IMPLEMENTATION OF THE OFDM BASED OTFS TRANSMITTER USING CORDIC IN VERILOG

SUBMITTED BY:

Kapil Yadav - 20je0468

PROJECT INSTRUCTOR:

PROFESSOR HIMANSHU BHUSHAN MISHRA

21 AUGUST 2023

Table of Contents

Introduction

The objective of the project is to implement a 16X16 point OTFS transmitter and receiver system using FFT (Fast Fourier Transform) and IFFT with the help of CORDIC Algorithm (radix -2 method). We used DIF FFT method to calculate Discrete Fourier Transform. CORDIC (Coordinate Rotation Digital Computer) algorithm is a simple method to calculate trigonometric functions with only shifting, addition and subtraction of bits in Verilog. It calculates trigonometric functions iteratively that is a good efficient method.

Description of CORDIC

As we know that the digital signal is nothing but a complex number, so we are giving a complex input to the cordic architecture and another input as an angle so that the complex input should be rotated by the given angle.

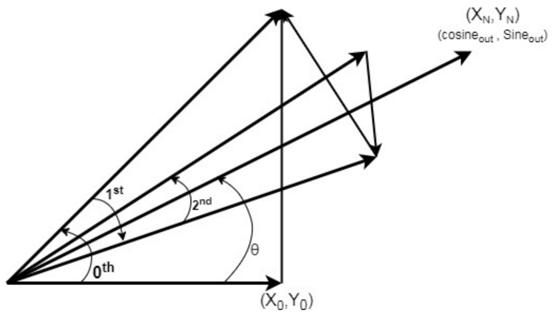
Beyond going in deep mathematics of cordic, our focus will be on method that how can we import the structure in Verilog and binary bits.

A signed angle lookup table is stored corresponds to $\tan^{-1}(2^{-i})$. In our project we are keeping the value $0 \le i \le 15$.

The angle 45° is and represented by 2^{29}(simple for binary representation) and whole table is scaled accordingly.

The iteration follows the equations written below.

$$x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}$$
$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$
$$z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i})$$



Here x and y represent the value at x axis and y axis and d represent the sign of z at ith iteration. Again, we are not going in deep mathematics as we are focusing on Verilog implementation.

Fast Fourier Transform (FFT)

FFT is a way of calculation of discrete Fourier transform that reduces the calculation complexity of DFT. Here we will use DIF FFT method.

Every N point DFT can be divided into two N/2 point DFTs of x(2m), x(2m+1). further N/2 point DFT can be divided into two N/4 DFTs and so on.

Let's take a look on 16-point DIF FFT butterfly diagram.

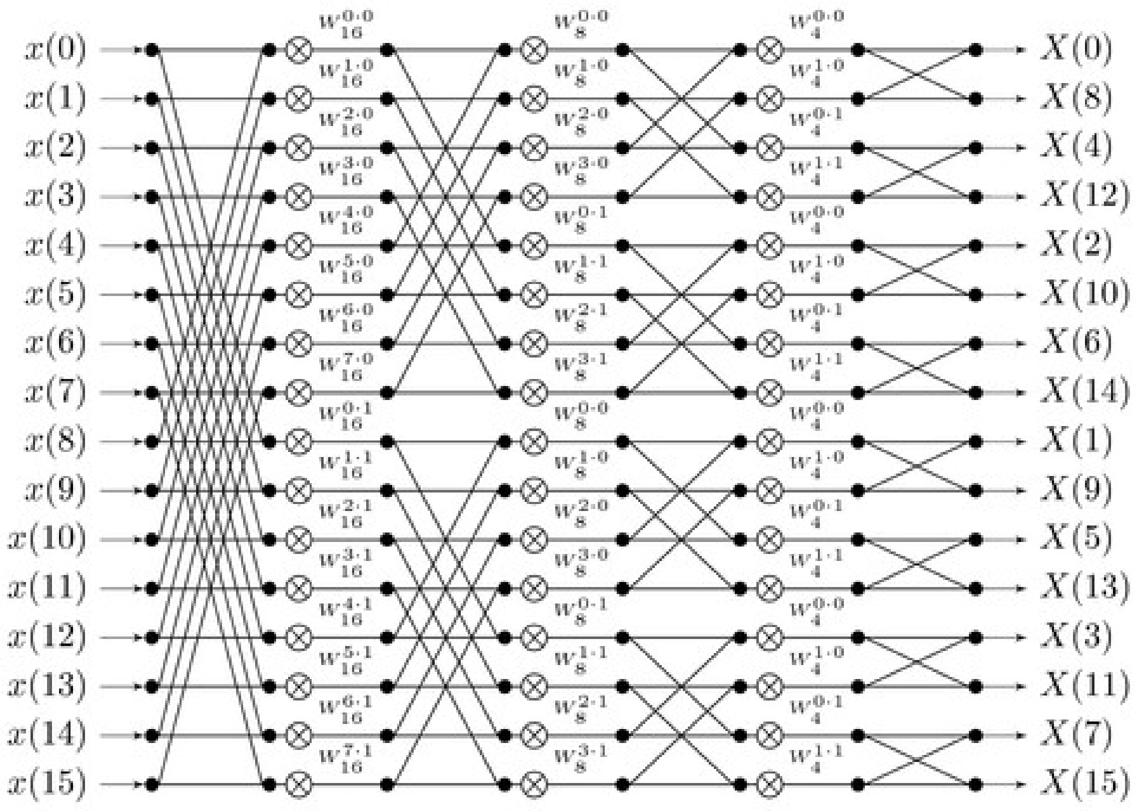


Fig – 1. 16-point DIF FFT Butterfly Diagram.

