# DOUBLE PRECISION FLOATING POINT CORE IN VERILOG

Aparna CV[1] and Mary Joseph[2]

[1]Department of electronics and communication engineering, MACE, Kothamangalam, A P J Abdul Kalam Technological University, Kerala, India

[2]Associate Professor, Department of Electronics and Communication, M. A College of Engineering, Kothamangalam

## *ABSTRACT*

*A floating-point unit (FPU) is a math coprocessor, a part of a computer system specially designed to carry out operations on floating point numbers. The term floating point refers to the fact that the radix point can "float"; that is, it can placed anywhere with respect to the significant digits of the number. Double precision floating point, also known as double, is a commonly used format on PCs due to its wider range over single precision in spite of its performance and bandwidth cost. This paper aims at developing the verilog version of the double precision floating point core designed to meet the IEEE 754 standard .This standard defines a double as sign bit, exponent and mantissa. The aim is to build an efficient FPU that performs basic functions with reduced complexity of the logic used and also reduces the memory requirement as far as possible.*

## *KEYWORDS*

IEEE 754, ModelSim, Double precision floating point format, Verilog

## 1.INTRODUCTION

In computing, double precision is a computer number format that occupies two adjacent storage locations in memory of computer. A double precision number, sometimes simply called the double may be defined as integer, fixed point or floating point. 32 bit modern computers use two memory locations to store 64 bit double precision number. Double precision floating point is an IEEE 754 standard used to encode binary or decimal floating point numbers in 64 bits (8 bytes).[1].

In computing floating point is a method of used to represent real numbers in a way that can support a wide range of values. Numbers in general represented as a fixed number of significant digits and scaled using an exponent. The base for this scaling is normally 2, 10 or 16.The typical number that can be represented exactly is of the form

$$\text{Significant digit} * base^{exponent}.$$

The term floating point refers to the fact that the radix point can float i.e., it can be placed anywhere to the significant digits of the number. This position is indicated separately in the internal representation and the floating point representation can thus be thought of as a computer realization of scientific notations. Since the 1990s, the most commonly encountered floating point representation is that defined by IEEE 754.

Verilog is standardized as IEEE 1364.it is a hardware description language used to model electronic systems. It is most commonly used for digital circuit design and verification at the register transfer level of abstraction. It is used in the verification of analog circuits and mixed signal circuits. [2] A subset of statement in verilog language is synthesizable. Verilog modules that obey a synthesizable coding style known as RTL (Register Transfer Level) .A synthesis software can be used to physically realize RTL. The Synthesis software algorithmically transforms the verilog source into a netlist, a logically equivalent description including only of elementary logic primitives [AND, OR etc] that are available in a specific FPGA or VLSI technology. Further manipulations to the netlist ultimately lead to a circuit fabrication blue print.

The Double precision floating point core in verilog was designed with three objectives in mind. First, develop efficient algorithms for Floating Point operations like addition, subtraction, division, multiplication, rounding and exception handling. Second, implement the proposed algorithm using Verilog. Third synthesize the above proposed algorithm.

## 2.PROPOSED SYSTEM

In the early days of digital computers, it was quite common that machines from different vendors have different word lengths and unique floating-point formats. This caused many problems, especially in the porting of programs between different machines (designs). The IEEE-754 floating point standard, formally named NSI/IEEE Std 754-1985, introduced in 1985 tried to solve these problems.

This paper implements a floating-point system confirming to this standard can be realized in software, entirely in hardware, and combination of hardware and software. The standard specifies two formats for floating-point numbers, basic (single precision) and extended (double precision), it also specifies the basic operations for both formats which are addition and subtraction of operations. Then different conversions are needed, as integer to floating-point, basic to extend and vice versa. Finally, it describes the different floating-point exceptions and their handling, including non-numbers (NaNs).

