Asian Journal of Information Technology 15 (12): 1906-1915, 2016
ISSN: 1682-3915
© Medwell Journals, 2016

FFT Implementation with Fused Floating Point Dot Product Using Radix-8 Butterfly

[1]Sharmila Hemanandh and [2]A. Sivasubramanian
[1]Sathyabama University, Tamil Nadu, India
[2]Tagore Institute of Engineering and Technology, Deviyankurichi, Tamil Nadu, India

Abstract: This study describes Fast Fourier Transform implementation using fused floating point operations in parallel fashion. The Fast Fourier Transform processors use butterfly unit for computations on complex data. These operations are performed by FFT processors using complex butterfly operations that consist of multiplication, addition and subtraction operations. The main contribution in this research includes a radix-8 butterfly unit with higher efficiency. Also this butterfly unit performs faster than the conventional butterfly. The area required is reduced with the use of FFT Floating Point Butterfly unit as compared to the conventional butterfly unit. The complete architecture is synthesized and simulated using Xilinx ISE Software. The comparison of our proposed method with similar FFT architecture using radix-4 exhibited about 26.36% reduction in area and about 50.22% reduction in overall power consumption.

Key words: Fast Fourier Transform (FFT), radix-8, butterfly unit, floating point, Xilinx

INTRODUCTION

Fast Fourier Transform (FFT) is used in many applications like spectrum analyzers, multidimensional transform, spectral music, OFDM based wireless broadband communication system and many signal processing applications. Discrete Fourier Transform is a powerful tool to perform frequency analysis of a signal. The main disadvantage being the number of computations required to calculate the DFT coefficients. Fast Fourier Transform is an efficient algorithm that avoids redundant calculations in computing the DFT of a signal. The FFT algorithm also reduces the computation time as well as computation complexity when compared to direct computation of DFT.

Types of FFT algorithms

Decimation in Time (DIT): In Decimation in time algorithm, the input sequence is divided into subsequences, i.e., even and odd sequences. These subsequences are again divided in possible manner. The DFTs of these sequences are calculated and then finally combined. In this algorithm, sequence is split into smaller subsequences in time domain. Figure 1 show how the Decimation in Time algorithm works. Here, we have considered 8-Point DFT using Decimation in Time algorithm. The steps are given below:

- First 8-points are divided into two N/2 points DFTs
- Then, these N/2 point DFTs are again divided into N/4 DFTs
- Finally, outputs of these N/4 DFTs are combined to get FFT

We divide until, we get 2-point DFT as shown in Fig. 2-5.

Decimation in Frequency (DIF): In decimation in Frequency algorithm, instead of dividing the sequence in time, Frequency samples of DFT are decomposed into smaller sequences. In this algorithm frequency samples are divided into even and odd samples. This decomposition is continued till we get the 2-point DFTs. In Decimation in Frequency algorithm, symmetry of decomposition is reversed from Decimation in time, i.e., decomposition proceeds from left to right. Figure 6 shows the final flow graph of 8-point DFT using Decimation in Frequency FFT algorithm.

Realization of DIF And DIT FFT: Now days, FFT processors are very popular in the field of digital signal processing applications (Takahashi, 2003). The Fast Fourier Transform is one of the important functional unit in various signal processing and wireless communication applications (Ayinala *et al.*, 2012).

Corresponding Author: Sharmila Hemanandh, Sathyabama University, Tamil Nadu, India

