A PIPELINED MULTIPLIER ACCUMULATOR FOCUSING ON PIPELINE ARCHITECTURE

By

LAILI ERNI HITAM

WEK000336

A final year project submitted to the
Faculty of Computer Science and Information Technology
University of Malaya
In fulfillment of the requirements for the degree of
Bachelor of Computer Science

Session 2002/2003

A PIPELINED MULTIPLIER ACCUMULATOR
FOCUSING ON PIPELINE ARCHITECTURE

BY:

LAILI ERNI HITAM

DEPARTMENT OF SYSTEM AND COMPUTER TECHNOLOGY

SESSION 2002/2003

FACULTY OF COMPUTER SCIENCE
AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA

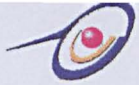
ABSTRACT

The Pipelined Multiplier Accumulator has brought many approaches and changes to the conventional MAC. The Pipelined MAC is believe can speed up the operations of the previous conventional MAC plus it can reduce the time and cost. The effectiveness of pipelining concept in the system's implementation is undeniable as the performance of the system is improved where the multiplication process is pipelined with the addition process. Pipelining reduces cycle time but does not reduce the total time required for multiplication. One way to speed up multiplication is Booth Algorithm, which perform several steps of the multiplication at once. Booth's algorithm takes advantage of the fact that an adder-subtractor is nearly as fast and small as a simple adder. The implementation of Accumulator using Carry Look-ahead Adder (CLA) technique has brought to fast operation achieved in addition process of 8-bit data. The pipeline MAC is designed to increase the speed of MAC operations, decrease the cycle time and to avoid the delay in the conventional MAC.

ACKNOWLEDGEMENT

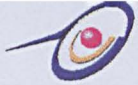
First and foremost 'Alhamdulillah' to Allah, my supervisor Encik Mohd. Yamani Idna Idris for the guidance, advices, tolerant and consideration given in making this project a success. I also would like to thank you my moderator, Encik Zaidi Razak for his comments and evaluation on this project. Not forget my colleagues, Rabiatul Adawiyah Jamil and Norfadilah Khalil for the endurance, tolerant and time spent in making this project. In the way morale support from family deeply appreciated, roommates (for laugh and jokes) and all my friends. In the end, all of this cannot be true if not all the support that I get in making this project success. Last but not least, I want to thank you all members of Faculty of Computer Science and Information Technology for all the information given to complete this course.

TABLE OF CONTENTS



2.0 Literature Review

3.0 Methodology



LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

CHAPTER 1 INTRODUCTION

1.1 Introduction

Multiplier Accumulator also known as MAC is design for a stream of complex numbers. The MAC Unit can be broken up into two distinct units: a data storage path and a data processing path. The data storage path consists of a set of registers that hold data values. The data processing path consists of multiplier unit and adder unit, which perform data multiplication and accumulation sequentially. The control signals provided by the i-unit (instruction unit) designate what process the MAC will perform to get the desired data in one clock cycle.

In the MAC design, both the multiplier and adder are complete custom designs. The multiplier uses the combination of a Booth bit pair encoding algorithm, a sign extension technique, and a carry look-ahead adder with two levels of look-ahead. The Booth encoding algorithm is a technique that will reduce the number of partial products generated. Using the booth-encoding algorithm, fewer partial products will have to be added and therefore the overall speed of the multiplication will be faster.

A complex MAC operates on two sequences of complex numbers, $\{x_i\}$ and $\{y_i\}$. The MAC multiples corresponding elements of the sequences and accumulates the sum of the products. The result is

$$\sum_{i=1} x_i y_i$$

Where N is the length of the sequences. Each complex number is represented in Cartesian form, consisting of a real and an imaginary part. If we are given two complex numbers x and y, their product is a complex number p, calculated as follows :

$$p\_real = x\_real \times y\_real - x\_imag \times y\_imag$$

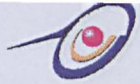
$$p\_imag = x\_real \times y\_imag + x\_imag \times y\_imag$$

The sum of x and y is a complex number s calculated as follows :

$$s\_real = x\_real + y\_real$$

$$s\_imag = x\_imag + y\_imag$$

MAC calculates its results by taking successive pairs of complex numbers, one each from the two input sequences, forming their complex product and adding it to an accumulator register. The accumulator is initially cleared to zero and is reset after each pair of sequences has been processed.



If we count the operations required for each pair of input numbers, the MAC must perform four multiplications to form partial products, then a subtraction and an addition to form the full product and finally two additions to accumulate the result. Since the operations must be performed in this order, the time taken to complete processing one pair of inputs is the sums of the delays for the three steps.

In a high-performance digital signal processing application, this delay may cause the bandwidth of the system to be reduced below a required minimum. Because of that, the Pipelined Multiplier Accumulator is design by pipelining the MAC to avoid the delay at non-pipelined MAC. The pipelining allows the overlapped of the task in multiplier operations.

1.2 Problems Definition

The problem that occurs in this project is time delay. Before pipeline is used, to finish a complete instruction in MAC takes 3 clock cycles. Therefore, CPU must wait for 3 clock cycles to run the complete instruction before it will be used in additions operation in accumulator. The delay can be avoided by pipelining the MAC.

The other problem is in the multiplier part, pipelining reduces cycle time but does not reduce the time required for multiplication. The design of multiplier is important since it determines the overall performance (in term of speed) of the whole system. In adder part, the problem is overflow case. It occurs when the sum of the product is out of the range. Possibility of overflow is high and the system does not work correctly when it happen.

1.3 Scope

The researches will mainly concentrate on the problem that occurs in pipelined part besides the problem in multiplier part and accumulator part. In case to avoid delay, the pipeline method is used in the multiplier accumulator. But, pipelining only reduces cycle time but does not reduce the total time required for multiplication. So, another method is needed as a solution for this problem. For overflow condition, instead of the possibility of overflow is low, one solution is needed to make sure the overflow cannot affects the system. This project will be finished by the simulation part only because there is no implementation part. The topics that covered for every chapter is describe below:

Chapter 1 of Pipelined Multiplier Accumulator project is about the introduction of this project. This chapter will be discussing about problems definition in this project, objective of the Pipelined Multiplier Accumulator, scope of project and works plan and scheduling.

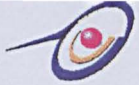
Chapter 2 is about the literatures review. This chapter will de discussing about problem researches that has been occurred before this project will be implement. The discussion will include the researches and analysis for the methods and technique that will be used in this project. There is a comparison between some methods. The most important thing in pipeline MAC is pipeline method.

Chapter 3 will be discussing about the methodology that will be used in this project. Some of the method that will be discuss in this chapter is VHDL, FPGA, CPLD and ASIC. All the method will be discuss but only the best method will be used for this project.

Chapter 4 will be discussing about system analysis and design. The functional, hardware and software requirement will also be discussed in this chapter. Besides that, in this chapter, combinational of the Pipelined Multiplier Accumulator will also be represented.

1.4 Objective

The main goal of this project is to reduce time delays in MAC. In a high-performance digital signal processing application, this delay may cause the bandwidth of the system to be reduced below a required minimum. Because of that, the Pipelined Multiplier Accumulator is design by pipelining the MAC to avoid the delay at non-pipelined MAC.



In order to operate the system under a fast, continuous stream of data, the concept of pipelining these multiplication and addition process is implemented in the design of the system units. Pipelining reduces cycle time but doesn't reduce the total time required for multiplication. One way to speed up multiplication is **Booth Algorithm**, which performs several steps of the multiplication at once. [Wayne, 98] Booth Algorithm will be discussed later in next chapter (Chapter 3: Literature Reviews).

As a conclusion, the goals of Pipeline Multiplier Accumulator design is:

- Avoid the delay
- Speed up multiplication
- Easy for stream of complex numbers

1.5 Constraints

Some limitation can be expected in designing this pipelined multiplier accumulator, because some standard need to be familiarize so it can be implemented. Pipeline in MAC only reduces the cycle time but do not reduce the total time required for multiplication. It means, besides pipelining the MAC another algorithm is needed to speed up the multiplication. Beside that, the needs for development tools are limited. Tools such as VHDL simulator and test or demo board to download the design are hard to obtain. Hence, due to these limitations the some of the feature maybe can't be implemented fully.

1.6 Works Plan and Scheduling

The works will be based on the planning and timetable that has been created earlier. This important because each work must be done in time and careful planning will make it possible.

TOPIC	JUNE	JULY	AUGUST	SEPTEMBER	OCTOBER	NOVEMBER	DECEMBER	JANUARY
Research								
Analysis								
Design								
Simulation								
Report								

Table 1.1 Work plan and scheduling for Pipelined MAC

Chapter 2

LITERATURE REVIEW

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction

This chapter will be discussing about the technique and algorithm that used in this project and why it is chosen. There are three parts, which is literature review in pipeline, multiplier and accumulator. In pipeline, it will discuss about the basic concept of pipeline, why pipelining is needed in multiplier accumulator, pipeline operation and pipeline concept in system's arithmetic operations. In multiplier, there will be a comparison between some of the method. But Booth Algorithm will be used with the purpose to speed up multiplication rather than other method. Besides that, in adder part, a few method of adder will also be discussed. But the best adder will be used in this project is Carry Look-Ahead Adder (CLA).

2.2 Pipeline

Pipelined Multiplier Accumulator will be designed to avoid the delay in conventional MAC. This method is based on the concept of the pipelining, which is can make the cycle time become faster. Discussion below will be discussed about the concept of pipelining and how it will reduces the cycle time.

2.2.1 Pipeline Overview

Pipeline is an implementation technique in which multiple instructions are overlapped in execution. Today, pipelining is the key to make fast processors. The pipeline approach will take much less time. Pipelining is a logic design technique that adds ranks of memory elements to reduce clock cycle time at the cost of added latency. Pipelining is organizational approach is quite common used to reduces cycle time but doesn't reduce the total time required for multiplication. That's why pipeline is suitable to use in Multiplier Accumulator (MAC).

2.2.2 Operations of Pipeline

Most of the complex arithmetic functions encountered in computation can, in principle, be implemented by pipelining. Basic operation on fixed point and floating point numbers can be efficiently partitioned into sub-operations suitable for pipelining.

Pipelining is a method, which can be used to increase the speed of operation of the control processor on arithmetic function operations circuitry. They are often applied to the internal design of high speed computers, including advanced microprocessors as a type of multiprocessing.

Pipelining is a technique in which a task or operation is divided into a number of subtasks that are perform in sequence. Its own logic unit performs each subtask, rather than by a single unit, which performs subtasks. The units are connected together in a serial fashion with the output of the connecting to the input of the next and all the units operate simultaneously. While one unit is performing a subtask of the *i*th task, the proceeding unit in the chain is performing a different subtask on the (*i*+1)th task. [Barry, 91]

In pipelining, a task is presented to the first unit. Once the first subtask of this task is completed, the results are presented to the second unit and another task can be presented to the first unit. Results from one subtask are passed to the next unit as required and a task is completed when all the units have processed the subtasks.

Suppose each unit in the pipeline has the same operating time to complete a subtask and that the first task is completed and a series of task is presented. The time to perform one complete task is same as the time for one unit to perform one subtask of the task, rather than summation of all the unit times. Ideally, each subtask should take the same time, but if this is not the case, the overall processing time will be that of the slowest unit with the faster units being delayed. It may be advantages to equalize stage-operating times with the insertion of extra delays.

2.2.3 Pipelined Concept In System's Arithmetic Operations

During the design of the arithmetic circuitry, speed improvement is considered, as a continuous data stream is processed. The technique of pipeline process is recommended due to the increase speed achieved, compared with other conventional methods.

In conventional arithmetic design, most increase in the number of tasks that could be executed in a unit time interval by arithmetic processor have been achieved by reducing the length of time required to perform a single task, using faster logic circuitry. For instance, faster adders are designed to allow simultaneous addition of many numbers.

However, the circuit technology has almost reached its ultimate limit of light speed. Furthermore, when dueling with large processing works, it could not provide significant increases in computing speed. This problem of speed improvement could be solved through allowing simultaneous execution of many tasks by multiple arithmetic units, which is referred as pipelining operation.

Pipelined approach is a type of architectural design, which significantly increases the number of task that can be executed in a unit time interval, with only a moderate increase in hard ware investment compared with conventional design.

Pipelining arithmetic operation refers to the subdivisions of the total computation workload into individual tasks, so that they can be executed in an overlapped fashion by each own logical unit or segment rather than by a single unit, which performs the whole workload. This overlapped executions are often used in central processor design, in which the fetch, decode, effective address calculation and operand fetch of the next instruction can be overlapped with execution of the current instruction. In this case, when the instruction overhead and the execution times are nearly equal, the overlapped processor will be twice as fast as the conventional design.

A pipelined arithmetic operating unit is defined as a collection of series of hardware resources (units of task), which are arranged as a pipeline with synchronized timing control (for synchronous pipeline data transfer), such that a flow of subdivided tasks can be simultaneously executed by the successive units of the pipeline, as illustrated in Figure 2.1.

In this pipelined arithmetic operation, each operational unit is a special purpose combinational arithmetic logic circuitry with delay T, such as an adder, a multiplier etc. data latches are used as synchronized registers, in order to hold the input and output data of successive units. Each of these latches will latch its data to the next unit when triggered by its external clock signal. In the normal cases, two additional latches are

added to the two end units, which handle the inputs and outputs of the entire pipelined system.

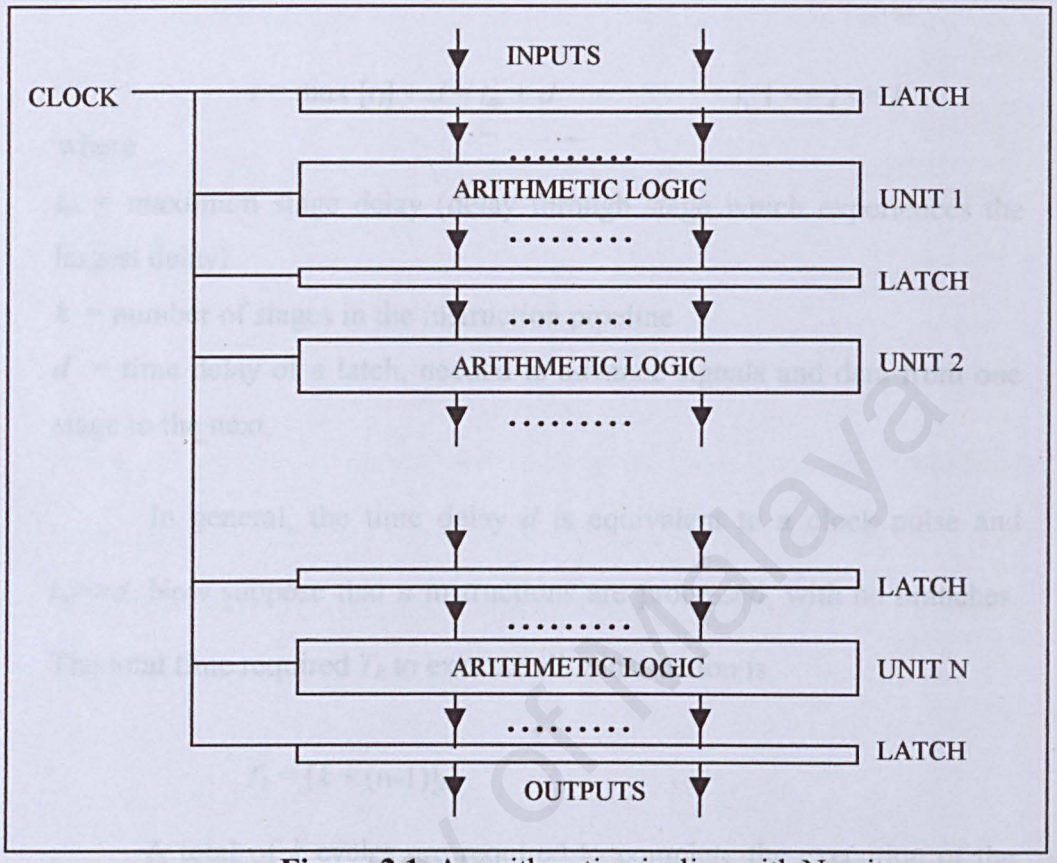


Figure 2.1 An arithmetic pipeline with N units

2.2.4 Advantages of Pipelined

There are many advantages of pipeline that make it suitable to use in MAC to reduces the latency and time delays problem. The most important advantage of pipeline is it increasing the speed of the system. It makes time to finished the clock cycle become more faster than not the time to finished without it. The cycle time T of an instruction pipeline is

the time needed to advance a set of instruction one stage through the pipeline. The cycle time can be determined as

$$t = \max [t_i] + d = t_m + d \qquad i, 1 <= I <= k$$

where

t_m = maximum stage delay (delay through stage which experiences the largest delay)

k = number of stages in the instruction pipeline

d = time delay of a latch, needed to advance signals and data from one stage to the next.

In general, the time delay d is equivalent to a clock pulse and $t_m >> d$. Now suppose that n instructions are processed, with no branches. The total time required T_k to execute all n instruction is

$$T_k = [k + (n-1)]_t$$

A total of k cycles are required to complete the execution of the first instruction and the remaining n-1 cycles.

The speedup factor for the instruction pipeline compared to execution without pipeline is defined as

$$S_k = T_1/T_k = nkt / [k + (n-1)]t = nk / k + (n-1)$$

Besides increasing the speed of system, in some cases, the pipelining technique has the advantage of requiring less logic than a non-pipelined system. Obviously, it could be seen that, the rate of the pipelined system depends on the unit with maximum delay time. [William, 96]

2.2.5 Conventional MAC

In conventional MAC, a complex MAC operates on two sequences of complex numbers, $\{x_i\}$ and $\{y_i\}$. The MAC multiples corresponding elements of the sequences and accumulates the sum of the products. The result is

$$\sum_{i=1}^{N} x_i y_i$$

where N is the length of the sequences. Each complex number is represented in Cartesian form, consisting of a real and an imaginary part. If we are given two complex numbers x and y, their product is a complex number p, calculated as follows :

$$p\_real = x\_real \times y\_real - x\_imag \times y\_imag$$
$$p\_imag = x\_real \times y\_imag + x\_imag \times y\_imag$$

The sum of x and y is a complex number s calculated as follows :

$$s\_real = x\_real + y\_real$$
$$s\_imag = x\_imag + y\_imag$$

MAC calculates its results by taking successive pairs of complex numbers, one each from the two input sequences, forming their complex product and adding it to an accumulator register. The accumulator is initially cleared to zero and is reset after each pair of sequences has been processed.

