

Double Precision Floating Point Fft Processor using Vedic Mathematics



Anitha R

Abstract: *There should be rapid, efficient and simple process for every scenario now a day. To compute the N point DFT, Fast Fourier Transform (FFT) is a productive algorithm. It has great applications in communication, signal and image processing and instrumentation. In the implementation of FFT one of the challenges is the complex multiplications, so to make this process rapid and simple it's necessary for a multiplier to be fast and power efficient. To tackle this problem Karatsuba sutra and Nikhilam sutra are an efficient method of multiplication in Vedic Mathematics. This paper will present a design methodology of Double Precision Floating Point Fast Fourier Transform (FFT) Processor. The execution time and complexity can be reduced by the algorithm which is there in Vedic. The main aim is to make FFT Processor process rapid and simple by designing a multiplier which is fast and power efficient by using double precision floating point and Vedic Mathematics concepts.*

Keywords: DPFP, FFT, Vedic Mathematics, FPA, RCA, MSB, IEEE-754

I. INTRODUCTION

Fast Fourier Transform is mainly used in various applications such as signal processing applications, filtering applications and to solve various differential and difference equations. Basically, FFT will transform any time domain specifications to frequency domain specifications which helps to analyze some parameters like bandwidth, resonant peak, resonant frequency etc. Multiplication is one of the most commonly used one in Fast Fourier Transform. This is also effectively used in floating point algorithm (FPA) which is a crucial basic building block for many applications such as scientific, numeric and signal processing applications. The most common choice for many scientific computations is floating point number system due to its wide dynamic range feature. The standard for floating point is IEEE 754. The multiplication is the major core operation in a large number of scientific and signal processing computations among most of the floating point arithmetic operations. These applications aim at high performance and area-efficient implementation of FPA operation.

To improve the performance of floating point arithmetic lot of works have been carried out lately in general flow of floating point has been done using FPGA both in algorithm and as well as implementation level[1]. Floating point multipliers are complex which required larger area, so the power consumption also increase when compare to the fixed point multipliers. Accuracy of a floating point unit increases and becomes a major issue. Area, delay and power will increase as the precision increases. In the proposed architecture which uses the Karatsuba Vedic structure along with the truncated block multiplier method is implemented, which reduces the power and execution time, so that after processor gains high speed [2,5].

Most signal processing is now a day implemented on VLSI chip technology keeps on growing. These signal processing applications such as FIR, IIR, FFT, DFT operations not only having the computational demands but it consumes lot amount of energy. In most of the signal processing algorithms multiplication is the fundamental operation. Multipliers which consumes more power, area and has latency also [3,4,6]. So that proposed the multiplier design which consumes less area and high speed which in terms speed up the FFT processor.

The paper has been organized in the following sections

Section 2 – This gives brief about the previous work.

Section 3 – proposed system has been explained

Section 4 – the methodology of proposed system is clearly narrated.

Section 5 – details about the results of proposed work.

Section 6 – where concluded the work.

II. RELATED WORK

The multiplier block generally takes more time to perform and slowest element compared to the other operations in the system. In 2010, Nicola Petra et al. proposed truncated multipliers with a suitable compensation function which reduces the mean square error. This paper focused on variable correction truncated multipliers to reduce the complexity some of the partial products are discarded. They introduced sub optimal compensation function which is for best hardware implementation.[7] Ibrahim Abdelghany et al. in 2013 [8] proposed several truncated multiplier blocks. The partial product are produce using smaller blocks and divide and conquer method is used to adder their results. The first order in truncated multiplier uses $n/2$ sub blocks and in second order six $n/4$ sub blocks and follows. In 2015, Sivanandam and Kumar P, [9] designed redundant data bit detector based on Urdhva Triyakbhyam sutra which has less critical delay when compared to the Booth multiplier.

Manuscript published on November 30, 2019.

* Correspondence Author

Anitha R*, Professor School of Electronics Engineering, VIT University, Vellore-632014, Tamil Nadu, India

© The Authors. Published by Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP). This is an open access article under the CC-BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>)

Double Precision Floating Point Fft Processor using Vedic Mathematics

So this design leads to very less time delay. They implemented in floating point multiplier and analyzed its performance.

