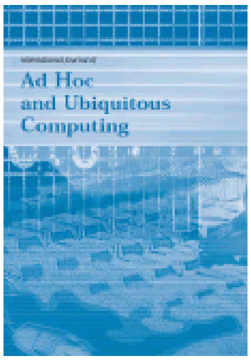


International Journal of Ad Hoc and Ubiquitous Computing

ISSN online: 1743-8233 - ISSN print: 1743-8225
https://www.inderscience.com/ijahuc

FPGA-based reduction in complexity of FFT twiddle factor butterfly with embedded CORDIC module

Priya C. Mule, Sudhakar S. Mande

DOI: 10.1504/IJAHUC.2024.10068081

Article History:

Received:	18 February 2024
Last revised:	13 June 2024
Accepted:	25 June 2024
Published online:	30 December 2024

FPGA-based reduction in complexity of FFT twiddle factor butterfly with embedded CORDIC module

Priya C. Mule*

Department of Electronics and Telecommunication Engineering,
RAIT College of Engineering,
Mumbai University,
Nerul, Navi Mumbai, India
Email: priya.mule@rait.ac.in
*Corresponding author

Sudhakar S. Mande

Department of Electronics and Telecommunication Engineering,
Donbosco Institute of Technology,
Mumbai University,
Vidyavihar, Mumbai, India
Email: ssmande@dbit.in

Abstract: In conventional FFT, techniques reducing twiddle factor complex multiplication have become a research hotspot topic. CORDIC FFT architecture overcomes the existence of multiplier blocks, which raise hardware complexity costs, increase power consumption, and lower maximum operating clock frequency. An iterative-based CORDIC architecture with a compensated barrel shifting network to provide gain k = 0.6 is proposed. This algorithm substitutes sine and cosine convolution factors with repeated KORDIC convolutions, allowing for reduced ROM. A high-speed FFT processor comprising of twiddle factor W_N is replaced with an embedded CORDIC module on FPGA. It is observed that CORDIC FFT eliminates the use of 20 DSP components thereby reducing computational complexity at the maximum achievable speed of 75 MHZ. Output is shown successfully on the Artix7 board with a minimal error of approx. 0.25% and an accuracy of 99.75%. Latency and throughput issues are improved in this. Static power in CORDIC FFT is approximately 114 mw.

Keywords: CORDIC; embedded; twiddle factor; computation; complexity.

Reference to this paper should be made as follows: Mule, P.C. and Mande, S.S. (2025) ‘FPGA-based reduction in complexity of FFT twiddle factor butterfly with embedded CORDIC module’, *Int. J. Ad Hoc and Ubiquitous Computing*, Vol. 48, No. 1, pp.34–45.

Biographical notes: Priya C. Mule is currently working as an Assistant Professor at the Ramrao Adik Institute of Technology, Navi Mumbai. She is pursuing her PhD from the Mumbai University. She obtained her ME in electronics from RAIT, Mumbai University. Her areas of interest include VLSI signal processing and embedded systems respectively. She has published several research papers in international conferences. She has 19 years of experience in all.

Sudhakar S. Mande is currently working as the Principal at the Don Bosco Institute of Technology, Mumbai. He obtained his MTech and PhD from the IIT Bombay in 2000 and 2011 respectively. His research areas include nanoscale device and circuit design, embedded system design, low-power VLSI design and VLSI signal processing. He has published several research papers in peer-reviewed international journals and in international conferences. He has more than 30 years of experience.

1 Introduction

Rotation algorithm is generally used for generating sine and cosine waves. The rotary CORDIC algorithm is a powerful and widely used tool for DSP applications. So the role of rotation CORDIC in VLSI is essential based on the overview provided by Hsiao (2010). The beauty of rotation CORDIC lies in the fact by simply moving and adding functionality, it can compute trigonometric, complex multiplications. The architectural CORDIC module improvement eliminates the need for embedded dedicated functional blocks (DSP, BRAM) (Zhao and Lv, 2022). To achieve high-performance computation and scientific calculus, a floating-point sine/cosine function with a minimal error is essential. According to Zhu and Lei (2017), in Intel 8087, Motorola, Cyrix coprocessors, and processors such as 486DX and Pentium, sine/cosine computation takes place in hardware. FPGA and ASIC provide accurate platforms for the completion of the addition task of sine/cosine variables. Thus, the required count of the number of multiplications and additions is more. A polynomial is usually merged with a lookup table. A look-up table compresses the variable's range, and the polynomials determine the End result instantly. as explained by Orian and Yahav (2023). However, implementing the cost of this combination of multipliers, adders and arrays for high-precision sine/cosine calculations is high. Moreover, according to Manasa and Noorbasha (2017) the commonly used software solutions for the digital implementation of these functions are digit-by-digit methods referred to as frequent pseudo division and frequent pseudo multiplication processes Meggit (1962), which resemble repeated-addition multiplication and repeated subtraction division. In the case of conventional radix-2 FFT architectures computation takes place with the support of a butterfly unit performing complex addition and complex subtraction process. Twiddle factors are an integral part of fast Fourier transform (FFT) processes. Traditionally, they are either calculated at runtime, which makes the calculation more complex, or they are pre-calculated and stored in ROM as shown by Arunkumar and Kirthika (2013), which requires a lot of memory space and increases power consumption. Again time is also required more for this operation. In the FPGA design of a CORDIC architecture for biomedical signal processing the structure of a CORDIC-based FFT processor testing on FPGA is very much more preferred than ASIC due to the low prototype investment as suggested by Banerjee and Dhar (2001). Among other benefits are the less duration design cycle and the chance of re-programmability and testing for improvement with zero cost. In traditional CORDIC architecture, $\delta i \in (-1, 1)$. It means that all the rotations are accomplished, which leads to a constant gain of the rotator. The iteration length, gain factor, and the pipelined computing of the micro-rotations are the suggested areas of the CORDIC algorithm efficiency, and these modifications are relevant in real-time applications as per Changela and Zaveri (2023). Another utilisation of the CORDIC algorithm and buttery operation without a multiplier is seen in the communication field work by Peng and Li (2011), with many features such as smaller chip area, more precision, and accuracy, reduced power consumption, and high SNR. An FFT processor with high-performance SNR can be designed. In the proposed eight-point radix-2 FFT processor involving an embedded CORDIC module, the problem of complex multiplication and the space required by the ROM has been resolved. Also, the time required to complete the task is less. The rotational CORDIC algorithm is a type of digital iterative method that can calculate various transcendental functions, including the sine/cosine function in circular coordinates and rotational mode. Discrete sinusoidal transformations and adaptive filters in DSP are two quick and real-time applications that have made use of the sine/cosine convolution factor computation in FFT to reduce complexity. But in signal processing, there are a lot of iterations that must be done in that order., which is the bottleneck of rotation CORDIC optimisation. Rotational cordial accuracy approaches linearly with the number of repetitions. Due to the simplicity of CORDIC operations, it is suitable for VLSI design according to Sapper mand Paim (2021).

The rest of the paper is organised as follows. Section 2 presents the background of the FFT algorithm in conventional radix-2, eight-point DITFFT and the traditional CORDIC architecture. Section 3 gives details of the proposed FFT architecture involving the CORDIC module. Section 4 demonstrates the performance analysis of the CORDIC algorithm and CORDIC FFT. Section 5 displays the simulation-based CORDIC FFT results and simplified results of the CORDIC unit and butterfly unit. In Section 6 Finally, the results are summarised, and compared. Section 7 presents the conclusions.

2 Background

The design of the FFT processor and the degree of implementation difficulty are significantly influenced by the algorithm's radix as discussed by Mehrotra and Pandey (2017). A small radix results in a simple butterfly. On the other hand, a high radix reduces the number of twiddle factor multiplications. The radix r^k algorithms simultaneously achieve a simple butterfly and a reduced number of twiddle factor multiplications. The radix-2 algorithm is a well-known simple algorithm for FFT processors, but it requires many complex multipliers.

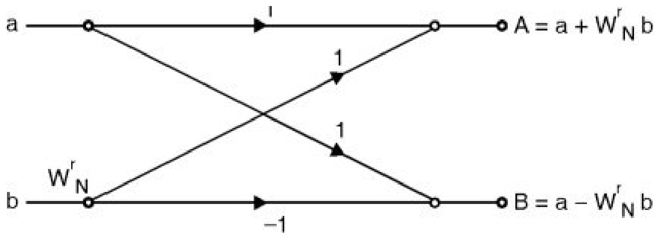
2.1 Radix-2 FFT algorithm

The decimation in time (DIT) radix-2 FFT algorithm is a widely used method for computing the Fourier transform (DFT) of a time-domain sequence discretely. Initially, the time-domain sequence $x(n)$ is 'decimated' into smaller two-point sequences in this approach. The efficiency of the radix-2 FFT algorithm, which is the subject of Enis and Marsida (2015), is attributed to its ability to reduce the no of mathematical operations needed to compute the DFT. It has a time complexity of $O(N \log 2N)$, making it faster than the straightforward $O(N^2)$ method of calculating the DFT. A basic radix-2 butterfly diagram of calculation elements is shown in Figure 1. Single Butterfly is made up of one complex adder and one complex subtraction unit with the same computational complexity.

