EE5516 VLSI Architectures for Signal Processing and Machine Learning
Mini project Report

AMITH P JOY-122001006

Abstract

This project designs the architecture for an 8-point radix 2 decimation in time fast Fourier transform. The project also discusses how the same architecture used for FFT can be used to compute the IFFT. The FFT architecture is then pipelined to get reduce the critical time. The FFT/IFFT and its pipelined version are implemented in Verilog.

1. Introduction

A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. In the FFT formula, the DFT equation $X(k) = \sum x(n)W_N^{nk}$ is decomposed into a number of short transforms and then recombined. FFT is a fast and efficient algorithm that can transform signals quickly compared to other methods. This makes it useful in real-time applications where speed is critical. FFT is an algorithm that uses a divide-and-conquer approach to compute the DFT with fewer computations. Instead of computing the DFT directly, FFT breaks the signal into smaller sub-signals and applies the DFT recursively to these sub-signals. This allows FFT to compute the DFT with O(N log N) computations, which is much faster than the direct method for large N.

2. Implementation of FFT

The FFT architecture is implemented as similar to the architectural diagram given above. To perform complex additions and multiplications, the real part and imaginary part of the numbers are calculated separately. Different wires have been defined for the real part and imaginary part of the numbers (wires with names ending with '_r' are for the real part and wires ending with names ending with '_i' are for the imaginary part of numbers. The multipliers multiply the corresponding twiddle factors at each stage. Since twiddle factors are also complex numbers, the calculation of the real part and the imaginary part is done separately. We assume that the inputs are given at the same time parallelly.

The inputs and twiddle factors are in 1.15 fixed point representation, but the addition in each stage requires an additional bit in the integer part of the fixed point representation for getting the correct output. **Since there are 3 stages, we need 3 additional bits in the integer part of the fixed point number. To maintain the number of bits in the output as 16, the last 3 bits in the integer part must be removed. Approximation by adding 1 to the 3rd bit from the right end has been done. By analyzing the bit growth in the 8-point FFT architecture, to get accurate results, 4.12 fixed point representation has been used.**

The 1.15 inputs are first converted to 4.15 by bit extension and then the last 3 bits are removed to get 4.12 fixed point numbers. The multiplication in each stage makes the number 5.27 fixed point which is then converted to 4.12 by selecting the [30:15] bits of the product. Since the twiddle factors are 1 or less than 1, bit growth does not occur due to multiplication.

[Edaplayground Link]

3. Experimental Procedure

A testbench has been designed which gives inputs from the text file ‘inputs.txt’. The FFT has been calculated using the architecture and the magnitude of the output complex number will be written in the textfile ‘output_tracker_mag.txt’. The real and imaginary parts of the outputs are also being written to the text file ‘output_tracker.txt’. The FFT output magnitudes are then compared with the FFT magnitudes obtained from Matlab using the fft() function. The testbench also drives the reset signal, sets the input ‘mode’, and calculates the magnitude of the FFT outputs from the real and imaginary parts of the outputs.

Matlab code calculating the magnitude of the fft of the inputs:

```
in=[0.5,-1,-1,-1,-1,-1,-1,-1,
-1,0.5,-1,-1,-1,-1,-1,-1,
-1,-1,0.5,-1,-1,-1,-1,-1,
-1,-1,-1,0.5,-1,-1,-1,-1,
-1,-1,-1,-1,0.5,-1,-1,-1,
-1,-1,-1,-1,-1,0.5,-1,-1,
-1,-1,-1,-1,-1,-1,0.5,-1,
-1,-1,-1,-1,-1,-1,-1,0.5];
X=fft(in);
disp(abs(X));
```

4. Experimental Results

Matlab output:(stored as columns)

```
6.5000   6.5000   6.5000   6.5000   6.5000   6.5000   6.5000   6.5000
 1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000
 1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000
 1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000
 1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000
 1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000
 1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000
 1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000
```

