Biosc.Biotech.Res.Comm. Special Issue Vol 13 No 13 (2020) Pp-34-44

BBRC
Bioscience Biotechnology Research Communications

Design of 128-bit Complex Number Multipliers for Co-Processor

Subodh Kumar Panda[1], Rekha P[2] and Bindu S[3]
Department of Electronics & Communication Engineering, BNM Institute of Technology, Visvesvaraya Technological University, Karnataka, 560070 India

ABSTRACT

Multipliers are the backbone of high-performance computing systems such as Microprocessors and Digital signal processors. Multipliers require more hardware resources and processing time, hence they are the slowest elements in the system. Multipliers are mainly used in today's high-end Digital Signal Processors and they occupy a larger chip area because of their inherent internal circuit complexity. Present-day co-processors are designed to support different size computations to achieve high performance. Researchers have worked on signed and complex number multipliers used in co-processors of 64 bit and below. There is a scope for designing a higher bit complex number multiplier to achieve higher performance. In this context, we proposed to design of 128-bit complex number multiplier of various architectures such as Booth Multiplier, Modified Booth Multiplier, Urdhva Multiplier and Nikhilam Multiplier using ModelSim SE 6.4 and Xilinx Vivado. In this work, various architectures such as Booth Multiplier, Modified Booth Multiplier, Urdhva Multiplier and Nikhilam Multiplier for 8-bit, 16- bit, 32-bit, 64-bit and 128-bit designed using Verilog for complex number multiplication. Nikhilam is one of the sutras of Vedic mathematics which is chosen for the implementation of complex number multiplier for area reduction. Synthesis reports are generated using the Xilinx Vivado tool for speed and power comparison. From the comparison, we have observed that Booth, Modified Booth, Urdhva, and Nikhilam occupy an area of 324.32%, 292.79%, 56.36%, and 46.70% respectively for 128-bit implementations. Among all the implemented methods, the Nikhilam method for complex number multiplier occupies very less area i.e. only 46.70% on-chip area.

KEY WORDS: COMPLEX, CO-PROCESSORS, MULTIPLIERS,NIKHILAM, URDHVA

INTRODUCTION

Basic arithmetic operators – adders, subtractors and multipliers are the core hardware sub-blocks of any computational engine. It is a well-known fact that out of these three units, the multiplier is the most area-hungry unit as it has to deal with a lot of internal operations. Multiplication operation is very important for an arithmetic operation like correlation & convolution as it has to perform information extraction from images, frequency analysis, image processing, etc. The ancient multipliers used the conventional method of repetitive additions to calculate the product. If this conventional design of a typical multiplier is to be implemented, it requires around 200 full adders, which is extremely area and power inefficient. Many multiplication methods have been proposed and experimented with, to obtain the most efficient architecture with efficient parameters such as area, speed and power for the multiplier (P Subramani, et al. 2019).

The most popular methods of multiplication consist of Booth's algorithm, Modified Booth's algorithm, Braun multiplication, and Wallace tree multiplication (M Gudhimetla et al.,2017; Soniya, S Kumar et al.,2013). Though these methods provide better speeds, the computations involved are too complex, that they increase the on-chip area consumption. The Indian Mathematics, well known as Vedic Mathematics was revisited by Rupanagudi (Huddar S.R. et al.,2013) and implemented a Vedic Mathematics multiplier on FPGA. Since then, several authors have been implementing Vedic

ARTICLE INFORMATION

*Corresponding Author: subodhpanda2013@gmail.com
Received 1th Oct 2020 Accepted after revision 27th Dec 2020
Print ISSN: 0974-6455 Online ISSN: 2321-4007 CODEN: BBRCBA

Thomson Reuters ISI Web of Science Clarivate Analytics USA and Crossref Indexed Journal*

Clarivate Analytics   crossref

*NAAS Journal Score 2020 (4.31)
A Society of Science and Nature Publication, Bhopal India 2020. All rights reserved.
Online Contents Available at: http//www.bbrc.in/
Doi: http://dx.doi.org/10.21786/bbrc/13.13/6*

Mathematics based multipliers in several applications related to the fields of communication, cryptography and DSPs (S. R. Rupanagudi et al.,2014; S. Rao Rupanagudi et al., 2019).

Related Work: As per the survey conducted, there are numerous papers available that showcase different architectures of multipliers that can be implemented on a chip. The related work shows that the efforts were put to improve the parameters such as area and power to be reduced and speed to be increased. Meanwhile, there are few drawbacks in the existing work done. For example, the offset binary code (OBC) along with the distributed arithmetic (DA) method, a multiplier is designed (A. P. Pascual et. al.,1999). The drawback is that this method is more complex and the area occupied is higher. A faster multiplier is developed using Wen-Chang's Modified Booth Encoder (MBE) (Razaidi Hussin et. al., 2008). The disadvantage over here is MBE is not the smallest scheme and hence size is larger which occupies more area.

Vedic Multiplier of 8-bit is implemented and the propagation delay parameter is enhanced compared to an array, Brawn, Modified Booth and Wallace tree Multiplier (Pavan Kumar U.C.S et al., 2013). Vedic real Multipliers are designed using Urdhva Sutra for 32 x 32-bit complex number multiplier. Here, a comparison between path delay and power consumption is done for the Booth complex multiplier and hence observed that Vedic is good which has the least power consumption and path delay (K.Deergha Rao et al.,2016). Also, multipliers are designed using Vedic mathematics sutras such as Urdhva and Nikhilam. These methods have used modified full adders and improved the speed parameter (Savita Patil et al.,2014). A 32-bit complex multiplier is designed using the Vedic algorithm and a comparison of its parameters power and delay is carried out with a lower bit multipliers such as 8bit and 16bit (Prof S. B. Somani et al.,2016; Ankush Nikam et al.,2015).

Also, there are 32 x 32-bit multipliers implemented for signed numbers using Urdhva and Nikhilam sutras. The propagation delay of these multipliers is compared and the outcome of the result says that the Urdhva multiplier is faster for lower bit numbers whereas the Nikhilam multiplier is faster for larger bit numbers (Nikhil R. Mistri et al.,2016). These Vedic multipliers are designed for a maximum of 64 bit signed numbers and maximum 32-bit complex numbers in the existing work and improved combinational delay and power (Manjunath et al.,2015; Sai Venkatramana Prasada G S et al.,2018). The related work consists of the multipliers designed for signed and complex numbers using conventional methods like Booth and Modified Booth algorithms and also using Vedic sutras for the lower bits i.e. 32bit or 64 bit and are resource expensive.

From the related work, we can observe that the various multipliers are designed using different approaches and methods to enhance the parameters such as area, speed and power. Many have used Vedic mathematics to improve the performance of Complex Number Multiplier. There is a scope for designing a higher bit complex number multiplier to reduce the on-chip area consumption. In this context, we have proposed to design a128 bit complex number multiplier of various architectures such as Booth, Modified Booth, Urdhva, and Nikhilam Multiplier using ModelSim SE 6.4 and Xilinx Vivado.