2.2 Radix-2 FFT algorithm implementation

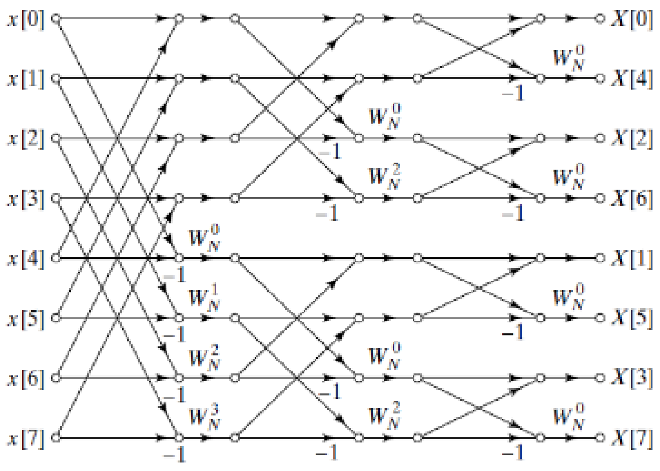
The basic implementation follows the core idea of executing the SFG of the eight-point radix-2 FFT as shown in Figure 2 This implementation depicts that the output has to be in normal sequence and input should be ordered in a bit reversed manner. In DFT for $N = 8$, no. of complex additions $N(N-1)$ and complex multiplications N^2 are 56 and 64 respectively as per Pal and Bhattacharjee (2020). In radix-2 FFT for $N = 8$, no. of complex additions $N1og2N$ and complex multiplications $(N/2)1og2N$ are 24 and 12 respectively.

Figure 1 A basic butterfly structure



Source: Enis and Marsida (2015)

Figure 2 Signal flow graph for eight-point radix-2 DITFFT



Source: Enis and Marsida (2015)

From Figure 2, we can see that FFT consists of 12 butterfly units, and it becomes more difficult for computation. Butterfly units allow the usage of multipliers, addition, and subtraction units for calculating outputs at different stages and this needs more area, power, and high cost. To avoid the limitation of the butterfly unit and to speed up the performance, the use of CORDIC in the butterfly unit is an unavoidable demand of today.

2.3 *Traditional CORDIC architecture*

To ensure high performance, limit hardware complexity, and minimise implementation latency, the CORDIC module is developed based on the CORDIC algorithm.

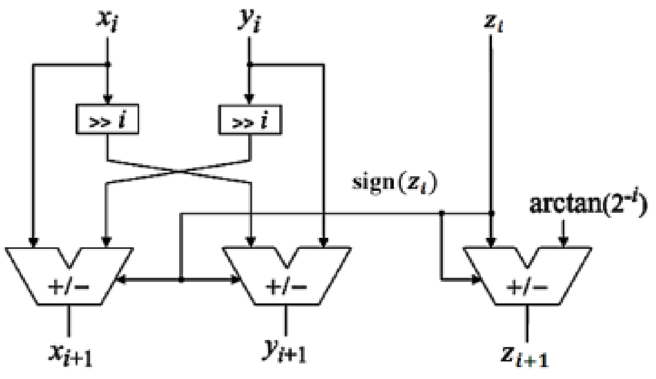
- For high-performance computing: Parallel and pipeline CORDIC is proposed.
- To limit the complexity of the application: In the CORDIC engine, the scaling complexity and barrel shifter implementation must be reduced.
- To minimise implementation latency: Angular transcoding systems, mixed grain rotation, and higher base number CORDIC techniques are developed.

2.4 *CORDIC techniques*

2.4.1 *Iterative decomposition of the rotation angle*

The CORDIC algorithm according to Meher and Valls (2009) does the rotation repeatedly as discussed by Bhukya and Inguva (2021) dividing the rotation angle into a set with small predetermined angles as shown in Figure 3 so that $\tan \alpha i = 2^i$ can be realised in hardware by moving through the i-bit positions.

Figure 3 Iterative decomposition of angle of rotation



2.4.2 *Avoidance of scaling*

Basic KORDIC iterations are obtained by applying vector pseudorotation with the irreversible decomposition of selected angles αi as follows as discussed by Meher and Valls (2009).

$$x[i+1] = x[i] – \sigma i 2^{-i}.y[i] \tag{1}$$

$$y[i+1] = y[i] + \sigma i 2^{-i}.x[i] \tag{2}$$

$$z[i+1] = z[i] – \sigma i \tan^{-1}.2^{-i} \tag{3}$$

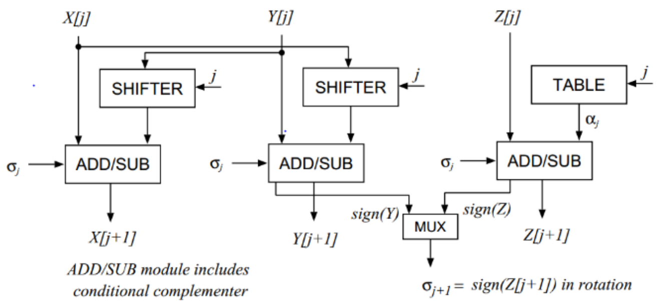
CORDIC iterations are mainly carried out in two active modes, the rotary mode (RM) and vector mode (VM) defined by Volder (2000). In rotation mode, a new vector pn is created by rotating the vector p by an angle θ. In this mode, the sign of $z[i]$ determines the direction of each microrotation σi. If $z[i]$ has a positive sign, then $\sigma i = 1$; if not, $\sigma i = -1$. To make the y-component closer to zero in vector form, the vector p is rotated in the direction of the x-axis. The rotation angle of the vector p is equal to the total of all the microrotation angles (output angle zn), and its magnitude is equal to the output. Figure 4 illustrates the specific hardware implementation method for CORDIC arithmetic. It contains two shifters to supply the terms $2^{-i}.x$ and $2^{-i}.y$ into adder/subtractor units., three registers (x, y, and z), and a lookup table to hold the values $\alpha.i = \tan^{-1}.2^{-i}$. These αi are stored as LUT angles in the CORDIC hardware ROM. The factor of Di (–1 and 1) is accumulated by choosing the (shift) operand or its complement. We have $\sum \alpha i = z$, and the KORDIC expression forms. The constant K in the following equation is k = 0.6073 (Manasa and Noorbasha, 2017).

$$\begin{aligned} k &= \sum_{i=0}^{7} \cos \phi_i \\ &= \cos \phi_0 . \cos \phi_1 . \cos \phi_2 ... \cos \phi_6 . \cos \phi_7 \end{aligned} \tag{4}$$

$$\begin{aligned} &\cos 45^\circ . \cos 26.565^\circ . \cos 14.036^\circ ... \cos 0.4469^\circ \\ &= 0.6073 \end{aligned} \tag{5}$$

Figure 4 shows how the CORDIC algorithm works for a single iteration.

Figure 4 Implementation of equations (1), (2) and (3) for one iteration



CORDIC is generally faster as compared to other alternatives when there is no need for a hardware multiplier in the case of a Microcontroller or when the logic function is to be implemented with a limited no of gates as in the case of FPGA. On the other hand in a DSP microprocessor where usage of the multiplier cannot be dropped, to increase speed table lookup methods discussed by Tang (1991) and power series approach is more suitable than CORDIC. Digital signal processing is needed more as technology spreads. Today, we want to lower cost, area, electricity, and speed prompting the creation of more advanced DSP algorithms to improve performance. DSP's most powerful tool is DFT. DFT uses arithmetic to break a sequence which is slow and difficult to compute. Two methods are available to calculate the FFT, where the input sequence is divided into even and odd periods, called frequency decimation. In the FFT processor, the main focus is on the butterfly unit. FFT requires $N. \log_2 .N$ calculations. Using CORDIC instead of a twiddle multiplier can reduce complex multiplication (Choudhary and Karmakar, 2011). It can also solve the problem of the ROM required and the time required to complete the function.

2.4.3 Concept of CORDIC gain

For eight-bit CORDIC hardware, Table 1 depicts the CORDIC gain.

If 'L' denotes the power of base 2 such as 0, 1, 2, etc. K will thereafter be reduced by two powers, starting with 2^0 = 1.0. The CORDIC algorithm does not need any multiplies due to the use of powers of two for the K values, simple shifting and adding of binary numbers is possible. For example: in the phase column for ideal values of K, if phases are (half the previous values) such as 45°, 22.5°, 11.25° then $K \neq 2^{-L}$. Values of K will be 1.0, 0.414, 0.199, etc. Multiply using just shift/add's not possible (which would eliminate the major benefit of the algorithm). The difference in accuracy between the ideal K's and this binary K's is generally negligible so we use binary Ks. Accuracy can be improved by increasing the no of iterations. In the rotation process to get the true value of magnitude we multiply it with the reciprocal of cumulative CORDIC gain of value 1.647 which comes to 0.6073. The value of this magnitude factor K is a function of n iterations which can be a constant for any given computer for the factor n = 24 (Volder, 1959). To achieve this I have used the following compensated barrel-shifting network in my proposed architecture to provide a gain of 0.60 with precision up to 0.44690 for 8-bit CORDIC hardware as mentioned by Chaitra and Sheshadri (2018).