To count the operations required for each pair of input numbers, the MAC must perform four multiplications to form partial products, then a subtraction and an addition to form the full product and finally two additions to accumulate the result, this is shown in Figure 2.2. Since the operations must be performed in this order, the time taken to complete processing one pair of inputs is the sums of the delays for the three steps.

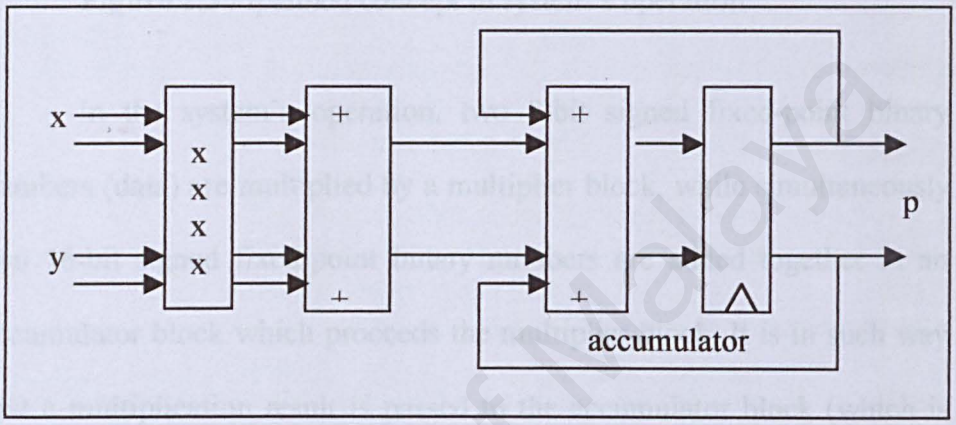


Figure 2.2 Dataflow diagrams showing order of operations by the conventional MAC

2.2.6 Why Pipelined MAC?

The previous research done by many researches have pointed out the issue to create a MAC with higher execution speed and decrease cycle time. Therefore, the best solution is used pipeline technique in multiplication unit. MAC is pipelining to avoid the delay in the process. This design of pipelined multiplier accumulator (MAC) is for a stream of complex number.

In order to operate the system under a fast, continuous stream of data, the concept pipelining these multiplication and addition process is

implemented in the design of the system units, as shown in a block diagram below in Figure 2.3.

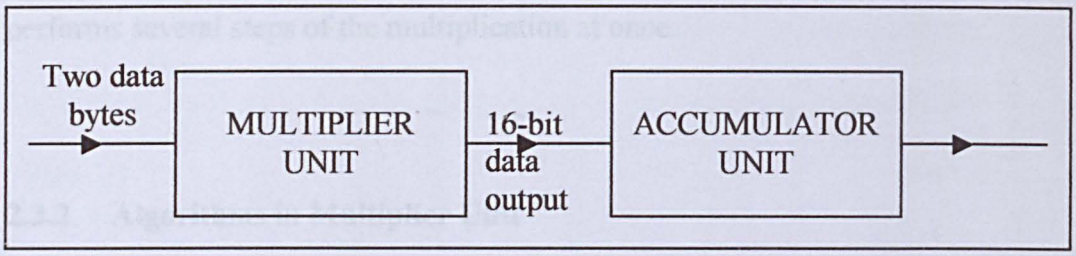


Figure 2.3 Pipelined concept in system's operation

In the system's operation, two 8-bit signed fixed-point binary numbers (data) are multiplied by a multiplier block, while simultaneously two 16-bit signed fixed-point binary numbers are added together in an accumulator block which proceeds the multiplier block. It is in such way that a multiplication result is passed to the accumulator block (which is feedback to the block for next additional operation); in order to obtain the next output result.

2.3 Multiplier

2.3.1 Multiplier Overview

Multiplier is one part of Pipelined Multiplier Accumulator. Multiplier design starts with the elementary school algorithm for multiplication. The computation of partial products and their accumulation into the complete product can be optimized in many ways, but an understanding of the basic steps in multiplication is important to a full

appreciation of those improvement. One approach to speed up the operation of speed up multiplication is **Booth Algorithm** [Boo, 51], which performs several steps of the multiplication at once.

2.3.2 Algorithms in Multiplier Unit

Approach	Characteristic
Bit-Pair Recording	-Multiplication speed up technique that guarantees that an n-bit multiplier will generate at most n/2 summands and will uniformly handle the signed- operand case (Cavanagh, 84). -The total number of clock pulses needed is small. -This technique can multiply the products two times than the Booth Algorithm. -In this technique, three data bits are checked in each time.
Add-Shift Sequential Multiplication	-Only required positive multiplier. If negative multiplier is used, both multiplicands had to be in 2's complement before the multiplication is performed. -Method becomes complex when perform negative multiplier because it need to change the binary representation to 2's complement. -Inflexible due to the need to exchange the circuit to execute certain task.
2's Complement Modular Arrays Multiplication Technique	-Using general multiply algorithm with special feature where the whole operation is modularized into section. -Used Carry-Save and Carry Look-Ahead adders approach to sum up these module outputs to get the final product. - This technique leads to delay where it need more time to perform the local multiplication; speed of overall operation is increased.
Booth Algorithm	-Generates 2n-bit product with n-bit input. -Direct algorithm. Treats positive and negative numbers equally (in the same manner).

Table 2.1 Comparison between algorithms in multiplier unit

In Pipelined MAC, the speed of multiplier in performing its task is essentially important. Therefore, a fast multiplication technique is needed. In multiplication, four algorithms have been considered to implement in the design. Each of every algorithm has own characteristic as shown in Table 2.1.

2.3.3 Booth Algorithm

The Booth Algorithm is chosen for its speed up operations and simplicity. This algorithm is believed can achieve certain goals that have been highlighted earlier. Booth's algorithm takes advantage of the fact that an adder-subtractor is nearly as fast and small as a simple adder. It treats the negative and positive number uniformly. With this technique, system can multiply operand to get the partial products more quickly with the decoding method. With streams of bit 0's instead of the alternate streams of 0's and 1's, doing multiplication is not a nightmare anymore.

The implementation of this approach in hardware is the most consideration for choosing this method. The simplicity and ease understanding the hardware suits the design purposes. The components can be used more than once and this approach can save the cost in developing the system.

2.4 Accumulator

2.4.1 Accumulator Overview

As for the design requirement where synchronous type of pipeline data transfer is used, only synchronous adders are considered in the construction of the Accumulator unit. In order to select the most suitable adders which meet the requirements, a good understanding of the carry speed-up techniques used in these adders is essential. Therefore, a thorough study on these adders's hardware organization is needed.

2.4.2 Approach in Accumulator Unit

Approach	Characteristic
Ripple-Carry Adder	- Get the name due to the result of an addition of two bits depends on the carry generated by the addition of the previous two bits. - Has considerably low speed due to large propagation delay in its operation. - It limits the frequency rate of data stream to be processed, although its implementation is rather simpler than other type of adders being discussed later.
Carry-Select Adder	- Using the carry in assumption technique which can increase the speed of operation. - This technique is implemented for each of partition of four bit groups or section adders which consist of the same design.
Conditional Sum Adder	- Using the carry in assumption technique but for individual bit.
Carry Look-Ahead Adder	- Solves the slow speed problem by calculating the carry signals in advance, based on the input signals.

Table 2.2 Comparison between adders in Accumulator

2.4.3 Carry Look-ahead Adder

Based on the comparison above, Carry Look-ahead Adder is the best choice to use in Accumulator unit. Using Carry Look-ahead Adder will solve the slow speed problem that occurs when many bits need to add. Carry Look Ahead solves this problem by calculating the carry signals in advance, based on the input signals. It is based on the fact that a carry signal will be generated in two cases:

1. when both bits A_i and B_i are 1, or
2. when one of two bits is 1 and the carry-in (carry of the previous stage) is 1.

The Carry Look-ahead Adder can be broken up in two modules:

1. the Partial Full Adder, PFA, which generates S_i, P_i and G_i as defined by equations below:

$$G_i = A_i . B_i$$
$$P_i = (A_i \$ B_i)$$
$$S_i = A_i \$ B_i \$ C_i = P_i \$ C_i$$

2. the Carry Look ahead Logic, which generates the carry-out bits according to equations below:

$$C_1 = G_0 + P_0 C_0$$
$$C_4 = G_3 + P_3 G_2 + P_3.P_2.G_1 + P_3.P_2.P_1.G_0 + P_3.P_2.P_1.P_0.G_0$$

The 4-bit adder can then be built by using 4 PFAs and the Carry Look-ahead Logic.

2.5 Conclusion

From all the topics that discuss above, that's why pipelined multiplier accumulator (MAC) is designed. It supposes to make the operation in the multiplier accumulator (MAC) faster and avoid the delay and latency in that operation. Pipelining will increase the system speed up by allow the overlapped of the tasks. Pipelined MAC makes the operation of conventional MAC become more faster. This is because the pipelined MAC has pipeline register that will store the input temporarily before it is used in summation, while the system will fetch the next input. As mentioned earlier, with pipeline, three steps delay have been reduced. This will make the process more faster because the system didn't need to wait for the first input to finished it summation before the second input will entered.

The another approach to boost or speed up the operation of Pipelined MAC is Booth Algorithm and Carry Look-Ahead Adder make the operation of Pipeline MAC become more faster than the conventional one. Besides that, the pipelined MAC also has the overflow status signal to control or reduce any possibilities of the overflow to happen in summation.

The Multiplier unit mainly determines the overall performance of the system. The multiplier design is much emphasized in order to select a suitable method for its construction. Although Bit-Pair Recoding is faster than Booth Algorithm technique, the complexity of its design is very much higher than the ladder. With large data processing, the overall operation's performance of Booth

Algorithm is considerably high compared with Bit-Pair Recoding. Furthermore, the improvement in speed of the Multiplier unit could be achieved.

As a conclusion, the effectiveness of pipelining concept in the system's implementation is undeniable as the performance of the system is improved where the multiplication process is pipelined with the addition process. The pipelined MAC has been increased the conventional execution's speed and decrease the cycle time but doesn't reduce the total time required for multiplication.

The next chapter will cover about the methodology that will be used in designing a Pipelined multiplier Accumulator. In the chapter, methods of design that will be used in this project will be discussed. Hardware description language (VHDL) that will be used in simulation and testing will also be discussed later, in chapter 3. Furthermore, the chapter also will be discuss about ASIC and programmable devices, which is CPLD and FPGA and the comparison between these two devices will also be presented.

Chapter 3

METHODOLOGY

CHAPTER 3 METHODOLOGY

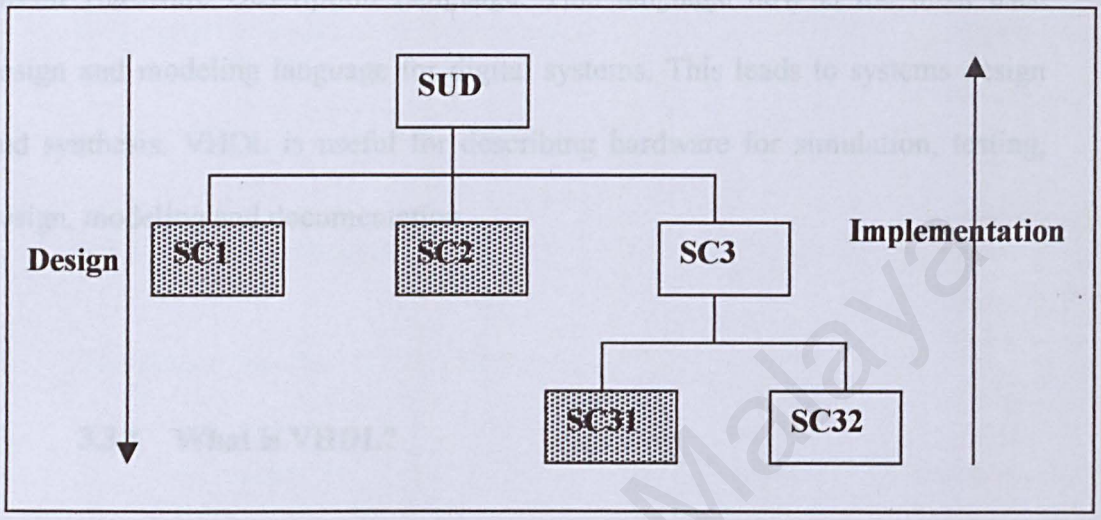
3.1 INTRODUCTION

This chapter will be discussing about the methodology that will be used in designing a Pipelined Multiplier Accumulator. There are two methods of design used in this project, which are top-down design and bottom-up design. Hardware description language (VHDL) that will be used in simulation and testing will also be discussed in this chapter. Besides that, this chapter also will discuss about ASIC and programmable devices, which is CPLD and FPGA and the comparison between these two devices will also be presented.

3.2 Method of Design

There are two design method will be used in this project, top-down design and bottom-up design. Top-down design technique is recursively partitions a system into its sub-component until all sub-components become manageable design parts. Design became manageable when the component is available as part of the library. It can be implemented by modifying an already available part.

Mapping to hardware depends on target the technology, available libraries and tools. Generally, a system can be further partitioned into its simpler components. Figure 3.1 shows the implementation of top-down design and bottom-up design.



SUD – system under design
SSC – system sub-component
Shaded areas designate sub-component with hardware implementation

Figure 3.1 Top-down design and bottom-up implementation

3.3 VHDL

In this project, all the design will be using a hardware description language or better known as VHDL. VHDL stand for Very High-speed integrated circuit Hardware Description Language. This language now is the most used design and modeling language for digital systems. This leads to systems design and synthesis. VHDL is useful for describing hardware for simulation, testing, design, modeling and documentation.

3.3.1 What is VHDL?

VHDL it is language that can be used for modeling a digital system at many level of abstraction, from algorithm level to the gate level. The complexity could vary from simple gate to a complete digital electronic system, or anything in between. VHDL always regarded as integrated amalgamation of some language, which is:

- Sequential language
- Concurrent language
- Net-list language
- Time specification
- Waveform generation language

The language has feature or constructs that enables users to express the concurrent or sequential behavior of a digital system as an interconnection of components. By using this constructs, test waveforms can be generated. In the end, the constructs can provide a comprehensive description of system in a single model.

3.3.2 The Advantages

The advantages that VHDL offers are :

- ❖ Portability
  - Because of the code used can be simulated and used in many design tools and in different stages, it reduces dependency for the set of tools whose limited in capability. The VHDL standard also transform design data much easier than a design database of a proprietary design tool.
- ❖ Modeling capability
  - It is developed to model all level of designs, from electronic boxes to transistors. It can accommodate behavioral constructs and mathematical routines that describe complex models. It allows use of multiple architectures and associates with the same design during various stages of the design process.

- Reusability
  - Design can be describes, verified and modified for future use. This eliminates reading and marking changes to schematic pages that are time consuming beside subject to error.
- Documentation
  - VHDL is a description language, which allows documentation to be located in single place by embedding it in the code. The combining of comments and the code actually dictates what the design should do reduces the ambiguity between specification and implementation.
- New design methodology
  - Using VHDL and synthesis creates a new methodology that increases the design productivity, shortens the design cycle and lower costs.
- Technology and foundry independence
  - The functionality and behavior of the design can be described with VHDL and verified, making it foundry and technology independent. This frees the designer to proceed without having to wait for the foundry and technology to be selected.

3.3.3 New Design Methodology

The introduction of VHDL and synthesis enables the design community to explore a new design methodology. With the traditional approach, starts with schematics drawing and then performs functional and timing simulation based on the same schematics. If occur errors it back to update the schematics again. After the layout, function and back-annotated timing are verified again with the same schematics. With VHDL, the design is functionally describe.

3.3.4 Hardware Abstraction

VHDL is used to describe a model for a digital hardware device, which specifies the external view of the device and one or more internal views. The internal view specifies the functionality or structure, while the external view specifies the interface of the device through which it communicates with the other model in its environment.

The device-to-device model mapping is strictly one-to-many. For example, a device modeled at high level of abstraction may not have been used in the description. Also, the data transfer at the interface may be treated in terms of integer values, instead of logical values. In VHDL, each device model is treated as a distinct representation of a unique device, called an *entity* in this text.

3.3.5 Basic Concept

Since VHDL is a hardware description language, it has features, which are conceptually different than other languages. These represents special characteristic of hardware components and carries. These are:

- Timing

Timing is associated with the values that are assigned to the hardware carriers. Signal represent real wires, where the delays of the transfer through wire are concern, thus assignment to signal in VHDL, involve timing.

- Concurrency

The terms refer to the simultaneous operation of various components. The VHDL has constructs that allow a virtually concurrent environment to be created. These constructs satisfy concurrency required for the description of the hardware. Through the use of concurrent constructs, timing of the interconnecting signals and order of the simulation construct or components, a VHDL simulator makes us think that the execution is being done concurrently.

3.3.6 VHDL vs. Verilog

The table below is shown about the comparison between VHDL and Verilog.

	VHDL	Verilog
Data Types	A multitude of language or user defined data types can be used.	Very simple, easy to use and very much geared towards modeling hardware structure as opposed to abstract hardware modeling
Design reusability	Procedures and functions may be placed in a package so that they are avail able to any design-unit that wishes to use them.	There is no concept of packages. Functions and procedures used within a model must be defined in the module.
Managing large designs	Configuration, generate, generic and package statements all help manage large design structures.	There are no statements that help manage large designs.
Procedures and tasks	Allows concurrent procedure calls	Does not allow concurrent task calls.
Libraries	A library is a store for compiled entities, architectures, packages and configurations. Useful for managing multiple design projects.	There is no concept of a library. This is due to it's origins as an interpretive language

Table 3.1 Comparison between VHDL and Verilog

From all the advantages and specification of VHDL that discuss above, VHDL is choosing to be used in this project. From the comparison above, it shows that VHDL is better than Verilog to use in this project.

3.4 ASIC, CPLD and FPGA

Application Specific Integrated Circuit, or ASIC, is a chip that can be designed by an engineer with no particular knowledge of semiconductor physics or semiconductor processes. Ideally, the hardware designer wanted something that gave the flexibility and complexity of and ASIC but with the shorter turn-around time of a programmable device. The solution came in the form of two new devices - the Complex Programmable Logic Device (CPLD) and Field Programmable Gate Array (FPGA). CPLD are as fast as Programmable Array Logic (PAL) but more complex. FPGA approach the complexity of Gate Arrays but are still programmable.

3.4.1 ASIC

The Application Specific Integrated Circuit (ASIC) vendor has created a library of cells and functions that the designer can use without needing to know precisely how these functions are implemented in silicon. The vendor then lays out the chip, creates the masks and manufactures the Asics.

