

AN 16 – BIT FIXED – POINT SQUARE ROOT OPERATION USING VHDL

AHMAD JUZAILI BIN ALIAS

UNIVERSITI MALAYSIA PAHANG

AN 16 – BIT FIXED – POINT SQUARE ROOT OPERATION USING VHDL

AHMAD JUZAILI BIN ALIAS

A report submitted in partial fulfillment of the
requirements for the award of the degree of
Bachelor of Electrical (Electronics) Engineering

Faculty of Electrical and Electronics Engineering
University Malaysia Pahang

OCTOBER 2008

“All the trademark and copyrights use herein are property of their respective owner. References of information from other sources are quoted accordingly; otherwise the information presented in this report is solely work of the author.”

Signature : _____

Author : AHMAD JUZAILI BIN ALIAS

Date : 12 NOVEMBER 2008

To my beloved parents and my siblings, I'm nothing without them.

ACKNOWLEDGEMENT

First of all, I present my gratitude towards the almighty god for being able to finish this project this far. Without his blessing, this project wont even started. During doing this project, I realize many great people who is around me, my friends, my supervisor, my family and the lecturers of Faculty of Electrical and Electronic Engineering (FKEE).

Without their support, I won't able to understand my project and do it throughout the semesters. Special thanks and gratitude for my supervisor, Puan Nor Farizan Zakaria for her kind support, guidance, knowledge and motivation on doing this project. Without her help, I won't have a clue of what I would be doing in this project. How fortunate I feel to be supervised by this great and considerable person.

Lastly, I would want to thank my friends and my family for their earnest support and motivation in order for me to think positive and strive to do the best in my project. I am nobody without them besides me, may god bless you all in life.

ABSTRACT

Digital design is a part of human life nowadays; we cannot deny its existence in our life. The simple example would be our computer. Behind its functionality in doing its jobs and task, there is a complex design of digital system that play the role part of executing the operation so that our computer can perform its task when given one. On the other hand, square root is one of an important part in scientific calculation, computer graphic applications. Hence, square root is one of the operation that important for computer to performs its task. The programming languages used is VHDL (Very High Speed Integrated Circuit Hardware Description Language). The software used is ISE 10.1 that were specially made to interface with Xilinx development board. Through successfully creates it, simulation can be done and verify the system with it functionality. Hence, a digital system that operates as fixed –point square root is created.

ABTRAK

Sama ada sedar atau tidak, sistem digital merupakan perkara yang menjadi satu keperluan masakini. Contoh yang jelas sekali ialah komputer peribadi. Suatu operasi ringkas yang dilaksanakan oleh komputer mempunyai mekanisme yang menggerakkannya. Terdapat suatu reka bentuk sistem digital yang memainkan peranan membolehkan fungsi yang dijalankan beroperasi dengan baik dan sempurna. Semakin rumit tugas yang dilakukan, semakin rumit juga binaan sistem digital untuk menggerakkan fungsi tersebut. Melihat kepada operasi punca kuasa, ia adalah suatu operasi penting dalam pengiraan saintifik dan aplikasi imej. Dengan ini, suatu sistem digital direka dengan tujuan untuk menjalankan operasi punca kuasa dengan memasukkan suatu nilai dan secara automatik sistem tersebut akan mengira nilai punca kuasa nilai yang dimasukkan. VHDL digunakan untuk memprogram sistem digital tersebut. Aplikasi VHDL yang akan digunakan ialah ISE 10.1 yang dibuat oleh syarikat Xilinx untuk FPGA keluaran syarikat tersebut. Konsep atau teori operasi yang digunakan ialah Non – Restoring Square Root Algorithm. Di akhir projek ini, simulasi ke atas sistem yang direka dapat dilakukan dan diuji samada ia dapat berfungsi sebagai yang dijangka. Dengan itu, suatu sistem digital yang berfungsi telah berjaya direka.

TABLE OF CONTENT

CHAPTER	TITLE	PAGE
	DECLARATION	ii
	DEDICATION	iii
	ACKNOWLEDGEMENT	iv
	ABSTRACT	v
	ABTRAK	vi
	TABLE OF CONTENTS	vii
	LIST OF TABLES	x
	LIST OF FIGURES	xi
	LIST OF ABBREVIATIONS	xii
	LIST OF APPENDICES	xiii
1	INTRODUCTION	1
	1.1 Overview	1
	1.2 Objective	2
	1.3 Scope of the Project	2
	1.4 Problem Statement	3
	1.5 Project Contribution	3
	1.6 Thesis Organization	3

2	LITERATURE REVIEW	5
	2.1 Digital System Design	5
	2.2 VHDL	8
	2.3 Square – Root Algorithm	9
	2.3.1 Mathematical Calculation	10
	2.3.2 Algorithm Calculation	13
3	METHODOLOGY	15
	3.1 Introduction	15
	3.2 Research Methodology	15
	3.3 Square – Root Algorithm	18
	3.4 Digital System Design	20
	3.4.1 Overall Design	20
	3.4.2 Data Path Unit	21
	3.4.3 Control Unit	22
	3.5 VHDL Coding	24
	3.5.1 Overall System	25
	3.5.2 Data Path Unit	27
	3.5.3 Control Unit	30

4	RESULT AND DISCUSSION	32
4.1	Introduction	32
4.2	Data Path Simulation Result	33
4.3	Control Unit Simulation Result	34
4.4	Overall Simulation Result	35
4.5	Performance	36
4.6	Costing & Commercialization	37
5	CONCLUSION & RECOMMENDATION	38
5.1	Conclusion	38
5.2	Recommendation	48
	REFERENCES	40
	Appendixes A - B	42 - 67

LIST OF TABLES

TABLE NO.	TITLE	PAGE
3.1	Control Vector signal for Each State	23

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE
2.1	Typical Activity Flow in Digital Design	7
2.2	Binary Calculation Using Algorithm	14
3.1	Flow Chart of the Project	17
3.2	Pseudo Code of the Algorithm	18
3.3	Flow Chart of the Algorithm	19
3.4	Block Diagram of Overall System	20
3.5	Data Path Unit	21
3.6	Control Unit Block Diagram	22
3.7	State Transition	24
3.8a	VHDL coding for Master (overall system)	26
3.8b	VHDL coding for TMAP (Data path unit)	27
3.8c	VHDL coding for TMAP (Data path unit)	28
3.9	VHDL coding for TMAP (Data path unit)	29
3.10	VHDL coding for Cont_Unit (Control unit)	31
4.1	Data Path Simulation Result	33
4.2	Control Unit Simulation	34
4.3	Overall Simulation	36

LIST OF ABBREVIATION

ASIC	-	Application – Specific Integrated Circuit
DoD	-	Department of Defense
FPGA	-	Field Programmable Gate Array
FSM	-	Finite State Machine
GUI	-	Graphical User Interface
HDL	-	Hardware Description Language
RTL	-	Register Transfer Level
VHDL	-	Very high speed integrated circuit Hardware Description Language

LIST OF APPENDICES

APPENDIX	TITLE	PAGE
A	Behavioral Code for Each Components	42
B	ISE Software Tutorial Lab	50

CHAPTER 1

INTRODUCTION

1.1 Overview

Digital system design has becoming a crucial technology that moves the modern world. It has been contributing its hands in variety of field of activities. From industrial to daily life, mankind cannot deny that digital system has been an important need in this modern world now and future.

So, this development of technology of digital system is going forward for the sake of modern technology in trying to reduce the cost production and maximized the output of production as example for industrial field. In people daily life, they expected in to do various kind of task that would ease our job despite being portable and has limited resources. For example a handset, which in nowadays users not only can use it as communication tools but also as entertainment tools. This is thanks to digital system technology that has been developed and still developing as it offers many possibilities in improving it.

In developing digital system design, a common techniques use is to used VHDL language in order to programmed it in software where simulation can be perform to do analysis of designed system. This approached has its advantages as its does not make any cost as the programmed system can be programmed and erased without the effort to alter the hardware

VHDL stands for (very high speed integrated circuit hardware description language) is languages that enable the programmer describe the circuits of digital design in textual form. So, it is preferred than other programming language such as C++, Visual Basic and MATLAB which is usually a sequential languages.

Usually the hardware used would be a development board such as FPGAs that being offers by many manufacturer, for example is Spartan-3 from Xilinx. This development board has a chip that can be used to implement the designed digital system for analysis afterward.

1.2 Objective

This project has 3 objectives;

1. To use a description language to creates digital system design.
2. To choose and understand a suitable algorithm to be implemented.
3. To operates a Fixed-point square root function with a digital system design by simulation.

1.3 Scope of project

1. Output of the system would be in Fixed-point only, which means no floating point will be expected to be in the output.
2. The language used would be VHDL that stands for (very high speed integrated circuit description language).
3. The input would have maximum range of 16 bit which means the range would be 0 to +65535 of unsigned number.
4. The design would be running trough simulation only, no implementation into hardware involved.

1.4 Problem Statement

The square root function is a basic operation in computer graphic and scientific calculation application. Due to its algorithm complexity, the square root operation is hard to be designed in digital system. Digital system is the system that can realize the operation of square root operation in hardware. As known, digital system has been used in daily life or industrial purpose that may have been in need of square root operation to fully its functions.

So, this project is being done to help create a prototype of digital system design that can operate as square root operation that would be implemented in hardware devices. Furthermore, the design created is reduced in cost and high in performance by choosing the appropriate algorithm.

1.5 Project Contribution

A prototype of functioning digital system that operates the fixed-point square-root function with accurate output within the required limitation of Spartan-3 Xilinx FPGA board.

A systematic approach of designing a digital design using VHDL language with ISE 10.1 as the platform software used.

1.6 Thesis Organization

This thesis is organized into five chapters. The first chapter introduced the introduction of this project, project objective, scope of work, and contribution of this project.

Chapter 2 present the related reference studied that being used to do this project. The algorithm used is also introduced in this chapter.

Chapter 3 would explain about the project methodology which clearly explained about how this project is planned and organized in completing the project.

Chapter 4 presents the result for the system designed and discussion of overall result.

In the final chapter, the project research is summarized and the recommendations for future works are presented. The cost of the whole project and commercialization of it is also discussed here.

CHAPTER 2

LITERATURE REVIEW

This chapter explained the VHDL language, digital system design and the algorithm for square root function.

2.1 Digital System Design

Digital system can be defined as “a combination of devices designed to manipulate logical information or physical quantities that are represented in digital form; that is, the quantities can only take discrete value [7].

Other definition for Digital system is “an electronic system that operates on two-valued electric signals, referred to as ‘1’ and ‘0’ ” [5].

While, digital system design is defined as “a process that starts from the specification of requirements and produce a functional design that is eventually refined through a sequence of steps to a physical implementation.”[2].

As integrated technology has enable more and more component to be in a chip, digital system has become more complex. When digital system has become complex, detailed design of the system at gate and flip-flop level would be tedious

and time-consuming. For this reason, hardware description languages have become important in digital system design [8].

VHDL will naturally leads to top – down design methodology, in which the system is first specified at a high level and tested using a simulator. After the system is debugged at this level, the design can gradually be refined, leading to a structural description closely related to actual hardware implementation [8].

Consider the design development of application-specific integrated circuit (ASIC) for a specific purpose, unlike a microprocessor that being programmed to do variety of task. The Figure 2.1 shows us the typical sequence activities that typically takes place in ASIC design [2].

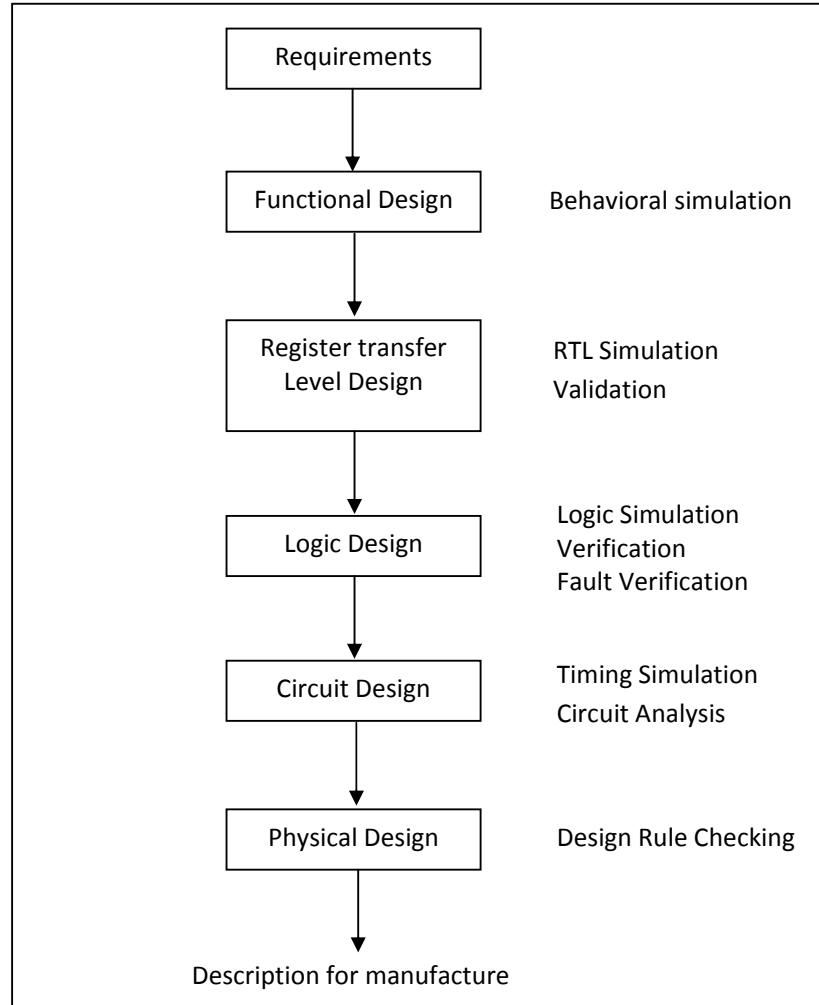


Figure 2.1: Typical activity flow in top-down digital system design

The first step is to consider the specification of the requirement that the chip is to satisfy. In other words, developers have to consider the limitations of the chips in designing a digital system so that the designed system is capable to operate on the chip. With these functional requirements, one can create a preliminary high-level functional design. Furthermore, simulation is often used to converge to a functional design that can meet the specified performance requirements [2].

