ISSN 2347 - 3983
Volume 7, No. 11 November 2019
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter437112019.pdf
https://doi.org/10.30534/ijeter/2019/437112019

Double Precession Floating Point Multiplier using Schonhage – Strassen Algorithm used for FPGA Accelerator

B. Srikanth[1], M. Siva Kumar[2], J.V.R. Ravindra[3], K. Hari Kishore[4]
[1]Research Scholar, K L University, Guntur, Andhra Pradesh, India, srikanth.vlsi.2011@gmail.com
[2]Associate Professor, K L University, Guntur, Andhra Pradesh, India, siva4580@kluniversity.in
[3]Professor, Vardhaman College of Engineering, Hyderabad, Telangana, India, jvrravindra@gmail.com
[4]Professor, K L University, Guntur, Andhra Pradesh, India, kakarla.harikishore@kluniversity.in

ABSTRACT

The Double Precession Floating Point (DPFP) multiplication algorithm generally use in several applications like arithmetic logic unit, scientific calculators, signal processing and so on. The major limitation of the DPFP arithmetic operation is difficult to calculate 53 x 53 mantissa multiplication, which requires more area. In this research work, a High Speed Schonhage – Strassen Algorithm used for FPGA accelerator (HS-SSA-FPGA) was implemented to provide a low amount of multiplication hardware compared to the conventional methods. The main advantage of the Schonhage Strassen multiplication is that, the multiplication of integer values greater than 5 digits ranging from 2215 to 2217 bit values proves to be efficient. This experimental research work describes SSA architecture and implementation of the FPGA accelerator to get a better output in terms of area and system speed. The FPGA accelerator Control Unit (CU) maximized the Processing Elements (PEs) based on external task request generated from Control Processing Unit (CPU). The SSA-FPGA method was implemented in Xilinx based on Virtex-5 xc5vlx20T by using Verilog HDL Code. The experimental result showed that the proposed method improved the performance of the system in FPGA accelerator compared to the existing methods: Karatsuba and Vedic multiplier. The HS-SSA-FPGA works at 230.14 MHz. The HS-SSA-FPGA multiplication speed is increased by 49 percentages compared with Karatsuba based floating point multiplication..

Key words : Control processing unit, Control unit, Double precision floating point, Processing element, and Schonhage Strassen algorithm.

1. INTRODUCTION

Digital arithmetic operations are most important in the design of digital processors and application specification systems. An arithmetic circuit forms a significant class of circuits in the digital systems [1]. The real numbers denoted in binary format known as Floating Point (FP). Based on IEEE-754 standard, FP formats categorized into binary and decimal interchange formats. The FP multipliers are most important in DSP applications [2]. The FP arithmetic mostly used in several areas, signal processing and scientific computation. A multiplication operation is the second fundamental operation of the arithmetic unit. However, most of these advanced applications require low latency, high-frequency operations with low area [3], [4], [5]. The FP arithmetic depends on FPGA implementation that defines different arithmetic operations such as addition, subtraction, multiplication, and division. The FP-FPGA arithmetic unit is used for single, double, quad precision. These precisions identify various bits of operation [6].

The FP- Multiply-Add Fused (MAF) based FPGA suitable for low-precision formats based on combining the addition, subtraction and multiplication operations are needed in the Mantissa-Data path processing into a single operation. This method requires one or more arithmetic operations, which occupy more area [7]. The DPFP using the Karatsuba algorithm requires more area and consume less power [9]. The FPGA devices have the benefit of reconfigurability compared to the ASIC platform. In the present days, new FPGA devices such as Virtex-4, Virtex-5, Virtex- 6 can provide a huge amount of high-speed logic resources and Intellectual Property (IP) core [10]. In this paper, the SSA-FPGA method introduced to overcome the above-mentioned problems. This method does not need several CPU cores. The PE requires the addition and multiplier blocks that are arranged like pipeline architecture. The DPFP multiplier by employing SSA used in PE module for efficient power and area. The SSA-FPGA method implemented in Xilinx Virtex-5 FPGA by using Verilog Code.

2. RELATED WORK

Researchers have suggested several methods on the DPFP. In this section, a brief evaluation of some significant contribution in this field presented below.

Srinivas et al, [11] proposed design and performance analysis of DPFP multiplier by employing Urdhva Tiruyagbhyam Sutra based on FPGA Virtex-5 platform. The performance of the Vedic Sutra multiplier is excellent with the proficient utilization of resources. This multiplier design was completely compatible with the standards of IEEE. In this paper, the Vedic multiplier was compared with karatsuba multiplier. The multiplication was difficult when the system became complex which was the major limitation of the proposed method.

Jovanovic, and V. Milutinović [12] presented the FPGA accelerator used for DPFP matrix multiplication. This architecture reduced the resource utilization and increased the clock frequency. In this research work, the architecture has a symmetric communication pattern, it was well suited for the Full Duplex (FD) communication link. The matrix multiplication was much more difficult to implement.

Jaiswal et al. [13] proposed VLSI implementation of double precision floating-point multiplier using karatsuba technique. In this paper, the multiplication process done by using Karatsuba and designed in FPGA platform. The karatsuba algorithm reduced the cost of the multiplier blocks, which requires extra subtractors and adders. This method requires a number of adder and subtractors that increased the slice count.

Ramesh et al. [14] proposed a high- speed IEEE 754 DPFP multiplier employing tiling method that was implemented in the FPGA Xilinx tool by using Verilog code. This design achieved 436.815 flip-flops with low latency of seven clock cycles, it was 97% faster compared to Xilinx floating point multiplier core. But the drawback of the proposed method required more area.