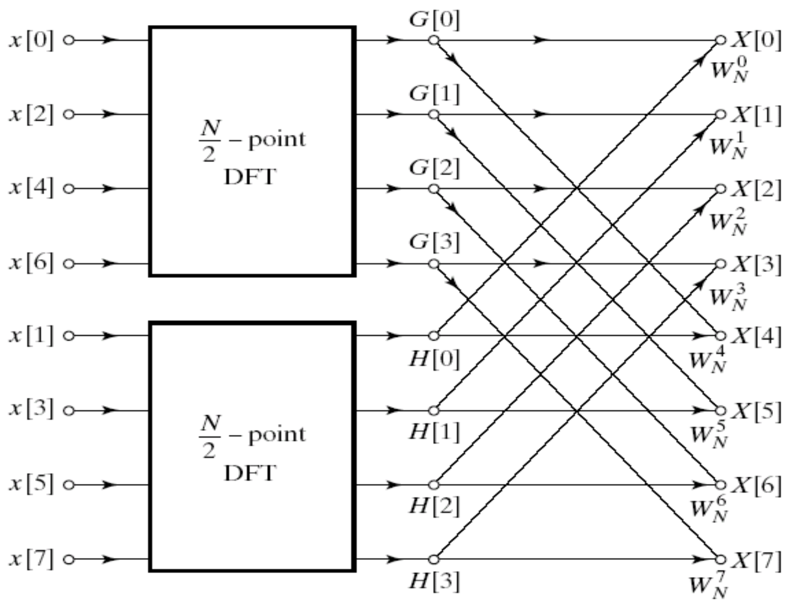


Fig. 1: Division in N/2-point DFTs

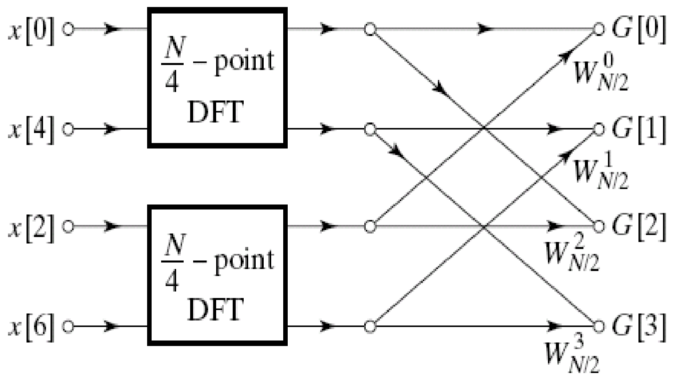


Fig. 2: Division in N/4-point DFTs and combining the outputs

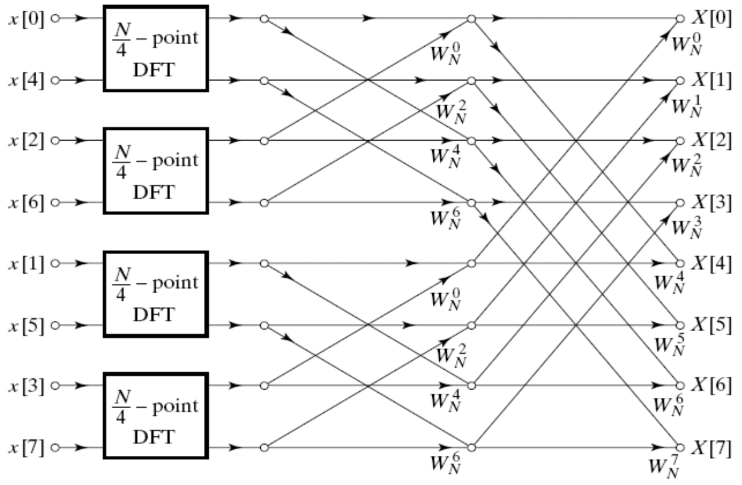


Fig. 3: Combine N/4 DFTs

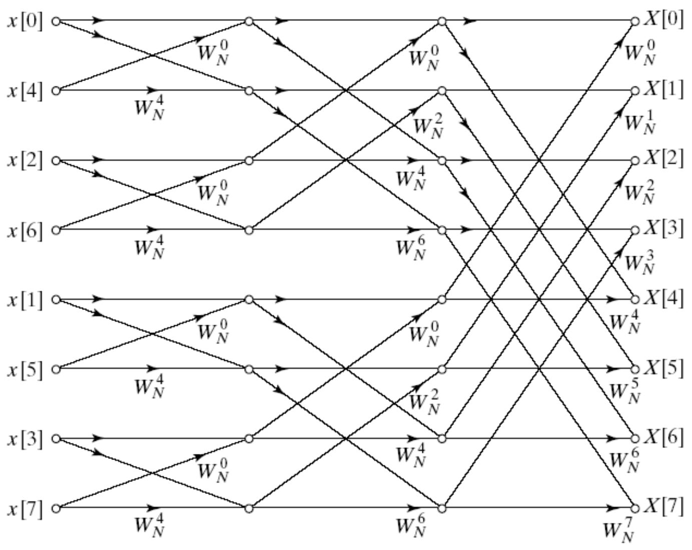


Fig. 4: Final 8-point DFT using decimation in time

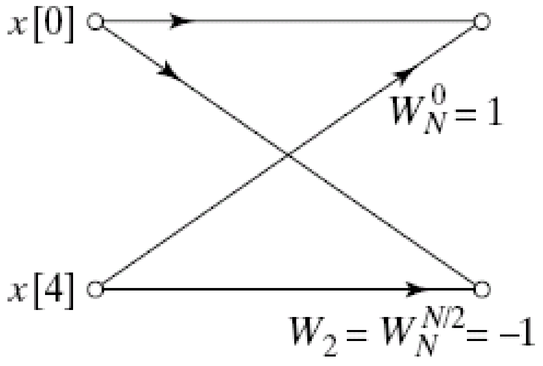


Fig. 5: The 2-point DFT

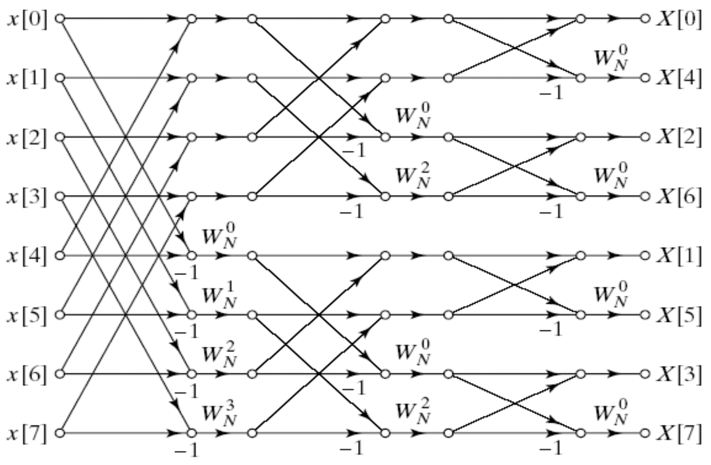


Fig. 6: Final 8-point DFT using decimation in frequency

The N-point FFT is computed using butterfly operations. Depending on the size of butterfly, FFT can be called as radix-r FFT where ‘r’ is the size of butterfly. The basic radix-r architecture is very well known for the

