International Journal of Engineering & Technology, 7 (4) (2018) 2646-2650

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet.v7i4.16063
Research paper

Implementation of fast FFT design for 128-point using Radix-2^2 CFA

Manish Bansal [1] *, Sangeeta Nakhate [1]

[1] EC Dept. MANIT Bhopal
**Corresponding author E-mail: manishbansal2008@gmail.com*

Abstract

In this paper, implementation of fast FFT design for 128- point using Radix-2^2 CFA is presented. This research uses a common factor algorithm which is based on Radix-2^2. A 2-point DFT butterfly structure is the lowest complexity structure and Radix-2^2 CFA is used to reduce logic and area by reducing the number of twiddle factors. The VHDL code is written and synthesized using Xilinx FPGA device xc7vx330t-3ffg1761 to implement the proposed design. This design is coded in VHDL and MATLAB. VHDL code is targeted to synthesize into Xilinx Virtex-7 FPGA and simulated into ModelSim PE Student Edition 10.4a. MATLAB code is simulated into MATLAB 2012. The proposed design achieves 149.822 MHz clock frequency, used 2802 slices on the Virtex-7 and SQNR 33.49 dB at 16-bit I/O word length.

Keywords: *CFA; Complex Multiplier; FFT; OFDM; Radix-2^2; SDF.*

1. Introduction

The FFT is a key component of a communication system and widely used for many applications in engineering, science, and mathematics. OFDM is a multicarrier modulation technology where the subcarriers are orthogonal to one another. The use of FFT made the OFDM practical and popular. MCM (Multicarrier modulation) is a technique to transmit data by dividing it into many components and transmitting each component over different carrier signals. Each carrier has low frequency, but the composite signal has a high frequency. FFT is used at the receiver side in several diverse wirelesses and wired applications, including video broadcasting and digital audio, digital subscriber lines (DSL), and wireless LANs [1] as shown in figure 1. Nowadays, Fast schemes have attracted many researchers for the FFT designs [2]. Pipeline-based, memory based and general-purpose DSP are few of the method to implement Fast Fourier Transform. Memory-based needs many computation cycles but most area efficient. The throughput rate of pipeline-based architecture is high at a low frequency and small chip area.

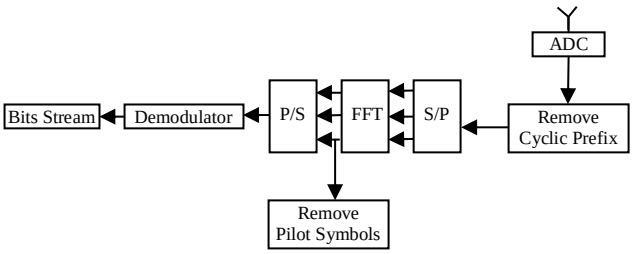


Fig. 1: Receiver of OFDM Communication System.

There are different pipelined structures to implement Fast Fourier Transforms - single path delay feedback (SDF) where first half of outputs from each stage are stored back to the same stage memory when the inputted data are passed to butterfly, single path delay commutator (SDC) where one path is used between two stages and first stage output is commutate to next stage delay element, multiple path delay commutator (MDC) where more than one path is used between two stages and first stage output is commutate to next stage delay element. The SDF structure is more efficient among these structures as –

1) SDF structure introduces a feedback technique to reduce the memory in comparison to MDC and SDC structures.
2) SDF structure is simpler to design the fixed point FFT [3].

The proposed FFT processor controller is simpler as overall structure and all components like-complex multiplier, trivial circuit, stages, registers and twiddle bank is controlled by counter signal. This prevalence presents the research interest for implementing 128 – point FFT processor with better performance. Moreover, such implementations should offer higher speed and lower area. This leads the scope for a novel approach for wireless communications applications. FFT design is influenced with radix. As radix increases, butterfly complexity increases and twiddle factors decreases. As radix decreases, butterfly complexity decreases and the number of twiddle factors increases. The proposed FFT is designed using Radix-2^2 CFA that uses the lesser twiddle factors in comparison of Radix-4 and uses 2-point DFT butterfly that is simpler butterfly structure.