The gate array is an ASIC with a particular architecture that consists of rows and columns of regular transistor structures. Each basic cells or gate consists of the same small number of transistors, which are not connected. In fact, none of the transistors on the gate array are initially

connected at all. The reason for this is that the connection is determined completely by the design that will implement.

3.4.2 CPLD

Complex Programmable Logic Device (CPLD) is exactly what they claim to be. Essentially CPLD are designed to appear just like a large number of Pals in a single chip, connected to each other through a cross point switch. The CPLD use the same development tools and programmers and based on the same technologies but they can handle much more complex logic and more of it.

3.4.2.1 CPLD Architecture

The diagram in Figure 3.1 shows the internal architecture of a typical CPLD. While each manufacturer has a different variation, in general they are similar in that they consists of function blocks, input/output block and an interconnect matrix.

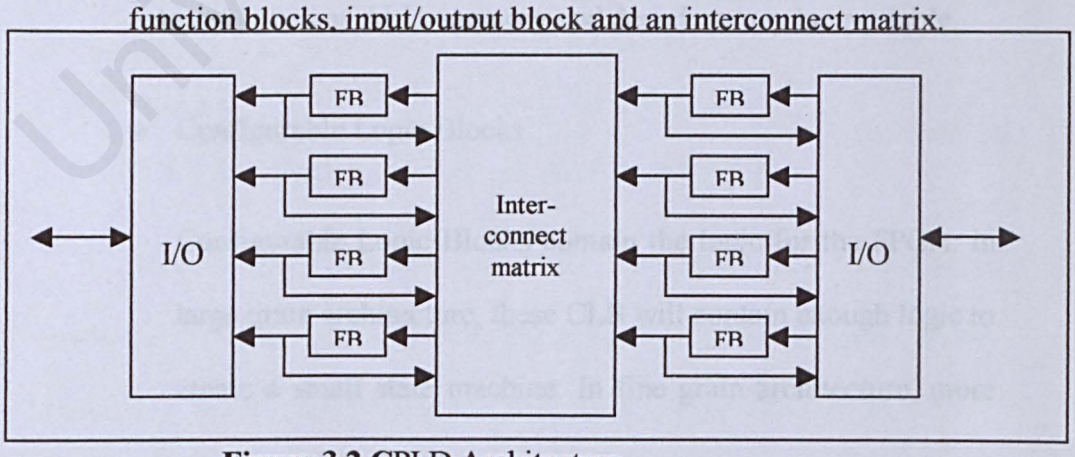


Figure 3.2 CPLD Architecture

3.4.3 FPGA

Field Programmable Gate Array (FPGA) are called this because rather than having a structure similar to a PAL or other programmable device, they are structured very much like a gate array ASIC. This makes FPGA very nice for use in prototyping ASIC or in places where and ASIC will eventually used. For example, an FPGA maybe used in designs that need to get to market quickly regardless of the cost. Later an ASIC can be used in place of the FPGA when the production volume increases, in order to reduce cost.

3.4.3.1 FPGA Architectures

Each FPGA vendor has its own FPGA architecture, but in general terms they are all a variation. The architecture consists of configurable logic blocks, configurable I/O blocks and programmable interconnect. Also, there will be clock circuitry for driving the clock signals to each logic block and additional logic resources such as ALU, memory and decoders may be available.

- Configurable Logic Blocks

Configurable Logic Blocks contain the logic for the FPGA. In large grain architecture, these CLB will contain enough logic to create a small state machine. In fine grain architecture, more

like a true gate array ASIC, the CLB will contain only very basic logic.

- Configurable I/O Blocks

A Configurable I/O Block is used to bring signals onto the chip and send them back off again. It consists of an input buffer and output buffer with three state and open collector output controls.

3.4.4 CPLD vs. FPGA

The table below is shown the comparison between FPGA and CPLD.

CPLD	FPGA
Complex Programmable Logic Device	Field Programmable Gate Array
PALs	Gate Arrays
Short lead rime	High density
Programmable	Can implement many logic functions
No NRE charges	Relatively fast

Table 3.2 Comparison between CPLD and FPGA

From all the specification of ASIC, CPLD and FPGA that discuss above, it shows that there are many advantages of FPGA compared to the others. That is why the FPGA is choosing for implement this project.

3.5 Conclusion

The discussion above is including the methodology that will be used in designing a Pipelined Multiplier Accumulator. There are two methods of design will be used in this project, which are top-down design and bottom-up design. Top-down design will be used in design part, meanwhile the bottom-up design will be used in implementation part. This chapter also discuss about the hardware description language that will be used in simulation and testing. VHDL is choosing considered on the comparison that had been discussed above. ASIC and programmable devices, which is CPLD and FPGA and the comparison of this two devices also discussed in this chapter. From all the fact above, FPGA is choosing to be used in this project.

The next chapter (Chapter 4) will discuss about Pipelined Multiplier Accumulator Analysis and Design. That chapter will discuss about system analysis and design for a pipelined MAC. The discussion will include discussion about the system overviews, conventional MAC and pipelined MAC. Besides that, the comparison between conventional MAC and pipelined MAC also will discuss in that chapter. That chapter also will discuss about signals controller that used in the pipeline multiplier accumulator and also the top-level design of this system.

Chapter 4

SYSTEM DESIGN

CHAPTER 4 SYSTEM DESIGN

4.1 Introduction

This chapter will be discussing about system analysis and design for a pipelined MAC. That discussion will be included with the pipelined MAC overview and design of the pipelined MAC. Furthermore, this chapter also will be investigated about the design of multiplier, accumulator and pipeline design in pipelined multiplier accumulator. The description of top-level design, black box and system's block diagram will also be presented to show in this chapter.

4.2 System Overview

The Pipelined Multiplier Accumulator, which is implemented, will facilitate the arithmetic operations of multiplying pairs of 8-bit binary number and adding 16-bit binary numbers that is initially the multiplication result obtained.

In order to operate the system under a fast, continuous stream of data, the concept pipelining these multiplication and addition process is implemented in the design of the system unit (Refer Figure 2.3: Pipelined concept in system's operation; Chapter 2: Literature review; pg 19). Each stage of system is pipelined, so that it can perform on a continuous data. A control unit controls each stages of

the pipelined unit. The details of the control unit will be discussed in the following sections. (Section 4.6 : Signal Controller).

In the system's operation, two 8-bit signed fixed-point binary numbers (data) are multiplied by a multiplier block, while simultaneously two 16-bit signed fixed-point binary numbers are added together in an accumulator block which proceeds the multiplier block. It is in such way that a multiplication result is passed to the accumulator block (which is feedback to the block for next additional operation); in order to obtain the next output result.

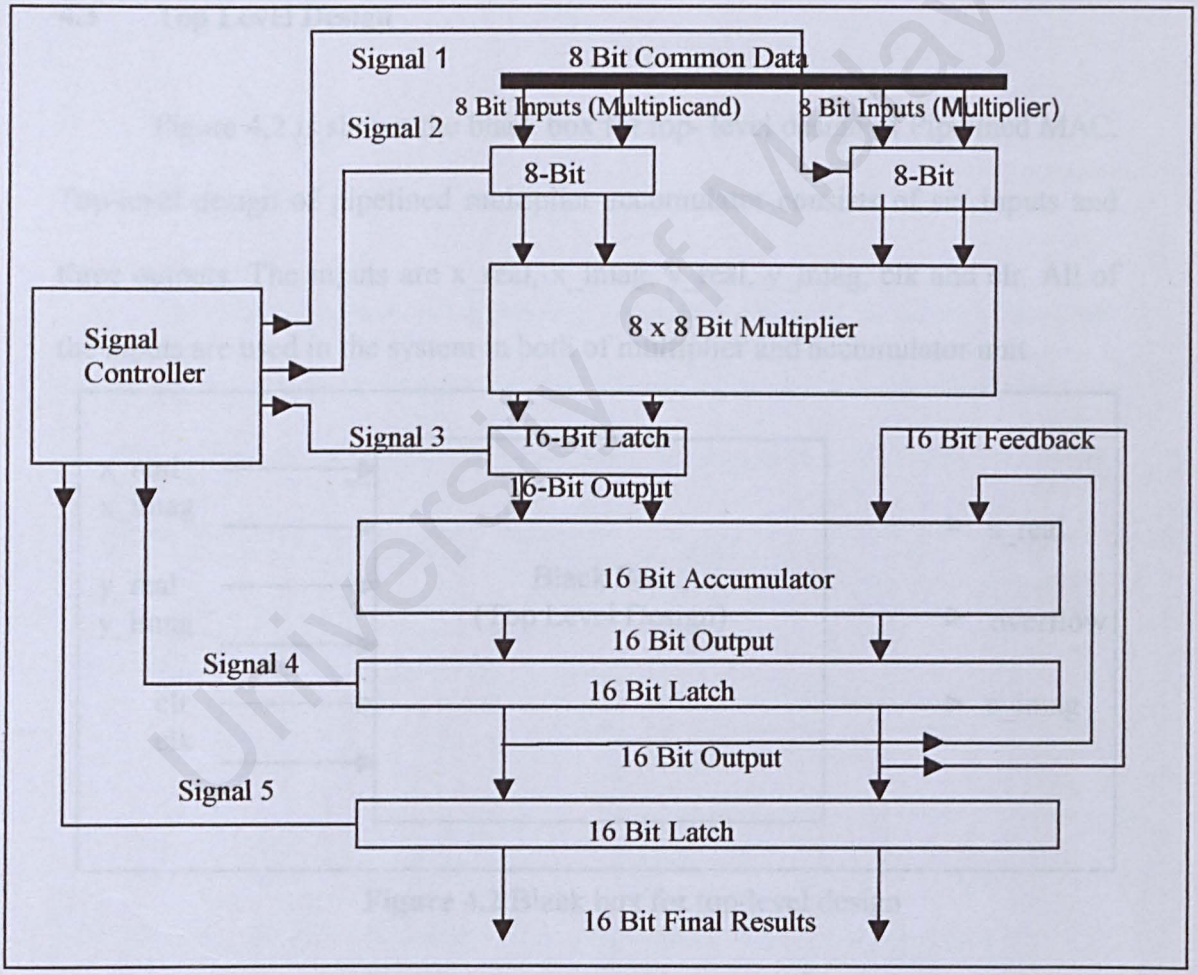


Figure 4.1 System's block diagram

For this design, synchronous transfer of data bit between the multiplier and accumulator blocks is used, where latches are used for latching data bits in and out of those block in order to synchronize the multiplication and addition operations. The clocking rates to these latches are controlled by a signal controller block, from which different rates of clock pulsing are generated and channeled to the respective latches. The system's block diagram is shown in Figure 4.1.

4.3 Top Level Design

Figure 4.2 is shown the black box for top- level design of Pipelined MAC. Top-level design of pipelined multiplier accumulator consists of six inputs and three outputs. The inputs are x_real, x_imag, y_real, y_imag, clk and clr. All of the inputs are used in the system in both of multiplier and accumulator unit.

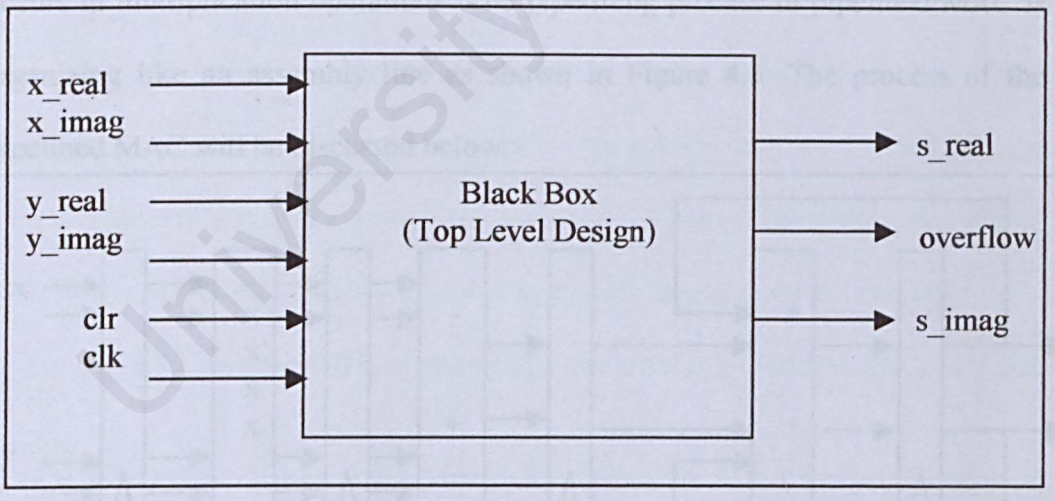


Figure 4.2 Black box for top-level design

The x_real, x_imag, y_real and y_imag inputs will be used in multiplication operation of multiplier part and addition operation in accumulator part in the pipelined MAC. The 'clr' input only use in accumulator part to clear the pipeline register to zero and reset the overflow condition. Meanwhile the 'clk' will be used to control the clock signal.

The outputs are s_real, s_imag and overflow. All of this output except for overflow will appeared as a final sum of all operation in multiplier and accumulator. Overflow happened only in the certain case.

4.4 Pipeline in MAC

The main purpose of designing pipelined MAC is to avoid delay that occurs in multiplication operations. The pipelining process in pipelined MAC is organizing like an assembly line as shown in Figure 4.3. The process of the pipelined MAC will be discussed below.

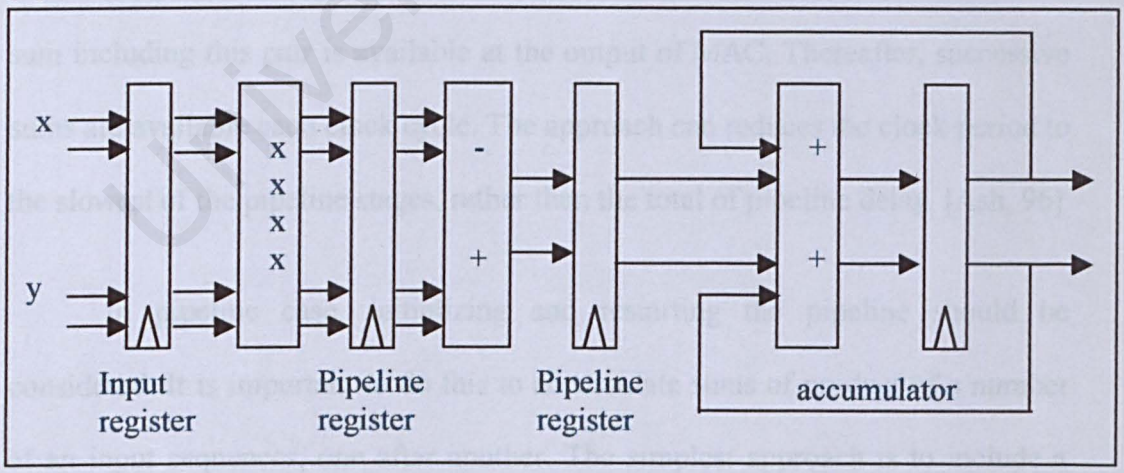


Figure 4.3 Dataflow diagrams showing order of operations by the pipelined MAC

Firstly, the first pair of input numbers is stored in the input register on the first clock edge. During the first clock cycle, the multipliers calculate the partial products, while the system prepares the next pair of inputs. On the second clock edge the partial product are stored in the first pipeline register and the second pair of inputs numbers is entered into the input register. During the second clock cycle, the subtracter and adder produce the full product for the first input pair; the multipliers produce the partial products for the second input pair, while the system prepares the third input pair.

On the third clock edge, all input is stored in the second pipeline register, the first pipeline register and the input register. Then in the third clock cycle, the address accumulate the product of the first pair with the previous sum and the preceding stage operates on the second and third pairs, while the system prepares the fourth pair.

The sum in the accumulator is updated on the fourth clock edge. Thus, three clock cycles after the first input pair was entered into the input latch, the sum including this pair is available at the output of MAC. Thereafter, successive sums are available each clock cycle. The approach can reduces the clock period to the slowest of the pipeline stages, rather than the total of pipeline delay. [Ash, 96]

In pipeline case, initializing and restarting the pipeline should be considered. It is important to do this to accumulate sums of product of a number of an input sequences, one after another. The simplest approach is to include a 'clear' input to the accumulator register that forces its content to zero on the next

clock edge. It's mean, for each pair of sequences to be multiplied and accumulated, the number start entering the input register on successive clock edge. Then, two clock cycles after the first pair input is entered, the clear input is asserts.

This causes the accumulator to reset at the same time as the product of the first pair of numbers reaches the second pipeline register. On the following cycle, this product will be added to the zero value forced into the accumulator. Three clock cycles after the last pair in the input sequence has been entered, the final sums will appears at the output of the MAC. Successive input sequences must separate by at least one idle cycle and reset the accumulator between summations. It is important to reset all values to zero before the next summation will be operates.

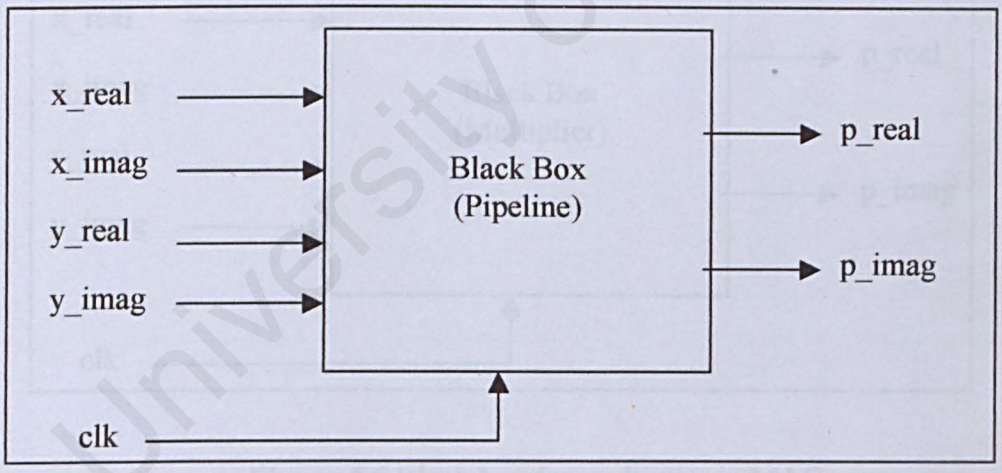


Figure 4.4 Black box for pipelined in MAC

4.5 Multiplier in MAC

To form a partial product, MAC must perform four multiplications. Then, a subtraction and addition is executed to form the full product and finally two additions to accumulate the result. Since the operations must be performed in this order, the time taken to complete processing one pair of inputs is the sum of the delays for the three steps.

This delay may cause the bandwidth of the system to be reduced below a required minimum, in a high-performance digital signal processing application. Figure 4.5 is shown the black box for multiplier in MAC. The multiplier unit is consisting of five inputs and two outputs.