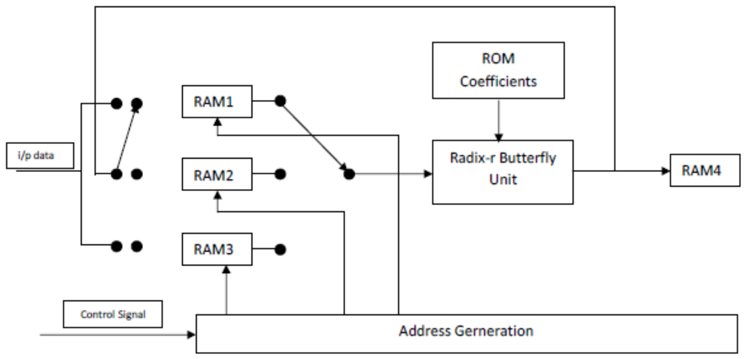


Fig. 7: Basic architecture of FFT processor

use of FFT processors. This radix-r butterfly unit is used to calculate the FFT of input signal. The radix-r butterfly consists of two main blocks. One Computational Elements (CE) and other is Twiddle Factor (TF). Radix-r FFT processor contains these two blocks along with one additional block called Reordering Element (RE). Signal Data from the input flows in one direction along the lines. These lines carry the r-complex data. While, other lines carry r-1 complex data from Twiddle Factor block to calculate the FFT of the input signal. These computational units vary depending upon the type of FFT algorithm used. i.e., decimation in Time or Decimation in Frequency algorithm.

Decimation in Time FFT butterfly unit calculates the FFT as complex multiplication followed by sum and difference network whereas Decimation in Frequency FFT butterfly unit calculates the FFT as sum and difference network followed by multiplication. The N radix-r butterfly unit calculates the r N-point FFT. The data rate on the FFT processor line can be calculated as r*clock rate on each clock cycle. From this we can conclude that radix-8 is 8 times faster than radix-2 and 2 times faster than radix-4.

Basic FFT structure: Figure 7 shows the general block diagram for the FFT processor. General FFT processor consists of RAM bank and radix-r butterfly unit. Radix-r butterfly unit consists of complex multiplication and add-subtract unit. The address generator generates the location of the input in the RAM bank that is to be fed to the butterfly unit. The butterfly unit computes the output by utilizing the coefficients that are stored in the ROM and the outputs thus computed are reloaded in the RAM banks. The process is repeated for the assigned number of stages. All the operations in the butterfly are carried out with use of Multipliers and adders. Previously Discrete fused floating point multipliers and adders are used in butterfly unit to calculate FFT. Second generation RISC CPU are implemented by using Multiply-add Fused Floating Point Unit (MAF). MAF can replace floating point multipliers and floating point adders.

Floating point multiplication is very important application in the DSP applications that are related to large dynamic range (Salehi *et al*., 2013) like Binary 32, 64, 128. Floating point multiplication of two numbers can be done in the following sequence:

- Add exponent of two numbers and subtract the bias from their result
- Multiply the significant of two numbers
- Sign of the final result can be calculated by XORing the sign of two numbers

In conventional method, multipliers or shift-add multipliers does multiplication of the significand bits of two floating point numbers. This multiplication is done in serial fashion and that is relatively slow. If they are performed in parallel fashion then silicon area on FPGA platform and power consumption increases. Hence conventional multipliers are not efficient for FFT computations. But the parallel method of floating point multiplication provides the best throughput. This study describes the power efficient method for floating point multiplication in parallel implementation that can be used in radix-8 and 4 butterfly unit.

Fused floating point arithmetic: In the previous years fixed point arithmetic is much popular than floating point arithmetic, since the delay is high, requires more area and power consumption is increased (Lienhart *et al*., 2006; Allan and Luk, 2001). But in the recent years, floating point arithmetic is more attractive in the implementation of Digital Signal Processors as it reduces the computational complexity. Now a days, after the realization of general purpose processors, fused floating point multipliers are given much attention than discrete floating point multipliers as it reduces the area on FPGA and also reduces the power consumption (Quinnell *et al*., 2008).

In case of FFT, complex butterfly operations are used to compute the transform. The computation of radix-2 butterfly requires 1 complex multiplication and 1 complex addition and subtractions operation. Whereas, radix-4 requires 3 complex multiplication and 8 complex addition and subtractions. This study presents, the radix-8 butterfly unit with fused dot product and Fused add-subtract unit for multiplication and additiion-subtract operation.

Two-term Fused Floating Point dot product is the extension of Fused Multiply-Add (FMA) (Huang *et al*., 2012) Unit. Equation 1 and 2 denotes conventional dot product and fused dot product, respectively:

$$X = PQ + RS \tag{1}$$

$$X = PQ \pm RS \tag{2}$$

Fused dot product allows the difference of the product too along with addition that is useful in the implementation of complex multiplication.