SushmaWadar et al. in 2016 they designed the multiplier using the Vedic sutra called Karastuba which using $O(n^{\log_2 3})$ for single digit multiplication. So the time complexity has been reduced. [10] Praveen Yadav et al [11] proposed 8*8 multiplier at 2018, where they focused on first 8 MSB of the carry output. They divided the architecture into two blocks as accurate and inaccurate block in which one to maintain the accuracy and the other one to reduces area, delay and power

III. PROPOSED SYSTEM

The project walkthrough is as follows

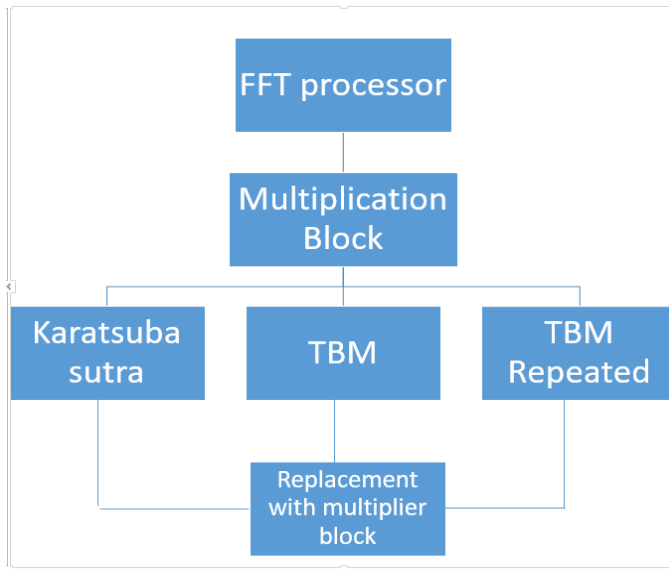


Figure 1: Block diagram of Project

The goals of this project consist of:

- Designing the FFT processor with less area consumption and which reduces complexity on the chip.
- Comparing different techniques which reduce the complete delay including combinational delay.
- Developing the best method which reduces number of blocks on the chip upon comparing those techniques
- Replacing the best technique with multiplication block for fast implementation of FFT processor.

Finally, the main aim is to make FFT Processor process rapid and simple by designing a multiplier which is fast and power efficient by using double precision floating point and Vedic Mathematics concepts which can be done by comparing several techniques. In this I would like to present a design methodology of Double Precision Floating Point Fast Fourier Transform (FFT) Processor Using Vedic Mathematics algorithm which reduces the execution time, complexity and ICs on the chip.

IV. METHODOLOGY

The design will be in such a way that the multiplication block in FFT processor will be replaced by any of the below

mentioned block. By comparing the design methodologies of three double precision floating point multipliers i.e.

1. Karatsuba Sutra multiplication method
2. Truncated block multiplication (TBM) method.
3. Truncated block multiplication reduced method

On comparing these techniques one which reduces hardware (indirectly the number of multiplications required) is to be implemented. This block which will actually reduce the hardware will be replaced by multiplier block in FFT processor to make it rapid and speed.

A. Karatsuba Sutra Implementation

Since implementing double double precision floating point numbers which need to multiply numbers in that form. So, in double precision floating point that have sign bit, exponent part and mantissa part. The design need to take sign bit and mantissa part while doing multiplication which is $(1 + 52) = 53$ bits because exponent bit just shifts which doesn't change the multiplication operation. The exponent bit doesn't change much in multiplication it just helps in shifting the bit position.

Karatsuba 3- partition method has been used here because the numbers of bits are more i.e. 53.

Now, general implementation of three partition Karatsuba sutra implementation for double precision floating point numbers is explained below

Inputs----- a (53 bit), b (53 bit)

Output----- c (106 bit)

Generally $c = a*b$

This operation of direct conventional multiplication requires one 53 bit unsigned multiplier.