With the initial functional design, developers refined it to produce a more detailed design description at the level of registers, memories, arithmetic units, and state machines. This is the register transfer level (RTL) of the design [2].

Subsequence refinement of RTL description produces a logic design that implements each of RTL components. Both RTL and logic simulation can ensure that the design meets its original specification [2].

At each level of these levels describe the design with various components. At higher or abstract level, it has a smaller number of more powerful components such as adders and memories. At lower and less abstract levels, it has a larger number of simpler, less powerful components, such as gates and transistors [2].

Each level of design hierarchy corresponds to a level of abstraction and has an associated set of activities and design tools that support the activities at this level. Moreover, throughout this hierarchy, simulation is commonly used technique. Hardware description languages such as VHDL are targeted for use throughout this design hierarchy and provide some degree of uniformity across the various levels [2].

2.2 VHDL

VHDL stands for Very High Speed Integrated Circuit Hardware Description Language. This VHDL language can be used for several goals in mind. “It may be used for the synthesis of digital circuits, verification and validation of digital designs, test vector generation for testing circuits, or simulation of digital systems” [2].

VHDL can be described as a general-purpose hardware description language that can be used to describe and simulate the operation of wide variety of digital systems, ranging in complexity from a few gates to an interconnection of many complex integrated circuits [8].

VHDL is one of three popular modern HDL languages. A second HDL is Verilog, it was developed to have a syntax similar to the C programming language. The third HDL is SystemC that is developed on 2000s by several companies. Some

people say that SystemC is not a hardware description language but rather a system description language [5].

Back to history of VHDL development, “The Department of Defense (DoD) sponsored this program with the goals of developing a new generation of high – speed integrated circuits” [2]. This development continues until a team of DOD contractor is awarded the contract to develop the language, and the 1st released in 1985 [2].

It was then transferred to IEEE for standardization, after which representatives from industries, government, and academic were further involved in its development. Many standards have been released since then, and the latest is IEEE 1164 standard [2].

Comparing with conventional procedural programming languages, such as C or Pascal, that’s describing procedures for computing a mathematical function or manipulating data, VHDL is different. Rather than the program is a recipe consisting of a sequence of steps defining how to perform a computation or manipulate data value, VHDL language describes digital systems.

One of the advantages of using VHDL languages is that it was designed to be technology independent. If a design is described in VHDL, and implemented in today’s technology, the same VHDL description could be used as a starting point for a design in some future technology [8].

2.3 Square Root Algorithm

There are many square root algorithms available for implemented using VHDL language. For example are these three algorithms: Newton-Raphson method, SRT-Redundant method and Non-Restoring Square Root Algorithm [1].

The Newton-Raphson Method operates with Iteration methods that start with initial (guess) value and improved accuracy of the result with each iteration. While the SRT-Redundant method based on recursive relation, in each iteration will be one digit shift left and addition. This method may generate a wrong resulting value at the last digit position [1].

Next, the Non-restoring method uses the two's complement representation for the square root result. With this method, an exact result value can be generated at each iteration even in the last bit. Furthermore, there is no need to do complex calculation as appear in SRT-Redundant method [1].

Non – restoring method is chosen to be used in this project, this is because of several advantages it has compared to other algorithms. Firstly, it only requires one traditional adder/subtractor in each iteration compared to Newton – Raphson Method which needs multipliers or even multiplexors [9].

Secondly, it generates the correct resulting value even in the last bit. Next, based on the resulting value of the last bit, a precise remainder can be obtained without any correction or addition operation. Finally, it can be implemented at very fast clock rate because of the very simple operation at each iteration. Hence, the Non – restoring algorithm is adopted to do this project [9].

2.3.1 Mathematical Calculation

In this section, an example is shown to show how is the calculation of square – root by hand. So that, a clear understanding how the square – root value is obtained without using calculator. The same method shown in [3] can be used to calculate the example below.

Example: Find $\sqrt{127}$ to one decimal place.

First group the numbers under the root in pairs from right to left, leaving either one or two digits on the left (6 in this case). For each pair of numbers it will get one digit in the square root.

To start, find a number whose square is less than or equal to the first pair or first number, and write it above the square root line (2).

$$\begin{array}{r} 1 \\ \sqrt{1.27} \end{array}$$

Square the 1, giving 1, write that underneath the 1, and subtract. Bring down the next pair of digits.

$$\begin{array}{r} 1 \\ \sqrt{1.27} \\ -1 \\ \hline 027 \end{array}$$

Then double the number above the square root symbol line (highlighted), and write it down in parenthesis with an empty line next to it as shown.

$$\begin{array}{r} 1 \\ \sqrt{1.27} \\ -1 \\ \hline (2_) 027 \end{array}$$

Next think what single digit number *something* could go on the empty line so that twenty-*something* times *something* would be less than or equal to 27.

$$21 \times 1 = 21$$

$$22 \times 2 = 44, \text{ so } 1 \text{ works.}$$

$$\begin{array}{r} 1 \\ \sqrt{1.27} \\ -1 \\ \hline (21) 027 \end{array}$$

Write 1 on top of line. Calculate 1×21 , write that below 027, subtract, bring down the next pair of digits (in this case the decimal digits 00).

$$\begin{array}{r} \underline{11} \\ \sqrt{1.27.00} \\ -1 \\ \hline (21) \ 027 \\ -021 \\ \hline \underline{\quad} \ 6 \end{array}$$

Then double the number above the line (11), and write the doubled number (22) in parenthesis with an empty line next to it as indicated:

$$\begin{array}{r} \underline{11} \\ \sqrt{1.27.00} \\ -1 \\ \hline (21) \ 027 \\ -021 \\ \hline (22_) \ 6 \ 00 \end{array}$$

Think what single digit number *something* could go on the empty line so that two hundred twenty-*something* times *something* would be less than or equal to 600.

$$222 \times 2 = 444$$

$$223 \times 3 = 669, \text{ so } 2 \text{ works.}$$

$$\begin{array}{r} \underline{11.2} \\ \sqrt{1.27.00} \\ -1 \\ \hline (21) \ 027 \\ -021 \\ \hline (222) \ 6 \ 00 \end{array}$$

Calculate 2×222 , write that below 600, subtract, and bring down the next digits. Then double the 'number' 112 which is above the line (ignoring the decimal point), and write the doubled number 224 in parenthesis with an empty line next to it as indicated:

$$\begin{array}{r}
 \underline{11.2} \\
 \sqrt{1.27.00.00} \\
 -1 \\
 \hline
 (21) 027 \\
 -021 \\
 \hline
 (222) 6 00 \\
 -4 44 \\
 \hline
 (224_) 1 56 00
 \end{array}$$

$2246 \times 6 = 13476$, $2246 \times 7 = 15729$, which is less than 15600, so 6 works.

$$\begin{array}{r}
 \underline{11.26} \\
 \sqrt{1.27.00.00} \\
 -1 \\
 \hline
 (21) 027 \\
 -021 \\
 \hline
 (222) 6 00 \\
 -4 44 \\
 \hline
 (224_) 1 56 00
 \end{array}$$

Thus to one decimal place, $\sqrt{127} = 11.3$

2.3.2 Algorithm Calculation

The algorithm used is Non – restoring square root algorithm. In this section, the algorithm is used to calculate the binary square – root value.

Binary Square Roots

In general, the procedure consists of taking the square root developed so far, appending 01 to it and subtracting it, properly shifted, from the current remainder. The 0 in 01 corresponds to multiplying by 2; the 1 is a new trial bit. If the resulting remainder is positive, the new root bit developed is truly 1; When the remainder goes negative, first enter a 0 as the next root bit developed. To this append 11. This result is shifted left the proper number of times and "added" to the present remainder. Using the same method in [4], it can solve the example as shown in Figure 2.2.

```

      1 0 1 1 .
-----
) 01 11 11 11 . 00
  -1
  ---
  00 11 <--- positive: first bit is a 1
  -1 01 <--- Developed root is "1"; appended 01; subtract
  -----
  11 10 11 <--- negative: 2nd bit is a 0
  +10 11 <--- Developed root is "10"; append 11 and add.
  -----
  11 11 10 11 <---Overflow: 3rd bit is a one
    1 00 11 <---Developed root is "101";append 01 and subtract
  -----
  1 00 00 11 10 <--- positive: 4th bit is a one

```

Figure 2.2: Binary calculation using algorithm

The binary number '01111111' equal to 127,

The fixed-point answer is '1110' that is equal to 11 in decimal with remainder of 6

$$127 - 11^2 = 6$$

CHAPTER 3

METHODOLOGY

3.1 Introduction

The methodology of this project is represented in this chapter, which will explain the steps and flow being done in order to complete this project. In other words, the development of the digital system will be explained in this chapter.

3.2 Research Methodology

Referring to the Figure 3.1, before we can develop the function, we have to understand the operation of the function, which is the square root function. So we have to know first how to yield a correct value of square root input value using the algorithm we have chosen. In other words, we have to know and understand the operation of the algorithm. In order to do that, we have to use the algorithm to calculate a square root value.

After that, we have to understand some digital design component involved in our algorithm operation. Some of the component is shift register, counter, adder and data register. After we understand each component involved, it's time to design the digital system. We have to keep in mind the system design need some control operation as these components cannot be enabled at the same time.

As mentioned earlier, there would be a sequence of operation involved in this system. So, we have to make a Control Unit to control the operation of each component in the system that we describe as Data Path Unit. When we finally incorporate these Control Unit and Data Path Unit, a completely functional system is produced that enable to operates square root operation.

As described in Figure 3.1, the Data Path Unit needs to be verified first before designing the Control Unit. In order to do that, we need software that compatible with VHDL language and can done a simulation of designed system. So, we use ISE 10.1 software that also use for implementing digital design to Xilinx manufactured FPGA board.

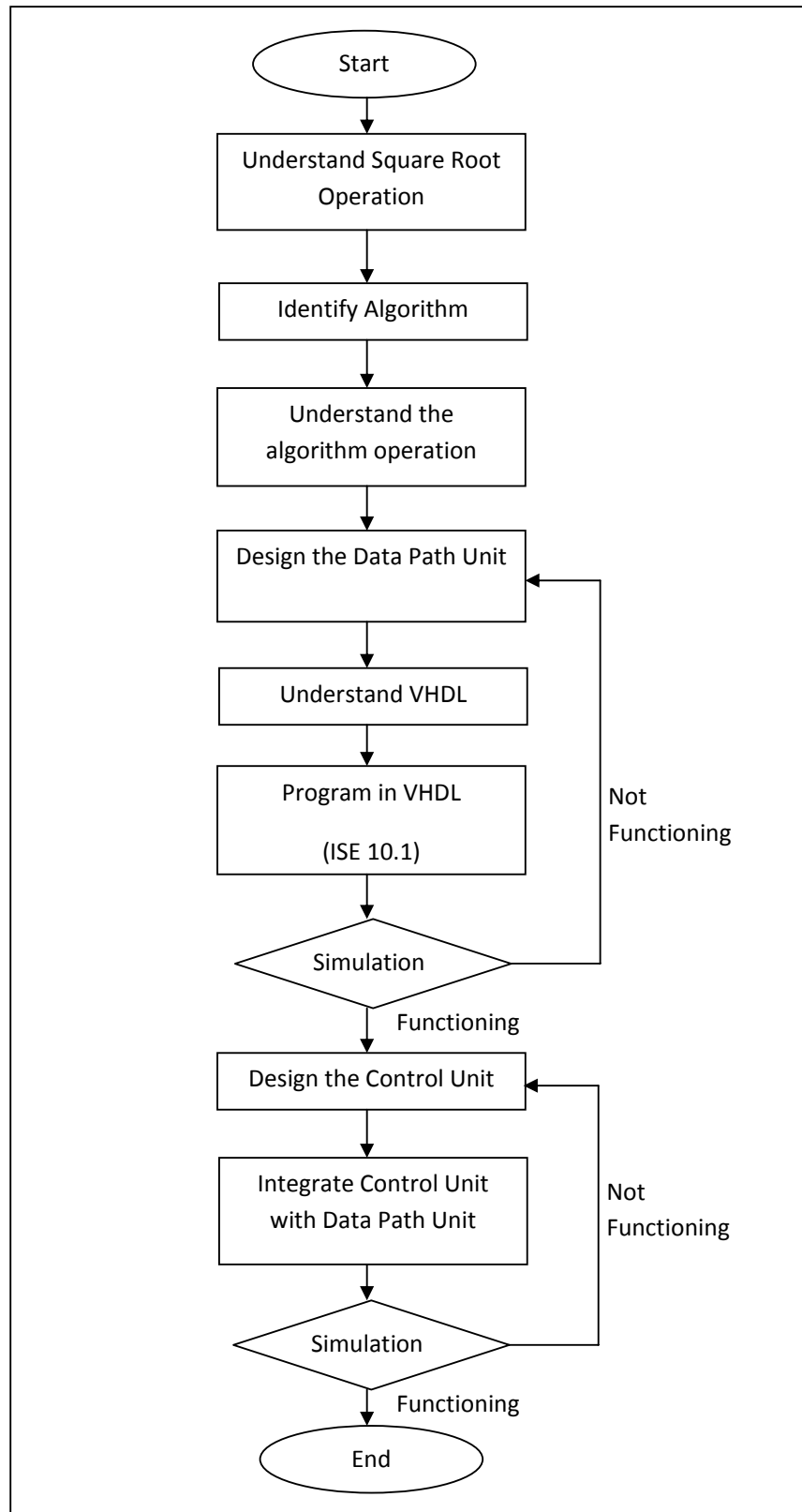


Figure 3.1: Flow Chart of the Project

3.3 Square Root Operation and Algorithm

The square root algorithm chosen in this project is Non-Restoring Square root algorithm. This algorithm was chosen for its simplicity in its operation compared to other algorithm. Thus, it would avoid using much component to compute each operation in the algorithm. The pseudo code for this algorithm is enlisted in Figure 3.2.

```

Let
  D be 32-bit unsigned integer           /*input value*/
  Q be 16-bit unsigned integer (Result)  /*Square-root value*/
  R be 17-bit integer ( $R = D - Q^2$ )    /*Remainder*/
Algorithm
Q = 0;
R = 0;
for i = 15 to 0 do                       /*for each root bit*/
  if (R >= 0)
    R = (R << 2) or (D >> (i + i) & 3);  /*new remainder:*/
    R = R - ((Q << 2) or 1);             /*-Q01*/
  Else
    R = (R << 2) or (D >> (i + i) & 3);  /*new remainder:*/
    R = R - ((Q << 2) or 3);             /*+Q11*/
  End if
  if (R >= 0) then                       /*new Q:*/
    Q = (Q << 1) or 1;
  Else
    Q = (Q << 1) or 0;                   /*newQ:*/
  End if

```

Figure 3.2: Pseudo Code of the Algorithm

The focus of the algorithm is on the partial remainder with each iteration. It generates a correct resulting bit in each iteration. The operation is subtraction or addition based on the sign of previous iteration.