Another operation performed using fused floating point arithmetic is the fused add-subtract operation. Computing the sum and difference of two floating point operands is used many times in DSP algorithms. Fused add-subtract unit is based on conventional add-subtract unit. Fused add-subtract unit performs the addition and subtraction on same pair of floating point operands in parallel fashion. Equation 3 given below shows the fused add-subtract operation on same operands:

$$\left.\begin{array}{l} X = P + Q \\ X = P + Q \end{array}\right\} \tag{3}$$

Related work: Over the years many floating point architecture are proposed for FFT computation. Hokenek *et al.* (1990) proposed floating point with multiply-add fused architecture for Second generation RISC chips. Multiply-add Fused (MAF) Unit described increases throughput. It increases floating point performance and accuracy of complex floating point calculations. It can execute the double precision instruction, i.e., W = (X×Y)+Z in pipeline manner in two cycles with just one rounding error. About 40 ns cycle time was comparable to other CMOS RISC systems. Also, the performance of this floating point unit gets extended to Bipolar RISC systems.

Low-cost reconfigurable architecture for FFT processor was proposed by Xiao *et al.* (2008). Fixed point FFT is used in this architecture. It employs shared memory and pipelined architecture design to make single port SRAM available. Core area and power is greatly reduced at the cost of Execution time.

Fused Floating Point arithmetic architecture was proposed by Hani and coauthors for DSP applications. Fused Floating Point unit is used in radix-2 butterfly unit for the computation of FFT. This simplifies the computation complexity. When, compared to conventional floating point multipliers and adders, area is reduced to 72% using fused FFT butterfly. Also, the time required for floating point operation is reduced to 87% to that of conventional floating point operation.

Ashrafy *et al.* (2011) proposed efficient floating point multiplier. Overflow and Underflow cases in the computation of compex operations in FFT is handled by this multiplier. This gives more precision when used in Multiply and Accumulate Unit excluding the Rounding operation. The performance is improved due to pipelining. This also increases the maximum operating frequency of the multiplier.

Seidel and Even (2004) proposed delay optimized implementation of Floating Point Addition. Floating point adder is presented for Floating point addition and subtraction operation. Using various optimization technique, the proposed Floating Point adder (FP Adder) reduces the latency. All rounding modes of IEEE standards are supported by this architecture and the output is rounded, normalized as per IEEE standard.

Low power multiplier with bypassing and tree structure was proposed by Kuo and coauthors. In this technique switching activities are minimized using bypassing in the multiplier. Also, the critical path is decreased due to tree structure of the multiplier. So, the low power advantage is achieved in this multiplier architecture.

Garrido *et al.* (2013) implemented the radix-2^k feedforward FFT architectures. The implemented designs include radix- 2^2, radix-2^3 and radix- 2^4 architectures. The paper showed that radix-2^k could be used for any number of parallel samples with a power of two. Accordingly, radix-2^k FFT architectures for 2, 4 and 8 parallel samples were bestowed. Those architectures were shown to be more hardware-efficient than previous feedforward and parallel feedback designs in the literature. That makes them very attractive for the computation of the FFT in the most demanding applications.

A high-speed low-complexity modified radix-25 FFT Processor for High Rate WPAN Applications was implemented by Cho *et al.* (2011). In their research, the modified radix-2^5 algorithm and the eight parallel data-path 512-point modified radix 25 FFT processor have been implemented with 2.5 GS/s for OFDM-based WPAN applications. The number of complex Booth multipliers and twiddle factor LUTs were reduced using the modified radix 2^5 algorithm. Their modified radix 2^5 FFT processor was the most area-efficient architecture for the eight parallel 512-point MDF FFT processors. The highest throughput rate is up to 2.5 GS/s at the clock frequency of 310 MHz. In addition, SQNR can reach 35 dB for 16-QAM modulations with a 12 bit word length. The architecture has potential applications in high-rate OFDM-based WPAN systems.

Problem statement: Addition, subtraction and multiplication are the major operations in an FFT processor. In conventional methods, these operations are performed using conventional adders and Multipliers. Among these the multiplier occupies more silicon area and power consumption when comparing with addition and subtraction. Hence most researches contribute on area and power reduction of multipliers. Mottaghi-Dastjerdi *et al*. (2009) proposed low power multiplier called bypass zero that minimized switching activities of the shift add multiplier. Digital signal processing is classified into two major categories based on their representation: fixed and floating point. Floating point DSPs typically use a minimum of 32 bits to store each value whereas the Fixed point DSPs usually represent each number with a minimum of 16 bits. However, with DSPs the speed is about the same, a result of the hardware being highly optimized for math operations. The internal architecture of a floating point DSP is more complicated than for a fixed point device. All the registers and data buses must be 32 bits wide instead of only 16; the multiplier and ALU must be able to quickly perform floating point arithmetic. Floating point arithmetic units when used leads to better precision in results with more hardware and less throughput. Hence, in recent years various architectures on fused floating point are proposed that can reduce the delay and area on FPGA platform for the processors. The floating-point FMA has several advantages over discrete floating-point adders and multipliers in a general purpose processor. The FMA reduces the latency of a multiplication followed by an addition. Also, a single FMA may be used to replace the floating-point adder and the floating-point multiplier in a system. Some DSP algorithms have been rewritten to take advantage of the presence of FMA units.