Table 1 CORDIC gain table

L	$K = 2^{-L}$	$R = 1 + jK$	*Phase of R in degrees = atan (K)*	*CORDIC gain (K)*
0	1.0	$1 + j1.0$	45	1.414
1	0.5	$1 + j0.5$	26.565	1.5811
2	0.25	$1 + j0.25$	14.036	1.6298
3	0.125	$1 + j0.125$	7.125	1.6424
4	0.0625	$1 + j0.0625$	3.576	1.6456
5	0.03125	$1 + j0.03125$	1.7876	1.6464
6	0.015625	$1 + j0.0156$	0.8938	1.6467
7	0.0078125	$1 + j0.00781$	0.4469	1.6467

3 Proposed CORDIC module for FFT

Figure 5 depicts the respective CORDIC module consisting of the Arithmetic shifter network where the inputs Xo, Yo are initially applied to the shifters 1, 4, 5, 6 and added together to provide respective scaled inputs xin, yin with constant CORDIC gain K = 0.60. In the CORDIC rotation module initially, zangle is set for reference. Then scaled inputs xin and yin are passed on to Pseudo rotation for 16 iterations with small predefined angles. LUT stores all predefined angles depicted by Manasa and Noorbasha (2017) required for each iteration to work out and at the end of the 16th stage, we get $xout = x_{15}$ and $yout = y_{15}$.

3.1 Working of the proposed CORDIC module

Initially Xo = 32,000 and Yo = 0. To achieve constant gain K = 0.60, a barrel shifting network is used where

$$\begin{aligned} &xin = Xo * (2^{-1} + 2^{-4} + 2^{-5} + 2^{-6}) \\ &xin = 32,000 * (0.5 + 0.0625 + 0.03125 + 0.01562 \\ &xin = 32,000 * 0.6094 = 19,500 \\ &yin = 0 \text{ as } Yo = 0. \end{aligned}$$

Figure 5 Proposed CORDIC module (see online version for colours)

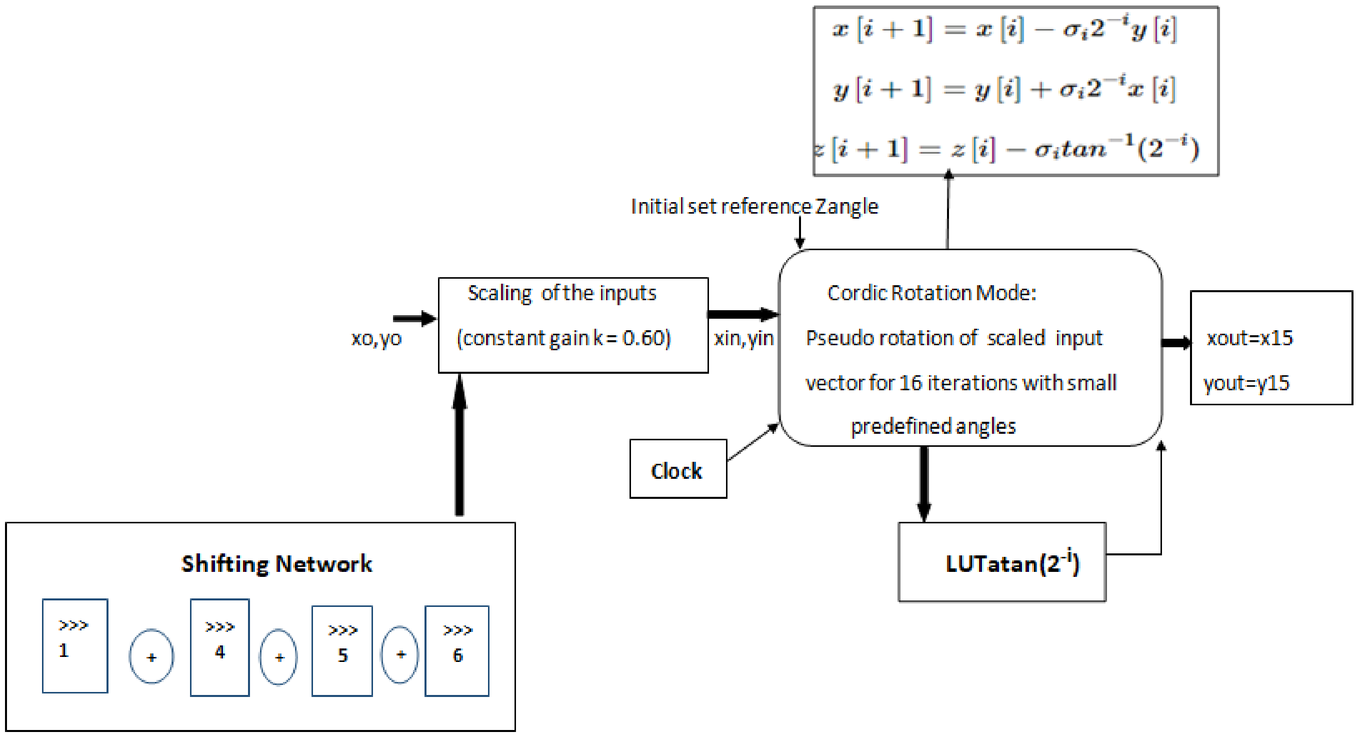

$x[i+1] = x[i] - \sigma_i 2^{-i} y[i]$
$y[i+1] = y[i] + \sigma_i 2^{-i} x[i]$
$z[i+1] = z[i] - \sigma_i tan^{-1}(2^{-i})$
Initial set reference Zangle
xo,yo
Scaling of the inputs
(constant gain k = 0.60)
xin,yin
Cordic Rotation Mode:
Pseudo rotation of scaled input vector for 16 iterations with small predefined angles
xout=x15
yout=y15
Clock
LUTatan(2^{-i})
Shifting Network
>>> 1
+
>>> 4
+
>>> 5
+
>>> 6

Table 2 Precomputed CORDIC angles

Sr. no.	$tan(\phi i) = 2^{-i}$	$\phi i = tan^{-1}(2^{-i})$	ϕi *in radians*
1	1	45	0.7854
2	0.5	26.565	0.4636
3	0.25	14.036	0.2450
4	0.125	7.125	0.1244
5	0.0625	3.576	0.0624
6	0.03125	1.7876	0.0312
7	0.015625	0.8938	0.0156
8	0.0078125	0.4469	0.0078
9	0.00390625	0.2238	0.0039
10	0.001953125	0.1119	0.0019
11	0.0009765625	0.05595	0.00098
12	0.000488281	0.02798	0.0005
13	0.000244141	0.01398	0.000244
14	0.00012207	0.006994	0.000122
15	6.10352E-05	0.003497	0.000061
16	3.05176E-05	0.001748	0.000031

One iteration is carried out in one clock pulse. Like wise 16 iterations are considered. Initially set reference Zangle as 165°. To bring the angle in the first quadrant, the reference angle is subtracted by 90°. The reflected angle is (165 – 90 = 75°). Then according to the CORDIC equations (1), (2) and (3), the computation of the CORDIC algorithm works as depicted in Table 3.

CORDIC angles required to replace the twiddle factor in the proposed FFT are shown in Figure 6. In CORDIC-based FFT processor CORDIC replaces twiddle factor to solve the complexity of multiplication. In this paper, implementation of radix-2, eight-point FFT involving an embedded CORDIC unit is undertaken. The twiddle factor (Enis and Marsida, 2015) for radix-2 is as follows

$$W_N^2 = e^{-j2\pi 2/(N)} = e^{-j2\pi/(N/2)} \tag{6}$$

Based on the above CORDIC angles, CORDIC-based FFT is designed which make use of 12 butterfly units and five CORDIC modules replacing complex adders and complex multipliers.