2. Implementation design of the proposed FFT using radix-2^2 CFA

The proposed FFT is implemented for 128-point using Radix-2^2 CFA (common factor algorithm) that is the most popular algorithm.

Copyright © 2018 Manish Bansal, Sangeeta Nakhate. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This algorithm uses 2-point DFT butterfly and the butterflies are connected in SDF pipeline structure to design 128-point FFT. This structure is structure that is more efficient and reduces the FFT design complexity. Each stage of the proposed design consists of a 2-point DFT butterfly. The proposed FFT design, common factor algorithm based on Radix-2^2, 2-point DFT butterfly, trivial circuit are described under this section for 128-point FFT.

The DFT is to compute the X(k) sequence of complex-valued numbers N given another sequence of data x(n) of length N, expressed as [4]:

$$X(k) = \sum_{n=0}^{N-1} x(n).e^{\frac{-j2\pi nk}{N}} \quad (1)$$

$$0 \le k \le N-1$$

Where $e^{\frac{-j2\pi nk}{N}}$ is a twiddle factor W_N^{nk}. n represents time sequence and k represents frequency sequence. The common factor algorithm is formulated using Radix-2^2 and 3-dimensional liner index mapping. The Radix-2^2 common factor algorithm for 128-point is expressed as [5] –

For 3- Dimensional Index Mapping –

$$n = 64n_1 + 32n_2 + n_3 \quad (2)$$
$$k = k_1 + 2k_2 + 4k_3 \quad (3)$$

Where

$$n_1, n_2 = 0, 1 \text{ and } n_3 = 0, 1, \ldots\ldots, 31$$
$$k_1, k_2 = 0, 1 \text{ and } k_3 = 0, 1, \ldots\ldots, 31$$

The CFA is applied in this work as [4] –

$$X(k_1 + 2k_2 + 4k_3) = \sum_{n_3=0}^{31}\sum_{n_2=0}^{1}\sum_{n_1=0}^{1} x(64n_1 + 32n_2 + n_3).\ W_N^{nk}$$
$$= \sum_{n_3=0}^{31}\sum_{n_2=0}^{1}\sum_{n_1=0}^{1} x(64n_1 + 32n_2 + n_3).\ W_N^{(64n_1+32n_2+n_3)(k_1+2k_2+4k_3)} \quad (4)$$

The W_N^{nk} twiddle factor is expanded as follows –

$$W_N^{nk} = W_N^{(64n_1+32\,n_2+n_3)(k_1+2k_2+4k_3)}$$
$$= \underbrace{(-1)^{n_1k_1}}_{\text{BF}}\underbrace{(-j)^{n_2k_1}}_{\text{PF}}\underbrace{(-1)^{n_1k_2}}_{\text{BF}}\underbrace{W_{128}^{n_3(k_1+2k_2)}}_{\text{PF}} W_{32}^{n_3k_3} \quad (5)$$

Where n_1, n_2, n_3 represents the time index terms of the input data sequence n and k_1, k_2, k_3 are the frequency index terms of the output data sequence k. $(-1)^{n_1k_1}$ and $(-1)^{n_1k_2}$ are 2-point DFT butterfly factor (BF). $(-j)^{n_2k_1}$ and $W_{128}^{n_3(k_1+2k_2)}$ are processing factor (PF).

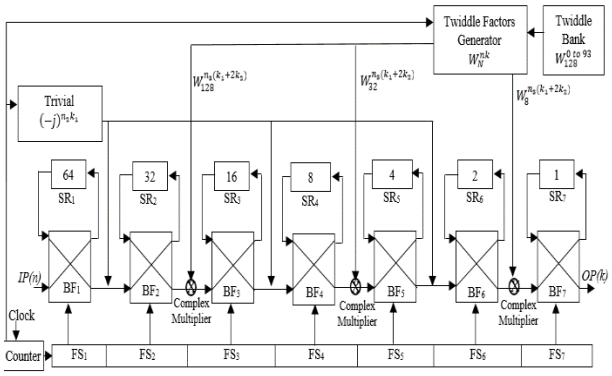


Fig. 2: Structure of the Proposed FFT for 128-Point.