This study describes fused floating point operation using radix-8 butterfly unit in the FFT computation. Modified fused dot product unit for radix-8 butterfly is proposed. The proposed architecture is implemented using radix-8 DIT FFT butterfly with fused floating point arithmetic. This architecture also proposes a power efficient method of multiplier for floating point multiplication.

MATERIALS AND METHODS

Proposed method

Fused floating point dot product unit: As discussed earliar, Fused dot product Unit is same as conventional dot product that extend to calculate the difference of two dot products along with addition. Proposed Fused Floating Point Dot Product Unit is shown in Fig. 8. It consists of four main blocks that perform the complex multiplication operation as follows:

- Exponent compare
- Two proposed radix-8 multipliers
- Alignment
- Leading zero anticipator and normalize
- Rounding

The Fused Dot Product Unit computes the sum and difference of the two term dot product as we know. From Fig. 8, 2's complement is used for subtraction of two term dot product.

Exponent logic extracts the exponent and adds them and subtracts the bias. The Exponent compare element consists of two 8-bit exponent adders. These adders add the exponents of the input C&D and A&B following single precision IEEE floating point format If two exponent adders are used the delay is twice when compared to the exponent adder that works in parallel.

Alignment element determines the operation to be performed between sum and carry of A*B and C*D. Alignment block consists of Shifter block. Shifter block arranges the output sum and carry of the two term dot product significand multiplication i.e. output sum and carry of C×D and A×B significand multiplication.

Leading Zero Anticipator has Pre-encoder and Leading Zero Detector as shown in Fig. 9. Normalization counts the leading zeros in the output of significand adder and shifts the sum to left to get leading one as the left most digit. So the Normalization is done with the help of LZA (Leading Zero Anticipator).

The result is always rounded to one of the selected IEEE rounding modes in the case of floating point arithmetic. Rounding is followed by post normalization in case of overflow. Post normalization after rounding is accomplished by shifting the bits.

Proposed radix-4 and 8 butterfly unit: In this study, we investigate the performance of radix 4 and radix 8 butterfly unit designed using the proposed fused dot product unit.