### 2.1.IEEE 754 Standard

The IEEE 754 standard is a technical standard established by IEEE and the most widely standard for floating point computation. It was created in the early 1980s after the introduction of word sizes of 32 bits (or 16 or 64). it based on a proposal from Intel who were designing the 8087 numerical coprocessor. Prof. W. Kahan was the first architect behind this proposal, for which he was awarded the 1989 Turing award. Almost all the modern machines follows IEEE 754 standard. Notable exceptions include IBM, main frames and Cray vector machines. Where the T90 series add an IEEE version but the sv1 still uses Cray floating point format.

## 2.2.Verilog

Verilog, standardized as IEEE 1364, is a hardware description language used for modeling electronic systems. It is most commonly used for digital circuits design and verification at the register transfer level of abstraction. It also used in the verification of analog circuits and mixed signal circuits. A subset of statement in verilog language is synthesizable. Verilog modules that conform to a synthesizable coding style known as RTL. RTL can physically realized by synthesis software. Synthesis software algorithmically transforms the verilog source into netlist; it is a logically equivalent description consisting of elementary logic primitives [AND, OR etc] which are available in FPGA and VLSI technology. Further manipulations to the netlist ultimately lead to a circuit fabrication blue print (such as photo, mask set for an ASIC or a bit stream fill for an FPGA).Verilog is very much compatible with C language.Its control flow keywords (if/else, for, while, case, etc.) are equivalent, and its operator precedence is compatible. So it's easy to work with verilog by knowing C language. And the main reason to choose verilog as HDL language is to learn new language.

## 2.3.ModelSim

ModelSim is a multi-language HDL simulation environment by Mentor Graphics, for simulation of hardware description languages such as VHDL, Verilog etc.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 ModelSimPE, ModelSimSE and ModelSimPE.ModelSimcan also be used with MATLAB/Simulink, using Link for ModelSim.

## 2.4.Double precision floating-point format

Double precision is a computer numbering format in which it uses two adjacent storage locations in computer memory. The IEEE 754 standard definition of a double is:

--Sign bit: 1 bit

--Exponent width: 11 bits

--Significant precision: 53 bits (52 explicitly stored)

The real values are assumed by a given 64-bit double-precision data with a given biased exponent e and a 52-bit fraction is:

$$-1^{sign} * \ 2^{\exp - 1023} * 1.mantissa$$

Using floating-point variables and mathematical functions (sin (), cos (), log (),

Exp (), sqrt () are the most popular ones) of double precision as opposed to single precision comes at the execution cost: the operations with the double precision are usually slower.

Typically, a floating-point operation takes two inputs with p bits of precision and returns a p-bit result. The ideal algorithm would compute this by first performing the operation exactly, and then rounding the result to p bit (using the current rounding mode).

The algorithms for various operations are given below. According to which the program coded in Verilog.

### 2.4.Implementation

A floating-point operation uses two inputs with p bits of precision. The ideal algorithm compute the result by first performing the operation, and after that rounding the result to p bit (using the rounding mode).As an example, the operation of the system is explained in next section with addition and subtraction operation.

#### 2.4.1.Block diagram

The double precision floating point adder / subtractor performs addition, subtraction operations. The block diagram of the current system (floating point unit double) is shown in figure (Fig.1)

For addition, the input operands are separated into their mantissa and exponent components, and the larger operand stored into mantissa large and exponent large, the other operand, i.e. the smaller operand populating mantissa small and exponent small. The comparison of the operands to find out which is larger operand only compares the exponents of the two operands, if the exponents are equal, the smaller operand should populate the mantissa large and exponent large registers. It is not an issue because the operands are compared to find the operand with the larger exponent, so that the operand with smaller exponent's mantissa can be right shifted before performing the addition. If the exponents are equal, the mantissas can be added without shifting
The block diagram can be explained more elaborately with two operands naming A and B. Such explanation is provided with the Algorithm part in forthcoming section.

