RESEARCH ARTICLE OPEN ACCESS

IMPLEMENTATION OF DOUBLE PRECISION FLOATING POINT MULTIPLIER USING VEDIC MULTIPLIER

M.VIJAY KUMAR[1],G.NARESH[2],M.NOMITHA REDDY[3],B.L.KLUSUMA[4]

1.(Assistant Professor,Electronics and Communication Engineering ,Aditya College of Engineering Madanapalle

Email:molakavijay@gmail.com)

2.(Assistant Professor, Electronics and Communication Engineering,Aditya College of Engineering,Madanapalle

Email:gnaresh427@gmail.com)

3.(Electronics and Communication Engineering, Aditya College of Engineering, Madanapalle

Email:nomithareddym@gmail.com)

4.(Electronics and Communication Engineering, Aditya College of Engineering, Madanapalle

Email:blkusuma21@gmail.com)

Abstract:

Floating point multiplier is one of the vital concerns in every digital system. In this paper, Vedic multiplier is one such high speed, low area multiplier architecture are used for the implementation of a High speed double precision binary Floating point multiplier by using IEEE 754 standard. Since Vedic multiplier is special kind of multiplier which is capable to multiply more number of bits at a time, the use of this multiplier makes the multiplier faster as compared to the conventional multiplier. For Mantissa calculation, a 53×53 bit multiplier has been developed by using this Vedic multiplier.

INTRODUCTION

Floating point multiplication units are essential Intellectual Properties (IP) for modern multimedia and high performance computing such as graphics acceleration, signal processing, image processing etc. There are lot of effort is made over the past few decades to improve performance of floating point computations. Floating point units are not only complex, but also require more area and hence more power consuming as compared to fixed point multipliers. And the complexity of the floating point unit increases as accuracy becomes a major issue. Even a minute error in accuracy can cause major consequences. These errors are possible in floating point units mainly because of the discrete behavior of the IEEE-754 floating point representation, where fixed number of bits is used to represent numbers. Due to the high computational requirements of scientific applications such as computational geometry, climate modeling, computational physics, etc., it is necessary to have extreme precision in floating point calculations. And these increased precision may not be provided with single precision or double precision format. That further increases the complexity of the unit. But some
applications do not require high precision. Even anapproximate value will be sufficient for the correct operation.For applications which require lower precision, the use ofdouble precision or

quadruple precision floating point units will be a luxury. It wastes area, power and also increases latency.

For devices such as portable or wearable devices in which accuracy requirement varies with different applications and also power consumption is a very important factor, use of high precision floating point multipliers is not a good option. In such cases a variable precision multiplier will be a good option which can save much power and time when application doesn't need high precision. Most of such designs make use of already available IPs such as DSP (Digital Signal Processing) unit sand 18x18 multiplier units.

High processing performance and low power dissipation are the most important objectives in many multimedia and digital signal processing (DSP) systems, where multipliers are always the basic arithmetic unit and significantly influence the system's performance and power dissipation. Multipliers using floating point numbers are in great demand because floating point numbers have good precision, since they never deliberately discard information. So a fast and energy-efficient floating point unit is always needed in electronics industry.

Field Programmable Gate Arrays (FPGA's) are broadly used for scientific computation because of the ease of customizing the hardware for the application. The limited size and architecture of FPGAs are not well-suited for floating-point applications. On the other hand, ASICs can be very efficient at floating-point operations, but lack the programmability and flexibility that is desired in many situations, and the cost of an ASIC can be prohibitively high. By overcoming the limitations of FPGAs, it will be very attractive platform for floating point applications. So for the better utilization of the floating point multiplier unit, reconfigurable computing is added. A new computing method using reconfigurable architectures promises an intermediate trade-off between flexibility and performance. Reconfigurable computing uses hardware that can be adapted at run-time to facilitate greater flexibility without compromising on performance. The re-configurability of the hardware permits adaptation of the hardware for specific computations in each application to achieve higher performance compared to software. Here the re-configurability is applied to perform single precision and double precision floating point multiplication. In order to evaluate the proposed FPGA architecture, one of the most widely performed operations in DSP namely finite impulse response (FIR) filter is used for evaluation.