The SSA algorithm based DPFP used for FPGA accelerator to overcome the above-mentioned problem. This algorithm implemented to improve the FPGA performance parameters such as LUT, slice, flip-flop, and frequency.

3. SSA-FPGA METHODOLOGY

Now a days, there are many algorithms available on DPFP to minimize the multiplications in which SSA is one of the most suitable and popular algorithms. This SSA based DPFP multiplier consume low area compared to the conventional normal multiplication method. The SSA-FPGA method implemented using the Xilinx ISE synthesis tool, a simulation tool, and Xilinx Virtex-5 xc5vlx20T FPGA platform. The Figure.1 shows the block diagram of the FPGA accelerator with CPU. The FPGA accelerator consists of CU and PE. The CU of the FPGA accelerator searches the arithmetic operation in the CPU.

3.1 Control Unit Process

The CU of FPGA accelerator block performs following below operations. The CU can search for arithmetic operation in the CPU. The CU Sends the request data to CPU to perform an arithmetic operation in the PEs. Once the acknowledgment receives from the CPU, the CU diverts all arithmetic operations from the CPU to PEs in the FPGA accelerator. The CU sends the PEs arithmetic outputs to CPU immediately after arithmetic operations are calculated. Increment the PE in FPGA accelerator based on external other task generated from the CPU. The limit of the PEs is based on the input-output block of targeted FPGA devices.

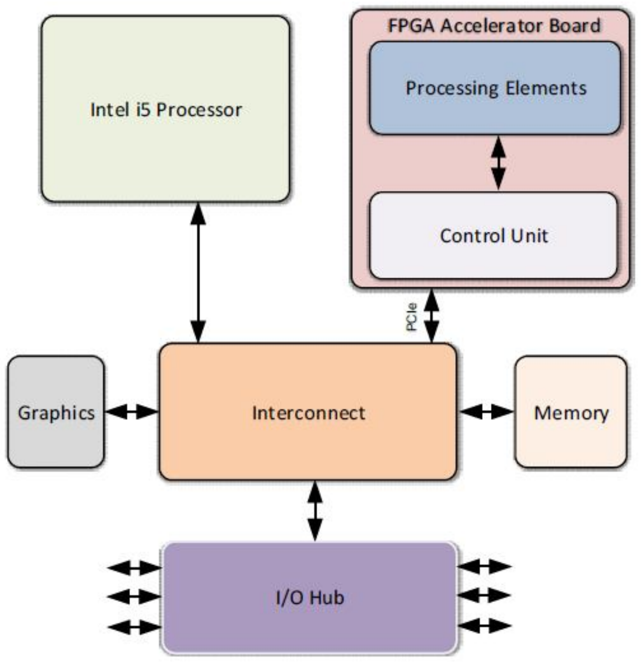


Figure 1: Block diagram of FPGA accelerator communication with CPU

3.2 Processing Element Process

The PEs of FPGA accelerator performs following below operations. The PEs receives the request from CU to perform an arithmetic operation such as addition and multiplication. Once CU receives an acknowledgment (ready) from the PE, and then the CU sends an entire arithmetic operation to PEs. Each and every PE has addition and multiplier blocks ordered as pipelined to decrease the clock latency. Multiplier block is implanted by employing SSA floating point multiplication technique. The PEs are generated in a generic manner based on request raised by the CU to perform the arithmetic operation. The main challenge in DPFP operations in all processors is hardware implementation. The FP multiplication is much difficult to develop between the FP arithmetic operations. In this experimental research paper, mainly focus on multiplications due to the multiplication operation consumes more area and power compared to other arithmetic operations.

3.3. Schonhage – Strassen Multiplication Algorithm

The Strassen algorithm is one of the most suitable options for huge integer multiplication. General multiplication schemes such as Karatsuba, classical school book multiplication are not much suitable for large number multiplication. So, an efficient NTT multiplication process based Stassen approach choose from large operand sizes, which achieves low area, power and increase the system speed. The procedure of Strassen Algorithm includes five steps: the first step conventional theorem consists of two vectors such as X and Y. In the second step, FFT used for every vector. In the third step, multiply the resulting vectors element by element. In the fourth step, take the inverse FFT of the product and integrate the vector, and finally got the multiplication result. The SSA is Number Theoretic Transform (NTT) based huge integer multiplication algorithm with a runtime of *O(NlogNloglogN)*. For an N - digit number, NTT is calculated by employing the $R_N = Z / (2^N+1) Z$, here N represents a power of 2. The SSA described as follows:

Algorithm.1. Schonhage Strassen Algorithm

```
Input: Multiplicand operands X and Y, base R=2k
Output: Product Z
1. Calculate the NTT of the digit (respect to the base) of
   X and Y
2. Multiply the NTT outputs, element by element:
3. Set Z[i] NTT(A)[i]*NTT(B)[i]
4. Calculate Inverse NTT of the Z
5. Set Z` = INTT (Z)
6. Accumulate the carries
7. if Z[i] ≥R then
8. Set Z [ i+1 ] = Z[i+1]+ [ Z[i] / R ]
9. Set Z[i] = Z(mod R)
10. return Z
```

