HIGH PRECISION COMPUTATION OF DECIMAL LOGARITHM

by-

Riaz-ul-Haque Mian

MASTER OF ENGINEERING

IN

INFORMATION AND COMMUNICATION TECHNOLOGY

Institute of Information and Communication Technology

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

2017

The project titled “High Precision Computation of Decimal Logarithms” submitted by Riaz-ul-haque Mian, Roll No. 0411312027, and Session April-2011 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Master of Engineering in Information and Communication Technology (ICT) at 29/04/2017

Candidate’s Declaration

It is hereby declared that this project or any part of it has not been submitted elsewhere for the award of any degree or diploma.

____

Riaz-ul-haque Mian

Dedicated

to

My beloved son and his mother

Contents

TABLE OF FIGURE

LIST OF TABLE

LIST OF ABBREVIATIONS OF TECHNICAL SYMBOLS AND TERMS

ASIC	Application Specific Integrated Circuit
CLB	Configurable Logic Block
DPA	Differential Power Analysis
DSP	Digital Signal Processor
e.g.	For example
EDA	Electronic Design Automation
Eq.	Equation
FIPS	Federal Information Processing Standards
FIPS PUB	Federal Information Processing Standards Publications
FPGA	Field Programmable Gate Array
FPL	Field Programmable Logic
Gbps	Giga bits per second
HDL	Hardware Description Language
ISCAS	International Symposium of Circuits And Systems
LAB	Logic Array Block – a physically grouped set of logic cells
LC	Logic Cell
LPM	Library of Parameterized Module
LUT	Look-Up Table
M4K	Memory block of 4096 bits
M512	Memory block of 512 bits
M-RAM	Mega RAM
NIST	National Institute of Standards and Technology
PDA	Personal Digital Assistant
PLD	Programmable Logic Device
RAM	Random Access Memory
ROM	Read Only memory
VHDL	Very High Speed Integrated Circuit Hardware Description Language
Word	A group of 32 bits

Acknowledgement

During the time I was performing my master’s project, I came across many people who have supported and assisted me. First of all, I would like to thank the almighty Allah for giving me the opportunity to conduct this project. I would like to express my heartiest thanks to my supervisor, Professor Dr. Md. Liakot Ali, for giving me the opportunity to do my master’s project under his supervision. I am also grateful to him for all his support, advice and encouragement throughout this project.

I gratefully acknowledge the valuable advice and support from Professor and Director Dr. Md Saiful Islam, and all other faculty members of IICT, BUET.

I also express my gratitude to Bangladesh University of Engineering and Technology for allowing me to conduct the research project using its all kinds of facilities.

Finally, I want to thank all my friends and family members for their continuous support and inspirations, making this work a nice experience.

Abstract

This report presents the design of an efficient low power decimal logarithmic converter. The technique is based on shift operation along with a precision unit which is used to prevent unnecessary scan of full memory table. Precision level is user programmable. It does not require multiplication or division circuitry which in turn makes it low power. Proposed decimal logarithmic converter has been designed using Verilog HDL and then compiled and simulated using Altera provided Quartus II compiler and Modelsim simulator. The synthesis results show that the proposed design outperforms all the existing proposed decimal logarithmic converters by other researchers.

The calculation is based on memory. Some predefined value and there log value are stored in memory and by using that value we calculate the logarithm of any value. Another feature which is called precision unit is used for controlling unnecessary scan of memory table. According to input of precision value the algorithm calculates logarithm value. Thus by increasing of precision value we can increase accuracy of the result.

This algorithm does not need multiplication circuitry and required very low memory thus it makes the algorithm very low power.

Synthesis results show that the proposed technique costs 6002 logic element and consuming power only 49.37mW. Although the logic element is little bit higher but the power is extremely lower than those of proposed by other researchers. The logarithmic converter can be used in many embedded applications where low power is a concern.

Chapter 1

Introduction

1.1 Introduction

In all computer system, elementary functions like logarithm plays a vital role. It has many applications, both within and outside mathematics. It is very useful in many applications such as tax calculations, financial analysis, internet based applications etc. [1]. Because the logarithmic function log(x) grows very slowly for large x, logarithmic scales are used to compress large-scale scientific data [1-2]. Logarithms also occur in numerous scientific formulas, such as the Tsiolkovsky rocket equation, the Fenske equation, or the Nernst equation. It is due to the fact that there are many applications which needs direct computation of decimal logarithm, such as measurement of P^H, earthquake intensity, optical density in spectrometry and optics, brightness of stars in astronomy, ratio of voltage and power (called bel) in telecommunications, electronics and acoustics [1]. Researches have been conducted to find radix-10 logarithmic converters where decimal input is first converted to binary and then base-2 logarithm operations are performed and again results are converted back to decimal radix [3]. It causes errors due to the back and forth conversations of the bases. To overcome this problem, a lot of researches are going on for developing algorithms in various technique [2-3] iterative technique, using multiplication [3-5], using lookup table [6-8]. Recently, IEEE754-1985 binary standard has been revised to IEEE754-2008 standard where decimal floating point operation is included because it is more human friendly [9]. There are some other research regarding decimal multiplication [10-16] where multiplication approach is used and thereby power consumption is very high.

In all proposed algorithm there is a common scope of improvement which is power consumption. In every proposed algorithm either there is a power consuming multiplication circuitry or there is large memory size which is also power consuming. Thus if we calculate logarithm without large memory and/or multiplication circuit the power consumption will automatically reduce. In proposed algorithm multiplication circuit is fully avoid and also use very few memory which make the algorithm low power.

The research was supposed to conduct for developing 64 bit high precision high decimal logarithmic converter but during the course of this research it is already proposed by other researcher which in turn necessitate another scope for computing decimal logarithm is explored low power computation of decimal logarithm. It is because low power has emerged as a principal theme in today's electronics industry. The need for low power has caused a major paradigm shift where power dissipation has become as important a consideration as performance and area [17]. In the past, the major concerns of the ASIC designer were area, performance, cost and reliability; power consideration was mostly of only secondary importance. In recent years, however, this has begun to change and, increasingly, power is being given comparable weight to area and speed considerations. [3][5][6][17]

1.2 Objectives

The aim of this project is to develop a low power decimal logarithmic converter. To fulfill this aim, we have the following objectives:

- To develop an algorithm for computing base-10 logarithm where low power is a concern
- To design the logarithmic converter using Verilog HDL
- To simulate design in FPGA environment

1.3 Organization of the Project

Chapter 1 of this report starts with the importance of elementary function followed by a brief background of latest research work in this area.

Chapter 2 of this report describes the details of Decimal Logarithm and its needs, IEEE Standards floating point number systems.

Chapter 3 of this report describes main Algorithm with Pseudo-code, flowchart and Complete Example with simulation result it also describe Precision unit with example. All other modules are also described in this chapter. Simulation using compiler (Quartus II) and Simulator (ModelSim)of decimal logarithm technique step by step with proper example are also describe in this chapter

Chapter 4 Describes results and performance analysis of proposed logarithmic converter with various aspects.

Chapter 5 of this report describes conclusion and future work of the research.

Chapter 2

Decimal Logarithmic Technique

2.1 Introduction

This chapter describes foundations and principles of decimal logarithm. The common logarithm is the logarithm with base 10. It is also known as the decimal logarithm, named after its base. The needs of decimal logarithm will be a part of this chapter. This chapter also introduce with floating point number as proposed algorithm of calculating decimal logarithm is design for floating point number. FPGA (Field-programmable gate array) and its advantages will be also presented at end of this chapter.

2.2 Decimal logarithm

The common logarithm is the logarithm with base 10. It is also known as the decimal logarithm, named after its base. The fractional part of a logarithm is usually written as a decimal. The whole number part of a logarithm and its decimal part have been given separate names because each plays a special part in relation to the number which the logarithm represents. The whole number part of a logarithm is called the CHARACTERISTIC. This part of the logarithm shows the position of the decimal point in the associated number. The decimal part of a logarithm is called the MANTISSA.
For a particular sequence of digits making up a number, the mantissa of a common logarithm is always the same regardless of the position of the decimal point in that number. For example, log 5270 = 3.72181; the mantissa is 0.72181 and the characteristic is 3.

In mathematics, the common logarithm is the logarithm with base 10. It is also known the decimal logarithm, named after its base, an English mathematician who pioneered its use, as well as "standard logarithm". It is indicated by $\log_{10}(x)$, or sometimes Log(x) with a capital L [15].

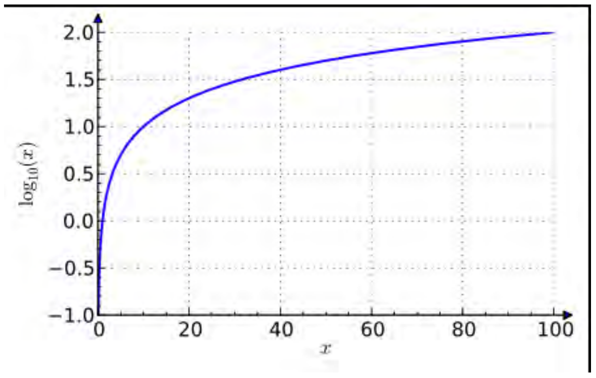


Figure 1 : A graph of the common logarithm function for positive real numbers

Figure 1shows the value of $\text{Log}_{10}(x)$ is slowly increases with the increased value of x. It shows that if X =100i.e.10^2 then $\text{Log}_{10}(x) = \text{Log}_{10}(100) = 2$. Similarly if x= 10000000 i.e. 10^7then $\text{Log}_{10}(x) = \text{Log}_{10}(10000000) = 7$thus it is clear from logarithm scale that it can use to compress large-scale scientific data .

An important property of base-10 logarithm which makes it so useful in calculation is that the logarithms of numbers greater than one which differ by a factor of a power of ten all have the same fractional part. The fractional part is known as the mantissa. The integer part, called the characteristic, can be computed by simply counting how many places the decimal point must be moved so that it is just to the right of the first significant digit. For example, the logarithm of 120 is given by:

$$\log_{10} 120 = \log_{10}(10^{\,2} * 1.2) = 2 + \log_{10} 1.2 \;\approx 2 + .07928$$

The last number (0.07918)—the fractional part or the mantissa of the common logarithm of 120. The location of the decimal point in 120 tells us that the integer part of the common logarithm of 120, the characteristic, is 2.

Numbers greater than 0 and less than 1 have negative logarithms. For example,

$$\log_{10}. 120 = \log_{10}(10 - {}^{2} * 1.2) = -2 + \log_{10} 1.2 \;\approx -2 + .07928$$

An important feature of logarithms is that they reduce multiplication to addition, by the formula:

$$\log(xy) = \log x + \log y.$$

That is, the logarithm of the product of two numbers is the sum of the logarithms of those numbers.