The structure of the proposed design for 128-point is shown in figure 2 that retains seven numbers of FFT stages - FS_1, FS_2, FS_3, FS_4, FS_5, FS_6, FS_7 and shift registers - SR_1, SR_2, SR_3, SR_4, SR_5, SR_6, SR_7. These shift registers save complex number values. For 128-point, SR_1 saves 64 values, SR_2 saves 32 values, SR_3 saves 16 values, SR_4 saves 8 values, SR_5 saves 4 values, SR_6 saves 2 values and SR_7 saves 1 value. Twiddle Factors are produced by twiddle factor generator as per $W_{128}^{n_3(k_1+2k_2)}$, $W_{32}^{n_3(k_1+2k_2)}$, $W_8^{n_3(k_1+2k_2)}$ using n_3, k_1, and k_2 values. The trivial circuit in case of $-j$ multiplication, inverts the sign of real part of a complex number then swaps inverted real part and imaginary part of a complex number. The complex multiplier multiplies between input data and twiddles factors data. The 2-point DFT butterfly performs bypass, addition and subtraction operations. The 2-point DFT butterfly (BU) is shown in figure 3 that conducts the operation between $(n + N/2)^{th}$ and n^{th} value in the 1st $N/2$ cycles. In this 1st $N/2$ cycles, when multiplexers, M_2, M_3, M_4 are on position "0" then data is bypassed from input side to the shift registers and stored in shift registers, and in the next $N/2$ cycles, when multiplexers M_1, M_2, M_3, M_4 are switched to position '1' then 2-point DFT butterfly starts subtraction and addition between incoming data and the data saved in the SRs [6]. The subtracted outputs are saved in the same stage SRs, first half number of added outputs are saved in the next stage SRs and second half number of added outputs start to subtract and add with the saved half added output data.

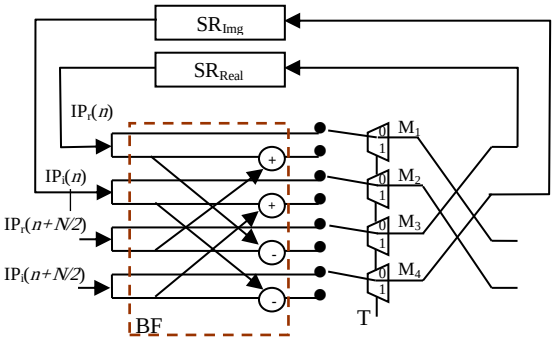


Fig. 3: 2-Point DFT Butterfly with the Shift Register.

In this way, this process runs from one stage to the next stage up to the final stage. These 2-point DFT butterflies of each stage are connected in pipelined structure to flow data serially in each clock.

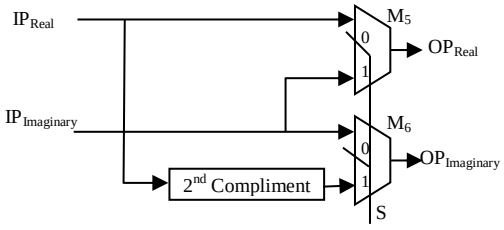


Fig. 4: Trivial Circuit.

The trivial circuit is shown in figure 4 that performs trivial $(-j)^{n_2k_1}$ multiplication. Trivial $(-j)^{n_2k_1}$ value can be either "1" or "$-j$". In case of "1" just bypass the input data instead of multiplication and in case of "$-j$" also just invert the sign of real part and swap the inverted real part and imaginary part instead of multiplication. The Multiplexers signal 'S' controls the bypass, sign inversion and swapping operation. When switch 'S' of multiplexors M_5 & M_6 is at '0' position then trivial circuit bypass input data. When switch S of multiplexors M_5 & M_6 turn to '1' then trivial circuit invert the sign of real part either from '+' to '–' or '–' to '+' by 2nd complement and then swap the inverted real part and an imaginary part. Thus, whenever "$-j$" multiplication is needed, sign inversion and swapping operations are performed there. This approach decreases the hardware and enhances the performance.