Output_tracker_mag.txt:(It is stored as a single column)

```
6.5
1.5
1.5
1.5
1.5
1.5
1.5
1.5
6.5
1.500012417582929
1.5
1.500185041140405
1.5
```

```
1.500012417582929
1.5
1.500185041140405
6.5
1.5
1.5
1.5
1.5
1.5
1.5
1.5
6.5
1.500185041140405
1.5
1.500012417582929
1.5
1.500185041140405
1.5
1.500012417582929
6.500244140625
1.499755859375
1.499755859375
1.499755859375
1.499755859375
1.499755859375
1.499755859375
1.499755859375
6.500244140625
1.499667150602215
1.499755859375
1.499494507174402
1.499755859375
1.499667150602215
1.499755859375
1.499494507174402
6.500244140625
1.499755859375
1.499755859375
1.499755859375
1.499755859375
1.499755859375
1.499755859375
1.499755859375
6.500244140625
1.499494507174402
1.499755859375
1.499667150602215
1.499755859375
1.499494507174402
```

1.499755859375
1.499667150602215

The output magnitudes can be seen to be the same.

Matlab code for displaying complex FFT:
```
in=[0.5,-1,-1,-1,-1,-1,-1,-1,
   -1,0.5,-1,-1,-1,-1,-1,-1,
   -1,-1,0.5,-1,-1,-1,-1,-1,
   -1,-1,-1,0.5,-1,-1,-1,-1,
   -1,-1,-1,-1,0.5,-1,-1,-1,
   -1,-1,-1,-1,-1,0.5,-1,-1,
   -1,-1,-1,-1,-1,-1,0.5,-1,
   -1,-1,-1,-1,-1,-1,-1,0.5];
X=fft(in);
disp((X));
```

Matlab output:

```
Column 1

 -6.5000 + 0.0000i
  1.5000 + 0.0000i
  1.5000 + 0.0000i
  1.5000 + 0.0000i
  1.5000 + 0.0000i
  1.5000 + 0.0000i
  1.5000 + 0.0000i
  1.5000 + 0.0000i

 Column 2

 -6.5000 + 0.0000i
  1.0607 - 1.0607i
  0.0000 - 1.5000i
 -1.0607 - 1.0607i
 -1.5000 + 0.0000i
 -1.0607 + 1.0607i
  0.0000 + 1.5000i
  1.0607 + 1.0607i

 Column 3

 -6.5000 + 0.0000i
  0.0000 - 1.5000i
 -1.5000 + 0.0000i
  0.0000 + 1.5000i
  1.5000 + 0.0000i
  0.0000 - 1.5000i
```

```
-1.5000 + 0.0000i
 0.0000 + 1.5000i

Column 4

-6.5000 + 0.0000i
-1.0607 - 1.0607i
 0.0000 + 1.5000i
 1.0607 - 1.0607i
-1.5000 + 0.0000i
 1.0607 + 1.0607i
 0.0000 - 1.5000i
-1.0607 + 1.0607i

Column 5

-6.5000 + 0.0000i
-1.5000 + 0.0000i
 1.5000 + 0.0000i
-1.5000 + 0.0000i
 1.5000 + 0.0000i
-1.5000 + 0.0000i
 1.5000 + 0.0000i
-1.5000 + 0.0000i

Column 6

-6.5000 + 0.0000i
-1.0607 + 1.0607i
 0.0000 - 1.5000i
 1.0607 + 1.0607i
-1.5000 + 0.0000i
 1.0607 - 1.0607i
 0.0000 + 1.5000i
-1.0607 - 1.0607i

Column 7

-6.5000 + 0.0000i
 0.0000 + 1.5000i
-1.5000 + 0.0000i
 0.0000 - 1.5000i
 1.5000 + 0.0000i
 0.0000 + 1.5000i
-1.5000 + 0.0000i
 0.0000 - 1.5000i

Column 8
```

```
 -6.5000 + 0.0000i
  1.0607 + 1.0607i
  0.0000 + 1.5000i
 -1.0607 + 1.0607i
 -1.5000 + 0.0000i
 -1.0607 - 1.0607i
  0.0000 - 1.5000i
  1.0607 - 1.0607i
```

Real part of output from edaplayground:

```
-6.5
1.5
1.5
1.5
1.5
1.5
1.5
1.5
-6.5
1.060546875
0
-1.060791015625
-1.5
-1.060546875
0
1.060791015625
-6.5
0
-1.5
0
1.5
0
-1.5
0
-6.5
-1.060791015625
0
1.060546875
-1.5
1.060791015625
0
-1.060546875
-6.500244140625
-1.499755859375
1.499755859375
-1.499755859375
1.499755859375
-1.499755859375
```

```
1.499755859375
-1.499755859375
-6.500244140625
-1.060546875
0
1.060302734375
-1.499755859375
1.060546875
0
-1.060302734375
-6.500244140625
0
-1.499755859375
0
1.499755859375
0
-1.499755859375
0
-6.500244140625
1.060302734375
0
-1.060546875
-1.499755859375
-1.060302734375
0
1.060546875
```