Similarly, logarithms reduce division to subtraction by the formula:

$$\log\left(\frac{x}{y}\right) = \log x - \log y.$$

2.3 Needs of Decimal Logarithm

Decimal Logarithm plays a vital role in finance and astronomy. Logarithms are used in measuring many physical quantities. The scale that is used to measure earthquakes, the Richter scale, involves a logarithm. Likewise the scale that is used to measure the loudness of sound in decibels involves a logarithm. They are often used in studying population growth and radioactive decay. Logarithms also arise in some financial calculations, for example those involving interest rates. Logarithms are useful in solving equations in which exponents are unknown. They have simple derivatives, so they are often used in the solution of integrals. In hyperbolic geometry the logarithm is used in measuring the distances between points

To convert multiplications to additions, Logarithms can be useful. For example, calculating loan with continuous interest and calculating how long it will take to pay it back, you need to use logarithms.

Anything with exponential growth, like biologists studying populations, physicists studying nuclear reactions, chemists studying chain reactions; or a banker calculating how long it will take for investments to reach certain levels at given interest rates, its need to use logarithms.

In physiology looking at how the eye reacts to light or the ear to sound, both are so-called "logarithmic devices", i.e., if someone double the light or the sound, the reaction of the eye or ear is to signal not a double strength, but rather an increase by a fixed step. That way anyone can see light or hear sound in a huge range without having a complex signaling system.

Electrical engineers use logarithms to work on signal decay. Computer scientists looking at how fast programs run with respect to the size of the inputs use logarithms often. For example, most good sorting programs take time proportional to n*log(n) to sort a list of n items.

2.4 Floating-Point Arithmetic

IEEE Standard for Floating-Point Arithmetic (IEEE 754) is the most widely-used standard for floating-point computation, and is followed by many hardware (CPU- Central Processing Units and FPU- floating point unit) and software implementations. Many computer languages allow or require that some or all arithmetic be carried out using IEEE 754 formats and operations. The current version is IEEE 754-2008, which was published in August 2008; it includes nearly all of the original IEEE 754-1985 (which was published in 1985) and the IEEE Standard for Radix-Independent Floating-Point Arithmetic (IEEE 854-1987).

For example in Bangladeshi currency 100 paisa makes one taka thus the presentation of Taka in number is fixed point like 2.50 means 2 taka 50 paisa. Here position of decimal point is fixed and always after two digit form right side which is known as fixed point number. One the other hand pi = 3.14159265... e = 2.71828 which are real number where position of decimal point is not fixed form its right most position, Computer arithmetic that supports such numbers is called Floating Point.
A detail section regarding floating point standard and arithmetic has been discussed in annexure 4.

2.6 FPGA

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, partial re-configuration of the portion of the design and the low non-recurring engineering costs relative to an ASIC offer advantages for many applications.

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like a one-chip programmable breadboard. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like ANDandXOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

FPGA is a semiconductor device containing programmable logic components and programmable interconnects. It contains up to thousands of gates. The programmable logic components can be programmed to duplicate the functionality of basic logic gates such as AND, OR, XOR, NOT or more complex combinational functions such as decoders or simple math functions. In most FPGAs, these programmable logic components (or logic blocks, in FPGA parlance) also include memory elements, which may be simple flip-flops or more complete blocks of memories. These logic blocks and interconnects can be programmed after the manufacturing process by the customer/designer (hence the term "field programmable", i.e. programmable in the field) so that the FPGA can perform whatever logical function is needed.
There are various vendor manufacturers for different types of FPGA chip such as Altera, Xilinx, Lattice Semiconductor, Actel, Quick Logic, Cypress Semiconductor, Atmel, Achronix Semiconductor etc. Among them Altera and Xilinx are the most famous FPGA companies since both of the companies have lot of varieties of FPGA device from small number of gate counts to higher number of gate counts. However Altera devices offer the general benefits of PLDs as innovative architectures, advanced process technologies, state-of-the-art development tools, and a wide selection of mega function. The common advantages of Altera devices include: High performance, High-density logic integration, Cost-effectiveness, Short development cycles with the Quartus II software, Mega Core functions, Benefits of in-system programming. In this work the FPGA device used is Altera provided EP2C35F672C6 from Cyclone II family.

2.5 Advantages of FPGAs

The main advantage is everything is programmable in FPGA. Here we collect 20 top advantages form various source form internet (books and website) including Altera website.

1. It is possible to customize the process fully without wasting the resources as what we do in controllers; also that FPGA has ability to carry out very big and complex process.
2. FPGA is a customized IC. It can implement most digital logic. BUT CPU perform an operation by instructions. FPGA is more powerful. For instance, you can implement a PCI bridge by FPGA . CPU cannot.
3. FPGA is excellent to implement the glue logic of the system of different chips. It really glues all of them together.
4. FPGAs can be very powerful, much more powerful than any microcontroller.
5. The real advantage of FPGA's are the ease of prototyping, time to market and no NRE and Low volume to use.
6. A micro-controller won't have enough processing power in most tele-com applications, especially the data path. In those applications, one needs direct hardwired logic to process the packets. The only choices are FPGA or ASIC
7. It is reconfigurable and strongly flexible. The limitation of microcontroller is not existed.
8. It is possible to make a processor a special purpose processor not a general purpose processor. Also it can operate at very high clock speeds than to controllers.
9. It is not necessary to think about layout design, it is enough to repair the code only.
10. It is a matter of time: when the micro is not fast enough, we have to go with hardware. ie super-fast filtering, sub-microsecond timing, fast counters, video signals. And here comes the FPGA's: hardware that we can configure to do exactly what we need.
11. FPGA can be field programmed. But if the power is turn off, the logic will be lost. If we need change some logic in our product we can easily change it.
12. We can work with micro's in any applications, it is true. But mcu cannot meet to everything. Many field need high speed parallel processing.
13. The project on high speed communication protocol can't be implemented only using

microcontroller. In such case, FPGA and ASIC are needed.

14. FPGA is parallel processing. So it is widely used in high-speed and real-time processing field. That means speed can be very fast, and multiple control loops can run on a single FPGA device at different rates.
15. Speed. Parallel operations. Flexibility. Those are just several of many advantages.
16. FPGA design tools are increasingly available, allowing embedded control system designers to more quickly create and adapt FPGA hardware.
17. Because the processing paths are parallel, different operations do not have to compete for the same processing resources.
18. The configurability of FPGAs can provide designers with almost limitless flexibility.
19. In manufacturing and automation contexts, FPGAs are well-suited for use in robotics and machine tool applications, as well as for fan, pump, compressor and conveyor control.
20. Unlike processors, FPGAs use dedicated hardware for processing logic and do not have an operating system.

Chapter 3

Design, Implementation and simulation of the Algorithm

3.1 Introduction

In previous chapter the brief description of the logarithm technique is provided. In this chapter, design procedure of the proposed technique for calculating Decimal Log value will be described. Functional block diagram, detail discussion of each individual module with a flowchart and complete mathematical example will also be described in this chapter. Finally complete simulation process using various tools with proper examples and discussion regarding various tools with their using procedure will also describe at the end of the chapter.

3.2 Functional Block Diagram

A functional block diagram of decimal logarithmic converter is shown in figure 2. The main module is controller which is responsible for all arithmetic calculation. It picks necessary value from memory. Control unit maintains user defined precision level during scanning of memory table for calculating logarithm value. Control unit is also responsible for managing input/output and does necessary shift and counting operation.

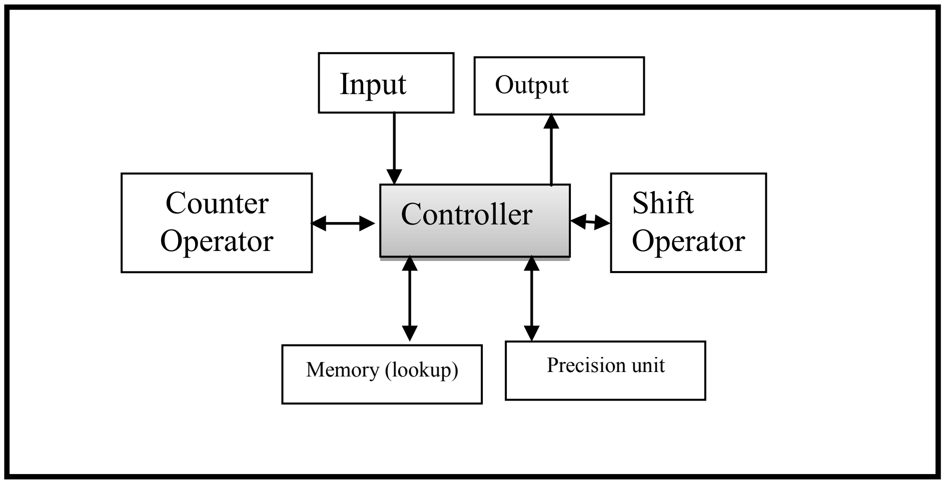


Figure 2: Basic Functional Block Diagram

Short description of each individual module is given bellow:

1. **Input Module**

The main task of this module is storing user provided value into memory. This unit contains three registers. One is for which value we want to calculate logarithm, as we calculate floating point number we must have to have a decimal point locator and another is precision value. The name and size of register is shown in bellow table:

Table 1: Input Unit Variable

SL	Name	Type	Size (bit)
1	Data	input	[31:0]
2	D_Point	input	[4:0]
3	Precision	input	[4:0]

2. **Shift Operator Module**

This operator is used for multiplication and division operation. In this algorithm we avoid multiplication and do all multiplication operation by shift operation. To know how we perform multiplication and division operation by using shift operation consider the bellow example :

0000111 = 7 in decimal. Now implement <<1 (i.e. left shift by 1digit) the digit became 0001110 = 14 now if we again do <<2 (i.e. left shift by 2 digit) then the digit become 0011100 = $7*2^2$ = 28 now similarly once again if we do <<2 (i.e. left shift by 2 digit) the binary number became 0111000 = 7*2*2*2 = 28*2 = 56

Now it is clear that we can do any multiplication with number 2,4,16,256 etc. by shift operation .

