Introduction

The objective of the project is to implement a 16 point FFT (radix-2) using the CORDIC algorithm on Verilog. Any algorithm that can perform a discrete Fourier transform (DFT) in time is classified as a fast Fourier transform (FFT). CORDIC stands for COordination ROtation DIgital Computer. It is a simple and efficient algorithm to calculate trigonometric functions iteratively. CORDIC can be used to perform complex vector rotations by arbitrary angles. This is utilised in the FFT computation.

Description of CORDIC

Given a complex vector that must be rotated by an angle , the CORDIC algorithm performs the rotation as follows. Initialize

The negative of the total rotational angle is assigned to z_i and the value of z_i is updated after every iteration of the cordic algorithm. The angles corresponding to $\tan^{-1}(2^{-i})$, where $0 \leq i \leq 15$, is stored in a LUT(look up table), with 16 entries and each entry of the LUT is 32 bits wide. For example, the first entry of the LUT is $\tan^{-1}(2^{-0}) = \tan^{-1}(1) = 45°$. And $(45/360)*2^{32} = 2^{29}$, which is represented using 32 bits. Thus the LUT has all values of angles$\leq 45°$.

For the i^{th} iteration of the cordic module, the sign of z_i is stored as . Then according to the value of the and values are calculated. It is important to ensure that the rotation is done considering the sign of z_i, to ensure convergence of the cordic algorithm after 16 iterations.

These operations only consist of additions, subtractions, and table look-up operations (for the arctan function). The number of iterations can be chosen depending on an error criterion or arbitrarily.

Description of FFT

If it is assumed that N is even, the *N* point DFT of *x(n)* can be written as:

$$= +$$
$$=$$

where = ,is the *N/2* point DFT of *x(2m),* and is the *N/2* point DFT of *x(2m+1).* These can be computed recursively using the above formulation until the base case is reached when *N*=1, in which case the DFT is the sample *x(0)* itself. The algorithm is of the order of *O(N.logN)*. All *W* need not be calculated, since

The recursive formulation can be broken down into modules. The base module is generally called a butterfly module, and is shown in Fig. 1.

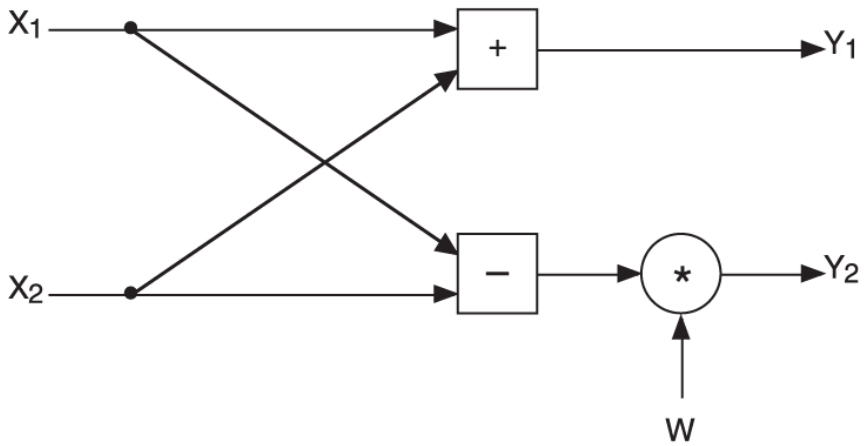

Fig. 1 *Base butterfly module*

In the base butterfly module, the output corresponds to a 2 point DFT. Fig. 2 shows many butterfly modules connected together to form the architecture for the 16-point FFT