In the SSA, for the NTT computation, sample the numbers X and Y that fits into N - digit with a sampling size. The selected *q* is a prime number with a Primitive Root (PR) *u*, for example $u^q = 1(mod\ q)$. Then, NTT forms of the number represents as,

$$X_k = \sum_{k=0}^{N-1} u^k x_k \tag{1}$$

$$Y_k = \sum_{k=0}^{N-1} u^k y_k \tag{2}$$

After that, the elements are multiplied to form $c_k : c_k = X_k.Y_k(mod\ q)$. By using inverse NTT to compute following below equations (3)

$$Z_k = \sum_{k=0}^{N-1} u^{-k} z_k \tag{3}$$

As the final step, accumulate the carry additions finalize the evaluation of Z. The fast NTT and inverse NTT calculation method used in SSA multiplication. In this work, the significant general method is used for calculating the Fast Fourier Transform (FFT). The SSA computes the FFT of a sequence A:

$$A_k = \sum_{j=0}^{N-1} a_j e^{-i2\Pi k \frac{j}{N}}$$

By turning the length N transform evaluation into two N/2 size Fourier Transform (FT) computations as follows the equation(4),

$$A_k = \sum_{\substack{j=0 \\ m\ even}}^{N/2-1} a_{2m} e^{-i2\Pi k \frac{m}{N/2}} + e^{-2\Pi ik / N/2} \sum_{\substack{m=0 \\ m\ odd}}^{N/2-1} a_{2m} + 1e^{-i2\Pi k\, m / N/2} \tag{4}$$

From the equation (4), change $e^{-2\Pi ik / N}$ with power of *u* and perform the divisions into 2-halves odd and even indices, recursively. In this research work, the Strassen multiplication evaluated in *O(NlogNloglogN)* time with the help of the FT technique.

3.3.1 Number Theoretic Transform

The NTT is the type of the FFT that performs with integer number in the domain modulo. The significant form of NTT is the fermat transform modulo 2^n+1. So, this NTT takes place in the field of integers modulo (2^n+1), it is commonly employed in the SSA approach. The NTT function presented in equation (5) as follows,

$$C_j = \sum_{i=0}^{N-1} C_i g^{ij} (\mathrm{mod}\, q) \tag{5}$$

In this equation (5), the array of N transformed into the array Cj, it consists of N numbers, Where both i and j range from 0 to N-1. The array of Cj can be taken as input while the array Cj considered as throughput. Moreover, g is the Nth root of unity mod q, wherever q is a number of the form 2^n+1. The number of the output array Cj is in the form mod q.

3.3.2 Root of Unity

The nth root of unity is the number that raised to the nth power as result is 1. The Primitive Root of Unity is introduced, nth root of unity C is primitive if $C^n=1$ and all powers < n do not result in 1. An nthroot of unity modulo *q* is a number of *q*, which raised to the nth power (mod *q*) has 1 as output. The nth root of unity S is primitive if $C^n = 1(mod\ q)$ and power value is smaller compared to *n* doesn't yield *1(mod q)*.

3.3.3 Twiddle-Factor and Inverse NTT process

The twiddle factor is the nth primitive root of unity. For any value of N, the twiddle factor is N/2. In this paper, the SSA multiplication method uses product of X and Y (mod 2n+1), when input number X and Y both are ≤ 2n+1. The multiplication of two numbers with a maximum of *n* bits results in a number of 2^n bits at maximum, so the modules are different. Hence, applying the acyclic conventional in this SSA multiplication process. It splits into N/2 equal parts and each part consists of an equal number of bits. Then the first half elements with 'content bits' and last half with zeros called as 'zero padding'. In the SSA algorithm, an inverse NTT evaluation used for input sample *a* and *b*, *C = ab(mod q)* by using inverse NTT equation (6),

$$C_n = \sum_{n=0}^{N-1} C_n w^{-n} \tag{6}$$

In the final step, this algorithm accumulates the carry addition to finalize the computation of the C.In this paper an efficient SSA implemented by utilizing fast NTT and inverse NTT computation method. In section 4, described about the results and discussion of SSA-FPGA method.

4. RESULT AND DISCUSSION

In this work, the DPFP using SSA approach is implemented in the Xilinx ISE 14.5 tool based on the Virtex5 xcvlx20T by using Verilog Code. The Strassen multiplication employed to design the FPGA accelerator to improve the FPGA performances such as slice, LUT, flip-flop, frequency. In this work, both the exiting and SSA-FPGA methods implemented by using Verilog code and the outputs tabulated in Table 1.

The Table.1 shows the comparison of the parameters like LUTs, slices, flip-flop and operating frequency of existing methods such as Karatsuba and Vedic multiplier and SSA-FPGA method in the Virtex-5 xc5vlx20T. Two 53-bits given to the input of the SSA circuit, which produced the 105-bits output. The number of LUT, flip-flop and slice reduced in the SSA-FPGA method compared to the existing method, which is shown in the Table.1. Due to the reduction of those FPGA parameters, the area optimized in the SSA-FPGA method. The operating frequency also computed in the Virtex-5 xc5vlx20T. Figure.2 shows the FPGA performance for existing and SSA-FPGA method. These results took for Virtex-5 from the Xilinx tool. From the Figure.2, it is clear that all the FPGA performance improved in the SSA-FPGA method compared to the existing method. Figure.2 shows the comparison of the operating frequency for the existing and proposed method. Table.2 results taken from cadence RTL compiler tool. The comparison graph of area and power for various floating point multiplication techniques are shown in Figure.3.