EXISTING SYSTEM

Array multiplier is well known due to its regular structure. Multiplier circuit is based on add and shift algorithm. Each partial product is generated by the multiplication of the multiplicand with one multiplier bit. The partial product are shifted according to their bit orders and then added. The addition can be performed with normal carry propagate adder. N-1 adders are required where N is the multiplier length.As it increases the complexity we are going to the proposed system called as vedic multiplier.

			x	A3	A2	A1	A0	Inputs
				B3	B2	B1	B0	
			C	B0 x A3	B0 x A2	B0 x A1	B0 x A0	
		+	B1 x A3	B1 x A2	B1 x A1	B1 x A0		
		C	sum	sum	sum	sum		
	+	B2 x A3	B2 x A2	B2 x A1	B2 x A0			Internal Signals
	C	sum	sum	sum	sum			
+	B3 x A3	B3 x A2	B3 x A1	B3 x A0				
C	sum	sum	sum	sum				
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Outputs

PROPOSED SYSTEM

DOUBLE PRECISION FLOATING POINT ALGORTHIM

Multipliers are key components of many high performance systems such as FIR filters, Microprocessors, Digital Signal Processors etc. A system's performance is generally determined by the performance of the multiplier, because the multiplier is generally the slowest element in the system. Furthermore, it is generally the most area consuming. Hence, optimizing the speed and area of the multiplier is a critical issue for an effective system design.

FLOATING POINT ALGORTHIM FOR MULTIPLICATION

To multiply two floating pointing point numbers the following steps are done.
1.obtaining the sign s1 xor s2.
2 . Adding the exponents i.e;E1 and E2 and then subtracting the bias value.
3. Multiplying the significand i,e M1 and M2.
4. Placing the decimal point in the result.
5. Normalizing the result ; i.e.obtaining 1 at the MSB of the results significands.
6. Rounding the result to fit in the available bits.
7.Checking for underflow\overflow occurance.

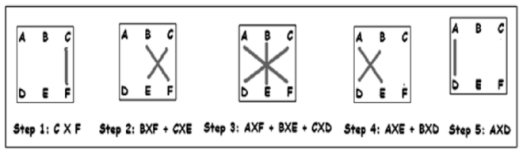
DESIGN OF FLOATING POINT

The multiplier for the floating point numbers represented in IEEE 754 format can be divided into three different units: Mantissa Calculation unit, Exponent Calculation unit and Sign Calculation unit.The Multiplier receives two 64-bit floating point numbers. First these numbers are unpacked by separating the numbers into sign, exponent,and mantissa bits. The floating point multiplication iscarried out in following three parts. Sign calculation unit:- In this unit, we determine the sign of the product by performing a XOR operation on the sign bits of the two operands. Exponent calculation unit:- This unsigned adder is used for adding the exponent of the first input to the exponent of the second input and subtracting the Bias (1023) from the addition result.

The exponent of the result must be 11 bits in size, and must be between 1 and 2046 otherwise the value is not a normalized one. Overflow/underflow means that the result‟s exponent is too large/small. Mantissa Calculation unit:- The multiplication is done in two steps, partial product generation and partial product addition. For double precision operands (53-bit fraction fields), a total of 53*53-bit multiplier is required. Mantissa multiplier unit performs multiplication operation. After this the output ofmantissa is normalized, i.e. if the MSB of the result
obtained is not 1, then it is left shifted to make the MSB 1. If changes are made by shifting then corresponding changes has to be made in exponent also. The mantissa of operand A and the leading „1‟ (for normalized numbers) are stored in the 53-bit register (Ma). The mantissa of operand B and the leading „1‟ (for normalized numbers) are stored in the 53-bit register (Mb). Multiplying all 53bits of Ma by 53 bits of Mb would result in a 106-bit
product. Rounding is used to fit the result in available bit then output of mantissa multiplication is 56 bits.

VEDIC MULTIPLIER

The "UrdhvaTiryagbhyam" Sutra is a generalmultiplication formula applicable to all cases ofmultiplication such as binary, hex, decimal and octal. TheSanskrit word "Urdhva" means "Vertically" and"Tiryagbhyam‟ means "crosswise".



