Reduced Area and Low Power Implementation of FFT/IFFT Processor

Shefa A. Dawwd
Computer Engineering Department
University of Mosul
Mosul-Iraq
shefadawwd@gmail.com

Suha. M. Nori
Computer Engineering Department
University of Mosul
Mosul-Iraq
suhamudhafer@yahoo.com

Abstract *The Fast Fourier Transform (FFT) and Inverse FFT(IFFT) are used in most of the digital signal processing applications. Real time implementation of FFT/IFFT is required in many of these applications. In this paper, an FPGA reconfigurable fixed point implementation of FFT/IFFT is presented. A manually VHDL codes are written to model the proposed FFT/IFFT processor. Two CORDIC-based FFT/IFFT processors based on radix-2and radix-4 architecture are designed. They have one butterfly processing unit. An efficient In-place memory assignment and addressing for the shared memory of FFT/IFFT processors are proposed to reduce the complexity of memory scheme. With "in-place" strategy, the outputs of butterfly operation are stored back to the same memory location of the inputs. Because of using DIF FFT, the output was to be in reverse order. To solve this issue, we have re-use the block RAM that used for storing the input sample as reordering unit to reduce hardware cost of the proposed processor. The Spartan-3E FPGA of 500,000 gates is employed to synthesize and implement the proposed architecture. The CORDIC based processors can save 40% of power consumption as compared with Xilinx logic core architectures of system generator.*

***Index Terms*— Fast Fourier Transform, CORDIC, Field Programmable Gate Array, In-Place RAM.**

I. INTRODUCTION

Many applications of digital signal processing applications such as linear filtering [1], spectrum analysis [2], digital video broadcasting [3] and Orthogonal Frequency Division Multiplexing (OFDM) [4] employ the Discrete Fourier Transform (DFT) as a significant part of their design. However, an efficient algorithm for calculating the (DFT) is The Fast Fourier Transform (FFT). Only O(N log N) multiply and add operations are required in FFT to implement the DFT algorithm, vs. N^2 operations in the basic DFT. The difference in speed can be enormous, especially for long data sets where N may be in the thousands or millions. The Fast Fourier Transform (FFT) and its Inverse (IFFT) are widely used in communication system especially in orthogonal frequency division multiplexing(OFDM) systems (Wi-Fi) , wireless-LAN, ADSL, VDSL systems and WIMAX. Therefore, the attention to the real-time implementation of FFT/IFFT processors has been directed and efficient implementation of it became a significant topic of nowadays researches [5]. Many different hardware architectures for FFT/IFFT have been proposed for different applications and implemented in FPGAs. The small area, low power, and high-throughput is a significant challenge for efficient hardware realization of FFT/IFFT. The challenge is even more pronounced when large transform lengths (>1024 points) need to be realized in embedded hardware. The architectures of FFT processors mainly labeled as three types: the parallel architecture, the pipeline architecture and the sequential architecture. The parallel and pipeline architecture have more butterfly processing units to achieve high performance but they consume larger area than the sequential architecture. On the other hand, the sequential architecture requires only one butterfly processing unit and has the advantage of area efficiency. But the sequential architecture has a

drawback of low throughput and requires a complex circuit design of memory address controller. We focus on the sequential architecture for area efficiency and hardware simplicity [6].

In FFT/IFFT processor, the butterfly operation is the most computationally demanding stage. Traditionally, a butterfly unit is composed a complex adders and multipliers, and the multiplier is usually the bottleneck in FFT/IFFT processor The multiplication is the most expensive operation in the FFT/IFFT blocks. When *N* (the number of points) increases, the area and power consumption rapidly increases due to increasing demands of complex multiplier. So there is a need to reduce or eliminate the multipliers [3]. Of many ways to achieve this, the Coordinate Rotation Digital Computer (CORDIC) algorithm was chosen. In CORDIC algorithm, add and shift operation is only required as an alternative to the multiplication. This leads to reduce the required hardware to perform the butterfly process in FFT. In addition, as an alternative of using the ROM-based lookup table storing twiddle factors that consumes large area in case of long-length FFT computation, the CORDIC base FFT/IFFT processor needs to store only the twiddle factor angles in a small ROM for butterfly operation.

There is a growing number of recently reported works on variable length, fixed length, small area, and low power FFT processors. In what follows, some of these works are presented. In [7] an FPGA implementation of 16-point FFT processor based on radix-2,radix-4, and split-radix is proposed. Vedic multiplier is used to speed up the multiplication. A low power 1024-point split-radix FFT is presented in [8]. The proposed design achieve over 20% reduction on power when compare to radix-2 based FFT.

Due to the many advantages of CORDIC algorithm which helped to achieve low power, low area, low cost, flexibility and scalability for FFT/IFFT processor, the CORDIC algorithm has been shown to be an efficient way to obtain these specifications. The advantage of this algorithm is that it can be implemented with a very small FPGA footprint. It requires only a small lookup table, along with some other logic to perform shifts and additions. Importantly, the algorithm requires no dedicated multipliers or dividers. However, the large number of iterations that needed in CORDIC algorithm leads to increase the latency. Many published works and article addressed the above issues. Fixed point [9, 10], or floating point [11] representations of data are employed. Small size [12], scalable[13],large size[14,15], and variable size[16] CORDIC based FFT with different implementation techniques are designed.

In this paper, two scalable, configurable data width / sample points of FFT/IFFT processors are designed using Radix-2/Radix-4 and implemented in FPGA. The CORDIC algorithm is used to generate the twiddle factors. To enhance the speed and reduce the delay, pipeline CORDIC architecture is used. Also, using higher radix rather than of lower one will enhance the computation time for large number of points. The FFT/IFFT processor is designed using radix-4 and radix-2 algorithms.

The paper is organized as follow: section 2 presents the mathematic and background of CORDIC and different FFT/IFFT techniques. The CORDIC based FFT design and architecture of the proposed processor is given in some details in section 3. While the most important experimental results, discussions, and comparisons are presented in section 4. Finally, the proposed work is concluded in section 5.