Figure 6 Radix-2, eight-point FFT CORDIC angles (see online version for colours)

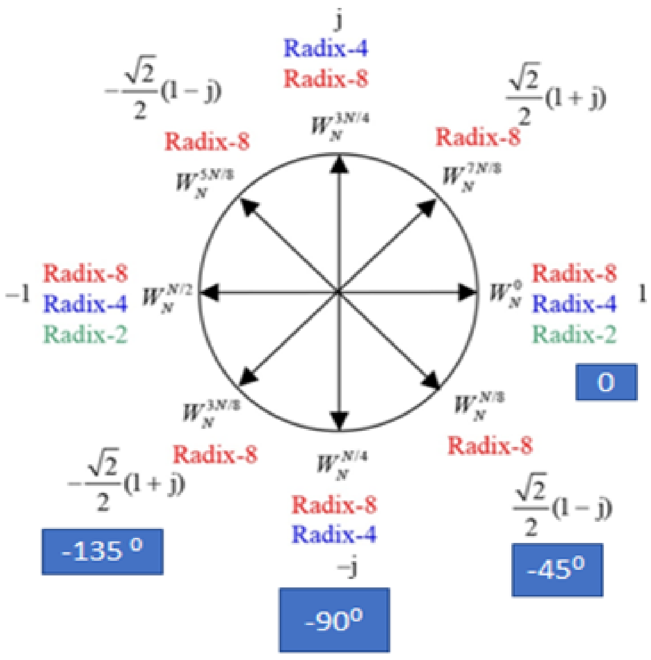


Figure 7 Radix-2, eight-point CORDIC FFT (see online version for colours)

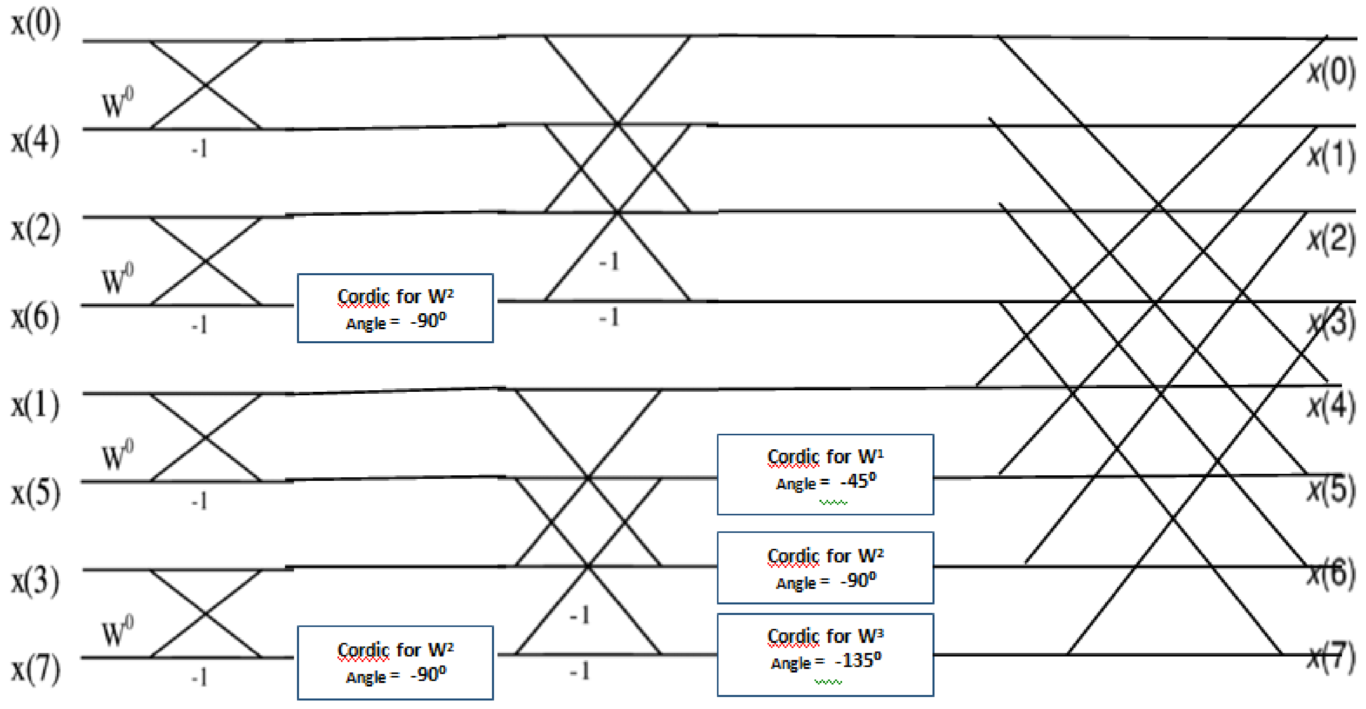

x(0)
x(4)
x(2)
x(6)
x(1)
x(5)
x(3)
x(7)
W0
W0
W0
W0
-1
-1
-1
-1
-1
-1
-1
-1
Cordic for W2 Angle = -90°
Cordic for W2 Angle = -90°
Cordic for W1 Angle = -45°
Cordic for W2 Angle = -90°
Cordic for W3 Angle = -135°
x(0)
x(1)
x(2)
x(3)
x(4)
x(5)
x(6)
x(7)

Figure 7 depicts the proposed FFT architecture using CORDIC. With reference to the signal flow graph in Figure 2, two CORDIC modules are required for twiddle factors $W_4^1[-90°]$ and three are utilised for $W_8^1[-45°]$, $W_8^2[-90°]$, $W_8^3[-135°]$ respectively. For W_2^0, W_4^0, W_8^0 there is no effect of rotation, So it is not replaced by the CORDIC algorithm.

Table 3 Computation of rotation CORDIC

Sr. no.	*σi*	*X[i]*	*Y[i]*	*Z[i]°*	*LUT angles*
–	–	19,500	0	165 – 90 = 75	45
0	1	0	19,500	75 – 45 = 30	26.57
1	1	–19,500	19,500	30 – 26.57 = 3.435	14.03
2	1	–29,250	9,750	3.435 – 14.03 = –10.63	7.13
3	–1	–31,687	2,437	–10.63 + 7.13 = –3.5	3.6
4	–1	–31,383	6,398	–3.5 + 3.6 = 0.1	1.79
5	1	–30,984	8,360	0.1 – 1.79 = –1.690	0.895
6	–1	–31,245	7,391	–1.690 + 0.895 = –0.795	0.448
7	–1	–31,130	7,880	–0.795 + 0.448 = –0.347	0.224
8	–1	–31,069	8,124	–0.347 + 0.224 = –0.123	0.112
9	–1	–31,038	8,246	–0.123 + 0.112 = –0.0114	0.056
10	–1	–31,022	8,307	–0.0114 + 0.056 = 0.04460	0.0280
11	1	–31,014	8,338	0.4460 – 0.0280 = 0.0166	0.0140
12	1	–31,018	8,322	0.0166 – 0.0140 = 0.00264	0.00700
13	1	–31,020	8,314	0.00264 – 0.0070 = –0.0044	0.0035
14	–1	–31,021	8,310	–0.0044 + 0.0035 = –0.0009	0.00175
15	–1	–31,021	8,812		

4 Performance analysis

4.1 CORDIC module

In this, we apply the CORDIC algorithm to calculate the sine and cosine of 165° with n = 16 iterations. We choose Xo = 32,000 and Yo = 0. Initially, Xo and Yo are scaled down to achieve CORDIC gain K. The changed inputs are xin = 19,500 and yin = 0 respectively. Table 3 shows how the successive values are calculated from the expressions (1), (2) and (3) to achieve output $xout[15]$ and $yout[15]$ respectively. Initially reference angle is brought to the first quadrant by subtracting 90°. We now get the set angle $Z[i]$ as 75°. Table 3 shows the computation of rotation CORDIC.

From the specified table, we get $xout[15]$ = –31,021 and $yout[15]$ = 8,812 respectively.

4.2 Radix-2, eight-point CORDIC FFT processor

For radix-2, eight-point FFT processor consider Table 4 consisting of inputs as well as stage 1, stage 2, and final outputs. For example, $x0$ = 3,200, $x1$ = 1,200, $x2$ = 0, $x3$ = 0, $x4$ = 0, $x5$ = 0, $x6$ = 0, $x7$ = 0. Similarly $y0$ = 0, $y1$ = 0, $y2$ = 0, $y3$ = 0, $y4$ = 0, $y5$ = 0, $y6$ = 0, $y7$ = 0. Table 4 signifies that inputs are in bit reversed order and outputs are in normal sequence. Sample working of final output $X(0)$ and $X(1)$ from stage 2 values considering butterfly computation is shown below.

$$X(0) = 3,200 + 1,200 = 4,400(1130\ in\ HEX)$$
$$X(1) = 3,200 - 1,200 = 2,000(07d0\ in\ HEX)$$

Figure 8 Schematic diagram of CORDIC FFT for eight-point DITFFT (see online version for colours)