METHODOLOGY

Complex Multipliers: The logic for using these multipliers as complex multipliers is shown in figure 1. If (a+ib) is the first complex number and (c+id) is the second complex number, then the product is obtained as shown in Eq. 1.

$$(a+ib)\ .\ (c+id) = ac + iad + ibc + i^2db \ \dots\ (1)$$

As i^2 is -1, the above equation becomes

$$(a+ib)\ .\ (c+id) = ac + iad + ibc + (-1)\ db \quad (2)$$

Taking 'I' in common, the final equation (3) becomes

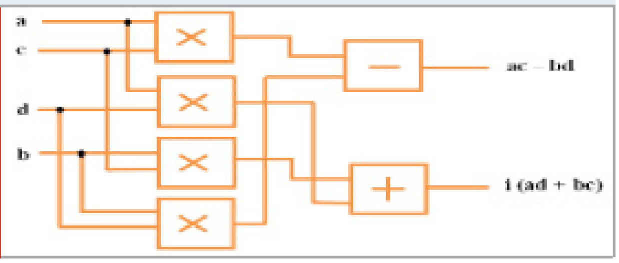
$$(a+ib)\ .\ (c+id) = (ac - db) + i\ (ad + bc)\ \dots(3)$$

Thus, the real part of the product is (ac-db) and the imaginary part is (ad+bc)
The multipliers designed for complex numbers using four methods are,

1. Booth Complex Number Multiplier
2. Modified Booth Complex Number Multiplier
3. Urdhva Complex Number Multiplier
4. Nikhilam Complex Number Multiplier

To develop a 128-bit complex number multiplier, initially, an 8-bit code is implemented using Verilog for various methods and it is used to describe 16, 32, 64, and 128-bit code. The design is simulated and synthesized using ModelSim SE 6.4 and Xilinx Vivado respectively.

Figure 1: Basic block diagram of complex number multiplier



3.1 Booth Complex Number Multiplier

The Booth multiplication algorithm is used to perform multiplication operation between two's complement of signed binary numbers. The booth algorithm is a serial computation method as it depends on the previous iteration value to compute the next steps in the procedure. The logic flow of the algorithm is as shown below. Considering 'X' as Multiplicand, 'Y' as Multiplier, and 'Z' as the output, Booth's algorithm works as per the table I. A zero is appended to the LSB of two's complement of Multiplier 'Y'. Consider Yi and Yi+1 from LSB to MSB pairwise and follow the Table I i.e. if the bit pair is 00 and 11 then do arithmetic right shift by 1. If the bit pair is 01 do +X and ASR by 1 and if 10 then do –X and ASR by 1. Continue the procedure until the last iteration.

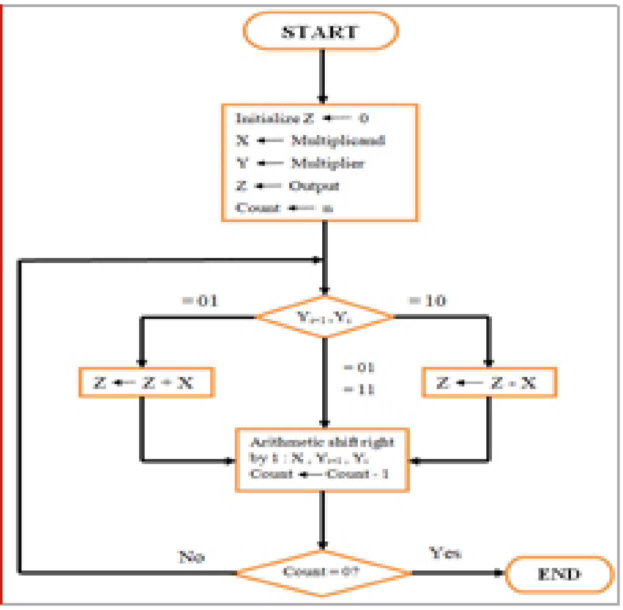


Table 1. Logic of Booth Multiplier

Y_{i+1}	Y_i	Cases
0	0	Arithmetic shift right (ASR) Z by 1
1	1	Arithmetic shift right (ASR) Z by 1
0	1	Z+X and ASR sum by 1
1	0	Z-X and ASR difference by 1

Functional flow of the Booth Algorithm:

A complex number multiplier of 128 bit is designed and in figure 2 the block diagram of the 128-bit complex number is as shown.

3.2. Modified Booth Complex Number Multiplier: The Modified Booth algorithm in two's complement multiplies the signed binary numbers. But, in Modified Booth Multiplier pair of 3-bits is considered from LSB to MSB of the multiplier operand and then the logic is applied to compute the operation. The number of iterations is reduced in the Modified Booth algorithm when compared to the Booth algorithm and hence computation time is saved. In figure 3, the general computation of multiplication using the Modified Booth algorithm is shown. Considering 'X' as Multiplicand, 'Y' as Multiplier, and 'Z' as the output, in table II working of Modified Booth's algorithm is shown.

Figure 2: Block diagram of 128-bit complex number Booth Multiplier

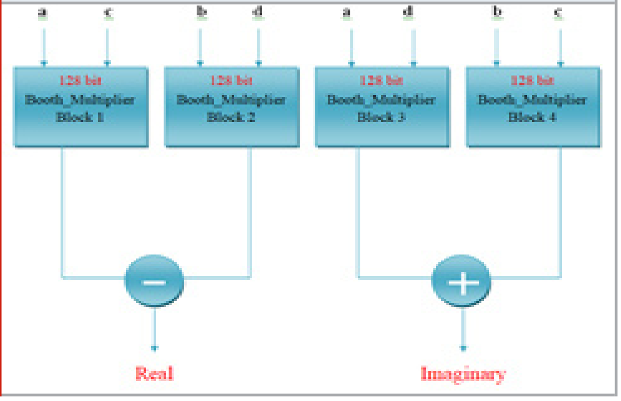
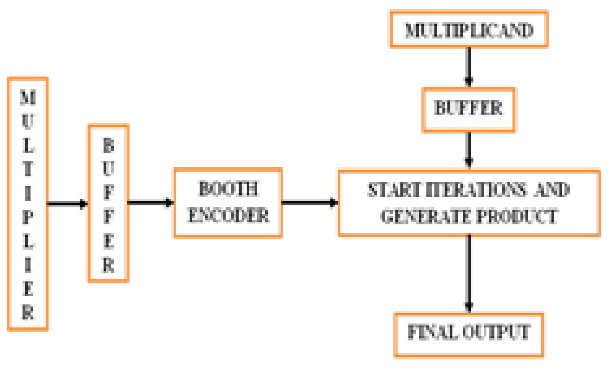

a
c
b
d
a
d
b
c
128 bit Booth_Multiplier Block 1
128 bit Booth_Multiplier Block 2
128 bit Booth_Multiplier Block 3
128 bit Booth_Multiplier Block 4
−
+
Real
Imaginary

Figure 3: Block diagram of the Modified Booth Algorithm



Functional flow of Modified Booth Algorithm: A zero is appended to the LSB of two's complement of Multiplier 'Y'. Consider Yi, Y_{i+1}, and Y_{i+2} from LSB to MSB pairwise and follow Table II. Continue the procedure until the last iteration. In figure 4, the Modified Booth multiplier of the signed 128 bit is used to build the 128-bit complex number Modified Booth multiplier.

3.3. Urdhva Complex Number Multiplier: Urdhva Tiryagbhyam ("vertically and crosswise") is the ancient Sutra of Vedic Mathematics and is the easiest method for multiplication.