II. FFT/IFFT ALGORITHM

A. FFT Butterfly Algorithm

The theoretical background and representation of different FFT algorithms, each with its advantages and disadvantages are presented in this section. Fast Fourier Transform is an efficient method for calculating the Discrete Fourier Transform (DFT) and it's inverse [17]:

$$X[k] = \sum_{n=0}^{N-1} x(n) W_N^{nk} \quad 0 \le k \le N-1 \qquad (1)$$

W_N is shorthand for $\exp^{-i2\pi/N}$ and represents the twiddle factor. The most common FFT algorithms is Cooley–Tukey FFT algorithm, which is called the divide and conquer algorithm that recursively breaks down a DFT of any composite size into many smaller DFTs sizes and combining them to get the total transform. There are basically two

typical forms of Cooley–Tukey FFT algorithm:the Decimation In Time(DIT) and the Decimation In Frequency(DIF). These two types are categorized according to the order of input and output samples.

A radix-2 DIT FFT is the simplest and most common form of the Cooley–Tukey algorithm. Radix-2 divides a DFT of size N into two interleaved DFTs (hence the name "radix-2") of size N/2 with each recursive stage [2]. If $N=2^v$, then $v=\log_2 N$. Now, the number of complex multiplication and the number of complex addition are reduced to ($N/2 \log_2 N$) and ($N \log_2 N$) respectively in comparison with the order of N^2 in DFT[18]. The butterflies for DIF and DIT are shown in Fig. 1a and Fig. 1b respectively.

The radix-4 FFT algorithm decimates the N-point input sequence of the discrete Fourier transform (DFT) equation into 4 subsequences (x(4n), x(4n+1), x(4n+2), x(4n+3), n = 0, 1, ... , N/4-1). The radix-4 butterfly is depicted in Fig.1(c,d). It should be noticed that $W_N^0=1$, three complex multiplications and twelve complex additions are involved for each butterfly. For radix-4, v equal to 4 , then, the number of points $N=4^v$. v stages, each of N/4 butterflies are consisted in FFT algorithm. Consequently, $3vN/4=(3N/8)\log_2 N$ complex multiplications and $(3N/2)\log_2 N$ complex additions are required. In comparison with radix-2, a 25% reduction of multiplications is achieved, but, only 50% of additions are required in radix-2 ($N\log_2 N$) vs. $(3N/2)\log_2 N$ additions in radix-4[19].

Using radix-4 algorithm has the advantage of additional savings in the required number of complex multiplications as compared to radix-2 algorithm but it needs 3N complex additions. So there is another decomposition similar to radix-4, it is radix-2^2 algorithm [20] which simplifies the complex radix-4 butterfly by reducing the number of complex additions to 2N. Radix-2^2 and radix-4 requires the same number of complex multiplications. Fig.1(e), illustrates the basic butterfly for radix-2^2 DIF FFT algorithm. In radix-2^2 algorithm, there are N/4 butterflies for each stage as radix- 4 FFT algorithm but with good regularity. So, it has the same structure of radix-2 algorithm and the same identical computational requirement of radix-4 algorithm.

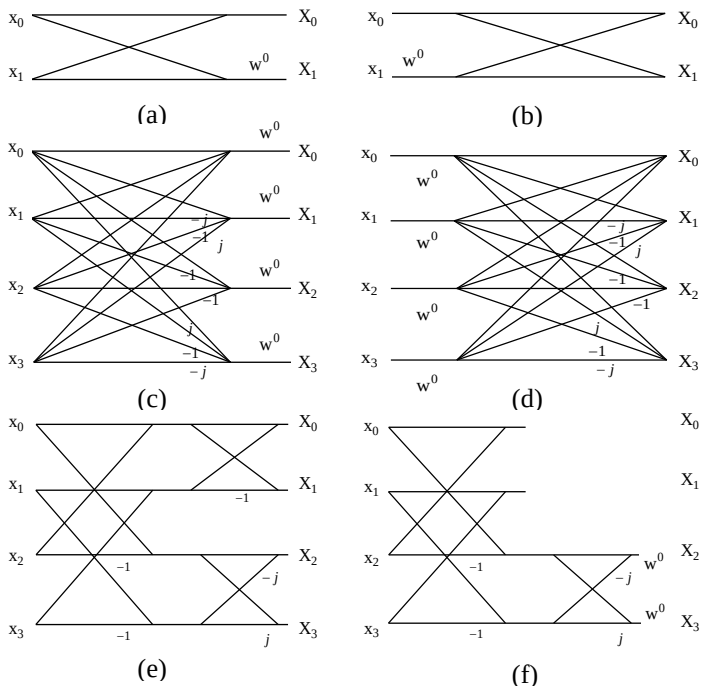

Fig. 1 Different butterfly configuration, (a) DIF radix-2 (b) DIT radix-2 (c) DIF radix-4 (d) DIT radix-4 (e) Radix2^2 (f) Split radix.

It can be observed that the even and odd-numbered points of the DFT can be computed independently, therefore the idea of using different computational methods for independent parts of radix-2 algorithm was suggested, with the objective of reducing the number of computations. The Split Radix FFT (SRFFT) algorithm was developed by Duhamel and Hollmann in 1984 [21]. It combines radix-2 and radix-4 decompositions. The SRFFT algorithm is based on decomposing an *N*-point DFT into an N/2-point DFT and two N/4-point DFTs (see Fig. 1f). The split-radix algorithm can be also derived by combining the radix-2 and radix-8 decompositions. The multiplicative complexity of the split-radix algorithm is $(N/3)\log_2 N$. It is only about two-thirds that of the radix-2 FFT, and it is better than the radix-4 FFT or any higher radix as well. Moreover, the addition complexity remains the same as radix-2 FFT about (Nlog2 N). Despite the advantage of SRFFT in that it has considerably fewer number of arithmetic computations compared to that of radix-4 and radix-2 FFT, it has irregular butterfly (L-shaped butterfly) which yields irregular hardware.

The irregular shape has uneven latencies between

data paths and is not suited for high throughput operation. Higher-radix algorithms, such as the radix-8, radix-16 require fewer computations and can produce simple but worthwhile savings (reduced memory accesses so power consumption can be reduce). However, the disadvantages are that traditional direct mapping implementation of high-radix butterfly element requires more complex operations and thus large silicon area will be consumed [22]. There isn't much difference among them, except that the series 32 division is N/8 and N/16 instead of N/4 or N/2 as in radix-2 and radix-4, the number of inputs being processed in a single butterfly (8-points and 16-points), the addressing of twiddle factors and the number of stages being $\log_8(N)$ and $\log_{16}(N)$ respectively, etc.