Based on the flow in Figure 3.3, we can see the operation is mostly depends on the remainder of iteration. The operation starts with initial condition of remainder equal to zero. After iteration happens, its will use the current remainder to examine

the sign of the remainder. Depending on the remainder sign it will enter the square-root value which is Q. This process will continue until i equal to zero which is representing each bits of the answer. So, if the answer is 4 bits, its will loop the operation four times.

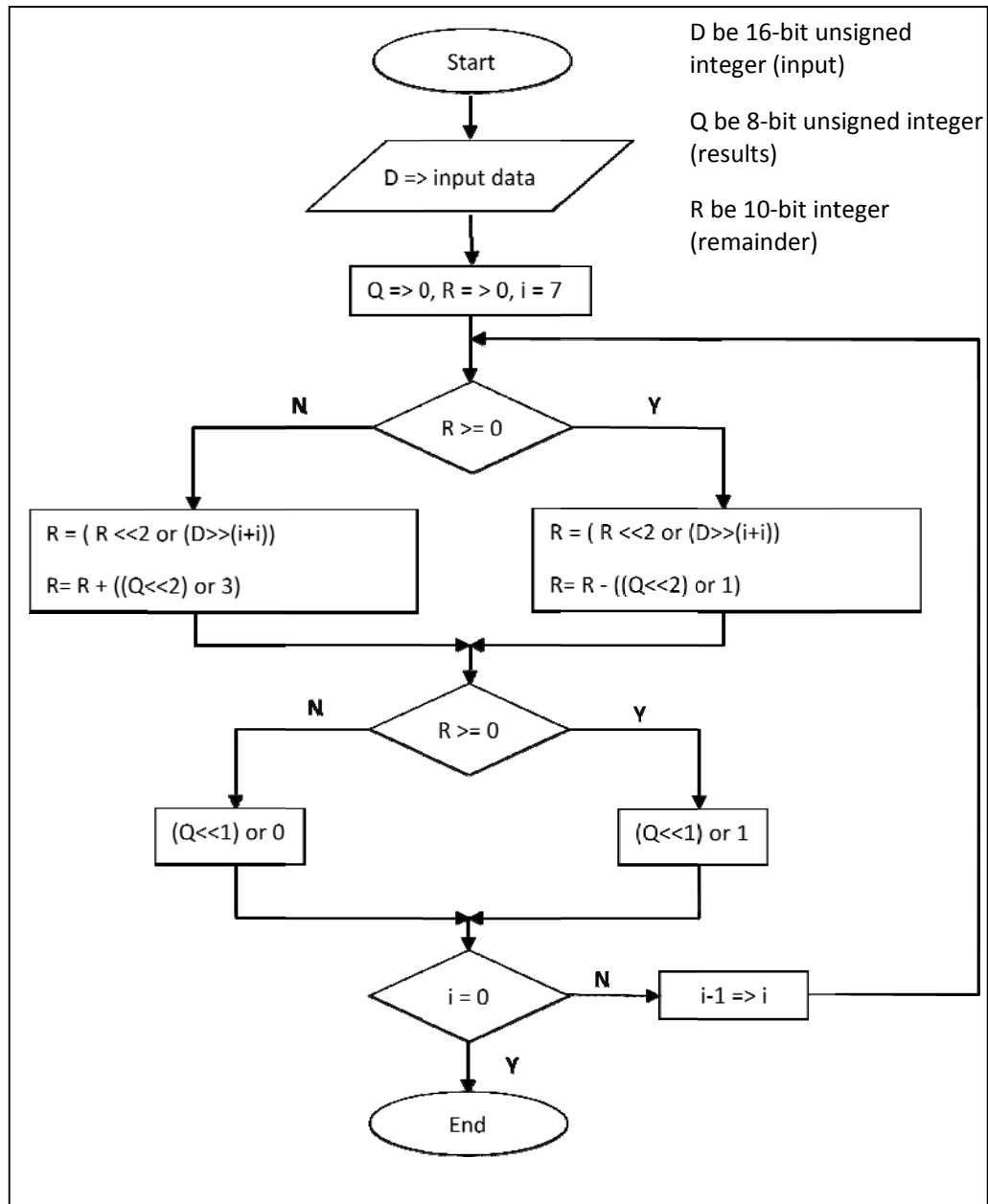


Figure 3.3: Flow Chart of the Algorithm

3.4 Digital System Design

In this part, the structural description of the digital system designed is presented here. It is divided into three parts, the overall system, data path unit and control unit.

3.4.1 Overall System

Figure 3.4 shows the block diagram of the whole system. As you can see, there are two main components that integrate to create the system. In these components also have a few smaller components operating inside it. The algorithm used is implemented in the Data Path Unit. So, the design of the algorithm is the Data Path Unit itself. The Control Unit is the one that controls the Data Path Unit operation. With integrating these two equally important components, the system will function correctly.

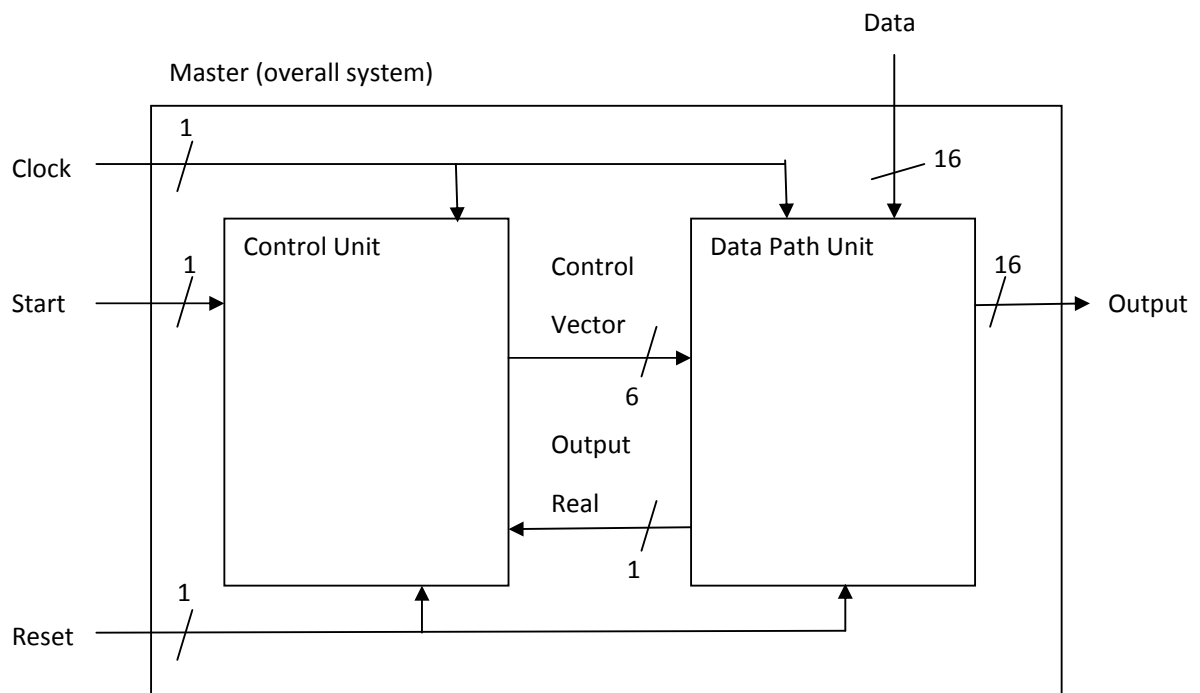


Figure 3.4: Block diagram of overall system

3.4.2 Data Path Unit

Figure 3.5 shows us the data path unit and its components inside it with some interconnection between them are visible. An adder/subtractor is used in this Data Path Unit. When the control input is 0, it will subtract, otherwise it adds. One resulting bits of the answer will generated for each clock cycle. In this case, for input value of 16 bits, the total clock cycle for generating the answer bits is 8 clock cycle.

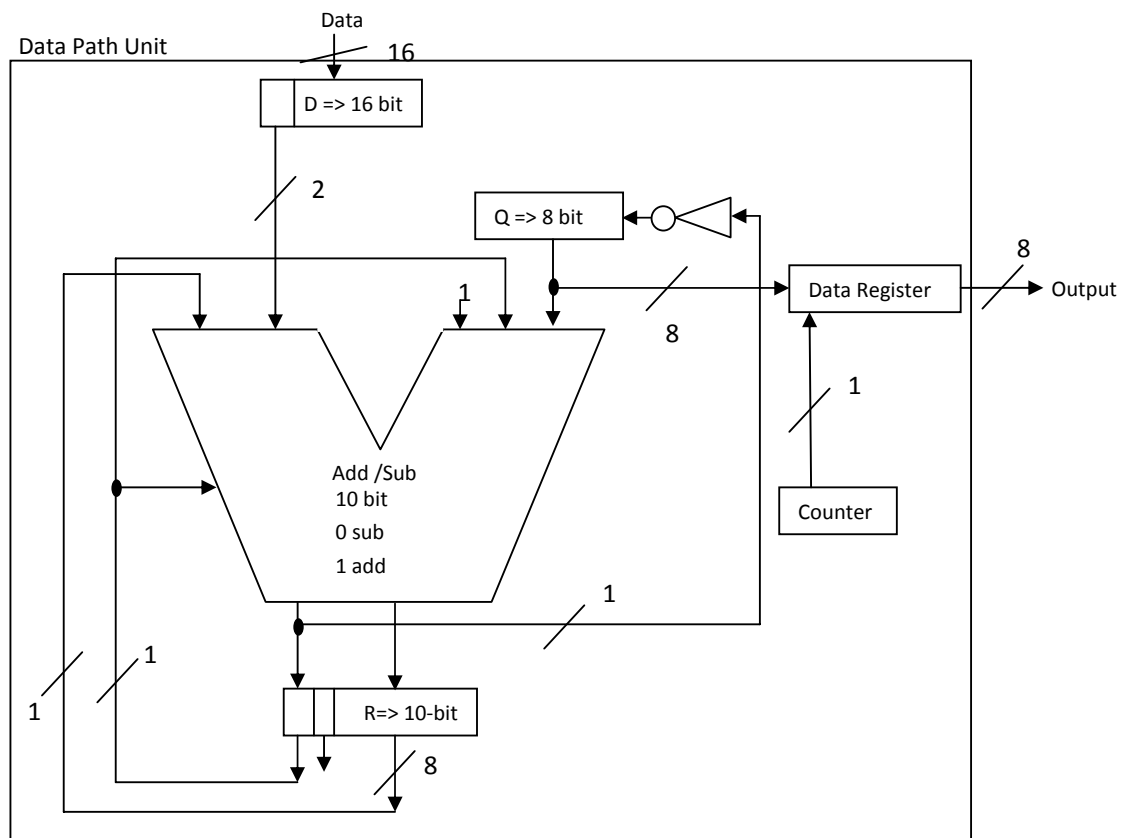


Figure 3.5 Data Path Unit

3.4.3 Control Unit

Figure 3.6 shows the block diagram of Control Unit, which consists of a finite state machine. The Control Unit's main purpose is to control the Data Path Unit's operation. This is achieved by connecting the enable signal for each component in the Data Path Unit to a data bus known as the Control Vector. With the enable signal connected to the Control Unit, it can control for which component would be enabled and operate at one time. With this in hand, the operation of the Data Path Unit can be operating smoothly and the output can be obtained.