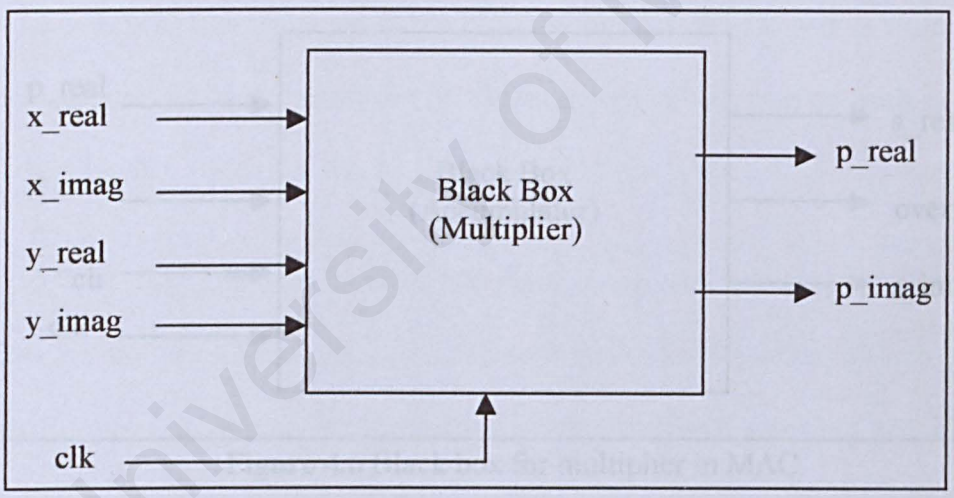


Figure 4.5 Black box for multiplier in MAC

4.6 Accumulator in MAC

MAC calculates its result by taking successive pairs of complex number (that have been discussed in Chapter 2, Section 2.2.5), one of each from the two input sequences, forming their complex product and adding it to an accumulator register. Accumulator is initially cleared to zero and is reset after each pair of sequences has been processed. Refer Figure 2.2: Dataflow diagrams showing order of operations by the conventional MAC in Chapter 2: Literature review (pg 18) to look for dataflow in accumulator unit of Pipelined MAC. Figure 4.6 is shown the black box for accumulator in MAC. The multiplier unit is consisting of four inputs and three outputs.

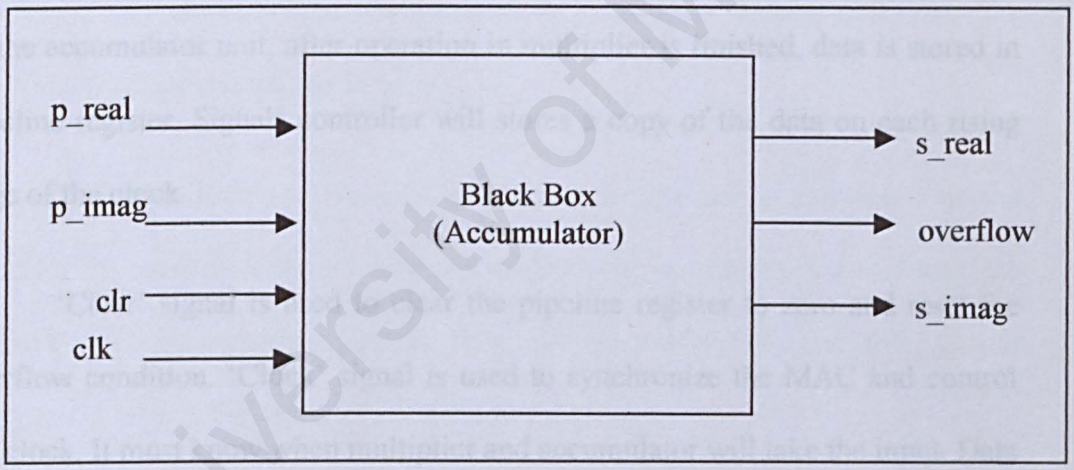


Figure 4.6 Black box for multiplier in MAC

4.7 Signals Controller

It is an essential part of the whole system to generate clock pulse signals, which synchronize the processes in the multiplier and accumulator units, besides providing interrupt signal for the printer port. Signals controller is important to control stages of pipeline and the continuous of input data in pipeline MAC. Signals controller will work in both of multiplier and accumulator units.

In the multiplier unit, signals controller in the system will take the new input value when the clock is '1'. It is because at the time clock is '1'; latch is opened to allow data entered into the system. When the clock is '0', latch is closed and data cannot enter into the system. The process will operate at this time. In the accumulator unit, after operation in multiplier is finished, data is stored in pipeline register. Signals controller will stores a copy of the data on each rising edge of the clock.

'Clear' signal is used to clear the pipeline register to zero and reset the overflow condition. 'Clock' signal is used to synchronize the MAC and control the clock. It must know when multiplier and accumulator will take the input. Data entered when the clock is '1'.

Efficiency of signals controller part is greatly affects the reliability of the overall system. The design has been done carefully through the use of GateSim simulator, in order to obtain correct and stable output control signals for the entire system's operation. The simulator has been as essential supporting tool in

detecting the weak points of the design. The weaknesses are fundamental basis for improvement of the design, although there are differences between the simulation and actual design. Basically, it operates in the same manner as the actual operation. As long as the differences are well defined and understood, the use of simulator is obvious, to be an important reference for the actual work.

4.8 Conclusion

The design and construction of the system is done in the way where simplicity and fast speed criterions are emphasized. Selections of construction method of Accumulator and Multiplier have been done through analyzing few method which are commonly used.

The Multiplier unit mainly determines the overall performance of the system. The multiplier design is much emphasized in order to select a suitable method for its construction. Besides that, efficiency of signals controller part is also greatly affects the reliability of the overall system.

The effectiveness of pipelining concept in the system's implementation is undeniable as the performance of the system is improved where the multiplication process is pipelined with the addition process.

The next chapter (Chapter 5) will discuss about systems implementation. That chapter will discuss about system implementation for a pipelined MAC. The discussion will include discussion about the system implementation, description of **PeakFPGA Designer Suite FPGA Synthesis Edition** and the system coding. Besides that, the behavioral model also will discuss in that chapter.

Chapter 5

SYSTEM IMPLEMENTATION

CHAPTER 5 SYSTEM IMPLEMENTATION

5.1 INTRODUCTION

To bring forward on designing a logic device for Pipelined Multiplier Accumulator, this chapter will take the reader to the description of **PeakFPGA Designer Suite FPGA Synthesis Edition.** These tools will be extensively used throughout the design implementation process. The coding for the system's modules is developed using the VHISC Hardware Description Language (VHDL) programming language (discussed before in Chapter 3.3 page 29). This chapter also will discuss about system development, the description of all pins in top-level design and system coding for Pipelined Multiplier Accumulator.

5.2 PeakFPGA DESIGNER SUITE FPGA SYNTHESIS EDITION

PeakVHDL is an advanced software product intended to help you use VHDL for digital design projects. PeakVHDL includes an integrated VHDL simulator, VHDL source file editor, Hierarchy Browser and other resources for VHDL users. To get started using PeakVHDL, we should load one of the sample projects included in the Examples subdirectory of PeakVHDL installation. The examples provided are intended to demonstrate a variety of useful VHDL

concepts, including various methods of writing test benches. These examples will also help us to understand how to create and manage a PeakVHDL project.

Figure 5.1 : Main Application Window

To load a sample project, select Open Project from the PeakVHDL File menu, and navigate to the Examples subdirectory of the PeakVHDL installation directory. Select one of the sample projects and open the .ACC file associated with that project. When we have opened a sample project, we will see two or more .VHD source files listed in the Hierarchy Browser window. We can double click on any file name listed to open a VHDL source file-editing window.

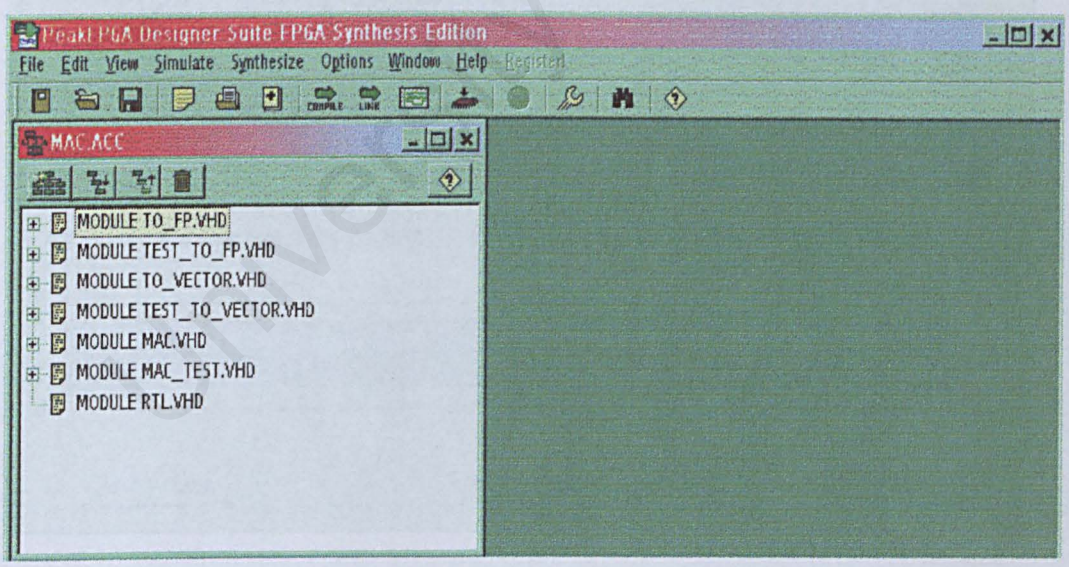


Figure 5.2 : Hierarchy Browser Window

To process the project and start the PeakVHDL simulator (PeakSIM), select the top-most VHDL source file (the test bench) by clicking on it then choose Load Selected from the Simulate menu or click on the Load Selected button from the PeakVHDL toolbar. When we highlight the top-most module and choose Load Selected, the following occurs:

1) All VHDL source file modules in the project are compiled in bottom-up order as determined by the Hierarchy Browser.
2) The compiled source file modules are linked together (elaborated), and a .VX simulation executable is generated.
3) The .VX simulation executable is loaded into the PeakSIM simulation application.

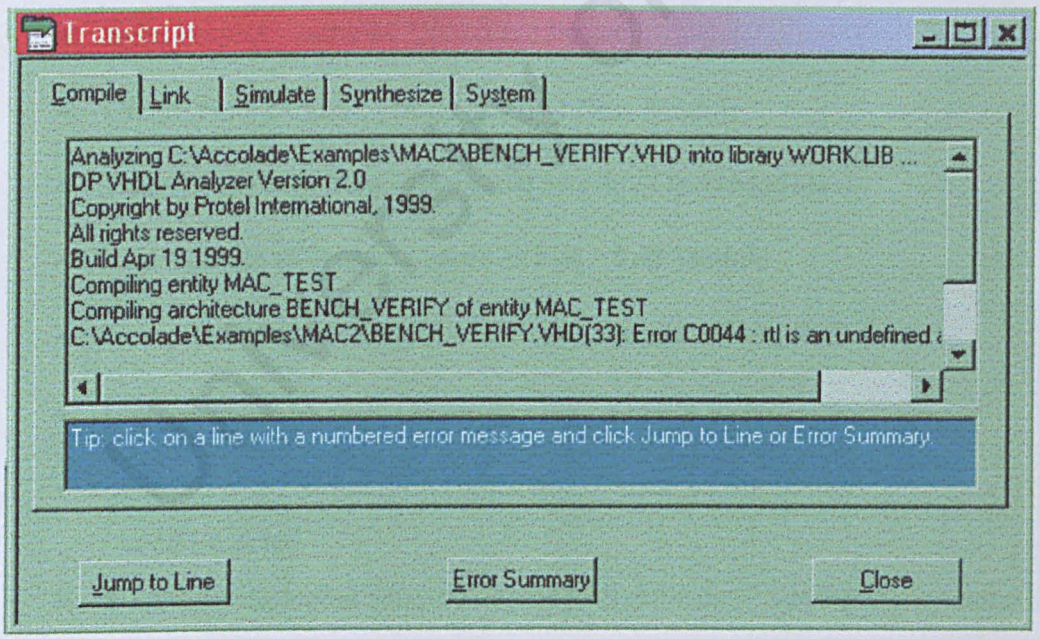


Figure 5.3 : Transcript Window

If there are any errors during this process, they are reported to the PeakVHDL transcript window. If there are no errors, the PeakSIM application appears with your project loaded, ready for simulation. Refer to the PeakSIM on-line help for information about how to control simulation, monitor signals and debug your design.

The main application window includes 13 toolbar buttons. These buttons, which can be toggled on or off are summarized below, from left to right. Note that as we move our cursor over a toolbar button, a tip appears that explains the function of that button.

- Create New Project - same as File / New Project
- Open Existing Project - same as File / Open Project
- Save Project - same as File / Save Project
- Create New Module - same as File / New Module
- Open Module or Text File - same as File / Open Module
- Add Module to Project - same as File / Add Module
- Compile Selected Module - same as Compile / Compile Selected
- Link Selected Module - same as Link / Link Selected
- Load Selected Simulation Executable - same as Simulation / Load Selected
- Synthesize Selected Module - same as Synthesize / Synthesize Selected
- Display or Change Program Options - opens the Options dialogue with the Compile folder active (same as Compile / Options... or Options / Compile...)

- Search Project - same as Edit / Search Project
- Help Contents - same as Help / Contents

5.3 PIN DESCRIPTION

To describe the behavior of digital systems in VHDL code, a designer must plan the specification of each pin and register. Therefore, the following discussion will be concentrated on the design specification of the Pipelined Multiplier Accumulator pins and registers. The function of each pin and register will also be discussed in this section.

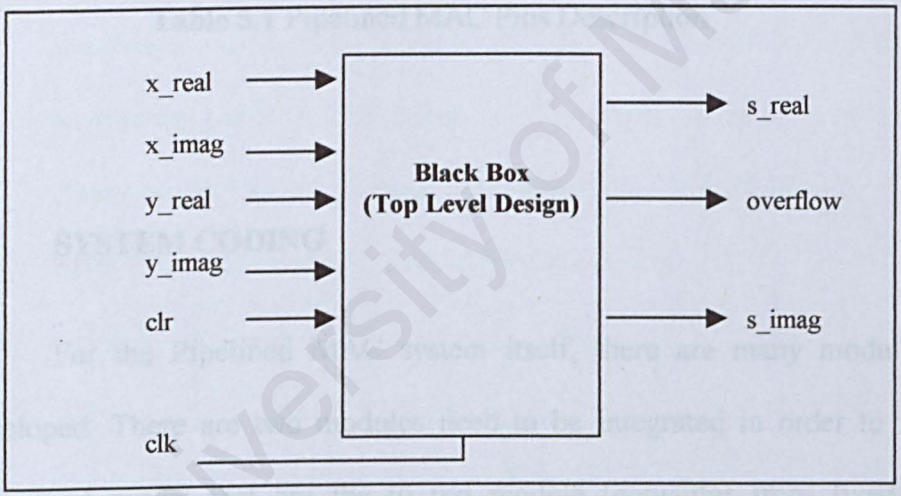


Figure 5.4 : Pipelined MAC Top Level Design Symbol

Table 5.1 will describe the function and the description of all Pipelined MAC pins available at the top-level design (**Figure 5.4**) of the VHDL implementation. The 9 pins Pipelined MAC are describe as follows:

Pin	In/out	Description
x_real	IN	**X Real Number** 8 bit input for first real number
x_imag	IN	**X Imaginary Number** 8 bit input for first imaginary number
y_real	IN	**Y Real Number** 8 bit input for second real number
y_imag	IN	**Y Imaginary Number** 8 bit input for second imaginary number
clr	IN	**Clear** Reset the input in register
clk	IN	**Clock** Input in at each rising edge
s_real	OUT	**Sum Real** Produce 16 bit real product (output)
s_imag	OUT	**Sum Imaginary** Produce 16 bit imaginary product (output)
Overflow	OUT	**Overflow Control** Produce overflow value from the system

Table 5.1 Pipelined MAC Pins Description

5.4 SYSTEM CODING

For the Pipelined MAC system itself, there are many modules being developed. There are two modules need to be integrated in order to form the behavioral model that are the to_fpo module (converter from fixed-point to floating-point representation) and to_vector module (converter from floating-point to fixed-point representation). The behavioral model allows us to focus on the algorithm without being distracted by other details at this early stage of the design. When we have the behavioral model working, we will be able to use it to generate test data for more detailed implementations.

To form the Register-Transfer-Level, there are eight modules to be integrated which is pipeline register module for 8-bit and 16-bit, multiplier module, accumulator adder module, set/reset flipflop module, adder/subtracter module, accumulator register module and overflow logic block module. This chapter only will discuss about two models, 8-bit and 16-bit pipeline register module for and set/reset flipflop module.

5.4.1 The Behavioral Model

There are two modules need to be integrated in order to form the behavioral model that are the to_fpo module (converter from fixed-point to floating-point representation) and to_vector module (converter from floating-point to fixed-point representation). **Figure 5.5** is shown the hierarchy to develop the MAC behavioral model.

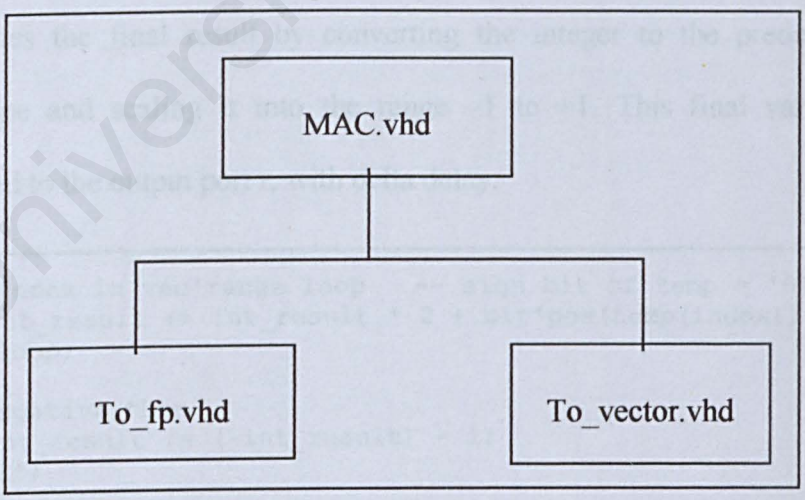


Figure 5.5 : Hierarchy tree for Pipelined MAC behavioral model

5.4.1.1 To_fpo Module

The process to convert from fixed-point binary representation to floating-point representation is sensitive to the input vector. Whenever the vector changes value, the process first converts it from std_ulogic_vector type to a bit vector in the variable temp. **Figure 5.6** is shown the coding for this process.

```
temp := to_bitvector(vec);
negative := temp(temp'left) = '1';

if negative then
    temp := not temp;
end if;
```

Figure 5.6 : Coding to convert input to a bit vector

The process then treats the bit vector as a signed binary number and converts it to an integer in the variable int_result. The process computes the final result by converting the integer to the predefined real_type and scaling it into the range –1 to +1. This final value is assigned to the output port r, with delta delay.

```
for index in vec'range loop   -- sign bit of temp = '0'
   int_result := int_result * 2 + bit'pos(temp(index));
end loop;

if negative then
   int_result := (-int_result) - 1;
end if;

-- convert to floating point and scale to [-1,+1)
    r <= real(int_result) / real(2**7);
```

Figure 5.7 : Loop to get the final result

5.4.1.2 To_vector Module

The process to convert from floating-point to fixed-point representation is sensitive to changes in the floating-point input port. The number is assumed in the range –1.0 (inclusive) to +1.0 (exclusive). If it is outside of this range, the entity will not convert the number correctly. **Figure 5.8** is shown the coding for this process.