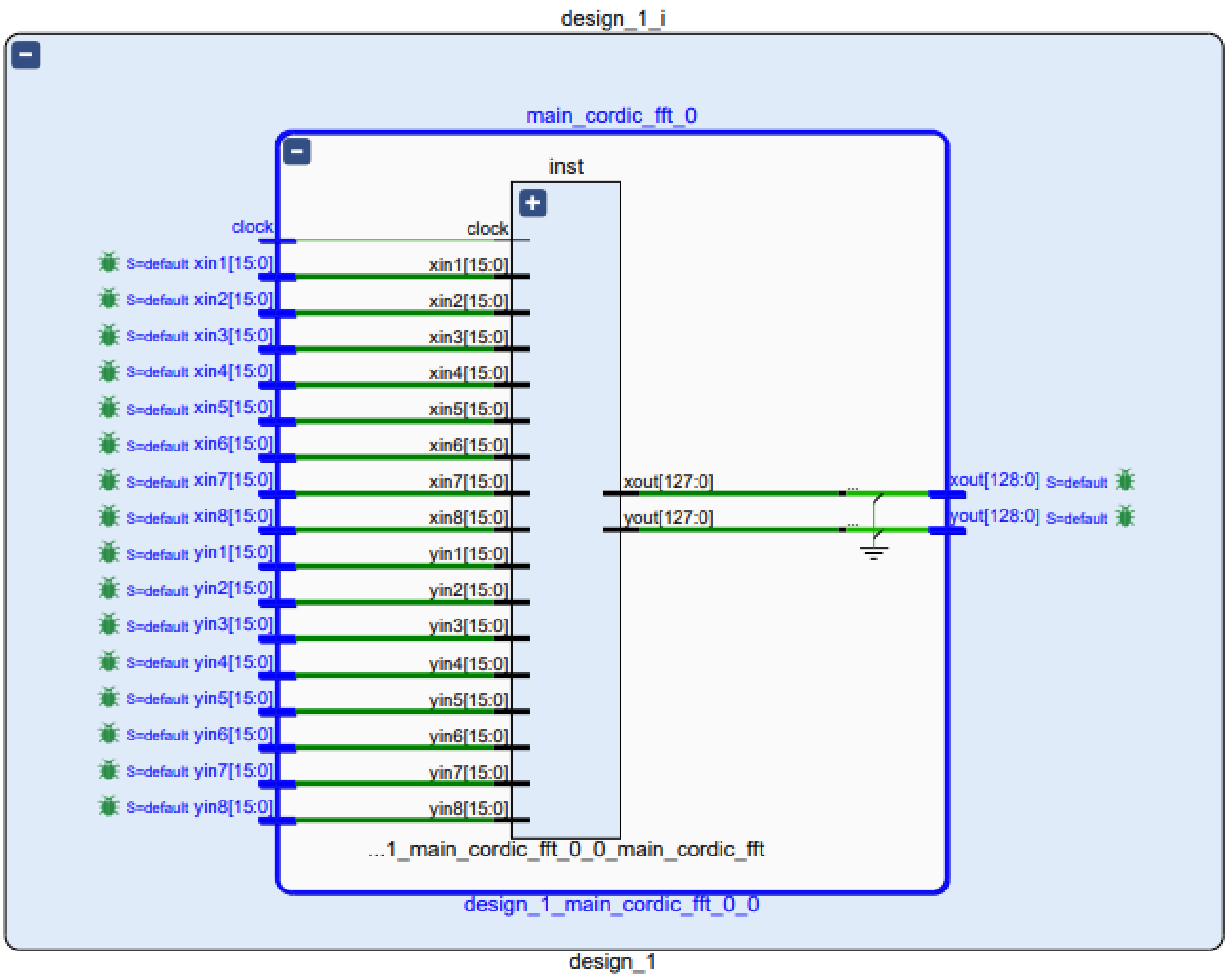
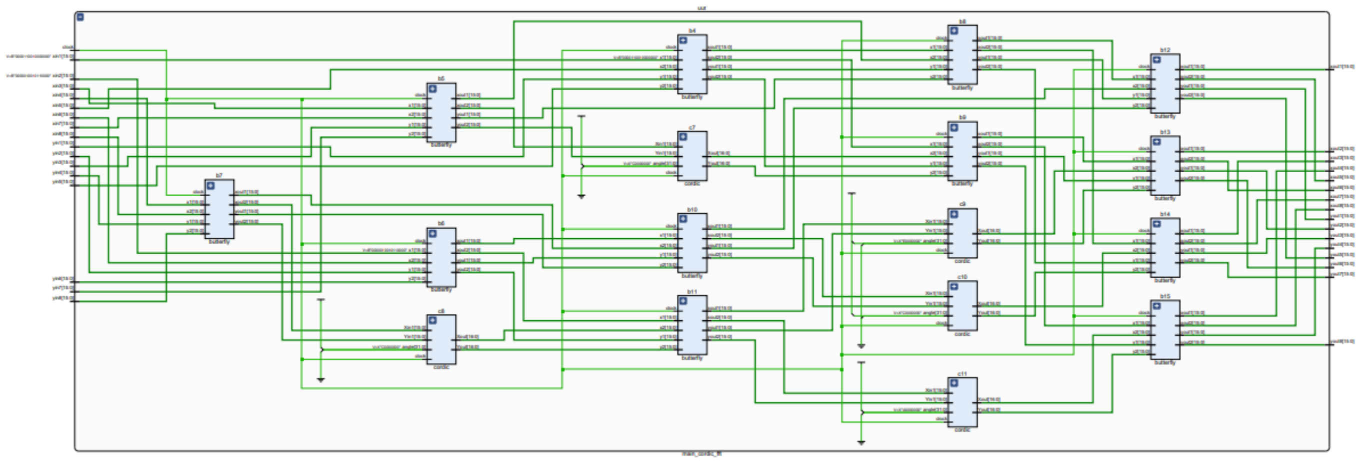


Figure 9 RTL schematic of CORDIC FFT for eight-point DITFFT radix-2 butterfly using Xilinx 14.7 ISE (see online version for colours)



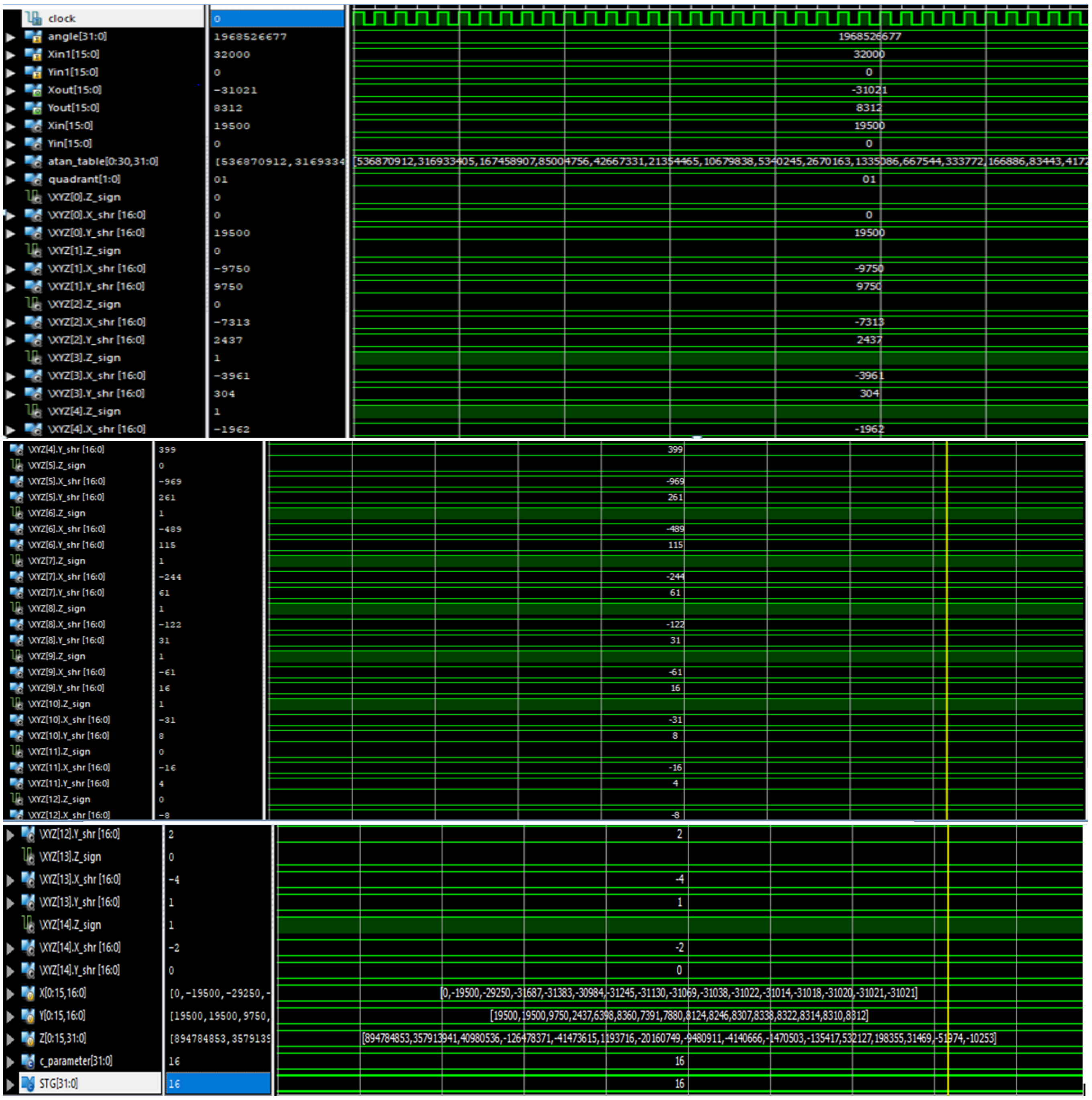
5 Simulation results

From Figure 8, we can see 12 instances of the butterfly module as per the DITFFT algorithm and five instances of the CORDIC module for multiplication of butterfly results with twiddle factor.