Imaginary part of output from edaplayground:

```
0
0
0
0
0
0
0
0
0
0
-1.060791015625
-1.5
-1.060791015625
0
1.060791015625
1.5
1.060791015625
0
-1.5
0
1.5
```

```
0
-1.5
0
1.5
0
-1.060791015625
1.5
-1.060791015625
0
1.060791015625
-1.5
1.060791015625
0
0
0
0
0
0
0
0
0
1.060302734375
-1.499755859375
1.060302734375
0
-1.060302734375
1.499755859375
-1.060302734375
0
1.499755859375
0
-1.499755859375
0
1.499755859375
0
-1.499755859375
0
1.060302734375
1.499755859375
1.060302734375
0
-1.060302734375
-1.499755859375
-1.060302734375
Outputs are the same
```

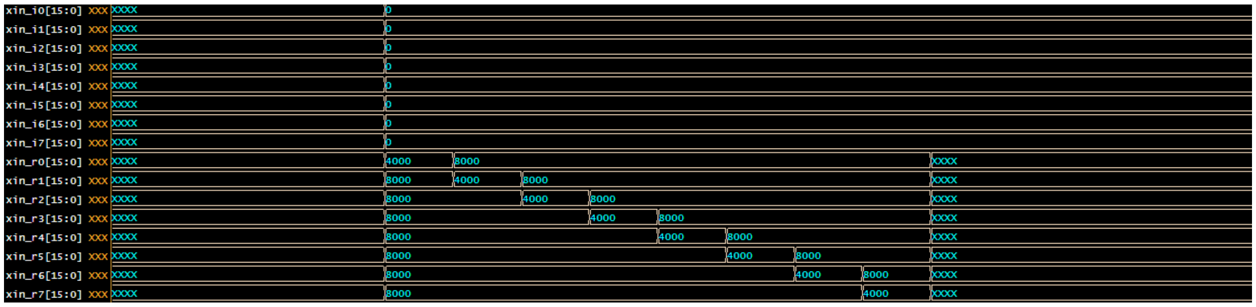
xin_i0[15:0] xxx xxxx 0
xin_i1[15:0] xxx xxxx 0
xin_i2[15:0] xxx xxxx 0
xin_i3[15:0] xxx xxxx 0
xin_i4[15:0] xxx xxxx 0
xin_i5[15:0] xxx xxxx 0
xin_i6[15:0] xxx xxxx 0
xin_i7[15:0] xxx xxxx 0
xin_r0[15:0] xxx xxxx 4000 8000 xxxx
xin_r1[15:0] xxx xxxx 8000 4000 8000 xxxx
xin_r2[15:0] xxx xxxx 8000 4000 8000 xxxx
xin_r3[15:0] xxx xxxx 8000 4000 8000 xxxx
xin_r4[15:0] xxx xxxx 8000 4000 8000 xxxx
xin_r5[15:0] xxx xxxx 8000 4000 8000 xxxx
xin_r6[15:0] xxx xxxx 8000 4000 8000 xxxx
xin_r7[15:0] xxx xxxx 8000 4000 xxxx

Fig : Inputs given to FFT/IFFT architecture(1.16 fixed point), total of 64 different inputs given 8 at a time