It can be seen that there is a basic building block is repeating again and again that is called butterfly module that is shown in a picture.

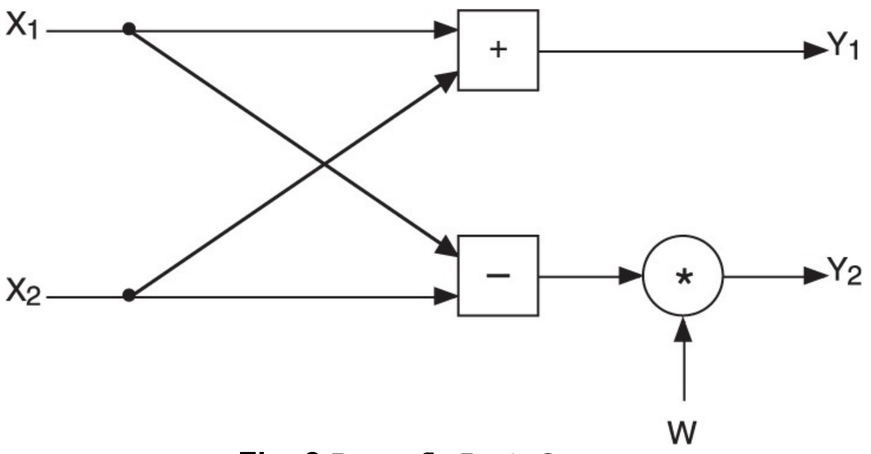


Fig. 2 Butterfly Basic Structure.

In the upper block the basic structure is explained that a butterfly takes two complex input and gives two outputs. One of those is a sum of input and another is difference of the input that is
rotated by an angle, so here comes the concept of cordic rotation.

The FFT Architecture

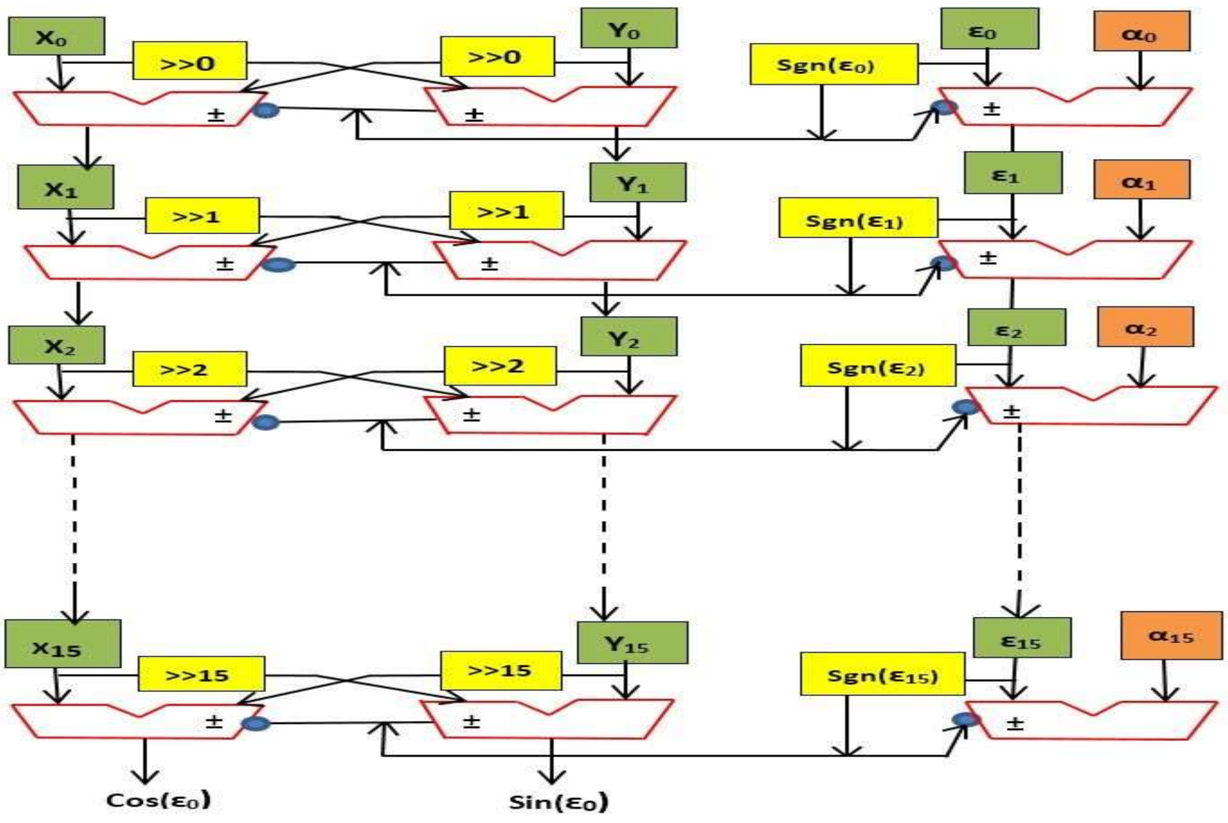


Fig-3. Pipelined Cordic Hardware

As we discussed earlier that cordic architecture is nothing but a group of iterations. IN above picture a pipelined structure of cordic is shown that is easily understandable that how iterations are taking place.