PROPOSED MULTIPLIER

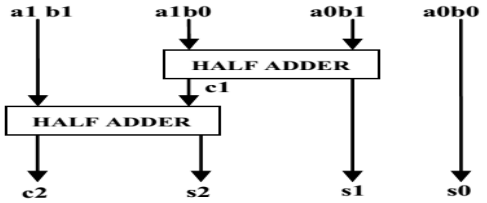


Figure 4:2x2 Vedic Multiplier

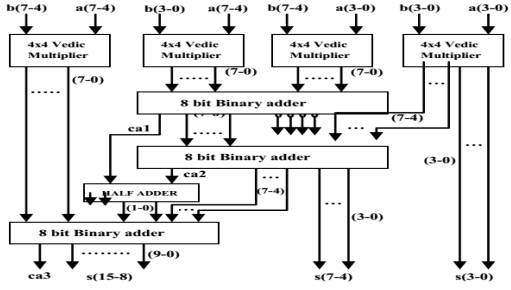
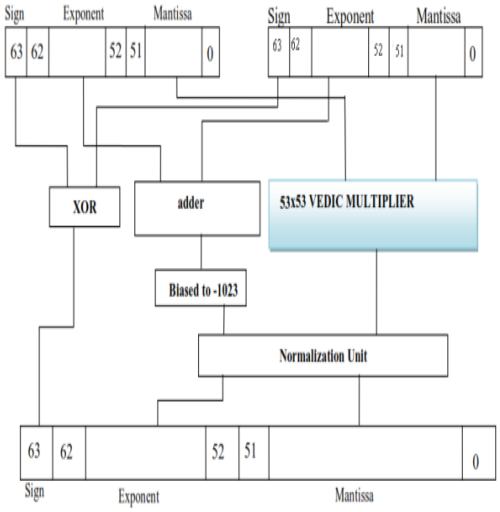
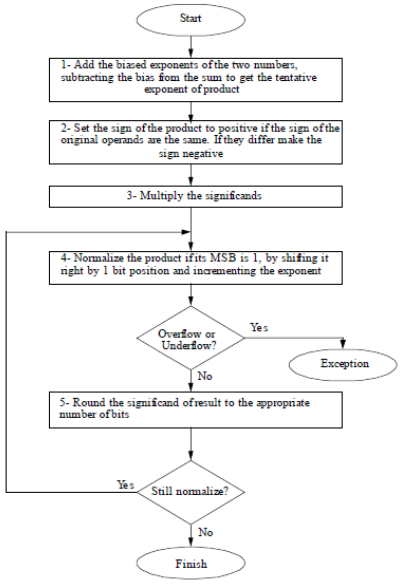


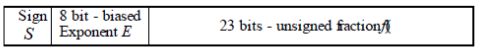
Figure 6:8x8 Vedic Multiplier

PROPOSED ARCHITECTURE OF FLOATING POINT MULTIPLIER

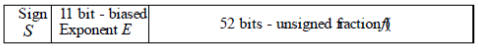


FLOWCHART





(a) IEEE single precision data format



(b) IEEE double precision data format

EXAMPLE

Multiply the following two numbers. Use IEEE 754 standard:

A= 25.5_{10} B= -0.375_{10}

Answer:

A can be represented as A= 1.10011 * 2^4or exp= 127+ 4 = 131_{10}, Sig =1.10011, S=0

0	1 000 0011	10011000000000000000000

B can be represented as B= 1.1 * 2^{-2} or exp= 127 -2= 125_{10} , sig =1.1 , S=1

1	0 1111101	10000000000000000000000

Add exponent and subtract bias
Exponent = 1 000 0011 + 0 1111101 - 01111111 = 1000 0001

Multiply Significands 1.1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 *
1.1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
10.011001000000000000000000000000000000000
Now round the results.

After rounding we get: 10. 01100100000000000000
After normalization we get: 1. 0 01100100000000000000 * 2^1

New exponent after normalization 1 000 0001 +00000001 = 10000010
Final result

1	10000010	00110010000000000000000

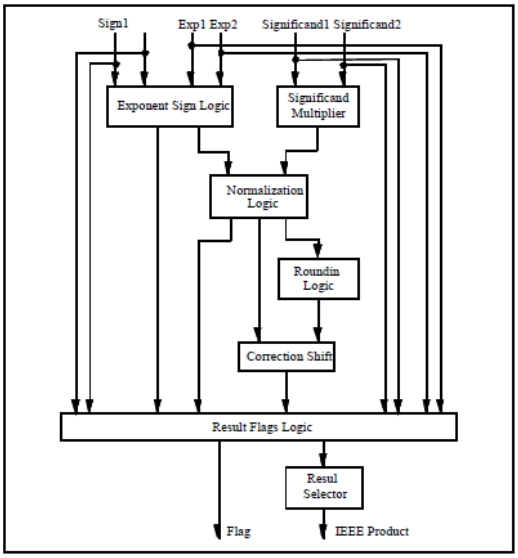
Unbiased value of exponent is 1000 0010 -0111 1111 = 0000-0011 ie $(130\text{-}127 =3)_{10}$