When the number changes, the new value is scaled to an integer in the range -2^7 to $+2^7-1$ in the variable temp. This is then converted into signed binary form in the standard-logic vector result and then assigned to the output with delta delay. Lastly, temp is dividing by two to move the next most-significant bit to the least-significant bit position, in preparation for the next iteration of the loop. **Figure 5.9** is shown the coding for this process.

```
-- scale to [-2**7, +2**7) and convert to integer
   if r*real(2**15) < real(-2**7) then
      temp := -2**7;
   elsif r*real(2**7) >= real(2**7-1) then
      temp := 2**7-1;
   else
      temp := integer(r*real(2**7));
   end if;

   negative := temp < 0;
   if negative then
      temp := -(temp + 1);
   end if;

   result := (others => '0');
```

Figure 5.8 : Coding to convert input into the correct range

```
for index in result'reverse_range loop
    if ((temp mod 2) = 1) then
        result(index) := '1';
    else
        result(index) := '0';
    end if;
    temp := temp / 2;
        --result(index) := to_X01(bit'val(temp rem 2));
        -- temp := temp / 2;
    exit when temp = 0;
end loop;

if negative then
    result := not result;
    result(result'left) := '1';
end if;

return result;
vec <= result;
```

Figure 5.9 : Loop to bet the floating-point result

5.4.1.3 MAC Module

The behavioral architecture module is shown in **Appendix**. The process behavior implements the MAC algorithm. This process is sensitive to the clk signal and performs a new calculation on each rising edge. It works from the output end of the pipeline back towards the input end to avoid overwriting intermediates results from the previous clock cycle before they have been used in the current cycle.

The process first calculates the new sum and overflow status. If clr input is '1', both the accumulator and overflow variables are reset. Otherwise the process accumulates a new complex sum, based on the previous complex sum an the contents of the product registers and stores it in the accumulator register variables. The output data signals are assigned

the new contents of the accumulators and the overflow signal is set if either of the overflow register variables is set or if either of data outputs falls outside the range –1.0 to +1.0. Next, the process updates the partial products using the previously stored input values and finally stores the new input data values in the input register variables.

5.4.2 The Register-Transfer-Level Model

To form the Register-Transfer-Level model, there are eight modules to be integrated which is pipeline register module for 8-bit and 16-bit, multiplier module, accumulator adder module, set/reset flipflop module, adder/subtracter module, accumulator register module and overflow logic block module. **Figure 5.10** is shown the hierarchy to develop the Pipelined MAC Register-Transfer-Level model.

All the modules in the hierarchy will integrate to form the Register-Transfer-Level model. is shown in **Appendix**. The system coding for this process and the design for the Register-Transfer-Level model is included in **Appendix** (please refer to Appendix 6).

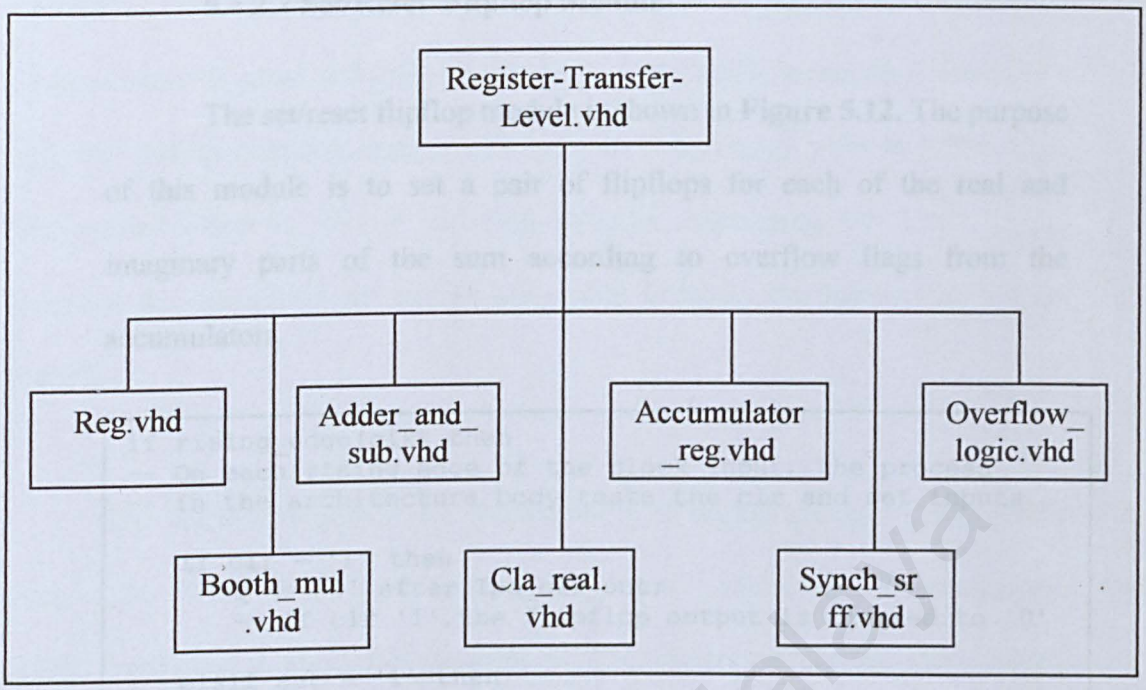


Figure 5.10 : Hierarchy tree for Pipelined MAC Register-Transfer-Level model

5.4.2.1 Pipeline register Module

The description of the pipeline register module is shown in **Figure 5.7**. The register has a clock input port, clk and stores a copy of the input data on each rising edge of the clock. The behavioral architecture body for register module contains a single process that is sensitive to changes on the *clk* port.

```
if rising_edge(clk) then
q <= d;
end if;
```

Figure 5.11 : Architecture Body for a Pipeline Register Module

5.4.2.2 Set/Reset Flipflop Module

The set/reset flipflop module is shown in **Figure 5.12.** The purpose of this module is to set a pair of flipflops for each of the real and imaginary parts of the sum according to overflow flags from the accumulators.

```
if rising_edge(clk) then
-- On each rising edge of the clock input, the process
-- in the architecture body tests the clr and set inputs

    if clr = '1' then
        q <= '0' after Tpd_clk_out;
        -- If clr '1',the flipflop output is cleared to '0'

    elsif set = '1' then
        q <= '1' after Tpd_clk_out;
        -- if set is '1', the output is set to '1'
        -- If neither input '1',flipflop state is unchanged
    end if;
end if;
```

Figure 5.12 : Architecture for Set/Reset Flipflop Module

5.5 CONCLUSION

In developing the Pipelined Multiplier Accumulator system, the software based system development is chosen to use. The system development is comprises of describing the behavioral of the digital design of the system, this behavioral description can be used for at least two purposes, the first is for the simulation of the digital circuits. A simulator uses the VHDL description to conduct a simulation that *behaves like* the physical system. Such simulation can be used to verify the behavior of the digital circuit prior to expensive fabrication. The

simulation can in fact serves as a virtual prototype in making and evaluating design trade-offs prior to finalizing the design. The VHDL simulation serves as a basis for testing complex designs and validating the design prior to fabrication. The overall effect is that of reducing redesign, shortening the design cycle, reducing the probability of design error, and bringing the product to market sooner.

The second purpose is for the synthesis of digital circuits. Design tools analyze the VHDL description and produce digital circuits that implements the behavior captured in the VHDL description. The resulting circuit descriptions can be processed rapidly to produce custom hardware and can be used to configure re-programmable hardware components to implement the design.

Thus the VHDL descriptions can in fact be used to support two complementary processes found in the design of digital systems : **simulation and synthesis**.

The next chapter (Chapter 6) will discuss about testing the VHDL model for Pipelined Multiplier Accumulator. That chapter also will discuss about testing that have been done to the Pipelined Multiplier Accumulator. The testing will be dividing to two sections that is unit testing and system integration testing.

Chapter 6

TESTING

CHAPTER 6 TESTING

6.1 INTRODUCTION

This chapter will discuss how to testing the VHDL model for Pipelined Multiplier Accumulator. We will use test benches to test this model. The test bench is a structural model with two components that is a tester and a model under test. The model under test may be a behavioral or structural VHDL model of a digital system. The tester is usually a behavioral model written using the constructs described below. Typical segments of VHDL code that can be found in the tester modules include :

- Processes to generate waveforms
- VHDL statements to read test vectors from input files and apply them to the model under test, and
- VHDL statements to record the outputs that are produced by the model under test in response to the test vectors.

For the Pipelined MAC system itself, there are test benches being constructed for each of the modules being developed. Each of the modules has their own test benches to test the input and also the output for each module.

This chapter also will discuss about testing that have been done to the Pipelined Multiplier Accumulator. The testing will be dividing to two sections that is unit testing and system integration testing. Each section has different type

of testing but still use the same testing method that is test bench. Firstly, testing will be done in unit testing then it will continue with the system integration testing.

6.2 Simulation Using PeakFPGA

To test the test benches, we will use the PeakFPGA software. In PeakFPGA, there are four options to simulate the test benches that are compiled selected, link selected, load selected and options.

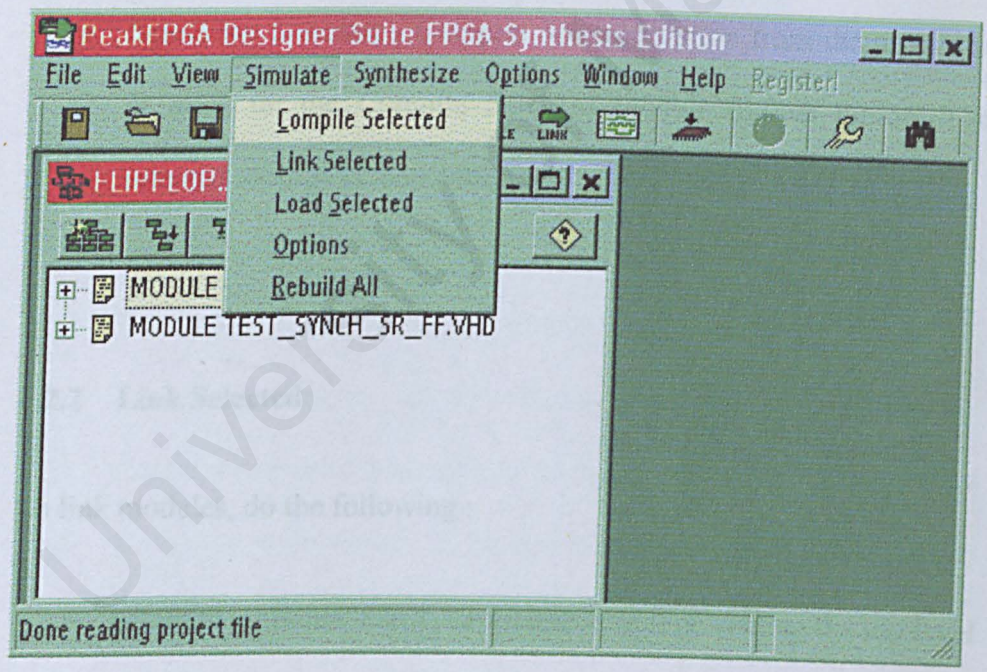


Figure 6.1 : Window show the simulate menu options

6.2.1 Compiled Selected

To compile selected VHDL modules, do the following :

- Select the module to be compiled by clicking on it once in the Hierarchy Browser.
- Select Options / Compile... from the menu bar to bring up the Compile Options dialog. Alternatively, you can bring up the dialog by clicking on the Display-or-Change-Program-Options toolbar button. Set compile options as needed. Click on the Close button to close the dialog.
- Select the Simulate / Compile Selected option from the menu bar or click on the Compile Selected Module toolbar button. The selected module is then compiled.

6.2.2 Link Selected

To link modules, do the following :

- Select the module, entity, or architecture representing the top level for the link operation by clicking on the appropriate item once in the Hierarchy Browser.
- Select Options / Link... from the menu bar to bring up the Link Options dialog. Alternatively, you can bring up the dialog by clicking on the Display-or-Change-Program-Options toolbar

button and then clicking on the Link folder tab..Once the dialog is displayed, set link options as needed. Click on the Close button to close the dialog.

- Select the Simulate / Link Selected option from the menu bar or click on the Link Selected Module toolbar button. The link operation then takes place.

6.2.3 Load selected

To load a selected simulation executable, do the following:

- Select the module, entity, or architecture you wish to load by clicking on the appropriate item once in the Hierarchy Browser.
- Select Options / Simulation... from the menu bar to bring up the Simulation Options dialog. Alternatively, you can bring up the dialog by clicking on the Display-or-Change-Program-Options toolbar button and then clicking on the Simulation folder tab..Once the dialog is displayed, set simulation options as needed. Click on the Close button to close the dialog.
- Select the Simulate / Load Selected option from the menu bar or click on the Load Selected Simulation Executable toolbar button. The PeakSIM application is then invoked and the selected simulation executable is loaded.

6.2.4 Options

To set Simulation options, select Options / Simulation... from the menu bar to bring up the Simulation Options dialog. Alternatively, you can bring up the dialog by clicking on the Display-or-Change-Program-Options toolbar button and then clicking on the Simulation folder tab..Once the dialog is displayed, set simulation options as needed. The various simulation options are discussed below :

- Update simulation executable before loading - If this option is checked, the Link process will be invoked if the simulation executable is out of date (as determined by checking the date and time stamps of the object files).
- Vector display format - This pull-down list allows you to specify the vector data display format for the waveform. Use the list to select binary, octal, decimal, or hexadecimal.
- Run to time - This field shows the default duration for the simulation run. You can reset this value by clicking on the Run to Time field and typing in a new value. This value can be overridden for individual simulation runs as needed by changing the value in the GO field in the Waveform Display.
- Step value - This field shows the default step time interval for a step simulation run. You can reset this value by clicking on the

Step Value field and typing in a new value. This value can be overridden for individual step simulation runs as needed by changing the value in the Step field in the Waveform Display.

- Unit - This field shows the unit of time to be used during simulation. To select a different unit of time, click on the Unit field to display the various options. Then click on the desired unit to select it. Valid units of time are those units defined by the VHDL language are fs (femtosecond), ps (picosecond), ns (nanosecond), us (microsecond), ms (millisecond), sec (second), min (minute) and hr (hour).
- Max signal depth - This field specifies the depth of signals to be loaded for into the Available Signals list in the Waveform Display. The depth of a signal is determined by its position in the design hierarchy. For example, a signal DUT.Clk has a signal depth of 2, while signal DUT.U1.ControlSM.Var1 has a depth of 4. You can use this option to reduce the number of signals and speed simulation loading when simulating large structural models.
- When you are finished setting options, click on the Close button to close the dialog.

6.3 Unit Testing

Unit testing is a process of testing the individual modules in the Pipelined Multiplier Accumulator. The testing conducted to ensure that the lowest levels of code are ready to assemble into the final system and that all necessary logic is present works properly.

The test bench that has been created will check the output for the input. From the waveform that has been generated by PeakFPGA, we will know either the output is correct or not. This subchapter will discuss about unit testing that have been done to pipeline register module and set/reset flipflop module.

6.3.1 Pipeline Register Module

The process for pipeline register is sensitive to changes of the clk input. The register can be test either for 8 bits or 16 bits. The process uses the rising edge to test whether the change is from a '0' state to '1' state. If so, the process updates the output using the input data. So, if the input (d) is set to "00000010" at the rising edge, the output is also "00000010". From the output, we can know that the theory for this process is correct.

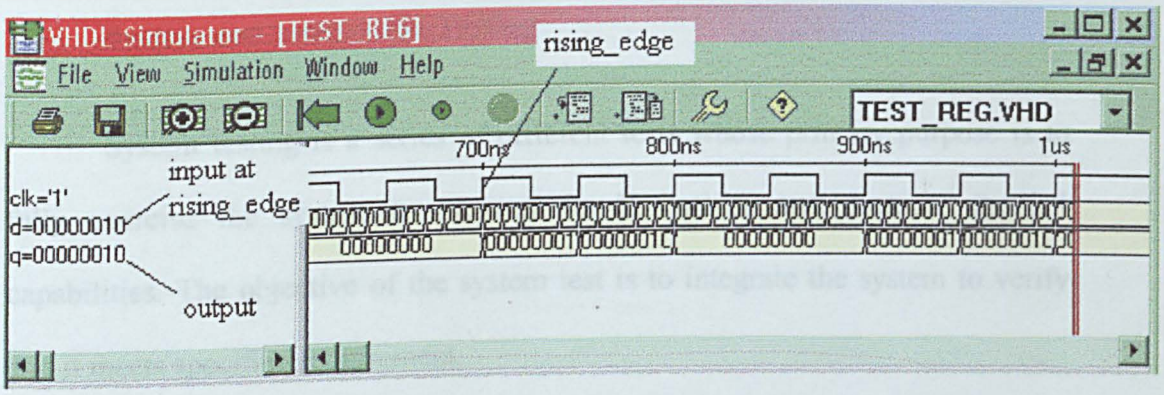


Figure 6.2 : Waveform from test bench for pipeline register

6.3.2 Set/Reset Flipflop Module

The process for set/reset flipflop will test the clr and set input on each rising edge of the clock input. The flipflop output is cleared to '0' on each rising edge of the clock input. Otherwise, if set is '1', the output is set to '1'. If neither input is set to '1', the flipflop state is unchanged. So, if the value for clr is '1', the output is '0' (cleared to zero).