The cordic algorithm uses simple shift and add operations to iteratively rotate a vector from an initial angle to a desired angle, while maintaining a certain scaling factor. This makes it particularly efficient for hardware implementations, as it relies on basic arithmetic operations that can be easily executed in digital circuits.

A 32-bit signed angle lookup table of arctan(2^-2i) is stored in Verilog program to fed as input that is scaled by some factor as we discussed earlier. The number of iterations is kept 16 that are sufficient for reaching an expected output. So, it will take 16 clock cycles to available the output at output port.
After completing the iterations, the rotated vector should be scaled by the factor of 0.607 that can be achieved by simple shifting and addition operations.

The cordic algorithm can achieve the angle from -90 to 90 only. For Higher angler some trigonometric identities can be applied to convert that angle into smaller one.

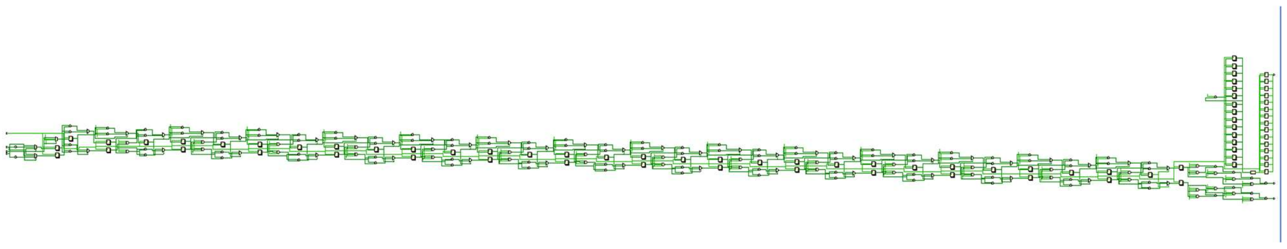


Fig-4. Verilog Created Pipelined Cordic Hardware.

A pipelined cordic module created by Verilog HDL is shown in the figure 4.

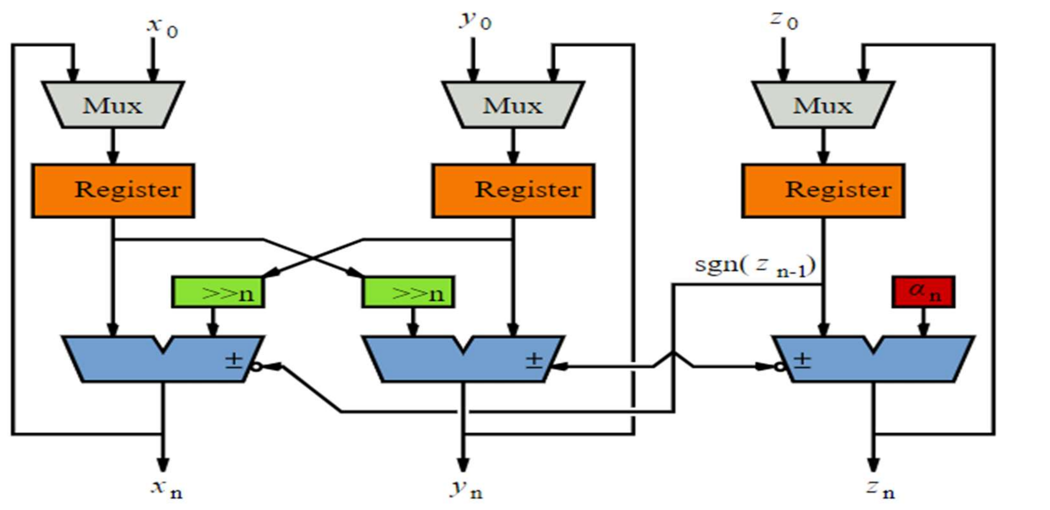


Fig-5. Bit parallel Cordic Hardware.

There is another way that is bit parallel cordic hardware to (shown in above figure 5) that simply reduces number of logic circuit, instead of going pipelined, it stores the next iteration in a register.
In both cases we have to compromise either hardware(space) or time.