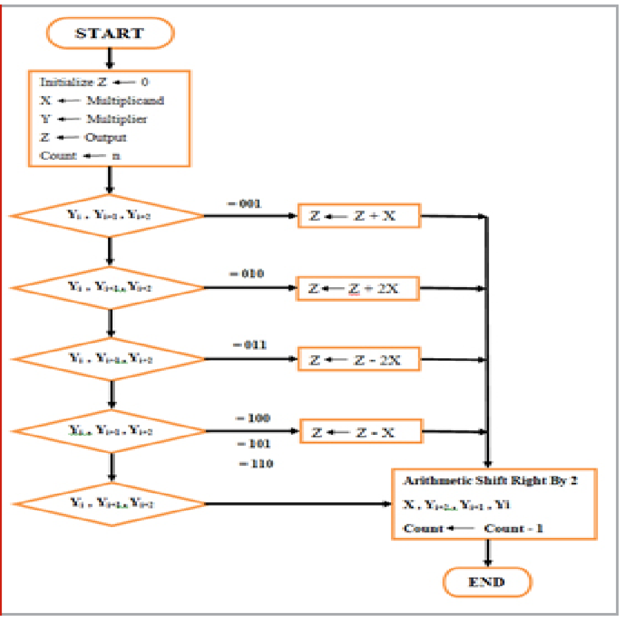


Table 2. Modified Booth Multiplier Logic

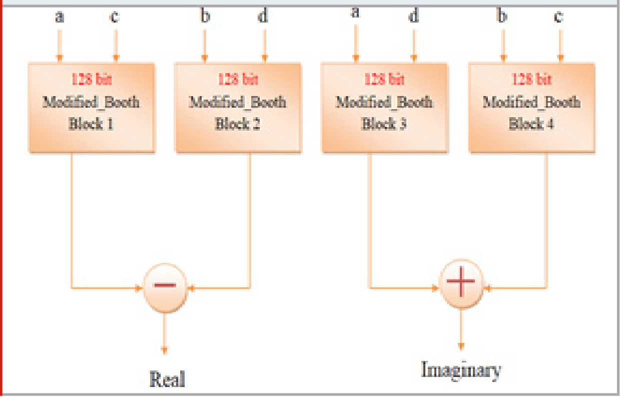
Y_{i+2}	Y_{i+1}	Y_i	Case
0	0	0	Arithmetic shift right(ASR) by 2
0	0	1	+X and Arithmetic shift right(ASR) by 2
0	1	0	Z+2X and ASR sum by 2
0	1	1	Z-2X and ASR difference by 2
1	0	0	Z-X and ASR difference by 2
1	0	1	Z-X and ASR difference by 2
1	1	0	Z-X and ASR difference by 2
1	1	1	Arithmetic shift right by 2
0	1	1	Z-2X and ASR difference by 2

The steps for the Urdhva method is shown below:

- In the above steps, initially, multiplication starts from the right.
- Multiply the extreme right column, in the obtained product LSB is written, and MSB bits are carried over for the next steps i.e. carry generated is added in the next step.
- From the left, the digits of the next column must be cross multiplied.
- For 4-digit X 4-digit, do a cross multiplication of extremes and cross multiplication of middle digits.
- Then start skipping the column of digits on the right and move leftwards till you reach the extreme column of digits on the left.
- In each step carry generated is added to the next step product.

In figure 5, a block diagram of the 128-bit complex number Urdhva multiplier is shown.

Figure 4: Block diagram of 128-bit Complex number Modified Booth Multiplier


a
c
b
d
a
d
b
c
128 bit Modified_Booth Block 1
128 bit Modified_Booth Block 2
128 bit Modified_Booth Block 3
128 bit Modified_Booth Block 4
Real
Imaginary

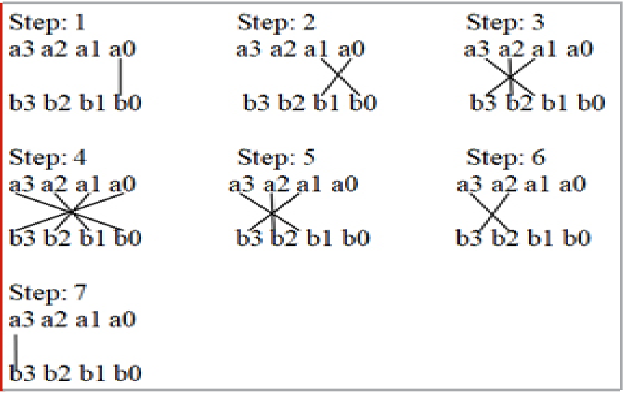
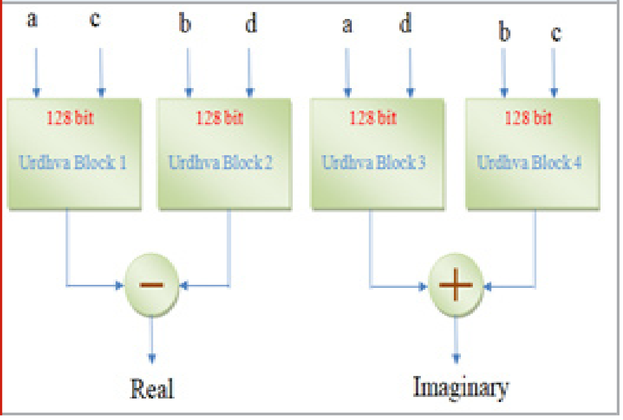


Figure 5: Block diagram of 128-bit Complex number Urdhva Multiplier



Functional flow of Nikhilam complex number multiplier

3.4 Nikhilam Complex Number Multiplier: Nikhilam is the second sutra of 16 sutras of Vedic Mathematics and is the easiest and shortcut method adopted for multiplication and division. The usage of these methods leads to the faster Multiplication of larger digit numbers. The multiplication of various digit numbers using this method includes few add, subtract, and shift operations. It can save time when multiplying the numbers that are nearer to the base of 2, 10, 100, 1000...etc.

The steps followed in Nikhilam sutra are as follows:

Step 1: Consider x1 and y1 as multiplicand and multiplier, and always x1 should be greater than y1.

Step 2: Compare the multiplicand y1 with the bases of 2 i.e. 21, 22, 23 ... so on such that y1 should be greater than powers of 2.

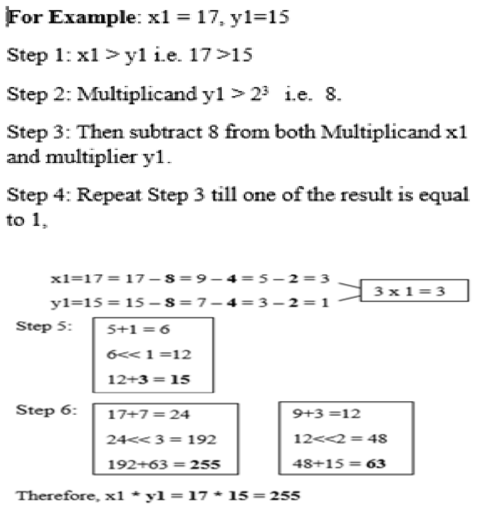
Step 3: Subtract the power of 2 from x1 and y1 and the resultants will be now x2 and y2.

Step 4: The above process continues until one of the results of x1 and y1 equals to 1.

Step 5: The last terms whose one of the result equals to 1 should be multiplied. Ex: if x3 and y3 are the results in which one of them is equal to 1 the do x3*y3.

Step 6: Then cross addition is done between one of the final results and multiplicand of the previous stage. Also, the final product is added to it. Later, the added result is left-shifted by the Nth number of base 2 i.e. (2N).

Step 7: Step 6 is repeated till the initial stage of multiplicand and multiplier but instead of final product addition, the previous stage result is added, and hence finally at the initial stage we obtain the result for multiplication of large numbers.



For Example: x1 = 17, y1=15

Step 1: x1 > y1 i.e. 17 >15

Step 2: Multiplicand y1 > 2^3 i.e. 8.

Step 3: Then subtract 8 from both Multiplicand x1 and multiplier y1.

Step 4: Repeat Step 3 till one of the result is equal to 1.

x1=17 = 17 – 8 = 9 – 4 = 5 – 2 = 3
y1=15 = 15 – 8 = 7 – 4 = 3 – 2 = 1
3 x 1 = 3

Step 5:
5+1 = 6
6<< 1 =12
12+3 = 15

Step 6:
17+7 = 24
24<< 3 = 192
192+63 = 255

9+3 =12
12<<2 = 48
48+15 = 63

Therefore, x1 * y1 = 17 * 15 = 255

Hence by following the above steps a 128-bit Nikhilam signed number multiplier is designed. A Nikhilam 128 bit signed multiplier design is used to build the block diagram of 128-bit Nikhilam complex number multiplier.