Now consider reversely

0111000= 56 in decimal now implement >>2 (i.e. right shift by 2digit) then it became 0001110 = $(56/2^2)$ =14 = 0001110 again if we right shift by 1digit then it became 0001110>>1 = 0000111= 14/2 = 7. This is the way to avoid multiplication and division operation through shift operation

3. Memory Module

Most important unit that basically stores 32 bit predefined logarithm value by using which we can calculate any log value. It it's a register array of 32 bit. Each array element contains a log value. A sample of the memory unit is shown by the table 2 bellow

Table: 2 Memory Table of logarithm value

K	**65536**	**256**	**16**	**4**	**2**	**3/2**	**5/4**	**9/8**	**17/16**
$\text{Log}_{10}(\text{K})$	**4.81647993**	2.4082	1.2041	.6020	.30102	0.1760	0.0969	0.0511	0.0263

We stored in memory table at binary format. Consider $\log_{10}(2)$ = 0.30102999566398119521373889472449. If we take first 10 digit then the digit became $\log_{10}(2)$ = 0.3010299957 here we make last digit 7 as because of the immediately right digit after 10th digit is higher than 5. Now if we convert the digit to binary we get 1011 0011 0110 1101 1000 1000 0011 0101 in memory. This is the way we store log value in memory.

4. Precision Unit

This strategy makes system computation faster. We make a precision unit for avoiding unnecessary scan of all memory Array. Detail discussions with example of precision unit are shown in section 3.6. Precision unit compare user provided precision value with zero and maintains down counter. After each iteration the precision value is decreased by one. Once the value is equal to zero, we stop fetching memory table from start to end thus we did not require scan all lookup table. Figure 3 bellow shows the precision unit flow diagram.

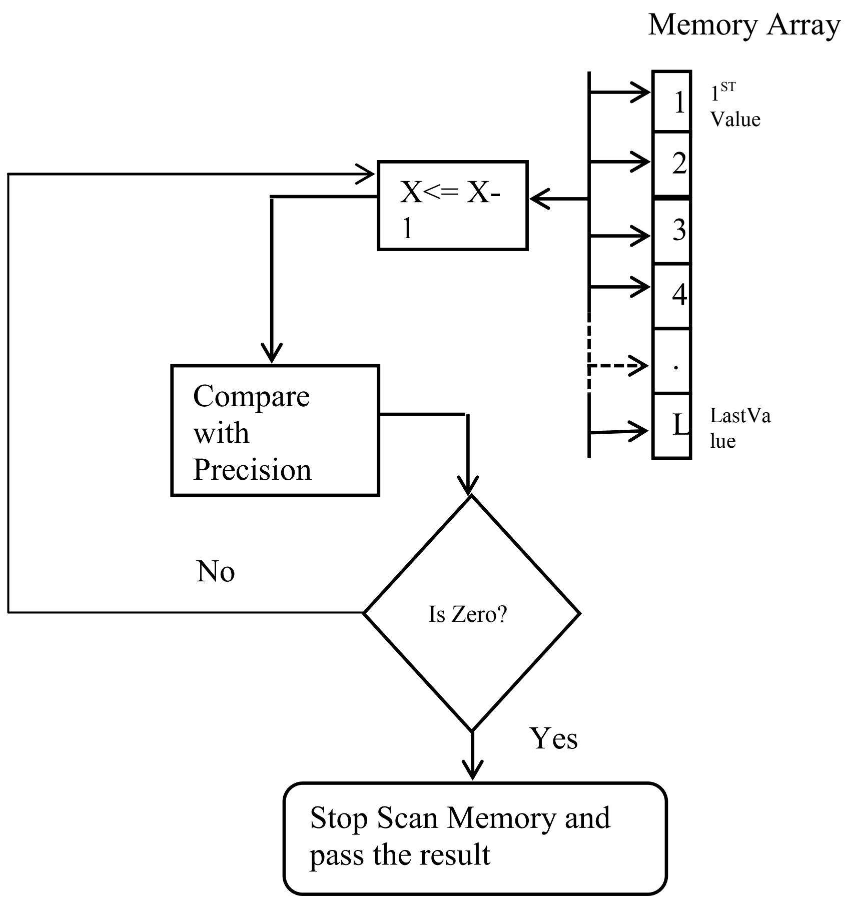


Figure 3 : Precision Unit Operation

5. Controller Module

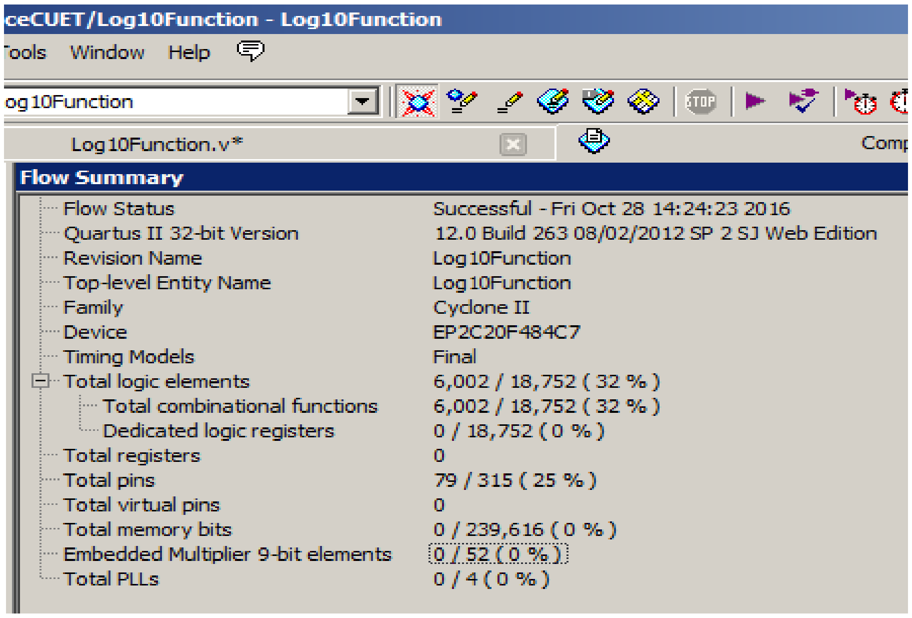
Main module that basically connects with all other units/modules is Controller module. Like CPU control module takes value form input and calculate logarithm value using other modules and finally sends result in output module.

6. Counter Module

This module is used by precision unit. The main component of the module is a down counter. When user put precision value through input unit it reduces the precision value by 1 at each iteration. Once the value becomes zero it stopped the iteration and the result is passed through output.

3.3 Summary of Compiling Result

The overall project design, developed and compile in Altera provided Quarts II compiler and using EP2C20F484C7 device from Cyclone II family. After compiling successfully the compilation result is shown in figure 4



Flow Summary

```
Flow Status                          Successful - Fri Oct 28 14:24:23 2016
Quartus II 32-bit Version            12.0 Build 263 08/02/2012 SP 2 SJ Web Edition
Revision Name                        Log10Function
Top-level Entity Name                Log10Function
Family                               Cyclone II
Device                               EP2C20F484C7
Timing Models                        Final
Total logic elements                 6,002 / 18,752 ( 32 % )
    Total combinational functions    6,002 / 18,752 ( 32 % )
    Dedicated logic registers        0 / 18,752 ( 0 % )
Total registers                      0
Total pins                           79 / 315 ( 25 % )
Total virtual pins                   0
Total memory bits                    0 / 239,616 ( 0 % )
Embedded Multiplier 9-bit elements   0 / 52 ( 0 % )
Total PLLs                           0 / 4 ( 0 % )
```

Figure 4 Compilation Result in QUARTUS II

3.4 Pseudo-code with flowchart

In this algorithm we focus high speed and low power. Accuracy depends on precision unit. It can be increased as expected. Precision unit prevents unnecessary scan to Memory. This algorithm is perfect where power, time and hardware cost is essential rather precision.

Table 3: Memory table (Predefine K value &$\log_{10}$(K) value which is used for calculation)

K	65536	256	16	4	2	3/2	5/4	9/8	17/16
$\log_{10}$(K)	**4.81647993**	2.4082	1.2041	.6020	.30102	0.1760	0.0969	0.0511	0.0263

Flowchart

The pseudo-code of the proposed algorithm is summarized below and illustrated in Fig. 5

1. Read input of decimal number, decimal point position of that number and precision level from user.

2. Initialize X, DP, Y and Precision register. X, DP = Input given by user; Y= 0; Precision = value assign by user.

3. The algorithm generates a sequence of values for X and Y.

Condition

i) For first time, as all the K values in the memory are greater than 1. [K value is predefine log value of some number which is used to calculate log value see memory table above see Table 3]

ii) We will have to start with X less than 1, Thus we need a value of K which immediately larger then X

iii) Begin to multiply X by 1/K which we called (**Kmax**) and, to maintain the function, keeping log(Kmax)= (predefine stored value in memory) to Y. This is only for initial X and Y register assign value for calculation.

4. The algorithm now check the maximum value of K which is called **Kpx**. **Kpx** is the biggest K we can multiply by X and the result is below 1 (keeping X<1).

5. Once we get the value of X using **Kpx** now we need to fix corresponding Y value form the memory by Y= Y-log(Kpx). The value of K and corresponding log(K) is already stored in memory if we know the value of K we can easily extract the value of log(K) thus we can calculate the Y value by using Y=Y-log(K).

6. Each iteration Precision unit decreases the value by one. This process is preventing further scanning memory table.

7. Compare the value of precision with 0 if it is equal to zero the result is Y otherwise step 4, 5 and 6 will repeat until the value of precision unit zero. End

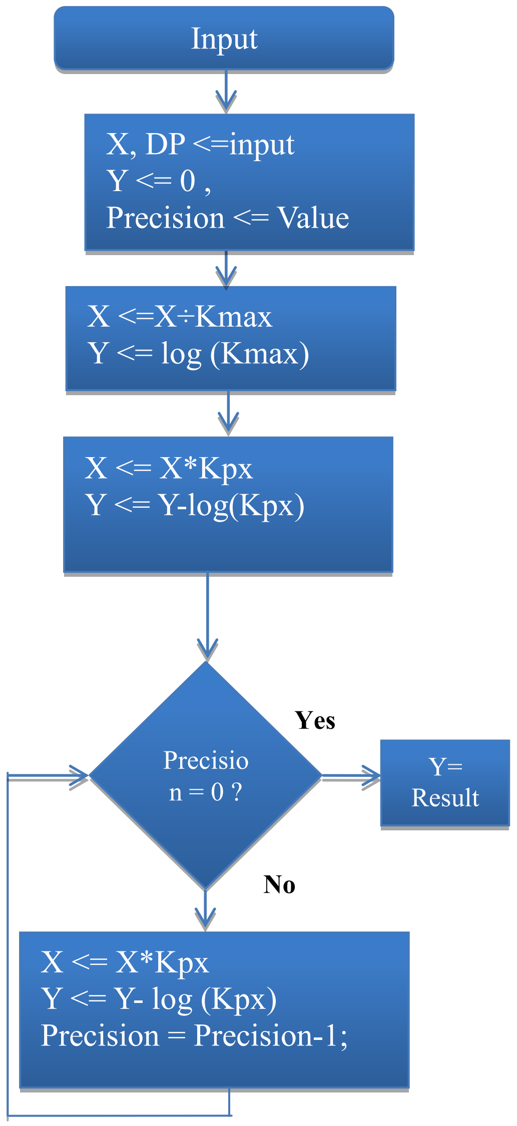


Figure 5 : Flowchart for logarithm calculation

3.5 Example Step by Step with Simulation

In the proposed algorithm logarithm calculation follows three basic steps.