But by using karatsuba sutra (3 partition splitting), the output c can be splitted

a ----- a2 a1 a0
 b ----- b2 b1 b0
 c ----- z5 z4 z3 z2 z1 z0

Here a0, a1, b0, b1 are 18 bit numbers and a2, b2 are 17 bit numbers

The input a can be represented as

$$a = a2*B^{2m} + a1*B^m + a0$$

The input b can be represented as

$$b = b2*B^{2m} + b1*B^m + b0$$

Where, B represents base representation of a number for example B = 10 for decimal numbers, B = 2 for binary numbers, B=8 for octal numbers and B = 16 for hexadecimal numbers.

The output can be calculated as

$$c = a*b = (a2*B^{2m} + a1*B^m + a0) * (b2*B^{2m} + b1*B^m + b0)$$

This can be represented on multiplication as

$$c = z2*B^{4m} + z1*B^{2m} + z0 + z5*B^{3m} + z4*B^{2m} + z3*B^m$$

Design equations:

$$z2 = a2*b2$$

$$z1 = a1*b1$$

$$z0 = a0*b0$$

$$z5 = a2*b1 + a1*b2$$

$$z4 = a2*b0 + a0*b2$$

$$z3 = a1*b0 + a0*b1$$

Which requires two unsigned multiplications for implementing the equations z5, z4, z3.

These can be reduced upon simplification as,

$$z5 = (a2+a1)*(b2+b1) - z1 - z2$$

$$z4 = (a2+a0)*(b2+b0) - z2 - z0$$

$$z3 = (a0+a1)*(b0+b1) - z0 - z1$$

In this simplified equations number of multiplications in each equation is reduced from two to one with some more additions and subtractions which doesn't require more area. Overall it can reduce the number of multiplications from nine to six.

This will reduce the hardware design and area on chip because multiplication requires more area than additions.

B. Truncated Block multiplication

The output demands only the MSB products, where the precision is less, so the truncated block multiplier can be implemented. The TBM which gives the less area, since some of the lower partial products can be discarded and further addition of them. In the Double precision multiplication, the 53*53 multiplication is taking place. In the end the resulted has to be truncated to get the 53 bit mantissa. It can be done by rounding the result of truncating the result.

For that purpose, 53-bit mantissa operands a and b have been partitioned as below. Now, general implementation of TBM can be explained below. The block diagram is shown in the figure 2 Normal Truncated block multiplication

Inputs----- A (53 bit), B (53 bit)

Output----- C (106 bit)

Generally $C = A*B$

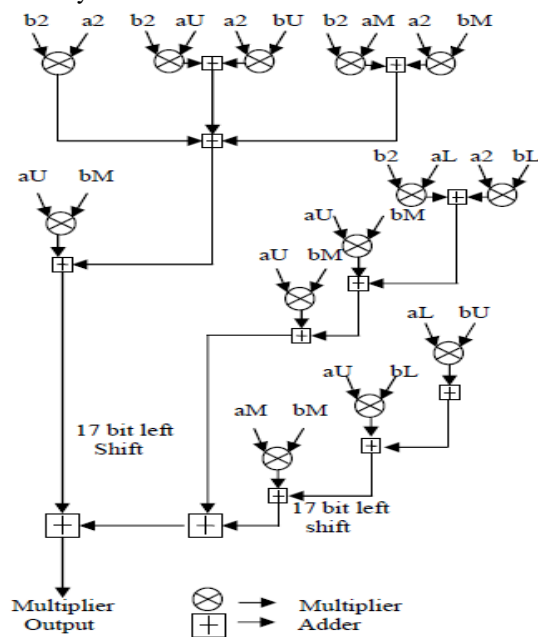


Figure 2: block diagram of TBM [9]

The splitting can be done as

$$A =$$

a4 (2 bits)	a3 (17 bits)	a2 (17 bits)	a1 (17 bits)
B =			
b4 (2 bits)	b3 (17 bits)	b2 (17 bits)	b1 (17 bits)