Butterfly Module

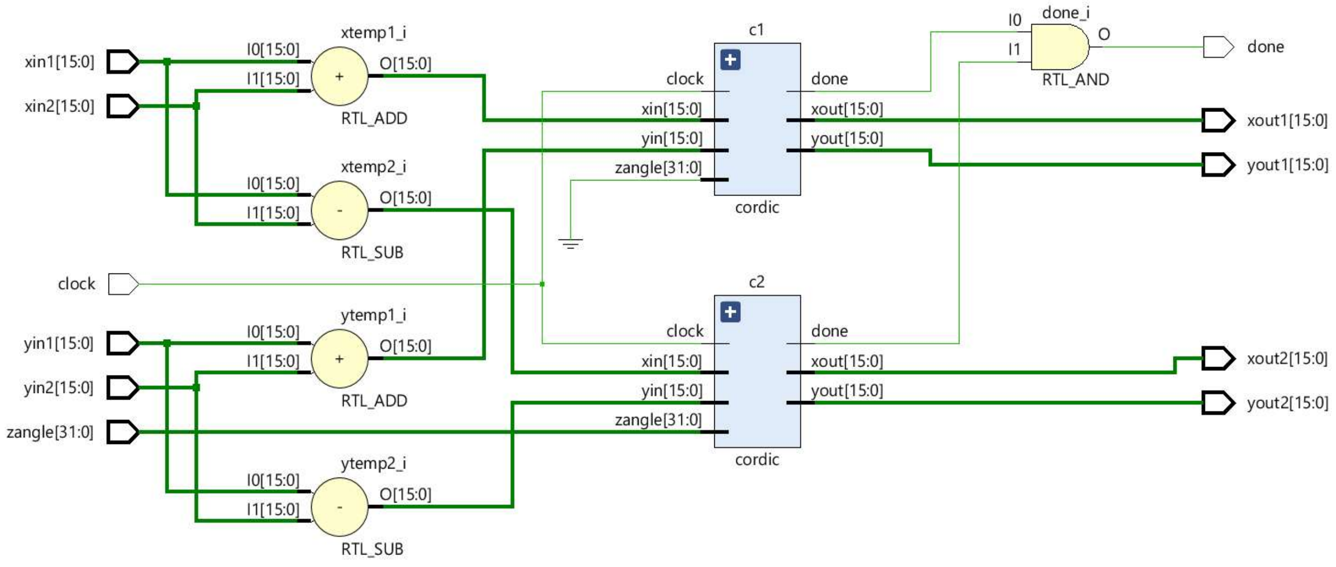


Fig-6. Verilog created Butterfly Module

Here the Verilog created butterfly module is shown in the figure 6. As we discussed earlier that it is nothing but a interface with cordic rotation of sum and difference of inputs.

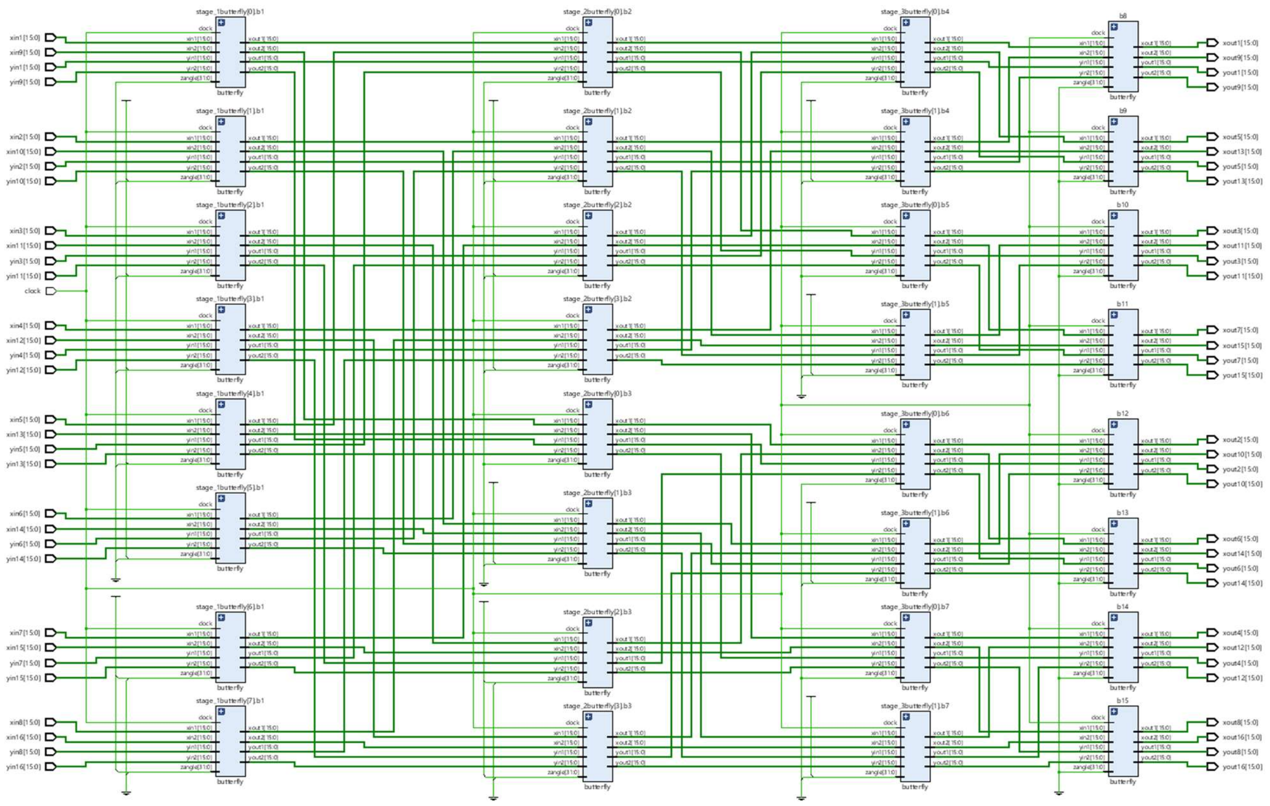


Fig-7. Verilog created FFT Module.