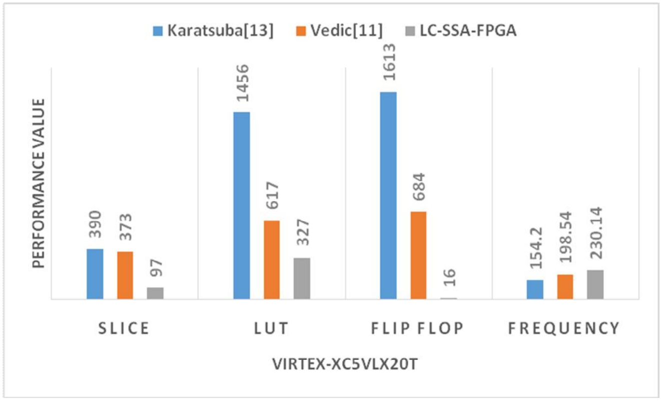


Figure 2: Comparison graph of FPGA performance for existing and SSA-FPGA method.

Table 1: Logic utilization comparison of existing and SSA-FPGA methods

FPGA Target	Methodology	Slice	LUT	Flip Flop (FF)	Frequency (MHz)
Virtex5 xc5vlx20T	Karatsuba [13]	390/7200	1456/28800	1613/28800	154.2
	Vedic [11]	373/7200	617/28800	684/28800	198.54
	SSA-FPGA	97/7200	327/28800	16/28800	230.14

Table 2: Comparison of Area and Power Analysis for various floating point multiplication techniques

Instance	Cells	Leakage Power (nW)	Dynamic Power (nW)	Total Power (nW)
Normal Floating Point Multiplication	11125	515.57	6079.66	6595.23
Karatsuba Floating Point Multiplication	9144	412.88	1362.81	1775.7
Schonhage – Strassen Floating Point Multiplication	5314	153.349	6592.238	6745.588

Table 3: Experimental results of FPGA performance of SSA-FPGA Accelerator

Circuit	FPGA Device	Slices / Slice registers	LUT	FF/ FF-LUT Pairs	Frequency (MHz)
SSA-FPGA Accelerator	Virtex4- xc4vfx12	645/5472	819/10944	760/10944	58.65
	Virtex5-xc5vlx20t	1014/12480	9754/12480	463/10305	15.33
	Virtex6–xc6vcx75t	884/93120	2299/46560	579/2604	10.835
	Virtex7–xc7vx330t	884/408000	2299/204000	579/2604	12.262

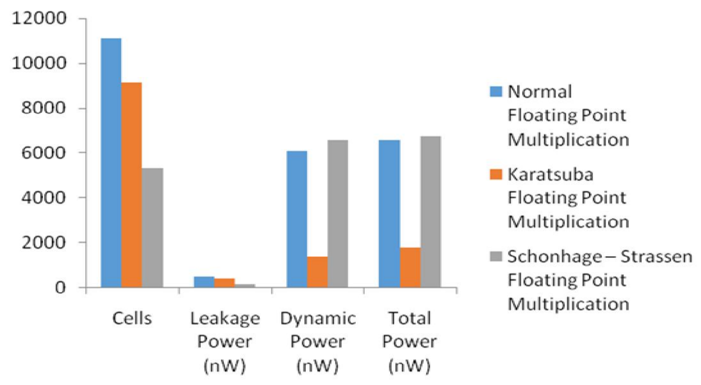


Figure 3: Comparison graph of Area and Power for various floating point multiplication techniques

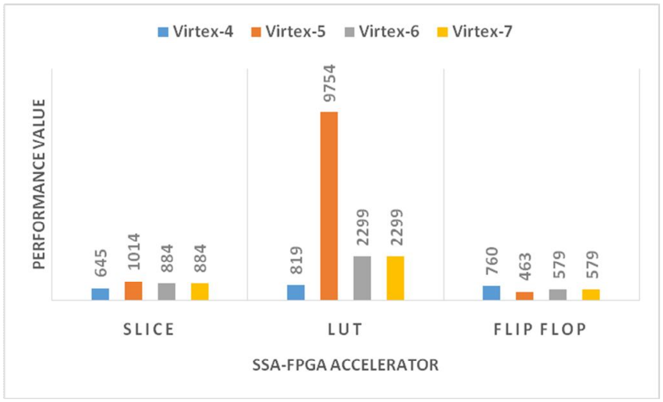


Figure 4: FPGA experimental result for SSA-FPGA Accelerator

The Table.3 shows the experimental results of FPGA performance for SSA-FPGA accelerator. In this research work, the FPGA accelerator design implemented in the Xilinx tool based on different FPGA devices such as Virtex-4, Virtex-5, Virtex-6, and Virtex-7. In the FPGA accelerator, two 53-bit given to the input of the FPGA accelerator, which produced the 106-bit output. In Figure.4 denotes the RTL schematic of 64-bit multiplier using SSA multiplication technique. In the Figure.5, the XL-53bits and RL-53bits are the input operand, and Y = 105 bits represent the output multiplier. The Figure.6 shows the internal block of the SSA multiplier. In this blue colored wires are interconnecting wiring network of the SSA circuit. The SSA components are internally connected through wiring network to build the real circuit.