But by using TBM, the inputs A, B and output C can be spitted

$$A \text{ ----- } a4 \ a3 \ a2 \ a1$$

$$B \text{ ----- } b4 \ b3 \ b2 \ b1$$

$$C \text{ ----- } c7 \ c6 \ c5 \ c4 \ c3 \ c2 \ c1 \ c0$$

Here a1, a2, a3, b1, b2, b3 are 17 bit numbers and a4, b4 are 2 bit numbers

The input a can be represented as

$$A = a4*B^{3m} + a3*B^{2m} + a2*B^m + a1$$

The input b can be represented as

$$B = b4*B^{3m} + b3*B^{2m} + b2*B^m + b1$$

Where, B represents base representation of a number for example B = 10 for decimal numbers, B = 2 for binary numbers, B=8 for octal numbers and B = 16 for hexadecimal numbers.

The output can be calculated as

$$C = A*B \\ = (a4*B^{3m} + a3*B^{2m} + a2*B^m + a1) * (b4*B^{3m} + b3*B^{2m} + b2*B^m + b1)$$

This can be represented on multiplication as

$$C = c7*B^{6m} + c6*B^{5m} + c5*B^{4m} + c4*B^{3m} + c3*B^{2m} + c2*B^m + c1.$$

Design equations:

$$c7 = a4*b4$$

$$c6 = a4*b3 + a3*b4$$

$$c5 = a4*b2 + a3*b3 + a2*b4$$

$$c4 = a4*b1 + a3*b2 + a2*b3 + a1*b4$$

$$c3 = a3*b1 + a2*b2 + a1*b3$$

$$c2 = a2*b1 + a1*b2$$

$$c1 = a1*b1$$

Finally, the output C can be obtained by implementing all the design equations and then properly arranging them to obtain multiplication output.

V. RESULTS

The karatsuba sutra 53X53 implementation can be proceeded as follows Here, the design uses three partition karatsubasutra technique by splitting the two inputs a and b into three equal halves that means a will be converted into three numbers and similarly for b also. It has been designed all the required design parameters z0, z1, z2, z3, z4 and z5. For getting proper output z it need to pad all the design elements in such a way that the representation of the number should be same. The simulation waveform for 53 bit Karastuba Sutra, TBM increased RCA and TBM reduced RCA is shown in Figure 3, 4, and 5 respectively. The TBM 53X53 implementation can be proceeded as follows Here, the partitioning the two inputs a and b into four halves that means a will be converted into four numbers in which three are of 17-bit numbers and one 2-bit number and similarly for b also. Here, there are two methods of implementing these design equations based on usage of ripple carry adder For example consider the design parameter c4

$c4 = a4*b1 + a3*b2 + a2*b3 + a1*b4$, this parameter can be rewritten as

$$c4 = d9 + d8 + d7 + d6$$

Now output for 53-bit Karatsuba sutra is obtained.

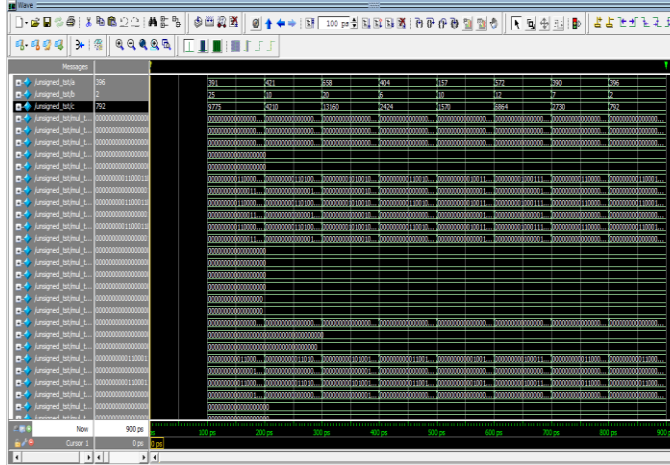


Figure 3: Simulation result of 53X53 karatsuba sutra First Method of using this RCA block is by increased number of blocks means here to implement this c7 there using RCA X34, RCA X35 and RCA X36. Now output for 53X53 TBM by using increased RCA blocks is obtained. Here first d6 and d7 are taken and they are sent into 34 bit ripple carry adder block resulting in sum and carry. After concatenating these two the resulting output with d8 will be sent to 35 bit ripple carry adder block which results in both sum and carry. Upon proper concatenation it will get the result and this result with d9 will be sent to 36 bit ripple carry adder block resulting in sum and carry.