3. Data flow of the proposed design

The proposed design is implemented using Radix-2^2 CFA for 128-point. The 2-point DFT butterflies, FFT stages, shift registers,

twiddle factors generator, twiddle bank and trivial circuit are configured and operated by the counter using the value of n_1, n_2, n_3, k_1, k_2, k_3 as represented in figure 2.

Table 1: Processing Factors Parameters Values for the Proposed FFT

PEs / *N*	After 1st Stage $(-j)^{n_2k_1}$	After 2nd Stage $W_N^{n_3(k_1+2k_2)}$	After 3rd Stage $(-j)^{n_2k_1}$	After 4th Stage $W_N^{n_3(k_1+2k_2)}$	After 5th Stage $(-j)^{n_2k_1}$	After 6th Stage $W_N^{n_3(k_1+2k_2)}$	After 7th Stage $(-j)^{n_2k_1}$
128	$n_2, n_1 = 0, 1$ $k_2, k_1 = 0, 1$ $k_3, n_3 = 0, 1..., 31$ *N*=128		$n_2, n_1 = 0, 1$ $k_2, k_1 = 0, 1$ $k_3, n_3 = 0, 1..., 7$ *N*=32 and same repeats for every next 32-point up to 128-point		$n_2, n_1 = 0, 1$ $k_2, k_1 = 0, 1$ $k_3, n_3 = 0, 1$ *N*=8 same repeats for every next 8-point up to 128-point		No PF

The proposed design generates [7] stages, [7] SRs, and [1] twiddle generator. SR_1 saves 64 values, SR_2 saves 32 values, SR_3 saves 16 values, SR_4 saves [8] values, SR_5 saves 4 values, SR_6 saves 2 values and SR_7 saves 1 value. Twiddle $W_N^{n_3(k_1+2k_2)}$ generate twiddle factors $W_{128}^{n_3(k_1+2k_2)}, W_{32}^{n_3(k_1+2k_2)}$ and $W_8^{n_3(k_1+2k_2)}$. Trivial $(-j)^{n_2k_1}$ among FS_1 & FS_2 and twiddle factor $W_{128}^{n_3(k_1+2k_2)}$ among FS_2 & FS_3 perform operations as per 128-point samples. Trivial $(-j)^{n_2k_1}$ among FS_3 & FS_4 and twiddle factor $W_{32}^{n_3(k_1+2k_2)}$ among FS_4 & FS_5 perform operations as per 32-point samples, and the same repeats for every next 32-point up to 128-point. Trivial $(-j)^{n_2k_1}$ among FS_5 & FS_6 and twiddle factor $W_8^{n_3(k_1+2k_2)}$ among FS_6 & FS_7 perform operations as per 8-point samples, and the same repeats for every next 8-point up to 128-point as shown in table 1.

