High Speed Implementation of Floating Point Multiplier for Low Power Design Applications

Sachin Aralikatti and Reshma Nadaf

Abstract--- *The floating point arithmetic operations are giving excellent results for scientific applications like quantum computing, climatic condition predictions where high precision is obligatory. The internal hardware of floating point multiplication is most time and power consuming operation, so it can be improved by replacing the internal components by the delay efficient designs so that overall speed is increased. The other sub blocks like floating point addition and subtraction are also included with floating point multiplier for decimation in time FFT algorithm to prove its performance. The results observed on vertex-5 FPGA with xilinx isim 14.5v design suite.*

Keywords--- *Vertex-5 FPGA, Floating Point Multiplier, FFT Application.*

I. INTRODUCTION

DURING 1990's the important design parameter for system were memory and speed. The floating point units are omnipresent in all Pc's to supercomputer, along with that most of the languages will include it as part of data types. The scientific calculations including investigative venture, where vital precision needed is of major concern. So the binary floating point format expressing its importance by less size and performing high speed progression. The international acceptances of estimation criterions for this IEEE format are well suited for non proprietary conformations. Here confabulation on floating point will concentrate on the IEEE 754 floating-point standard because of its quickly increasing acceptance world wide and its usage in biometric, medical imaging, graphic accelerators, image processing and signal processing applications. The time and power consuming operation in floating point operation is floating point multiplier. Thus, after providing brief knowledge about floating point basics, we will introduced a more detailed analysis of efficient floating point operations in a particular application^{[1][2]}.

The IEEE carefully crafted the floating point representation of the real numbers under the sponsorship of intel for their microprocessor development from 1976. William kahan the professor at university of california was chief resource person for the development of floating point numbers for the future processor. Later all the hardware started to use same format. The newly introduced standard was made available in July 2008. Lets begin the introduction

of floating point number by introducing the disadvantage associated with positional number system. Consider the example where 3×2 (To the power 70) to be represented, it would be represented as binary 11 followed by 70 zeros. This type of expressing the number is memory consuming. So IEEE has come up with the standard format of representation:

$$Y = (-1)^S \cdot M \cdot 2^E$$

The symbol 'S' represent the sign bit, for positive number it is taken as zero and for negative number it is considered as one. The symbol 'M' signify the mantissa i.e fractional binary number. The Letter 'E' express the exponent value, the power to which the base will be raised.

The normal precision associated with standard binary formats will be single value additional for the mantissa. One additional part called hidden bit is considered as always present with mantissa. Normally the normalization is related to left justification of the obtained value, that is if the MSB bit of the result is zero that shifting to the left is carried on the result such that MSB bit should be one and it should be followed by binary point and then fractional part of the result. The IEEE has introduced format for binary inflated dot values under standard subsection numbered as 754-2008. This standard is universally accepted by all modern high performance computing machines. The IBM mainframe and CRAY vector machine are still following their standards for some processors^[2].

Single precision floating point number which require 4 bytes for representation, it is said that the C language which we use make use of this format for representing data of type float. The mantissa require 24 bits from the total 32 bits and 1 bit for sign. This format will provide the range in decimal as-1038 to +1038, and bias is 127. The steps involved in expressing the decimal number into single precision format:

1. Convert the given decimal number into binary form.
Decimal = 0.1015625
Binary = 0.0001101
Normalizing the binary value = 1.101×2^{-4} .
2. The mantissa is represented as:
101000000000000000000000.
3. The exponent is expressed as : $-4+127=123_{10} = 01111011_2$.
4. Sign bit as zero.
5. Finally single precision value is
:00111101110100000000000000000000.

Similarly double precision number is normally uses 8 byte of data, which include 52 bits for mantissa and eleven bits of exponent and single bit lumped for sign. In the compiling language this format is used to express the data type float. The

Sachin Aralikatti, M.Tech (Digital Electronics), SDM College of Engineering and Technology.

Reshma Nadaf, Assistant Professor, SDM College of Engineering and Technology, Dharwad.