A * B = 1. 0011001 * 2^3 = -9.5625_{10}

MULTIPLIER ARCHITECTURE

The exponent logic is responsible for extracting the exponents and adding them and subtracting the bias. The Control/Sign logic, decodes the sign bit (EXOR) and directs the significands to the integer multiplier.

The results of the significands multiplication is rounded by the rounding logic and if necessary is normalized through a shift operation. The exponent in updated by the exponent increment block and the final results, are presented to the output logic. This architecture is shown in figure below



Xilinx Verilog HDL Tutorial

On the off chance that you wish to take a shot at this instructional exercise and the lab at home, you should download and introduce Xilinx and ModelSim. These apparatuses both have free understudy renditions. It would be ideal if you finish Appendix B, C, and D in a specific order before proceeding with this instructional exercise. Also on the off chance that you wish to buy your own Spartan3 board, you can do as such at Digilent's Website. Digilent offers scholastic valuing. If it's not too much trouble take note of that you should download and introduce Digilent Adept programming. The product contains the drivers for the board that you require furthermore gives the interface to program the board.

Introduction

Xilinx Tools is a suite of programming devices utilized for the outline of computerized circuits actualized utilizing Xilinx Field Programmable Gate Array (FPGA) or Complex Programmable Logic Device (CPLD). The configuration technique comprises of (an) outline section, (b) union and usage of the configuration, (c) practical recreation and (d) testing and check. Computerized outlines can be entered in different ways utilizing the above CAD instruments: utilizing a schematic passage device, utilizing an equipment depiction dialect (HDL) – Verilog or VHDL or a mix of both. In this lab we will just utilize the configuration stream that includes the utilization of Verilog HDL.

1. Make verilog outline information file utilizing format driven proofreader.
2. Order and actualize the verilog outline file
3. Make the test vectors and reproduce the configurations without utilizing a PLD
4. Relegant information\pins to execute the configuration on an objective gadget.
5. Download bitstream to a FPGA or CPLD gadget.
6. Test plan on FPGA\CPLD gadget.

A Verilog information record in the Xilinx programming environment comprises of the accompanying fragments:

Header: module name, rundown of info and yield ports.

Revelations: information and yield ports, registers and wires.

Rationale Descriptions: conditions, state machines and rationale capacities.

End: endmodule

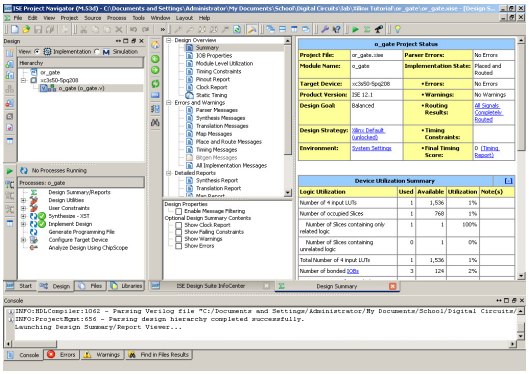
All your plans for this lab must be determined in the above Verilog information group. Note that the state chart fragment does not exist for combinational rationale plans.

Programmable Logic Device: FPGA

In this lab computerized outlines will be actualized in the Basys2 board which has a Xilinx Spartan3E –XC3S250E FPGA with CP132 bundle. This FPGA part has a place with the Spartan group of FPGAs. These gadgets arrive in an assortment of bundles. We will utilize gadgets that are bundled in 132 pin bundle with the accompanying part number: XC3S250E-CP132. This FPGA is a gadget with around 50K doors. Definite data on this gadget is accessible at the Xilinx site.