The data flow for the proposed design is shown in the signal flow graph (SFG) in figure 5 for 128-point. Initially out of 128 input point (IPs), IP(0) to IP(63) point are passed and saved from the input side to SR_1 by BF_1 in each clock cycle. Thus, it uses 64 clock cycles. At the 65th clock as IP(64) point enters in FS_1, BF_1 starts to add and subtract between IP(0) & IP(64) points, the added output is saved in SR_2 and subtracted output is saved in SR_1. At 66th clock as IP (65) point enters in FS_1, BF_1 again starts to add and subtract between IP(1) & IP(65) points, this added output is saved in SR_2 and subtracted output is saved in SR_1. In the same way, Data flows up to the 96th clock. At 97th clock as IP(96) point enters in FS_1, BF_1 again starts to add and subtract between IP(32) & IP(96) points, this subtracted output is saved in SR_1 and this added output as inputted in butterfly BF_2 then butterfly BF_2 also starts to add and subtract between this added output and first saved added data in SR_2 and then this subtracted output is saved in SR_2 and added output is saved in SR_3. This process continuously runs up to the 112th clock. Thus, at the 113th clock, BF_3 also starts to add and subtract along with BF_1 and BF_2. At the 121st clock, BF_4 also starts to add and subtract along with BF_1, BF_2, and BF_3. At 125th clock, BF_5 also start to add and subtract along with BF_1, BF_2, BF_3, and BF_4. At 127th clock, BF_6 also starts to add and subtract along with BF_1, BF_2, BF_3, BF_4, and BF_5. At 128th clock, BF_7 also starts to add and subtract along with BF_1, BF_2, BF_3, BF_4, BF_5, and BF_6, and added outputs are saved in next SR_3, SR_4, SR_5, SR_6, SR_7 registers and subtraction in SR_2, SR_3, SR_4, SR_5, SR_6, SR_7. BF_7 outputs are outputs of the proposed design FFT. In this way, the 1st output of the proposed design gets at the 128th clock cycle, the 2nd output gets at the 129th clock and so on.

In the proposed work, only one twiddle bank $W_{128}^{0\ to\ 93}$ is used instead of using three different twiddle banks W_{128}, W_{32}, W_8. The twiddle bank $W_{128}^{0\ to\ 93}$ has 94 values of twiddle factors from W_{128}^{0} to W_{128}^{93}. Twiddle generator generates twiddle factors for each position. After the 2nd stage, twiddles are based on W_{128} and after the 4th stage, twiddles are based on W_{32}, and after the 6th stage, twiddles are based on W_8. These all twiddles value either based on W_{128} or W_{32} or W_8 are fetched from twiddle bank $W_{128}^{0\ to\ 93}$ using periodicity $W_N^{k+N} = W_N^k$ and symmetric $W_N^{k+N/2} = -W_N^k$ properties. Figure 6 represents the number of twiddle factors used after every even stage and the method to find all twiddle values from twiddle bank. The method to find some of the twiddles value based on W_{32} and W_8 from twiddle bank $W_{128}^{0\ to\ 93}$ is shown in table 2. This method is reducing hardware as W_{32} and W_8 twiddle values are fetched from twiddle bank $W_{128}^{0\ to\ 93}$ instead of using different twiddle banks, and W_N^0 value wherever is required is bypassed rather than multiplying in the proposed design as $W_N^0 = 1$. Thus, these all techniques reduce hardware and increase the performance of the proposed design.

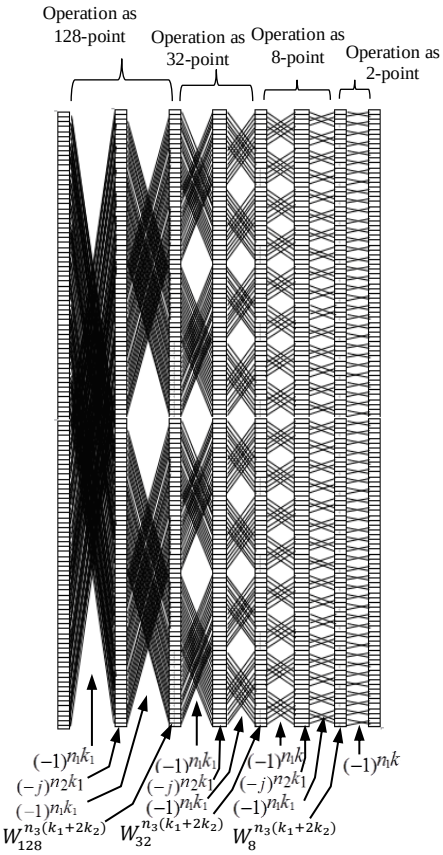


Fig. 5: SFG of the Proposed Design for 128-Point.