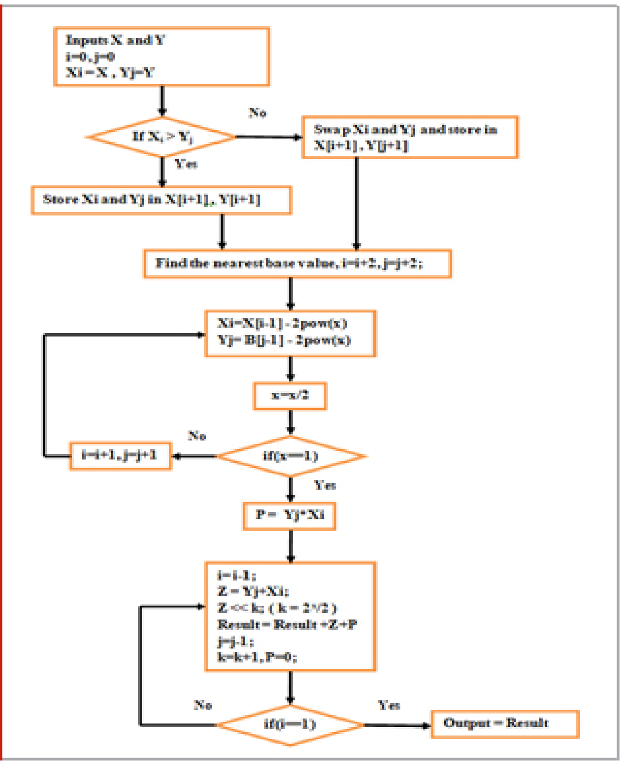
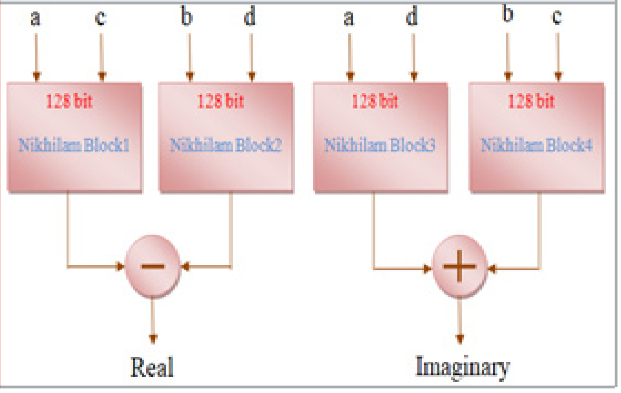


Figure 6: Block diagram of 128-bit Complex number Nikhilam Multiplier



RESULTS AND DISCUSSION

Two 128 bit complex numbers (a+ib) and (c+id) are multiplied using different methods using the formula, (a+ib) . (c+id) = (ac -db) + i (ad + bc). The complex number's real part of the product is (ac-bd) and imaginary part of the product is (ad+bc).

4.1. Simulation results of Booth Multiplier for 128 bit complex numbers.

4.2. Simulation results of Modified Booth Multiplier for 128-bit complex numbers.

4.3. Simulation results of Urdhva Multiplier for 128 bit complex numbers

Figure 7: Simulation results of 128-bit Complex Number Booth Multiplier

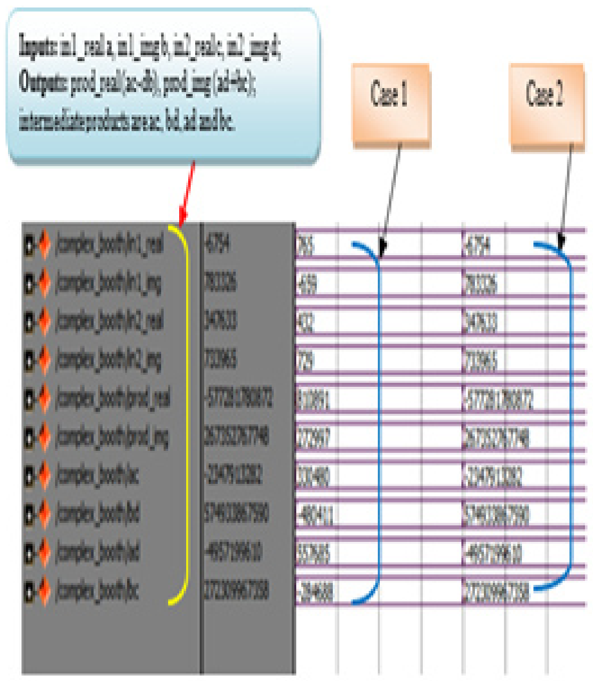


Figure 8: Simulation results of 128-bit Complex Number Modified Booth Multiplier

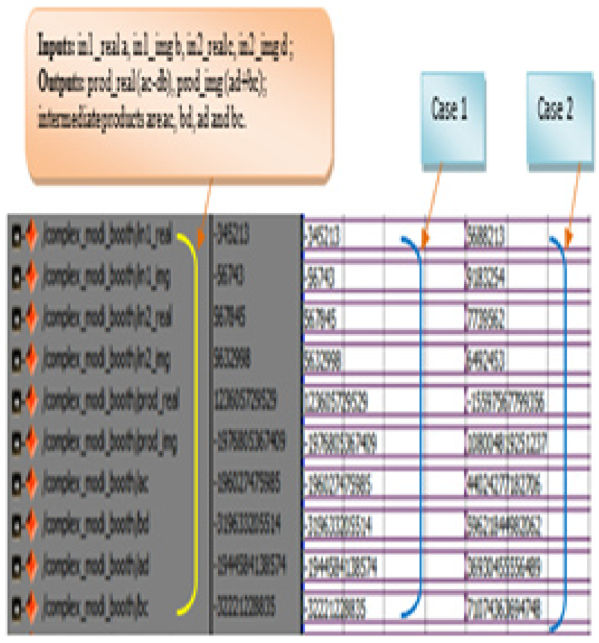
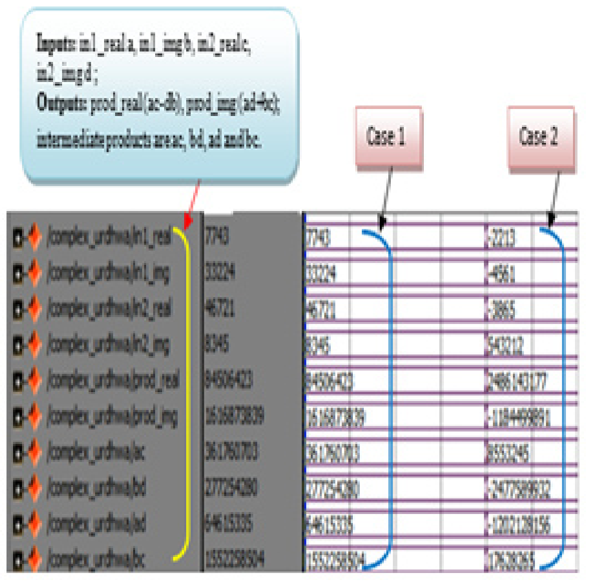

Inputs: in1_real a, in1_img b, in2_real c, in2_img d;
Outputs: prod_real(ac-db), prod_img (ad+bc);
intermediate products are ac, bd, ad and bc.
Case 1
Case 2
/complex_mod_booth/in1_real
/complex_mod_booth/in1_img
/complex_mod_booth/in2_real
/complex_mod_booth/in2_img
/complex_mod_booth/prod_real
/complex_mod_booth/prod_img
/complex_mod_booth/ac
/complex_mod_booth/bd
/complex_mod_booth/ad
/complex_mod_booth/bc

Figure 9: Simulation results of 128-bit Complex Number Urdhva Multiplier



case (1) : in1_real a = -7743, in1_img b = 33224, and in2_real c = 46721, in2_img d = 8345

prod_real = (ac-db) = {(-7743x46721) - (8345x33224)}= 84506423

prod_img = (ad+bc) = {(-7743x8345) + (33224x46721)}= 1616873839

case (2) : in1_real a = -2213, in1_img b = -4561, and in2_real c = -3865, in2_img d = 543212 .

prod_real = (ac-db) = {(-2213 x -3865) - (543212 x -4561)}= 2486143177

prod_img = (ad+bc) = {(-2213 x 543212) + (-4561 x-3865)}= -1184499891

These results are obtained using ModelSim SE 6.4 as shown in figure 9. The result obtained using manual calculation and simulation is verified.