Creating a New Project

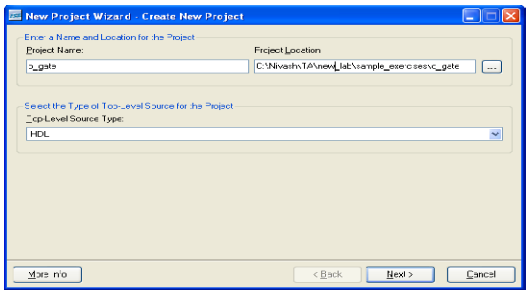
Xilinx Tools can be begun by tapping on the Project Navigator Icon on the Windows desktop. This ought to open up the Project Navigator window on your screen. This window demonstrates the last got to extend.



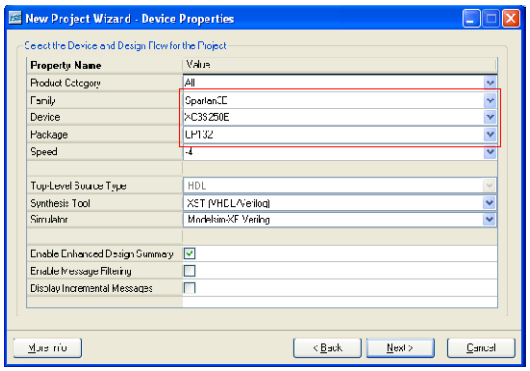
Xilinx Project Navigator window (snapshot from Xilinx ISE software)

Opening a project

Select File->New Project to make another venture. This will raise another task window on the desktop. Top off tthe fundamental passages as takes after



New Project Initiation window (snapshot from Xilinx ISE software)



Device and Design Flow of Project (snapshot from Xilinx ISE software)

For each of the properties given below, click on the 'value' area and select from the list of values that appear.

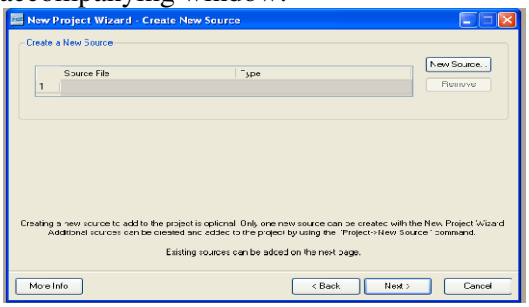
- Device Family: Family of the FPGA/CPLD utilized. In this research facility we will utilize the Spartan3E FPGA's.
- Device: The quantity of the real gadget. For this lab you may enter XC3S250E (this can be found on the joined prototyping board)
- Package: The sort of bundle with the quantity of pins. The Spartan FPGA utilized as a part of this lab is bundled in CP132 bundle.
- **Speed** Grade: The Speed evaluation is "-4".
- Synthesis Tool: XST [VHDL/Verilog]
- Simulator: The instrument used to recreate and check the usefulness of the configuration. Modelsim test system is coordinated in the Xilinx ISE. Henceforth pick "Modelsim-XE Verilog" as the test system or even Xilinx ISE Simulator can be utilized.
- Then click on NEXT to spare the sections.

All anticipate records, for example, schematics, netlists, Verilog documents, VHDL documents, and so forth., will be put away in a subdirectory with the venture name. An undertaking can just have one top level HDL source record (or schematic). Modules can be added to the task to make a measured, progressive outline.

Keeping in mind the end goal to open a current undertaking in Xilinx Tools, select File->Open Project to demonstrate the rundown of ventures on the machine. Pick the undertaking you need and snap OK.

Tapping on NEXT on the above window raises the accompanying window:

Tapping on NEXT on the above window raises the accompanying window:



Create New source window (snapshot from Xilinx ISE software) If creating a new source file, Click on the NEW SOURCE.

Creating a Verilog HDL input file for a combinational logic design

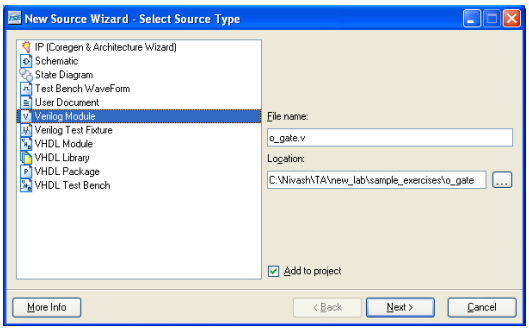
In this lab we will enter an outline utilizing a basic or RTL portrayal utilizing the Verilog HDL. You can make a Verilog HDL info document (.v record) utilizing the HDL Editor accessible as a part of the Xilinx ISE Tools (or any word processor).