B. CORDIC Algorithm

CORDIC is defined as an iterative algorithm designed to calculate mathematical, hyperbolic, and trigonometric functions. It is a set of shift-add algorithms for rotating vectors in a 2D plane. It uses simple shift, add, subtract and table look-up operations to achieve the objective. The FFT complex twiddle factor multiplications can be eliminated by transforming them into CORDIC operations(Any complex multiplier based FFT architecture has its CORDIC based equivalent). This can simply be implemented. The representation of the following complex multiplication:

$$\mathrm{Re}(X)+j\,\mathrm{Im}(X)=[\mathrm{Re}(x)+j\,\mathrm{Im}(x)].e^{-j\theta} \qquad (2)$$

is represented in matrix form as follows:

$$\begin{bmatrix}\mathrm{Re}(X)\\ \mathrm{Im}(X)\end{bmatrix}=\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}\mathrm{Re}(x)\\ \mathrm{Im}(x)\end{bmatrix} \qquad (3)$$

However, Generalized equation governing CORDIC operation is given by[23]:

$$\begin{bmatrix}x_{i+1}\\ y_{i+1}\end{bmatrix}=\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}x_i\\ y_i\end{bmatrix} \qquad (4)$$

$$x_{i+1}=x_i.\cos\theta-y_i.\sin\theta$$
$$y_{i+1}=y_i.\cos\theta+x_i.\sin\theta$$

This algorithm is generalized to evaluate a set of arithmetic functions such as multiplication, division, arctangent, sine, cosine, etc. No multipliers and dividers are required. A given rotation transform equation is used in this algorithm. A new vector $V_{i+1}(x_{i+1},y_{i+1})$ is obtained by rotating a vector $Vi(x_i,y_i)$ over an arbitrary angle θ (see Fig.2).

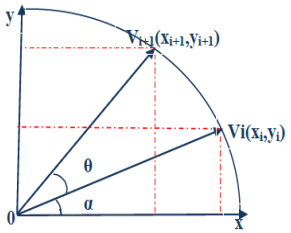

Fig. 2 Vector $V_i(x_i,y_i)$ is rotated to $V_{i+1}(x_{i+1},y_{i+1})$.

The equation above can be rearranged so that:

$$x_{i+1}=\cos\theta(x_i-y_i.\tan\theta)$$
$$y_{i+1}=\cos\theta(y_i+x_i.\tan\theta) \qquad (5)$$

If the rotation angles are restricted to $\tan\theta$, the multiplication is simply reduced to shift operation. Arbitrary angles of rotation are attainable by performing a series of successively smaller elementary rotations. If the decision at each iteration i is in which direction to rotate, then the term $\cos\theta_i$ becomes a constant (because $\cos\theta_i = \cos(-\theta_i)$). Now, the iterative rotation can be represented as:

$$x_{i+1}=K_i(x_i-d_iy_i2^{-i})$$
$$y_{i+1}=K_i(y_i+d_ix_i2^{-i}) \qquad (6)$$

And

$$K_i=\cos(\tan^{-1}2^{-i})=\frac{1}{\sqrt{1+2^{-i}}} \qquad (7)$$

rotate direction(d_i)=±1

The product of the K_i represents the K value or scaling factor. It can be removed from the equation. The value of K is treated as part of system processing gain.

$$x_{i+1}=x_i-d_iy_i2^{-i}$$
$$y_{i+1}=y_i+d_ix_i2^{-i} \qquad (8)$$
$$K=\prod_{i=0}^{w-1}K_i=\prod_{i=0}^{w-1}\frac{1}{\sqrt{1+2^{-i}}}$$

w represent number of iterations. The K represents the gain and it's approximately 1.647 if *i* goes to infinity. The angle of a combination rotation is uniquely determined by the series of directions of the elementary rotations. That series can be performed by a decision vector. The set of all possible decision vectors can be accomplished using small look-up table so the angle accumulator adds a third difference equation to the CORDIC algorithm:

$$z_{i+1} = z_i - d_i \tan^{-1}(2^{-i})$$
$$z_{i+1} = z_i - d_i \theta_i \quad (9)$$

The sum of rotating angles give the desired angle:

$$\theta = \sum d_i \tan^{-1} 2^{-i} \quad (10)$$

Radix-4 CORDIC algorithm is an extension of the radix-2 algorithm. The powers of four is used instead of powers of two so the number of iterations is reduced to half, therefore the speed of CORDIC algorithm implementation can be improved by using radix-4 CORDIC algorithm.
The iteration equations for the radix-4 CORDIC algorithm in rotation mode are derived at the (+1) th and are given by:

$$x_{i+1} = x_i - d_i y_i 4^{-i}$$
$$y_{i+1} = y_i + d_i x_i 4^{-i} \quad (11)$$
$$z_{i+1} = z_i - d_i \tan^{-1}(4^{-i})$$

where $d_i \in \{-2, -1, 0, 1, 2\}$. The final and coordinates are scaled by:

$$K = \prod_{i=0}^{w-1} K_i = \prod_{i=0}^{w-1} \frac{1}{\sqrt{1 + d_i^{\,2} 4^{-2i}}} \quad (12)$$

C. IFFT Algorithm

The IFFT of N oint sequence X(k), k=0,1,…..,N-1 is defined as:

$$X[n] = \frac{1}{N} \sum_{n=0}^{N-1} x(k) W_N^{-nk} \quad 0 \le k \le N-1 \quad (13)$$

Looking at FFT and IFFT equations (1)&(13), they look very similar but with two differences: they are divided by N and the sign of the twiddle factor. |Inverse FFT can be implemented by using the following technique which is shown in Fig.3. The inverse FFT scheme can be got without

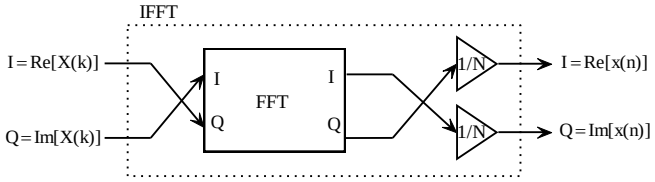

Fig. 3 IFFT circuit made by FFT module.

bothering with conjugation. Instead, swapping the real and imaginary parts of sequences of complex data will be taken.