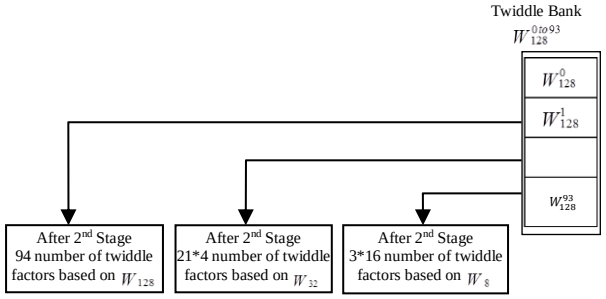


Fig. 6: Twiddle Values Fetched from Twiddle Bank $W_{128}^{0\ to\ 93}$.

Table 2: Method to Find the Twiddle W_{32} and W_8 Values from Twiddle Bank $W_{128}^{0\ to\ 93}$

PEs / N	After 2nd Stage $W_{128}^{n_3(k_1+2k_2)}$	After 4th Stage $W_{32}^{n_3(k_1+2k_2)}$	After 6th Stage $W_8^{n_3(k_1+2k_2)}$
128	$W_{128}^{60} = W_{128}^{60}$ $W_{128}^{24} = W_{128}^{24}$ $W_{128}^{93} = W_{128}^{93}$	$W_{128}^{56} = W_{32}^{14}$ $W_{128}^{28} = W_{32}^{7}$ $W_{128}^{84} = W_{32}^{21}$	$W_{32}^{32} = W_8^{2}$ $W_{128}^{16} = W_8^{1}$ $W_{128}^{48} = W_8^{3}$

Processing factors (PFs) - twiddle factors and trivial values are found out on the bases of n_1, n_2, n_3, k_1, k_2, k_3 values as represented in table 1. These processing factors do operations with inputted data at each point as represented in the signal flow graph (SFG).

These all addition, subtractions, by-pass operations, swapping and complex multiplications are controlled by counter signal.

4. Comparison

The proposed design is coded in VHDL and MATLAB. VHDL code is synthesized in xc7vx330t-3ffg1761 FPGA using Xilinx tool. Simulation results by simulating VHDL code using ModelSim and MATLAB code using MATLAB 2012R are verified with actual FFT results.

Table 3 states the comparison of the proposed design and conventional design [7] between the components used. Figure 7 states the comparison in maximum frequency and SQNR of the proposed design and conventional FFT. The comparison results show that the proposed design used 41% less number of twiddle factors which causes less number of multiplications count and uses less hardware.

Table 4 states the comparisons between minimum period, slices used, combinational path delay and max clock frequency parameters of the synthesized proposed design and conventional design [7]. Register – Transfer Level (RTL) view is shown in figure 8 that is generated after synthesizing VHDL code by Xilinx tool. This RTL view is a schematic of the VHDL source file. This schematic represents the design in terms of generic symbols, such as multipliers, adders, counters, OR gates, and AND gates.

Table 3: Comparison between Different Parameters

Design	Number of Point	Structure	Radix	Number of stages	Number of Twiddle Factors	Memory
R2^{2}SDF[7]	128	Pipeline	Radix -2^2	7	384	127
Proposed Design FFT	128	Pipeline	Radix -2^2	7	225	127

Table 4: Implementation Results

Parameters	FFT [7] Xilinx Virtex-4 FPGA	Proposed Design FFT Xilinx Virtex-7 FPGA
No. of Point (N)	128	128
No. of Slice Registers	64	989
No. of Slice LUTs	5633	2170
No. of fully used LUT-FF pairs	-	909
No. of bonded IOBs	-	102
Max Frequency (MHz)	140	149.822
SQNR	30 dB	33.49 dB
Min period in ns	-	6.675
Max combinational path delay (ns)	-	0.989

