems", PhD thesis, Stanford University, October 1984.

[DRS93] D. Ross, O. Vellacott, and M. Turner, "An FPGA-based Hardware Accelerator for Image Processing," in More FPGAs: Proceedings of the

1993 International workshop on field-programmable logic and applications (editors: W. Moore and W. Luk), (Oxford, England), 1993, pp. 299-306.

[EBD93] E. Brunvand, "The NSR Processor", Proceedings of the 26th International Conference on System Sciences, January 1993.

[FLX10] Data sheets for Altera FLEX10K family. www.altera.com

[HJM86] H. J. Mitchell, "32-bit Microprocessors", Collins, 1986.

[HTA99] H. Terada, S. Miyata, and M. Iwata, "DDMPS: Self-timed super-pipelined data-driven multimedia processors", Proceedings of the IEEE, vol. 87, no. 2, Feb 1999.

[HVG98] H. Van Gageldonk, D. Baumann, K. van Berkel, D. Gloor, A. Peeters and G. Stegmann, "An Asynchronous Low-Power 80C51 Microcontroller", Proceedings of Async'98, April 1998.

[HWO88] H. W. Ott, "Noise reduction techniques in electronics systems", John Wiley & Sons, 1988.

[IES89] I. E. Sutherland, "Micropipelines", *Communications of the ACM*, Vol. 32, no. 6, June 1989, pp. 720-738.

[IES01] Ivan E. Sutherland and Jon K. Lexau. "Designing Fast Asynchronous Circuits", Proceedings of the Seventh International Symposium on Advanced Research in Asynchronous Circuits and Systems, USA, March 2001, pp. 184-193.

[JBT99] J. Silc, B. Robic and T. Ungerer, "Processor Architecture–From Dataflow to Superscalar and Beyond", Springer, 1999.

[JDG93] Jim D. Garside, "A CMOS VLSI implementation of an asynchronous ALU", In Asynchronous Design Methodologies (S. Furber and M. Edwards: editors), volume A-28 of IFIP Transactions, Elsevier Science Publishers, 1993, pp. 181-207.

[JDH95] J. D. Hadley and B. L. Hutchings, "Designing a partially reconfigured system," in Ficld Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing, *Proc. SPIE 2607* (editor: J. Schewel), (Bellingham, WA), SPIE – The International Society for Optical Engineering, 1995, pp. 210-220.

[JDM01] J. D. Meindl, R. Venkatesan, J. A. Davis, J. Joyner, A. Naeemi, P. Zarkesh-Ha, M. Bakir, T. Mule, P. A. Kohl and K. P. Martin, "Interconnecting Device Opportunities for Gigascale Integration (GSI)", in International Electron Devices Meeting (IEDM) Technical Digest, 2001, pp. 525-528.

[JMD00] J. M. Ditmar, "A Dynamically Reconfigurable FPGA-based Content Addressable Memory for IP Characterization," Master's thesis, KTH-Royal Institute of Technology, Stockholm, Sweden, 2000.

[JTL04] J. Teifel, R. Manohar, Highly pipelined asynchronous FPGAs, Proceeding of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays, February 22, 2004, pp. 133 – 142.

[JTU86] J. T. Udding, "A formal modelfor defining and classifying delay-insensitive circuits and systems", Distributed Computing, vol. 1, no. 4, 1986, pp.197-204.

[KLE93] K. Lee, M. Shur, T. A. Fjeldly and T. Ytterdal, "Semiconductor Device Modeling for VLSI", Prentice Hall, New Jersey, 1993.

[KVB92] Kees van Berkel, "Beware the isochronic fork", Integration, the VLSI journal, vol.13, no.2, 1992, pp.103-128.

[KYY95] K. Y. Yun and D. L. Dill, "A high-performance asynchronous SCSI controller", In Proceedings of International Conference on Computer Design (ICCD), October 1995, pp. 44-49.

[LLO93] L. Lavagno and A. Sangiovanni-Vincentelli, "Automated synthesis of asynchronous interface circuits", In Asynchronous Design Methodologies (S. Furber and M. Edwards, editors), volume A-28 of IFIP Transactions, Elsevier Science Publishers, 1993, pp. 107-121.

[MBJ92] M. B. Josephs, R. H. Mak, J. T. Udding, T. Verhoeff, and J. T. Yantchev, "High-level design of an asynchronous packet-routing chip", In Designing Correct Circuits (J. Staunstrup and R. Sharp: editors), volume A-5 of IFIP Transactions, Elsevier Science Publishers, 1992, pp. 261-274.