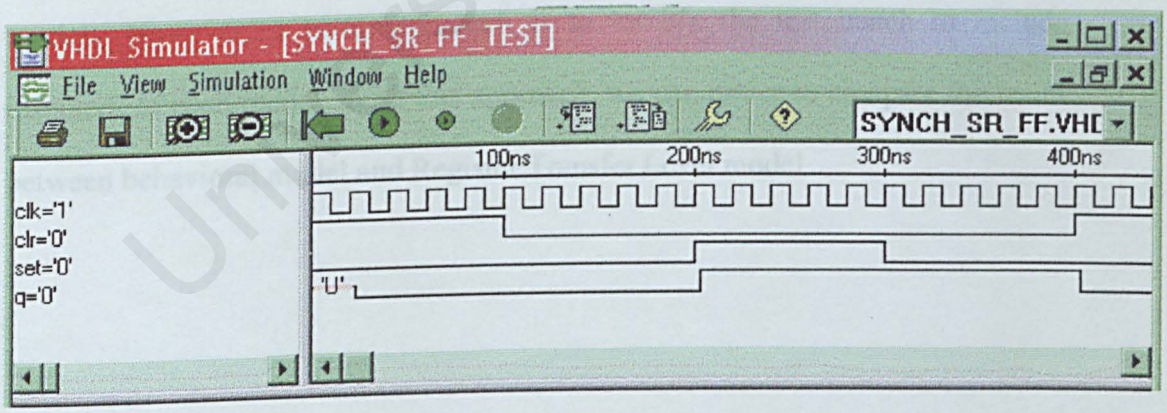


Figure 6. 3: Waveform from test bench for set/reset flipflop

6.4 System and Integration Testing

System testing is a series of different tests whose primary purpose is to fully exercise the system to uncover its limitations and measure its full capabilities. The objective of the system test is to integrate the system to verify that it meets specified requirement.

Integration system is an orderly progression of testing in which software and/or hardware elements are combined and tested until the entire system has been integrated. The purpose is to measure the correctness of each program's unit behavior once the program has been combining with other programs.

There are two models that develop for the Pipelined Multiplier Accumulator project that is behavioral model and Register Transfer Level model. Each model has own module to develop and also has specified test bench to test. We also can simulate the test bench and compare the result between both of the models. However, a better approach is to modify the test bench to include instances of each model. With this approach, we can easier to compare the result between behavioral model and Register Transfer Level model.

6.5 Conclusion

The method that use for testing in Pipelined Multiplier project is test benches. The test bench is a structural model with two components that is a tester and a model under test. The model under test may be a behavioral or structural VHDL model of a digital system.

PeakFPGA is used to create the test benches for each module and also each model. The test bench that has been created will check the output for the input. Then, each test bench is simulating to make sure the output is correct for the process in each module.

Testing will be done in two steps that are unit testing and system integration testing. Firstly, testing will test by unit or module. Then, it will integrate to test as a system. It is easier to find error in unit testing compare to system integration testing.

Chapter 7

SYSTEM EVALUATION

CHAPTER 7 SYSTEM EVALUATION

7.1 INTRODUCTION

This chapter will be discussing about the evaluation of the Pipelined Multiplier Accumulator system and the problems encountered in developing this system. The discussion will be included with the system strengths, system constraints and future enhancements for Pipelined Multiplier Accumulator. The future enhancement part will be discussed about the method that will use to improve the performance of Pipelined Multiplier Accumulator and the use of Pipelined MAC in digital processing algorithm such as filtering and equalization. The knowledge and experience gained in developing this system also included in this chapter.

7.2 DISCUSSIONS

The simulation and unit tests have shown and ensured the capability and reliability of the pipelined MAC. However, as discussed before, the reasons for the pipelined MAC is to avoid the delay in the process. In order to operate the system under a fast, continuous stream of data, the concept pipelining these multiplication and addition process is implemented in the design of the system units. The conventional MAC calculates its results by taking successive pairs of

complex numbers, one each from the two input sequences, forming their complex product and adding it to an accumulator register. The accumulator is initially cleared to zero and is reset after each pair of sequences has been processed. This process takes more time than use the pipelining technique.

In the pipelined MAC operation, two 8-bit signed fixed-point binary numbers (data) are multiplied by a multiplier block, while simultaneously two 16-bit signed fixed-point binary numbers are added together in an accumulator block which proceeds the multiplier block. It is in such way that a multiplication result is passed to the accumulator block (which is feedback to the block for next additional operation); in order to obtain the next output result. This process will take least time than the conventional process.

The implementation of pipeline concept in this system makes the operation in the faster multiplier accumulator and avoids the delay in operation. Pipelining will increase the system speed up by allow the overlapped of the tasks. The pipelined MAC has been increased the conventional execution's speed and decrease the cycle time but doesn't reduce the total time required for multiplication.

The important part in developing Pipelined MAC system is to integrate all the modules becomes a system model. There are two models in this system, which is behavioral model and Register-Transfer-Level model. Both of these models have their own module to integrate to make a complete model and system. When

problem encountered in one of the module, either the model will run with the false result or the model cannot be run.

The behavioral model is including the module to convert from fixed-point to floating-point representation and the module to convert from floating-point to fixed-point representation. The problem that encountered in module to convert from floating-point to fixed-point representation makes the problem to the behavioral model. The problem is the value of the converter is false according to the manual calculation. As the solution, we have to do a lot of reading on floating point and vector, which are the two main data types being used in the Pipelined MAC system implementation and the representation exchange between them. We also maximize the understanding of the process flow of the system. From the understanding about the flow of this system, the module has been changed and the new module for this process is created. When the problem is solve, the behavioral model can run successfully and the exactly result appear in the simulation test.

In Register-Transfer-Level model, there are seven modules must be integrate to develop the complete model. All the modules must be test before it will integrate. There no problem encountered in the Register-Transfer-Level (RTL) module or any module but the RTL module cannot be test. The problem in the simulation test is no object is available for display. The problem cannot be solving because of some reason. So, this model cannot be ensuring either can produce the exact output or not. Theoretically, the Pipelined MAC system can be

developed faster but due to the complexity in creating the VHDL module for the vector data representation, the system take more time to be complete.

7.3 SYSTEM STRENGTHS

The Pipelined MAC is able to increase the speed of the digital signal processing by implementing the following techniques:

- **Pipelining of the Multiplier Accumulator**

The previous research done by many researches have pointed out the issue to create a MAC with higher execution speed and decrease cycle time. Therefore, the best solution is used pipeline technique in multiplication unit. MAC is pipelining to avoid the delay in the process. This design of pipelined multiplier accumulator (MAC) is for a stream of complex number.

In the conventional MAC, a complex MAC operates on two sequences of complex numbers, $\{x_i\}$ and $\{y_i\}$. The MAC multiples corresponding elements of the sequences and accumulates the sum of the products. The result is

$$\sum_{i=1}^{N} x_i y_i$$

where N is the length of the sequences. Each complex number is represented in Cartesian form, consisting of a real and an imaginary part. If we are given two complex numbers *x* and *y*, their product is a complex number *p*, calculated as follows :

$$p\_real = x\_real \times y\_real - x\_imag \times y\_imag$$
$$p\_imag = x\_real \times y\_imag + x\_imag \times y\_imag$$

The sum of *x* and *y* is a complex number *s* calculated as follows :

$$s\_real = x\_real + y\_real$$
$$s\_imag = x\_imag + y\_imag$$

MAC calculates its results by taking successive pairs of complex numbers, one each from the two input sequences, forming their complex product and adding it to an accumulator register. The accumulator is initially cleared to zero and is reset after each pair of sequences has been processed.

To count the operations required for each pair of input numbers, the MAC must perform four multiplications to form partial products, then a subtraction and an addition to form the full product and finally two additions to accumulate the result, this is shown in Figure 2.2 (please refer to page 18). Since the operations must be performed in this order, the time taken to complete processing one pair of inputs is the sums of the delays for the three steps.

language, as the knowledge gained increases, the writer also had the chance to improve on skills using the PeakFPGA software, for example while using the tools provided in the software such as the compiler and the simulator.

The other constraints in develop the Pipelined Multiplier Accumulator system is time constraints. According to proposal, this system will use Booth Algorithm as the algorithm for multiplier unit and Carry Look-ahead Adder as the algorithm for accumulator unit. However, lack of time makes both of the algorithm cannot be implement and only multiplier module and accumulator adder is used in this system.

7.5 FUTURE ENHANCEMENTS

7.5.1 Module Changing

For the future enhancement, the Pipelined MAC system will upgrade by using more efficient algorithm. In the Register-Transfer-Level model, the multiplier module can change to Booth Algorithm. In Pipelined MAC, the speed of multiplier in performing its task is essentially important. Therefore, a fast multiplication technique is needed. Booth Algorithm is one approach to speed up the operation of speed up multiplication, which performs several steps of the multiplication at once. Multiplier is one part of Pipelined Multiplier Accumulator. Multiplier design starts with the elementary school algorithm for multiplication. The

computation of partial products and their accumulation into the complete product can be optimized in many ways, but an understanding of the basic steps in multiplication is important to a full appreciation of those improvement.

The Booth Algorithm is chosen for its speed up operations and simplicity. This algorithm is believed can achieve certain goals that have been highlighted earlier. Booth's algorithm takes advantage of the fact that an adder-subtracter is nearly as fast and small as a simple adder. It treats the negative and positive number uniformly. With this technique, system can multiply operand to get the partial products more quickly with the decoding method. With streams of bit 0's instead of the alternate streams of 0's and 1's, doing multiplication is not a nightmare anymore.

The implementation of this approach in hardware is the most consideration for choosing this method. The simplicity and ease understanding the hardware suits the design purposes. The components can be used more than once and this approach can save the cost in developing the system.

Besides, the accumulator module also can change to Carry Look-ahead Adder. The Carry Look-ahead Adder will solves the slow speed problem by calculating the carry signals in advance, based on the input signals. Simplicity of the designed CLA adder is obvious as the hardware construction is direct and straightforward.

Using Carry Look-ahead Adder will solve the slow speed problem that occurs when many bits need to add. Carry Look Ahead solves this problem by calculating the carry signals in advance, based on the input signals. It is based on the fact that a carry signal will be generated in two cases:

1. when both bits A_i and B_i are 1, or
2. when one of two bits is 1 and the carry-in (carry of the previous stage) is 1.

The Carry Look-ahead Adder can be broken up in two modules:

1. the Partial Full Adder, PFA, which generates S_i, P_i and G_i as defined by equations below:

$$G_i = A_i . B_i$$
$$P_i = (A_i \$ B_i)$$
$$S_i = A_i \$ B_i \$ C_i = P_i \$ C_i$$

2. the Carry Look ahead Logic, which generates the carry-out bits according to equations below:

$$C_1 = G_0 + P_0 C_0$$
$$C_4 = G_3 + P_3 G_2 + P_3.P_2.G_1 + P_3.P_2.P_1.G_0 + P_3.P_2.P_1.P_0.G_0$$

The 4-bit adder can then be built by using 4 PFAs and the Carry Look-ahead Logic.

Both Booth Algorithm and Carry Look-ahead Adder will boost or speed up the operation of Pipelined MAC compare the system, which is use the multiplier and accumulator module. The Multiplier unit mainly

determines the overall performance of the system. The multiplier design is much emphasized in order to select a suitable method for its construction.

7.5.2 Implementation of Pipelined MAC

Pipelined Multiplier Accumulator will implement in the digital processing algorithm such as filtering and equalization. *DSP Instructions* and *Execution* may specify multiple operations in a single instruction and it also must support Multiplier Accumulator (MAC). DSP usually have special loop support to reduce branch overhead that is loop an instruction or sequence. The 0 value in register usually means loop maximum number of times. If calculate loop count, must be sure that 0 does not mean 0. DSP has a specialized and complex instruction and it also has multiple operations per instruction such as *mac x0,y0,a x: (r0) + ,x0 y: (r4) + ,y0.*

7.6 KNOWLEDGE AND EXPERIENCE GAINED

The most important experience being gained is the exposure to the real world of doing programming especially in developing a system using the VHDL programming language. The lectures about VDHL solely could not contribute anything much than experiencing doing the programming itself. The writer has to deal with a lot of stages before being able to master the VHDL programming

language. The exposure to the latest software being used to do the programming is also an advantage since the software offers a lot of newly added tools that can assist in doing the programming.

The other experience in doing this project is able to learn thoroughly about the VHDL programming language while developing the module for the system. The knowledge gained on the process of designing the digital system is one of the most precious experiences that can be implemented in the working world. Knowledge about cooperation among team members, to be an interactive communicator, to be a team player and also to be creative as in the programming world there is no fixed method in creating the VHDL model, there is always some other techniques can be used to create certain models.

The most meaningful knowledge from this project is learnt about determination and also to be strong in facing a lot of obstacles in completing the system. The failure while creating the VHDL model for each module is one of the most frustrating parts in developing this system. However, with full determination and support from the advisor, the writer managed to get through all the difficulties.

7.7 CONCLUSION

The Pipelined Multiplier Accumulator system has been successfully completed and had some improvement compare to the system before. In the view of the results obtained throughout the testing and analysis done to the system, the actual performance of the Pipelined Multiplier Accumulator could be figured out, as discussed in this report. The design and construction of the system is done in the way where simplicity and fast speed criterions are emphasized.

The thesis has proved that the implementation of pipeline concept in this system make the operation in the multiplier accumulator (MAC) faster and avoid the delay and latency in that operation. Pipelining will increase the system speed up by allow the overlapped of the tasks. Pipelined MAC makes the operation of conventional MAC become more faster. This is because the pipelined MAC has pipeline register that will store the input temporarily before it is used in summation, while the system will fetch the next input. As mentioned earlier, with pipeline, three steps delay have been reduced. This will make the process more faster because the system didn't need to wait for the first input to finished it summation before the second input will entered.

The effectiveness of pipelining concept in the system's implementation is undeniable as the performance of the system is improved where the multiplication process is pipelined with the addition process. This concept should be introduced into deeper level in which internal operation of respective units could be pipelined, in order to fully implement the pipelined concept in this system. The

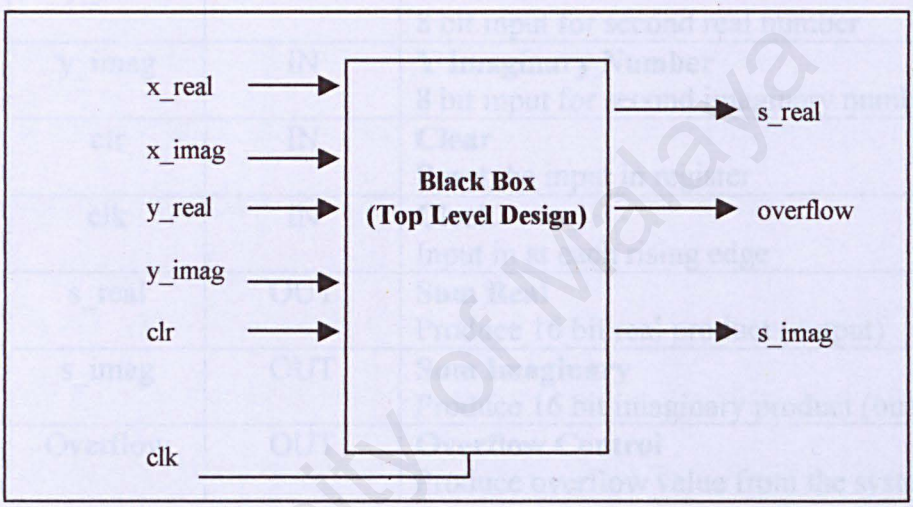
pipelined MAC has been increased the conventional execution's speed and decrease the cycle time but doesn't reduce the total time required for multiplication.

The Multiplier unit mainly determines the overall performance of the Pipelined Multiplier Accumulator system. Its design is much emphasized in order to select a suitable method for its construction. VHDL is also proven to be one of the most dominant language-based-tools, which allowed quick design-entry suites to describe the structure and behavior of digital electronic hardware designs.

APPENDIX

Appendix 1 : Pipelined MAC Pins Description

To describe the behavior of digital systems in VHDL code, a designer must plan the specification of each pin and register. Therefore, the following discussion will be concentrated on the design specification of the Pipelined Multiplier Accumulator pins and registers. The function of each pin and register will also be discussed in this section.



Pipelined MAC Top Level Design Symbol

Table below will describe the function and the description of all Pipelined MAC pins available at the top-level design of the VHDL implementation. The 9 pins Pipelined MAC are describe as follows:

Pin	In/out	Description
x_real	IN	**X Real Number** 8 bit input for first real number
x_imag	IN	**X Imaginary Number** 8 bit input for first imaginary number
y_real	IN	**Y Real Number** 8 bit input for second real number
y_imag	IN	**Y Imaginary Number** 8 bit input for second imaginary number
clr	IN	**Clear** Reset the input in register
clk	IN	**Clock** Input in at each rising edge
s_real	OUT	**Sum Real** Produce 16 bit real product (output)
s_imag	OUT	**Sum Imaginary** Produce 16 bit imaginary product (output)
Overflow	OUT	**Overflow Control** Produce overflow value from the system

Pipelined MAC Pins Description

Appendix 2 : Pipeline

Pipeline is an implementation technique in which multiple instructions are overlapped in execution. Today, pipelining is the key to make fast processors. The pipeline approach will take much less time. Pipelining is a logic design technique that adds ranks of memory elements to reduce clock cycle time at the cost of added latency. Pipelining is organizational approach is quite common used to reduces cycle time but doesn't reduce the total time required for multiplication. That's why pipeline is suitable to use in Multiplier Accumulator (MAC). Basic operation on fixed point and floating point numbers can be efficiently partitioned into sub-operations suitable for pipelining.

Pipelining is a method, which can be used to increase the speed of operation of the control processor on arithmetic function operations circuitry. They are often applied to the internal design of high speed computers, including advanced microprocessors as a type of multiprocessing. Pipelining is a technique in which a task or operation is divided into a number of subtasks that are perform in sequence. Its own logic unit performs each subtask, rather than by a single unit, which performs subtasks. The units are connected together in a serial fashion with the output of the connecting to the input of the next and all the units operate simultaneously. While one unit is performing a subtask of the ith task, the proceeding unit in the chain is performing a different subtask on the $(i+1)$th task. [Barry, 91]

There are many advantages of pipeline that make it suitable to use in MAC to reduces the latency and time delays problem. The most important advantage of pipeline is it increasing the speed of the system. It makes time to finished the clock cycle become more faster than not the time to finished without it. The cycle time T of an instruction pipeline is the time needed to advance a set of instruction one stage through the pipeline. The cycle time can be determined as

$$t = \max [t_i] + d = t_m + d \qquad i, 1 <= I <= k$$

where

t_m = maximum stage delay (delay through stage which experiences the largest delay)

k = number of stages in the instruction pipeline

d = time delay of a latch, needed to advance signals and data from one stage to the next.