1. Input the decimal floating point number and precision level.
2. Iterative calculation according to algorithm up to precision level
3. Output the result of decimal logarithm

Let's explore the algorithm with a very easy example, First we store decimal logarithm value of some standard number starting from 65536 and corresponding log value like table 4 below:

Table 4: Memory table (Predefine K value & $\log_{10}$ (K) value which is used for calculation)

K	**~~65536~~**	**256**	**~~16~~**	**4**	**2**	**~~3/2~~**	**~~5/4~~**	**~~9/8~~**	**~~17/16~~**	**33/32**	**.**
Log_{10}(K)	**~~4.8164799~~**	2.4082	~~1.204~~	~~.602~~	.30102	~~0.1760~~	~~0.0969~~	~~0.0511~~	~~0.0263~~	0.0133	.

The value 65536 is square of 256 and also 256 is square of 16 similarly 16 is square of 4 and 4 is square of 2. We can start with value form 4,294,967,296 which is square of 65536 in calculation , that case we can calculate logarithm value of any number which is smaller than 4,294,967,296 so the range it is dependable. If we start form 65536 then we can calculate any log value which is smaller than 65536.

Now we calculate Y= log(X) and to explore the example consider the value of X=123 then the algorithm technique generates a sequence of values for X and Y.
Initial value according to algorithm is as follows:

Name of Register	Value	Comment
X (Input)	123	We wants to calculate the $\log_{10}$(123)
DP(Decimal point)	NA	As we calculate integer value we does not require decimal point location
Y(result)	0	Initially result is assign to zero
P (Precision value)	3	This value is set by user for accuracy level

All the stored K values in the memory table are greater than 1 (see table 4) bellow. We will have to start with X less than 1, so we begin to divide X by 256 and, to maintain the function, put the $\log_{10}(256) = 2.4082$ to Y; this is really just a scaling of the input and an initialization of Y.

Table 4: Memory table (Predefine K value & $\log_{10}$ (K) value which is used for calculation)

K	~~**65536**~~	**256**	~~**16**~~	**4**	**2**	~~**3/2**~~	~~**5/4**~~	~~**9/8**~~	~~**17/16**~~	**33/32**	
Log$_{10}$(K)	~~**4.81647993**~~	2.4082	~~1.2041~~	~~.6020~~	.30102	~~0.1760~~	~~0.0969~~	~~0.0511~~	~~0.0263~~	0.0133	

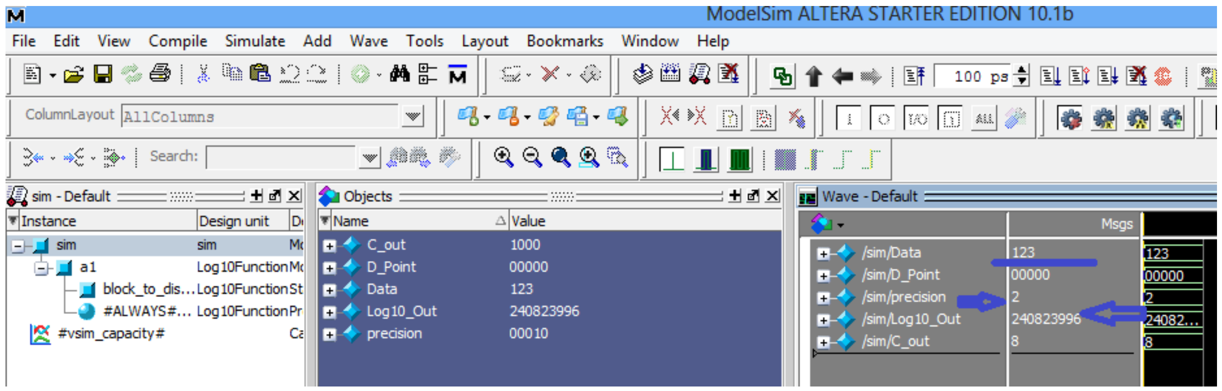
After this preparatory step the value of all register is as follows in table 5

Table 5: After first step (division) the register value

Name of Register	Value	Comment
X (input)	123/256 = .4805	The immediate higher value of 123 in memory table is 256. According to algorithm we need to divide by 256
DP(decimal point)	NA	As we calculate integer value we does not require decimal point location
Y(result)	Log(256) = 2.4082	We assign Y = $\log_{10}$(k) where k = 256 and log (256) = 2.4082
P (precision value)	3	This value is remain same as iterative calculation is not start yet

Here note that the input 123 is smaller than both 65536 and 256, as 123 is nearest smaller than 256 thus according to algorithm we need to divide by 256.

In this state if we simulate, the following output (figure 6) will show according to algorithm logic


ModelSim ALTERA STARTER EDITION 10.1b
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
ColumnLayout AllColumns
Search:
sim - Default
Instance Design unit
sim sim
a1 Log10FunctionMc
block_to_dis... Log10FunctionSt
#ALWAYS#... Log10FunctionPr
#vsim_capacity#
Objects
Wave - Default
Msgs

Name	Value
C_out	1000
D_Point	00000
Data	123
Log10_Out	240823996
precision	00010

	Msgs	
/sim/Data	123	123
/sim/D_Point	00000	00000
/sim/precision	2	2
/sim/Log10_Out	240823996	24082...
/sim/C_out	8	8

Figure 6 : Calculating log value (simulator result) step1

Step 2: Algorithm now start iterative calculation with multiplication. Now X=0.4805, the maximum K we can multiply by X is 2 (keeping X<1) if we multiply with next bigger number which is 4 then the value of X (0.4805) is become 0.4805 *4 = 1.922 which is greater than 1 , in case of 16 also the value of X will be greater than 1 thus we need to select 2 from memory table for multiply and then the value of X will 0.4805 *2 = .961 then we will calculate Y by to subtracting Log value of K with Y . Here log value of K i.e. 2 is 0.3010 need to subtract form Y. In this way 16 and 4 are not use in memory table which is yellow background color and ~~strikethrough~~. Now the new value of X became 0.961 and Y will 2.4082-0.3010 = 2.1672.

In this step we decrease precision value by 1 which is provide by user and say it is 3 then in this stage precision value will 2.

Table 4: Memory table (Predefine K value & $\log_{10}$ (K) value which is used for calculation)

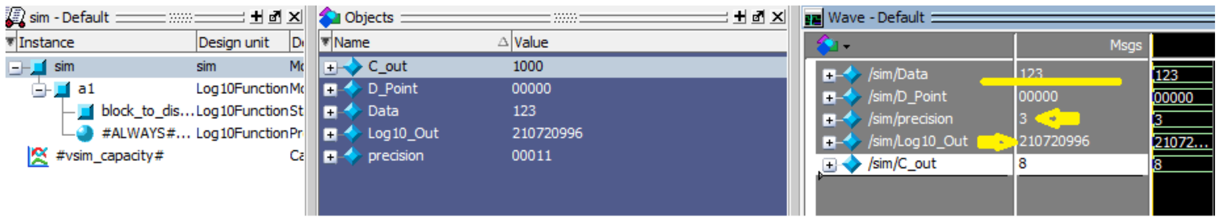
K	~~65536~~	256	~~16~~	4	2	~~3/2~~	~~5/4~~	~~9/8~~	~~17/16~~	33/32	
Log_{10}(K)	~~4.81647993~~	2.4082	~~1.2041~~	~~.6020~~	.30102	~~0.1760~~	~~0.0969~~	~~0.0511~~	~~0.0263~~	0.0133	

Summary of calculation is shown in bellow table 6

Table 6: After second step (multiplication) the register value

Name of Register	Value	Comment
X (input)	1. 123/256 = .4805 2. .4805 * 2 = .961	The highest value of K that we can multiply with X by keeping result bellow one is 2
DP(decimal point)	NA	As we calculate integer value we does not require decimal point location
Y(result)	1. Log(256) = 2.4082 2. 2.4082 – log(2) = 2.4082 - 0.3010 = 2.1672	We assign Y = Y- $\log_{10}$(k) where k = 2 and $\log_{10}$(2) = . 3010
P (precision value)	3-1 = 2	The value of precision will reduce by 1

In this state simulator result show as follows in figure 7


```
sim - Default
Instance              Design unit        D
sim                   sim                M
  a1                  Log10FunctionM
    block_to_dis...   Log10FunctionSt
    #ALWAYS#...       Log10FunctionPr
  #vsim_capacity#                        Ca

Objects
Name           Value
C_out          1000
D_Point        00000
Data           123
Log10_Out      210720996
precision      00011

Wave - Default
                    Msgs
/sim/Data           123          123
/sim/D_Point        00000        00000
/sim/precision      3            3
/sim/Log10_Out      210720996    21072...
/sim/C_out          8            8
```

Figure 7 : Simulator result in step 2

Step 3: Step 3 is repetitive step of 2. Similarly as step 2 by keeping X value bellow 1 the biggest K in memory is 33/32 then the value of X= .961 *33/32=0.9916 and Y = 2.1672-.0133=2.0931. Precision unit again decrease precision value by 1 and it will 2-1 = 1.