By concatenating both sum and carry of the resulting output will get the design parameter c4.

The code snippet for this is as follows

```
rca_34bit g4(p2,cout4,d7,d6,cin1);
assign z4 = {cout4,p2};
rca_35bit q5(p3,cout5,z4,{1'b0,d8},cin1);
assign z5 = {cout5,p3};
rca_36bit g6(p4,cout6,z5,{2'b0,d9},cin1);
assign c4 = {cout6,p4};
```

Second Method of using this RCA block is by repeated usage of blocks means here to implement this c7 can use three RCA X36 blocks. Now output for 53X53 TBM by using repeated RCA blocks is obtained.

Here first d6 and d7 are taken and they are sent into 36 bit ripple carry adder block resulting in sum and carry. After concatenating these two the resulting output with d8 will be sent again to 36 bit ripple carry adder block which results in both sum and carry. Upon proper concatenation there will get the result and this result with d9 will be sent to 36 bit ripple carry adder block resulting in sum and carry.

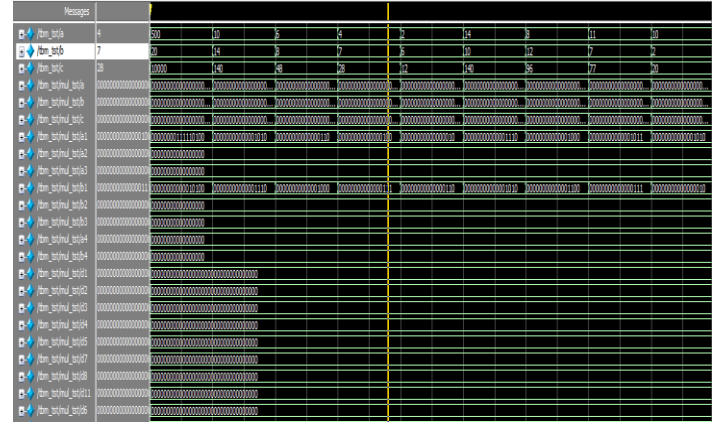


Figure 42: Simulation result of TBM increased RCA block

By concatenating both sum and carry of the resulting output can get the design parameter c4.

The extra manipulation which need to do here is to compensate the usage of same type of RCA block which need to use 1'b0 means one bit binary zero where ever necessary to increase the length of inputs. It has to use as many number of zeros required. Adding number of zeros at MSB side does not give any effect to the required number. In this way can manage using same block many times The code snippet for repeated block usage is as follows

```
rca_36bit g4(h3,cout4,{2'b0,d7},{2,b0,d6},cin1);
assign p2 = h3[33:0];
assign z4 = {cout4,p2};
rca_36bit q5(h4,cout5,{1'b0,z4},{2'b0,d8},cin1);
assign p3 = h4[34:0];
assign z5 = {cout5,p3};
rca_36bit g6(p4,cout6,z5,{2'b0,d9},cin1);
assign c4 = {cout6,p4};
```

By using Xilinx software, it has to be calculated the total delay which includes all combinational delays such as propagation delay for the proposed three double precision floating point multipliers. Out of these three the one which has less delay will be implemented. These delays will vary with the type of board that are using.

On comparing all the three design methodologies discussed above, the method which has least delay when compared to others is **Karatsuba sutra** which has least delay on **Artix 7, xc7a100t-3-csg324** board with delay of **11.695ns**. So, the sutra to be implemented is **Karatsuba sutra** on **Artix 7, xc7a100t-3-csg324** board. Same implemented on the ALTERA DE2-115 Board and the output shown in figure. 6. The delays for TBM with repeated RCA blocks using Xilinx is tabulated below. Table 1, 2 and 3 showing the delay of Karatsuba sutra, TBM increased RCA and TBM repeated RCA respectively.