DOI:10.9756/BJRCE.8213

range provided by this format in decimal is -10308 to +10308 and bias is taken as 1023. The representation of the double precision format is same as that of single precision with pre-described lengths. Double precision known as "extended precision format", which acquire 80 bits including hidden bit and provide the precision in the decimal upto 19 digits ,totally mantissa of 64 bits and exponent of 15 bits. Improving the precision/accuracy of the floating point notation will normally decreases the amount of associated round-off problem caused by partially obtained calculations. A Task of renewing the IEEE 754-2008 standard was began in 2000, Later finished, accepted in 2008. It consist of base-10 floating point representation, pair bytes floating point representations. Pair bytes consist of ten bits concording mantissa, 5 bit concording exponent, one bit affiliating sign. The higher necessary of precision is provided by IEEE extended precision representation standards so more precision and range requirement than basic format will be satisfied for particular applications. An extendable precision representation will give accuracy moreover exponent limits. An implementation can use defined parameter like base, precision and maximum value of exponent .The format does not required the representation to be with extendable precision standards. The format support the languages to provide a steps of specifying precision and maximum value of exponent for supporting base value. The format suggest the language and implementation recommends the extendable representation which is consisting a higher accuracy/precision than higher basic standards supported by every radix b. The IEEE introduce 5 rounding rules. Rounding to nearby value if linking with even rounding's that is if number comes halfway then it will be rounded with the closest number with an even (zero) LSB , This case will come across 50% during operations. The case is default condition with base-2 floating point numbers and delinquency for base-10 values. Rounding to nearby value if linking distantly against value 0, circulate that if the number comes with midway then it will be rounded to the closest value below or above. The declared statement taken as choice for base-10 numbers. Round toward zero, also considered as the truncation of the number method or directed rounding the number towards zero. Round toward plus infinity, also called as the ceiling approach or rounding of the number towards positive infinity .Round toward minus infinity, also considered as floor approach for given number or direct rounding towards negative infinity.

The IEEE defines five exception conditions where each one of will assume to a default number and it will be having corresponding status flag value which will be raised when the exception is introduced.

The 5 practicable peculiarity which are observed: Incapacitate behaviour like obtaining non-positive values to the power 0.5, returning to qNaN value by default are some of the common exceptions. Division of a number by zero value which will lead to the infinite value like 1/0 or logarithm of zero. Overflow exception will occur when number is very large to fit in the given space for particular format. Underflow exception condition will occur when obtained result is very small to be represented. Inexact form, that is returning to the correctly rounded output by default methods^[1].

II. FLOATING POINT MULTIPLIER BLOCK DIAGRAM

Figure 1: Block Diagram of Floating Point Multiplier

The floating point number format consist of mainly three fields, sign bit, exponent and mantissa of the given number. While performing the floating point multiplication operation on given two number, the operations involved are: 1. Sign calculation involve the xor operation on sign bit. 2. Exponent addition 3. Fraction multiplication involve the fixed point multiplier. Since the multiplier is the time and area consuming operation, designing the high speed building blocks will reduce the entire delay and area consumption.

a) 24 bit Binary Multiplier

The idea of introducing pipelining came from the concept of water flow through pipe with nonstop sending of water without allowing the water to come through pipe. This method will help in improving the speed for the applications like high speed processing systems. The knowledge of pipelining for combinational logical circuit comes with collections of storage components like registers where more amount of the parallelism will be obtained at the same time speed will be improved upto some extent. Since the multiplier involve the successive addition of partial products which is the time, area and power consuming process. Building such a building block with improved capability will definitely improve the entire performance of the system where it is used.

Figure 2: Block Diagram of 24 bit Binary Multiplier

The allocation of memory location for keeping partial products can be made by declaring a 2 dimensional memory locations, since we have usually represent two dimensional

memory in verilog as reg [k:0] ay [0:p]. The said instruction explain about array having (p+1) elements also every one elements will posses (k+1) bit locations. Therefore the mentioned variable signify storage elements. The described multiplier will perform its operation in the following way : Let the entire operation be carried under clock and reset , the intermediate results generated will be stored in the defined memory location and by doing the pipelined addition of the intermediate results will finally lead to the desired outputs.

While doing successive addition process of given inputs, suppose any one of the input is zero then no need to perform the operation the intermediate result will be zero^[11].

Suppose if one of the input is one bit then while multiplying shifted another input will be considered directly since in binary only two values can be either zero or one then multiplying with one will lead to same value. Next followed by the operation with one right shift. Using the conditional check method it is noticed that lets take one example where bitwise comparison is done, take two operands with some bit length and keep the one operand fixed and check conditionally bitwise from the left side of given input. If the input which is used for comparison having zero bit value then entire zero will be considered in the partial product similarly if bit value is one then necessary zeros will be added before and after and used as the partial product . Similarly for every iterations with one shift to the right will be taken an calculations are carried out. All this intermediate value will be stored in the memory will be added in the pipelined method .The addition will be taken at different pipeline stages and output will be obtained.

III. STEPS FOR OPERATION OF FLOATING POINT MULTIPLIER

Operation representing thirty-two bit inflated bit multiplier is depicted by performing the simple operation by taking two floating point number expressed in general format. Let us consider the two numbers to be:

$$X = A_1 \times B^{f1} \text{ and } Y = A_2 \times B^{f2}$$

Where A_1 and A_2 equals mantissa of operands.