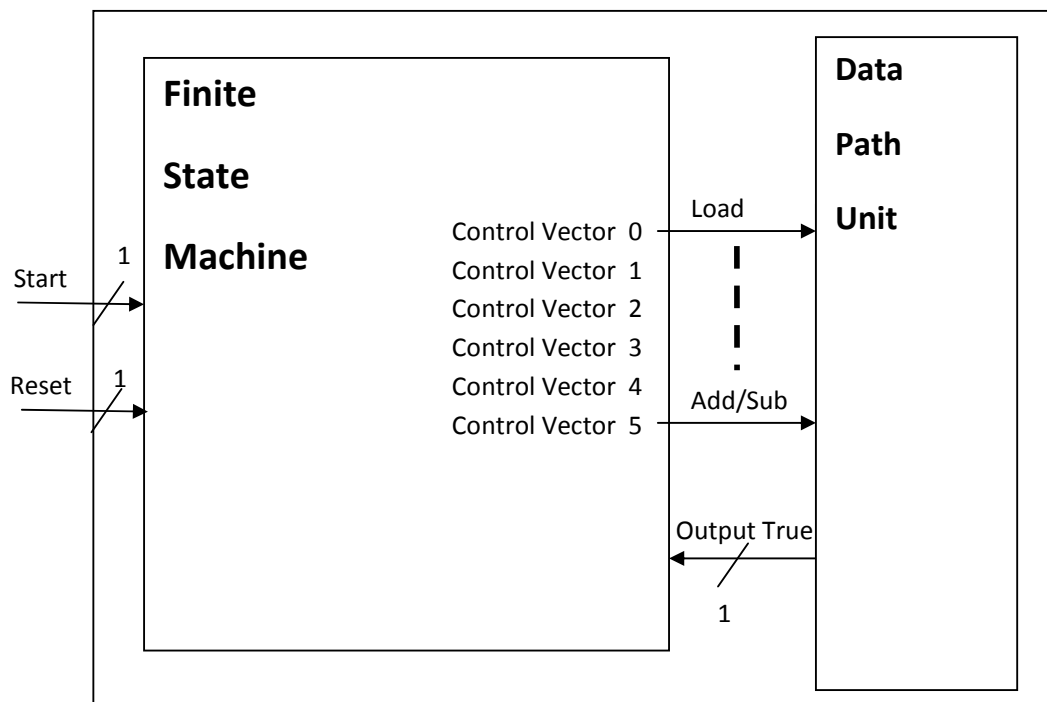


Figure 3.6: Control Unit Block Diagram

Figure 3.7 shows the state available in the Data Path operation and the Table 3.1 shows us the corresponding control vector value for each state given. From the state diagram we can see the operation of square root will execute only if the start signal is activated. When the operation is activated, it will go through a sequence of state and looping the sequence of state until the feedback signal from the Data Path Unit signifies that the current value in the output register is the real output.

We can see from here that, without the control output, the data path unit would be unable to process the data given correctly. Thus, it would be impossible to acquire the correct answer without it.

Table 3.1: Control Vector signal for each state

State	Load	D_Shift	Q_Shift	Remainder	Counter	AddSub
S0	1	0	0	0	0	0
S1	0	0	0	0	0	1
S2	0	0	0	1	0	0
S3	0	0	1	0	0	0
S4	0	1	0	0	1	0
S5	0	0	0	0	0	0

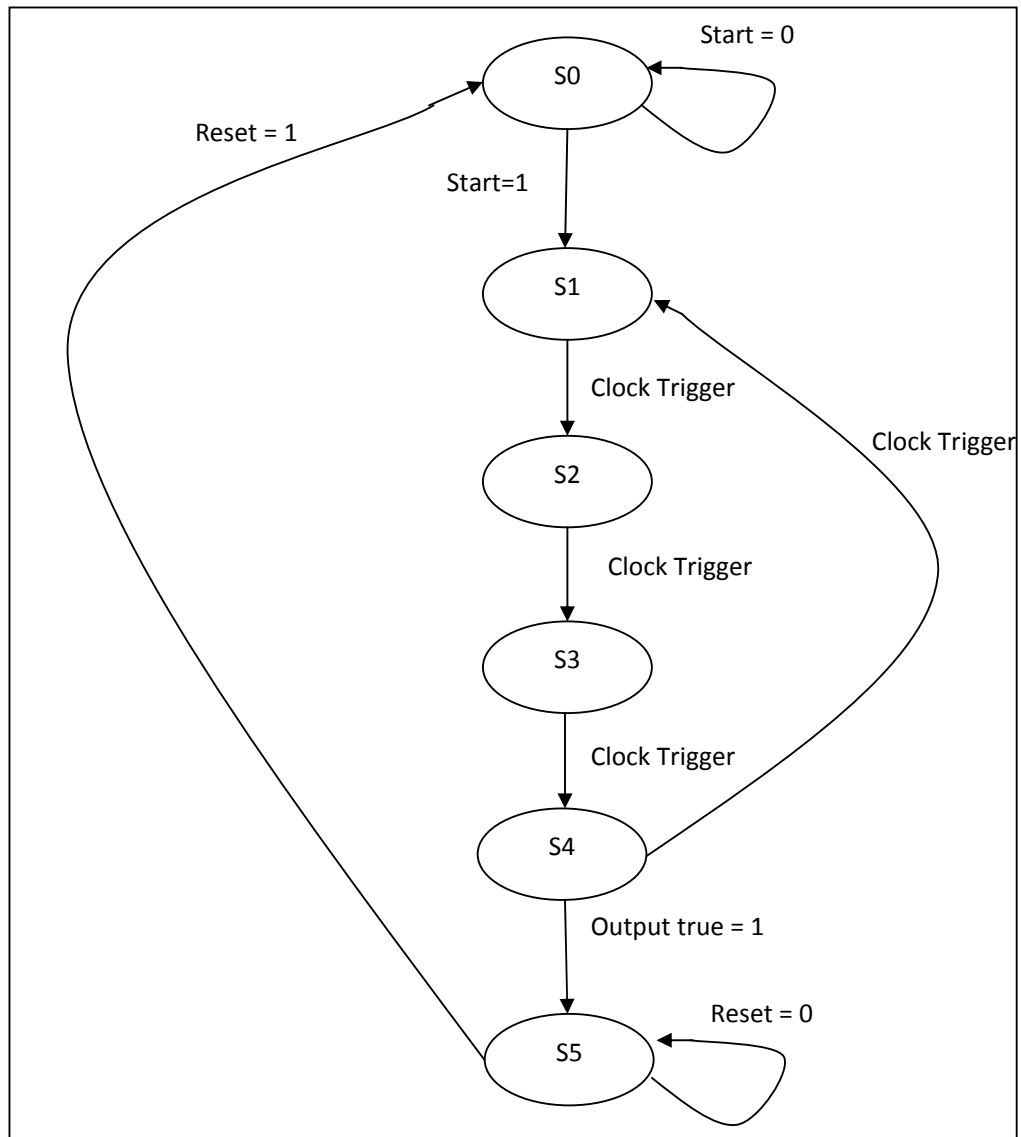


Figure 3.7: State Transition

3.5 VHDL Coding

The VHDL coding as mention earlier is the language that describes the digital system. With structural description is acquired in above part, we can do the coding with VHDL. This part is also divided by three parts that is Overall System, Data Path unit and

Control Unit. The software used is ISE 10.1 which is capable of programming in both Verilog and VHDL language. Simulation also can be done using the same software.

3.5.1 Overall System

The coding in Figure 3.8 describes the Figure 3.4, which is the block diagram of the whole system. These coding describe the behavioral or operation of the system and the interconnection between its components, that is Data path Unit and Control Unit. Notice the italic word of *TMAP* and *Cont_Unit* which represent the Data Path Unit and the Control Unit. The bold word of **M1** is describing the connection of Data Path Unit while **M2** is for Control Unit.

```

architecture Behavioral of master is
signal CV : STD_LOGIC_VECTOR(5 downto 0);
signal CountStop : STD_LOGIC;
component TMAP - - - Data Path Unit
generic(width:integer:= 16);
Port ( enable_D2 : in STD_LOGIC;
      enable_R2 : in STD_LOGIC;
      enable_Q2 : in STD_LOGIC;
      enable_count2:in STD_LOGIC;
      enable_as2 : STD_LOGIC;
      Load2 : in STD_LOGIC;
      Reset: in STD_LOGIC;
      clock_du : in STD_LOGIC;
      count_i2 : out STD_LOGIC;
      Data : in STD_LOGIC_VECTOR ((width-1) downto 0);
      Output : out STD_LOGIC_VECTOR(((width/2)-1) downto 0);
      Output_2 : out STD_LOGIC_VECTOR(((width/2)-1) downto 0));
end component;
component Cont_Unit - - - control unit
Port ( clock_CU : in STD_LOGIC;
      reset_CU : in STD_LOGIC;
      start_CU : in STD_LOGIC;
      I : in STD_LOGIC;
      Cont_Vector : out STD_LOGIC_VECTOR (5 downto 0));
end component;
begin
M1: TMAP generic map(16) port map (Load2 => CV(5), enable_D2 => CV(4), enable_Q2 =>
CV(3), enable_R2 => CV(2), enable_count2 => CV(1),enable_as2 => CV(0),Reset => reset_m,
clock_du => clock_m, count_i2=> CountStop,Output_2 => output_final, Data => data_m,
Output => output_m);
M2: Cont_Unit port map (clock_CU => clock_m, reset_CU => reset_m,
start_CU => start_m, I => CountStop, Cont_Vector => CV);
output_true <= CountStop;
end Behavioral;

```

Figure 3.8: VHDL coding for Master (overall system)

3.5.2 Data Path Unit

The coding in Figure 3.9a, 3.9b, 3.9c describes the Figure 3.5, which is the block diagram of Data Path Unit. The VHDL code describes the behavioral of the Data Path Unit and interconnection of its components. Notice the italic word of *D*, *Q*, *R*, *N*, *AddSub*, *count_reg* and *Out_reg* which represent the input register, solution register, remainder register, not gate, adder/subtractor, counter and the output register. The bold word of **U1** is describing the connection of *D* while **U2** is for *Q*, **U3** is for *R*, **U4** is for *N*, **U5** is for *AddSub*, **U6** is for *count_reg* and **U7** is for *Out_reg*. The behavioral description in VHDL for each component is in the appendix.

```

architecture Behavioral of TMAP is
  signal A1 : std_logic_vector (1 downto 0);
  signal A2,B3: std_logic_vector (((width/2)-1) downto 0);
  signal Add2 : std_logic_vector (((width/2)+2)-1) downto 0);
  signal B2,Q1,count_i3:std_logic;
  constant mask : std_logic := '1';
  component D - - - D register
  generic(width:integer:= 16);
  Port (   Output_D1 : out STD_LOGIC_VECTOR ( 1 downto 0);
  Data_D : in  STD_LOGIC_VECTOR ((width-1) downto 0);
  Enable_D : in  STD_LOGIC;
  Load : in  STD_LOGIC;
  Clock_D : in  STD_LOGIC;
  Reset_D : in  STD_LOGIC);
  end component;
  component Q - - - Q register
  generic(width:integer:= 16);
  Port (   Output_Q : out  STD_LOGIC_VECTOR (((width/2)-1) downto 0);
  Left1_Q : in  STD_LOGIC;
  Enable_Q : in  STD_LOGIC;
  Clock_Q : in  STD_LOGIC;
  Reset_Q : in  STD_LOGIC);
  end component;

```

Figure 3.9a: VHDL coding for TMAP (Data path Unit)

```

component R - - - R Register
generic(width:integer:= 16);
Port ( Data_R : in STD_LOGIC_VECTOR (((width/2)+2)-1) downto 0);
Enable_R : in STD_LOGIC;
Reset_R : in STD_LOGIC;
Clock_R : in STD_LOGIC;
Output_R3 : out STD_LOGIC_VECTOR (((width/2)-1) downto 0);
Output_R1 : out STD_LOGIC);
end component;

component N - - - Not gate logic
Port ( In_N : in STD_LOGIC;
Out_N : out STD_LOGIC);
end component;

component AddSub - - - Adder/Subtractor
generic(width:integer:= 16);
Port ( A_in2 : in STD_LOGIC_VECTOR (1 downto 0);
A_in1 : in STD_LOGIC_VECTOR (((width/2)-1) downto 0);
B_in3 : in STD_LOGIC;
B_in2 : in STD_LOGIC;
B_in1 : in STD_LOGIC_VECTOR (((width/2)-1) downto 0);
S_out : out STD_LOGIC_VECTOR (((width/2)+2)-1) downto 0);
control_op : in STD_LOGIC;
Clock_as : in STD_LOGIC;
enable_as : in STD_LOGIC;
reset_addsub : in STD_LOGIC);
end component;

component count_reg - - - Counter
Port ( clock_count : in STD_LOGIC;
reset_count : in STD_LOGIC;
enable_count : in STD_LOGIC;
count_i : out STD_LOGIC);
end component;

```

Figure 3.9b: VHDL coding for TMAP (Data path Unit)

```

component Output_reg - - - Output Register
generic(width:integer:= 16);
Port ( in_reg : in STD_LOGIC_VECTOR (((width/2)-1) downto 0);
      out_reg : out STD_LOGIC_VECTOR (((width/2)-1) downto 0);
      clk_reg : in STD_LOGIC;
      enable_reg : in STD_LOGIC;
      rst_reg : in STD_LOGIC);
end component;
begin
-----
U1: D generic map(16) port map ( Output_D1 => A1,Enable_D => enable_D2, Load => Load2,
Reset_D => Reset,Data_D => Data, Clock_D => clock_du);
-----
U2: Q generic map(16) port map ( Enable_Q => enable_Q2, Clock_Q => clock_du,
Reset_Q => Reset, Output_Q => B3, Left1_Q => Q1);
-----
U3: R generic map(16) port map ( Data_R => Add2, Enable_R => enable_R2,
Reset_R => Reset, Clock_R => clock_du, Output_R1 => B2, Output_R3 => A2);
-----
U4: N port map ( In_N => Add2(5), Out_N => Q1);
-----
U5: AddSub generic map(16) port map ( A_in1 => A2, A_in2 => A1, B_in1 => B3,
B_in2 => B2, B_in3 => mask,S_out => Add2, control_op => B2, reset_addsub => Reset,
Clock_as => clock_du,enable_as => enable_as2);
-----
U6: count_reg port map (reset_count => Reset, enable_count => enable_count2,
clock_count => clock_du, count_i => count_i3);
-----
U7: Output_reg generic map (16) port map ( in_reg => B3, out_reg => Output_2 , clk_reg =>
clock_du,
enable_reg => count_i3, rst_reg => Reset);
Output <= B3;
count_i2 <= count_i3;
end Behavioral;

```

Figure 3.9c: VHDL coding for TMAP (Data path Unit)

3.5.2 Control Unit

The coding in Figure 3.10 describe the Figure 3.6, which is the block diagram of the Control Unit. These coding describe the behavioral or operation of the Control Unit as a Finite State Machine. Notice the italic word is the coding that does the state transition and conditioning according to Figure 3.7. While the bold one is the coding that assigns the value of control vector for each corresponding state computed from Table 3.1.

```

architecture Behavioral of Cont_Unit is
type state is (S0,S1,S2,S3,S4,S5);
signal y :state;
begin
state_transition:
    process(reset_CU,clock_Cu,y)
    begin
    if reset_CU = '1' then
    y <= S0;
    elsif ( clock_CU= '0' and clock_CU'event ) then
        case y is
            when S0 => if start_CU = '1' then y <= S1; else y <= S0;end if;
            when S1 => y <= S2;
            when S2 => y <= S3;
            when S3 => y <= S4;
            when S4 => if I = '1' then y <= S5; else y <= S1;end if;
            when S5 => y <= S5;
        end case;
    end if;
    end process state_transition;
output:
    process(y)
    begin
    Cont_Vector <= (others => '0');
    case y is
        when S0 => Cont_Vector <= "100000";
        when S1 => Cont_Vector <= "000001";
        when S2 => Cont_Vector <= "000100";
        when S3 => Cont_Vector <= "001000";
        when S4 => Cont_Vector <= "010010";
        when S5 => Cont_Vector <= "000000";
    end case;
    end process output;
end Behavioral;

```

Figure 3.9: VHDL coding for Cont_Unit (Control Unit)

CHAPTER 4

RESULT AND DISCUSSION

4.1 Introduction

The result of this project is represented in this chapter, which mostly from simulation graph. Each stage result would be represented here by form of simulation graph.