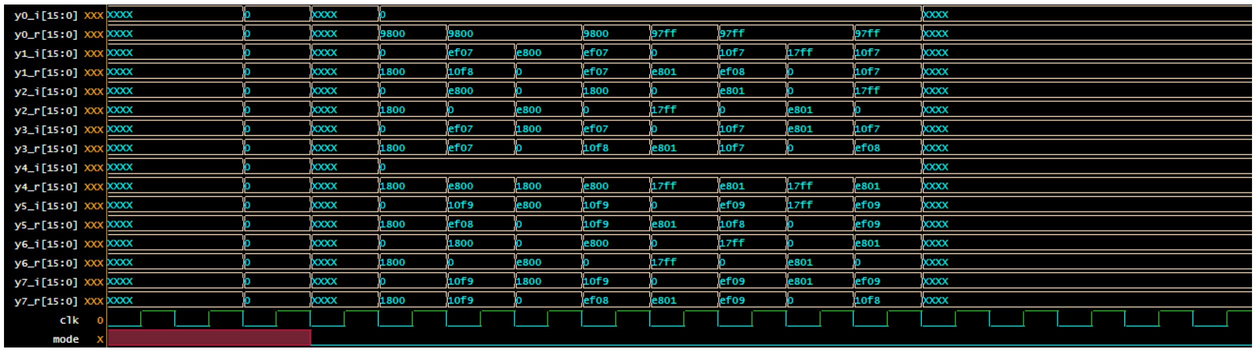
y0_i[15:0] xxx xxxx 0 xxxx 0 xxxx
y0_r[15:0] xxx xxxx 0 xxxx 9800 9800 9800 97ff 97ff 97ff xxxx
y1_i[15:0] xxx xxxx 0 xxxx 0 ef07 e800 ef07 0 10f7 17ff 10f7 xxxx
y1_r[15:0] xxx xxxx 0 xxxx 1800 10f8 0 ef07 e801 ef08 0 10f7 xxxx
y2_i[15:0] xxx xxxx 0 xxxx 0 e800 0 1800 0 e801 0 17ff xxxx
y2_r[15:0] xxx xxxx 0 xxxx 1800 0 e800 0 17ff 0 e801 0 xxxx
y3_i[15:0] xxx xxxx 0 xxxx 0 ef07 1800 ef07 0 10f7 e801 10f7 xxxx
y3_r[15:0] xxx xxxx 0 xxxx 1800 ef07 0 10f8 e801 10f7 0 ef08 xxxx
y4_i[15:0] xxx xxxx 0 xxxx 0 xxxx
y4_r[15:0] xxx xxxx 0 xxxx 1800 e800 1800 e800 17ff e801 17ff e801 xxxx
y5_i[15:0] xxx xxxx 0 xxxx 0 10f9 e800 10f9 0 ef09 17ff ef09 xxxx
y5_r[15:0] xxx xxxx 0 xxxx 1800 ef08 0 10f9 e801 10f8 0 ef09 xxxx
y6_i[15:0] xxx xxxx 0 xxxx 0 1800 0 e800 0 17ff 0 e801 xxxx
y6_r[15:0] xxx xxxx 0 xxxx 1800 0 e800 0 17ff 0 e801 0 xxxx
y7_i[15:0] xxx xxxx 0 xxxx 0 10f9 1800 10f9 0 ef09 e801 ef09 xxxx
y7_r[15:0] xxx xxxx 0 xxxx 1800 10f9 0 ef08 e801 ef09 0 10f8 xxxx
clk 0
mode x

Fig: Output values(both real and imaginary parts)

The latency is seen to be 0 clock cycles. The throughput of the pipelined architecture is 1 output per clock cycle.

5. Implementation of IFFT

Inverse Fast Fourier Transform can be calculated using the same architecture used to calculate FFT. This is done by swapping the real and imaginary part of the primary inputs and performing FFT on these values and then swapping the real and imaginary part of the outputs. The architecture now has another input called 'mode' whose value determines whether to perform FFT(mode=0) or IFFT(mode=1). In the architecture, this is done using two 2:1 multiplexers which have mode as their control signal. The real part and imaginary part of input are given to both these multiplexers. The output of the 1st mux feeds the real part of the input to the corresponding input wire of the architecture, and the output of the 2nd mux feeds the imaginary part of the input to the corresponding input wire of the architecture.

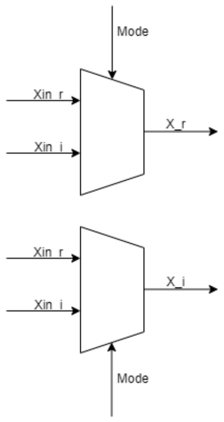

Fig : Multiplexers for mode selection at each input.