III. THE FFT/IFFT PROCESSOR

To generate the twiddle factors in the butterfly unit, and instead of storing it in the ROM, the CORDIC algorithm has been used. Another advantage of CORDIC algorithm is to realize the butterfly operation without using any devoted multiplier hardware. Two FFT/IFFT processors are designed, the first one uses radix-2 and the other uses radix-4 structure. Two scalable, configurable data width and configurable sample points of FFT/IFFT processors are designed using Radix-2/Radix-4 and implemented in FPGA. The FFT is computed using the decimation-in frequency algorithm. The two processors are capable of performing IFFT without changing the internal coefficients because of the use of the swapping method as mentioned in last section. Input/output pins of FFT/IFFT processors are shown in Fig 4.

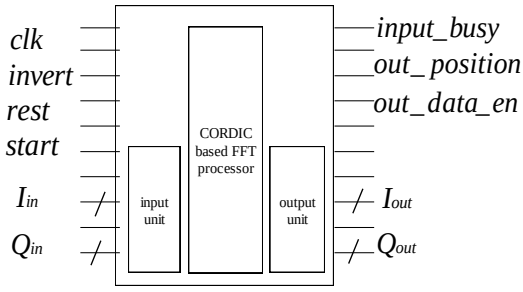

Fig 4 The CORDIC based FFT/IFFT chip.

Where: I_{in}, Q_{in} pins are used to represent the real and imaginary part of input samples respectively. The *clk* pin is the clock of processor. The *rest* pin is the reset. The *start* pin is used to trigger beginning of computation. The *invert* pin is used to assign FFT or IFFT operation. The *input_busy* pin becomes active during the computation. The

out_data_en pin becomes active at the beginning of output data. The *out_position* is used to display the output index. The I_{out} , Q_{out} pins represent the real and imaginary part of output signal respectively.

The CORDIC algorithm is used to generate the twiddle factors as shown in Fig.5. The address generation unit generate addresses to the dual ported RAM which fetch the input samples from the selector unit that is functioned as a memory buffer and used to determine the memory allocated for the written input samples. Start signal is then sent from the control unit to the radix-2/radix-4 butterfly and rotate factor generation unit to compute the 2 or 4-points FFT in radix-2/radix-4. The phase that required is generated using the rotate factor generation unit. The truncate and round unit is employed to resize the width of data. In what follows, some details are presented for the above units.

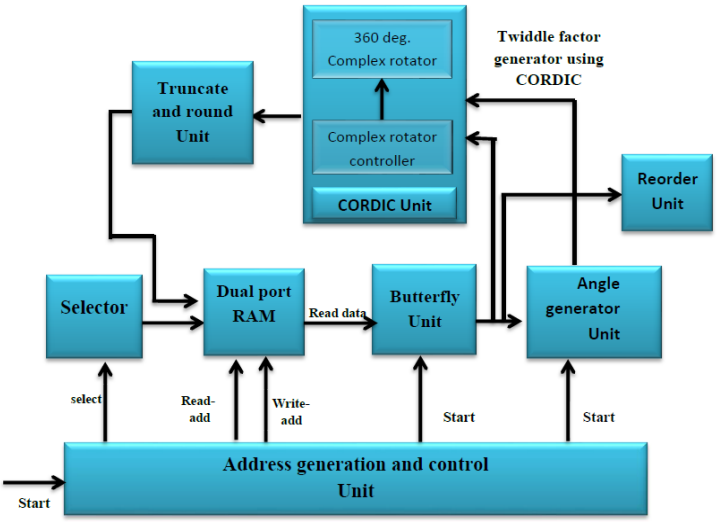

Fig 5 Architecture of CORDIC based FFT/IFFT processor

A. *Butterfly Unit*

The butterfly unit was designed to support radix-2/radix-4 DIF butterfly operation and based on CORDIC algorithm. The radix-2 butterfly unit requires only one complex addition and one complex subtraction while twelve complex addition/subtraction are required in radix-4 butterfly. The complex multipliers was eliminated by using the CORDIC algorithm.

B. *CORDIC Unit*

The CORDIC unit used to generate the twiddle factor through using it in rotation mode. It generates the cosine and sine function, which are the main components of the twiddle factor:

$$W_N^{nk} = e^{-2\pi kn/N} = [\cos(-\frac{2\pi kn}{N}) + j\sin(-\frac{2\pi kn}{N})] \quad (13)$$

Another advantage of CORDIC unit is that it eliminates complex multiplication. The expression of complex multiplication can be defined as follows:

$$\begin{aligned} &G(k).W_N^{nk} = [real(G) + j.imag(G)]. \\ &\cos(-\frac{2\pi kn}{N}) + j\sin(-\frac{2\pi kn}{N}) \\ &= real(G)\cos(-\frac{2\pi kn}{N}) - imag(G)\sin(-\frac{2\pi kn}{N}) + \\ &j[real(G)\sin(-\frac{2\pi kn}{N}) + imag(G)\cos(-\frac{2\pi kn}{N})] \end{aligned} \quad (14)$$

In return to section 2, $\theta = -2\pi kn/N$, $x = imag(G)$, $y = real(G)$, then is the real result of complex multiplication, x_{i+1} and y_{i+1} are the resulted real and image of complex multiplication respectively. There are three types of CORDIC architectures in rotation mode: Sequential / iterative, Parallel / cascaded and Pipelined. Each one of them has its own advantages and disadvantages depending upon the type of use intended. Based on the study of these advantages and disadvantages Pipelined architecture is chosen because it is comparatively the most efficient one.

Pipelined CORDIC architecture uses the basic CORDIC architecture that was described above. If a Sequential / iterative architecture was used, the generator unit would take n clock cycles to build a single output sample, where n represents the number of iterations. But using pipeline converts iterations into pipeline phases, and thus an output is obtained at every clock cycle after pipeline stages propagation delay which takes n clock cycles. It provides much faster throughput but it needs more hardware resources as compared to the Sequential module. Pipelined CORDIC contains n number of CORDIC module which is cascaded. It contains fixed shift registers at each pipelined stage and performs fixed number of shifts every time. It contains registers

at every stage to store the fixed angle for the particular micro rotation at each block in pipeline architecture. Fig.6 shows the architecture.