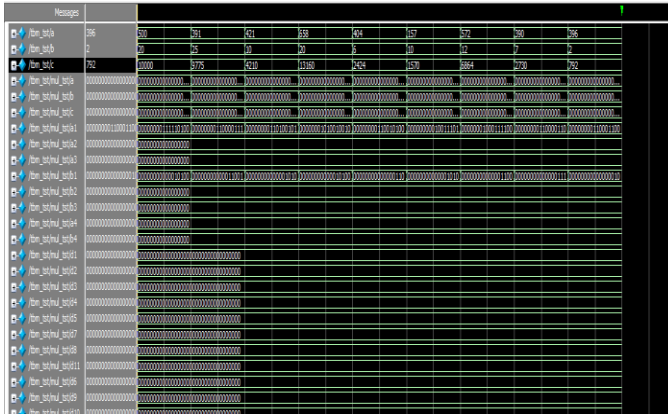


Figure 53: Simulation result of TBM repeated RCA block

This is implemented for

Inputs A = 11101110100 = 190810

B = 10000001111 = 103910

Output C = 1;1110;0011;1111;1100;1100

The hardware block or the Register Transistor Logic for Karatsuba sutra is shown in the figure 7.

The delays for Karatsuba Sutra using Xilinx is tabulated below

Table 1: Delay for karatsuba sutra

Type of method	Hardware FPGA board	Amount of delay
53 bit (KBS)	Spartan 3, xc3s50-5-pq208, speed 1	49.756ns
	Spartan 6 Low Power, xc6slx41-1L-tqg144	34.417ns
	Virtex 4, xc4vfx12-12-sf363	22.019ns
	Artix 7, xc7a100t-3-csg324	11.695ns
	Virtex 5, xc5vlx20t-2-ff323	16.907ns

Table 2: Delay for TBM increased RCA

Type of method	Hardware FPGA board	Amount of delay
53 bit (TBM)	Spartan 3, xc3s50-5-pq208, speed 1	59.253ns
	Spartan 6 Low Power, xc6slx41-1L-tqg144	52.186ns
	Virtex 4, xc4vfx12-12-sf363	28.842ns

	Artix 7, xc7a100t-3-csg324	17.852ns
	Virtex 5, xc5vlx20t-2-ff323	18.438ns

Table 3: Delay for TBM repeated RCA

Type of method	Hardware FPGA board	Amount of delay
53 (TBMR)	Virtex 4, xc4vfx12-12-sf363	29.207ns
	Spartan 3, xc3s50-5-pq208, speed 1	68.445ns
	Spartan 6 Low Power, xc6slx41-1L-tqg144	50.640ns
	Artix 7, xc7a100t-3-csg324	16.770ns
	Virtex 5, xc5vlx20t-2-ff323	19.229ns

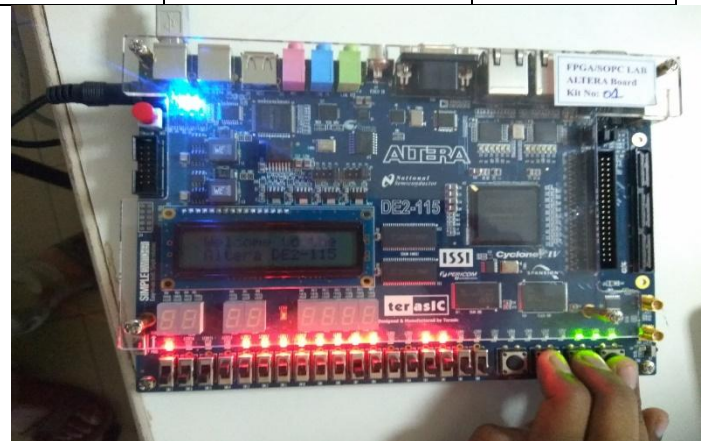


Figure 6: Hardware Implementation

VI. CONCLUSION

Thus finally the fast way to implement multiplication block is found to be Karatsuba sutra by using Vedic mathematics concept and double precision floating point concepts. This is predicted by comparing various multiplication techniques and finally sorted out the one which has less delay that is found to be Karatsuba sutra.