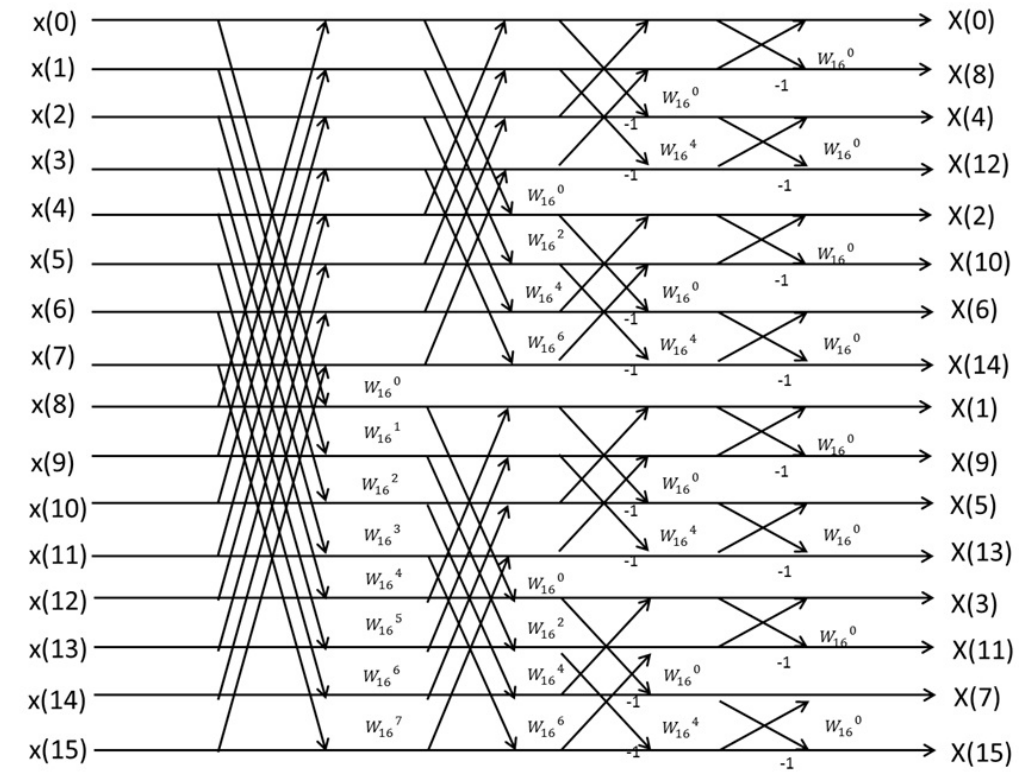

Fig. 2 *Butterfly diagram for 16-point FFT*

Description of Architecture

CORDIC module

A generic CORDIC module was created in Verilog, that took a complex input (with real and imaginary parts separately), a clock signal, and the angle that the complex input vector was expected to be rotated by. The angle (in degrees) is scaled by and fed as input, to enable simpler bit representation. The look-up table for *arctan* operations was fed as input to the module, with 32 bit signed precision. 16 entries were required for for *i*=0 to *i*=15. A *generate* block with 16 iterations was used to iteratively update the vector coordinates. The block executes once every clock cycle. Until *i*-1 clock cycles are over, the input of the stage will remain in high impedance mode. Thus the module requires 16 cycles to calculate the output.

As the number of iterations , the magnitude of the rotated vector is scaled with respect to the original vector be a factor of 1.64. Hence the result needs to be divided by *A*, or equivalently multiplied by 1/*A* 0.60. 1/2 + 1/16 + 1/32 = 0.59375, which is a good approximation of 1/*A*. Hence, by simple shifting and adding operations, the output can be approximately scaled to the desired output without the need for multiplication.

The CORDIC rotation was defined for only angles from to degrees. Hence, all angles in other quadrants needed to be corrected for. The output of the module is . Hence, for angles in the range to 180 degrees, 90 degrees was subtracted from the angle, and the initial values for *x* and *y* were assigned as - and , where and are the imaginary and real parts the of the vector to be rotated. The output would then be = . Similarly, for angles in the range from -180 to 90 degrees, 90 degrees was added to the angle, and the initial values for *x* and *y* were assigned as and .