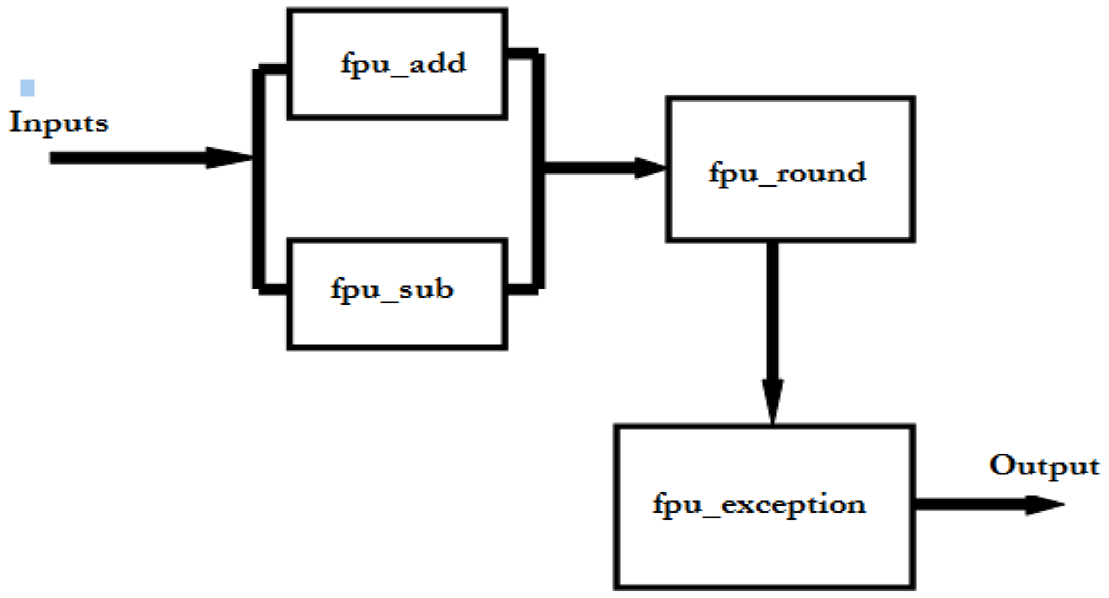


Fig.1Block Diagram of Floating Point Adder/Subtractor

### 2.4.2.Exceptions

Exception is an event that occurs when an operation on some particular operands have no outcome suitable for the reasonable application. That operation should signal one or more exceptions by invoking the default or, if explicitly requested, a language-defined alternate handling.