6. Pipelining the FFT/IFFT Architecture

The critical time of this architecture is $3T_M + 3T_A$. To reduce the critical time to $T_M + T_A$ we use cutset pipelining. The cutsets are shown below:

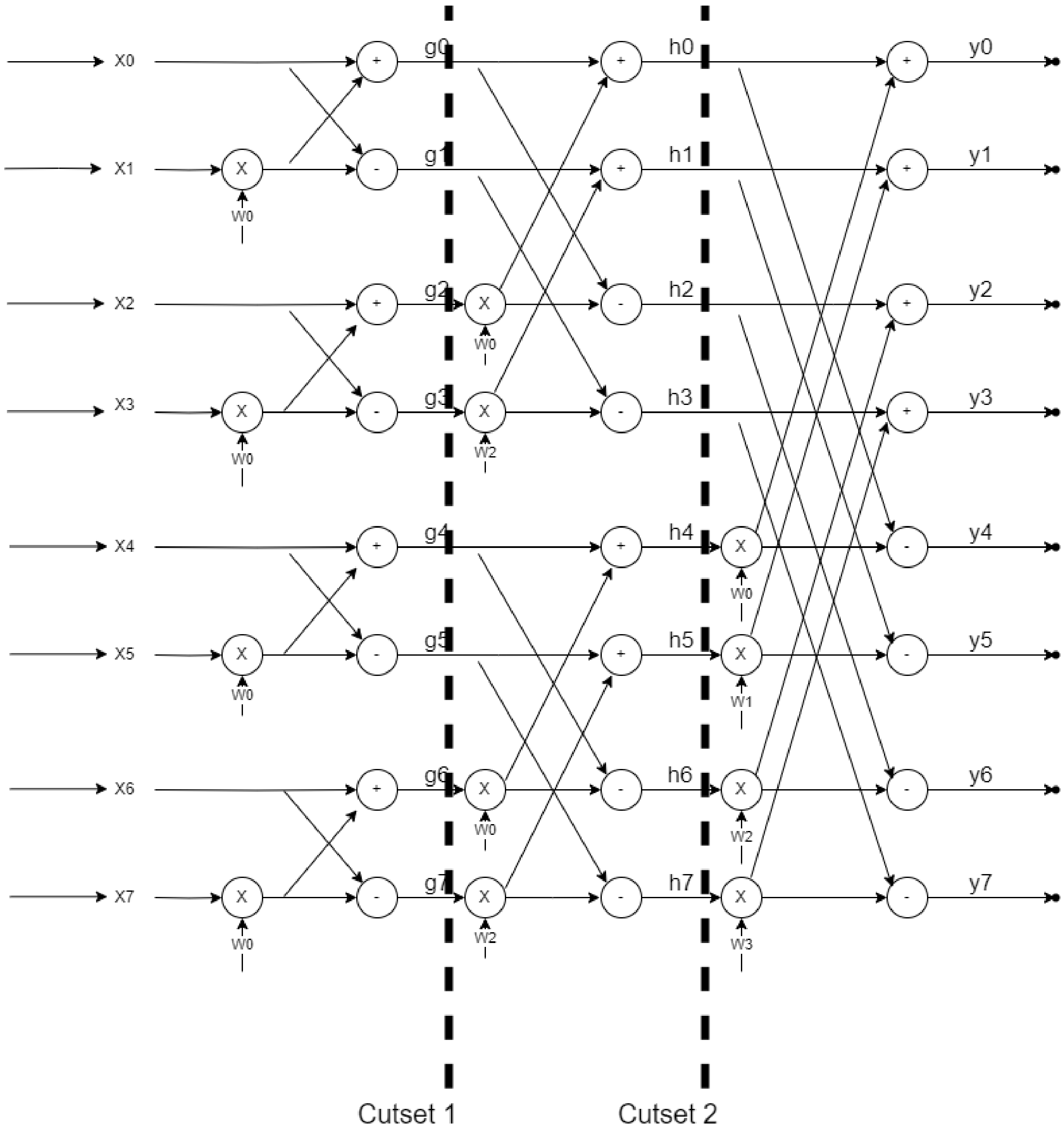

Fig : Cutsets for pipelining.