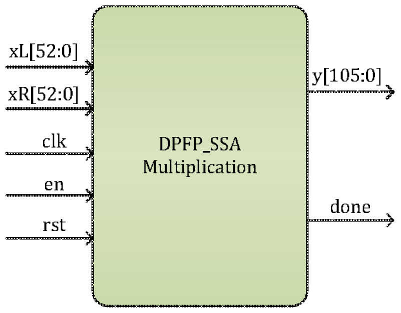


Figure 5: RTL schematic of 64-bit SSA multiplier

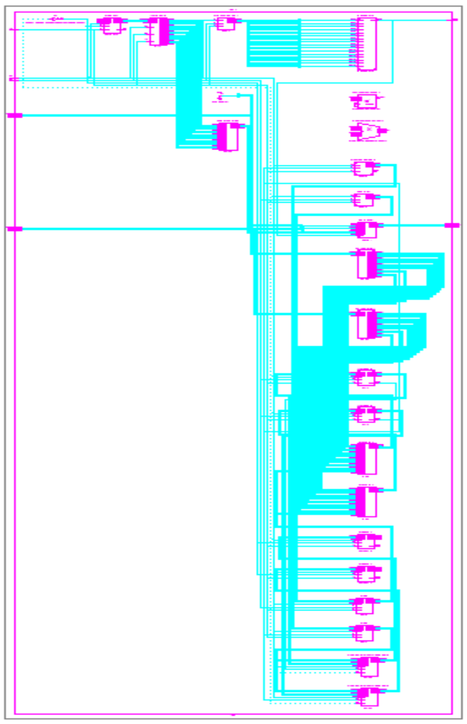


Figure 6: Internal blocks of SSA

The Figure. 7 shows the RTL schematic of the FPGA accelerator taken from Xilinx tool. In this work, the FPGA accelerator designed by using 16-bits address data, 32-bits write data are the input of the FPGA accelerator, it has provided accel_out 64-bits. The figure shows the internal blocks of the FPGA accelerator design. Figure.8 shows how the components are internally associated. It also shows how the circuits are actually arranged in the FPGA accelerator. The Figure.9 took from Xilinx tool shows the Virtex-5 FPGA results for verification purpose.

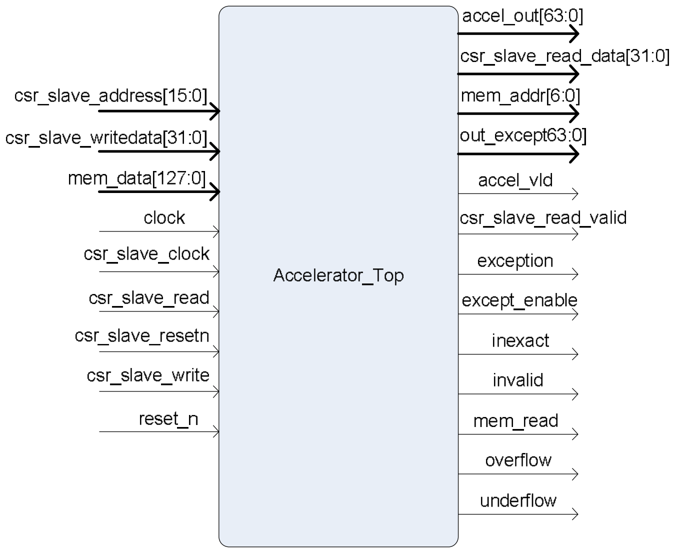


Figure 7: RTL schematic of the FPGA accelerator.