In general, the time delay d is equivalent to a clock pulse and $t_m >> d$. Now suppose that n instructions are processed, with no branches. The total time required T_k to execute all n instruction is

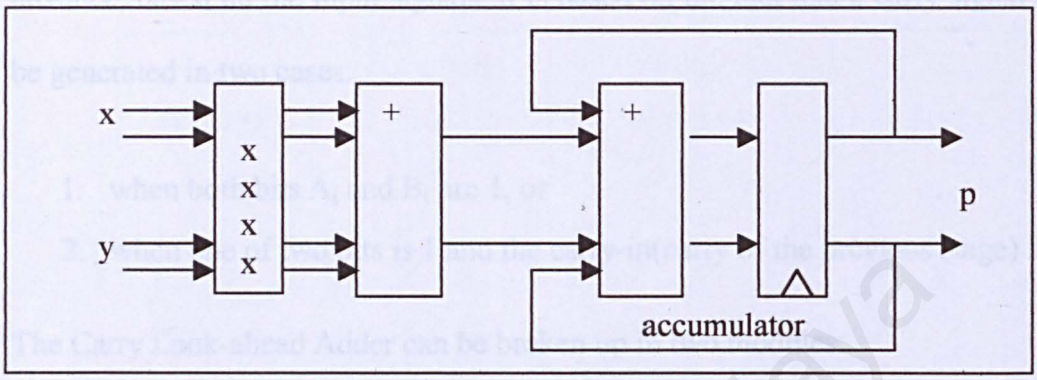
$$T_k = [k + (n-1)]_t$$

A total of k cycles are required to complete the execution of the first instruction and the remaining n-1 cycles.

The speedup factor for the instruction pipeline compared to execution without pipeline is defined as

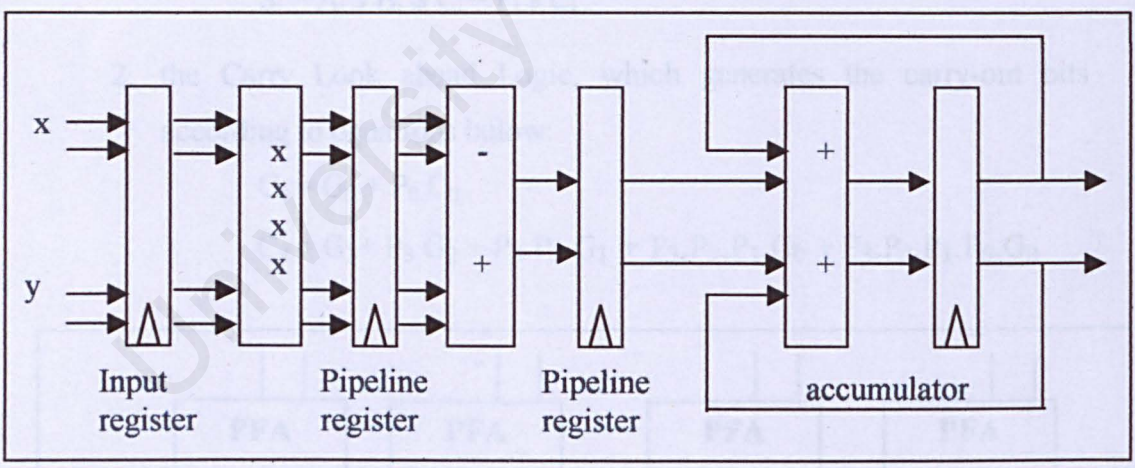
$$S_k = T_1/T_k = nkt / [k + (n-1)]t = nk / k + (n-1)$$

Besides increasing the speed of system, in some cases, the pipelining technique has the advantage of requiring less logic than a non-pipelined system. Obviously, it could be seen that, the rate of the pipelined system depends on the unit with maximum delay time. [William, 96]



Dataflow diagrams showing order of operations by the conventional MAC

Compare to conventional MAC, Pipelined MAC has pipeline register that can store the value for current process and the system will take the new input.



Dataflow diagrams showing order of operations by the pipelined MAC

Appendix 3 : Carry Look-Ahead Adder

Carry Look-ahead Adder is method that can be use in Accumulator unit. Carry Look Ahead solves slow speed problem by calculating the carry signals in advance, based on the input signals. It is based on the fact that a carry signal will be generated in two cases:

1. when both bits A_i and B_i are 1, or
2. when one of two bits is 1 and the carry-in(carry of the previous stage) is 1.

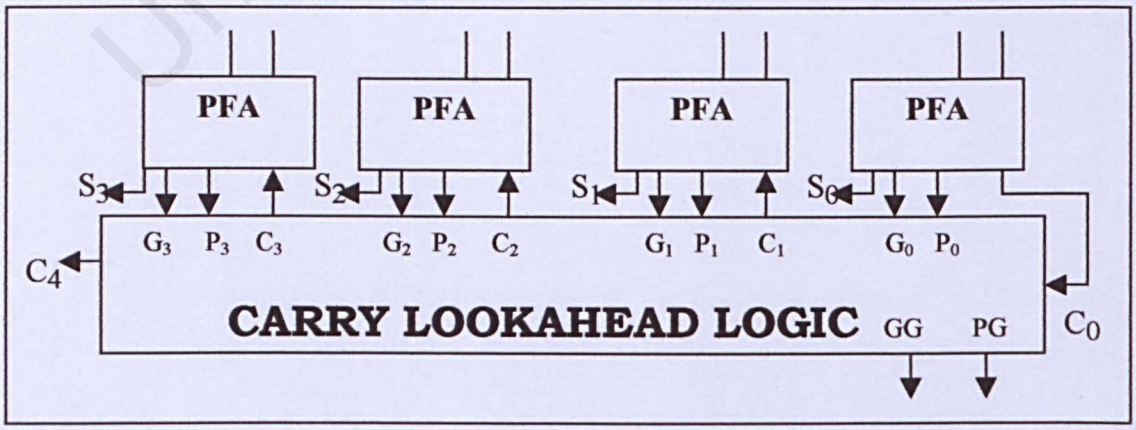
The Carry Look-ahead Adder can be broken up in two modules:

1. the Partial Full Adder (PFA), which generates S_i, P_i and G_i as defined by equations below:

$$G_i = A_i . B_i$$
$$P_i = (A_i \$ B_i)$$
$$S_i = A_i \$ B_i \$ C_i = P_i \$ C_i$$

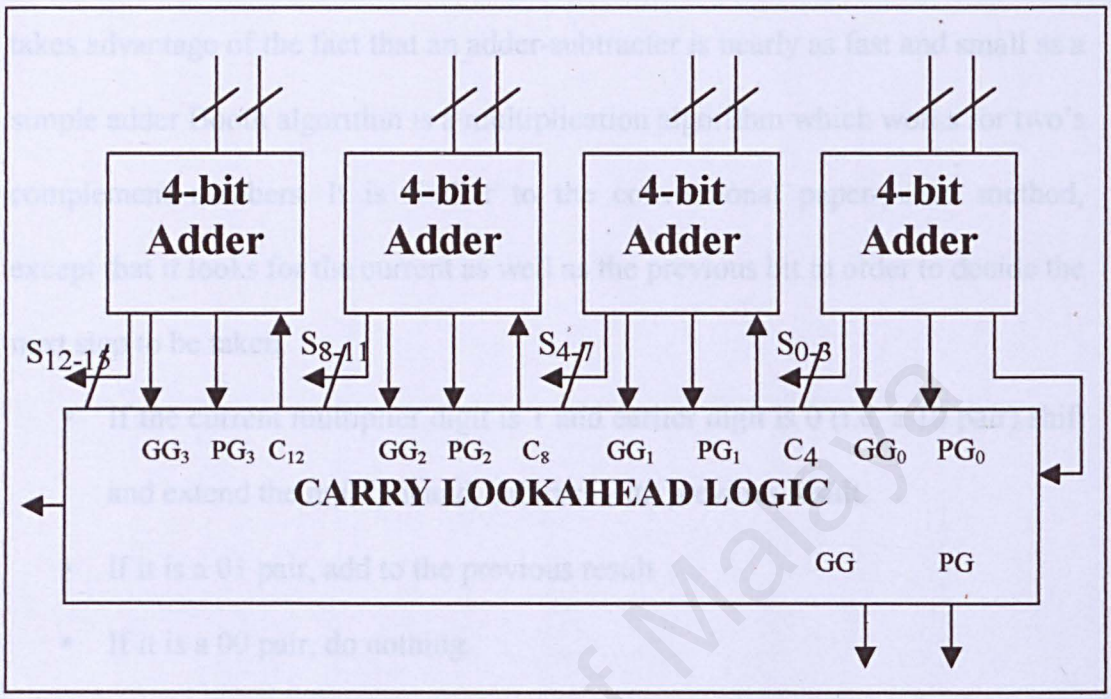
2. the Carry Look ahead Logic, which generates the carry-out bits according to equations below:

$$C_1 = G_0 + P_0 C_0$$
$$C_4 = G_3 + P_3 G_2 + P_3.P_2.G_1 + P_3.P_2.P_1.G_0 + P_3.P_2.P_1.P_0.G_0$$



Block Diagram of a 4-bit CLA

The 4-bit adder can then be built by using 4 PFAs and the Carry Look-ahead Logic.



Block Diagram of 16-bit CLA Adder

Appendix 4 : BOOTH ALGORITHM

Booth Algorithm is method that can be use in Multiplier unit. Booth's algorithm takes advantage of the fact that an adder-subtracter is nearly as fast and small as a simple adder Booth algorithm is a multiplication algorithm which works for two's complement numbers. It is similar to the conventional paper-pencil method, except that it looks for the current as well as the previous bit in order to decide the next step to be taken.

- If the current multiplier digit is 1 and earlier digit is 0 (i.e. a 10 pair) shift and extend the multiplicand, subtract with previous result.
- If it is a 01 pair, add to the previous result
- If it is a 00 pair, do nothing.

Based on the example given below, if the multiplicand and multiplier are n-bit two's complement numbers, the result is considered as 2n-bit two's complement value. The overflow bit (outside 2n bits) is ignored.

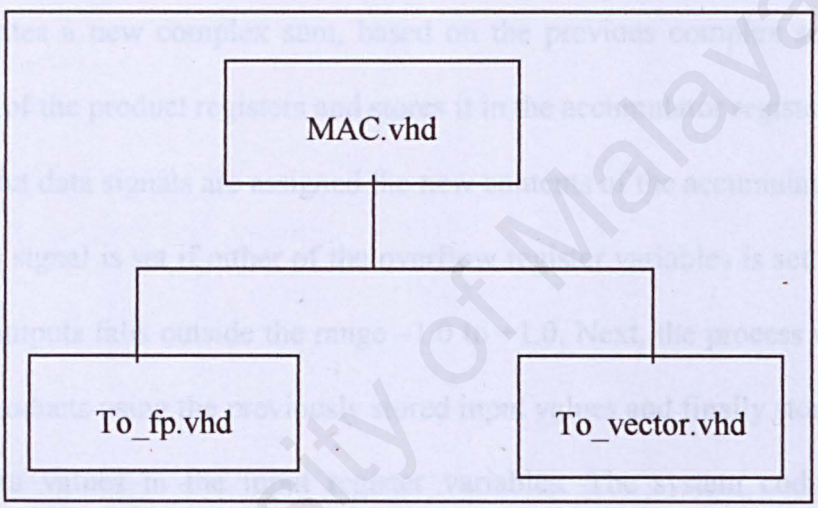
```
                4 bits
      ←  0110     6
      ←  x 0010   2
-------------------------------
        00000000
-       0110
-------------------------------
          11110100
  +         0110
-------------------------------
(1)  ←          00001100  12      (overflow bit ignored)
8 bits
```

Shown below is the proper way of the above computation:

0110 x 0010 = 0110 x (- 0010 + 0100) = - 01100 + 011000 = 1100

Appendix 5 : The Behavioral Model Of The Pipelined MAC

There are two modules need to be integrated in order to form the behavioral model that are the to_fpo module (converter from fixed-point to floating-point representation) and to_vector module (converter from floating-point to fixed-point representation). Figure below is shown the hierarchy to develop the MAC behavioral model.



Hierarchy tree for Pipelined MAC behavioral model

The process behavior implements the MAC algorithm. This process is sensitive to the clk signal and performs a new calculation on each rising edge. It works from the output end of the pipeline back towards the input end to avoid overwriting intermediates results from the previous clock cycle before they have been used in the current cycle. The input flow of this model is describe below.

The process first calculates the new sum and overflow status. If clr input is '1', both the accumulator and overflow variables are reset. Otherwise the process accumulates a new complex sum, based on the previous complex sum and the contents of the product registers and stores it in the accumulator register variables. The output data signals are assigned the new contents of the accumulators and the overflow signal is set if either of the overflow register variables is set or if either of data outputs falls outside the range –1.0 to +1.0. Next, the process updates the partial products using the previously stored input values and finally stores the new input data values in the input register variables. The system coding for the description of input flow is shown in the next page.

```
if rising_edge(clk) then
 -- work from the end of the pipeline back to the start, so as
 -- so as not to overwrite previous results in pipeline
 -- registers before they are used


 -- update accumulator and generate outputs
 if clr = '1' then
    real_sum := 0.0;
    real_accumulator_ovf := false;
    imag_sum := 0.0;
    imag_accumulator_ovf := false;
 else
    real_sum := real_product + real_sum;
    real_accumulator_ovf := real_accumulator_ovf or
                real_sum < -8.0 or real_sum >= +8.0;
    imag_sum := imag_product + imag_sum;
    imag_accumulator_ovf := imag_accumulator_ovf or
                imag_sum < -8.0 or imag_sum >= +8.0;
 end if;

 -- assigned new contents to output data signals
 fp_s_real <= real_sum;
 fp_s_imag <= imag_sum;

 -- set the overflow signal
 ovf <= boolean_to_stdulogic(
                        real_accumulator_ovf or
imag_accumulator_ovf
                        or real_sum < -1.0 or real_sum >= +1.0
                        or imag_sum < -1.0 or imag_sum >= +1.0);


 -- update product registers using partial products
 real_product := real_part_product_1 - real_part_product_2;
 imag_product := imag_part_product_1 + imag_part_product_2;

 -- update partial product registers using latched inputs
 real_part_product_1 := input_x_real * input_y_real;
 real_part_product_2 := input_x_imag * input_y_imag;
 imag_part_product_1 := input_x_real * input_y_imag;
 imag_part_product_2 := input_x_imag * input_y_real;

 -- update input registers using MAC inputs
 input_x_real := fp_x_real;
 input_x_imag := fp_x_imag;
 input_y_real := fp_y_real;

     input_y_imag := fp_y_imag;
```

The Input Flow for Behavioral Model of Pipelined MAC

The result for pipelined MAC operation is :

$$\sum_{i=1}^{N} x_i y_i$$

Based on formula above, assume that value of N is 4. From the coding of the previous page, the result for partial product is shown in the table below.

input	x_real	x_imag	y_real	y_imag	real_part1	real_part2	imag_part1	imag_pa
1	0.5	0.5	0.5	0.5	0.25	0.25	0.25	0.25
2	0.5	0.25	0.5	0.25	0.25	0.0625	0.125	0.125
3	0.25	0.5	0.25	0.5	0.0625	0.25	0.125	0.125
4	0.25	0.25	0.25	0.25	0.0625	0.0625	0.0625	0.0625

Table of input and calculation result

From the partial product, the product is calculated according to formula in the system coding. The result of product is shown in table below.

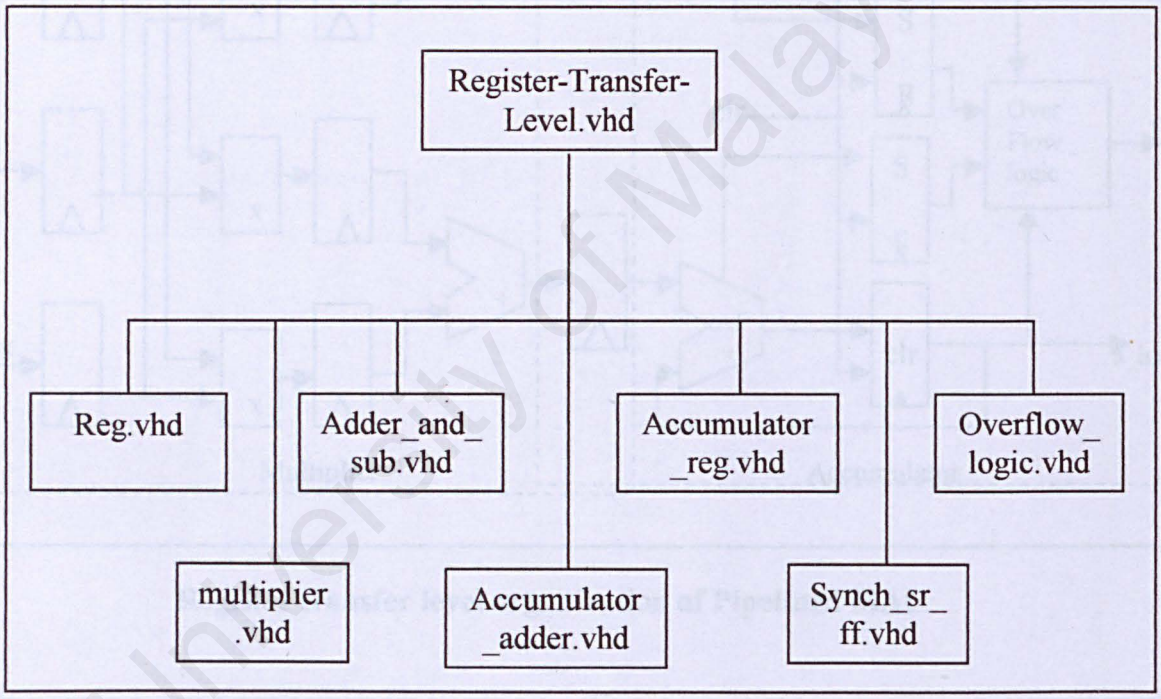
input	x_real	x_imag	y_real	y_imag	real_product	imag_product	real_sum	imag_su
1	0.5	0.5	0.5	0.5	0	0.5	0	0.5
2	0.5	0.25	0.5	0.25	0.1875	0.25	0.1875	0.75
3	0.25	0.5	0.25	0.5	-0.1875	0.25	0	1
4	0.25	0.25	0.25	0.25	0	0.125	0	1.125

Table of input and calculation result

The product will add to previous sum to get the new sum for the process. As the conclusion, the result for $\sum_{i=1}^{4} x_i y_i$ is 0.0 for real products and 1.125 for imaginary product.