We roughly can see in the figure-7 that FFT Module created by Verilog is looking same as FFT butterfly signal flow graph shown in figure-1.

Same procedure can be followed for IFFT that is Very similar to FFT, only angle of rotation and scaling factor will differ from FFT so we are not going to discuss in details here.

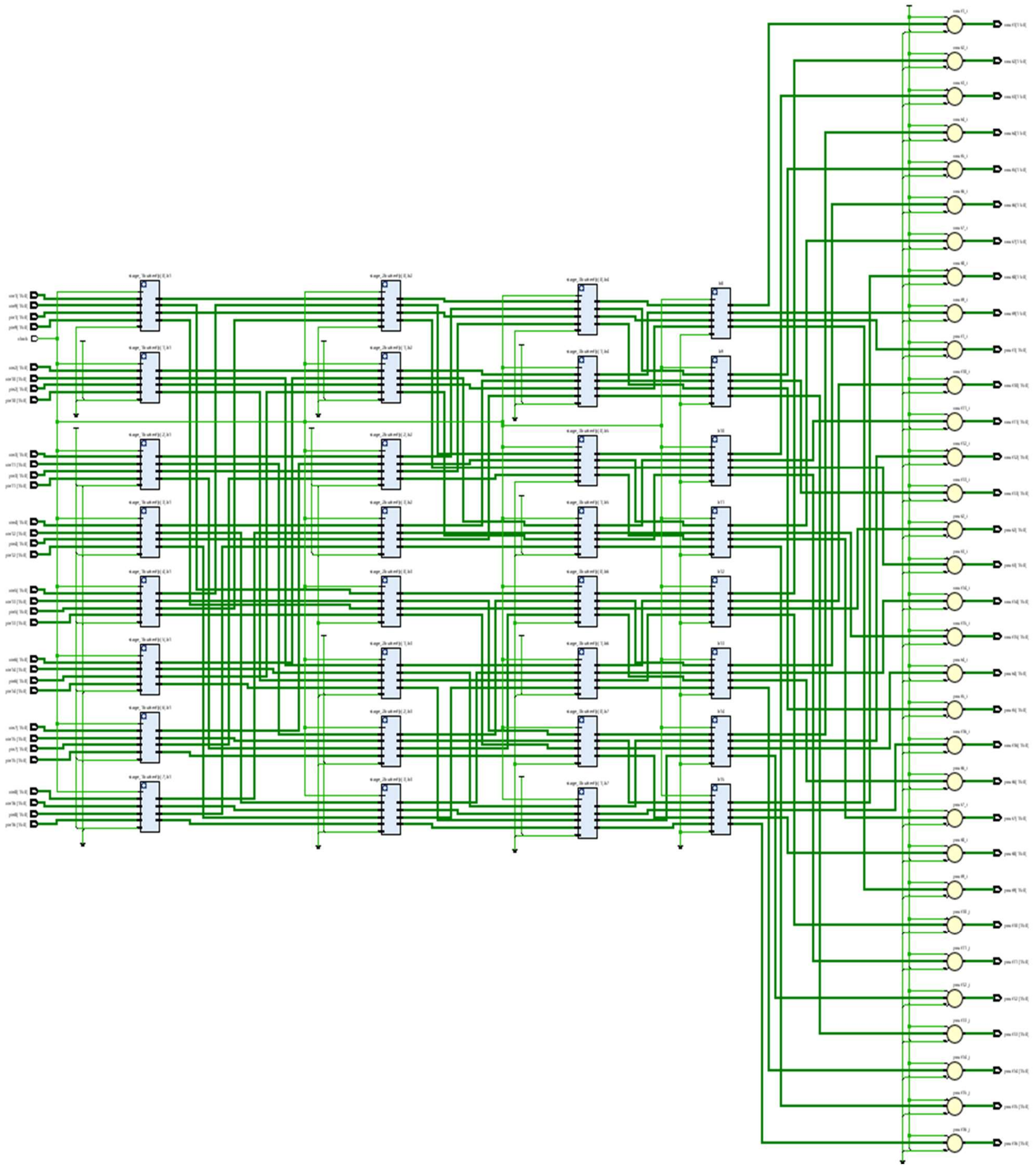


Fig-8. Verilog created IFFT module.

Description of OTFS

Orthogonal Time Frequency Space (OTFS) is a modulation and coding scheme used in wireless communication systems. It is designed to mitigate the effects of time-frequency dispersive channels, which are common in wireless communication environments. OTFS was developed as an alternative to traditional communication methods like OFDM (Orthogonal Frequency Division Multiplexing) and is intended to provide improved performance in scenarios where traditional methods may struggle due to channel impairments.

OTFS operates by converting the communication signal from the time-frequency domain to a new domain called the "delay-Doppler" domain. In this domain, the effects of channel dispersion are reduced, allowing for more reliable communication over dispersive channels. This is particularly useful in environments where the delay spread and Doppler spread of the channel vary significantly over time.