4.4. Simulation results of Nikhilam Multiplier for 128 bit complex numbers

case (1) : in1_real a = -4533211, in1_img b = -3378652, and in2_real c = 223742, in2_img d = -2343267.

prod_real = (ac-db) = {(-4533211x223742) - (-2343267x-3378652)}= 8931353431646

prod_img = (ad+bc) = {(-4533211x-2343267) + (-3378652x223742)}= 9866577384553

case (2) : in1_real a = 5429911, in1_img b = 5471223, and in2_real c = -5643239, in2_img d = 6789453.

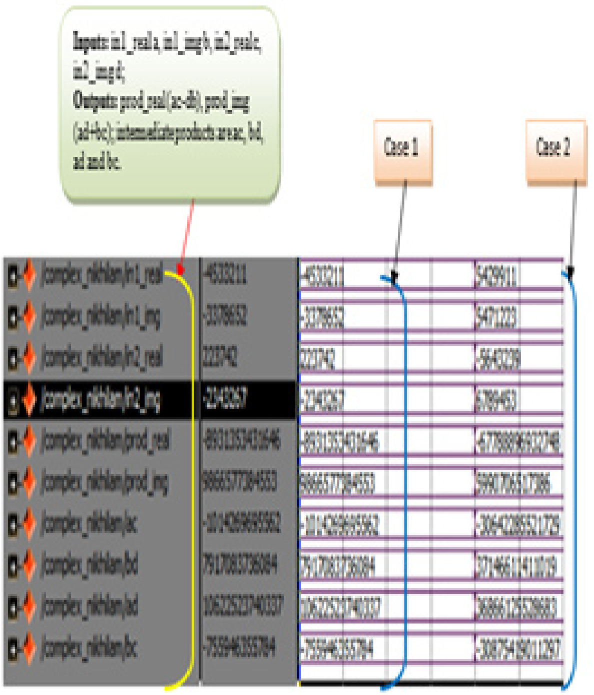
prod_real=(ac-db)={(5429911 x -5643239) - (6789453 x 5471223)}= -67788896932748

prod_img=(ad+bc)={(5429911 x 6789453) + (5471223 x -5643239)}= 5990706517386

These results are obtained using ModelSim SE 6.4 as shown in figure 10. The result obtained using manual calculation and simulation is verified.

4.5. Synthesis report of Area generated using Xilinx Vivado tool for Booth, Modified Booth, Urdhva and Nikhilam Complex Number Multipliers

Figure 10: Simulation results of 128-bit complex number Nikhilam Multiplier

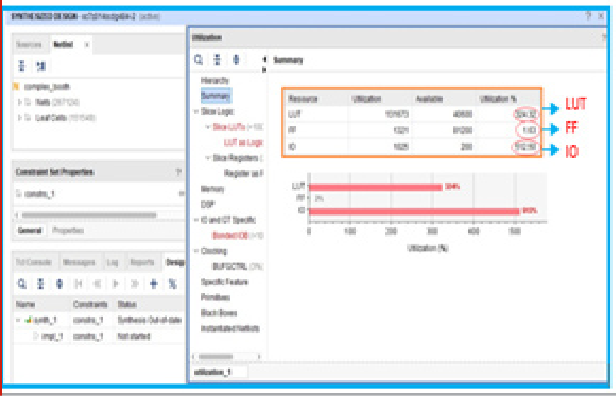

Inputs: in1_real a, in1_img b, in2_real c, in2_img d;
Outputs: prod_real (ac-db), prod_img (ad+bc); intermediate products are ac, bd, ad and bc.

Case 1

Case 2

```
/complex_nikhilam/in1_real   -4533211        -4533211        5429011
/complex_nikhilam/in1_img    -3378652        -3378652        5471223
/complex_nikhilam/in2_real   223742          223742          -5643239
/complex_nikhilam/in2_img    -2343267        -2343267        6789453
/complex_nikhilam/prod_real  -8931353431646  -8931353431646  -47788896932748
/complex_nikhilam/prod_img   9866577384553   9866577384553   5990706517386
/complex_nikhilam/ac         -1014269695562  -1014269695562  -30642085521729
/complex_nikhilam/bd         7917083736084   7917083736084   37146611411019
/complex_nikhilam/ad         10622523740337  10622523740337  36866125528683
/complex_nikhilam/bc         -7559463355784  -7559463355784  -30875489011297
```

Figure 11: Area utilization of 128-bit Complex Number Booth Multiplier



Area calculation: From the Synthesis report of area, it is observed that the percentage of area utilized by the resources such as Lookup Tables (LUTs), Flip Flops (FFs), and Input-Output (IO) pins are shown in figure 11, figure 12, figure 13 and figure 14. Lookup tables are the main building blocks of the FPGA. LUTs are a small piece of RAM loaded with data whenever the FPGA chip is powered up.

Figure 12: Area utilization of 128-bit Complex Number Modified Booth Multiplier

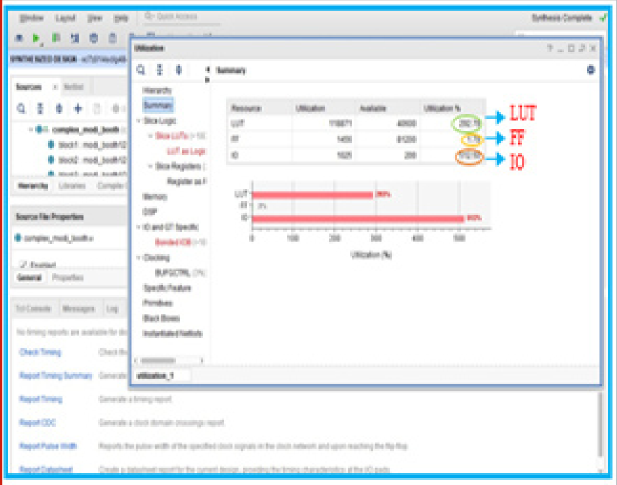


Figure 13 :Area utilization of 128-bit Complex Number Urdhva Multiplier

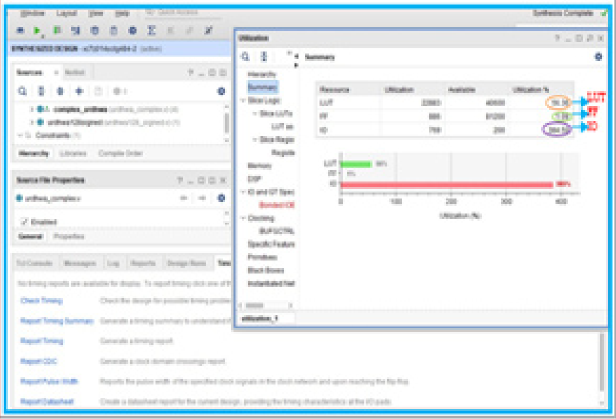
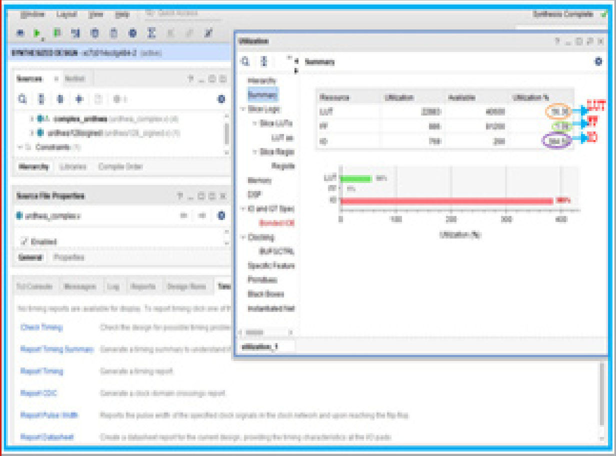