4.2 Data Path Simulation Result

From the Figure 4.1 below, the Data Path Unit is tested at 8 bit input value. Since the simulations only involve Data Path Unit, the enable signal for each component is manually configured. By referring to simulation result in Figure 4.1 and Figure 3.4 , enable_d2 is for D register(shifting enable), enable_r2 is for R register, enable_q2 is for Q register, enable_c is for counter, enable_as is for adder/subtractor and load is for D register also(store data). Clock_du is the main clock to the Data Path Unit, count_i2 is the feedback signal to Control Unit that triggered when the counter has reach the required cycle of operation to get the final correct answer.

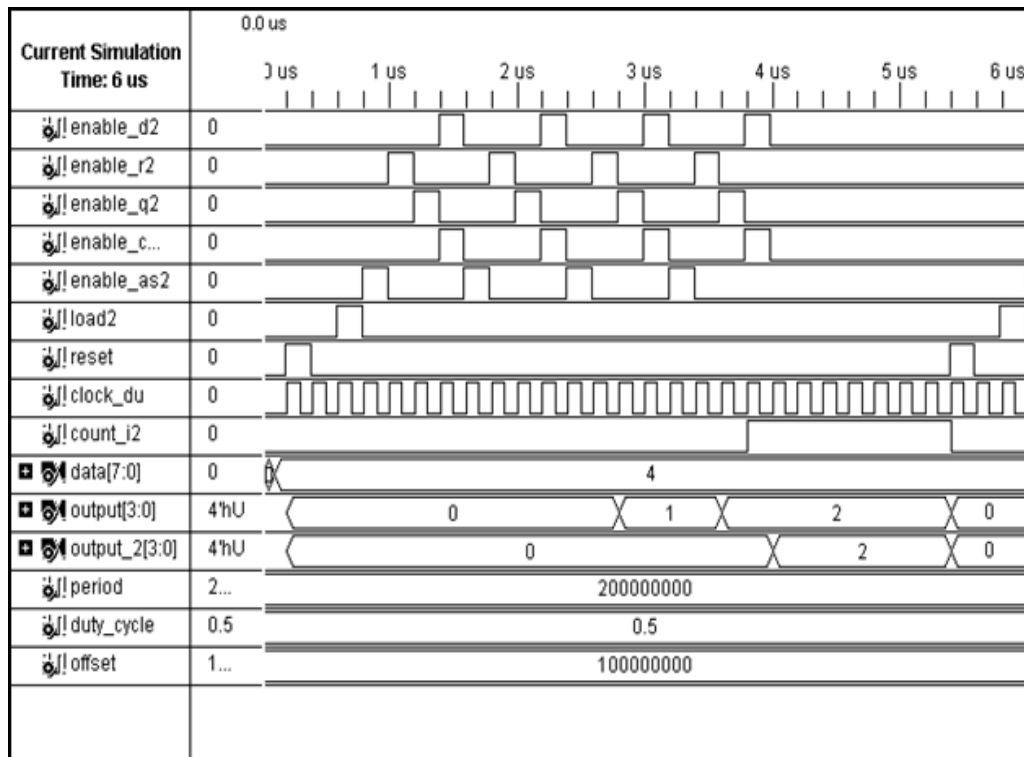


Figure 4.1: Data Path Simulation Result

'Data' is the input value given to be calculated, while 'output' is the output signal from register Q which store every answer generated for each cycle. Output_2 is the output of the output register in Data Path Unit that would be the real output of

the whole system. From the Figure 4.1 we can see the input of 8 bit with a value of 4 would compute answer after 4 cycles with value of 2. Notice that the all the enable signal is going through a sequence of pulse that cycle 4 times to compute the last answer. The signal named 'output' shows us the answer for each cycle, but the real answer is considered after the signal named 'count_i2 is triggered as 1. Hence, the signal named 'output_2' take the value from signal 'output' to be the real output of the system.

4.3 Control Unit Simulation Result

Figure 4.2 shows us the simulation results of the Control Unit. Since it only the Control Unit being simulated, the signal 'I', signal reset, signal and signal start is manually configured. Signal 'I' is the feedback signal from Data Path Unit that triggered the when the counter has reach the required cycle of operation to get the final correct answer. Start signal is the signal that triggered the operation to start its operation. The signal 'cont_vec' is the control vector signal that connected to all enable signal of Data Path Unit components.

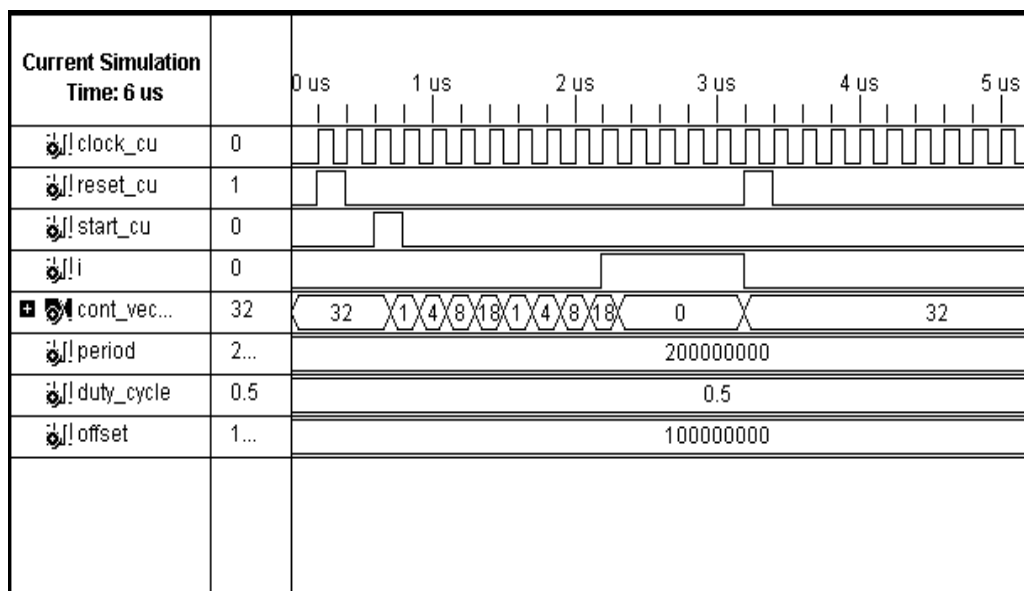


Figure 4.2: Control Unit Simulation

From the Figure 4.2 that before the signal start is triggered, the value of control vector signal will remain at 32 which corresponding to state S0 referring back to table 3.1. When the signal start is activated, the control vector output will sends a sequence of value that corresponding to a sequence of state accordingly to Table 3.1 and Figure 3.7. Until the signal 'I' is activated, it will loop the sequence value continuously. When the signal 'I' is activated, the control vector will constantly send 0 value as long as the signal 'I' is activated and the reset is not activated. When the reset signal is activated, the control vector signal will return to value 32 that is state S0.

We can see that the Figure 4.2 itself describe the behavioral of Table 3.1 and state diagram of Figure 3.7. So, the control vector is can be assure to be working accordingly as required.

4.4 Overall System Simulation Result

Figure 4.3 shows the overall system result of the simulation being done. The input data value is 25 in decimal. Output_m signal shows the answer for every cycle of operation till the last answer. While 'output_f' signal is the output of the system that only take the value when the signal 'output-true is triggered into 1 which means that is the correct value of output. In this case, the correct value is 5 in decimal.

As we know that the square root value of 25 is 5, this concluded that the simulation was successful as the operation done creates the correct answer of square-root value of input data.

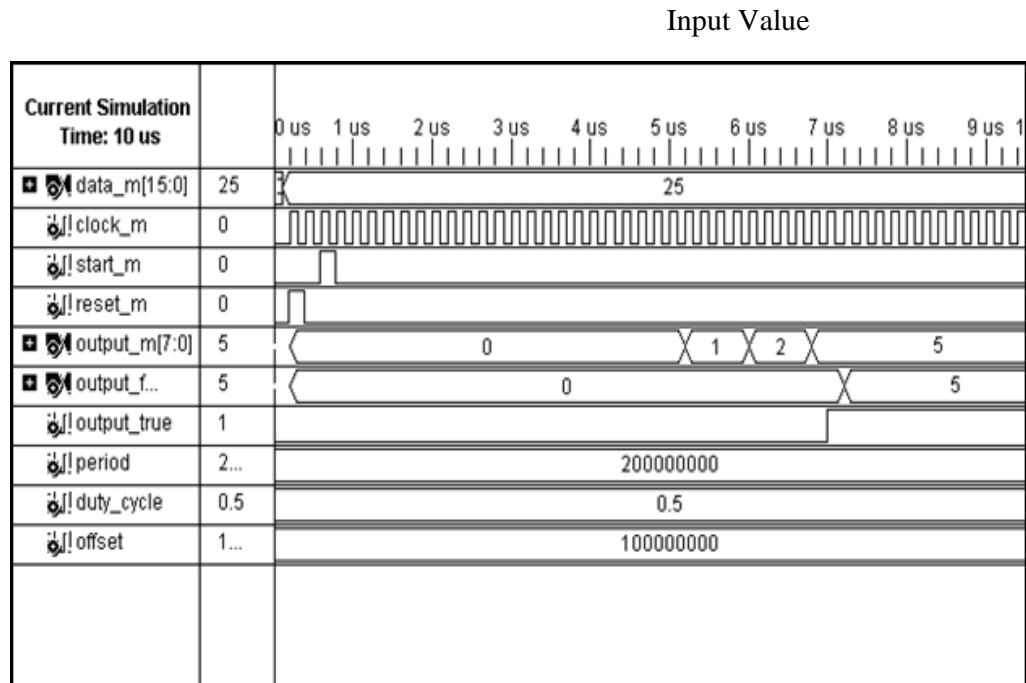


Figure 4.3: Overall Simulation result

4.5 Performance

Performance of the design can be extracted after the analysis of the design, from the simulation we have some of the performance result that is:

Total Memory Usage: 161216 kilobytes

Total Real time to Xst completion: 9.00 seconds

Total CPU time to Xst completion: 8.91 seconds

Time taken to complete process is 7.824 ns, with the frequency of 127.812 MHz.

4.6 Costing and Commercialization

The cost of this project is as listed below:

License of software ISE 10.1	: RM 200.00
Hardware used	: None
Total	: RM 200.00

While for potential of commercialization, this project is still in development or research. So it's not suitable to be commercializing it yet. The functions its offers are not significant enough to be commercialize on its own. Hence, its need to be accompanied with other product, implementation on hardware or component so that it can be commercializes.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

The proposed 16-bit Fixed-Point Square Root System is a digital system that is being developed using VHDL languages. It is being simulated using ISE 10.1 by Xilinx to verify the design and functionality. It appears to be working perfectly and is able to process a fixed-point square root value precisely.

The focus of this project is to implement a square-root algorithm that appears to be hard to implement on hardware. This was successfully done by creating a digital system to operate as according to the square root algorithm used, which is the Non-Restoring Square Root Algorithm. Using hardware description languages, the implementation of the system to hardware can be verified through simulation and it was proved to be successful.

5.2 Recommendation

The work in this project suggests that future improvement can be carried out to improve the design to achieve better output results and execution performance. Below is some proposed work:

The first recommendation is to create a floating point square-root output value. This can be done by adjusting the Data Path Unit digital design. However by adjusting the Data Path Unit, the Control Unit also has to be adjusted. In other words, we have to revise back the whole system design and control sequence of the Data Path Unit operation [10].

The second recommendation is to increase the performance of the execution time of the digital design operation of calculating the square-root value. In other word, we want the digital system to operate and compute output faster. This recommendation can be achieved by adjusting the digital design. We can try to reduce the clock cycle used to get the output, without changing the algorithm used. This approach however will increase the component used, hence increase the cost of the digital design [9].

Next recommendation is to creates a Graphical User Interface(GUI) in order for user to input the data to the digital system without the need to studies the VHDL languages. In other word, any user with various background can use and operates the digital system created to calculate the square-root of input data. This can be done by using other software that can interface with VHDL development software, for example is MATLAB. This mostly depends on the VHDL software and the developments board with which software of GUI can it interface with [11].

Finally, as we can see that this project only operates on simulation, we can implement the digital design to hardware. For the VHDL used, that is ISE 10.1, this software is specially developed to be compatible with Xilinx FPGA board. So, we can somehow implement it to Xilinx FPGA and operates the digital system in hardware. Further analysis can be done on the digital design on how effective it operates on real hardware compared to simulation result [1][9].