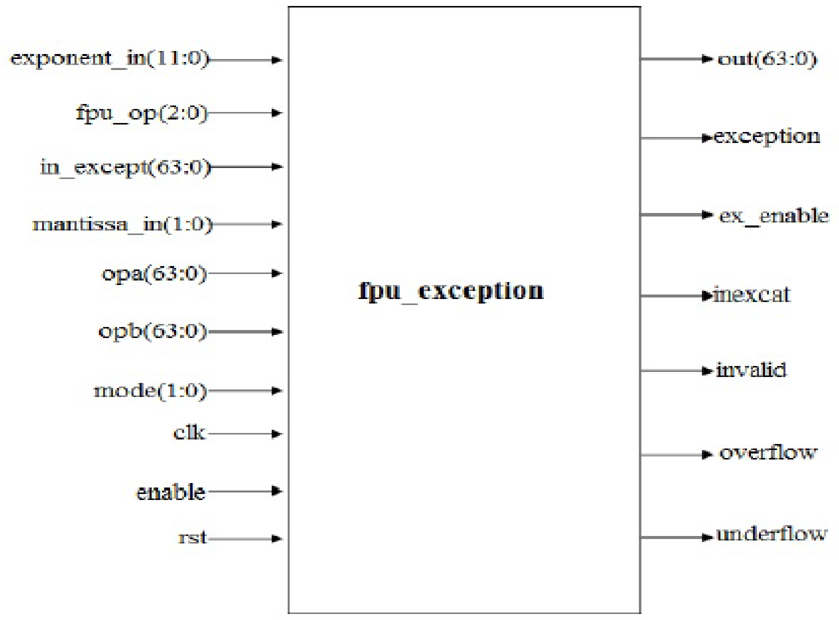


Fig.2 Exception Module

The exception signal will be asserted for some special cases, which are included with the exception algorithm. If the output is positive infinity, and the rounding mode is round to zero or round to negative infinity, then the output will be rounded down to the largest positive number. Likewise, if the output is negative infinity, and the rounding mode is round to zero or round to positive infinity, then the output will be rounded down to the largest negative number. The rounding of infinity occurs in the exceptions module, not in the rounding module. QNaN is defined as Quiet Not a Number. SNaN is defined as Signalling Not a Number. If either input is a SNaN, then the operation is invalid. The output in that case will be a QNaN. For all other invalid operations, the output will be a SNaN. If either input is a QNaN, the operation will not be performed, and the output will be a QNaN. The output in that case will be the same QNaN as the input QNaN. If both inputs are QNaNs, the output will be the QNaN in operand A.

## 2.5.Design

The algorithms to design the system are included for each operation.

### 2.5.1.Addition Algorithm

1. Feed two operands (say A and B), each 64 bits long, as the inputs.
2. The sign bit of operand A is the sign of the result.
3. Store the mantissa (bits 51-00) and exponent (62-52) of each operand.
4. Compare the 2 exponents and choose the largest exponent.
5. Find the difference of the 2 exponents (say S).
6. Write the mantissa of the operands as 1.mantissa.
7. Shift the mantissa of the smaller number to the right as many times as equal to the difference calculated in the step 4.
8. Add the 2 mantissas to obtain the final mantissa.

9. The result is obtained by combining sign, exponent and mantissa.

### 2.5.2.Subtraction Algorithm

1. Feed two operands (say A and B),each 64 bits long, as the inputs.
2. The sign bit of the result is 1 if the difference A-B is positive, else 0.
3. Store the mantissa (bits 51-00) and exponent (62-52) of each operand.
4. Compare the 2 exponents and choose the largest exponent.
5. Find the difference of the 2 exponents (say S).
6. Write the mantissa of the operands as 1.mantissa.
7. Shift the mantissa of the smaller number to the right as many times as equal to the difference calculated in the step 4.
8. Subtract the 2 mantissas to obtain the final mantissa.
9. The result is obtained by combining sign, exponent and mantissa.

### 2.5.3.Multiplication Algorithm

1. Calculate the sign = XOR of the sign bits of the operands.
2. The exponent is evaluated as: e1 + e21022
3. To evaluate the mantissa:

Consider the example given below: Say, M1= 101100101 M2= 100101100 Multiplication can be carried out in the following manner:

(a) Split the operands to smaller operands each 3 bits long.
For M1: Let M3=101, M4=100 and M5=101.
For M2: Let M6 = 100, M7 =101 and M8= 100.
(b) To evaluate the value of M1 * M8 (say the result is S1):

a. Find M3 * M8 (say A1), M4 * M8( say A2) and M5 * M8(say A3).
b. Add A3, A2 shifted by 3 bits and A1 shifted by 6 bits to get S1.

4. Find S2 (M1 * M7) and 3(M1* M6) following the procedure used to find S1.
5. Add S1, S2 shifted by 3 bits and S3 shifted by 6 bits.

### 2.5.4.Division Algorithm

1. Sign of the result = XOR of the sign bits of the operands.
2. The exponent is obtained using: e1 e2 + 1023
3. Division operation on the mantissas can be performed as:

Consider the operation:
Mantissa 1/ mantissa 2.
Initially, remainder = mantissa 1.

4. Perform remainder = remainder mantissa2.
5. If remainder > mantissa2, set the resultant bit (say mantissa 3) as 1, otherwise 0.
6. Perform the steps 4 and 5 until the process is completed.

### 2.5.5.Rounding Algorithm

1. Round to nearest: If first extra remainder bit is a 1, and the LSB of the mantissa is a 1 rounding is done. For rounding, rounding amount is added to the resultant mantissa.
2. Round to zero: No rounding is performed, unless the output is positive or negative infinity. The final output will be the largest positive or negative number in case of rounding in positive or negative infinity.
3. Round to positive infinity: The two extra remainder bits are checked, and if there is a 1 in either bit, and the sign bit is 0, then the rounding amount will be added to the resultant mantissa.
4. Round to negative infinity: Check the two extra remainder bits, and if there is a 1 in either bit, and the sign bit is a 1, then the rounding amount will be added to the resultant mantissa.