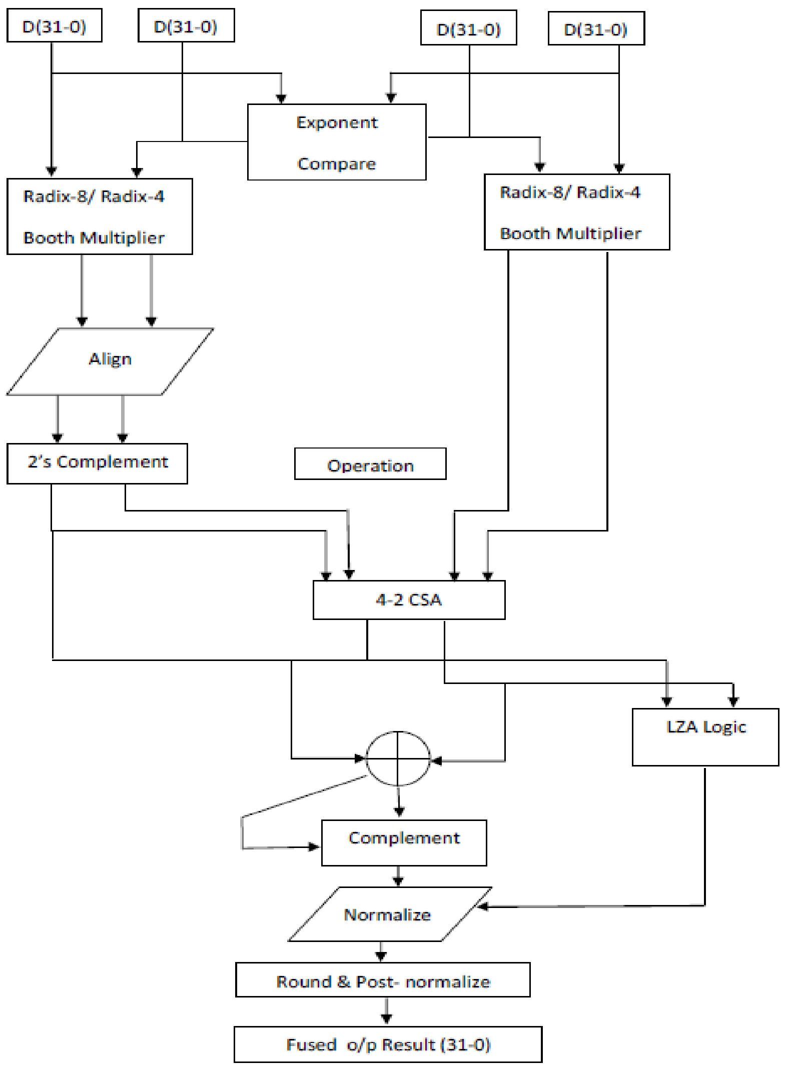


Fig. 8: Fused floating point dot product unit

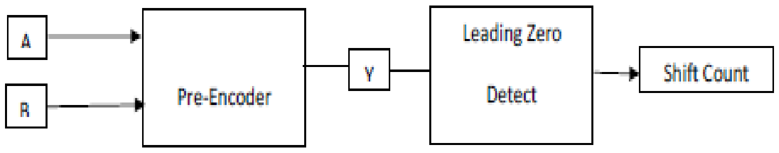


Fig. 9: Leading zero logic unit

The radix-4 Butterfly is shown in Fig. 10, radix-4 FFT reduces the number of stages required in the computation of FFT algorithm. But, the number of computations required to compute radix-4 FFT is more. The number of multiplications required to compute radix-4 FFT is less when compared to radix-2 FFT, to compute FFT of the same size. This is shown in Fig. 10. The discrete implementation of radix-4 butterfly requires 12 multipliers and 22 adders to perform 3 complex multiplications and 8 complex additions. But with the proposed fused implementation of radix-4 butterfly requires 6 Fused Dot Product (FDP) Units and 8 Fused Add-Subtract (FAS) Units to perform 3 complex multiplications and 8 complex additions required for radix-4 butterfly. This is shown in Fig. 11.

In a similar way, radix-8 butterfly is designed taking the advantage of fused dot product unit for FFT implementation. This radix-8 butterfly again reduces the

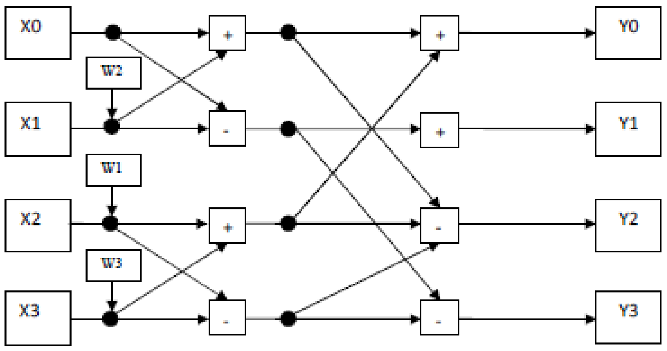


Fig. 10: The radix-4 DIT FFT butterfly unit

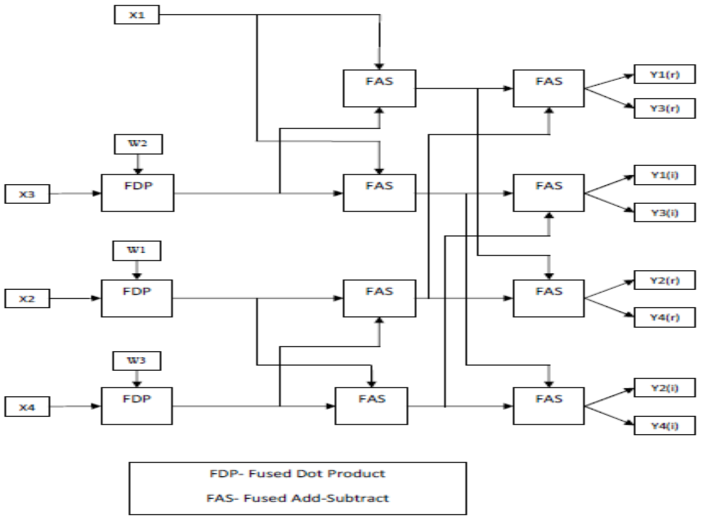


Fig. 11: Fused Floating Point implementation using radix-4

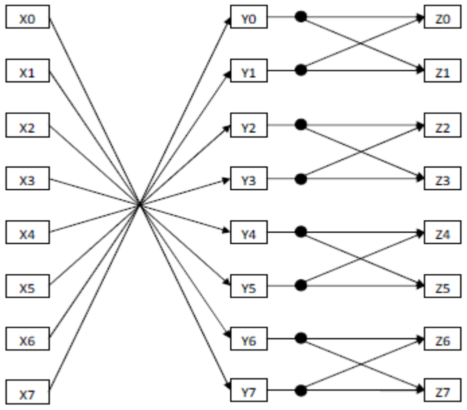


Fig. 12: The radix-8 butterfly

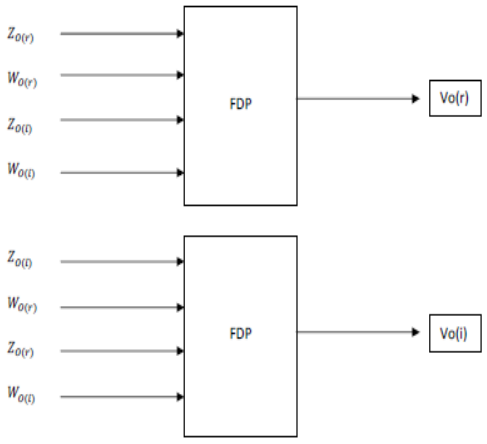


Fig. 13: Fused dot product unit using radix-8

number of stages and number of complex computations in the FFT algorithm. It requires less number of multiplications than radix-2 and 4 butterfly of same size. Radix-8 butterfly unit is shown in Fig. 12.

Figure 12-14 shows the implementation of radix-8 FFT using Fused Dot Product Unit (Fused DP) and Fused Add-Subtract (FAS) Unit.