In the previous window,

click on the NEW SOURCE

A window appears as appeared in Figure 4. (Note: "Add to venture" choice is chosen as a

matter of course. In the event that you don't choose it then you will need to add the new source document to the undertaking physically.)



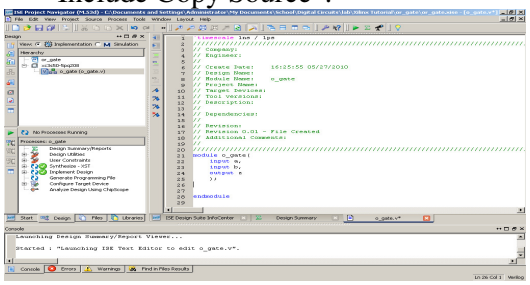
Select Verilog Module and in the "Record Name:" region, enter the name of the Verilog source document you are going to make. Additionally ensure that the alternative Add to venture is chosen so that the source need not be added to the task once more. At that point click on Next to acknowledge the sections. This appears the accompanying window.

In the Port Name segment, enter the names of all info and yield sticks and determine the Direction as needs be. A Vector/Bus can be characterized by entering suitable piece numbers in the MSB/LSB segments. At that point click on Next>to get a window demonstrating all the new source data. On the off chance that any progressions are to be made, simply tap on <Back to do a reversal and roll out improvements. In the case of everything is satisfactory, click on Finish > Next > Next > Finish to proceed.

On the off chance that a source must be expelled, simply right tap on the source record in the Sources in Project window in the Project Navigator and select Remove in that. At that point choose Project - > Delete Implementation Data from the Project Navigator menu bar to evacuate any related records.

Editing the Verilog source file

The source record will now be shown in the Project Navigator window. The source document window can be utilized as a content manager to roll out any essential improvements to the source record. All the info/yield pins will be shown. Save your Verilog program intermittently by selecting the File->Save from the menu. You can likewise alter Verilog programs in any content tool and add them to the undertaking index utilizing "Include Copy Source".



Verilog Source code editor window in the Project Navigator (from Xilinx ISE software)

A brief Verilog Tutorial is accessible in Appendix-A. Subsequently, the dialect sentence structure and development of rationale conditions can be alluded to Appendix-A.

The Verilog source code format produced demonstrates the module name, the rundown of ports furthermore the announcements (information/yield) for every port. Combinational rationale code can be added to the verilog code after the presentations and before the endmodule line.

For example, an output z in an OR gate with inputs a and b can be described as, assignz = a | b;

Different builds for displaying the rationale work: A given rationale capacity can be demonstrated from multiple points of view in verilog. Here is another case in which the rationale capacity, is actualized as a truth table utilizing a case explanation:

```
moduleor_gate(a,b,z);
input a;
input b;
output z;
reg z;
always @(a or b)
begin
case ({a,b})
00: z = 1'b0;
01: z = 1'b1;
10: z = 1'b1;
11: z = 1'b1;
End case
End
End module
```

Assume we need to portray an OR entryway. It should be possible utilizing the rationale condition as appeared in Figure 9a or utilizing the case explanation as appeared in Figure. These are only two case builds to outline a rationale capacity. Verilog offers various such develops to productively demonstrate plans. A brief

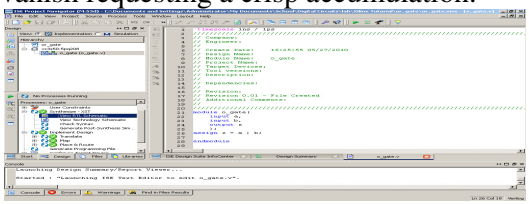
instructional exercise of Verilog is accessible in Appendix-A.

Synthesis and Implementation of the Design

The outline must be combined and executed before it can be checked for rightness, by running

useful recreation or downloaded onto the prototyping board. With the top-level Verilog document opened (should be possible by double tapping that record) in the HDL manager window in the right 50% of the Project Navigator, and the perspective of the venture being in the Module view , the actualize plan alternative can be found in the process view. Plan passage utilities and Generate Programming File alternatives can likewise be found in the process view. The previous can be utilized to incorporate client requirements, assuming any and the last will be examined later.