[MED92] M.E. Dean. "STRiP: A Self-Timed RISC Processor Architecture". PhD thesis, Stanford University, 1992.

[MIM96] M. I. Montrose, "Printed Circuit Board Design Techniques for EMC Compliance", IEEE Press, New York, 1996.

[MJS97] Micheal J. S. Smith, "Application-Specific Integrated Circuits", Addison Wesley, 1997.

[NDJ00] N. D. Jankovic and V. Brajovic, "Light–sensitive CMOS ring oscillator", Electronic Letters, vol. 36 no. 15, 20th July 2000, pp. 1281–1283.

[OAP97] O. A. Petlin and S. B. Furber, "Built-in self-testing of micropipelines", Proc. 3rd Int. Sym. on Adv. Research on Asynchronous Circuits and Systems, April 1997, pp. 22-29.

[POWS2] Power Estimator sheet for SPARTAN IIE. http://www.xilinx.com/ise/ power_tools/license_spartan2e.htm

[REM65] R. E. Miller, "Sequential Circuits", *Switching Theory*, Vol. 2: Sequential Circuits and Machines, John Wiley and Sons, NY, 1965.

[REP96] R. E. Payne, "Asynchronous FPGA Architectures", IEE Proc. on Computers and Digital Techniques, Special Issue on Asynchronous Processors, vol. 143, no. 5, Sept. 1996.

[RFR98] R. Farjad-Rad, C. K. Yang, M. Horowitz, and T. H. Lee, "A 0.4 mm CMOS 10 Gb/s 4-PAM pre-emphasis serial link transmitter," in *Symp. VLSI Circuits Dig. Tech Papers*, June 1998, pp. 198–199.

[RFS94] R.F. Sproull, I.E. Sutherland, and C.E. Molnar, "The Counter Flow Pipeline Processor Architecture", IEEE Design & Test of Computers, vol. 11, no. 3, Fall 1994, pp. 48–59.

[RFT91] Richard F. Tinder, "Digital Engineering Design–A Modern Approach", Prentice Hall, 1991.

[RSS90] Richard S. Sandige, "Modern Digital Design", McGraw-Hill Inc., 1990.

[RWR87] R. Walker, "A Monolithic High-Speed Voltage Controlled Ring Oscillator", Proceedings of the Hewlett-Packard 1987 VLSI Design Technology Conference, May 18-20, 1987, S10.6.1-5.

[SBF94] S.B. Furber, P. Day, J.D. Garside, N.C. Paver and J.V. Woods, "The Design and Evaluation of an Asynchronous Microprocessor", *Proceedings of ICCD'94*, October 1994, pp. 217-220.

[SBF96] S. B. Furber and Paul Day, "Four-phase micropipeline latch control circuits", IEEE Transactions on VLSI Systems, vol. 4, no. 2, June 1996, pp. 247-253.

[SBF97] S.B. Furber, J.D. Garside, S. Temple, J. Liu, P. Day and N.C. Paver. "AMULET2e: An Asynchronous Embedded Controller", Proceedings of Async'97, pp. 290-299, April 1997.

[SBF98] S.B. Furber and J. D. Garside, "AMULET3: A High–Performance Self–Timed ARM Microprocessor", Proceedings of IEEE International Conference on Computer Design, Austin, October 1998.

[SBF02] S. B. Furber, "Validating the AMULET Microprocessors", The Computer Journal, vol. 45, no.1, 2002, pp.19-26.

[SHK95] Scott Hauck, "Asynchronous Design Methodologies: An Overview", Proceedings of the IEEE, Vol. 83, No. 1, pp. 69-93, January, 1995.

[SHK98] S. Hauck, "The roles of FPGAs in reprogrammable systems," Proceedings of the IEEE, vol. 86, April 1998, pp. 615-638.

[SLB02] S. Lopez-Buedo, J. Garrido, and E. Boemo, "Dynamically Inserting, Operating, and Eliminating Thermal Sensors of FPGA-based Systems", IEEE Transactions on Components and Packaging Technologies (CPM), vol. 25, no. 4, December 2002, pp.561-566.

[SMM00] S. McMillan and S. Guccione, "Partial run-time reconfiguration using JRTR," in Field-Programmable Logic and Applications / The Roadmap to Reconfigurable Computing (FPL'2000), Austria, August 2000, pp. 352-360.