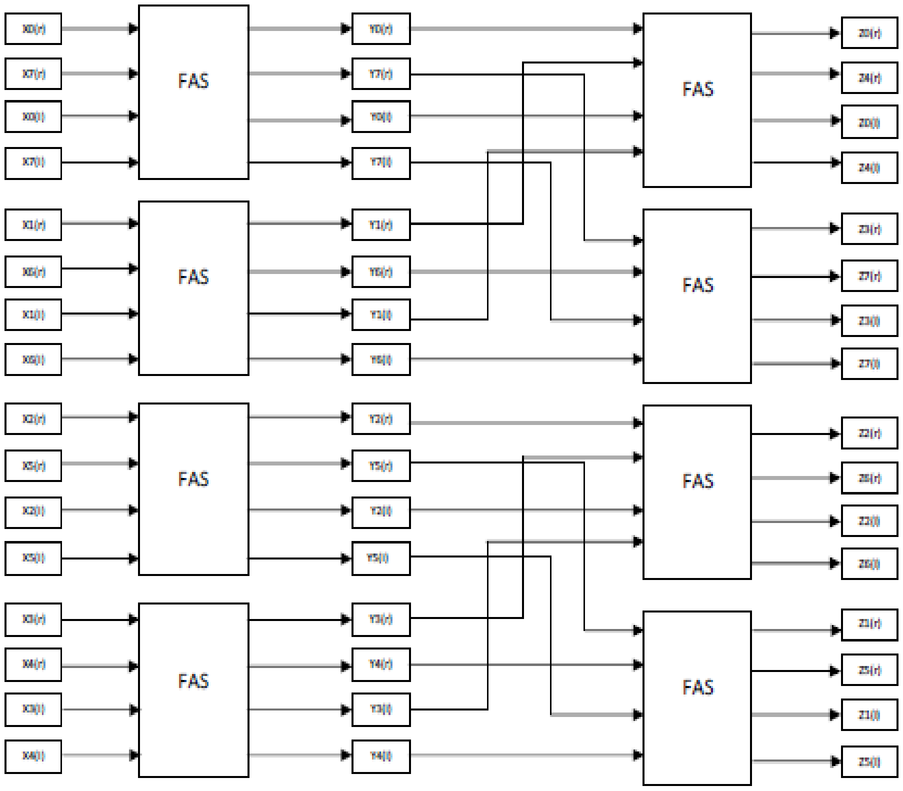
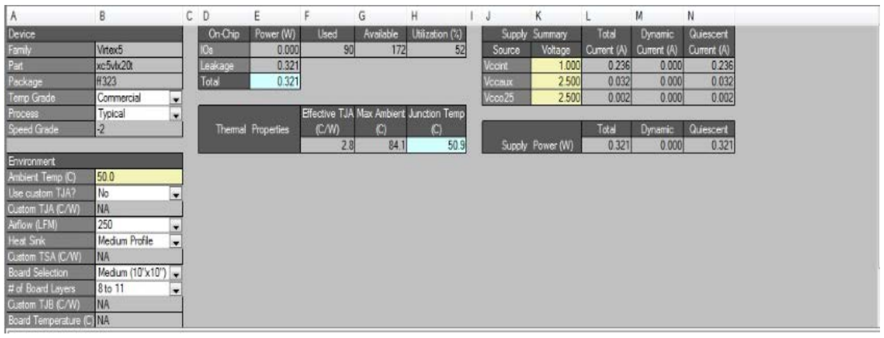


Fig. 14: Fused add-subtract unit using radix-8 butterfly



Device	
Family	Virtex5
Part	xc5vlx20t
Package	ff323
Temp Grade	Commercial
Process	Typical
Speed Grade	-2

Environment	
Ambient Temp (C)	50.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	250
Heat Sink	Medium Profile
Custom TSA (C/W)	NA
Board Selection	Medium (10"x10")
# of Board Layers	8 to 11
Custom TJB (C/W)	NA
Board Temperature (C)	NA

On-Chip	Power (W)	Used	Available	Utilization (%)
IOs	0.000	90	172	52
Leakage	0.321			
Total	0.321			

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	2.8	84.1	50.9

Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.000	0.236	0.000	0.236
Vccaux	2.500	0.032	0.000	0.032
Vcco25	2.500	0.002	0.000	0.002

	Total	Dynamic	Quiescent
Supply Power (W)	0.321	0.000	0.321

Fig. 15: Power consumption in the proposed radix-8 Fused Floating point butterfly

RESULTS AND DISCUSSION

In this study, the experimental results of our proposed method related to area and power are presented. The synthesis and simulation for the proposed module is performed using Xilinx ISE Software. The area report can be obtained from the device utilization summary generated from the synthesis of our design. Area of the proposed radix-8 Fused Floating point butterfly is shown in the following Table 1.

The proposed design use 844 slice registers among the available 12480 units in the target device. About 2948 slice LUTs out of 12480 are only used by our proposed design contributing about 23% utilization of the available. A 100% utilization of IOBs is obtained by the proposed architecture. The power report for our design is obtained using the Xpower analyzer tool in Xilinx ISE. Figure 14 shows the power consumption in the proposed radix-8 Fused Floating point butterfly.