The OTFS transmitter and Receiver

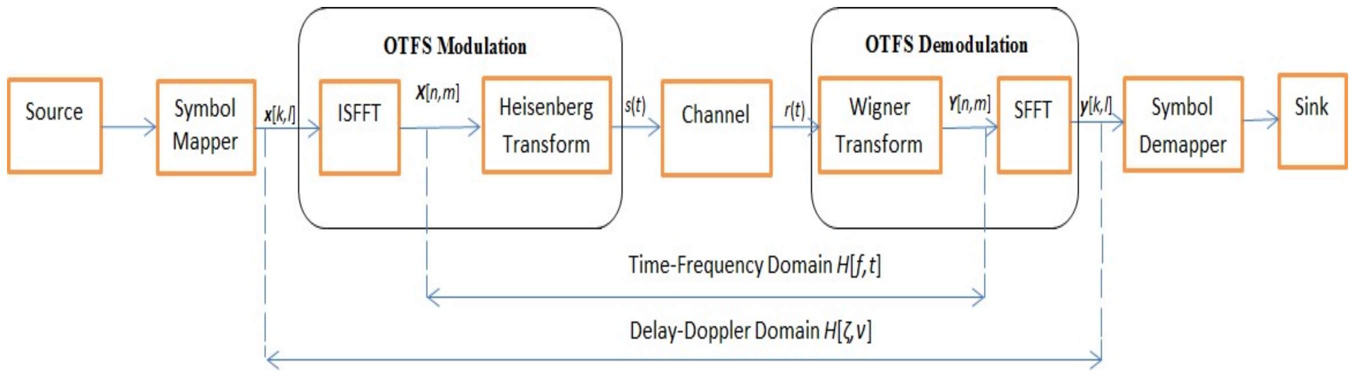


Fig – 9. OTFS Transreceiver Block Diagram.

The ISFFT operation at the transmitter can be written in matrix form as shown below:

$$\mathbf{X} = \mathbf{F}_{\mathrm{M}}\mathbf{D}\boldsymbol{F}_{N}^{H}$$

where, X represents a complex matrix with size M × N.

The signal must be delivered in the time-frequency domain. F_{M} and $\boldsymbol{F}_{N}^{H}$ are matrices that represent the FFT and IFFT, respectively. So as shown below, the Heisenberg transform equation can be stated as a matrix and converted into time domain.

$$S = (\boldsymbol{F}_N^H \otimes G_{tx})\ x$$

For the OTFS receiver, wigner transform along with SFFT, equation can be written as

$$Y = (\mathbf{F}_N \otimes \mathbf{G}_{rx})\ x$$

Here $\mathbf{G}_{tx}$ and $\mathbf{G}_{rx}$ are the pulse shaping matrix, for simplicity we are keeping these are unit matrix that represent the rectangular pulse.

So now the equations are reduced to simple FFT and IFFT as we started.
We are using 16x16 IFFT and FFT matrixes for transmitter and receiver here.