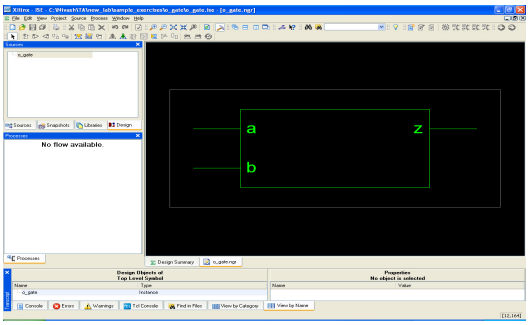
To actualize the configuration, double tap the Implement plan choice in the Processes window. It will experience steps like Translate, Map and Place and Route. On the off chance that any of these strides wasn't possible or finished with blunders, it will put a X mark before that, generally a tick imprint will be put after each of them to show the effective culmination. On the off chance that everything is done effectively, a tick imprint will be put before the Implement Design choice. In the event that there are notices, one can mark before the choice showing that there are a few notices. One can take a gander at the notices or blunders in the Console window present at the base of the Navigator window. Each time the outline document is spared; all these imprints vanish requesting a crisp accumulation.



Implementing the Design (snapshot from Xilinx ISE software)

The schematic chart of the integrated verilog code can be seen by double tapping View RTL Schematic under Synthesize-XST menu in the Process Window. This would be a convenient approach to investigate the code if the yield is not meeting our details in the proto sort board.

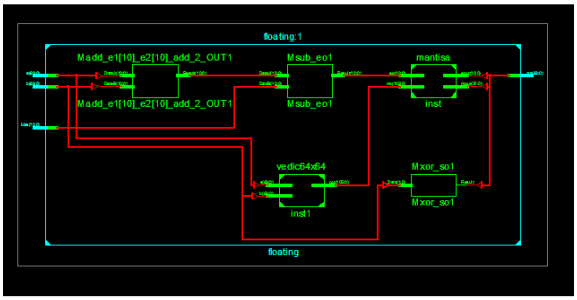
By double clicking it opens the top level module showing only input(s) and output(s) as shown below.



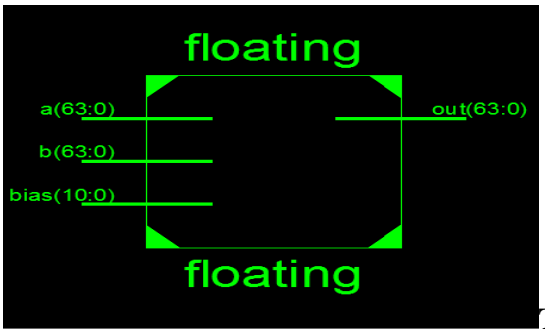
Realized logic by the XilinxISE for the verilog code