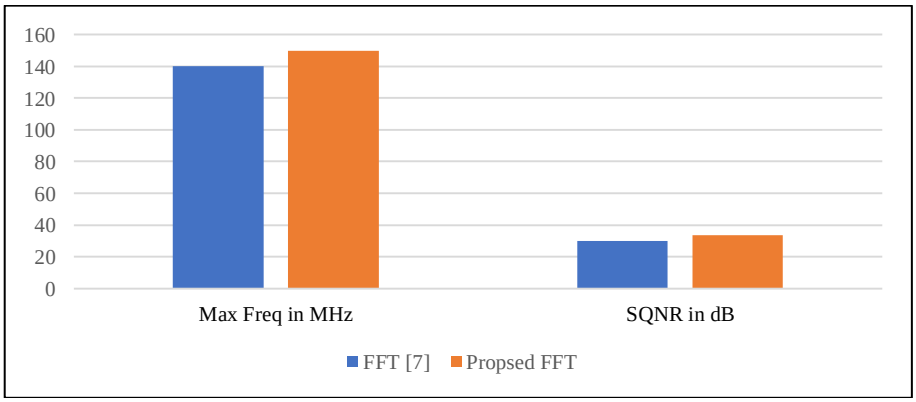


Fig. 7: Comparison between Frequency and SQNR of the Proposed and Conventional FFT.

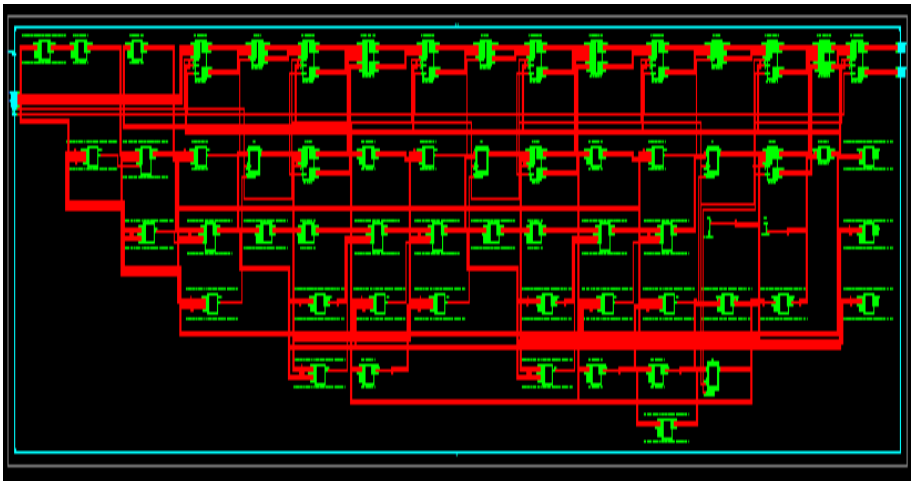


Fig. 8: Schematic (RTL) View of the Proposed Design.

5. Conclusion

The purpose of this research work is to design a high-performance FFT processor that can be used for 128-point FFT applications. This performance is enhanced by using approach Radix-2^2 CFA, swapping, reusing twiddle bank and bypass W_N^0 twiddle factor instead of multiplication that causes 41% less number of twiddle factors and less number of multiplications respectively than conventional design [7]. Thus, these all approaches reduce the memory, hardware, and the complexity, and enhances the performance of the proposed design.

References

[1] A. Saeed, M. Elbably, G. Abdelfadeel and M. I. Eladawy, "FPGA implementation of Radix-2^2 Pipelined FFT Processor", in *Proc. of the 3rd WSEAS Int. Symp. on Wavelets Theory and Applications in Applied Mathematics, Signal Processing & Modern Science*, 2009, pp. 109-114.

[2] Z. Wang, X. Liu, B. He, and F. Yu, "A Combined SDC-SDF Architecture for Normal I/O Pipelined Radix-2 FFT", *IEEE Trans. very large scale integration (VLSI) systems*, vol. 23 no. 5, 2014, pp. 973-977.

[3] J. Wang and L. A. Ronningen, "An Implementation of Pipelined Radix-4 FFT Architecture on FPGAs", *Journal of Clean Energy Technologies*, vol. 2, no. 1, 2014, pp. 101-103. https://doi.org/10.7763/JOCET.2014.V2.100.