### 2.5.6.Exception Algorithm

The exception signal will be asserted for following special cases.

The exception signal will be asserted for given special cases.

1. Divide by 0: The result is infinity, positive or negative, depends upon the sign of operand A.
2. Divide 0 by 0: Result is SNaN, and the signal asserted for invalid.
3. Divide infinity by infinity: Result will be SNaN, and the invalid signal will be asserted.
4. Divide by infinity: Result will be 0, positive or negative; depending on the sign of operand A the underflow signal will be asserted.
5. Multiply 0 by infinity: Result is SNaN, and the signal will asserted as invalid.
6. Add, subtract, multiply, or divide overflow: Result will be infinity, and the overflow signal will be asserted.
7. Add, subtract, multiply, or divide underflow: Result is 0, and the underflow signal will assert.
8. Add positive infinity with the negative infinity: Result is SNaN, and the invalid signal asserted.
9. Subtract positive infinity from the positive infinity: Result is SNaN, and the invalid signal will be asserted.
10. Subtract negative infinity from negative infinity: Result is SNaN, and the invalid signal asserted.
11. Any one or both inputs are QNaN: Output is QNaN.
12. One or both inputs are SNaN: Output will be QNaN, and the invalid signal will be asserted.
13. If either of the two remainder bits is 1: Inexact signal is asserted.

### 2.5.7.Mode Algorithm

For some cases, the modes have to be varied. These have been include as a new module called mode. The conditions under this module include:

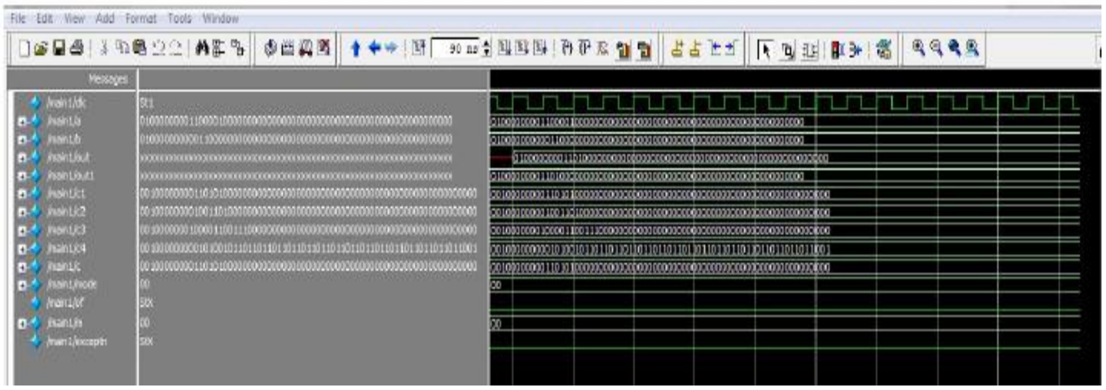
1. If subtraction of 2 numbers is involved where the second number is negative, the mode of operation has to be switched to subtraction.
2. If subtraction is the operation and the first operand is negative, then the mode is made addition.

3. If second operand is negative and the operation assigned is addition, then the mode has to be changed to subtraction.
4. If the first operand is negative when an operation of addition is called, change the mode to subtraction. In all other cases, the mode is applied as such.

The various modules like addition, subtraction, multiplication, division, mode, multiplexer, rounding and exception were separately done and compiled. They are finally called into the main program and compiled.