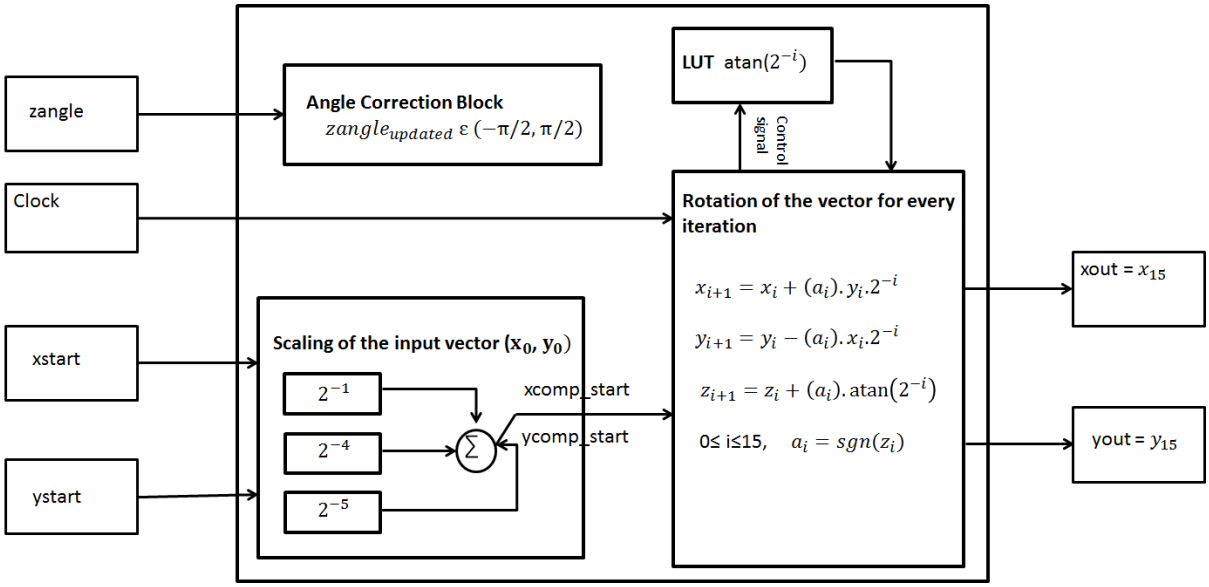

Fig. 3 *CORDIC module*

Butterfly module

A butterfly module was created, that took as input a clock signal, two complex inputs (two real parts and two imaginary parts), the angle of rotation, and produced two complex outputs. Since the CORDIC takes 16 clock cycles to complete, the output of the lower wing will be available to be added with the upper one only after 16 cycles. Hence the upper 'wing' of the butterfly was put through a CORDIC module that would turn 0 degrees. Thus the two outputs of the upper and lower wings would be available at the same time to be added and subtracted to produce the final result of the module.