B is the base for binary it is 2.

$f1$ and $f2$ equals exponent of operands.

Now performing the multiplication operation of both the inputs , $Z = X \cdot Y$

$$\begin{aligned} Z &= (A_1 \times B^{f1}) \cdot (A_2 \times B^{f2}) \\ &= (A_1 \cdot A_2) \cdot B^{(f1+f2)} \end{aligned}$$

By observing the above equation the observation can be made by performing the mantissa multiplication by fixed length multiplier architectures and the exponent addition is carried by any of the adder architecture. The steps to find the floating point multiplication are :

1. The new exponent is generated by adding the exponents of given operands.
2. The bias will be added twice, So bias should be subtracted once to compensate.
3. The mantissa multiplication is carried by the proposed 24 bit binary multiplier.

4. The normalization of the final result is done so the result will be in standard form then rounding of the result is carried out as per round to nearest rule, later final result will be obtained. 8 point DIT FFT application using floating point sub blocks.

The floating point adder and subtractor units are also developed along with floating point multiplier unit, and their performance is observed by using this units in an application,

Here we have used 8 point DIT FFT application using butterfly technique.

Figure 3: 8 point DIT-FFT Algorithm

The steps followed for the application are:

1. The 8 point DIT FFT algorithm using butterfly technique uses 3 stages, where at first stage 4 butterfly module will be used, for second stage two butterfly will be combined to form two butterfly and at the third stage second stage butterfly will be combined to form single butterfly stage.
2. The input taken will be in the bit reversed order and process will be continued.
3. The multiplication of the twiddle factor with input will be carried out by proposed floating point multiplier.
4. Addition and subtraction will be carried by floating point adder and subtractor.

IV. IMPLEMENTATION AND RESULTS

The overall aim of this presented paper is to implement faster floating point multiplier for high speed applications. Since the mantissa multiplication designed gives good improvement in speed but consume little more area than the existing model.

Whenever we come across the applications where speed is much needed, this floating point multiplier is good choice. The design is simulated and synthesized using Xilinx Synthesis tool (ISE 14.5) and targeted on vertex 5 FPGA. The above tables give detail device utilization summary and the delay in performing the operations. Comparing the proposed floating point unit with the existing one in terms of delay and area is done.

Device Utilization Summary (estimated values)				[-]
Logic Utilization	Used	Available	Utilization	
Number of Slices	541	6144	8%	
Number of Slice Flip Flops	8	12288	0%	
Number of 4 input LUTs	984	12288	8%	
Number of bonded IOBs	96	320	30%	

Device Utilization Summary				[-]
Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Latches	8	12,288	1%	
Number of 4 input LUTs	946	12,288	7%	
Number of occupied Slices	527	6,144	8%	
Number of Slices containing only related logic	527	527	100%	
Number of Slices containing unrelated logic	0	527	0%	
Total Number of 4 input LUTs	999	12,288	8%	
Number used as logic	946			
Number used as a route-thru	53			
Number of bonded IOBs	96	320	30%	
Average Fanout of Non-Clock Nets	2.84			

Figure 4: Device Utilization of Floating Point Multiplier

Device Utilization Summary					[-]
Slice Logic Utilization	Used	Available	Utilization	Note(s)	
Number of Slice Registers	1,105	69,120	1%		
Number used as Flip Flops	16				
Number used as Latches	1,030				
Number used as Latch-thrus	59				
Number of Slice LUTs	8,412	69,120	12%		
Number used as logic	8,208	69,120	11%		
Number using O6 output only	6,646				
Number using O5 output only	43				
Number using O5 and O6	1,519				
Number used as exclusive route-thru	204				
Number of route-thrus	251				
Number using O6 output only	245				
Number using O5 output only	4				
Number using O5 and O6	2				
Number of occupied Slices	3,661	17,280	21%		
Number of LUT Flip Flop pairs used	8,528				
Number with an unused Flip Flop	7,423	8,528	87%		
Number with an unused LUT	116	8,528	1%		
Number of fully used LUT-FF pairs	989	8,528	11%		
Number of unique control sets	1,041				
Number of slice register sites lost to control set restrictions	3,122	69,120	4%		
Number of bonded IOBs	581	640	90%		
IOB Flip Flops	2				
IOB Latches	1				
Number of BUFG/BUFGCTRLs	16	32	50%		
Number used as BUFGBs	16				
Average Fanout of Non-Clock Nets	4.28				

Figure 5: Device Utilization of 8 Point FFT Application

V. ADVANTAGES AND APPLICATIONS

The intended design obliterate less delay and improve performance of system. The hardware consumed is also less for the operation. The intended design provide good accuracy in computation. The power dissipation of circuitry is also less.