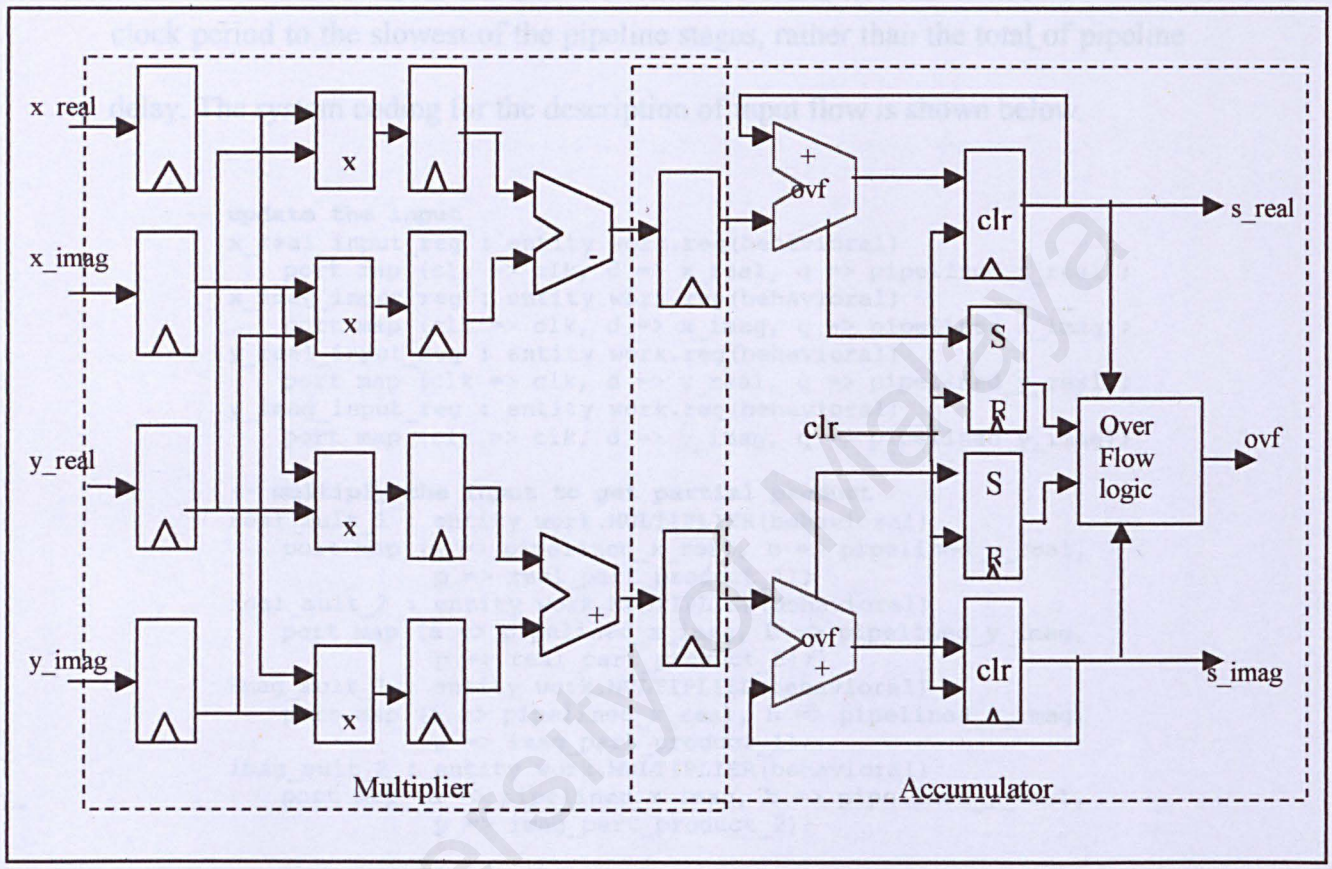
Appendix 6 : The Register-Transfer-Level model of the Pipelined MAC

To form the Register-Transfer-Level model, there are eight modules to be integrated which is pipeline register module for 8-bit and 16-bit, multiplier module, accumulator adder module, set/reset flipflop module, adder/subtracter module, accumulator register module and overflow logic block module. Figure below is shown the hierarchy to develop the Pipelined MAC Register-Transfer-Level model.



Hierarchy tree for Pipelined MAC Register-Transfer-Level model

All the modules in the hierarchy will integrate to form the Register-Transfer-Level model. The design for the Register-Transfer-Level model is shown in the figure below.



Register transfer level organization of Pipelined MAC

For RTL, the input flow of this system start when the first pair of input numbers entered the system, it is stored in the input register (register module). Then, the multiplier calculates the partial products (multiplier module) and the result stored in the first pipeline register (register module). The subtracter and adder produce the full product according to partial product (adder subtracter module). Then the

adders accumulate the product of the first pair with the previous sum and after that, the sum in the accumulator is updated (accumulator adder module). The sum including this pair is stored in the next register (accumulator register). Thereafter, successive sums are available each clock cycle. The approach can reduces the clock period to the slowest of the pipeline stages, rather than the total of pipeline delay. The system coding for the description of input flow is shown below.

```
-- update the input
   x_real_input_reg : entity work.reg(behavioral)
       port map (clk => clk, d => x_real, q => pipelined_x_real);
   x_imag_input_reg : entity work.reg(behavioral)
       port map (clk => clk, d => x_imag, q => pipelined_x_imag);
   y_real_input_reg : entity work.reg(behavioral)
       port map (clk => clk, d => y_real, q => pipelined_y_real);
   y_imag_input_reg : entity work.reg(behavioral)
       port map (clk => clk, d => y_imag, q => pipelined_y_imag);

   -- multiply the input to get partial product
   real_mult_1 : entity work.MULTIPLIER(behavioral)
       port map (a => pipelined_x_real, b => pipelined_y_real,
                 p => real_part_product_1);
   real_mult_2 : entity work.MULTIPLIER(behavioral)
       port map (a => pipelined_x_imag, b => pipelined_y_imag,
                 p => real_part_product_2);
   imag_mult_1 : entity work.MULTIPLIER(behavioral)
       port map (a => pipelined_x_real, b => pipelined_y_imag,
                 p => imag_part_product_1);
   imag_mult_2 : entity work.MULTIPLIER(behavioral)
       port map (a => pipelined_x_imag, b => pipelined_y_real,
                 p => imag_part_product_2);

   -- update pipeline registers using partial products
   real_part_product_reg_1 : entity work.reg16(behavioral)
       port map (clk => clk, d => real_part_product_1,
                 q => pipelined_real_part_product_1);
   real_part_product_reg_2 : entity work.reg16(behavioral)
       port map (clk => clk, d => real_part_product_2,
                 q => pipelined_real_part_product_2);
   imag_part_product_reg_1 : entity work.reg16(behavioral)
       port map (clk => clk, d => imag_part_product_1,
                 q => pipelined_imag_part_product_1);
   imag_part_product_reg_2 : entity work.reg16(behavioral)
       port map (clk => clk, d => imag_part_product_2,
                 q => pipelined_imag_part_product_2);

   -- adds and subtracts the partial products to get full products
   real_product_subtracter : entity work.adder_and_sub(behavioral)
       port map (mode => '1',
                 a => pipelined_real_part_product_1,
                 b => pipelined_real_part_product_2,
                 s => real_product);
   imag_product_adder : entity work.adder_and_sub(behavioral)
```

```
    port map (mode => '0',
              a => pipelined_imag_part_product_1,
              b => pipelined_imag_part_product_2,
              s => imag_product);

-- update pipeline registers using full products
real_product_reg : entity work.reg(behavioral)
    port map (clk => clk,
              d => real_product(16 downto 1),
              q => pipelined_real_product);
imag_product_reg : entity work.reg(behavioral)
    port map (clk => clk,
              d => imag_product(16 downto 1),
              q => pipelined_imag_product);

-- add full products with the previous accumulated sum
real_accumulator : entity work.accumulator_adder(behavioral)
    port map (a(9 downto 0)=> pipelined_real_product(9 downto 0),
              a(10) => pipelined_real_product(9),
              a(11) => pipelined_real_product(9),
              b => pipelined_real_sum,
              s => real_sum,
              ovf => real_accumulator_ovf); -- overflow output
imag_accumulator : entity work.accumulator_adder(behavioral)
    port map (a(11 downto 0)=>pipelined_imag_product(9 downto 0),
              a(10) => pipelined_imag_product(9),
              a(11) => pipelined_imag_product(9),
              b => pipelined_imag_sum,
              s => imag_sum,
              ovf => imag_accumulator_ovf); -- overflow output

-- update accumulator register using new sum
real_accumulator_reg : entity work.accumulator_reg(behavioral)
    port map (clk => clk, clr => clr
              d => real_sum, q => pipelined_real_sum);
imag_accumulator_reg : entity work.accumulator_reg(behavioral)
    port map (clk => clk, clr => clr,
              d => imag_sum, q => pipelined_imag_sum);

-- set the real and imaginary parts of the sum
real_accumulator_ovf_reg : entity work.SYNCH_SR_FF(behavioral)
    port map (clk => clk, clr => clr,
              set => real_accumulator_ovf,
              q => pipelined_real_accumulator_ovf);
imag_accumulator_ovf_reg : entity work.SYNCH_SR_FF(behavioral)
    port map (clk => clk, clr => clr,
              set => imag_accumulator_ovf,
              q => pipelined_imag_accumulator_ovf);

s_real <=pipelined_real_sum(11) & pipelined_real_sum(6 downto 1);
s_imag <=pipelined_imag_sum(11) & pipelined_imag_sum(6 downto 1);

-- determines the overflow output
result_overflow_logic : entity work.overflow_logic(behavioral)
    port map (
        real_accumulator_ovf =>  pipelined_real_accumulator_ovf,
        imag_accumulator_ovf => pipelined_imag_accumulator_ovf,
              real_sum => pipelined_real_sum(11 downto 7),
              imag_sum => pipelined_imag_sum(11 downto 7),
              ovf => ovf);
```

The Input Flow for RTL Model of Pipelined MAC

Appendix 7 : PeakFPGA DESIGNER SUITE FPGA SYNTHESIS EDITION

PeakVHDL is an advanced software product intended to help use VHDL for digital design projects. PeakVHDL includes an integrated VHDL simulator, VHDL source file editor, Hierarchy Browser and other resources for VHDL users. To get started using PeakVHDL, we should load one of the sample projects included in the Examples subdirectory of PeakVHDL installation. The examples provided are intended to demonstrate a variety of useful VHDL concepts, including various methods of writing test benches. These examples will also help us to understand how to create and manage a PeakVHDL project.

To load a sample project, select Open Project from the PeakVHDL File menu, and navigate to the Examples subdirectory of the PeakVHDL installation directory. Select one of the sample projects and open the .ACC file associated with that project. When we have opened a sample project, we will see two or more .VHD source files listed in the Hierarchy Browser window. We can double click on any file name listed to open a VHDL source file-editing window.

To process the project and start the PeakVHDL simulator (PeakSIM), select the top-most VHDL source file (the test bench) by clicking on it then choose Load Selected from the Simulate menu or click on the Load Selected button from the PeakVHDL toolbar. When we highlight the top-most module and choose Load Selected, the following occurs:

1) All VHDL source file modules in the project are compiled in bottom-up order as determined by the Hierarchy Browser.
2) The compiled source file modules are linked together (elaborated), and a .VX simulation executable is generated.
3) The .VX simulation executable is loaded into the PeakSIM simulation application.

If there are any errors during this process, they are reported to the PeakVHDL transcript window. If there are no errors, the PeakSIM application appears with your project loaded, ready for simulation. Refer to the PeakSIM on-line help for information about how to control simulation, monitor signals and debug your design.

If there are any errors during this process, they are reported to the PeakVHDL transcript window. If there are no errors, the PeakSIM application appears with your project loaded, ready for simulation. Refer to the PeakSIM on-line help for information about how to control simulation, monitor signals and debug your design.

The main application window includes 12 toolbar buttons. These buttons, which can be toggled on or off are summarized below, from left to right. Note that as we move our cursor over a toolbar button, a tip appears that explains the function of that button.

Icon	Icon Name
	*New Project*
	*Open Existing Project*
	*Save Project*
	*Create New Module*
	*Open Module or Text File*
	*Add module to project*
	*Compile Selected Module for Simulation*
	*Link Item for Simulation*
	*Load Selected Simulation*
	*Synthesizes Selected Module*
	*Display or Change Program Options*
	*Search Project Files*
	*Help*

Button in PeakFPGA Main Window

Menu Option for Simulation Using PeakFPGA

Compiled Selected

To compile selected VHDL modules, do the following :

- Select the module to be compiled by clicking on it once in the Hierarchy Browser.
- Select Options / Compile... from the menu bar to bring up the Compile Options dialog. Alternatively, you can bring up the dialog by clicking on the Display-or-Change-Program-Options toolbar button. Set compile options as needed. Click on the Close button to close the dialog.
- Select the Simulate / Compile Selected option from the menu bar or click on the Compile Selected Module toolbar button. The selected module is then compiled.

Link Selected

To link modules, do the following :

- Select the module, entity, or architecture representing the top level for the link operation by clicking on the appropriate item once in the Hierarchy Browser.
- Select Options / Link... from the menu bar to bring up the Link Options dialog. Alternatively, you can bring up the dialog by clicking on the Display-or-Change-Program-Options toolbar button and then clicking on the Link

folder tab..Once the dialog is displayed, set link options as needed. Click on the Close button to close the dialog.

- Select the Simulate / Link Selected option from the menu bar or click on the Link Selected Module toolbar button. The link operation then takes place.

Load selected

To load a selected simulation executable, do the following:

- Select the module, entity, or architecture you wish to load by clicking on the appropriate item once in the Hierarchy Browser.
- Select Options / Simulation... from the menu bar to bring up the Simulation Options dialog. Alternatively, you can bring up the dialog by clicking on the Display-or-Change-Program-Options toolbar button and then clicking on the Simulation folder tab..Once the dialog is displayed, set simulation options as needed. Click on the Close button to close the dialog.
- Select the Simulate / Load Selected option from the menu bar or click on the Load Selected Simulation Executable toolbar button. The PeakSIM application is then invoked and the selected simulation executable is loaded.

Options

To set Simulation options, select Options / Simulation... from the menu bar to bring up the Simulation Options dialog. Alternatively, you can bring up the dialog by clicking on the Display-or-Change-Program-Options toolbar button and then

clicking on the Simulation folder tab..Once the dialog is displayed, set simulation options as needed. The various simulation options are discussed below :

- Update simulation executable before loading - If this option is checked, the Link process will be invoked if the simulation executable is out of date (as determined by checking the date and time stamps of the object files).
- Vector display format - This pull-down list allows you to specify the vector data display format for the waveform. Use the list to select binary, octal, decimal, or hexadecimal.
- Run to time - This field shows the default duration for the simulation run. You can reset this value by clicking on the Run to Time field and typing in a new value. This value can be overridden for individual simulation runs as needed by changing the value in the GO field in the Waveform Display.
- Step value - This field shows the default step time interval for a step simulation run. You can reset this value by clicking on the Step Value field and typing in a new value. This value can be overridden for individual step simulation runs as needed by changing the value in the Step field in the Waveform Display.
- Unit - This field shows the unit of time to be used during simulation. To select a different unit of time, click on the Unit field to display the various options. Then click on the desired unit to select it. Valid units of time are those units defined by the VHDL language are fs (femtosecond), ps (picosecond), ns (nanosecond), us (microsecond), ms (millisecond), sec (second), min (minute) and hr (hour).

- Max signal depth - This field specifies the depth of signals to be loaded for into the Available Signals list in the Waveform Display. The depth of a signal is determined by its position in the design hierarchy. For example, a signal DUT.Clk has a signal depth of 2, while signal DUT.U1.ControlSM.Var1 has a depth of 4. You can use this option to reduce the number of signals and speed simulation loading when simulating large structural models.
- When you are finished setting options, click on the Close button to close the dialog.

REFERENCES

BOOKS REFERENCES

[Cavanagh, 84] Cavanagh, Joseph J.F., *Digital Computer Arithmetic Design And Implementation*, McGraw-Hill, Inc., 1984, page 98-233.

[Ash, 96] Ashenden, Peter. J., *The Designer's Guide to VHDL*, Morgan Kaufmann Publishers, Inc., 1996, page 161-188.

[Wayne, 98] Wayne Wolf, *Modern VLSI Design: System's on Silicon,* 2nd ed. Prentice-Hall, Inc., 1998, page 302,506.

[Boo, 51] Andrew D. Booth, "A signed binary multiplication technique, "*Quart. Journal of Mech. And Appl. Math*, Vol. IV, Pt. 2, page 236-240.

[William, 2000] William Stalling, *Computer Organization And Architecture,* 5th ed. Prentice-Hall International, Inc., 2000, page 423-425.

[Barry, 91] Wilkinson, Barry, *computer architecture design and performance*, Prentice-Hall International (UK) Ltd., 1991, page 102-143.

[LKS, 96] Lee Kap Soung (1996)., *The Pipelined Multiplier Accumulator in Digital Signal Processing.* Undergraduate Thesis. University of Malaya

WEB REFERENCES

[ACC, 02] http://www.accu.org/acornsig/public/caugers/volume2/fixedpoint.html

[ANG, 02] http://www.angelfire.com/in/rajesh52/verilogvhdl.html

[ATM, 02] http://www.atmel.com/atmel/acrobat/doc0467.pdf

[CHI, 02] http://www.chipcenter.com/SearchResults.jhtml

[ECE, 96] http://www-ece.rice.edu/Courses/422/1996/supafly/adder.html

[ECS, 02] http://www.ecs.umass.edu/ece/koren/arith/simulator/Add/ripple.htm

[EET, 02] http://www.eetasia.com/ART_8800132735_499481,499485.HTM

[HOW, 02] http://www.howstuffworks.com/boolean2.htm

[MAT, 97] http://www.math.toronto.edu/mathnet/answers/imaghard.html

[RES, 02] http://research.microsoft.com/~hollasch/cgindex/coding/ieeefloat.html

[SEA, 02] http://www.seas.upenn.edu/~ee201/lab/CarryLookAhead

[SYN, 02] http://www.synopsys.com/products/designware/docs/doc/dwf/datasheet

[TRA, 02] http://www.traquair.com/articles/mousetrap.pdf

[TUD, 99] http://ce.et.tudelft.nl/~robbert/mac/