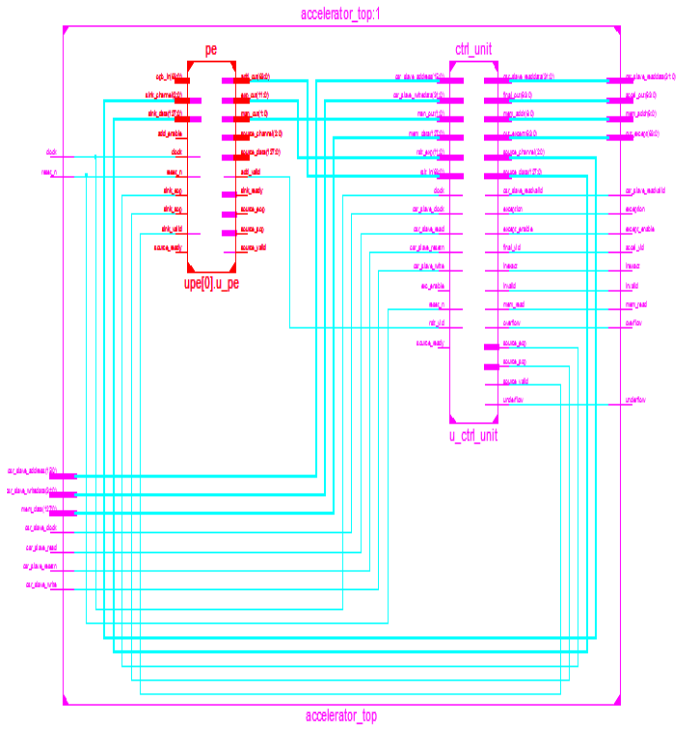
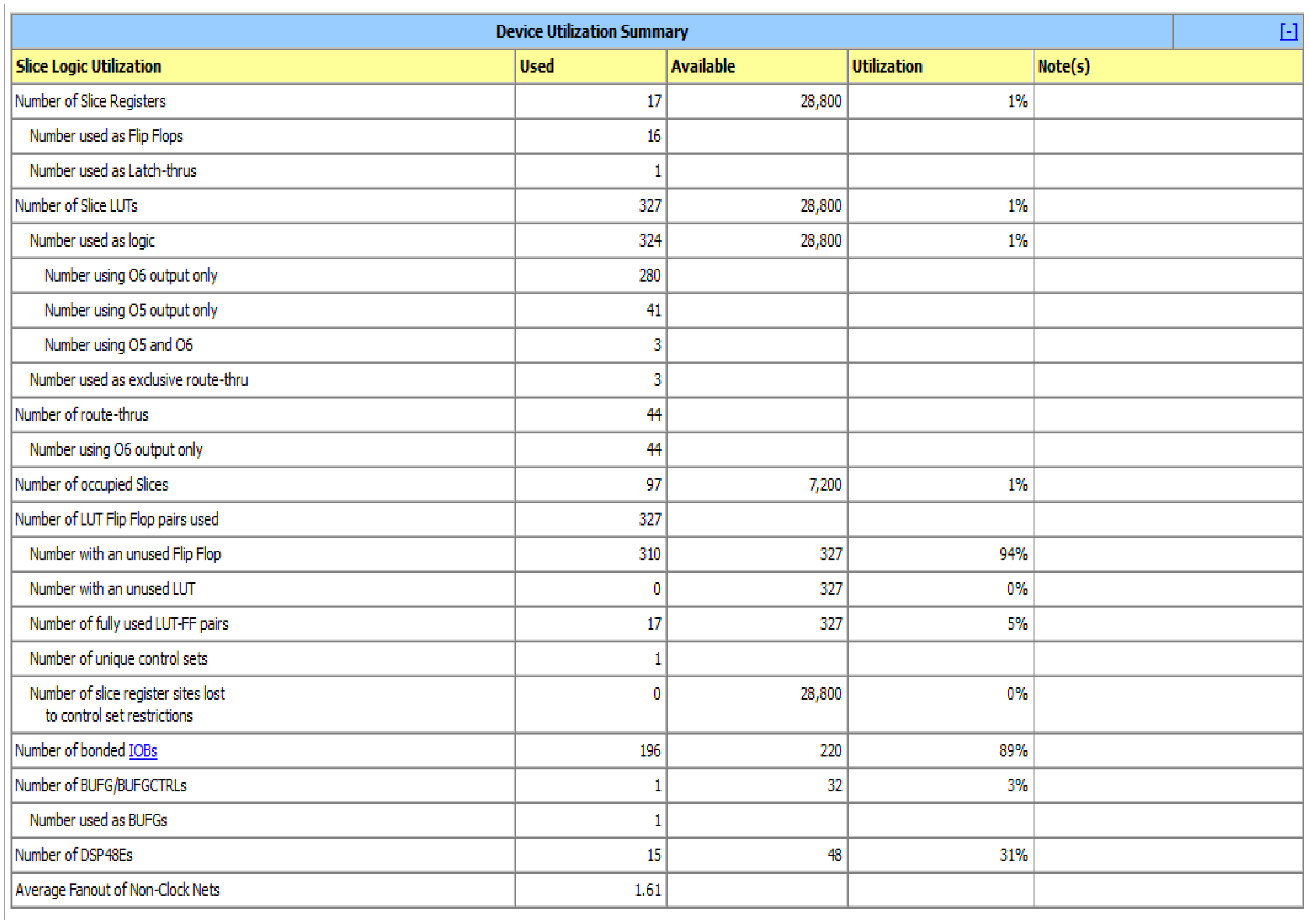


Figure 8: An internal block of the FPGA accelerator



Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	17	28,800	1%	
Number used as Flip Flops	16			
Number used as Latch-thrus	1			
Number of Slice LUTs	327	28,800	1%	
Number used as logic	324	28,800	1%	
Number using O6 output only	280			
Number using O5 output only	41			
Number using O5 and O6	3			
Number used as exclusive route-thru	3			
Number of route-thrus	44			
Number using O6 output only	44			
Number of occupied Slices	97	7,200	1%	
Number of LUT Flip Flop pairs used	327			
Number with an unused Flip Flop	310	327	94%	
Number with an unused LUT	0	327	0%	
Number of fully used LUT-FF pairs	17	327	5%	
Number of unique control sets	1			
Number of slice register sites lost to control set restrictions	0	28,800	0%	
Number of bonded IOBs	196	220	89%	
Number of BUFG/BUFGCTRLs	1	32	3%	
Number used as BUFGs	1			
Number of DSP48Es	15	48	31%	
Average Fanout of Non-Clock Nets	1.61			