Proposed radix-8 Fused Floating point butterfly is area efficient than radix-4 and 2. Power consumed by the fused floating point butterfly unit is also less. Comparing to radix-4 our method shows 26.36% savings in cell area and about 50.22% reduction in power consumption (Fig. 15).

Table 1: Area of proposed radix-8 Fused Floating point butterfly

	Device utilization summary (estimated values)		
Logic utilization	Used	Available	Utilization (%)
No. of slice register	844	12480	6
No. of slice LUTs	2948	12480	23
No. of fully used LUT-FF pairs	411	3381	12
No. of bonded IBOs	172	172	100
No. of BUFG/BUFGCTRLs	1	32	3

Table 2: Area and power comparison

	Butterfly unit	
Parameters	FP based radix-8	FP based radix-4
Cell area	1,84,184 um^2	250099 um^2
Power	142 mw	225 mw

CONCLUSION

In this research, a FFT implementation with fused floating point Dot Product using radix-8 butterfly parallel fashion was proposed. The background and the literatures relating to our work were clearly discussed and the research methodology was also analyzed. The complete description of our proposed method that we have discussed in Section 4 was coded in verilog-HDL using Xilinx-ISE. The generated results after the synthesis and simulation were also presented under the section results and discussion. The comparison of our proposed method with the similar architecture using radix-4 exhibited about 26.36% reduction in area and about 50.22% reduction in overall power consumption.

REFERENCES

Allan, J. and W. Luk, 2001. Parameterised floating-point arithmetic on FPGAs. Proceedings of the 2001 IEEE International Conference on Acoustics Speech and Signal Processing, (ICASSP'01), May 7-11, 2001, IEEE, Salt Lake City, Utah, ISBN: 0-7803-7041-4, pp: 897-900.

Ashrafy, M.A, A. Salem and W. Anis, 2011. An efficient implementation of floating point multiplier. Proceedings of the 2011 Saudi International Conference on Electronics, Communications and Photonics (SIECPC), April 21-26, 2011, IEEE, Riyadh, Saudi Arabia, ISBN: 978-1-4577-0068-2, pp: 1-5.

Ayinala, M., M. Brown and K.K. Parhi, 2012. Pipelined parallel FFT architectures via folding transformation. IEEE. Trans. Very Large Scale Integr. (VLSI) Syst., 20: 1068-1081.

Cho, T., H. Lee, J. Park and C. Park, 2011. A high-speed low-complexity modified radix-25 FFT processor for gigabit WPAN applications. Proceedings of the 2011 IEEE International Symposium on Circuits and Systems (ISCAS), May 15-18, 2001, IEEE, Rio de Janeiro, Brazil, ISBN: 978-1-4244-9473-6, pp: 1259-1262.

Garrido, M., J. Grajal, M.A. Sanchez and O. Gustafsson, 2013. Pipelined radix-feedforward FFT architectures. IEEE. Trans. Very Large Scale Integr. (VLSI) Syst., 21: 23-32.

Hokenek, E., R.K. Montoye and P.W. Cook, 1990. Second-generation RISC floating point with multiply-add fused. IEEE. J. Solid-State Circuits, 25: 1207-1213.

Huang, L., S. Ma, L. Shen, Z. Wang and N. Xiao, 2012. Low-cost binary128 floating-point FMA unit design with SIMD support. IEEE Trans. Comput., 61: 745-751.

Lienhart, G., A. Kugel, and R. Manner, 2006. Rapid development of high performance floating-point pipelines for scientific simulation. Proceedings of the 20th IEEE International Symposium on Parallel and Distributed Processing, April 25-29, 2006, IEEE, Rhodes Island, Greece, ISBN: 1-4244-0054-6, pp: 1-8.

Mottaghi-Dastjerdi, M., A. Afzali-Kusha and M. Pedram, 2009. BZ-FAD: A low-power low-area multiplier based on shift-and-add architecture. IEEE. Trans. Very Large Scale Integr. Syst., 17: 302-306.

Quinnell, E., Jr., Swartzlander, E. Earl and C. Lemonds, 2008. Bridge floating-point fused multiply-add design. IEEE. Trans. Very Large Scale Integr. (VLSI) Syst., 16: 1727-1731.

Salehi, S.A., R. Amirfattahi and K.K. Parhi, 2013. Pipelined architectures for real-valued FFT and hermitian-symmetric IFFT with real datapaths. IEEE. Trans. Circuits Syst. II: Express Briefs, 60: 507-511.

Seidel, P.M. and G. Even, 2004. Delay-optimized implementation of IEEE floating-point addition. IEEE. Trans. Comput., 53: 97-113.

Takahashi, D., 2003. A radix-16 FFT algorithm suitable for multiply-add instruction based on Goedecker method. Proceedings of the 2003 International Conference on Multimedia and Expo, ICME'03, July 6-9, 2003, IEEE, New York, USA., ISBN: 0-7803-7965-9, pp: II-845.

Xiao, H., A. Pan, Y. Chen and X. Zeng, 2008. Low-cost reconfigurable VLSI architecture for fast fourier transform. IEEE. Trans. Consum. Electron., 54: 1617-1622.