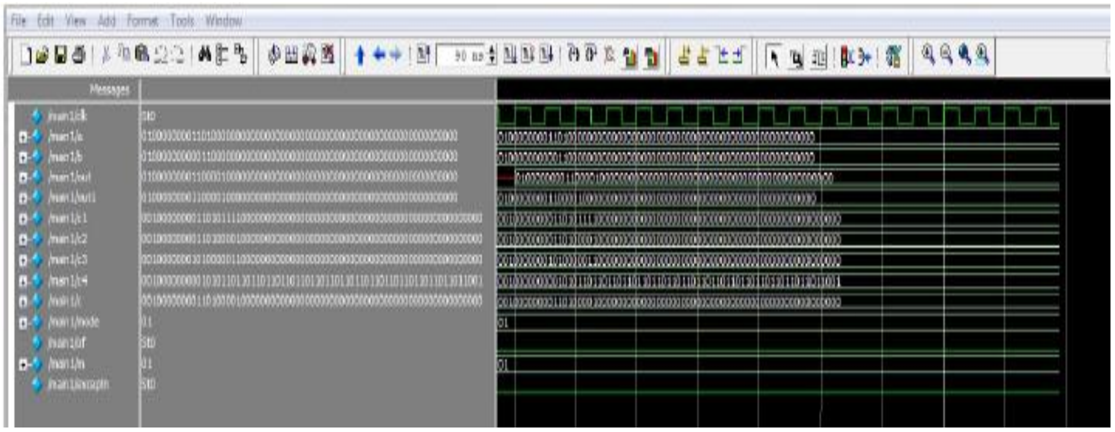
## 2.6.Results

The results were simulated using ModelSim. The obtained wave forms are given below.



```
/main1/clk      St1
/main1/a        [illegible]
/main1/b        [illegible]
/main1/out      [illegible]
/main1/out1     [illegible]
/main1/c1       [illegible]
/main1/c2       [illegible]
/main1/c3       [illegible]
/main1/c4       [illegible]
/main1/c        [illegible]
/main1/mode     00
/main1/of       StX
/main1/m        00
/main1/exceptn  StX
```

Fig.3.Simulated result of addition



```
/main1/clk      St0
/main1/a        [illegible]
/main1/b        [illegible]
/main1/out      [illegible]
/main1/out1     [illegible]
/main1/c1       [illegible]
/main1/c2       [illegible]
/main1/c3       [illegible]
/main1/c4       [illegible]
/main1/c        [illegible]
/main1/mode     01
/main1/of       St0
/main1/m        01
/main1/exceptn  St0
```