BLOCK DIAGRAM

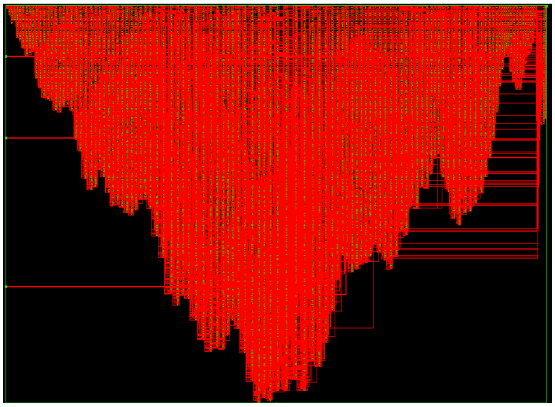
RTL SCHEMATIC



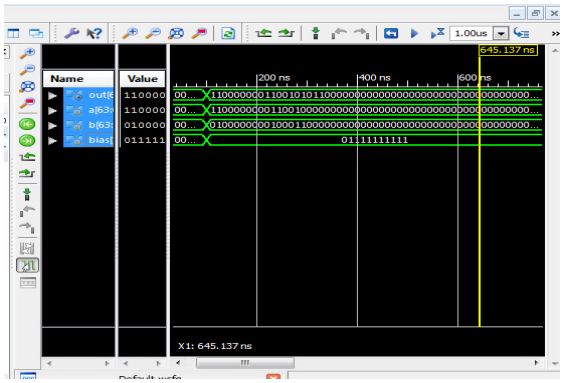
TECHNOLOGY SCHEMA



TIC



SIMULATION RESULT



CONCLUSION

In this paper, double precision floating point multiplier based on the IEEE-754 format is successfully implemented on FPGA. The modules are written in Verilog HDL to optimize implementation on FPGA. In this implementation they obtained maximum frequency that of the multiplier implemented by using Vedic algorithm. Since the main idea behind this implementation is to increase the speed of the multiplier by reducing delay at every stage using the optimal Vedic multiplier design, it gives the advantage of less delay in comparison to other method. The results obtained using the proposed algorithm and implementation is better not only in terms of speed but also in terms of hardware used.

REFERENCES

[1] PurnaRameshAddanki,VenkataNagaratnaTilakAlapati andMallikarjuna Prasad Avana, "An FPGA Based High Speed IEEE – 754Double Precision Floating Point Adder/Subtractor and Multiplier Using Verilog", International Journal of Advanced Science and Technology,Vol. 52, pp.61 -74, March 2013.

[2] RiyaSaini and R.D.Daruwala,"Efficient Implementation of PipelinedDouble Precision Floating Point Multiplier",International Journal ofEngineering Research and Applications, Vol. 3, Issue 1, pp.1676-1679,January - February 2013.

[3] Anna Jain, Baisakhy Dash and Ajit Kumar Panda,"FPGA Design ofa Fast 32-bit Floating Point Multiplier Unit", IEEE ConferencePublications, pp. 545 – 547, 2012.

[4] AddankiPurnaRamesh and Rajesh Pattimi," High Speed Double Precision Floating Point Multiplier",International Journal of Advanced Research in Computer and Communication Engineering, Vol. 1, Issue 9,pp. 647 – 650, November 2012.

[5] Xia Hong and JiaJinging,"Research and optimization on RoundingAlogrithms for Floating-Point Multiplier",IEEE ConferencePublications,International Conference on Computer Science andElectronics Engineering, pp. 137 – 142, 2012.

[6] Tashfia.Afreen, Minhaz. UddinMdIkram, Aqib. Al Azad, andIqbalurRahmanRokon, "Efficient FPGA Implementation of DoublePrecision Floating Point Unit Using Verilog HDL" ,InternationalConference on Innovations in Electrical and ElectronicsEngineering,Oct. 6-7, 2012 .

[7] PuneetParuthi, Tanvi Kumar and HimanshuSingh,"Simulation ofIEEE 754 Standard Double Precision Multiplier using BoothTechniques", International Journal of Engineering Research and Applications, Vol. 2, Issue 5, pp. 1761 -1766, September-October2012.

[8] B.SreenivasaGanesh, J.E.N.Abhilash and G. Rajesh Kumar, "Designand Implementation of Floating Point Multiplier for Better TimingPerformance", International Journal of Advanced Research in ComputerEngineering & Technology, Vol. 1, Issue 7, September 2012.

[9] Mohamed Al-Ashrafy, Ashraf Salem and WagdyAnis,"An EfficientImplementation of Floating Point Multiplier",IEEEElectronics,Communications and Photonics Conference (SIECPC),Saudi International , pp. 1 - 5, 2011.

[10] Soojin Kim And Kyeongsoon Cho, "Design Of High-SpeedModified Booth Multipliers Operating At Ghz Ranges", WorldAcademy Of Science, Engineering and Technology, 2010.

[11] Gong Renxi, Zhang Shangjun, Zhang Hainan, MengXiaobi, GongWenying, XieLingling and Huang Yang, "Hardware Implementation ofa High Speed Floating Point Multiplier Based on FPGA", IEEEConference Publications, 4th International Conference on ComputerScience & Education, pp.1902 – 1906, 2009.

[12] Saroja.VSiddamal, R.M Banakar and B.C. Jinaga, "Design ofHigh-Speed Floating Point Multiplier", 4th IEEE InternationalSymposium on Electronic Design, Test & Applications, pp. 285 – 289,2008.

[13] Manish Kumar Jaiswal and NitinChandrachoodan,"EfficientImplementation of IEEE Double Precision Floating-Point Multiplier onFPGA", IEEE Region 10 Colloquium and the Third ICIIS, Kharagpur,India, December 2008.