REFERENCES

- [1] K. Piromsopa, C. Aporn Dewan, P. Chongsatitvatana. An FPGA Implementation of a Fixed – Point Square Root Operation, Department of Computer Engineering, Chulalongkorn University.
URL www.cp.eng.chula.ac.th/~krerk/publication/iscit-sqrt.pdf
- [2] Sudhakar Yalamanchili, (2005). VHDL A Starter's Guide, Pearson Prentice Hall.
- [3] Square Root Algorithm
URL www.homeschoolmath.net/teaching/sqr-algorithm-why-works.php
- [4] Square Root Theory
URL <http://www.dattalo.com/>
- [5] Frank Vahid, Roman Lysecky, (2007). VHDL For Digital Design, Wiley.
- [6] Mohamed Khalil Hani, (2007). Starter's Guide to Digital Systems VHDL & Verilog Design, Pearson Prentice Hall.
- [7] Ronald J. Tocci, Neal S. Widmer, Gregory L. Moss, (2001). Digital Systems, Pearson Prentice Hall.
- [8] Charles H. Roth, Jr. (1998). Digital System Design Using VHDL, PWS Publishing Company.
- [9] Yamin Li, Wanming Chu, (1996). A New Non-Restoring Square Rot Algorithm and its VLSI Implementation, International Conference on Computer Design (ICCD'96).

[10] Anuja J. Thakkar . Design and implementation of Double Precision Floating Point Division and Square Root on FPGA, University of Central Florida, College of Electrical Engineering and Computer Science.

[11] Fearghal Morgan, Patrick Rocke, Martin O' Halloran. Applied VHDL Training Methodology, EDA Framework and Hardware implementation Platform, Dept of Electrical Engineering, National University of Ireland.

APPENDIX A

BEHAVIORAL CODE FOR EACH COMPONENT

A.1. 'D' Register

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity D is
```

```
generic(width:integer:= 16);
```

```
Port ( Output_D1 : out STD_LOGIC_VECTOR ( 1 downto 0);
```

```
        Data_D : in STD_LOGIC_VECTOR ((width-1) downto 0);
```

```
        Enable_D : in STD_LOGIC;
```

```
        Load : in STD_LOGIC;
```

```
        Clock_D : in STD_LOGIC;
```

```
        Reset_D : in STD_LOGIC);
```

```
end D;
```

```
architecture Behavioral of D is
```

```
Signal R: std_logic_vector((width-1) downto 0);
```

```
Begin process(Clock_D)
```



```

begin

if (Clock_D = '1' and Clock_D'EVENT) then

if (Reset_D = '1') then

R <= (others => '0');

elsif (Load = '1') then

R <= Data_D;

elsif (Enable_D = '1') then

R(0) <= '0';

R(1) <= '0';

for index in 0 to width-3 loop

R(index+2) <= R(index);

end loop; end if; end if; end process;

Output_D1 <= R((width-1) downto (width-2));

end Behavioral;

```

A.2. 'Q' Register

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

```

```

entity Q is

generic(width:integer:= 16);

Port ( Output_Q : out STD_LOGIC_VECTOR (((width/2)-1) downto 0);

      Left1_Q : in STD_LOGIC;

      Enable_Q : in STD_LOGIC;

```

```

    Clock_Q : in STD_LOGIC;

    Reset_Q : in STD_LOGIC);

end Q;

-----

architecture Behavioral of Q is

Signal R2: std_logic_vector(((width/2)-1) downto 0);

begin

process(Clock_Q)

begin

if (Clock_Q = '1' and Clock_Q'EVENT) then

if (Reset_Q = '1') then

R2 <= (others => '0');

elsif (Enable_Q = '1') then

R2(0) <= Left1_Q;

for index in 0 to (width/2)-2 loop

R2(index+1) <= R2(index);

end loop; end if; end if; end process;

Output_Q <= R2;

end Behavioral;

-----

```

A.3. 'R' Register

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

-----

```

entity R is

generic(width:integer:= 16);

Port (Data_R : in STD_LOGIC_VECTOR (((width/2)+2)-1) downto 0);

Enable_R : in STD_LOGIC;

Reset_R : in STD_LOGIC;

Clock_R : in STD_LOGIC;

Output_R3 : out STD_LOGIC_VECTOR ((width/2)-1) downto 0);

Output_R1 : out STD_LOGIC);

end R;

architecture Behavioral of R is

signal Data_R2_1,mask :STD_LOGIC;

signal Data_R2_2 :STD_LOGIC_VECTOR (((width/2)-1) downto 0);

begin

process(Clock_R)

begin

if(Clock_R'event and Clock_R = '1')then

if(Reset_R='1')then

Data_R2_1 <= '0';

Data_R2_2 <= (others => '0');

elsif Enable_R = '1' then

Data_R2_1 <= Data_R(((width/2)+2)-1);

mask <= Data_R(width/2);

Data_R2_2 <= Data_R(((width/2)-1) downto 0);

end if; end if; end process;

Output_R1 <= Data_R2_1;

Output_R3 <= Data_R2_2;

end Behavioral;

A.4. 'N' Gate Logic

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity N is
```

```
    Port ( In_N : in  STD_LOGIC;
          Out_N : out STD_LOGIC);
```

```
end N;
```

```
architecture Behavioral of N is
```

```
begin
```

```
    Out_N <= not In_N;
```

```
end Behavioral;
```

A.5. Adder/Subtractor

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity AddSub is
```

```
    generic(width:integer:= 16);
```

```

Port ( A_in2 : in STD_LOGIC_VECTOR (1 downto 0);

      A_in1 : in STD_LOGIC_VECTOR (((width/2)-1) downto 0);

      B_in3 : in STD_LOGIC;

      B_in2 : in STD_LOGIC;

      B_in1 : in STD_LOGIC_VECTOR (((width/2)-1) downto 0);

      S_out : out STD_LOGIC_VECTOR (((width/2)+2)-1) downto 0);

      control_op : in STD_LOGIC;

      Clock_as : in STD_LOGIC;

      enable_as : in STD_LOGIC;

      reset_addsub : in STD_LOGIC);

end AddSub;

-----

architecture Behavioral of AddSub is

signal R3 : std_logic_vector(((width/2)+1) downto 0);

begin

process(Clock_as,A_in1,A_in2,B_in1,B_in2,B_in3,control_op)

begin

if(Clock_as'event and Clock_as = '1')then

if (reset_addsub = '1') then

R3 <= (others => '0');

elsif (enable_as = '1') then

if (control_op = '1') then

R3 <= (A_in1 & A_in2) + (B_in1 & B_in2 & B_in3);

elsif (control_op = '0') then

R3 <= (A_in1 & A_in2) - (B_in1 & B_in2 & B_in3);

end if; end if; end if; end process;

S_out <= R3;

end Behavioral;

```

A.6. Counter

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

-----

entity count_reg is

    Port ( clock_count : in  STD_LOGIC;

          reset_count  : in  STD_LOGIC;

          enable_count  : in  STD_LOGIC;

          count_i       : out STD_LOGIC);

end count_reg;

architecture Behavioral of count_reg is

    signal count : std_logic_vector(3 downto 0);

    constant termcount : std_logic_vector(3 downto 0):="0000";

begin

    process(clock_count)

    begin

        if(clock_count = '1' and clock_count'event) then

            if reset_count = '1' then

                count <= "1000";

            elsif enable_count = '1' then

                count <= count-'1';

            end if; end if; end process;

            count_i <= '1' when count = termcount else '0';end Behavioral;
```

A.7. Output Register

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

-----

entity Output_reg is

generic(width:integer:= 16);

Port ( in_reg : in  STD_LOGIC_VECTOR (((width/2)-1) downto 0);

       out_reg : out STD_LOGIC_VECTOR (((width/2)-1) downto 0);

       clk_reg : in  STD_LOGIC;

       enable_reg : in STD_LOGIC;

       rst_reg : in  STD_LOGIC);

end Output_reg;

architecture Behavioral of Output_reg is

begin

process (clk_reg)

begin

if (clk_reg = '1' and clk_reg'EVENT) then

if (rst_reg = '1') then

out_reg <= (others => '0');

elsif (enable_reg = '1') then

out_reg <= in_reg;