The application like signal processing, image processing,

scientific calculations,graphic acceleration and communication systems.

Table 1: Results of the Proposed Floating Point Multiplication

<i>Virtex 4 , xc4v1x15</i>	<i>Delay (ns)</i>	<i>No of slices</i>	<i>No of four i/p LUT's</i>
24 bit multiplication	14.081	76/6144(7%)	874/12288 (7%)
Floating point multiplication_32	17.247	541/6144 (8%)	984/12288 (8%)

Table 2: Results of the Existing Fast Floating Point Multiplication

<i>Virtex4 , xc4vlx15^[6]</i>	<i>Delay(ns)</i>	<i>No of slices</i>	<i>No of 4 i/p LUT's</i>
24 bit multiplication	16.316	1269/6144(20%)	2270/12288(18%)
Floating point multiplication ₃₂	18.783	1306/6144(21%)	2329/12288(18.9%)

Reshma Nadaf, finished B.E in Electronics and Communication Engineering from STJIT, Ranebennur in 2006, M.Tech in Digital Electronics from Gogte Institute of Technology, Belgaum in 2012. Currently working as Assistant Professor in SDM College of Engineering and Technology, Dharwad. She is having 10 years of teaching experience. Her fields of interest are Wireless sensor networks, Cryptography.

(E-mail: reshma.nadaf27@gmail.com)

Table 3: Results of the Existing FFT Application

<i>Spartan 6 , xc6slx100T 4fgg900</i>	<i>Logic Delay(ns)</i>	<i>Routing Delay (ns)</i>	<i>Total Delay (ns)</i>
FFT application using Radix 2 ⁴ architecture ^[7]	22.085	79.173	101.258
Proposed FFT application	20.693	53.803	74.796

VI. CONCLUSION AND FUTURE SCOPE

The proposed floating point multiplier is synthesized selecting vertex 5 FPGA with package xc4vlx15. It is observed the modules are efficient in delay and power. The 984 number of 4 input look-up tables are used constituting 8% of utilization from available resources. The total Number of bounded slices consumed are 541 with constitute 8% usage. The future use of this multiplier can be used to in Scientific calculations, Graphic acceleration, Communication systems.

REFERENCES

- [1] A. Peter, "Digital design An embedded system approach", Elsevier, 2014.
- [2] J. Bashkar, "Introduction to verilog", BS publication, 2nd edition.
- [3] S. Palnitkar, "Verilog HDL: a guide to digital design and synthesis", Prentice Hall Professional, Vol.1, 2003
- [4] S. Arish ans R.K. Sharma, "An efficient floating point multiplier design for high speed applications using Karatsuba algorithm and Urdhva-Tiryagbhyam algorithm", 2015 International Conference on Signal Processing and Communication (ICSC), Pp.303-308, 2015.
- [5] N.V. Sunesh and P. Sathishkumar, "Design and implementation of fast floating point multiplier unit", Pp.1-5.
- [6] A. Jain, B. Dash, A.K. Panda and M. Suresh, "FPGA design of a fast 32-bit floating point multiplier unit", 2012 International Conference on Devices, Circuits and Systems (ICDCS), Pp.545-547, 2012.
- [7] S. Badar and D.R. Dandekar, "High speed FFT processor design using radix- 4 pipelined architecture", 2015 International Conference on Industrial Instrumentation and Control (ICIC), Pp.1050-1055, 2015.
- [8] R. kishore, L. Boppana, S. Sourabh and Yenamachint-ala, "FPGA Implementation of Vedic Floating Point Multip-lier", 2015.
- [9] A. Mehta, C.B. Bidhul, S. Joseph and P. Jayakrishnan, "Implementation of single precision floating point multiplier using Karatsuba algorithm", 2013 International Conference on Green Computing, Communication and Conservation of Energy (ICGCE), Pp.254-256, 2013.
- [10] IEEE 754-2008, IEEE Standard For Floating-Point Arithmetic, 2008.
- [11] S. Aralikatti and R. Nadaf, "An Efficient Implementation of 32 bit Binary Multiplier for High Speed Design Applications", IJEMR, Vol.6, No.2, 2016.

Sachin Aralikatti, finished B.E in Electronics and Communication Engineering from VDRIT Haliyal in 2012, Currently pursuing M.Tech in Digital Electronics from SDM College of Engineering and Technology. His fields of interest are Floating point arithmetic circuits, Digital Design using HDL. (E-mail:aralikatti.sachin@gmail.com)