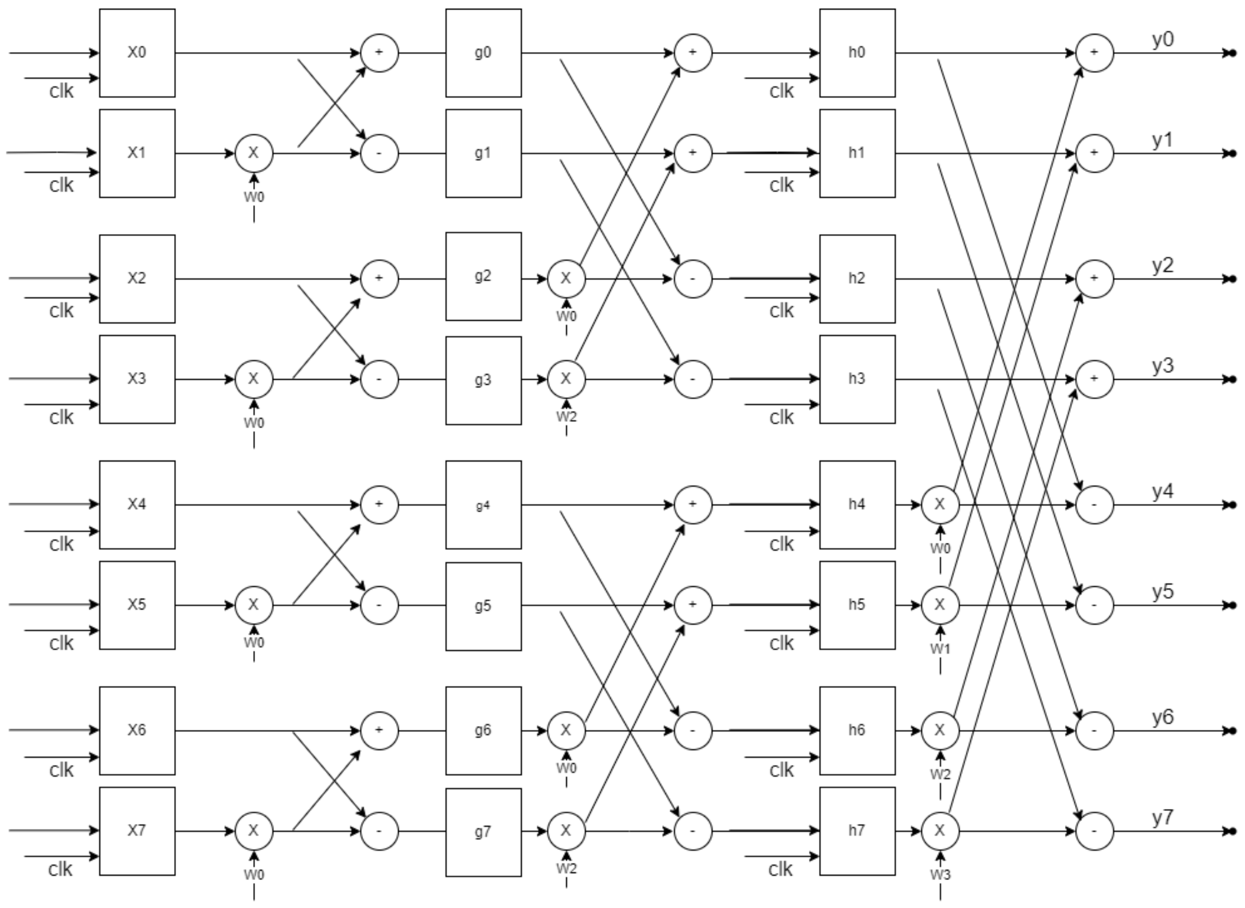

Fig : Pipelined FFT/IFFT architecture.

Registers have been added to take the inputs so that the inputs to the FFT/IFFT architecture can be given at the same time.

Edaplayground link

7. Experimental Procedure for Pipelined Architecture

The testbench designed for the initial FFT/IFFT design has been reused with the same inputs. The Matlab code for testing remains the same.

8. Experimental Results

Matlab output:(stored as columns)

```
6.5000   6.5000   6.5000   6.5000   6.5000   6.5000   6.5000   6.5000
 1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000
 1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000
 1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000
 1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000
 1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000
 1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000
 1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000   1.5000
```

Output_tracker_mag.txt:(It is stored as a single column)

```
6.5
1.5
1.5
1.5
1.5
1.5
1.5
1.5
6.5
1.500012417582929
1.5
1.500185041140405
1.5
1.500012417582929
1.5
1.500185041140405
6.5
1.5
1.5
1.5
1.5
1.5
1.5
1.5
6.5
1.500185041140405
1.5
1.500012417582929
1.5
1.500185041140405
1.5
1.500012417582929
6.500244140625
1.499755859375
1.499755859375
1.499755859375
1.499755859375
1.499755859375
1.499755859375
1.499755859375
6.500244140625
1.499667150602215
1.499755859375
1.499494507174402
1.499755859375
1.499667150602215
1.499755859375
```

```
1.499494507174402
6.500244140625
1.499755859375
1.499755859375
1.499755859375
1.499755859375
1.499755859375
1.499755859375
1.499755859375
6.500244140625
1.499494507174402
1.499755859375
1.499667150602215
1.499755859375
1.499494507174402
1.499755859375
1.499667150602215
```

The output magnitudes can be seen to be the same.