Angle $Z[i]$ and LUT angles with signed decimal representation in simplified results of the CORDIC algorithm are displayed in Table 5

$$degree\ angle = (bin\ to\ dec) * 360/2^{32} \tag{7}$$

Figure 10 Simplified simulation results of CORDIC algorithm (see online version for colours)



```
clock                  0
angle[31:0]            1968526677              1968526677
Xin1[15:0]             32000                   32000
Yin1[15:0]             0                       0
Xout[15:0]             -31021                  -31021
Yout[15:0]             8312                    8312
Xin[15:0]              19500                   19500
Yin[15:0]              0                       0
atan_table[0:30,31:0]  [536870912,3169334     [536870912,316933405,167458907,85004756,42667331,21354465,10679838,5340245,2670163,1335086,667544,333772,166886,83443,4172
quadrant[1:0]          01                      01
\XYZ[0].Z_sign         0
\XYZ[0].X_shr [16:0]   0                       0
\XYZ[0].Y_shr [16:0]   19500                   19500
\XYZ[1].Z_sign         0
\XYZ[1].X_shr [16:0]   -9750                   -9750
\XYZ[1].Y_shr [16:0]   9750                    9750
\XYZ[2].Z_sign         0
\XYZ[2].X_shr [16:0]   -7313                   -7313
\XYZ[2].Y_shr [16:0]   2437                    2437
\XYZ[3].Z_sign         1
\XYZ[3].X_shr [16:0]   -3961                   -3961
\XYZ[3].Y_shr [16:0]   304                     304
\XYZ[4].Z_sign         1
\XYZ[4].X_shr [16:0]   -1962                   -1962
\XYZ[4].Y_shr [16:0]   399                     399
\XYZ[5].Z_sign         0
\XYZ[5].X_shr [16:0]   -969                    -969
\XYZ[5].Y_shr [16:0]   261                     261
\XYZ[6].Z_sign         1
\XYZ[6].X_shr [16:0]   -489                    -489
\XYZ[6].Y_shr [16:0]   115                     115
\XYZ[7].Z_sign         1
\XYZ[7].X_shr [16:0]   -244                    -244
\XYZ[7].Y_shr [16:0]   61                      61
\XYZ[8].Z_sign         1
\XYZ[8].X_shr [16:0]   -122                    -122
\XYZ[8].Y_shr [16:0]   31                      31
\XYZ[9].Z_sign         1
\XYZ[9].X_shr [16:0]   -61                     -61
\XYZ[9].Y_shr [16:0]   16                      16
\XYZ[10].Z_sign        1
\XYZ[10].X_shr [16:0]  -31                     -31
\XYZ[10].Y_shr [16:0]  8                       8
\XYZ[11].Z_sign        0
\XYZ[11].X_shr [16:0]  -16                     -16
\XYZ[11].Y_shr [16:0]  4                       4
\XYZ[12].Z_sign        0
\XYZ[12].X_shr [16:0]  -8                      -8
\XYZ[12].Y_shr [16:0]  2                       2
\XYZ[13].Z_sign        0
\XYZ[13].X_shr [16:0]  -4                      -4
\XYZ[13].Y_shr [16:0]  1                       1
\XYZ[14].Z_sign        1
\XYZ[14].X_shr [16:0]  -2                      -2
\XYZ[14].Y_shr [16:0]  0                       0
X[0:15,16:0]           [0,-19500,-29250,-      [0,-19500,-29250,-31687,-31383,-30984,-31245,-31130,-31069,-31038,-31022,-31014,-31018,-31020,-31021,-31021]
Y[0:15,16:0]           [19500,19500,9750,      [19500,19500,9750,2437,6398,8360,7391,7880,8124,8246,8307,8338,8322,8314,8310,8812]
Z[0:15,31:0]           [894784853,3579139      [894784853,357913941,40980536,-126478371,-41473615,1193716,-20160749,-9480911,-4140666,-1470503,-135417,532127,198355,31469,-51974,-10253]
c_parameter[31:0]      16                      16
STG[31:0]              16                      16
```

165° = 1,968,526,677 in signed dec. From Figure 9 it is seen that degree angle is displayed in signed decimal representation. From Figure 10 we get

$$Xout1 = x1 + x2 = 11 + 54 = 65$$
$$Xout2 = x1 - x2 = 11 - 54 = -43$$

All are signed inputs.

6 Result summary

Table 6 refers to device utilisation summary of implementation of conventional and CORDIC FFT on SPARTAN 3A DSP evaluation BOARD.

Table 7 refers to timing summary reports of conventional and CORDIC FFT architecture on SPARTAN 3A DSP evaluation BOARD.

Table 8 depicts the power analysis reports of conventional and CORDIC FFT architecture on SPARTAN 3A DSP evaluation BOARD.

Figure 11 Simulation results of butterfly unit (see online version for colours)

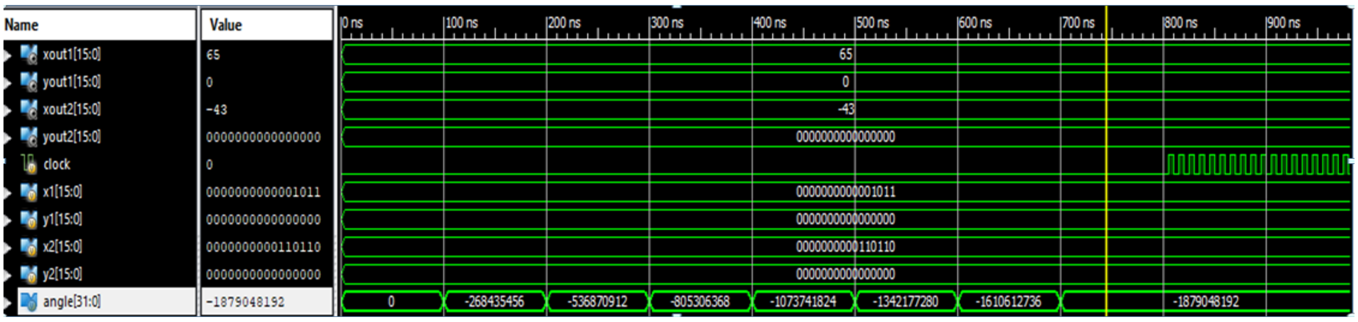


Figure 12 Simplified simulation results of CORDIC FFT architecture (see online version for colours)