[4] C.P. Fan, M.S. Lee, G.A. Su, "Efficient low multiplier cost 256-point FFT design with Radix-2^4 SDF architecture", *Journal of Engineering*, vol. 19, no. 2, 2006, pp. 61-74.

[5] K. Harikrishna, T. R. Rao, and V. A. Labay, "FPGA Implementation of FFT Algorithm for IEEE 802.16e (Mobile WiMAX)", *Int. Journal of Computer Theory and Engineering*, vol. 3, no. 2, 2011, pp. 197-203. https://doi.org/10.7763/IJCTE.2011.V3.305.

[6] W.H. Chang and T. Nguyen, "An OFDM-specified lossless FFT architecture", *IEEE Trans. on Circuits and Systems I*, vo. 53, no. 6, 2006, pp. 1235-1243. https://doi.org/10.1109/TCSI.2006.875167.

[7] H. Liu, and H. Lee, "A high performance four-parallel 128/64-point radix-2^4 FFT/IFFT processor for MIMO-OFDM systems", *IEEE Asia Pacific Conference on Circuits and Systems, (APCCAS)*, 2008, pp 834-837.

[8] Taesang Cho and Hanho Lee, "A High-Speed Low- Complexity Modified FFT Radix - 2^5 Processor for High Rate WPAN Applications", *IEEE Transactions on Very Large-Scale Integration (VLSI) Systems*, vol. 21, 2012, pp 187-191.

[9] V. Sarada, T. Vigneswaran, "Reconfigurable FFT Processor - A Broader Perspective Survey", *Int. Journal of Engineering & Technology*, vol. 5, no. 2, 2013, pp 949-956.

[10] T. Cho, H. Lee, J. Park, C. Park, "A high-speed low-complexity modified Radix - 2^5 FFT processor for gigabit WPAN applications", in *Proc. of IEEE Int. Symp. On Circuits and Systems*, 2011, pp. 1259–1262.

[11] U. Akshata and D. K. Gopika, "Hardware Implementation of Decimation in Time FFT," *MIT Int. Journal of Electronics and Communication Engineering*, vol. 3, no. 1, 2013, pp. 39-42.

[12] W.H. Chang and T. Nguyen, "Design and simulation of 64-point FFT using Radix - 4 algorithms for FPGA Implementation", *Int. Journal of Engineering Trends and Technology*, vol. 4, issue 2, no. 1, 2013, pp 109-113.

[13] J. Lee and H. Lee, "A High-Speed Parallel Radix-2^4 FFT/IFFT Processor for MB OFDM UWB System", in *IEICE Trans. Fundamentals*, vol. E91–A, no.4, 2013.

[14] M. Shin and H. Lee, "A High-Speed Four-Parallel Radix-2^4 FFT/IFFT Processor for UWB Applications", in *Proc. of IEEE Int. Symp. On Circuits and Systems*, 2008, pp. 960-963.

[15] M. Garrido, J. Grajal, M. A. Sanchez and O. Gustafsson, "Pipelined Radix-2^k Feedforward FFT Architectures", *IEEE Trans. on Very Large-Scale Integration Systems*, vol. 21, issue. 1, 2013, pp. 23-32. https://doi.org/10.1109/TVLSI.2011.2178275.

[16] S.-J. Huang and S.-G. Chen, "A green FFT processor with 2.5-GS/s for IEEE 802.15.3c (WPANs)", in *Proc. of Int. Conf. on Green Circuits and Systems*, 2010, pp 9–13.

[17] Weidong Li, Yutai Ma and Lars Wanhammar, "Word Length Estimation for Memory Efficient Pipeline FFT/IFFT Processors", *Conference on Signal Processing Applications & Technology (ICSPAT)*, 2006.

[18] X. Cui and D. Yu, "Digital OFDM transmitter architecture and FPGA design", in *Proc. of IEEE eighth Int. Conf. on ASIC*, 2009, pp 477-480.