Matlab code for displaying complex FFT:

```
in=[0.5,-1,-1,-1,-1,-1,-1,-1,
   -1,0.5,-1,-1,-1,-1,-1,-1,
   -1,-1,0.5,-1,-1,-1,-1,-1,
   -1,-1,-1,0.5,-1,-1,-1,-1,
   -1,-1,-1,-1,0.5,-1,-1,-1,
   -1,-1,-1,-1,-1,0.5,-1,-1,
   -1,-1,-1,-1,-1,-1,0.5,-1,
   -1,-1,-1,-1,-1,-1,-1,0.5];
X=fft(in);
disp((X));
```

Matlab output:

```
Column 1

 -6.5000 + 0.0000i
  1.5000 + 0.0000i
  1.5000 + 0.0000i
  1.5000 + 0.0000i
  1.5000 + 0.0000i
  1.5000 + 0.0000i
  1.5000 + 0.0000i
  1.5000 + 0.0000i

 Column 2

 -6.5000 + 0.0000i
  1.0607 - 1.0607i
 0.0000 - 1.5000i
-1.0607 - 1.0607i
-1.5000 + 0.0000i
-1.0607 + 1.0607i
 0.0000 + 1.5000i
 1.0607 + 1.0607i

Column 3

-6.5000 + 0.0000i
 0.0000 - 1.5000i
-1.5000 + 0.0000i
 0.0000 + 1.5000i
 1.5000 + 0.0000i
 0.0000 - 1.5000i
-1.5000 + 0.0000i
 0.0000 + 1.5000i

Column 4

-6.5000 + 0.0000i
-1.0607 - 1.0607i
 0.0000 + 1.5000i
 1.0607 - 1.0607i
-1.5000 + 0.0000i
 1.0607 + 1.0607i
 0.0000 - 1.5000i
-1.0607 + 1.0607i

Column 5

-6.5000 + 0.0000i
-1.5000 + 0.0000i
 1.5000 + 0.0000i
-1.5000 + 0.0000i
 1.5000 + 0.0000i
-1.5000 + 0.0000i
 1.5000 + 0.0000i
-1.5000 + 0.0000i

Column 6

-6.5000 + 0.0000i
-1.0607 + 1.0607i
 0.0000 - 1.5000i
 1.0607 + 1.0607i
-1.5000 + 0.0000i
 1.0607 - 1.0607i
 0.0000 + 1.5000i
-1.0607 - 1.0607i

Column 7

-6.5000 + 0.0000i
 0.0000 + 1.5000i
-1.5000 + 0.0000i
 0.0000 - 1.5000i
 1.5000 + 0.0000i
 0.0000 + 1.5000i
-1.5000 + 0.0000i
 0.0000 - 1.5000i

Column 8

-6.5000 + 0.0000i
 1.0607 + 1.0607i
 0.0000 + 1.5000i
-1.0607 + 1.0607i
-1.5000 + 0.0000i
-1.0607 - 1.0607i
 0.0000 - 1.5000i
 1.0607 - 1.0607i
```

Real part of output from edaplayground:

```
-6.5
1.5
1.5
1.5
1.5
1.5
1.5
1.5
-6.5
1.060546875
0
-1.060791015625
-1.5
-1.060546875
0
1.060791015625
-6.5
0
-1.5
0
1.5
0
-1.5
```

```
0
-6.5
-1.060791015625
0
1.060546875
-1.5
1.060791015625
0
-1.060546875
-6.500244140625
-1.499755859375
1.499755859375
-1.499755859375
1.499755859375
-1.499755859375
1.499755859375
-1.499755859375
-6.500244140625
-1.060546875
0
1.060302734375
-1.499755859375
1.060546875
0
-1.060302734375
-6.500244140625
0
-1.499755859375
0
1.499755859375
0
-1.499755859375
0
-6.500244140625
1.060302734375
0
-1.060546875
-1.499755859375
-1.060302734375
0
1.060546875
```