Figure 14: Area utilization of 128-bit Complex Number Nikhilam Multiplier



From the synthesis report of the area, it is observed that FFs occupy a very negligible area. The area occupied by the Input-Output Blocks (IOBs) should be ignored because they represent the pins of the FPGA and we will not use them. Usually, the area is measured in terms of percentage of occupancy and hence the percentage of area occupied by LUTs is considered. The area occupied by 128-bit Complex Number Booth, Modified Booth, Urdhva, and Nikhilam Multipliers are shown in figure 11, figure 12, figure 13 and figure 14 respectively.

Figure 15: Speed Calculation for Booth 128 bit Complex Number Multiplier

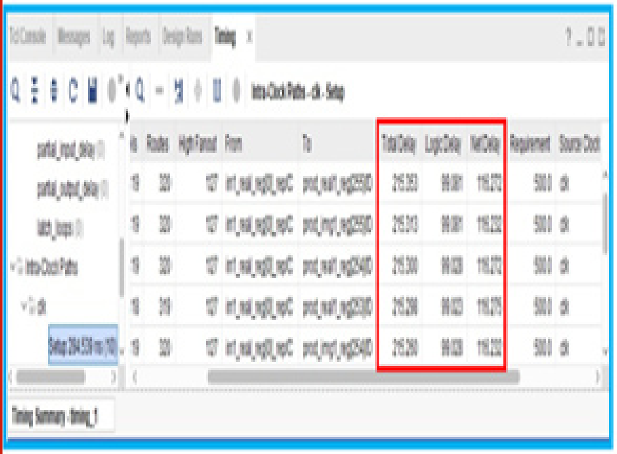
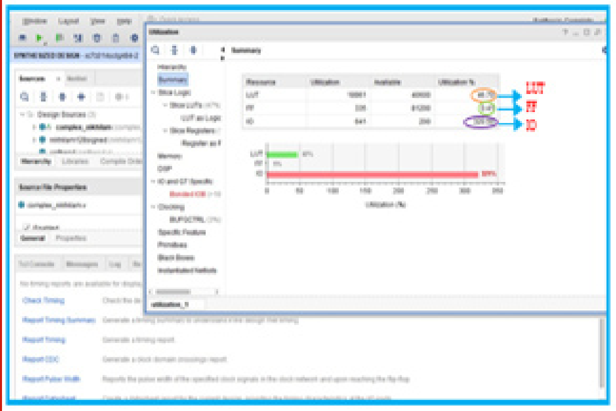


Figure 16: Speed Calculation of Modified Booth 128 bit Complex Number Multiplier



4.6. Synthesis report of Speed generated using Xilinx Vivado tool for Booth,

Modified Booth, Urdhva and Nikhilam Complex Number Multipliers

Speed Calculation: From the Synthesis report of Speed, we can observe the timing results such as Logic delay and Net delay of 128-bit complex number Booth, Modified Booth, Urdhva, and Nikhilam Multiplier. Logic delay is the measure of delay from input of logic gates to output of the logic gates; the Net delay is the measure of delay from output to input of the cell. The total delay indicates the amount of time required by the multiplier to perform the multiplication.

Total delay (ns) = Logic delay + Net delay
Speed (MHz) = 1/ Total delay (ns)

Total delay (ns) = Logic delay + Net delay = 99.081 + 116.272 = 215.353 ns

Figure 17: Speed Calculation of Urdhva 128 bit Complex Number Multiplier

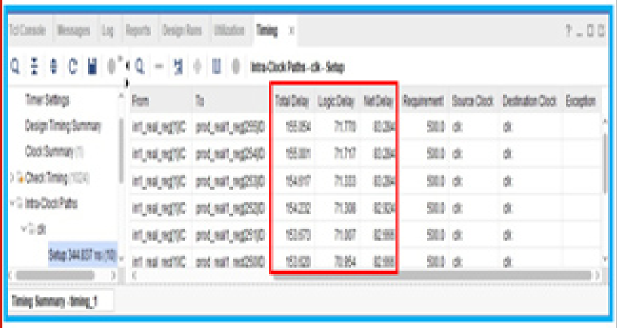
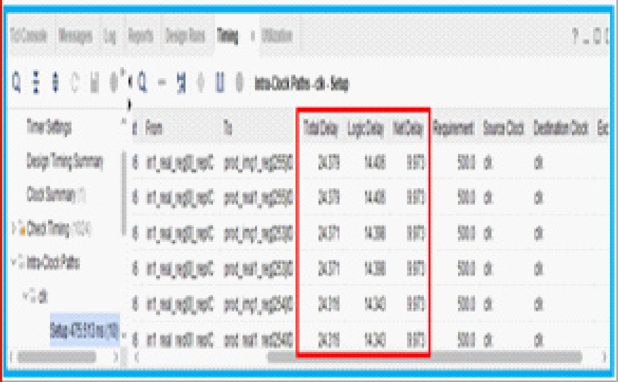


Figure 18: Speed Calculation of Nikhilam 128 bit Complex Number Multiplier



Speed (MHz) = 1/ Total delay (ns) = 1/ 215.353 (ns) = 4.643MHz

Therefore, the speed of the Booth 128 bit complex Multiplier is 4.643MHz

Total delay (ns) = Logic delay + Net delay = 71.770 + 83.284 = 155.054 ns

Speed (MHz) = 1/ Total delay (ns) = 1/ 155.054 (ns) = 6.449MHz

Therefore, the speed of the Modified Booth 128-bit complex Multiplier is 6.449MHz

Total delay (ns) = Logic delay + Net delay = 14.406 + 9.973 = 24.379 ns

Speed (MHz) = 1/ Total delay (ns) = 1/ 24.379 (ns) = 41.018MHz

Therefore, the speed of the Urdhva 128 bit complex Multiplier is 41.018MHz

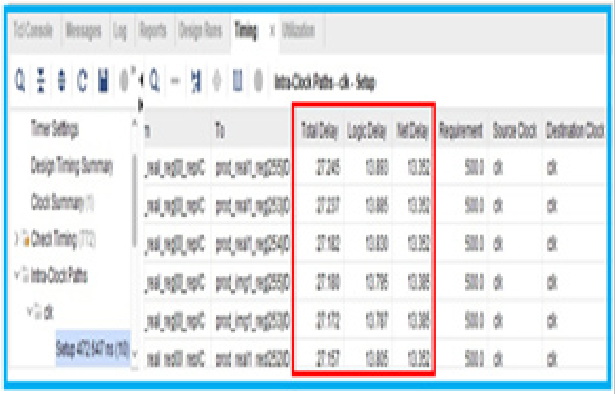
Total delay (ns) = Logic delay + Net delay = 13.893 + 13.352= 27.245 ns