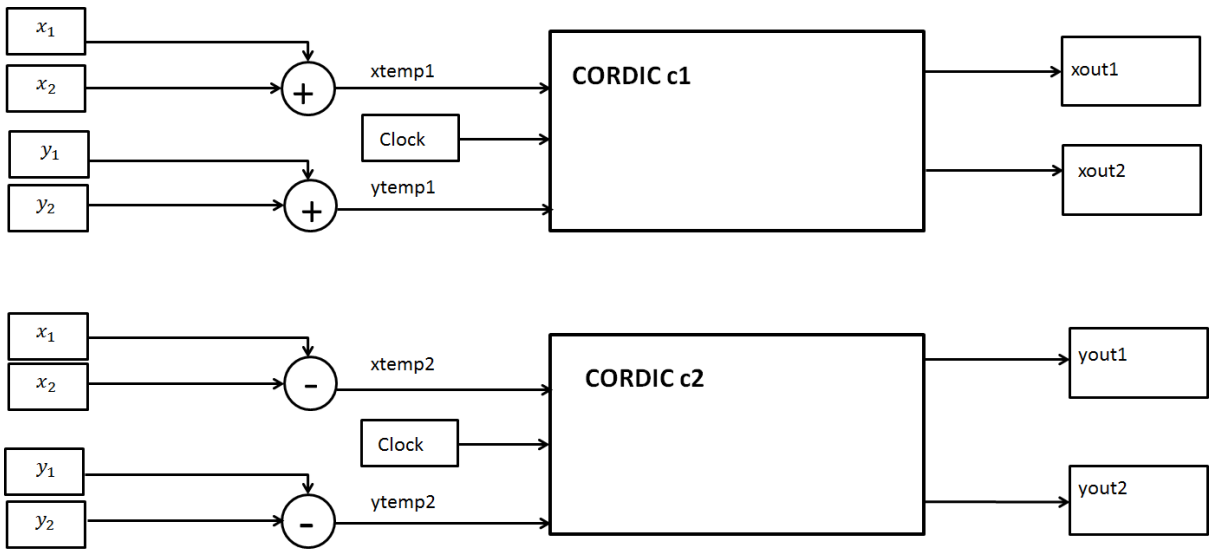

Fig. 4 *CORDIC modules after Butterfly addition and subtraction*

Main module

In the main module, the various butterfly modules required (based on Fig. 2) were created with proper indexing of the input and the output.

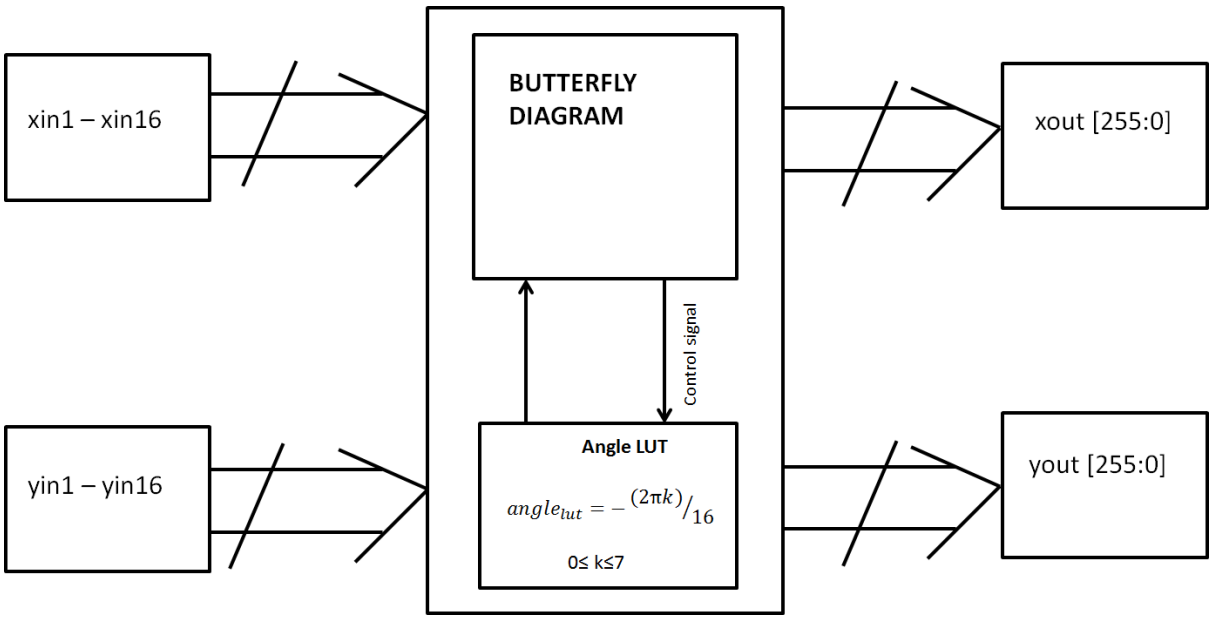

Fig. 5 *Main Module*

Issues faced during implementation

The butterfly modules were connected together at the end using structural modelling. Each module required 16 clock cycles for the output to be made available to the next module. However, there was no need for a delay unit, since until the output was made available, the unit produced a high impedance signal. Hence, the output of the series of butterfly modules was read after 64 clock cycles. Until then, the output was high impedance.

The CORDIC scaling approximation ($1/A$) as well as the iterative procedure produced errors in the implementation. The errors accumulate as they flow through the butterfly modules. Hence the final results are not 100% accurate. The relative error is about 10%, and hence it is safer to go for larger values of input.