Imaginary part of output from edaplayground:

```
0
0
0
0
0
0
```

```
0
0
0
0
-1.060791015625
-1.5
-1.060791015625
0
1.060791015625
1.5
1.060791015625
0
-1.5
0
1.5
0
-1.5
0
1.5
0
-1.060791015625
1.5
-1.060791015625
0
1.060791015625
-1.5
1.060791015625
0
0
0
0
0
0
0
0
0
1.060302734375
-1.499755859375
1.060302734375
0
-1.060302734375
1.499755859375
-1.060302734375
0
1.499755859375
0
-1.499755859375
0
1.499755859375
```

0
-1.499755859375
0
1.060302734375
1.499755859375
1.060302734375
0
-1.060302734375
-1.499755859375
-1.060302734375

Outputs are the same

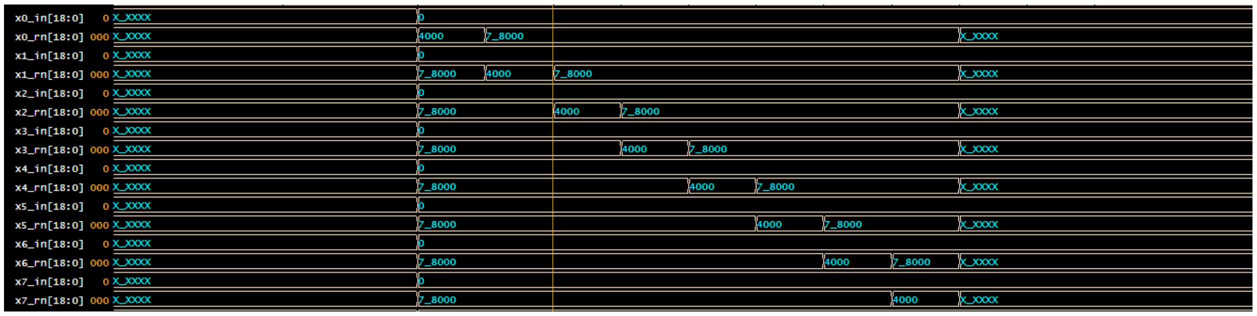

Fig : Inputs given to FFT/IFFT architecture(1.16 fixed point), a total of 64 different inputs given 8 at a time

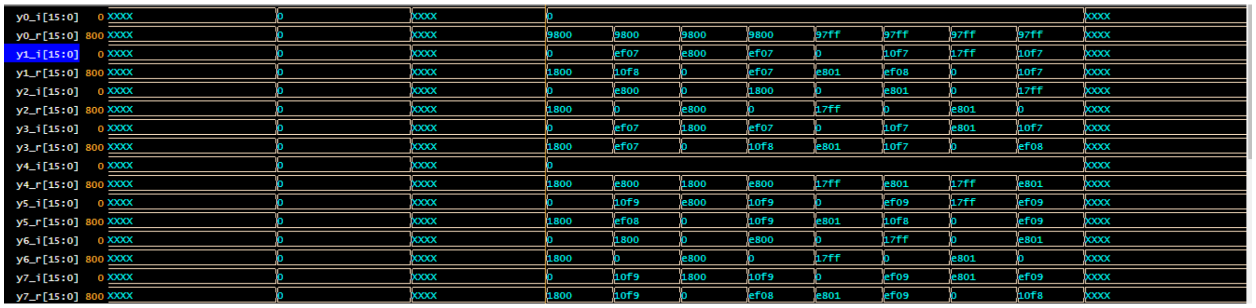

Fig: Output values(both real and imaginary parts)

The latency is seen to be 3 clock cycles. The latency would be 2 if the registers which were added at the input side to ensure that the inputs are given at the same time to the architecture had been removed. The throughput of the pipelined architecture is 1 output per clock cycle.

9. Conclusion

The FFT/IFFT architecture has been implemented in Verilog. The high critical time of the architecture has been reduced using cutset pipelining and the pipelined architecture has been implemented in Verilog. Bit growth has been observed and 4.12 fixed point representation was decided to use to accommodate this.