Also advancements can be made to reduce power consumption by extending this concept in future.

These can be replaced by multiplier block in FFT processor so as to reduce delay and number of ICs to be fabricated on chip. Since in real world it has to be considered power consumption effect also. So, this concept can be extended to do this. Using the Vedic mathematics concept is believed to reduce delay and complexity while fabrication.

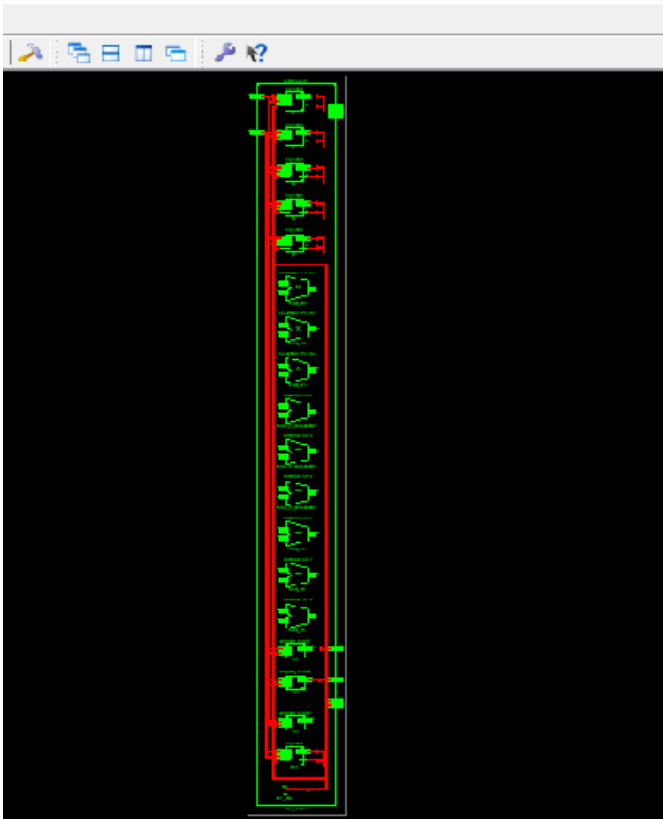


Figure 7: RTL view of Karastuba sutra

REFERENCES

1. Area-efficient architectures for double precision multiplier on FPGA, with run-time-reconfigurable dual single precision support, Manish Kumar Jaiswal^a, Ray C.C. Cheung, Department of Electronic Engineering, City University of Hong Kong, Hong Kong, *Microelectronics Journal* 44 (2013) 421–430.
2. Arish S, R.K.Sharma.” An efficient floating point multiplier design for high speed applications using Karatsuba algorithm and Urdhva-Tiryagbhyam algorithm”, *International Conference on Signal Processing and Communication (ICSC)*, 2015.
3. P. Belanovic, M. Leeser, A library of parameterized floating-point modules and their use, in: *Twelfth International Conference on Field-Programmable Logic and Applications (FPL-02)*, Springer-Verlag, London, UK, September 2002, pp. 657–666.
4. M. Huang, L. Wang, T. El-Ghazawi, Accelerating double precision floating-point Hessenberg reduction on FPGA and multicore architectures, in: *Symposium on Application Accelerators in High Performance Computing (SAAHPC’10)*, July 2010.
5. S.K. Venishetti, A. Akoglu, A highly parallel FPGA based IEEE-754 compliant double-precision binary floating-point multiplication algorithm, in: *International Conference on Field-Programmable Technology (ICFPT 2007)*, December 2007, pp. 145–152.
6. K. Underwood, FPGAs vs. CPUs: trends in peak floating-point performance, in: *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays ser. FPGA ’04*, ACM, New York, NY, USA, 2004, pp. 171–180.
7. Nicola Petra, Davide De Caro, Valeria Garofalo, Ettore Napoli, and Antonio G. M. Strollo, ”Truncated Binary Multipliers With Variable Correction and Minimum Mean Square Error” *IEEE Transactions on Circuits and Systems I*, 2010.
8. Ibrahim Abdelghany, Wajeb Saab, Tarek Sakakini, Abdul, Amir Yassine, Ali Chehab, Ayman Kayssi, Imad H. Elhaji” Energy-Efficient Truncated Multipliers”, *4th Annual International*