Speed (MHz) = 1/ Total delay (ns) = 1/ 27.245 (ns) = 36.703MHz

Therefore, the speed of the Nikhilam 128 bit complex Multiplier is 36.703MHz

4.7. Synthesis report of Power generated using Xilinx Vivado tool for Booth, Modified Booth, Urdhva and Nikhilam Complex Number Multipliers

Figure 19: Power calculation for Booth 128 bit Complex Number Multiplier



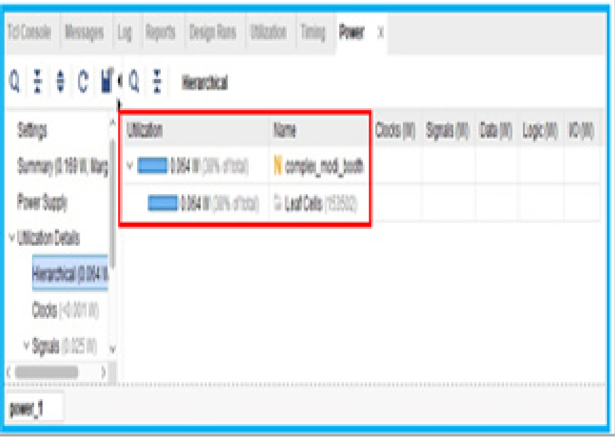
Tcl Console | Messages | Log | Reports | Design Runs | Timing | Utilization

Intra-Clock Paths - clk - Setup

Timer Settings
Design Timing Summary
Clock Summary (1)
Check Timing (772)
Intra-Clock Paths
clk
Setup 472.547 ns (10)

From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
_real_reg[0]_rep/C	prod_real1_reg[255]/D	27.245	13.893	13.352	500.0	clk	clk
_real_reg[0]_rep/C	prod_real1_reg[253]/D	27.237	13.885	13.352	500.0	clk	clk
_real_reg[0]_rep/C	prod_real1_reg[254]/D	27.182	13.830	13.352	500.0	clk	clk
_real_reg[0]_rep/C	prod_imag1_reg[255]/D	27.180	13.795	13.385	500.0	clk	clk
_real_reg[0]_rep/C	prod_imag1_reg[253]/D	27.172	13.787	13.385	500.0	clk	clk
_real_reg[0]_rep/C	prod_real1_reg[252]/D	27.157	13.805	13.352	500.0	clk	clk

Figure 21: Power calculation of Urdhva 128 bit Complex Number Multiplier



Tcl Console | Messages | Log | Reports | Design Runs | Utilization | Timing | Power

Hierarchical

Settings
Summary (0.169 W, Margin
Power Supply
Utilization Details
Hierarchical (0.064 W)
Clocks (<0.001 W)
Signals (0.025 W)

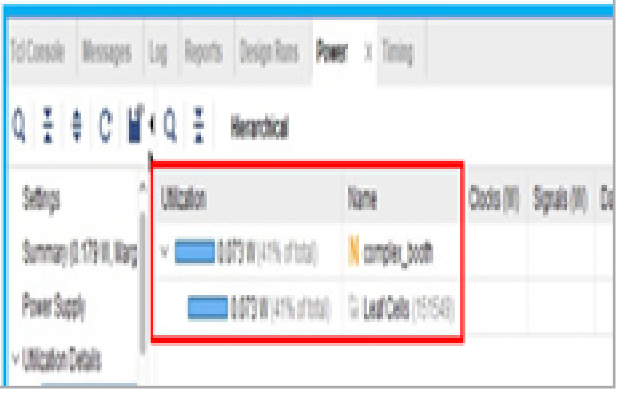
Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Logic (W)	I/O (W)
0.064 W (38% of total)	complex_mod_booth					
0.064 W (38% of total)	Leaf Cells (15350)					

power_1

Table 3. Area, Speed and Power Comparison of 128-bit Complex Number Multiplier

Sl. No.	Methods implemented	Area	Speed	Power
1.	Booth	324.32%	4.643 MHz	0.073W
2.	Modified Booth	292.79%	6.449 MHz	0.064W
3.	**Urdhva**	**56.36%**	**41.018 MHz**	**0.049W**
4.	**Nikhilam**	**46.70%**	**36.703 MHz**	**0.064W**

Figure 20: Power Calculation of Modified Booth 128 bit Complex Number Multiplier



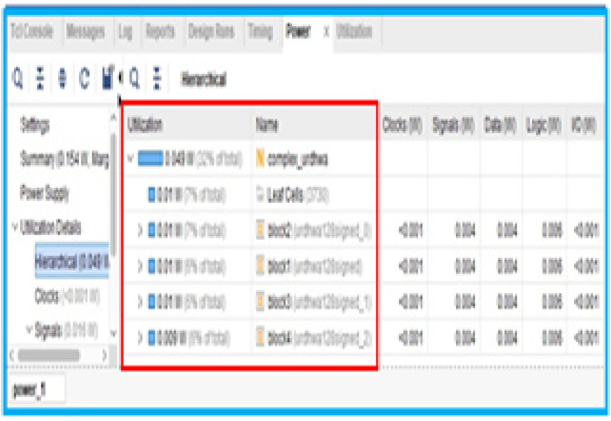
Tcl Console | Messages | Log | Reports | Design Runs | Power | Timing

Hierarchical

Settings
Summary (0.179 W, Margin
Power Supply
Utilization Details

Utilization	Name	Clocks (W)	Signals (W)	D
0.073 W (41% of total)	complex_booth			
0.073 W (41% of total)	Leaf Cells (15154)			

Figure 22: Power calculation of Nikhilam 128 bit Complex Number Multiplier



Tcl Console | Messages | Log | Reports | Design Runs | Timing | Power | Utilization

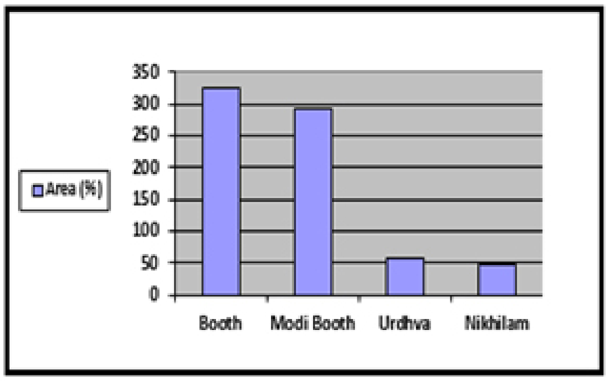
Hierarchical

Settings
Summary (0.154 W, Margin
Power Supply
Utilization Details
Hierarchical (0.049 W)
Clocks (<0.001 W)
Signals (0.016 W)

Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Logic (W)	I/O (W)
0.049 W (32% of total)	complex_urdhva					
0.011 W (7% of total)	Leaf Cells (373)					
0.011 W (7% of total)	block2 (urdhva128signed_0)	<0.001	0.004	0.004	0.006	<0.001
0.010 W (6% of total)	block1 (urdhva128signed)	<0.001	0.004	0.004	0.006	<0.001
0.010 W (6% of total)	block3 (urdhva128signed_1)	<0.001	0.004	0.004	0.006	<0.001
0.009 W (6% of total)	block4 (urdhva128signed_2)	<0.001	0.004	0.004	0.006	<0.001

power_1

Figure 23: Area Utilization comparisons of the complex number multipliers



From Table III, it is observed that the area utilization by 128-bit Nikhilam complex number multiplier is 46.70% as compared to the complex number Urdhva multiplier. Thus, 128-bit complex number Nikhilam multiplier is recommended to achieve a reduced chip area for a co-processor design. The power consumed by the 128-bit Nikhilam complex number multiplier is more than the Urdhva multiplier. The speed of computation of the Urdhva multiplier is better than the Nikhilam multiplier. It is observed that there is a tradeoff between area utilization and power consumption of Urdhva and Nikhilam complex number multiplier. If we switch from Nikhilam complex number multiplier to the Urdhva multiplier than the area requirement is more and the percentage change in area is by 20.68. Similarly, if we switch from Nikhilam to Urdhva multiplier than the power requirement is less and the percentage change of power is 30.61.