Table 4: Memory table (Predefine K value & $\log_{10}$ (K) value which is used for calculation)

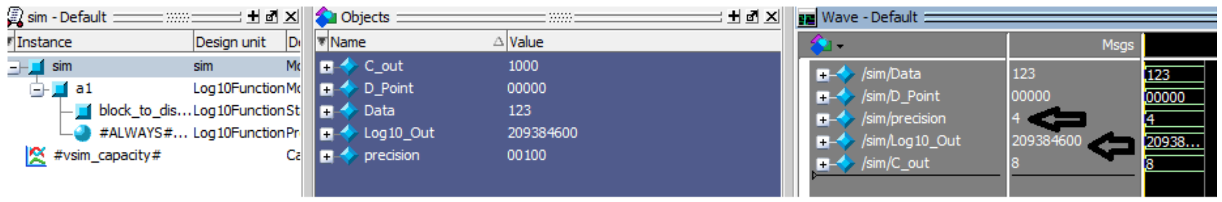
K	~~**65536**~~	**256**	~~**16**~~	**4**	**2**	~~**3/2**~~	~~**5/4**~~	~~**9/8**~~	~~**17/16**~~	**33/32**	
Log_{10}(K)	~~**4.81647993**~~	2.4082	~~1.2041~~	~~.6020~~	.30102	~~0.1760~~	~~0.0969~~	~~0.0511~~	~~0.0263~~	0.0133	

Summary of calculation is shown in bellow table 7

Table 7: After third step (multiplication) the register value

Name of Register	Value	Comment
X (input)	1. 123/256 = .4805 2. .4805 * 2 = .961 3. .961 *33/32=0.9916	The highest value of K that we can multiply with X by keeping result bellow one is 33/32
DP(decimal point)	NA	As we calculate integer value we does not require decimal point location
Y(result)	**1.** Log(256) = 2.4082 **2.** 2.4082 – log(2) = 2.4082 - 0.3010 = 2.1672 **3.** 2.1672 – log (33/32) = .996 - -.0133 =2.0931	We assign Y = Y- $\log_{10}$(k) where k = 33/32
P (precision value)	2-1 = 1	The value of precision will reduce by 1

In this stage simulator result is as follows:



```
sim - Default
Instance                Design unit       D
sim                     sim               M
  a1                    Log10Function     M
    block_to_dis...     Log10Function     St
    #ALWAYS#...         Log10Function     Pr
  #vsim_capacity#                         Ca

Objects
Name          Value
C_out         1000
D_Point       00000
Data          123
Log10_Out     209384600
precision     00100

Wave - Default
                     Msgs
/sim/Data            123          123
/sim/D_Point         00000        00000
/sim/precision       4            4
/sim/Log10_Out       209384600    20938...
/sim/C_out           8            8
```

Figure 8 : simulator result at step 3

Step 4: We will continue until precision value became zero. To do that by keeping X value bellow 1 we can multiply X by (128/127) and then the value of X will (.9925) and also Y= 2.0904 and the value of precision unit is now 0. Thus the result is 2.0904 which can also make more accurate by running the loop more by increasing precision value more than 3.

Now although the memory table may have 100 or more number of K value but as precision unit stop iteration and we will get the result only after 4 steps. By this way the system works very faster in low precision.

Table 4: Memory table (Predefine K value & $\log_{10}$ (K) value which is used for calculation)

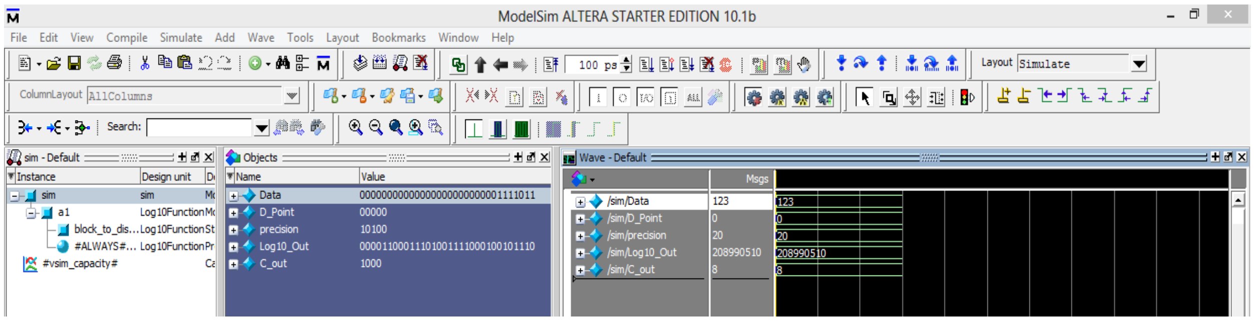
K	~~**65536**~~	**256**	~~**16**~~	**4**	**2**	~~**3/2**~~	~~**5/4**~~	~~**9/8**~~	~~**17/16**~~	**33/32**	~~**65/64**~~	**128/127**
Log_{10}(K)	~~**4.81647**~~	2.4082	~~1.2041~~	~~.6020~~	.301	~~0.1760~~	~~0.0969~~	~~0.0511~~	~~0.0263~~	0.0133	~~.006733~~	.0034

Summary of calculation is shown in bellow table 8

Table 8: After forth step (multiplication) the register value

Name of Register	Value	Comment
X (input)	1. 123/256 = .4805 2. .4805 * 2 = .961 3. .961 *33/32=0.9916 4. .9916 * 128/127	The highest value of K that we can multiply with X by keeping result bellow one is128/127
DP(decimal point)	NA	As we calculate integer value we does not require decimal point location
Y(result)	**1.** Log(256) = 2.4082 **2.** 2.4082 – log(2) = 2.4082 - 0.3010 = 2.1672 **3.** 2.1672 – log (33/32) = 2.1672 -.0133 =2.0931 **4.** 2.0931 – log (128/127) = 2. 2.0931 - .0034 = 2.0904	We assign Y = Y- $\log_{10}$(k) where k = 128/127
P (precision value)	1-1 = 0	The value of precision will reduce by 1

Similarly as we increase the precision value results will more accurate bellow is the result with precision value 20 which is accurate:


ModelSim ALTERA STARTER EDITION 10.1b
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Layout Simulate
ColumnLayout AllColumns
Search:
sim - Default
Objects
Wave - Default

Instance	Design unit
sim	sim
a1	Log10Function
block_to_dis…	Log10Function
#ALWAYS#…	Log10Function
#vsim_capacity#	

Name	Value
Data	00000000000000000000000001111011
D_Point	00000
precision	10100
Log10_Out	00001100011101001111000100101110
C_out	1000

	Msgs	
/sim/Data	123	123
/sim/D_Point	0	0
/sim/precision	20	20
/sim/Log10_Out	208990510	208990510
/sim/C_out	8	8

Figure 9 : Result in simulator (final result) with precision 20

3.6 Example Precision Unit

To understand the activity of the precision unit module here we explore another example.

Let we calculate $\log_{10}(999.99)$ using proposed algorithm .

LogValue = 999999

Decimal Point in log value = 2

Precision is = 3, 4, 5 (to make the example understandable here we consider very low value of precision)

We calculate

a. Log OUT =?

b. Decimal Point in LogOUT value =?

We calculate the value of log with three precision levels 3, 4, and 5

For first value test bench is as follows

```
module sim;
        reg  [31:0]              Data;
        reg  [4:0]               D_Point,precision;
        wire [31:0]              Log10_Out;
        wire [4:0]               C_out;

Log10Function a1(Data, D_Point,precision, Log10_Out, C_out );
initial
begin
 Data = 32'd999999;
D_Point = 5'd2;
precision = 5'd3;

end
endmodule
```

According to our algorithm first we calculate 9999.99/65536 and the result is 0.15259 now the value of X, Y and P is as follows

X= 0.15259

Y = 4.81647993 [$\log_{10}(65536)$ =4.81647993]

And precision value is 3

Now we calculate logarithm and according to algorithm we update the X, Y, Precision register value

So

K_{PX} = 4 because highest value of memory by which we can multiply by keeping X bellow 1 is 4

X= X* K_{PX} = .15259*4 = .61036

Y = Y- $\text{Log}_{10}(K_{PX})$ = 4.81648-.6020 = 4.21448

P = 2 (reduce by one is each cycle)

Now again we calculate logarithm and according to algorithm we update the X, Y, Precision register value

So

K_{PX} = 2 because highest value of memory by which we can multiply by keeping X bellow 1 is 2

X= X* K_{PX} = .15259*2 = .61036

Y = Y- $\text{Log}_{10}(K_{PX})$ = 4.211448 -.3010 = 3.91342

P = 1 (reduce by one is each cycle)

Now again we calculate logarithm and according to algorithm we update the X,Y, Precision register value

So

K_{PX} = 17/16 = 1.0625 because highest value of memory by which we can multiply by keeping X bellow 1 is 1.0625

X= X* K_{PX} = .15259*1.0625 = .972761

Y = Y- $\text{Log}_{10}(K_{PX})$ = .91342 -.0263 = 3.8871

P = 0 (reduce by one is each cycle)

Now for precision 3 the answer is $\log_{10}$(9999.99) = 3.8871 which is close to actual answer 3.9999 again we calculate by considering precision level value is 4

According to our algorithm first we calculate 9999.99/65536 and the result is 0.15259 now the value of X Y and P is as follows

X= 0.15259
Y = 4.81647993 [$\log_{10}$(65536) =4.81647993]
And precision value is 4

Now we calculate logarithm and according to algorithm we update the X, Y, Precision register value
So

K_{PX} = 4 because highest value of memory by which we can multiply by keeping X bellow 1 is 4
X= X* K_{PX} = .15259*4 = .61036
Y = Y- $\text{Log}_{10}(K_{PX})$ = 4.81648-.6020 = 4.21448
P = 3 (reduce by one is each cycle)
Now again we calculate logarithm and according to algorithm we update the X,Y, Precision register value
So
K_{PX} = 2 because highest value of memory by which we can multiply by keeping X bellow 1 is 2
X= X* K_{PX} = .15259*2 = .61036
Y = Y- $\text{Log}_{10}(K_{PX})$ = 4.211448 -.3010 = 3.91342
P = 2 (reduce by one is each cycle)

Now again we calculate logarithm and according to algorithm we update the X,Y, Precision register value

So

K_{PX} = 17/16 = 1.0625 because highest value of memory by which we can multiply by keeping X bellow 1 is 1.0625

X= X* K_{PX} = .15259*1.0625 = .972761

Y = Y- $Log_{10}(K_{PX})$ = 3.91342-.0263 = 3.8871

P = 1 (reduce by one is each cycle)

Now again we calculate logarithm and according to algorithm we update the X,Y, Precision register value

So

K_{PX} = 65/64 = 1.0625 because highest value of memory by which we can multiply by keeping X bellow 1 is 1.01562

X= X* K_{PX} = .972761*1.01562= .9887959

Y = Y- $Log_{10}(K_{PX})$ = 3.8871-.0069 = 3.9065

P = 0 (reduce by one is each cycle)

Now for precision 4 the answer is $\mathbf{log_{10}}$(9999.99) = 3.9065 which is more accurate then precision level 3 i.e. (3.8871) close to actual answer 3.9999 again we calculate by considering precision level value is 4