Figure 9: Virtex-5 FPGA results

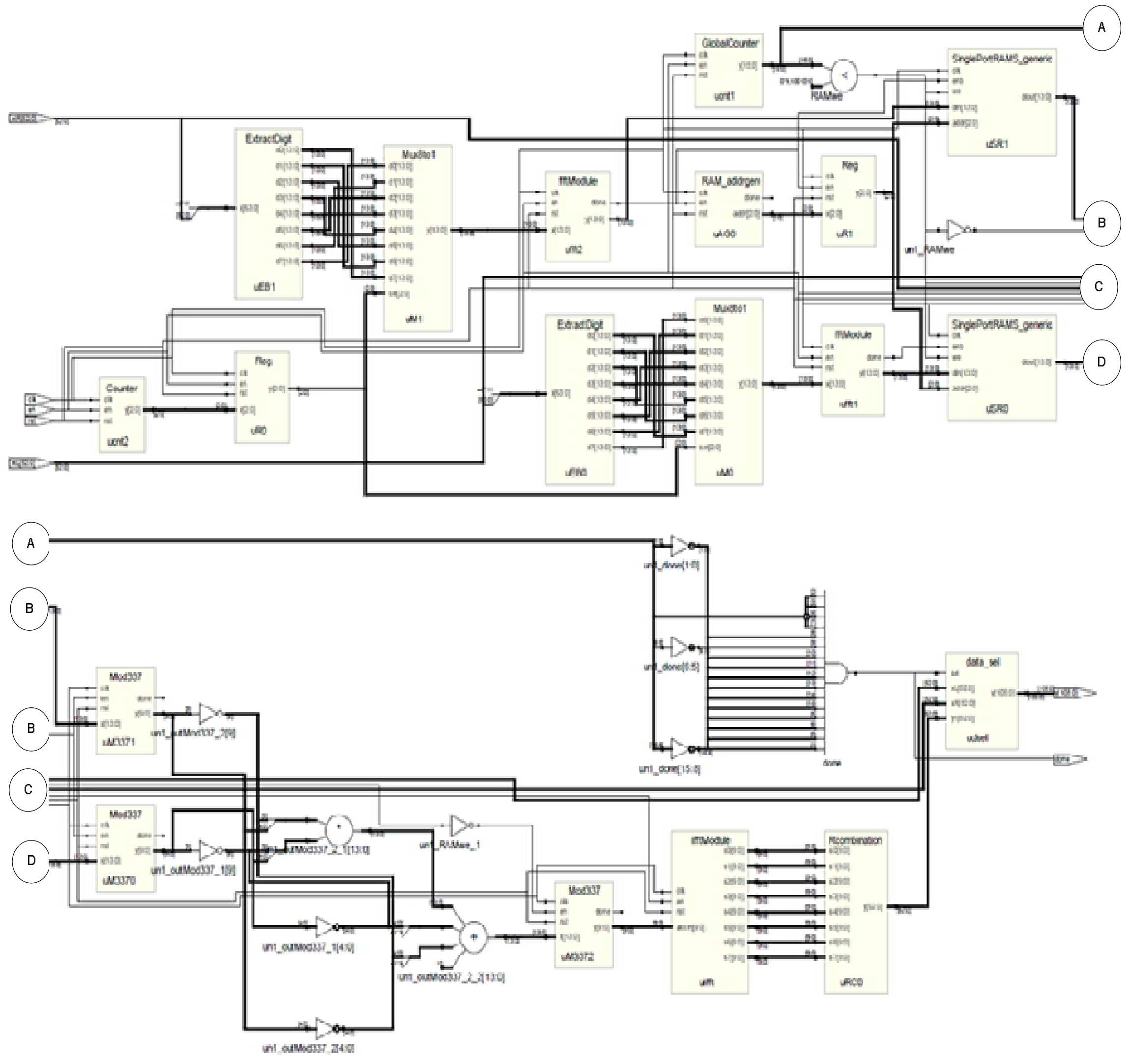


Figure 10: RTL schematic diagram of SSA-FPGA method

In Figure.10 shows the RTL schematic diagram of the SSA-FPGA method, which is taken from the synplify pro tool. The 3-bit input value gives to the counter. A counter is a device which is stores the number of times a particular has occurred, often in relationship to a clock cycle. The counter outputs are stored in the register. Next, the xR and xL bits are connected into extract digit module. That extract digit value gives to the mux module. The FFT is applied to the mux output which value is stored in the RAM. Find the mod of the RAM output after that IFFT algorithm is applied. At the final stage, then add all values based on the recombination technique. This work is further proceeded to generate GDS-II file using cadence Encounter tool gpdk 90nm technology. In this process, we have simulated SSA module using cadence NCLaunch tool. We have generated net-list file using cadence RTL compiler using gpdk 90nm technology. Later, we have done floor planning, added power rings strips, placement and routing using Encounter Tool. We have eliminated negative slack by choosing proper clock frequency. The SSA-FPGA Accelerator chip layout diagram is taken encounter tool shown in Figure.11.

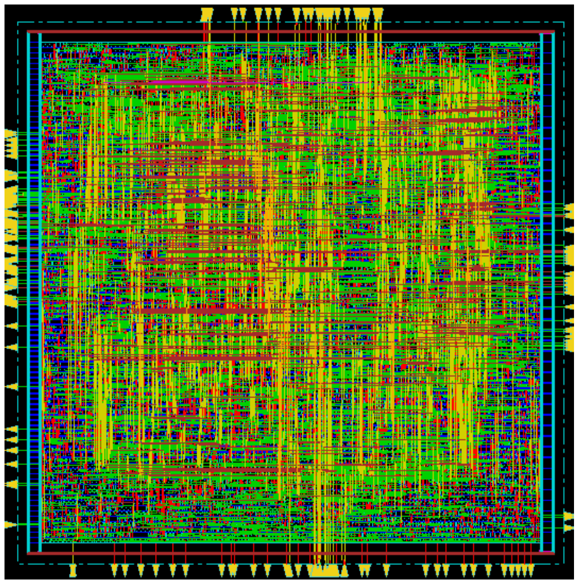


Figure 11: SSA-FPGA Accelerator Chip Layout

5. CONCLUSION

In this research work, the proposed SSA-FPGA method implemented in Xilinx Virtex-5 xc5vlx20T by using Verilog code. The DPFP using SSA multiplier for FPGA accelerator reduced the number of LUT, slices, flip-flop and increases the operating frequency in the FPGA accelerator. Furthermore, the FPGA accelerator synthesized in the Xilinx tool for different Virtex devices such as Virtex-4, Virtex-5, Virtex-6, and Virtex-7. The HS-SSA-FPGA method achieved low area and increase the system speed compared to the existing methods. In this experimental research work, the proposed method improved the FPGA performances such LUT, slice, flip-flop which has less area compared to karatsuba and Vedic multiplication algorithm. The future scope of FPGA accelerator is, it can play in diverse workloads in machine learning, cloud segments and other applications.