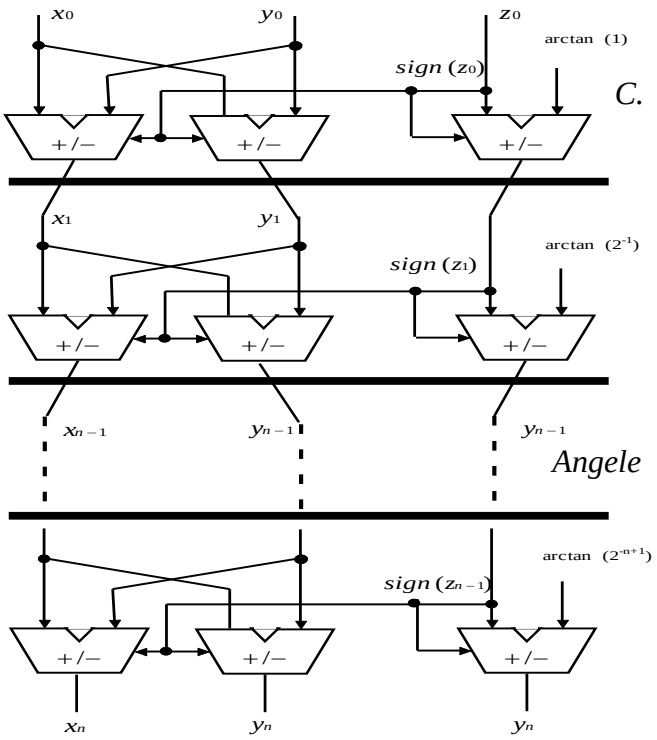

Fig 6 The pipelined CORDIC architecture.

C. Generator Unit

The Angle generator Unit generates the angle sequence that is required to generate the twiddle factor of the CORDIC unit. An N-point FFT can be divided into n = $\log_2 N$ stages in radix-2 algorithm. In this case, the twiddle factor angle sequence at the stage S ∈ {1 . . . n} has a length L = 2^s and is repeated $N/2^s$ times in order to perform the N required rotations. A counter with n bits will be used to generate the angle sequence, The most significant bit of the counter at each stage will be multiplied with the rest of counter's bits to obtain the angle sequence for that stage. During the transition from one stage to another, the number of bits for counter will be decreased by one bit in each stage, even it reaches the last stage with one bit of counter. Where no angle sequence will be generated in the last stage because the twiddle factors in last stage for DIF FFT will be zero.

In radix-4 FFT algorithm, an N-point FFT can be divided into n = $\log_4 N$ stages. In this case, the twiddle factor angle sequence at the stage S ∈ {1 . . . n} has a length L = 4^s and is repeated $N/4^s$ times in order to perform the N required rotations. The S^{th} stage has $N/4^s$ unique twiddle factor. A counter with n bits will be used to generate the angle sequence. The counter will count from 0 to $N-1$ of an N-point FFT to obtain these sequences.Fig.7 shows the mechanism that used for angle generating. In radix-4 FFT, the most significant bit of the counter and its next bit will be multiplied at each stage with the rest of counter's bits to obtain the angle sequence for that stage. Here the number of bits for counter will be decreased by two bits in each stage even reaches the last stage with two bit of counter. Where no angle sequence will be generated in the last stage because the twiddle factors in last stage for DIF FFT will be zero.

Fig.7 shows the mechanism that used for angle generating in radix-2 and radix-4.

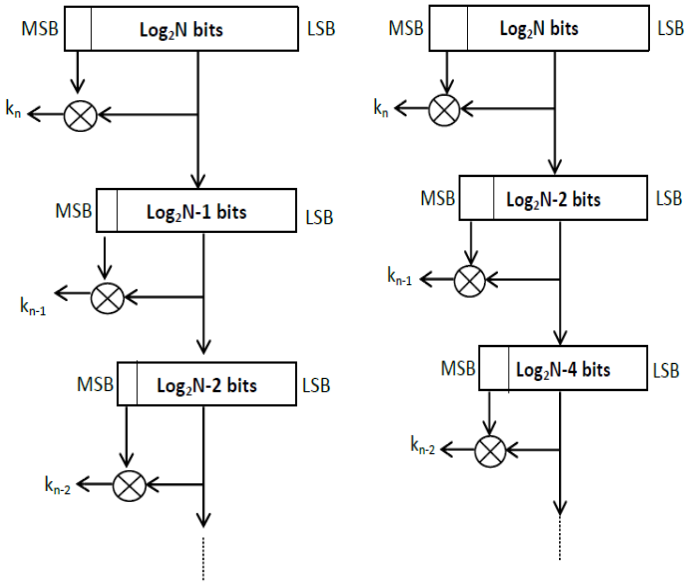

Fig 7 The mechanism of angle generation for radix-2(left) and radix-4 (right) architectures.

D. Data memory (Dual port RAM)

The input samples, intermediate samples and the output samples of FFT computation are stored in data memory. Data memory is a dual port RAM (DPRAM). One is used to hold the real components and the other to hold the imaginary components. The real and imaginary parts of the point can be read or written at the same time. The size and the data width of the dual port RAM are reconfigurable. The FFT algorithm uses “in lace”

com utation, i.e. the data after each butterfly computation is placed back in the same locations that the input data was read from, with bit-reversed addresses. So maximum memory efficiency will be achieved. This requirement has no any effect on the specification, but complicates the design of the address generator.

E. Address generation and Control Unit

The design utilizes two address generators to control the input and output of data from the dual port RAM. One is used to generate the read addresses and the other to generate the write addresses. Both create the same address pattern. The Address generation unit performs the effective address calculations (using counters) to address data operands in memory and contains the registers used to generate the addresses. It also keeps track of which butterfly is being computed in which stage. For radix-2 of an N point complex FFT, there are s stages, where $s=\log_2 N$, each stage consists of (N/2) butterflies for radix-2 structure that shown earlier. While for radix-4 there are v stages where $v=\log_4 N$. Each stage consists of (N/4) butterflies.