Again if we compute against precision level 5 the last step will as follows

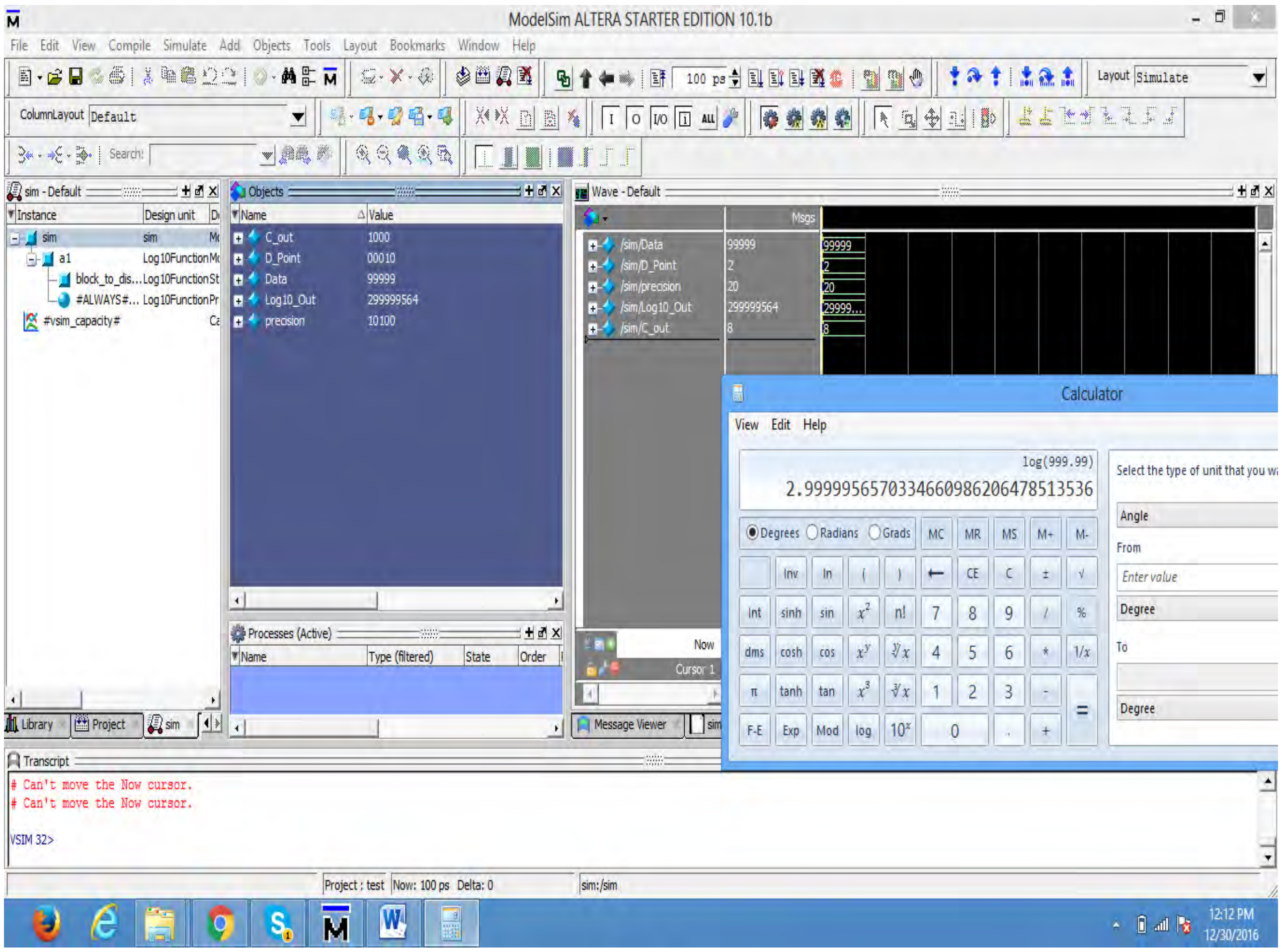
K_{PX} = 129/128 = 1.007815 because highest value of memory by which we can multiply by keeping X bellow 1 is 1.00781

X= X* K_{PX} = .972761*1.01562= .99652

Y = Y- $Log_{10}(K_{PX})$ = 3.9065-.0.0034 = 3.9065

P = 0 (reduce by one is each cycle)

Simulation result after putting precision value 20



Instance	Design unit
sim	sim
a1	Log10FunctionM
block_to_dis...	Log10FunctionSt
#ALWAYS#...	Log10FunctionPr
#vsim_capacity#	

Name	Value
C_out	1000
D_Point	00010
Data	99999
Log10_Out	299999564
precision	10100

	Msgs	
/sim/Data	99999	99999
/sim/D_Point	2	2
/sim/precision	20	20
/sim/Log10_Out	299999564	29999...
/sim/C_out	8	8

```
log(999.99)
2.9999956570334660986206478513536
```

```
# Can't move the Now cursor.
# Can't move the Now cursor.

VSIM 32>
```

Project : test   Now: 100 ps   Delta: 0   sim:/sim

Figure 10 : Result of Algorithm with precision 20

3.7 Simulation and FPGA Implementation

To compile, simulate and implement the algorithm we use various tools are summarized as follows in table 2:

SL	Name/Type	Example
1	Programming HDL language	Verilog
2	Compiler	Quartus II 12.0sp2 Web Edition (32-Bit)
3	Simulator	ModelSim-Altera 10.1b (Quartus II 12.1)
4	Device	Cyclone II
5	FPGA	EP2C35F672C6

Table 2: Summary of tools

Brief description of all tools and software are given bellow

3.7.1 Verilog HDL (Hardware Definition Language)

In the earlier, the conventional approach such as hand-draw and schematic based design technique was the only choice to the designer to design a digital system. But now millions of transistors are being integrated on a single chip integrated circuit (IC) where the conventional design technique is insufficient to be used. It points towards having a new approach for designing today’s complex digital system and that is hardware description language (HDL). HDL based design technique has been emerged as the most efficient solution. It offers the following advantages over conventional based design approaches.

- It is technology independent. If a particular IC fabrication process becomes outdated, it is possible to synthesize a new level design by only changing the technology file but using the same HDL code.
- HDL shortens the design cycle of a chip by efficiently describing and simulating the behavior of the chip. A complex circuit can be designed using a few lines of HDL code.
- It lowers the cost of design of an IC.
- It improves design quality of a chip. Area and timing of the chip can be optimized and analyzed in different stages of design.

There are different types of HDL available in the market. Some of these are vendor dependent where the HDL code is only useable under the software provided by the specific vendor. For example, Altera hardware description language (AHDL) from Altera company, Lola (Logic Language) from European Silicon Structure (ES2) company etc. However Verilog and VHDL (very high speed IC hardware description language) are the two vendor independent HDL which are now widely accepted industry standard electronic design automation (EDA) tool for designing digital system. Verilog HDL is introduced by Cadence Data Systems, Inc. and later its control is transferred to a consortium of companies and universities known as open Verilog international (OVI) whereas VHDL is used primarily by defense contractors. Currently Verilog is widely used by IC designers. Verilog HDL is IEEE standard and easier than VHDL. It is less error prone. It has many pre-defined features very specific to IC design. For this reason Verilog is chosen to design and implement of the proposed system.

3.7.2QuartusII Compiler

Altera Quartus II is programmable logic device design software produced by Altera. Quartus II enables analysis and synthesis of HDL designs, which enables the developer to compile their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Quartus includes an implementation of VHDL and Verilog for hardware description, visual editing of logic circuits, and vector waveform simulation.

Features
Quartus II software features include:
SOPC Builder, a tool in Quartus II software that eliminates manual system integration tasks by automatically generating interconnect logic and creating a testbench to verify functionality

Qsys, a system-integration tool that is the next generation of SOPC Builder. It uses an FPGA-optimized network-on-chip architecture that doubles the fMAX performance vs. SOPC Builder.

SoCEDS, a set of development tools, utility programs, run-time software, and application examples to help you develop software for SoC FPGA embedded systems.

DSP Builder, a tool that creates a seamless bridge between the MATLAB/Simulink tool and Quartus II software, so FPGA designers have the algorithm development, simulation, and verification capabilities of MATLAB/Simulink system-level design tools

External memory interface toolkit, which identifies calibration issues and measures the margins for each DQS signal.
Generation of JAM/STAPL files for JTAG in-circuit device programmers.

Editions

Web Edition
The Web Edition is a free version of Quartus II that can be downloaded or delivered by mail for free. This edition provided compilation and programming for a limited number of Altera devices.

The low-cost Cyclone family of FPGAs is fully supported by this edition, as well as the MAX family of CPLDs, meaning small developers and educational institutions have no overheads from the cost of development software.

License registration is required to use the Web Edition of Quartus II, which is free and can be renewed an unlimited number of times.

Subscription Edition
The Subscription Edition is also available for free download, but a DRMed license must be paid for to use the full functionality in the software. The free Web Edition license can be used on this software, restricting the devices that can be used.

The supported operating systems are:

Microsoft Windows Vista (32-bit and 64-bit)

Microsoft Windows XP (32-bit and 64-bit)

Microsoft Windows 2000

Solaris 8 and 9 (32-bit and 64-bit)

SUSE Linux Enterprise 9 (32-bit and 64-bit)

Red Hat Enterprise Linux 5 (32-bit and 64-bit)

Red Hat Enterprise Linux 4 (32-bit and 64-bit)

Red Hat Enterprise Linux 3 (32-bit and 64-bit)

3.7.3Modelsim Simulator

ModelSim is a multi-language HDL simulation environment by Mentor Graphics, for simulation of hardware description languages such as VHDL, Verilog and SystemC, and includes a built-in C debugger. ModelSim can be used independently, or in conjunction with Altera Quartus or Xilinx ISE.Simulation is performed using the graphical user interface (GUI), or automatically using scripts.

ModelSim is offered in multiple editions, such as ModelSim PE, ModelSim SE, and ModelSimXE.

ModelSim SE offers high-performance and advanced debugging capabilities, while ModelSim PE is the entry-level simulator for hobbyists and students. ModelSim SE is used in large multi-million gate designs, and is supported on Microsoft Windows and Linux, in 32-bit and 64-bit architectures.

ModelSimXE stands for Xilinx Edition, and is specially designed for integration with Xilinx ISE.[5] ModelSimXE enables testing of HDL programs written for Xilinx Virtex/Spartan series FPGA's without needed physical hardware.

ModelSim can also be used with MATLAB/Simulink, using Link for ModelSim. Link for ModelSim is a fast bidirectional co-simulation interface between Simulink and ModelSim. For such designs, MATLAB provides a numerical simulation toolset, while ModelSim provides tools to verify the hardware implementation & timing characteristics of the design.

Language support

ModelSim uses a unified kernel for simulation of all supported languages, and the method of debugging embedded C code is the same as VHDL or Verilog.