end if; end if; end process; end Behavioral;
```

APPENDIX B

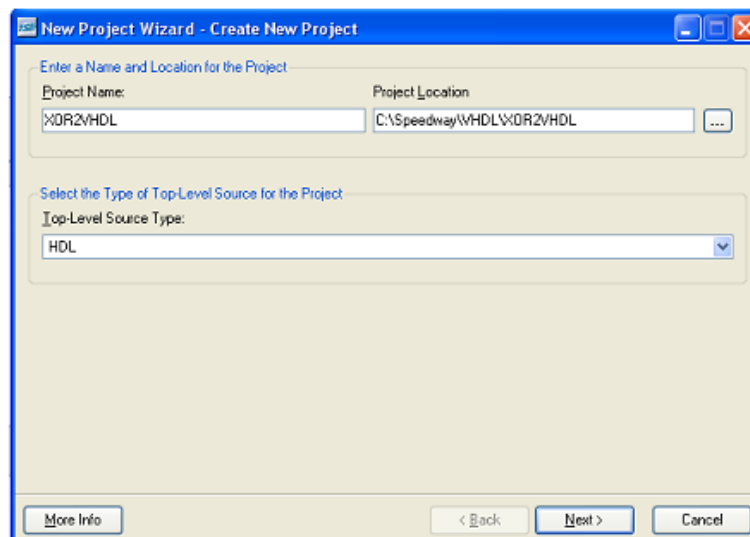
ISE SOFTWARE TUTORIAL LAB

B.1 Lab 1

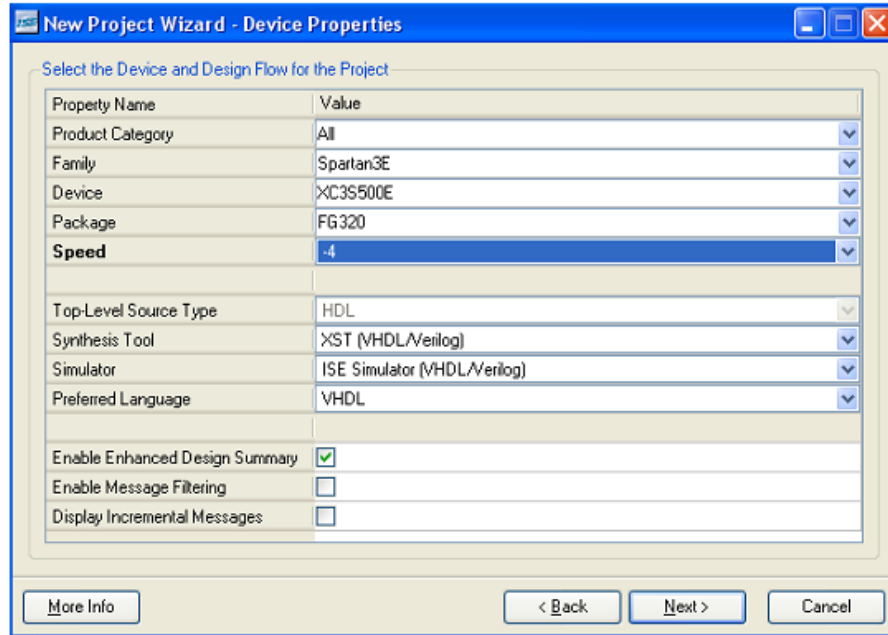
PROCEDURE

Create a new project

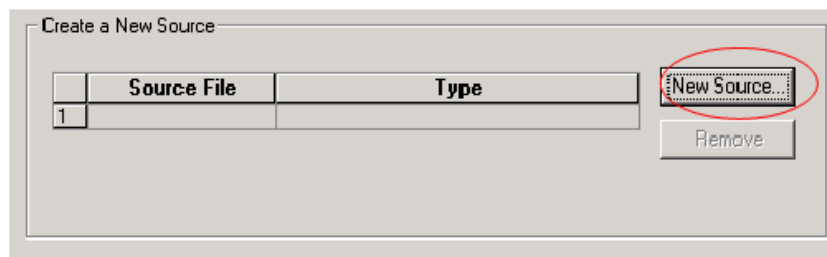
1. Open the Xilinx Foundation ISE design tools from **Start>Programs>Xilinx ISE 9.2>Project Navigator**
2. Create a New Project from the Project Navigator's menu. **FILE ->New Project**
3. Fill in the New Project Dialogue box as follows: Note: To set the Project Location field to C:\Speedway\VHDL\XOR2VHDL; navigate to the C:\Speedway\VHDL directory in the Project Location and the enter XOR2VHDL in the Project Name field.



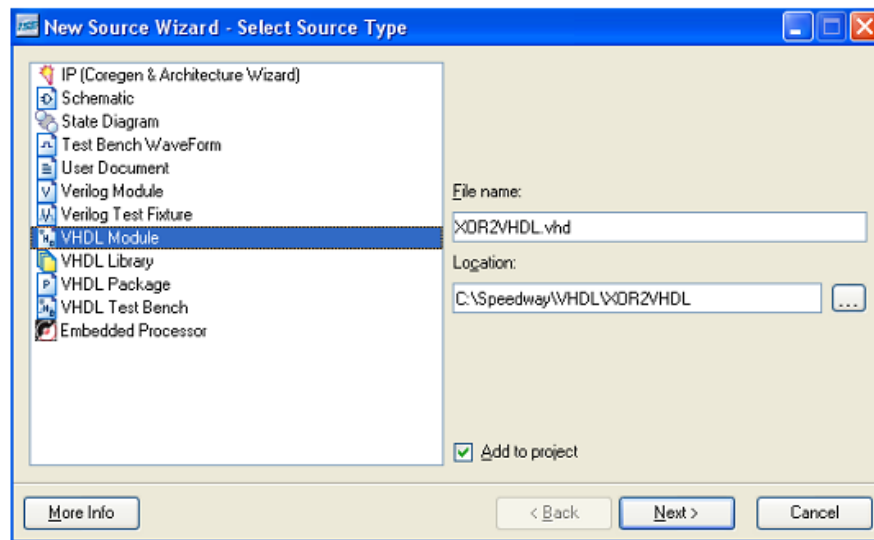
4. Enter the project options:



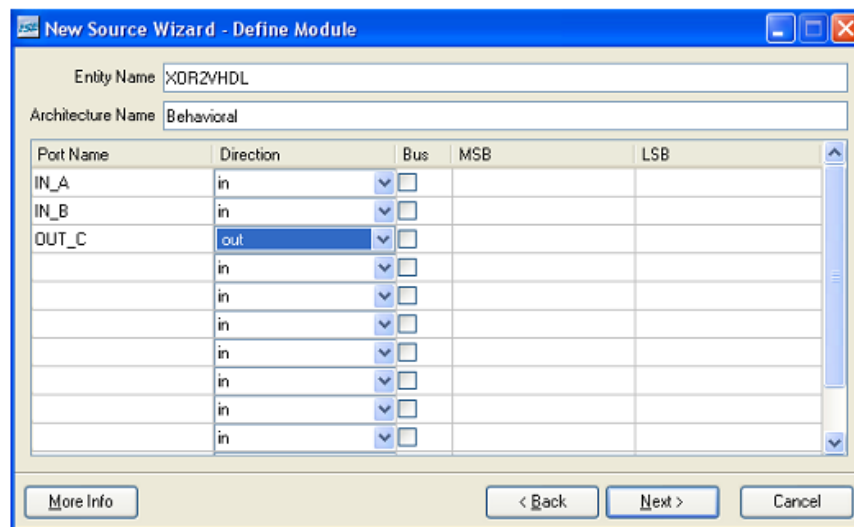
5. Create a new source by selecting the **New Source** button.



6. Fill in the New source window as follows



7. Select **Next** button
 8. Fill in the **Define VHDL Source** window as follows to set up the I/Os.

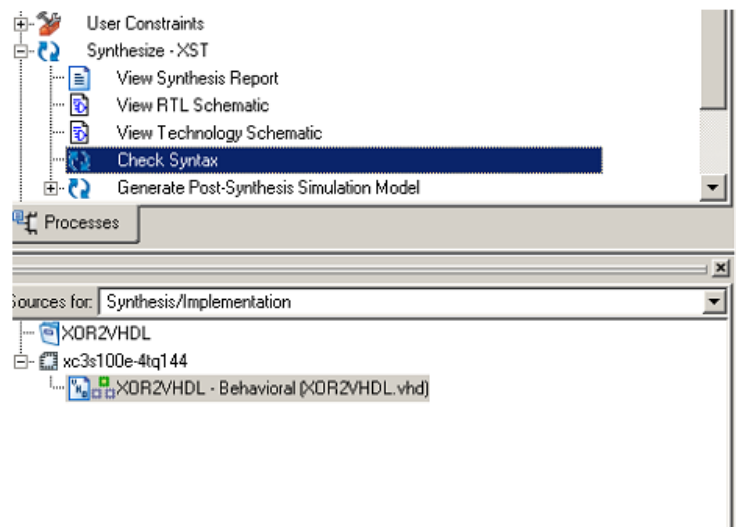



9. Select the **Next** button.
 10. Click **Finish** -> **Yes** -> **Next** -> **Next** -> **Finish**.
 11. A summary window opens.

12. Click next or OK until the form completes. Do not add any other files. This will bring up ISE with the HDL Editor open ready to edit the created VHDL file.
13. Add the VHDL code for the 2-input XOR function within the Architecture.
(Hint: `OUT_C <= IN_A XOR IN_B;`)
14. Save the file by selecting **FILE -> SAVE** or by clicking on the floppy disk icon on the tool bar

Check the Syntax

15. Make sure the file XOR2VHDL is selected in the **Sources in Project** window.
16. Check that Sources for **Synthesis/Implementation** is selected.
17. In the Processes for Current Source window expand the **Synthesize-XST** process and double click **Check Syntax**.



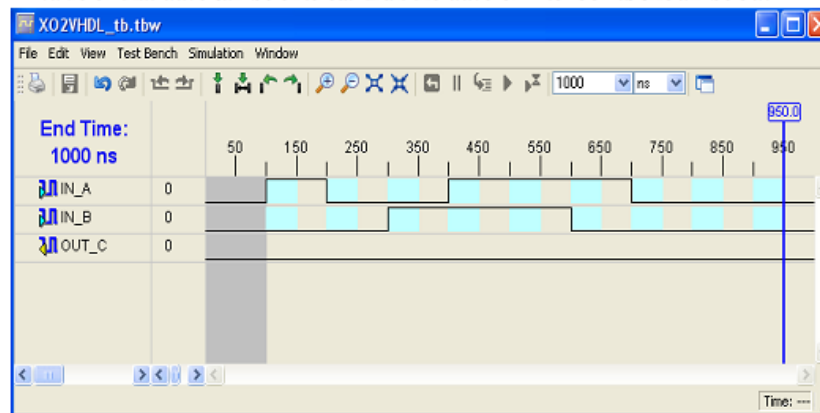
18. If the syntax check fails fix any errors before proceeding. Syntax errors can be located by checking the Project Navigator console window on the bottom of the screen. They are flagged with an  **ERROR:** marker.. Clicking on the hypertext of the file name will take you to the error.

Using the ISE Simulator

ISE Simulator is a design simulation tool included in with the Xilinx tools that allows a user to generate and simulate HDL test benches using a graphical representation of the input and expected output behavior of the code.

19. In the **Sources for** window switch to Behavioral Simulation

20. Double click **Create New Source** in the **Process** window. From the **New Source** window select **Test Bench Waveform** and name the file **XOR2VHDL_tb**.
21. In the **Associate with Source** window select **xor2vhdl** and click **Next**
22. HDL Benchmer will open with the initialize timing dialog window. Make sure that the **“Combinatorial Design (Or internal clock)”** box is selected. Accept the default values and click **Finish** to close the dialog box.
23. The HDL Benchmer waveform editor will open. The waveform editor allows the user to shape input waveforms at each time step. By clicking on a wave at a particular time the value of that wave can be entered. Edit the waveform to look as shown below.



24. Save the wave form and close the editor.
25. The waveform should be visible as the top file in the Behavioral simulation view of the design.
26. Select the testbench wave form in the **Sources For Behavioral Simulation** window. It might be necessary to set the **Behavioral Simulation** in the **Sources for** pull down box.
27. Launch a simulation by clicking the **“+”** tab next to Xilinx ISE Simulator in the process window and double clicking **“Simulate Behavioral Model”**.
28. The ISE Simulator should start up and run the simulation. In the ISE console window the message should appear **“*** Failure: Simulation successful (not a failure). No problems detected.”** Indicating that the simulation was successful.
29. Examine the waveform, does the output match what is expected from an XOR gate?
30. Exit **Project Navigator** and click **Yes** to close the simulation
31. End Lab 1

B.2 Lab 2

VHDL MUX lab

Overview: This lab will illustrate using a variety of VHDL statements to build a basic circuit.

Objective: To code and successfully simulate a 4 to 1 bus multiplexer using any valid construct discussed in Section 2 of the instruction. The multiplexer has four data bus inputs of eight bits each and a 2 bit select line that controls the 8-bit output of the multiplexer. A test bench that has already been created will be used to confirm the operation of the MUX.

PROCEDURE

Open an existing project

1. Open the Xilinx Foundation ISE design tools from Start>Programs>Xilinx ISE 9.2>Project Navigator
2. Open the project **Mux4x8**. From the Project Navigator menu select **File>Open Project** and navigate to the directory **C:\Speedway\VHDL\mux4x8**. Double click on the file **mux4x8.isc** to open the project.

Create a new design file

3. Right click in the **Sources In Project** window. Select **New Source** from the drop down menu.
4. Select **VHDL Module** from the New Sources menu.
5. Enter **mux4x8** in the **File Name** field. Click **Next**.
6. Fill out the **Define VHDL Source** window to match the figure below. Notice that all of the signals have a bus width. To enter a bus click the **Bus** select box then enter the width in the **MSB** field of that signal
7. Verify all signals entered have direction **IN** except for the signal **MUX_OUT** whose direction is **OUT** as shown in following figure. It is very important that the names be **EXACTLY** the same.

New Source Wizard - Define Module

Entity Name: mux4x8

Architecture Name: Behavioral

Port Name	Direction	Bus	MSB	LSB
in_a	in	<input checked="" type="checkbox"/>	7	0
in_b	in	<input checked="" type="checkbox"/>	7	0
in_c	in	<input checked="" type="checkbox"/>	7	0
in_d	in	<input checked="" type="checkbox"/>	7	0
sel	in	<input checked="" type="checkbox"/>	1	0
mux_out	out	<input checked="" type="checkbox"/>	7	0
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		

More Info < Back Next > Cancel

- Once the form has been filled out click **Next**, then **Finish** completing the setup and starting the VHDL editor.

Write VHDL to construct the MUX

- Using any of the methods outlined in the slides fill in the architecture to make a 4 to 1 mux and save the file.
- Use the syntax checker in the **Processes for Current Source** window to verify that the syntax is correct.
- Correct all syntax errors before continuing.

Simulate the design

- Check to insure that the Sources view is set to Behavioral Simulation. The sources should show the existing test bench MUX4X8_tb at the top of the hierarchy with your design file below it.
- Select the test bench file MUX4X8_tb in the Sources view. The Process window should now have the Xilinx ISE Simulator visible. Expand the Xilinx ISE Simulator icon by clicking on the "+" sign. The choice Simulate Behavioral Model should be available. Select Simulate Behavioral Model to launch the ISE simulator.
- The ISE simulator should run to completion. A message similar to:


```
** Failure:Simulation successful (not a failure). No problems detected.
User(VHDL) Code Called Simulation Stop
```

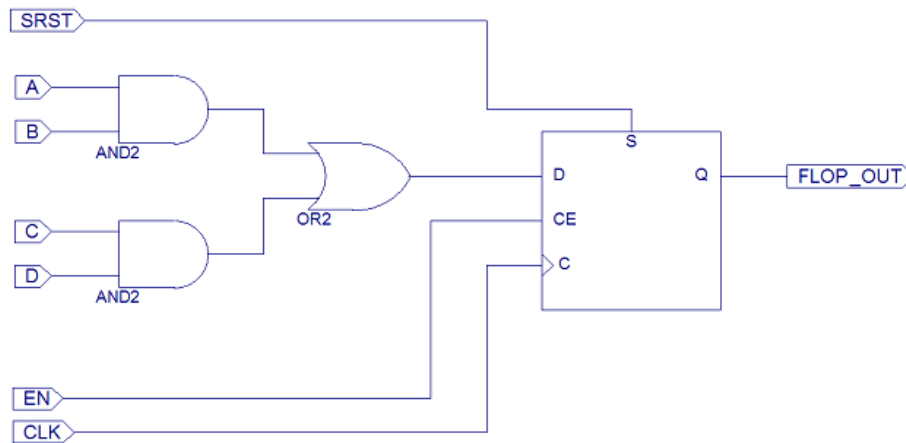
 should appear in the ISE console window. If the simulation failed for other reasons then correct the VHDL source code to get a successful simulation.
- Once the simulation is successful close the ISE simulator.

B.3 Lab 3

VHDL Flip-Flop lab

Overview: This lab will illustrate the use of signal definitions for VHDL coding.

Objective: To code and successfully simulate a flip-flop whose input is fed from combinatorial logic as shown by the schematic below. A test bench that has already been created will be used to confirm the operation of the circuit.

**PROCEDURE***Open the existing project*

1. Open the Xilinx Foundation ISE design tools from **Start>Programs>Xilinx Foundation Series ISE 9.2>Project Navigator**
2. Open the project **Flop_Lab**. From the Project Navigator menu select **File>Open Project** and navigate to the directory **C:\Speedway\VHDL\Flop_lab**. Double click on the file **Flop_lab.ise** to open the project. The project should open with the test bench already added but no actual design entered. It is the purpose of the lab to enter the design.

Create a new design file

3. Right click on the **Sources In Project** window. Select **New Source** from the drop down menu.
4. Select **VHDL Module** from the New Sources menu.
5. Enter **Flop_Lab** in the **File Name** field and click **Next**.
6. Fill out the **Define VHDL Source** to match the table below. Note that signals will have to be entered into the actual VHDL code because the define source routine only creates ports not signals.

Port Name	Direction	Description
A	In	Logic input
B	In	Logic input
C	In	Logic input
D	In	Logic input
EN	In	A Clock enable. When EN='1' the circuit can clock. When EN='0' clocking is disabled.
CLK	In	The system clock, active edge is rising.
SRST	In	A synchronous reset. When SRST='1' FLOP_OUT = '0' when the circuit is clocked. This control overrides the clock enable.
FLOP_OUT	Out	The circuit output.

7. Once the form has been filled out click **Next** then **Finish** to complete the setup and start the VHDL editor.

Write VHDL to construct the circuit

8. To code the circuit you will need at least one concurrent statement for the combinatorial logic and one process for the register.
9. Use the syntax checker in the **Processes for Current Source** window to verify that the syntax is correct.
10. Correct all syntax errors before continuing.

Simulate the design

11. Select the file **FLOP_LAB_TB.VHD** in the **Source in Project** window.
12. Expand the icon for **Xilinx ISE Simulator** in the **Processes for Current Source** window.
13. Select **Simulate Behavioral Model** and double click on it to start the simulation
14. The ISE simulator should run to completion. A message similar to:

“ Failure: Simulation successful (not a failure). No problems detected.

Time: 380 ns Iteration: 0 Instance: /testbench”

should appear in the ISIM console window. If the simulation failed for other reasons then correct the VHDL source code to get a successful simulation.

15. You may zoom to view all the simulation time in the WAVE window of ISIM by selecting the **View=>Zoom In and Out** menu and then **Zoom Full**.
16. Once the simulation is successful close the ISE simulator and Project Navigator.

B.4 Lab 4

VHDL Counter lab

Overview: This lab will illustrate the use of the language assistant, an additional tool available with the Xilinx tools that provides language templates for proper usage and syntax of HDL constructs.

Objective: Create and verify the correct operation of a loadable six-bit up counter with a clock enable and a terminal count. The counter has the following ports. A test bench that has already been created will be used to verify the operation of the circuit.

Port Name	Direction	Description
CLK	In	System clock
CE	In	Clock enable input. Active LOW ! When CE = 1 the counter remains in it's present state. When CE = 0 the counter increments or loads data. CE has no control of SRST.
L	In	Load input. When L=1 and CE=0 the counter loads the value presented at its DATA input. When L=0 and EN=0 the counter increments.
SRST	In	Synchronous reset input. Active High. Counter reset to "000000". Overrides the state of CE and L.
DATA	In	6 bit Data input to load the counter with a value.
COUNT_OUT	Out	The counter's output value. 6 bits. The count out should normally count up from 0 to "111011" and then return to 0.
TC	Out	Terminal Count output. TC should be '1' when COUNT_OUT = "111011" otherwise it should remain at '0'. TC is a combinatorial signal and should not be registered



PROCEDURE*Open an existing project*

1. Open the Xilinx Foundation ISE design tools from Start>Programs>Xilinx Foundation Series ISE 9.2i>Project Navigator
2. Open the project **Counter_Lab**. From the Project Navigator menu select **File>Open Project** and navigate to the directory **C:\Speedway\VHDL\counter_lab**. Double click one the file **counter_lab.ise** to open the project.

Open the design File

3. The design comes with the TestBench and a skeleton of the design file started. It is the goal of the lab to complete the design file, to meet the function described in the Objective section
4. Open the design file called counter.vhd by double clicking on it in the Sources Project Window.

Write VHDL using the Language assistant

5. A pre-made counter will be used as the start of the design
6. Click on the Language Assistant button along the top menu bar 
7. Highlight the template VHDL\Synthesis Constructs\Coding Examples\Counters\Binary\Up Counters
8. Locate the counter that most **closely** matches the requirement and select it in the language assistant.
9. Right click the  icon in the language assistant to add the **process** skeleton to the HDL code.
10. Complete the rest of the HDL to make the needed counter circuit.

A few things to notice

 - a. No Template counter has a TC so you will have to add one. Don't forget that TC also clears the counter.
 - b. You will need to make a local copy of COUNT_OUT.
11. Use the syntax checker in the Processes for Current Source window to verify that the syntax is correct.
12. Correct all syntax errors before continuing. While debugging the code you might get a console message to "Turn off Incremental Compilation". This can be down by selecting the testbench, "counter_ATB," in the sources window and then right clicking on Simulate Behavioral Model in the Processes window. Select Properties and uncheck the Incremental Compilation button.
13. Simulate the design, Select the file COUNTER_ATB.VHD in the Source in Project window
14. It might be necessary to type "run all " to force the simulator to run to completion.
15. Select Simulate Behavioral VHDL model. The ISE simulator should run to completion. A message similar to:


```
"Failure: Simulation successful (not a failure). No problems detected.
# Time: 1060 ns Iteration: 0 Instance: /testbench"
```

 should appear in the ISE console window.
16. If the simulation failed for other reasons then correct the VHDL source code to get a successful simulation.
17. Once the simulation is successful close the ISE simulator and Project Navigator

B.5 Lab 5

LAB 5*VHDL State Machine Lab*

Overview: To design and simulate a state machine using VHDL.

Objective: To encode a 5 state machine and verify its correct operation. The finite state machine should be coded to have its ports function as described below and to transition through the states as described in the bubble state diagram below. A test bench has already been created and will be used to verify the correct operation of the circuit.

Port Name	Direction	Description
CLK	In	System clock input
EN	In	Clock Enable input. When EN=0 the machine remains at it's present state. When EN=1 the state machine advances to the next state.
SRST	In	Synchronous Reset input. Active high
ColorOne, ColorTwo	In	The control inputs to the state machine
RED, GREEN, BLUE, YELLOW, START	Out	The state machine outputs

PROCEDURE***Open an existing project***

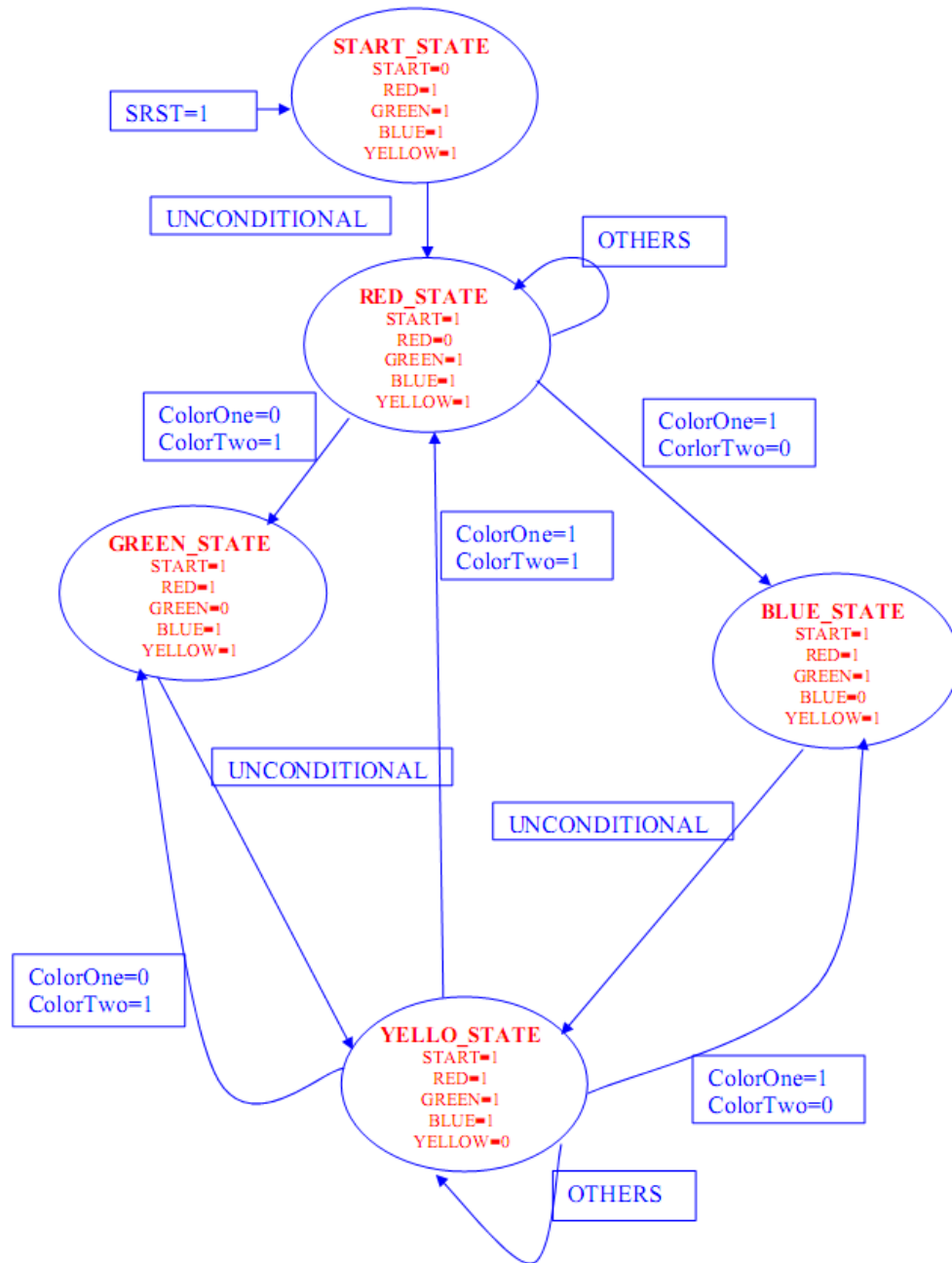
1. Open the Xilinx Foundation ISE design tools from Start>Programs>Xilinx Foundation Series ISE 9.2i>Project Navigator
2. Open the project C:\Speedway\VHDL\STATE\STATE.isc

Open the design file for editing

3. The design file is state.vhd
4. Fill out the Define VHDL Source window to match the given ports.

Write VHDL to construct the circuit

5. Reference the following state diagram. Complete the rest of the HDL to make the needed counter circuit.



6. Use the syntax checker in the **Processes for Current Source** window to verify that the syntax is correct.
7. Correct all syntax errors before continuing.

Simulate the design and access additional waveforms

8. Select the file **STATETB.VHD** in the **Source in Project** window.
9. Select **Simulate Behavioral Model**.
10. The ISE simulator should run, it might be necessary to type “run all” in the console window to complete the simulation.:

Failure: Simulation successful (not a failure). No problems detected.

Time: 1760 ns Iteration: 0 Instance: /testbench”

should appear in the ISE console window.

11. If the simulation failed for other reasons correct the VHDL source code to get a successful simulation.
12. A good technique to debugging the circuit is to add the PRESENT state signal to the simulation view. Lower level signals can be added to the simulation view by expanding the hierarchy of the test bench to open the UUT. Once the UUT signals are visible the signal PresentState can be dragged to the waveform window.
13. It might be necessary to restart the simulator and rerun to see all of PresentState. This can be done by typing “restart” and “run all” in the simulator console window.
14. Once the simulation is successful close the simulator.

B.6 Lab 6

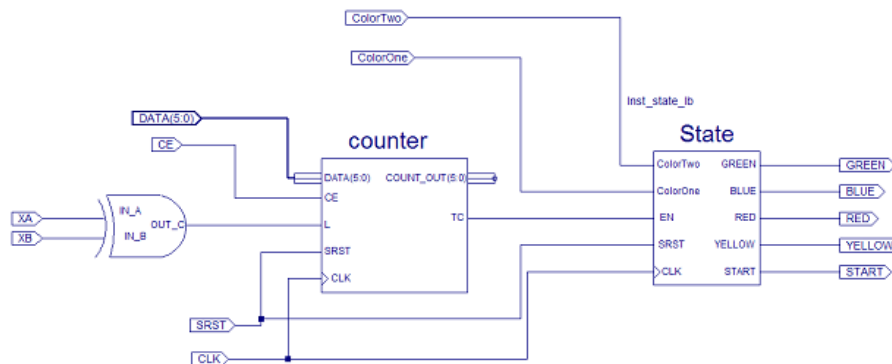
LAB 6*VHDL Port Map Lab*

Overview: This lab will illustrate hierarchical VHDL design methodology covering instantiation and packages.

Objective: To create a top level VHDL file that completes a hierarchical design. The design uses a package that contains all the component declarations; this package must be completed as part of the lab. The design instantiates the state machine used in lab 5 along with a counter and the XOR design created in during lab1. The project has been started for the student and working versions of the counter and state machine are available.

PROCEDURE*What is in the Project*

This project is partially complete; it contains a working version of the counter needed for this design COUNT.vhd file, a working XOR gate design, and the state machine from Lab 5. It also contains a package called My_pack. My_Pack is not complete; it is missing the **component** entry for the State Machine and the XOR2VHD. To help visualize the design a schematic is supplied below.

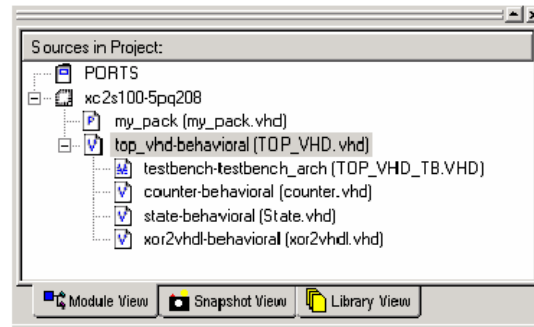
*Open an existing Project*

1. From the File menu in the Project Navigator tool bar select Open Project...
2. Open the project named C:\Speedway\VHDL\Ports\Ports.isc

Complete the design

3. Open the file **My_Pack.vhd** (Note: it might be necessary to search through the hierarchy of the design to find My_Pack, also one can find My_Pack by selecting the libraries tab in the sources window on opening the library **work**) and add component statements for the two missing components **State_1b** and **XOR2VHD** to be instantiated in the top-level design. Use the **View HDL Instantiation Template** feature of ISE to generate the component statement.
4. Notice that the terminal count for the COUNT circuit is being set up in My_Pack by defining the constant **t_count**.
5. Save and close My_Pack.
6. Create a new VHDL module using the Project Navigators **Project>New Source** option.
7. The file should be called "**TOP_VHD**" and should have the following ports.

Inputs	CLK, SRST, XA, XB, CE, Data(5:0), ColorOne, ColorTwo
Outputs	GREEN, BLUE, RED, YELLOW, START
8. Write the VHDL to correctly describe the design shown below. Make sure to include a call to use My_Pack library. Notice that a few signals will have to be added to the top level design to transport data from one instantiated entity to the next. Also notice that the counter port **C_OUT** is not connected. This is ok, however to suppress any warning statements for synthesis tools a connection can be made to the VHDL keyword **open** in the counters **port map** statement.
9. **Hint: Design Entry Utilities=>View VHDL Instantiation Template.** This could be helpful.
10. Select **TOP_VHD** in the Sources and when the file passes a syntax check save the file.
11. The ISE Project Navigator should now show the full hierarchy of the design in the Source in Project window as shown below.



Verify the design using the ISE Simulator

12. **BEFORE SIMULATING** Check that Incremental Compilation is turned off. Select the test bench then right click Simulate Behavioral Model and select Properties.
13. Make sure the **Display Level** is set to **Advanced**.
14. Select the testbench and double click on the Simulate Behavioral VHDL Model in the Processes for Current Source window.
15. It may be necessary to force the simulator to run for a longer time to get the completion message. Enter the command **>run ALL** to force the simulator to run to completion.
16. The simulation should complete successfully. If it failed, return to the source code and make any corrections necessary.