In the beginning of FFT computation, The write addresses are generated for the input samples in sequential (natural) order to store them in the RAM then bit reverse addressing is used for intermediate and output samples because DIF algorithm is used. Table I shows the mechanism for generating the addresses for the memory. For example in radix-2, if N=16 , N=24, then s=4, No. of counter bits (that used to generate write or read addresses) is also 4 bits. In radix-4, $v=\log_4 16$,s=2(see Table II).

TABLE I
ADDRESS GENERATION FOR RADIX-2 DIF FFT

Counter (b3b2b1b0)	Stage0 (b0b3b2b1)	Stage1 (b1b0b3b2)	Stage2 (b2b1b0b3)	Stage3 (b3b2b1b0)
0000	0000	0000	0000	0000
0001	1000	0100	0010	0001
0010	0001	1000	0100	0010
0011	1001	1100	0110	0011
0100	0010	0001	1000	0100
…..	…..	…..	…..	…..
1110	0111	1011	1101	1110
1111	1111	1111	1111	1111

TABLE II
ADDRESS GENERATION FOR RADIX-4 DIF FFT

Counter (b3b2b1b0)	Stage0 (b1b0b3b2)	Stage1 (b3b2b1b0
0000	0000	0000
0001	0100	0001
0010	1000	0010
0011	1100	0011
0100	0001	0100
…..	…..	…..
1110	1011	1110
1111	1111	1111

The simulation results of the address and control unit are shown in Fig.8.

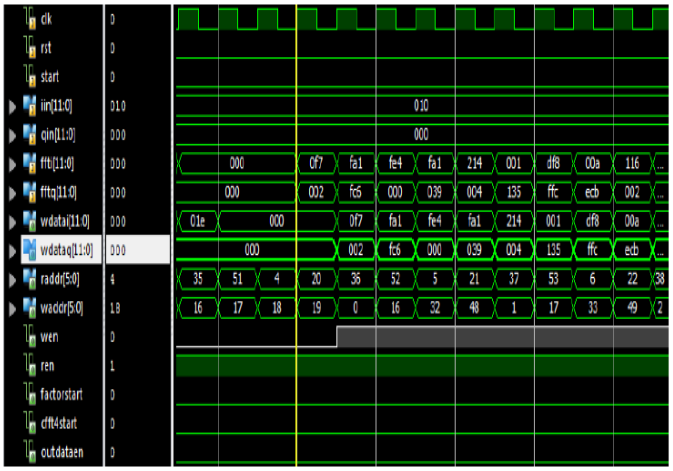

Fig 8 Address generation and control unit waveform for 64-points

Where: *Iin* and *Qin* represent the real and imaginary of input signal that come from the RAM. The RAM is used to store the input points. The *ffti* and *fftq* represent the intermediate values for real and imaginary of the point during the calculation. The *wdatai* and *wdataq* are used as registers. At the beginning, they store the real and imaginary of the input point (*Iin* and *Qin*). Then they store the intermediate result (*ffti* and *fftq*) in the same location of input point in the RAM to achieve "in place" computation. The *raddr* and *waddr* are used as counters to read and write the addresses for the memory. wen and *ren* are used to enable or disable the read and write process. The *factorstart* is used to inform the angle generator unit to start working. The *cfft4start* is used to inform the butterfly unit to start working. The *outdataen* becomes active at the beginning of output data.

F. Truncate and round Unit

The truncate and round unit is employed for resizing the word length of intermediate data that their numbers of bits are extended through the computation. The reason behind extending the word length is to get rid from the overflow issue. So, the intermediate data will be divided by two to resolve this issue in radix-2 while the intermediate data will be divided by four in radix-4 to resize it.

G. Reordering Unit

Reordering of the output samples in DIF FFT or reordering the input samples in DIT FFT is an inherent problem in FFT computation. For the proposed architecture, the outputs are obtained in the bit-reversal order. Using a memory of size N samples will solve the issue. The N samples are stored in the memory in natural order using a counter for the addresses and then they are read in bit-reverse order by reversing the bits of the counter. For example if N=16, $N=2^4$ then No. of counter bits is 4 i.e. b3b2b1b0. Reversing it will produce b0b1b2b3.

H. Implementation of IFFT architecture

The IFFT converts a spectrum (amplitude and phase of each component) into a time domain signal. A reverse implementation of butterfly diagram is done in this algorithm. Radix-2/Radix-4 Decimation-in-frequency IFFT is implemented. As mentioned earlier, we can use FFT architecture to implement IFFT with some modification, so no need to extra hardware. In FFT/IFFT processor, invert signal is responsible for the change between FFT and IFFT.

IV. EXPERIMENTAL RESULTS

The implementation of the CORDIC-based FFT/IFFT Processor on Spartan 3E-FPGA of 500.000 gates platform with operation frequency of 50MHz. The maximum operating frequency achieved after synthesis using XST version 14.2 synthesis tool is 162MHz. The design for mentioned architecture is modeled by using VHDL language through ISE12.1 program as a target technique and tested on different input signals of large data samples . Fixed point data representation of 12-bit word length is used for input data samples and 16-bit word length for twiddle factor.

The number of clock cycles, and resources usage required to calculate the FFT using both radix-2 and radix-4 are shown in Fig. 9.

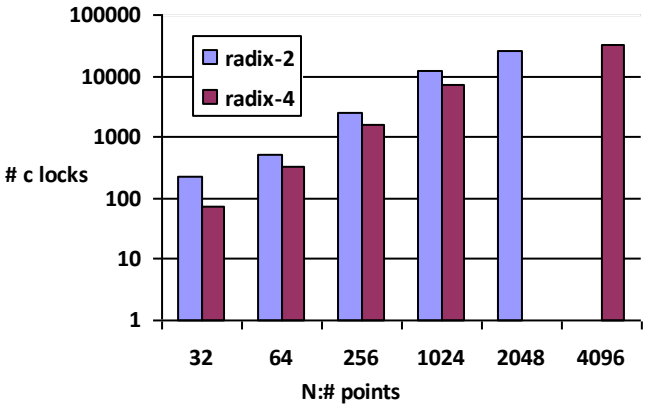

Fig 9 The radix-2 vs. radix-4 required clock cycles as a function of N.