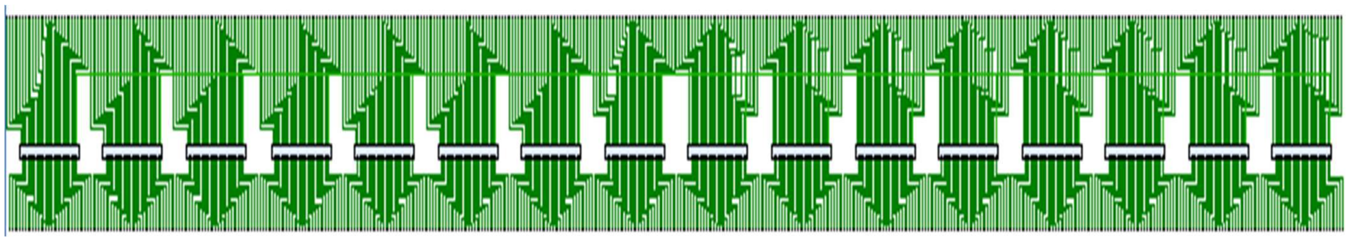
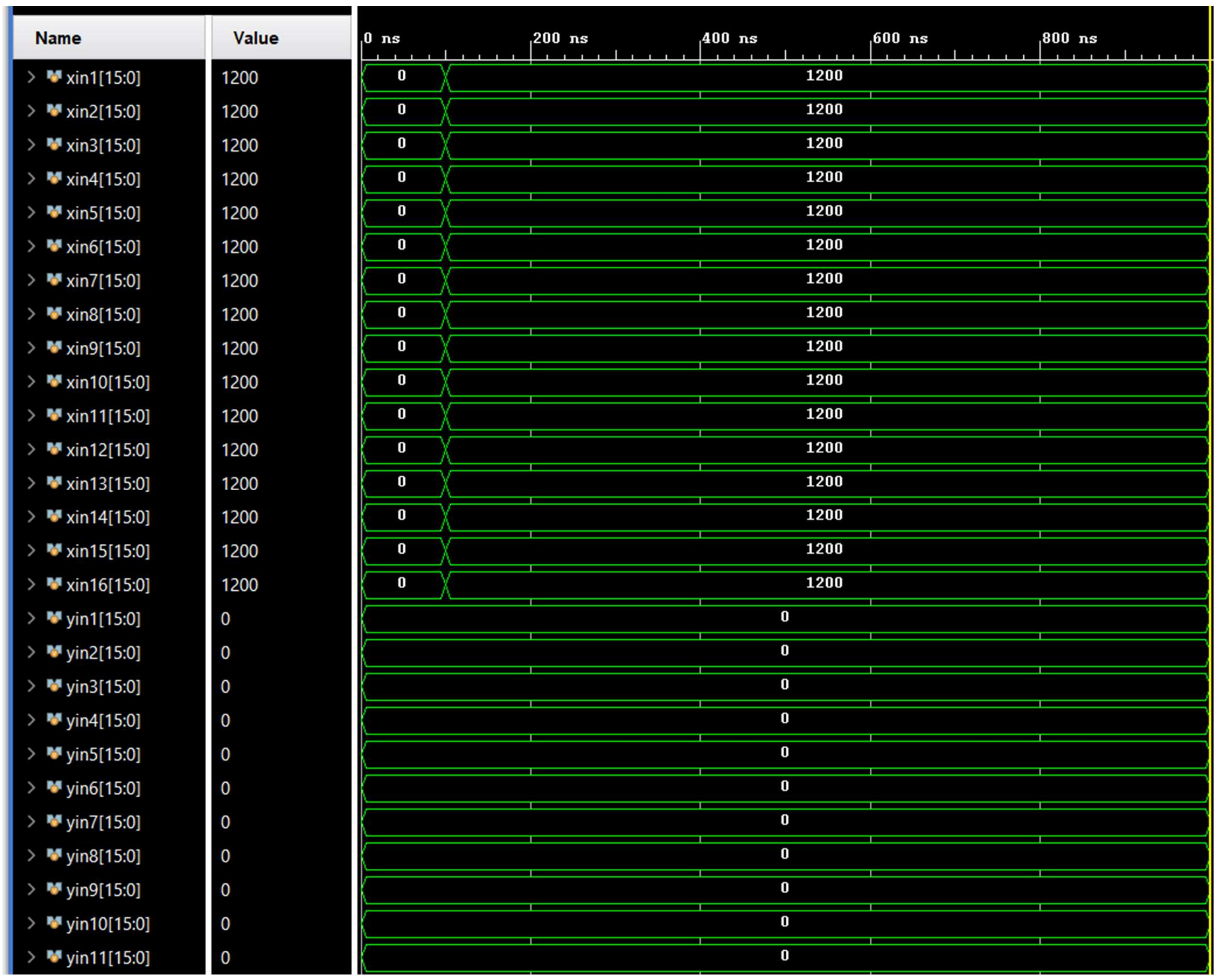


Fig – 10. Verilog created OTFS transmitter.

In the above picture a 16x16 input output Verilog created OTFS transmitter is shown, design is too complex to understand because of 16x16(256) inputs and outputs.
So, in results we will discuss only one column IFFT and FFTs for simplicity because to handle 256, a large number of inputs & outputs, will be not feasible.