[SMN93] Steven M. Nowick, Mark E. Dean, David L. Dill, and Mark Horowitz. "The design of a high-performance cache controller: a case study in asynchronous synthesis", Integration, The VLSI journal, vol. 15, no. 3, October 1993, pp. 241-262.

[SMN97] S. M. Nowick and A. Davis, "An Introduction to Asynchronous Circuit Design", Technical Report UUCS-97-013, Computer Science Department, University of Utah, Sep. 1997.

[SMN99] S. M. Nowick, K.van Berkel and M.B. Josephs, "Scanning the technology: Applications of asynchronous circuits", Proceedings of the IEEE special issue on asynchronous circuits & systems, vol. 87, no.2, Feb. 1999, pp.223–233.

[SMS96] S. M. Sze and C. Y. Chang, "ULSI Technology", McGraw-Hill, 1996.

[SSS95] S. Segars, K. Clarke and L. Goudge, "Embedded Control Problems: Thumb, and the ARM7TDMI", IEEE Micro, vol. 15, no. 5, October 1995, pp. 22-30.

[TEW91] T.E. Williams, "Self-timed rings and their application to division", Technical Report: CSL-TR-91-482, Computer Systems Laboratory, Stanford University, Ph.D. Thesis, June 1991.

[TEW91a] T. E. Williams and M. A. Horowitz, "A zero-overhead self-timed 160ns 54b CMOS divider", IEEE Journal of Solid-State Circuits, vol. 26, no. 11, November 1991, pp. 1651-1661.

[TMI04] T. Miyazaki, M. Hashimoto and H. Onodera, "A Performance Comparison of PLLs for Clock Generation Using Ring Oscillator VCO and LC Oscillator in a Digital CMOS Process", Proceedings of the 2004 Asia and South Pacific Design Automation Conference (ASP-DAC'04), 2004.

[TYA99] T. Yoneda, "Abstracted Instruction Cache of TITAC2 – As a Benchmark Circuit for Timed Asynchronous Circuit Verification", Tokyo Institute of Technology Technical Report, 99TR-0002, 1999.

[VTX25] Virtex 2.5V FPGA family datasheets. www.xilinx.com

[WAC67] W. A. Clark, "Macromodule Computer Systems", Proceedings of the Spring Joint Computer Conference, AFIPS, April 1967.

[WAC73] W.A. Clark and C.E. Molnar. "Macromodular system design: Technical Report –23", Computer Systems Laboratory, Washington University, April 1973.

[WFR96] W.F. Richardson and E. Brunvand, "Fred: An architecture for a self-timed decoupled computer", IEEE Int. Symposium on Advanced Research in Asynchronous Circuits and Systems, 1996.

[WJB01] W. J. Baindridge and S. B. Furber, "Delay Insensitive System-on-Chip Interconnect using 1-of-4 Data Encoding", 7th Int. Sym. on Asynchronous Circuits and Systems, March 2001, pp. 118-126.

[WSC93] W. S. Coates, A. L. Davis, and K. S. Stevens, "The Post Office experience: Designing a large asynchronous chip", Integration, the VLSI Journal, vol. 15, no.4, 1993, pp. 341-366.

[WSC01] W. S. Coates, J. K. Lexau, I. W. Jones, S. M. Fairbanks and I. E. Sutherland, "FLEETzero: An Asynchronous Switching Experiment", Proceedings of the Seventh International Symposium on Advanced Research in Asynchronous Circuits and Systems, USA, March 2001, pp. 173-182.

[XXC30] Xilinx XC3000 FPGA family datasheets. www.xilinx.com

[XXPOW]Xilinx XPowerTutorial, FPGA Design, Xilinx Inc., SanJose, California, 2002.

[YZR04] Y. Zafar and M. M. Ahmad, "A Novel FPGA Compliant Micropipeline", IEEE Transactions on Circuits & Systems II, vol. 52(9), 2005, pp. 611-615.

[YZR5a] Y. Zafar and M. M. Ahmad, "Globally Asynchronous Locally Synchronous Micropipelined Processor Implementation in FPGA", IEEE ICET'05, September 2005.

[YZR5b] Y. Zafar and M. M. Ahmad, "Adaptive On–chip Ring Oscillator for FPGAs Based Clocked Circuits", IEE Electronic Letters, (submitted).

[YZR5c] Y. Zafar and M. M. Ahmad, "Adaptive On–chip Oscillator for FPGA based Synchronous Designs", IEEE ICET'05, September 2005.

[YZR5d] Y. Zafar and M. M. Ahmad, "A Micropipelined Processor implementation in a reconfigurable medium", ETRI Journal. (submitted).