One can see that the retired number of clock cycles are greatly reduced in radix-4 in comparison to radix-2.

Area consumption of FFT/IFFT processor based on radix-2 and radix-4 architectures (in terms of number of slices) are discussed below. Fig. 10 represent the resource usage for different FFT sizes for radix-2 based architecture and radix-4 based architecture.

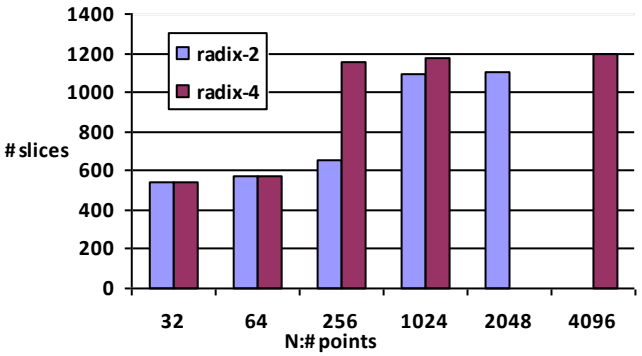

Figure 10: The radix-2 vs. radix-4 required number of slices as a function of N (The available slices of Spartan 3E are 4656).

From Fig.10, one can see that about 23% and 25% of the total available slices are only consumed in radix-2 and radix-4 respectively. However, the advantage on using radix-4 rather than radix-2 is clearly shown (the consumed

slices are approximately similar in radix-4 4096 points in comparison to radix-2 2048 point FFT).
After the synthesis process of the designs had been achieved, the Power consumption of FFT/IFFT processors was analyzed. Designing a low power FFT/IFFT processor is one of the aims in this thesis. It is determined using Xilinx Power Analyzer available in XST synthesis tool. The Xilinx Power Analyzer tool performs power analysis on the data obtained after synthesis. According to the power analysis results obtained from Xilinx Power Analyzer tool, the power consumption varies about an average value of approximately 81mW for different FFT sizes for each of radix-2 based architecture and radix-4 based architecture. The energy consumption can be calculated in micro Joules according to the equation: Energy=power×computation time . Fig.11 illustrates the total energy consumption with different FFT size for radix-2 and radix-4 architectures.

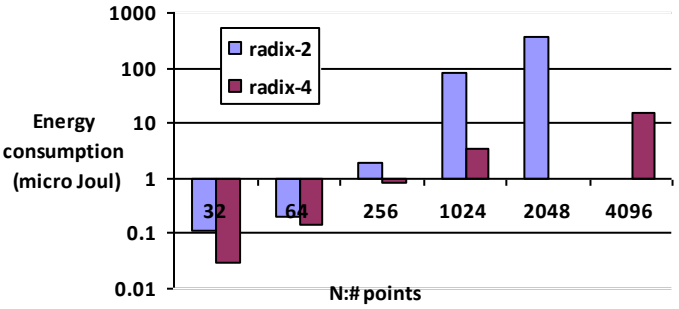

Fig 11 Energy consumption for radix-2 vs. radix-4 as a function of N.

From Fig.11, one can see that the energy consumption of radix-4 is reduced for larger FFT size in comparison to radix-2 based architecture.
The simulating and synthesizing Xilinx Logiccore FFT is performed in this paper for the comparison purpose. A comparison between our designs and Xilinx Logiccore FFT architectures is presented in term of hardware utilization, power consumption and latency. The system-level modelling tool (system generator) simplifies (FPGA) hardware design. It expands Matlab Simulink and offer an appropriated modelling environment for hardware design. Fig. 12, show the above comparisons and the advantage of the proposed processor is clearly shown.

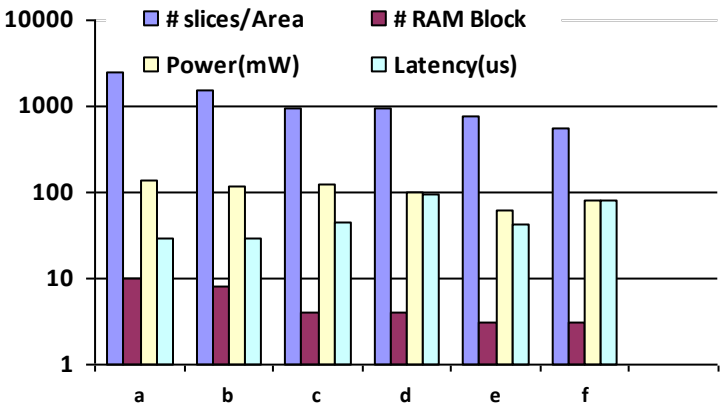

Figure 12: Comparative studies in terms of hardware utilization, power consumption and latency between CORDIC based FFT/IFFT architectures and different available system generator architectures (N= 1024, a:pipeline, streaming I/O, b:Radix-4 burst I/O, c:Radix-2 burst I/O, d:Radix-2 Lite burst I/O, e:CORDIC based radix-4, f:CORDIC based radix-2).

However, it can be noticed that the Pipelined-Streaming I/O, and Radix-4 burst I/O architectures have minimum latencies as compared to the CORDIC because they uses the pipelining technique and no computation of the twiddle factors are involved as the calculation begins.
A comparison of the proposed processors with previous research work is presented in Table III.

TABLE III
COMPARISON WITH PREVIOUS FFT PROCESSORS

	[11]	[8]	[13]	[16]	Our Design
platform	Virtix-5	Sparta n-6	Virtix-5	STRATI X-III	Sparta n-3E
N: # points	4096	1024	16	4096	4096
Data repr.	Floatin g point	Fixed point	Fixed point	Fixed point	Fixed point
f_{max}(MH z)	127	100	162	-----	162
Min. Power(m W)	-----	20	4	501	63
Scalabilit y	yes	yes	No	yes	yes
Proc. Tech.	CORD IC	Split-radix	CORD IC	CORDI C	CORD IC
# slices	-----	6041	1056	-----	1200

According to the above comparison, the proposed processor wins in term of power consumption. The computation time of this processor approximately equals the computation time of the previous research, especially for the processor which has scalability. The other feature of the proposed processor, which overcomes the previous processors, can perform the FFT and IFFT transform using the same hardware.