ModelSim enables simulation, verification and debugging for the following languages:

- VHDL
- Verilog
- Verilog 2001
- SystemVerilog
- PSL

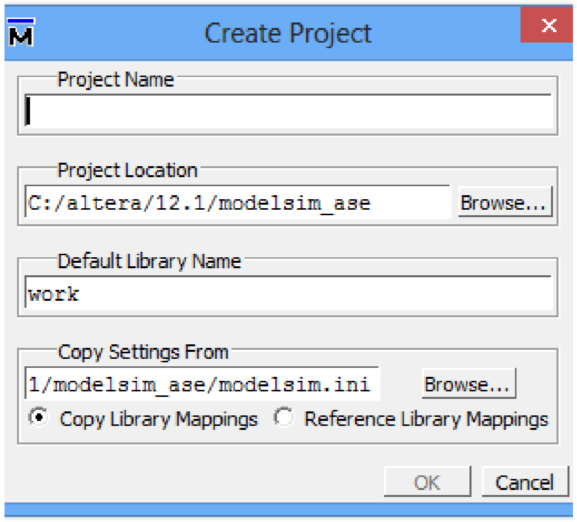
3.7.4Simulation Mode

Simulation allows testing a design thoroughly to ensure that it responds correctly in every possible situation before configuring a device. Depending on the type of information need, functional or timing simulation can be performed with the simulator. Functional simulation tests only the logical operation of a design by simulating the behavior of flattened netlist extracted from the design files, while timing simulation uses a fully compiled netlist containing timing information to test both the logical operation and the worst-case timing for the design in the target device. Before running a simulation, it is necessary to specify input vectors as the stimuli for the Quartus II Simulator. The simulator uses these input vectors to simulate the output signals that a programmed device would produce under the same conditions. The simulator supports input vector stimuli in the form of a vector waveform file (**.vwf**), vector table output file (**.tbl**), power input file (**.pwf**), or a Quartus II generated vector file (**.vec**) or simulator channel file (**.scf**).

Quartus II is a software tool produced by Altera for analysis and synthesis of HDL designs, which enables the developer to compile their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.

Quartus II Web Edition is a free version of Quartus II that can be downloaded or delivered by mail for free. This edition provided compilation and programming for a limited number of Altera devices. The low-cost Cyclone family of FPGAs is fully supported by this edition, as well as the MAX family of CPLDs, meaning small developers and educational institutions have no overheads from the cost of development software. License registration is required to use the Web Edition of Quartus II, which is free and can be renewed an unlimited number of times.

Modelsimanother simulator tools which is use two simulate the algorithm. First make two files test bench and source conde(.v) file. Then make a new project and browse project location



Project Name:

Project Location: C:/altera/12.1/modelsim_ase

Default Library Name: work

Copy Settings From: 1/modelsim_ase/modelsim.ini

[✓ Copy Library Mappings] Reference Library Mappings

Figure 11Modelsim start window

In next step following window appear

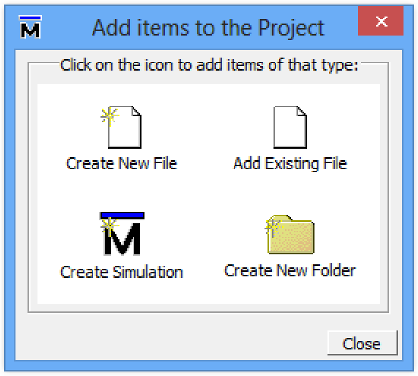


Figure 12 : window for adding Items

Then need to click add existing file and browsing both test bench and source file. One important thing is you must select Language during uploading existing file on bellow screen

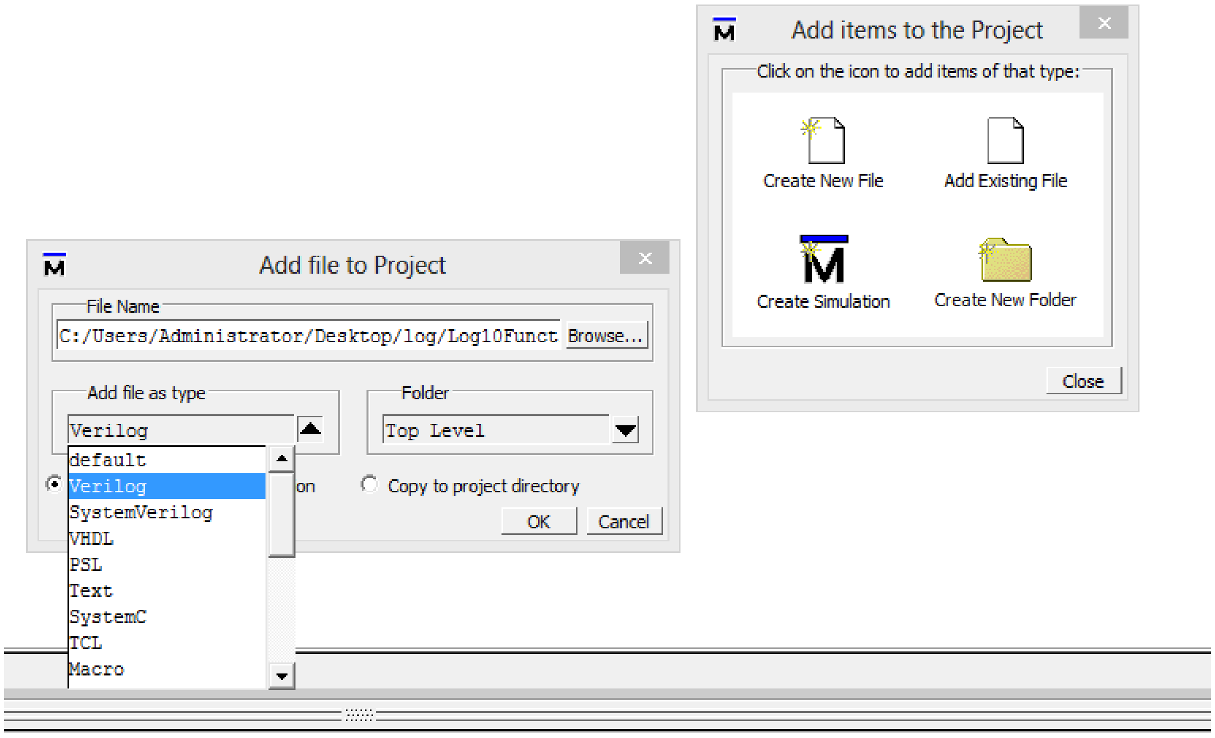
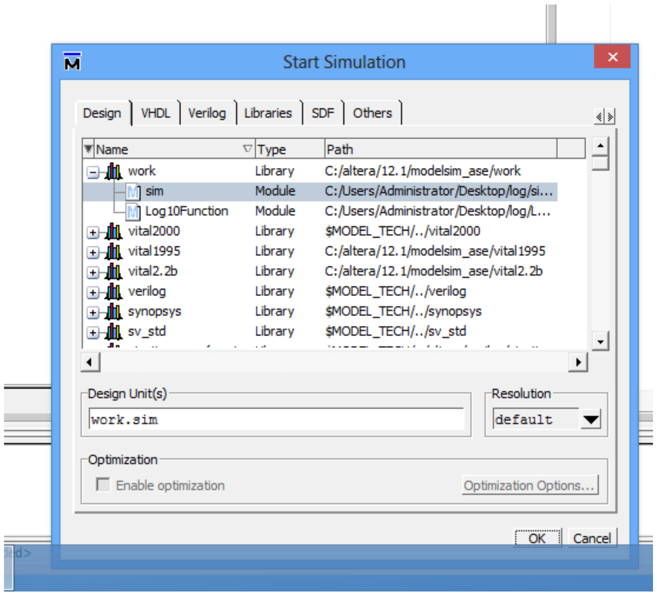


Figure 13 Selecting language type

Once all file are upload completely then first compile and then simulate and then run to get output

During simulate chose the test bench for start form bellow window here sim.v is the test bench



Name	Type	Path
work	Library	C:/altera/12.1/modelsim_ase/work
sim	Module	C:/Users/Administrator/Desktop/log/si...
Log10Function	Module	C:/Users/Administrator/Desktop/log/L...
vital2000	Library	$MODEL_TECH/../vital2000
vital1995	Library	C:/altera/12.1/modelsim_ase/vital1995
vital2.2b	Library	C:/altera/12.1/modelsim_ase/vital2.2b
verilog	Library	$MODEL_TECH/../verilog
synopsys	Library	$MODEL_TECH/../synopsys
sv_std	Library	$MODEL_TECH/../sv_std

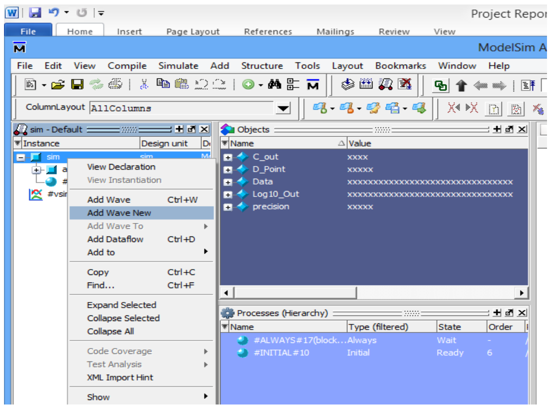
Design Unit(s): work.sim

Resolution: default

Optimization: Enable optimization [ ]

Figure 14 : Selecting test bench

After run you need to add web form by right click on source file.Find the bellow window for how to add web form



Name	Value
C_out	xxxx
D_Point	xxxxx
Data	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Log10_Out	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
precision	xxxxx

Name	Type (filtered)	State	Order
#ALWAYS#17(block...	Always	Wait	-
#INITIAL#10	Initial	Ready	6

Figure 15 : Adding Wave for result

Finally click run and the result will show in window.

Chapter 4

Results and Performance Analysis

4.1 Introduction

In this chapter we will compare proposed algorithm with similar related research/algorithm in various aspect like power, accuracy, number of logic cell used, speed etc. Performance with various facets will also discuss in this chapter.

4.2 Summary of proposed algorithm design

The proposed logarithmic converter is designed using Verilog HDL and synthesized by Quartus II 32-bit Version 12.0 Build 263 08/02/2012 SP 2 SJ Web Edition with Cyclone II, Device EP2C20F484C7.

Proposed system require 6002 logic elements, 79 PIN, LUT Flip Flop 40 , Input and output delay are 7 and 43ns, Power consumption is 49.60mW. A comparative evaluation in various expects are shown below:

4.3 Power Analysis

Our aim is to develop a system with very low power thus we emphasize our algorithm for very low power logic and found the lowest than any other system. Comparison table show various algorithms with our proposed Algorithms.