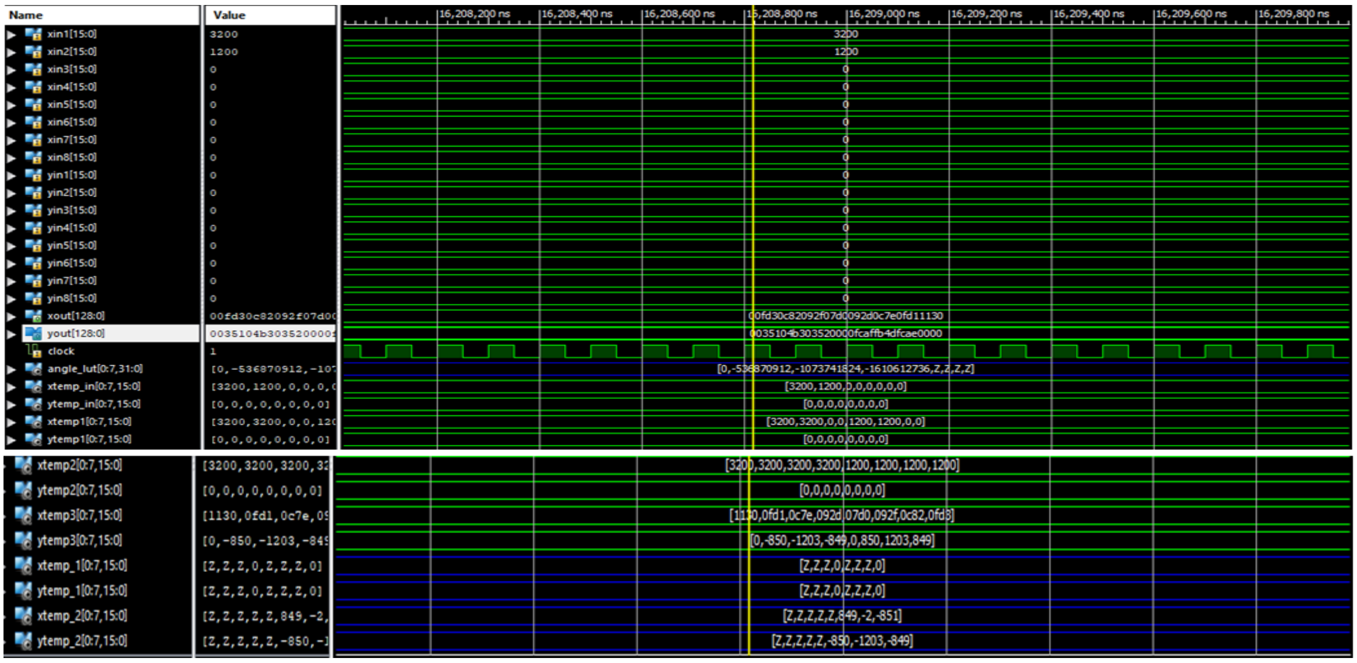

Name
Value
16,208,200 ns
16,208,400 ns
16,208,600 ns
16,208,800 ns
16,209,000 ns
16,209,200 ns
16,209,400 ns
16,209,600 ns
16,209,800 ns
xin1[15:0] 3200
xin2[15:0] 1200
xin3[15:0] 0
xin4[15:0] 0
xin5[15:0] 0
xin6[15:0] 0
xin7[15:0] 0
xin8[15:0] 0
yin1[15:0] 0
yin2[15:0] 0
yin3[15:0] 0
yin4[15:0] 0
yin5[15:0] 0
yin6[15:0] 0
yin7[15:0] 0
yin8[15:0] 0
xout[128:0] 00fd30c82092f07d0092d0c7e0fd11130
yout[128:0] 0035104b303520000fcaffb4dfcae0000
clock 1
angle_lut[0:7,31:0] [0,-536870912,-1073741824,-1610612736,Z,Z,Z,Z]
xtemp_in[0:7,15:0] [3200,1200,0,0,0,0,0,0]
ytemp_in[0:7,15:0] [0,0,0,0,0,0,0,0]
xtemp1[0:7,15:0] [3200,3200,0,0,1200,1200,0,0]
ytemp1[0:7,15:0] [0,0,0,0,0,0,0,0]
xtemp2[0:7,15:0] [3200,3200,3200,3200,1200,1200,1200,1200]
ytemp2[0:7,15:0] [0,0,0,0,0,0,0,0]
xtemp3[0:7,15:0] [1130,0fd1,0c7e,092d,07d0,092f,0c82,0fd8]
ytemp3[0:7,15:0] [0,-850,-1203,-849,0,850,1203,849]
xtemp_1[0:7,15:0] [Z,Z,Z,0,Z,Z,Z,0]
ytemp_1[0:7,15:0] [Z,Z,Z,0,Z,Z,Z,0]
xtemp_2[0:7,15:0] [Z,Z,Z,Z,Z,849,-2,-851]
ytemp_2[0:7,15:0] [Z,Z,Z,Z,Z,-850,-1203,-849]

Figure 13 Final simulation results of CORDIC FFT architecture (see online version for colours)

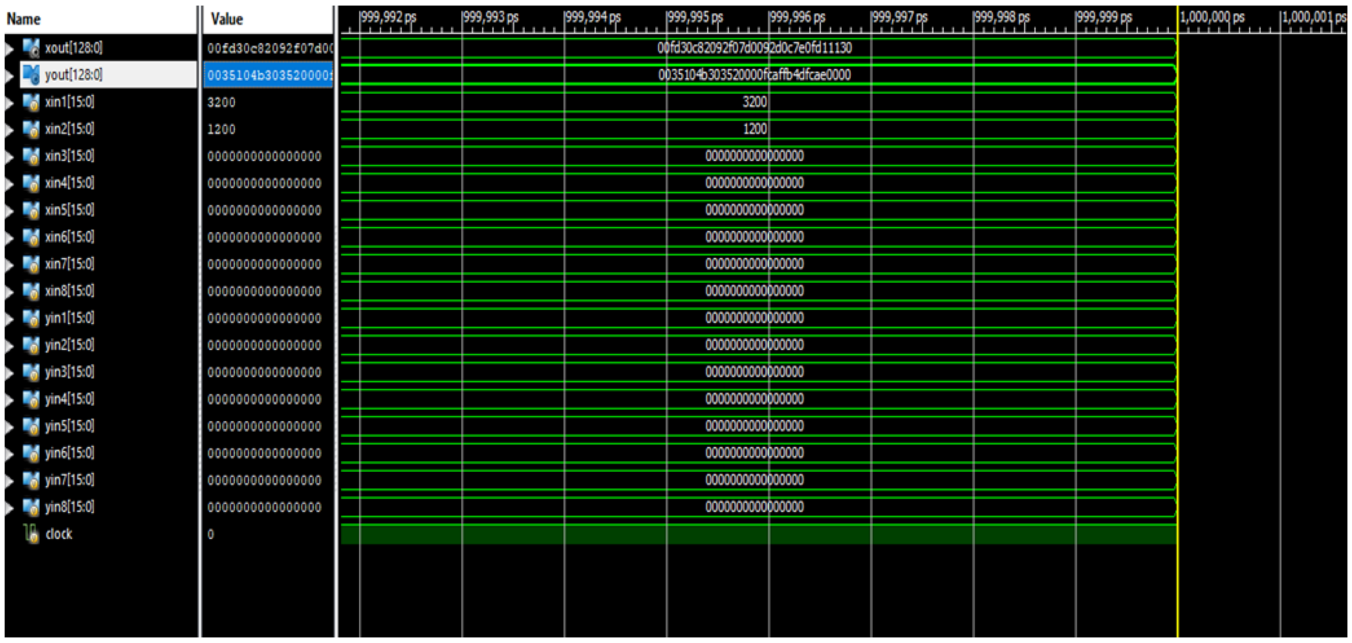
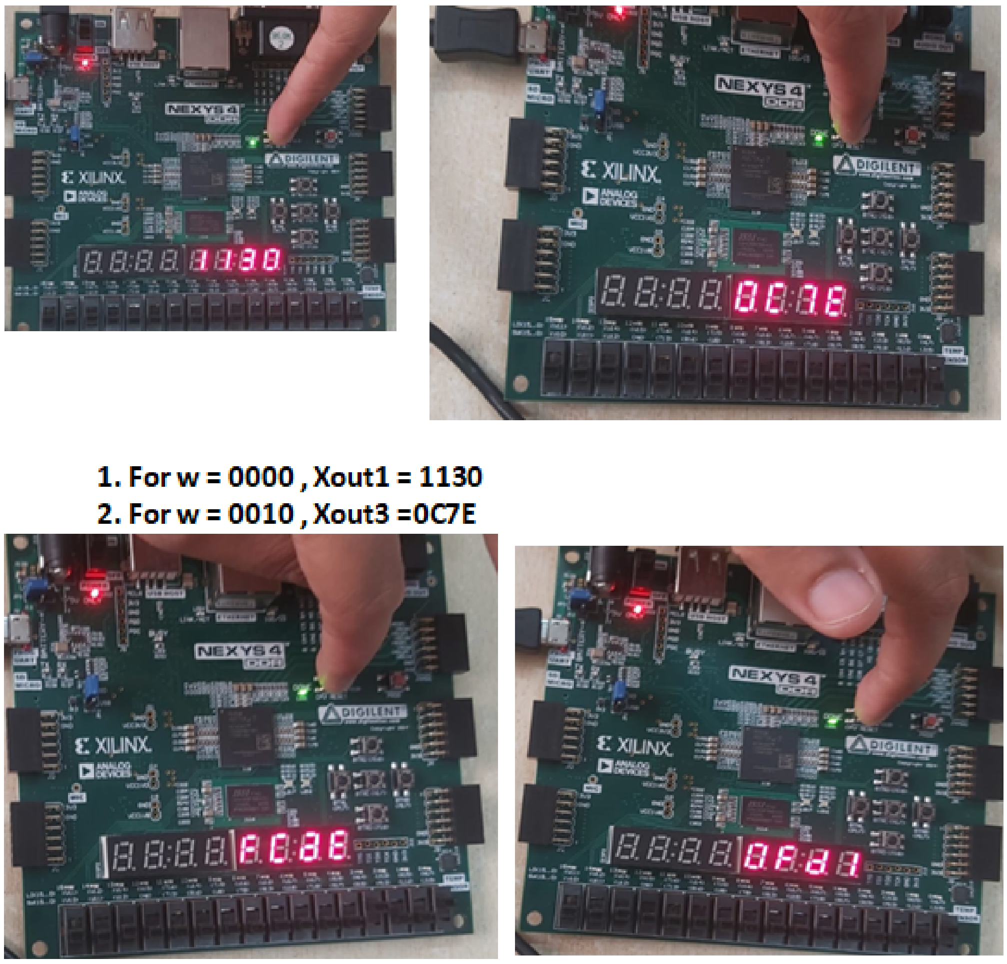

Name
Value
999,992 ps
999,993 ps
999,994 ps
999,995 ps
999,996 ps
999,997 ps
999,998 ps
999,999 ps
1,000,000 ps
1,000,001 ps
xout[128:0] 00fd30c82092f07d0092d0c7e0fd11130
yout[128:0] 0035104b303520000fcaffb4dfcae0000
xin1[15:0] 3200
xin2[15:0] 1200
xin3[15:0] 0000000000000000
xin4[15:0] 0000000000000000
xin5[15:0] 0000000000000000
xin6[15:0] 0000000000000000
xin7[15:0] 0000000000000000
xin8[15:0] 0000000000000000
yin1[15:0] 0000000000000000
yin2[15:0] 0000000000000000
yin3[15:0] 0000000000000000
yin4[15:0] 0000000000000000
yin5[15:0] 0000000000000000
yin6[15:0] 0000000000000000
yin7[15:0] 0000000000000000
yin8[15:0] 0000000000000000
clock 0

Figure 14 CORDIC FFT output on FPGA board (see online version for colours)