Fig.4.Simulated result of subtraction

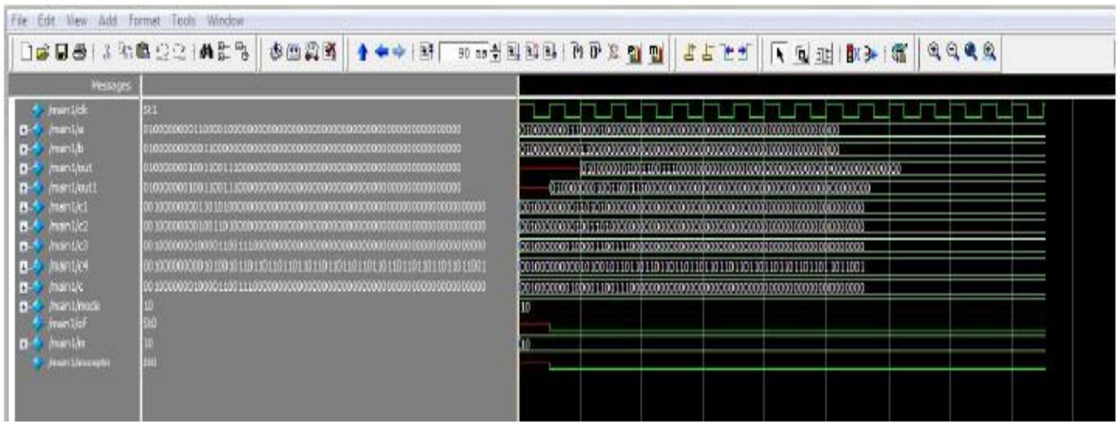


Fig.5.Simulated result of multiplication

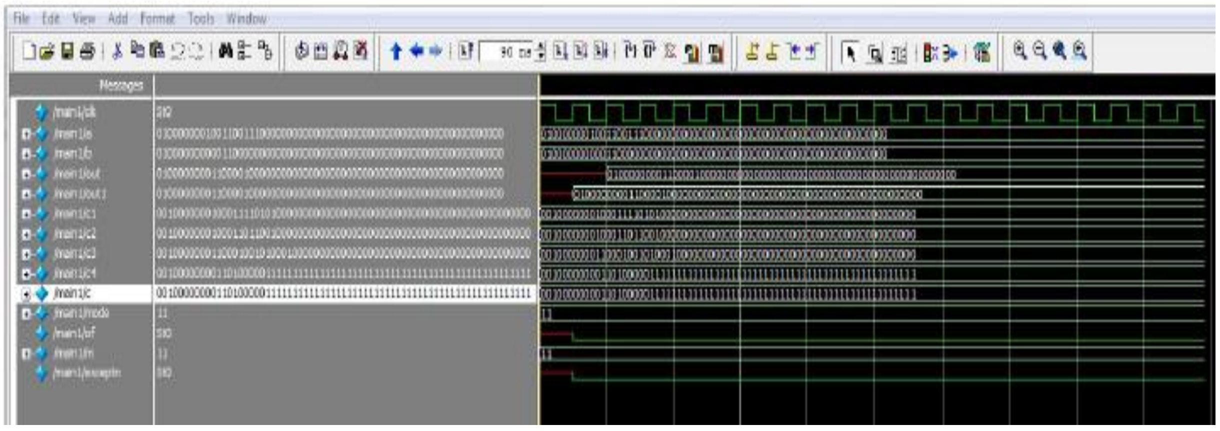


Fig.6.Simulated result of division

## 3.Conclusions

Double precision floating point core has been developed using verilog and simulated using ModelSim. The resulted simulations show an efficient double precision floating point arithmetic operation.

The project can be extended by including various other algorithms, say for calculating sine, cosine, tan functions and so on. The project has been implemented for double precision format which may be further extended using suitable algorithms, if available, for extended double and other formats. Similarly, faster algorithms may be utilized to provide a more efficient floating point core.

## ACKNOWLEDGEMENTS

This project was done as a curriculum work. I express my profound sense of gratitude and sincere thanks to my project guide Ass. Prof. Vijitha .S for her advice, encouragement, guidance and supervision for making this work a reality. And also would like to thank the Project co-

ordinator Mr.Vinod. G, Associate Professor, Department of Electronics and Communication Engineering, for the worthy advices towards improving the work. Here I conveying my heartfelt thanks to all who have been directly or indirectly involved in making this project. Above all I thank God Almighty for His bountiful blessings for the successful completion of this work.

## REFERENCES

[1] Charles Severance, 20th February 1998, "*An Interview with the Old Man of Floating-Point*". http://www.eecs.berkeley.edu/wkahan/ieee754status/754story.ht
[2] International Journal of Emerging Technology and Advanced Engineering, Volume 2, Issue 7, July 2012, FPGA based implementation of double precision floating point adder/ subtractor using verilog.
[3] IEEE XPLORE digital library.
[4] Implementation of IEEE-754 Addition and Subtraction for Floating Point Arithmetic Logic Unit, Volume 3, No. 1, 2010, pp. 131-140, International Transactions in Mathematical Sciences and Computer.

### AUTHOR

Aparna C V Graduated (B.Tech) in Electronics and Communication Engineering from Calicut University and currently pursuing M.Tech in VLSI and Embedded Systems from APJ Abdul Kalam Technological University, Kerala, India.

Mary Joseph received M.Tech Degree in Microwave and Radar from Cochin University of Science and Technology (CUSAT), Kochi, India, in 1997. Currently she is working as Associate Professor in M. A. College of Engineering, Kothamangalam. She has joined in M. A. College of Engineering in 1991 as Assistant Professor. In between she worked at Birla Institute of Science & Technology-Pilani's (BITS-PILANI) Dubai Campus for 9 years as Assistant Professor during 2000-2008. Her Research interests include Microstrip Antennas and Uniplanar Antennas.