To calculate power we use **power Play power analyzer** tool provided by Altera Quartus II

Total Thermal Power Dissipation:	49.60mW
Core dynamic Thermal Power Dissipation:	0 mW
Core static Thermal Power Dissipation:	25.61 mW
I/O Thermal Power Dissipation:	23.98 mW

Table 3: (power comparison)

Ref,	Model	Power (mW)
[7] D. Chen	Curve fitting	108
[3] Ramin T.	Shift and Multiplication	125
[6] Dongdong et al.	Iterative	79
[4]Dongdong C.·	Polynomial Approximation	------
Proposed	Shift whit precision unit	49.60

4.3 Hardware Comparisons

Beside power another important aspect is cost. Our proposed system costs6002 logic cell although it is high but it is due to our precision unit which costs 988 and in lookup table it cost only 40 latches which is also lower than any other LUT based algorithm. In iterative system proposed in [6] and high precision system proposed in [3] is much higher logic cell than our proposed system

We use Compilation Result summery report in QUARTUS II for counting total logic cell in the project

Table 4: Table (Hardware comparison)

	Logic Gates	LUT FF
[1] Haohuan F	------	48
[6] Dongdong Chen	9602	-----
[7]Younhee Choi	5,684	-----
[3] Ramin T.,	6752	----
Proposed	6002	40

4.4 Speed and Accuracy

After develop the system we manually calculate 165 random sample number among them 60 are integer and rest 105 are floating point. During our assessment we also monitor the performance of our accuracy unit with minimum and maximum cycle of precision.

Table 5: (Accuracy Assessment)

Decimal point	Precision unit cycle	Accuracy (%)
4	Min 10 Max 19	99.99
6	Min 10 Max 22	99.04
8	Min 11 Max 29	97.02

We use Altera U.P. simulator both Qsim and Modelsim to simulate our algorism Figure 5 is a sample decimal output web form shown bellow

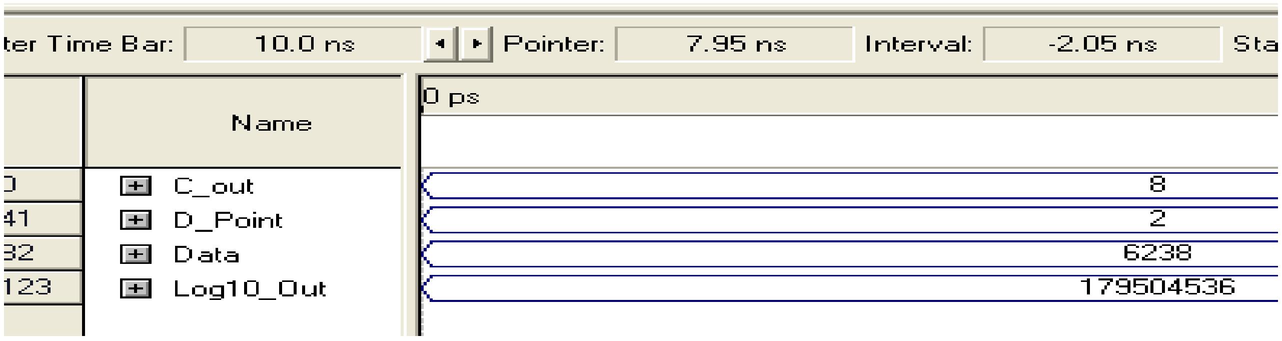


Figure 16: Simulation Result(Qsim) log10 (62.68)

Simulation of Algorithm using ModelSim provided by Altera is shown in the figure 17 bellow. All the cases the result is accurate up to 5-8 decimal points.

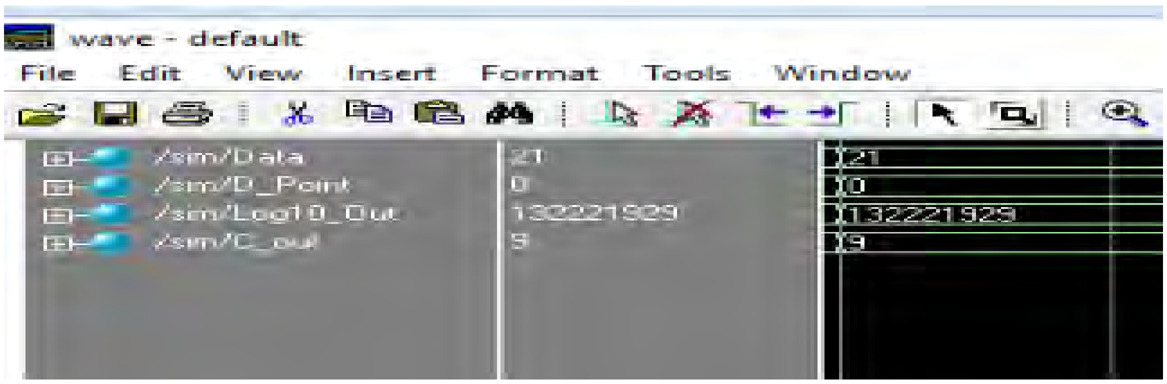


Figure 17: Result of $\log_{10}$ (21) = 1.322221929

There is a Speed-Accuracy Trade-Off: in the proposed algorithm more iteration is required to get more accurate result in that case memory (ROM) size need to larger only if the lookup table is reach the value of X is possible to more near to one. Table III shows the accuracy of our algorithm. Up to three decimal point it work very fast and almost error free. In our proposed algorithm we mention earlier that the accuracy is depend on user requirement and our accuracy can be more increase by updating lookup table and then of course the speed will also decrease.

Chapter 5

Conclusion

5.1 Conclusion

Low power has emerged as a principal theme in today's electronics industry. The need for low power has caused a major paradigm shift where power dissipation has become as important a consideration as performance and area. In the past, the major concerns of the ASIC designer were area, performance, cost and reliability; power consideration was mostly of only secondary importance. In recent years, however, this has begun to change and, increasingly, power is being given comparable weight to area and speed considerations.

A low power decimal logarithmic converter has been presented in this paper. The design of the converter is performed using Verilog HDL which is compiled and simulated in the Cyclone 2 FPGA platform. The simulation results show that the proposed converter is working properly with respect to different user defined precision requirement. The hardware requirements and power consumption of the design have been compared with that of other researchers which show its superiority over all the existing design in the said area. In terms of power 49.37 mW which is lowest and logic cell 6002 is also les as compared with other similar research. Beside those the accuracy is depends on memory which is possible to increase as per requirement.

5.2 Suggestions for Further Works

This project was focused on achieving low power computation of decimal logarithm that was targeted to ASIC. However various aspects such as area cost optimization, Precision were not addressed. Future work will focus on the development of a resource efficient technique for the logarithm in different bases. Here two major suggestions for future work is given

I. Overall performance can be developed by optimum combination of standard number (128, 64 etc.), which used in shift operation avoiding multiplication by keeping power very low.

II. The area cost may be reduced by resource sharing, minimizing the size of memory required. Optimize the precision unit by keeping memory low.

References

[1] Haohuan F., Oskar M., Wayne L. "Optimizing Logarithmic Arithmetic on FPGAs," Department of Computing, Imperial College London, United Kingdom. IEEE Transaction on Computers , Vol. 59 (7), pp 1000-1006, July 2010

[2] Alachiotis N., Stamatakis A. "Efficient Floating-Point Logarithm Unit for FPGA", TheExelixis Lab Dept. of Computer Science Technische University at Germany. Symposium on Parallel Distributed Processing, pp.1-8, 19-13 April 2010

[3] Ramin T., Ashraful I., and Khan A. W. "Fast Algorithm of A 64-bit Decimal Logarithmic Converter "JOURNAL OF COMPUTERS, VOL. 5, NO. 12, DECEMBER 2010

[4] Dongdong C.·Seok-Bum K. "Novel Decimal Logarithmic Converter Based on First-Order Polynomial Approximation" Circuits Syst Signal Process (2012)

[5] Tajallipour R., Teng D., Seok-Bum K., and Wahid K."On the Fast Computation of Decimal Logarithm", Proceedings of the International Conference on Computer and Information Technology (ICCIT), 21-23 December, 2009, Dhaka, Bangladesh

[6] Dongdong C., Zhang Y., Younhee C., Moon L.,Seok-Bum K., "A 32-bit Decimal Floating-Point Logarithmic Converter," Proc. of the IEEE Symposium on Computer Arithmetic, pp. 195-203, 2009.

[7] ChenD., ChoiY., Chen L., TengD., WahidK., "A Novel Decimal-to-decimal Logarithmic Converter", Proc. of the IEEE Int. Symposium on Circuits and Systems, pp.688-691, 2008.

[8] Salvador E., "FPGA Implementation of base-N logarithm", lectrónicaeInformáticauenos Aires, Argentin

[9] Hough, D., "Applications of the Proposed IEEE 754 Standard for Floating–Point Arithmetin.", Computer, vol.14, (3), pp. 70-74, 2006.

[10] Lzzeldin M. ,Chay F. "32 bit NXN Matrix Multiplication : Performance Evaluation for Altera FPGA" International Journal of Computer Application, Vol. 52(11), August 2012

[11] Cornea, M., Crawford, J., " IEEE 754R Decimal Floating-PointArithmetic: Reliable and Efficient Implementation for Intel Architecture Platforms.", Intel Technology Journal, Designing Technology with People in Mind, vol. 11(1), 2007.

[12] Thakarea L.P., Deshmukh Y. “Area Efficient Complex Floating Point Multiplier for Reconfigurable FFT/IFFT Processor Based on Vedic Algorithm” Procedia Computer Science 79 (2016) 434 – 440

[13] Purna, A., Venkata N. T., Mallikarjuna P. A. “An FPGA Based High Speed IEEE - 754 Double Precision Floating Point Adder/Subtractor and Multiplier Using Verilog” Vol. 52, March, 2013

[14] Mitchell, N. “Computer Multiplication and Division Using Binary Logarithms”, IRE Trans. Electron.Computer, pp. 512-517, 1962.

[15] http://en.wikipedia.org/wiki/Logarithm; last access on 03-09-2016

[16] ZhuM., “On High-Performance Parallel Fixed-Point Decimal Multiplier Designs,” Computers & Electrical Engineering, Volume 40, Issue 7, October 2014, Pages 2126–2138

[17] Pedram M., “Design Technologies for Low Power VLSI”, To appear in Encyclopedia of Computer Science and Technology, 2005 , Department of EE-Systems University of Southern California Los Angeles , CA 90089

Outcome of the Project

A conference paper has been published from the part of the research as proposed in this project.

Mian R, Ali L. “Design of Low Power Decimal Logarithmic Converter” International Conference on Electrical, Computer and Communication Engineering (ECCE), February 16-18, 2017, Cox’s Bazar, Bangladesh, Page 153-157