Figure 24: Speed comparisons of the complex number multipliers

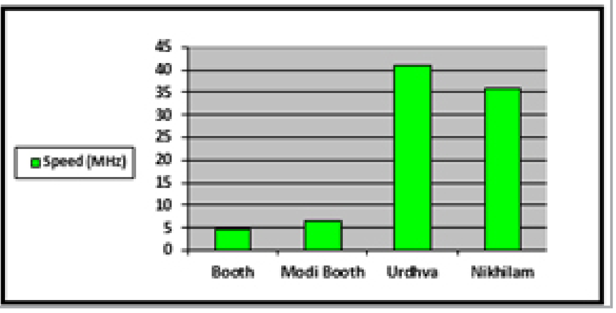
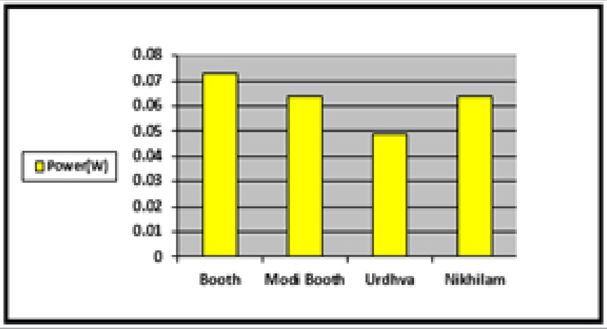


Figure 25: Power comparisons of the complex number multipliers



The area utilization, speed, and power comparison of Booth, Modified Booth, Urdhva, and Nikhilam for 128-bit complex number multipliers are shown in figure 23, figure 24, and figure 25.

CONCLUSION

In this paper, Complex number multipliers for 128 bits are designed using various methods such as the Booth algorithm, Modified Booth algorithm, Urdhva Sutra, and Nikhilam Sutra. The various architectures of 128-bit complex numbers designed are simulated using the ModelSim SE simulation tool and synthesized using the Xilinx Vivado tool. From the Synthesis reports area, speed and power results are obtained. The areas of all the 128-bit complex number multipliers are compared based on the number of LUTs occupied and since the area occupied by FFs is negligible.

From the area, speed, and power comparison of Booth, Modified Booth, Urdhva, and Nikhilam 128 bit complex number multipliers, it is observed that the area occupied by 128-bit Nikhilam complex number multiplier is least i.e. 46.70%. The power consumption of 128 bit Urdhva complex number multiplier is least compared to other 128-bit complex number multipliers. Thus, there is a tradeoff between area and power, and hence wherever area is preferred, 128-bit Nikhilam complex number multiplier can be used and where the power is preferred, 128 bit Urdhva complex number multiplier can be used. These designed 128-bit complex number multiplier architectures can be used in highly efficient co-processor design to reduce chip area.

REFERENCES

A. P. Pascual, J. Valls, and M.M. Peird (1999), Efficient Complex-Number multipliers mapped On FPGA, 6th IEEE International Conference on Electronics, Circuits and Systems, ISBN: 0-7803-5682-9.

Ankush Nikam, Swati Salunke, Sweta Bhurse(2015), Design and Implementation of 32bit Complex Multiplier using Vedic Algorithm,International Journal of Engineering Research & Technology, Vol. 4 Issue 03,Center for VLSI and Nanotechnology Visvesvaraya National Institute of Technology Nagpur, Maharashtra, India.

Huddar S.R., Rupanagudi S.R., Janardhan V., Mohan S., Sandya S. (2013) Area and Speed Efficient Arithmetic Logic Unit Design Using Ancient Vedic Mathematics on FPGA". In: Unnikrishnan S., Surve S., Bhoir D. (2013) Advances in Computing, Communication, and Control. Communications in Computer and Information Science, Springer, Berlin, Heidelberg, Vol 361.

K.Deergha Rao, Ch. Gangadhar, Praveen K Korrai(2016), FPGA Implementation of Complex Multiplier Using Minimum Delay Vedic Real Multiplier Architecture, International Conference on Electrical, Computer and Electronics Engineering. Indian Institute of Technology (Banaras Hindu University) Varanasi, India. ISBN:978-1-5090-5384-1.

M Gudhimetla, C M Ananda, (2017) Comparison of Different Types of Multipliers for Speed, Area, And Power, International Journal of Industrial Electronics and Electrical Engineering, 5(12): 83-87.

Manjunath, Venama, Harikiran, Kopparapu, Manikanta, Sivanantham S, Sivasankaran K(2015), Design and Implementation of 16x16 Modified Booth Multiplier, International Conference on Green Engineering and Technologies, 978-1-4673-9781-0, VIT University, Vellore, India.

Nikhil R. Mistri, Prof S. B. Somani, Prof Dr. V. V. Shete (2016), Design and Comparison of Multiplier using Vedic Mathematics, International Conference on Inventive Computation Technologies, ISBN:978-1-5090-1285-5, Pune, India.

Pavan Kumar U.C.S, Saiprasad Goud A, A.Radhika(2013), FPGA Implementation of high speed 8-bit Vedic multiplier using barrel shifter, International Conference on Energy Efficient Technologies for Sustainability, ISBN:978-1-4673-6150-7.

Prof S. B. Somani, Nikhil R. Mistri (2016), Study of Vedic Multiplier Algorithms using Nikhilam Method", International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering Vol. 5, Issue

Razaidi Hussin1, Ali Yeon Md. Shakaff2, Norina Idris1, Zaliman Sauli1, Rizalafande Che Ismail1, and Afzan Kamarudin1(2008), An Efficient Modified Booth Multiplier Architectur, International Conference on Electronic Design, Penang, Malaysia. ISBN :978-1-4244-2315-6/08.

S. R. Rupanagudi et al. (2014), Design of a low power Digital Down Converter for 802.16m - 4G WiMAX on FPGA, International Conference on Advances in Computing, Communications and Informatics, New Delhi, Pages 2303-2308.

S. Rao Rupanagudi et al. (2019), A Further Optimized Mix Column Architecture Design for the Advanced Encryption Standard, International Conference on Knowledge and Smart Technology, Phuket, Thailand, Pages 181-185.

Sai Venkatramana Prasada G S, G Seshikala, Niranjana Sampathila(2018), Performance Analysis of 64x64 bit Multiplier Designed Using Urdhva Tiryakbyham and Nikhilam Navatashcaramam Dashatah Sutras, IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics, Manipal Institute of Technology Manipal Academy of Higher Education Karnataka, Manipal, India.

Savita Patil, D.V.Manjunatha, Divya Kiran (2014), Design of Speed and Power Efficient Multipliers Using Vedic Mathematics with VLSI Implementation, International Conference on Advances in Electronics Computers and Communications, ISBN:978-1-4799-5496-4.

Soniya, S Kumar(2013), A Review of Different Type of Multipliers and Multiplier-Accumulator Unit, International Journal of Emerging Trends & Technology in Computer Science, 2(4) : 364 -368.

Subramani, Prabu, Ganesh Babu Rajendran, Jewel Sengupta, Rocío Pérez de Prado, and Parameshachari Bidare Divakarachari. "A Block Bi-Diagonalization-Based Pre-Coding for Indoor Multiple-Input-Multiple-Output-Visible Light Communication System." Energies 13, no. 13 (2020): 3466.