ACKNOWLEDGMENT

The authors express sincere gratitude to Dept. of. ECE at K L University for their encouragement during this work. Further, M. Siva Kumar would like to express his gratitude to DST through grant EEQ/2016/000604.

REFERENCES

1. Ramesh, Addanki Purna, A. V. N. Tilak, and A. M. Prasad. **An FPGA based high speed IEEE-754 double precision floating point multiplier using Verilog,** *Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System* (IC4EVENT), 2013 International Conference on. IEEE, 2013. https://doi.org/10.1109/ICEVENT.2013.6496575
2. Addanki, Purna Ramesh, Venkata Nagaratna Tilak Alapati, and Mallikarjuna Prasad Avana. **An FPGA based High Speed IEEE-754 double precision floating point Adder/Subtractor and Multiplier using Verilog**, *International journal of advanced science and technology* 52.1 (2013): 61-74.
3. Ramesh, Addanki Purna, A. V. N. Tilak, and A. M. Prasad. **FPGA based Implementation of High Speed Double Precision Floating Point Multiplier with Tiling Technique using Verilog**, *International Journal of Computer Applications* 58.21 (2012). https://doi.org/10.5120/9407-3814
4. Pratas, Frederico, Aleksandar Ilic, Leonel Sousa, Horácio C. Neto, and Rua Alves Redol. **Double-precision floatingpoint performance of computational devices: FPGAs, CPUs, and GPUs**, *Proc. of REC2010-VI Jornadas sobre Sistemas Reconfiguráveis* (2010): 83-90.
5. Doröz, Yarkin, Erdinç Öztürk, and Berk Sunar. **Evaluating the hardware performance of a million-bit multiplier**, *Digital System Design* (DSD), 2013 *Euromicro Conference on. IEEE*, 2013. https://doi.org/10.1109/DSD.2013.108
6. Rathor, Ajay, and Lalit Bandil. **FPGA implementation of Floating-Point Arithmetic**, *International Journal of Advanced Research in Computer Science and Electronics Engineering* (IJARCSEE) 1.9 (2012): pp-67.
7. Amaricai, Alexandru, Oana Boncalo, and Constantina-Elena Gavriliu. **Low-precision DSP-based floating-point multiply-add fused for Field Programmable Gate Arrays,** *IET Computers & Digital Techniques* 8.4 (2014): 187-197. https://doi.org/10.1049/iet-cdt.2013.0128
8. Zhang, Hao, Dongdong Chen, and Seok-Bum Ko. **High performance and energy efficient single-precision and double-precision merged floating-point adder on FPGA,** *IET Computers & Digital Techniques* 12.1 (2017): 20-29. https://doi.org/10.1049/iet-cdt.2016.0200
9. B. Srikanth, Dr. M. Siva Kumar, Dr. K. Hari Kishore, Dr. J.V.R. Ravindra, **Towards Reducing Area and Power of A Multiplier With Double Precision Floating Point Computations Using Fpga Accelerators**, *Journal of Advanced Research in Dynamical and Control Systems* Vol. 9. Sp– 18 / 2017.
10. Zhang, Hao, Dongdong Chen, and Seok-Bum Ko. **Area-and power-efficient iterative single/double-precision merged floating-point multiplier on FPGA**, *IET Computers & Digital Techniques* 11.4 (2017): 149-158. https://doi.org/10.1049/iet-cdt.2016.0100
11. Srinivasa Rao, T Subhashini, K Rambabu, **Design and Performance Analysis Of Double Precision Floating Point Multiplier Using Urdhva Tiryagbhyam Sutra**, *International Journal Of Vlsi And Embedded Systems-Ijves*, Vol 05, Article 10470, October 2014.
12. Jovanović,Ž. and V. Milutinović. **FPGA accelerator for floating-point matrix multiplication,** *IET Computers & Digital Techniques* 6.4 (2012): 249-256. https://doi.org/10.1049/iet-cdt.2011.0132
13. Jaiswal, Manish Kumar, and Ray CC Cheung. "**VLSI implementation of double-precision floating-point multiplier using karatsuba technique,** *Circuits, systems, and signal processing* 32.1 (2013): 15-27. https://doi.org/10.1007/s00034-012-9457-3
14. Ramesh, Addanki Purna, A. V. N. Tilak, and A. M. Prasad. **FPGA based Implementation of High Speed Double Precision Floating Point Multiplier with Tiling Technique using Verilog**, *International Journal of Computer Applications* 58.21 (2012). https://doi.org/10.5120/9407-3814
15. Ramireddy Venkata Suresh, K.Bala, **VLSI Implementation of High Speed and Area Efficient Double-Precision Floating Point Multiplier**, *International Journal of Advanced Research in Electronics and Communication Engineering* (IJARECE) Volume 5, Issue 10, October 2016.
16. Arish, S., and R. K. Sharma. **An efficient binary multiplier design for high speed applications using Karatsuba algorithm and Urdhva-Tiryagbhyam algorithm**, *In Communication Technologies* (GCCT), 2015 Global Conference on, pp. 192-196. IEEE, 2015. https://doi.org/10.1109/GCCT.2015.7342650