Floating point numbers are not handled but this can be remedied by simply scaling up the input by the desired precision for digits after the decimal point. At the output they can be scaled down.

Results

The following tables show the results for 16 point FFTs computed using the above codes and the MATLAB results for a delta input, constant input, and a sinusoidal input.

Input	MATLAB FFT	Verilog FFT
3200	3200	2921+2i
0	3200	2921+2i

0	3200	2921+2i
0	3200	2921+2i
0	3200	2921+2i
0	3200	2921+2i
0	3200	2921+2i
0	3200	2921+2i
0	3200	2921+2i
0	3200	2921+2i
0	3200	2921+2i
0	3200	2921+2i
0	3200	2921+2i
0	3200	2921+2i
0	3200	2921+2i
0	3200	2921+2i

Table 1 *Result comparison for delta function*

Input	MATLAB FFT	Verilog FFT
320	5120	4683 + 6i
320	0	0
320	0	0
320	0	0
320	0	0
320	0	0
320	0	0
320	0	0
320	0	0
320	0	0
320	0	0
320	0	0
320	0	0

320	0	0
320	0	0
320	0	0
320	0	0

Table 2 *Result comparison for constant*

Input	MATLAB FFT	Verilog FFT
1024	8192	7480+21i
1024	1024-5147.99i	909-4705i
1024	0	0
1024	1024-1532.52i	910-1385i
1024	0	0
1024	1024-684.21i	917-633i
1024	0	0
1024	1024-203.69i	923-178i
0	0	0
0	1024+203.69i	930+196i
0	0	0
0	1024+684.21i	927+624i
0	0	0
0	1024+1532.52i	943+1416i
0	0	0
0	1024+5147.99i	943+4700i

Table 3 *Result comparison for single pulse*

Input	MATLAB FFT	Verilog FFT
0	-9 + 0i	-49 + 22i
973	60.84 - 303.02i	33 - 257i

601	358.60 - 864.75i	311 - 788i
-602	4078.13 - 6105.12i	3702 - 5568i
-974	-1574 + 1575i	-1459 + 1456i
-1	-972.01 + 649.71i	-907+ 595i
973	-816.60 + 337.25i	-751 + 304i
601	-758.96 + 151.81i	-702 + 131i
-602	-743 + 0i	-681 - 6i
-974	-758.96 - 151.81i	-704 - 141i
-1	-816.60 - 337.25i	-748 - 307i
973	-972.01 - 649.71i	-900 - 594i
601	-1574 - 1575i	-1451 - 1448i
-602	4078.13+6105.12i	3728 + 5579i
-974	358.61 + 864.75i	326. + 773i
-1	60.84 + 303.02i	53 + 252i

Table 4 *Result comparison for sine wave*

Input	Relative mean square error
Delta	0.76 %
Constant	0.045 %
Pulse	0.46 %
Sinusoid	1 %

Table 5 *Relative errors*

Discussion

1. In our hardware implementation of the FFT algorithm, the final result is available only after 64 (16X4) clock cycles, as there are 4 stages in the butterfly diagram and for each stage the cordic runs for 16 iterations.
2. We have assumed the inputs and outputs of all the modules to be 16 bit signed decimal numbers. The angle look-up table and the angle update variable(z_i) are taken as 32 bit signed binary numbers.

3. An alternative method of creating the final module stage would be to simply use only 8 CORDIC modules (corresponding to angles from 0 to 90 degrees), instead of using different CORDIC modules at each stage. This requires different routing at the main module stage.
4. A pipelined architecture for CORDIC would give faster results if the same module is to be used for all operations. However, we are using CORDIC modules in parallel, preferring speed over reduction in hardware.

Conclusion

The fast Fourier transform has been successfully implemented in Verilog, making use of only shifting, addition, and subtraction operations through the CORDIC algorithm. Various optimizations and The relative error in the results are not significant and can be attributed to rounding errors, as well as to errors due to approximations in the scaling of the CORDIC result.