1. For w = 0000 , Xout1 = 1130
2. For w = 0010 , Xout3 =0C7E

1. For w = 1110 , Yout2 = FCaE
2. For w = 0010 , Xout2 =0Fd1

Table 4 Eight-point radix-2 CORDIC FFT computation

i/p	*Values*	*Stage1 o/p*	*Stage2 o/p*	*o/p*	*Final o/p in (hex)*
$x(0)$	3,200	3,200	3,200	$X(0)$	1130
$x(4)$	0	3,200	1,200	$X(1)$	0fd1
$x(2)$	0	0	3,200	$X(2)$	0c7e
$x(6)$	0	0	1,200	$X(3)$	092d
$x(1)$	1,200	1,200	3,200	$X(4)$	07d0
$x(5)$	0	1,200	1,200	$X(5)$	092f
$x(3)$	0	0	3,200	$X(6)$	0c82
$x(7)$	0	0	1,200	$X(7)$	0fd3
$y(0)$	0	0	0	$Y(0)$	0000
$y(4)$	0	0	0	$Y(1)$	face
$y(2)$	0	0	0	$Y(2)$	fb4d
$y(6)$	0	0	0	$Y(3)$	fcaf
$y(1)$	0	0	0	$Y(4)$	0000
$y(5)$	0	0	0	$Y(5)$	0352
$y(3)$	0	0	0	$Y(6)$	04b3
$y(7)$	0	0	0	$Y(7)$	0351

Table 5 Degree angle representation in signed decimal

Iteration	*Angle Z[i]*	*LUT angles*
0	894,784,853	536,870,912
1	357,913,941	316,933,405
2	40,980,536	167,458,907
3	–126,478,371	85,004,756
4	–41,473,615	42,667,331
5	1,193,716	21,354,465
6	–20,160,749	10,679,838
7	–9,480,911	5,340,245
8	–4,140,666	2,670,163
9	–1,470,503	1,335,086
10	–135,417	667,544
11	532,127	333,772
12	198,335	166,886
13	31,469	83,443
14	–51,974	41,271
15	–10,253	20,861

Table 6 Conventional and CORDIC FFT device utilisation summary

Logic utilisation	*Conventional FFT*			*CORDIC FFT*		
	Used	*Available*	*Utilisation*	*Used*	*Available*	*Utilisation*
No. of four input LUT's	1,484	33,280	4%	3,734	33,280	11%
No. of occupied slices	766	16,640	4%	1,945	16,640	11%
No. of slice FlipFlops	0	33280	0%	2,701	33,280	8%
Total no. of four input LUT's	1,484	33,280	4%	3,734	33,280	11%
No. of bonded IOBs	466	519	89%	515	519	99%
No. of DSP48As	20	84	23%	0	84	0%
Average fanout of non-clock nets	1.74			1.71		

Table 7 Conventional FFT and CORDIC FFT timing report summary

Constraint	*Conventional FFT*			*CORDIC FFT*		
	Check	*Worst case slack*	*Best case achievable*	*Check*	*Worst case slack*	*Best case achievable*
sysclkpin	SETUP	N/A	35.357 ns	SETUP	N/A	13.244 ns
100 MHz, high 50%	HOLD	6.893 ns	28.28 MHZ	HOLD	1.458 ns	75.51 MHZ

Table 8 Conventional and CORDIC FFT power report summary

FFT	*Thermal summary*	*Power supply summary (mw)*
Conventional	Effective TJA (C/W) = 15.9	Total = 128.04
	Max ambient (C) = 83.0	Dynamic = 12.47
	Junction temp (C) =27.0	Static = 115.57
CORDIC		Total = 128.04
		Dynamic = 12.47
		Static = 115.57

6.1 *Comparison of conventional FFT and CORDIC FFT*

In conventional FFT due to the usage of twiddle factors, complex adders and subtractors are involved. So 20 no. of DSPs out of 84 are involved in computations which takes device utilisation of 23%. In the case of traditional CORDIC FFT and proposed CORDIC FFT, there is no significant change in device utilisation. The maximum achievable frequency in the case of conventional FFT and traditional CORDIC FFT are 28.28 MHZ and 70.24 MHZ respectively and the proposed CORDIC FFT is 75.51 MHZ. The static power of CORDIC FFT is less than conventional FFT. Conventional FFT power is 115.57 mW and traditional and proposed CORDIC FFT power are approximately 114.25 mW respectively.

6.2 *FFT output on FPGA Nexys 4 DDR Artix 7 Board*

Setting seven segment display as a top module and considering CORDIC FFT module with decimal inputs. Output is displayed in hexadecimal format.

7 Conclusions and future scope

In this work, we have successfully simulated Verilog code for eight real and eight imaginary inputs for calculating FFT using the CORDIC algorithm. Output is shown successfully with minimal error of approx. 0.25% and an accuracy of 99.75%. Due to the use of an embedded CORDIC module, the max achievable frequency is 75.51 MHZ in the case of CORDIC FFT thereby reducing latency and throughput increases. present CORDIC FFT runs faster than conventional FFT. The computation using complex multipliers in the case of CORDIC FFT is also eliminated thereby reducing delay. The precision of FFT implementation can be further improved by providing floating-point input values (both single and double precision floating points) and an iterative CORDIC module with an increasing number of iterations. Future efficient parallel computers will improve the performance of floating-point operation. In the near future, the CORDIC module can be utilised for higher radix 16.32 point FFT with more no of iterations to achieve a faster data transfer rate.

References

Arunkumar, M. and Kirthika, N. (2013) 'Efficient implementation of ROM-LESS $FFT/IFFT$ processor using fused multiply and add unit', *International Journal of Electrical and Electronic Engg. and Telecommunications*, Vol. 2, No. 2, pp.2319–2518.

Banerjee, A. and Dhar, A. (2001) 'FPGA realization of a CORDIC based FFT processor for biomedical signal processing', *Microprocessors and Microsystems*, Vol. 25, No. 3, pp.131–142.

Bhukya, S. and Inguva, S.C. (2021) 'Design and implementation of CORDIC algorithm using integrated adder and subtractor', *International Conference for Convergence in Technology (I2CT)*.

Chaitra, Y. and Sheshadri, H.S. (2018) 'FPGA implementation of CORDIC algorithm', *International Journal of Advance Research, Ideas and Innovations in Technology*, Vol. 4, No. 3, pp.1179–1181.

Changela, A. and Zaveri, M. (2023) 'A comparative study on CORDIC algorithms and applications', *Journal of Circuits, Systems, and Computers*, Vol. 32, No. 5.

Choudhary, P. and Karmakar, A. (2011) 'CORDIC based implementation of fast Fourier transform', *International Conference on Computer and Communication Technology (ICCCT)*.

Enis, C.E.R.R.I. and Marsida, I.B.R.O. (2015) 'FFT implementation on FPGA using butterfly algorithm', *International Journal of Engineering Research and Technology*, Vol. 4, No. 2, pp.534–538.

Hsiao, S.F. (2010) 'Implementation of floating-point CORDIC rotation and vectoring based on look-up tables and multipliers', *IEEE*, pp.44–47.

Manasa, M. and Noorbasha, F. (2017) 'Comparative analysis of CORDIC algorithm and Taylor Series expansion', *Journal of Theoretical and Applied Information Technology*, Vol. 95, No. 9, pp.2015–2022.

Meggit, J.E. (1962) 'Pseudo division and pseudo multiplication processes', *IBM Journal of Research and Development*, Vol. 6, No. 2, pp.210–226.

Meher, P.K. and Valls, J. (2009) '50 years of CORDIC algorithms, architectures, and applications', *IEEE Transactions on Circuits and Systems – I*, Vol. 56, No. 9, pp.1893–1907.

Mehrotra, M. and Pandey, G. (2017) 'Implementation of FFT algorithm', *International Journal of Engineering Sciences and Research Technology*, Vol. 6, No. 5, pp.206–211.

Orian, L. and Yahav, B. (2023) 'High-throughput in-memory fast Fourier transform and polynomial multiplication', *Memories – Materials, Devices, Circuits and Systems, Elsevier Journal*, Vol. 4.

Pal, B. and Bhattacharjee, P. (2020) 'Comparison of discrete Fourier transform and fast Fourier transform with reduced number of multiplication and addition operations', *International Journal of Applied Mathematics and Informatics*, Vol. 14.

Peng, Q-b. and Li, F-j. (2011) 'Design of FFT processor based on CORDIC algorithm', *Computer Engineering*, Vol. 37, No. 23, pp.208–210.

Sapper, A. and Paim, G. (2021) 'Exploring the CORDIC algorithm and clock-gating for power-efficient fast Fourier transform hardware architectures', *Journal of Integrated Circuits and Systems*, Vol. 16, No. 2, pp.1–11.

Tang, P.T.P. (1991) 'Table-lookup algorithms for elementary functions and their error analysis', *Proc. 10th Symp. Computer Arithmetic*, pp.232–236.

Volder, J.E. (1959) 'The CORDIC trigonometric computing technique', *IRE Transactions on Electronic Computers*, Vol. EC-8, No. 3, pp.330–334.

Volder, J. (2000) 'The birth of CORDIC', *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, Vol. 25, pp.101–105.

Zhao, Y. and Lv, H. (2022) 'High performance and resource-efficient FFT processor based on CORDIC algorithm', *EURASIP Journal on Advances in Signal Processing*, Vol. 23, pp.1–14.

Zhu, B. and Lei, Y. (2017) 'Low latency and low error floating-point sine/cosine function based TCORDIC algorithm', *IEEE Transactions on Circuits and Systems-I: Regular papers*, Vol. 64, No. 4, pp.892–905.