V. CONCLUSIONS

Two configurable CORDIC-based FFT/IFFT processors with shared memory architecture (using In-Place strategy) have been designed. The configurable data width and number of samples of radix-2 and radix-4 fixed point FFT are implemented on FPGA and modeled on VHDL language. When comparing the results that obtained, better results are achieved by using the CORDIC-based FFT/IFFT processor than that used in FFT Xilinx Logiccore architectures in terms of area and power consumption because CORDIC was used for generating the twiddle factor. So, no ROM are required to store the twiddle factors. CORDIC also eliminates the need of complex multiplication. In term of latency, a reasonable results are also achieved. However, the implementation using pipeline streaming I/O, and Radix-4 burst I/O of system generator still give lower latency than the proposed CORDIC design. According to simulation and synthesis results, The proposed architecture gives an advantage in terms of power consumption, area and resource usage as compared to literature work and it meets the timing constraints of different wireless communication standards that use OFDM modulation such as IEEE 802.11a, DAB and VDSL.

REFERENCES

[1] M. Jaing, B. Yang, and W. Fu, "Design of FFT processor with low power complex multiplier for OFDM-based high-speed wireless applications", *IEEE International Symposium on Communications and Information Technology*, 2004, pp. 639 - 641.

[2] L. Lee, and A. Girgis, , "Application of DFT and FFT algorithms to spectral analysis of power system load variation", Charlotte, NC, USA,1988, *Proceedings of the Twentieth Southeastern Symposium on System Theory*, pp. 26 – 29**.**

[3] Z. Zhou, L. Zhi , D. Yunsongand Z. Xiaoyang, "DFT-based Carrier Recovery for Satellite DVB Receivers", Las Vegas, NV, 2007, *International Conference on Consumer Electronics*.

[4] F. Wang,and X. Wang,; "Coherent Optical DFT-Spread OFDM", *Advances in Optical Technologies*, Vol(2011), No. 689289, 2011.

[5] A. Cortes, I. Velez, M. Turrillas, and J. Sevillano, "Implementing FFT and IFFT Cores for OFDM Communication Systems", *Fourier Transform - Signal Processing, InTech*, 2012.

[6] P. Mankar, L. Thakare, and A. Deshmukh ,"Design of Reconfigurable FFT for Wireless Communication Application", *IJARCSSE*, 2014, vol. 4, Issue 4, pp.319-323.

[7] A. Chavan, K. Sowmya, and S. Mishra,"VLSI Implementation of Split-radix FFT for High Speed Applications", *International Journal of Computer Applications* ,Vol.157 , No. 7, January 2017.pp.22-*26.*

[8] Z. Qian, and M. Margala, "Low-Power Split-Radix FFT Processors Using Radix-2 Butterfly Units", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol.24, Issue: 9, Sept. 2016.

[9] S. Park, and Y. Yu, "Fixed-Point Analysis and Parameter Selections of MSR-CORDIC with Applications to FFT Design*", IEEE Transactions on Signal Processing*, Vol.60, Issue: 12, Dec. 2012.

[10]M. Garrido, R. Andersson, and F. Qureshi, ,"Multiplierless Unity-Gain SDF FFTs", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 24, Issue: 9, Sept. 2016.

[11] J. Chen, Y. Lei, Y. Peng, T. He, and Z. Deng, "Configurable Floating –Point FFT Accelerator FPGA Based Multiple-Rotation CORDIC". *Chinese Journal of Electronics*, Vol. 25, Issue 6, November 2016, p. 1063 – 1070

[12]A. Malashri, and C. Paramasivam, "Low Power and Memory Efficient FFT Architecture

Using Modified CORDIC Algorithm", *International Conference on Information Communication and Embedded Systems (ICICES)*, 2013.

[13]C. Paramasivam, and K. Jayanthi, "Modified Scaling-Free CORDIC Based Pipelined Parallel MDC FFT and IFFT Architecture for Radix 2^2 Algorithm", *World Academy of Science, Engineering and Technology International Journal of Electronics and Communication Engineering*, Vol.9, No:12, 2015.

[14]J. Zhang, H. Liu,T. Chen, D. liu,and B. Zhang, "Enhanced Hardware Efficient FFT Processor based on Adaptive Recoding CORDIC", *Electronika IR Eletrotechnika*, Vol. 19, No. 9, 2013, pp. 97-103 .

[15]M. Garrido, and J. Grajal, "Efficient Memoryless Cordic for FFT Computation", *IEEE International Conference on Acoustics, Speech and Signal Processing, 2007. ICASSP* 2007.

[16]Oruklu, E.; Xiao, X.; Saniie, J.;"Reduced Memory and Low Power Architecture for CORDIC-based FFT Processors", Journal of Signal Processing System, February 2012, Volume 66, Issue 2, pp 129–134.

[17]D. Lyon, "The Discrete Fourier Transform, Part 1", *Journal Of Object Technology*, Vol. 8, No. 3, May-June 2009, pp. 17-26.

[18] W. Cooley, and W. Tukey, "An algorithm for the machine calculation of complex Fourier series", *Mathematics of Computation*,Vol. 19, No. 90 (Apr., 1965), pp. 297-301

[19]G. Proakis , "Digital signal processing" *third edition, Prenticed Hall International*, 1996.

[20]S. Sukhsawas, and K. Benkrid, "A High-level Implementation of a High Performance Pipeline FFT on Virtex-E FPGAs", *Proceedings of the IEEE Comp. Society Annual Symp. on VLSI Emerging Trends in Systems Design*, 2004, pp. 229 – 232.

[21]P. Duhamel and H. Hollmann, "S lit-radix FFT algorithm," *Electron. Lett.*, vol. 20, no. 1, Jan. 1984, pp 14-16.

[22]T. Widhe, J. Melander, and L. Wanhammar, "Design of Efficient Radix-8 Butterfly PEs for VLSI", *Circuits and Systems*, Sweden, 9 Jun 1997, pp. 2084 – 2087.

[23]J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Transactions on Electronic Computers*, Vol. 8, No. 3, pp. 330–334, 1959.