- Conference on Energy Aware Computing Systems and Applications (ICEAC), 2013
9. Sivanandam K, Kumar P,” Run Time Reconfigurable Modified Vedic Multiplier for High Speed Multimedia Applications,” *IEEE 9th International Conference on Intelligent Systems and Control (ISCO)*, 2015.
10. SushmaWadar, YashwantVithalraoChavan,AvinashPatil, “Improved algorithm for floating point multiplication”, *International conference on Signal Processing, Communication, Power and Embedded System (SCOPE5)*, 2016
11. Praveen Yadav ; Anirudha Pandey ; Manikantta Reddy K ; K.J. Ravi Prasad ; M.H. Vasantha ; Y.B. Nithin Kumar,” Low Power Approximate Multipliers With Truncated Carry Propagation for LSBs”, *IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2018.
12. M.K. Jaiswal, N. Chandrachoodan, FPGA-based high-performance and scalable block LU decomposition architecture, *IEEE Trans. Comput.* 61 (2012) 60–62.
13. G. Zhi, N. Walid, V. Frank, V. Kees, A quantitative analysis of the speedup factors of FPGAs over processors, in: *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays (FPGA-04)*, ACM, New York, NY, USA, 2004, pp. 162–170.
14. H. Parizi, A. Niktash, A. Kamalizad, N. Bagherzadeh, A reconfigurable architecture for wireless communication systems, in: *Third International Conference on Information Technology: New Generations*, vol. 0, 2006, pp. 250–255.
15. S. Kestur, J.D. Davis, E.S. Chung, Towards a universal FPGA matrix-vector multiplication architecture, in: *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM-2012)*, April 2012, pp. 9–16.
16. IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985, 1985.
17. IEEE Standard for Floating-Point Arithmetic, Technical Report, August 2008.
18. S.F. Anderson, J.G. Earle, R.E. Goldschmidt, D.M. Powers, The IBM system/360 model 91: floating-point execution unit, *IBM J. Res. Dev.* 11 (January) (1967)
19. R. Strzodka, D. Goddeke,” Pipelined mixed precision algorithms on FPGAs for fast and accurate PDE solvers from low precision components, in: *Fourteenth Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM ’06*, 2006, pp. 259–270.
20. D.W. Sweeney, An analysis of floating-point addition, *IBM Syst. J.* 4 (March) (1965) 31
21. K.S. Hemmert, K.D. Underwood, Open source high performance floating-point modules, in: *Fourteenth Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM-06)*, April 2006, pp. 349–350.
22. G. Govindu, L. Zhuo, S. Choi, V. Prasanna, Analysis of high-performance floating-point arithmetic on FPGAs, in: *Proceedings of 18th International Parallel and Distributed Processing Symposium, IEEE*, 2004, pp. 149–156.
23. X. Wang, M. Leeser, VFloat: a variable precision fixed- and floating-point library for reconfigurable hardware, *ACM Trans. Reconfigurable Technol. Syst.* 3 (September) (2010).
24. K.S. Hemmert, K.D. Underwood, Fast, efficient floating point adders and multipliers for FPGAs, *ACM Trans. Reconfigurable Technol. Syst.* 3 (September (3)) (2010).
25. G. Lienhart, A. Kugel, R. Manner, Using floating-point arithmetic on FPGAs to accelerate scientific N-body simulations, in: *Tenth Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM’02)*, IEEE Computer Society, 2002.
26. O. Storaasli, R.C. Singletary, S. Brown, Scientific computation on a NASA reconfigurable hypercomputer, in: *MAPLD International Conference*, September 2002.