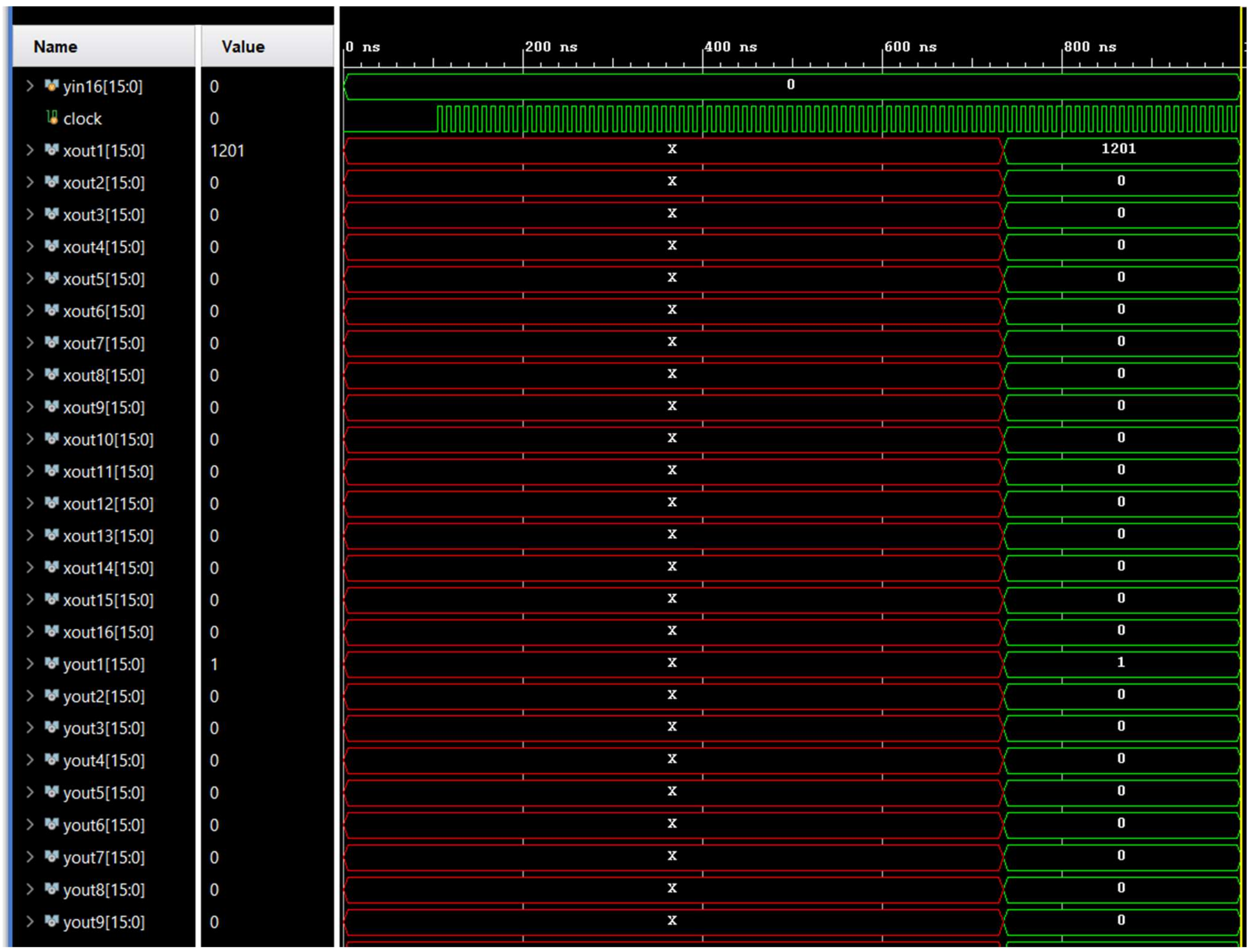
Results

The following results are observed for a 16-point IFFT of the first column of the 16x16 matrix.
Here we cannot show all the 256 outputs simultaneously so we are showing only IFFT of the first column although the behaviour of other columns is also same.
We will also tabulate the OTFS transmitter output for some standard inputs and also compare with MATLAB.



Name	Value
xin1[15:0]	1200
xin2[15:0]	1200
xin3[15:0]	1200
xin4[15:0]	1200
xin5[15:0]	1200
xin6[15:0]	1200
xin7[15:0]	1200
xin8[15:0]	1200
xin9[15:0]	1200
xin10[15:0]	1200
xin11[15:0]	1200
xin12[15:0]	1200
xin13[15:0]	1200
xin14[15:0]	1200
xin15[15:0]	1200
xin16[15:0]	1200
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
yin9[15:0]	0
yin10[15:0]	0
yin11[15:0]	0

Fig-11. Constant input to the IFFT module



Name	Value
yin16[15:0]	0
clock	0
xout1[15:0]	1201
xout2[15:0]	0
xout3[15:0]	0
xout4[15:0]	0
xout5[15:0]	0
xout6[15:0]	0
xout7[15:0]	0
xout8[15:0]	0
xout9[15:0]	0
xout10[15:0]	0
xout11[15:0]	0
xout12[15:0]	0
xout13[15:0]	0
xout14[15:0]	0
xout15[15:0]	0
xout16[15:0]	0
yout1[15:0]	1
yout2[15:0]	0
yout3[15:0]	0
yout4[15:0]	0
yout5[15:0]	0
yout6[15:0]	0
yout7[15:0]	0
yout8[15:0]	0
yout9[15:0]	0

Fig-12. Output IFFT of the constant input

INPUT	MATLAB IFFT	Verilog IFFT
1200	1200	1201+i
1200	0	0
1200	0	0
1200	0	0
1200	0	0
1200	0	0
1200	0	0
1200	0	0
1200	0	0
1200	0	0
1200	0	0
1200	0	0
1200	0	0
1200	0	0
1200	0	0

1200	0	0

Table-1: IFFT of Constant Input

INPUT	MATLAB IFFT	Verilog IFFT
1200	75	75
0	75	75
0	75	75
0	75	75
0	75	75
0	75	75
0	75	75
0	75	75
0	75	75
0	75	75
0	75	75
0	75	75
0	75	75
0	75	75
0	75	75
0	75	75

Table-2: IFFT of unit step input

INPUT	MATLAB IFFT	Verilog IFFT
1000i	450+312i	447+313i
1200	59+219i	58+220i
0	-12-75i	-13-75i
0	226+110i	225+110i
1200+1000i	125+62i	124+63i
1200	-264+203i	-266+202i
0	-137+75i	-139+74i
1200	210+108i	210+108i
0	-150+62i	-151+i62
0	10+37i	9+37i
0	-12-75i	-13-75i
1200+1000i	-118-329i	-120-331i
0	-125+62i	-126+61i
0	-55+89i	-56+90i
1200+1000i	-137+75i	-139+74i

1000i	-68+61i	-70+61i

Table-3: IFFT of any random input.

Conclusion

The OTFS transmitter by IFFT has been successfully implemented in Verilog using Cordic architecture. The relative errors are also negligible. So, the OTFS transmitter is successful and working correctly.