256 FFT core for Image Processing in Verilog

Group: 12

Ashwin Gopinath - 2018102015
Haripraveen S - 2018102031
Shreya Malkurthi - 2019122001

Discrete Time Fourier Transform - DTFT

- Like continuous time signal Fourier transform, discrete time Fourier Transform (DTFT) can be used to represent a discrete sequence into its equivalent frequency domain representation and LTI discrete time system.

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n}.$$

- Sample the spectrum X(ω) in frequency obtain DFT

$$X(k) = X(k\Delta\omega), \quad \Delta\omega = \frac{2\pi}{N}$$

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi\frac{kn}{N}}$$

Fast Fourier Transform - FFT

- FFT is a numerically efficient way of performing DFT and thus requires lesser computations
- So much of the modern technology that we have today like Wireless Communication, GPS and anything related to the vast field of Signal Processing rely on the insights of FFT.
- FFTs were initially developed by Cooley and Tukey though the initial contributions were made by Gauss as early as 1805.
- The Cooley-Tukey algorithm is the most commonly used algorithm to find FFT.
    - A good example of Divide and Conquer
- A discrete Fourier transform can be computed using an FFT if the number of points is a power of two.
  For example N = 8,32,128,256 etc

Fast Fourier Transform - FFT

- The FFT reduces the number of computations needed for a problem of size N from $O(N^2)$ to $O(N\log_2 N)$.

N	1000	10^6	10^9
N^2	10^6	10^{12}	10^{18}
$Nlog_2N$	10^4	$20\,x\,10^6$	$30\,x\,10^9$

- It does so by exploiting the symmetries of twiddle factors, W_N terms. It is based on the principle of decomposing the computation of sequence of length N into successively smaller DFT.
  - Complex conjugate symmetry

$$W_N^{k(N-n)} = W_N^{-kn} = (W_N^{-kn})^*$$

  - Periodicity in n,k

$$W_N^{kn} = W_N^{k(N+n)} = W_N^{(k+N)n}$$

Fast Fourier Transform - FFT

- FFT algorithm greatly increases the speed of computation of DFT.
- Computational Complexity
  - The number of multiplications required to compute N-point DFT using radix-2 FFT are $N\log_2 N$ as compared to N^2 in normal DFT.
- Consider N = 64
  - Number of complex multiplications required using normal computation are $N^2 = (64)^2 = 4096$ whereas those required if FFT is used are 192 only!
  - Speed improvement factor = 4096/192 = 21.33

FFT Algorithms:

- There are basically two types of FFT algorithms:
  - Decimation in time
  - Decimation in frequency
- In our implementation, we used Decimation in frequency

Decimation in Frequency

- This is obtained by using divide and conquer approach.
- In this the output sequence X(k) is divided into smaller and smaller subsequences, thus the name decimation in frequency.
- Initially the input sequence is divided into two sequences $x_1(n)$ and $x_2(n)$ containing first N/2 samples and last N/2 samples respectively.
  - Former N/2 - x(n) where $0 \le n \le N/2$
  - Latter N/2 - x(n + N/2) where where $0 \le n \le N/2$

$$X(k) = \sum_{n=0}^{(N/2)-1} x(n) W_N^{kn} + \sum_{n=N/2}^{N-1} x(n) W_N^{kn}$$

$$= \sum_{n=0}^{(N/2)-1} x(n) W_N^{kn} + W_N^{Nk/2} \sum_{n=0}^{(N/2)-1} x\left(n + \frac{N}{2}\right) W_N^{kn}$$

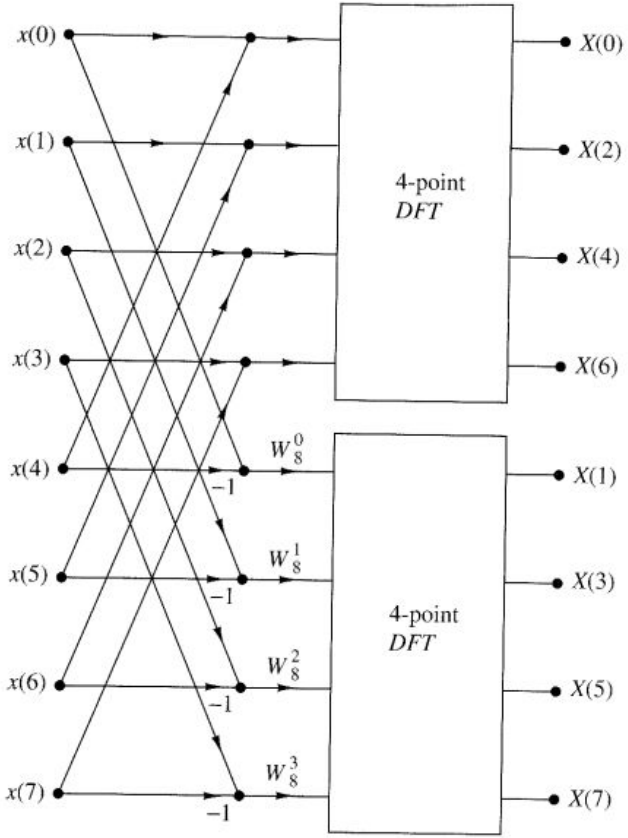
Decimation in Frequency

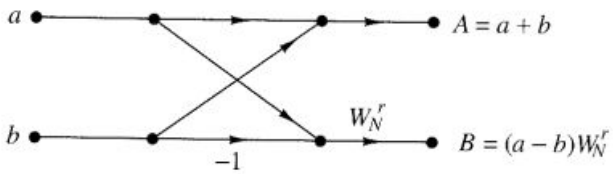
Separating even and odd samples:

$$
\begin{aligned}
X(2r) &= \sum_{n=0}^{N-1} x(n) W_N^{2rn} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{2rn} + \sum_{n=0}^{\frac{N}{2}-1} x\left(n+\frac{N}{2}\right) W_N^{2r\left(n+\frac{N}{2}\right)} \\
&= \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{2rn} + \sum_{n=0}^{\frac{N}{2}-1} x\left(n+\frac{N}{2}\right) W_N^{2rn} 1 \\
&= \sum_{n=0}^{\frac{N}{2}-1} \left(x(n) + x\left(n+\frac{N}{2}\right)\right) W_{\frac{N}{2}}^{rn} \\
&= \mathrm{DFT}_{\frac{N}{2}} \left[x(n) + x\left(n+\frac{N}{2}\right)\right]
\end{aligned}
$$

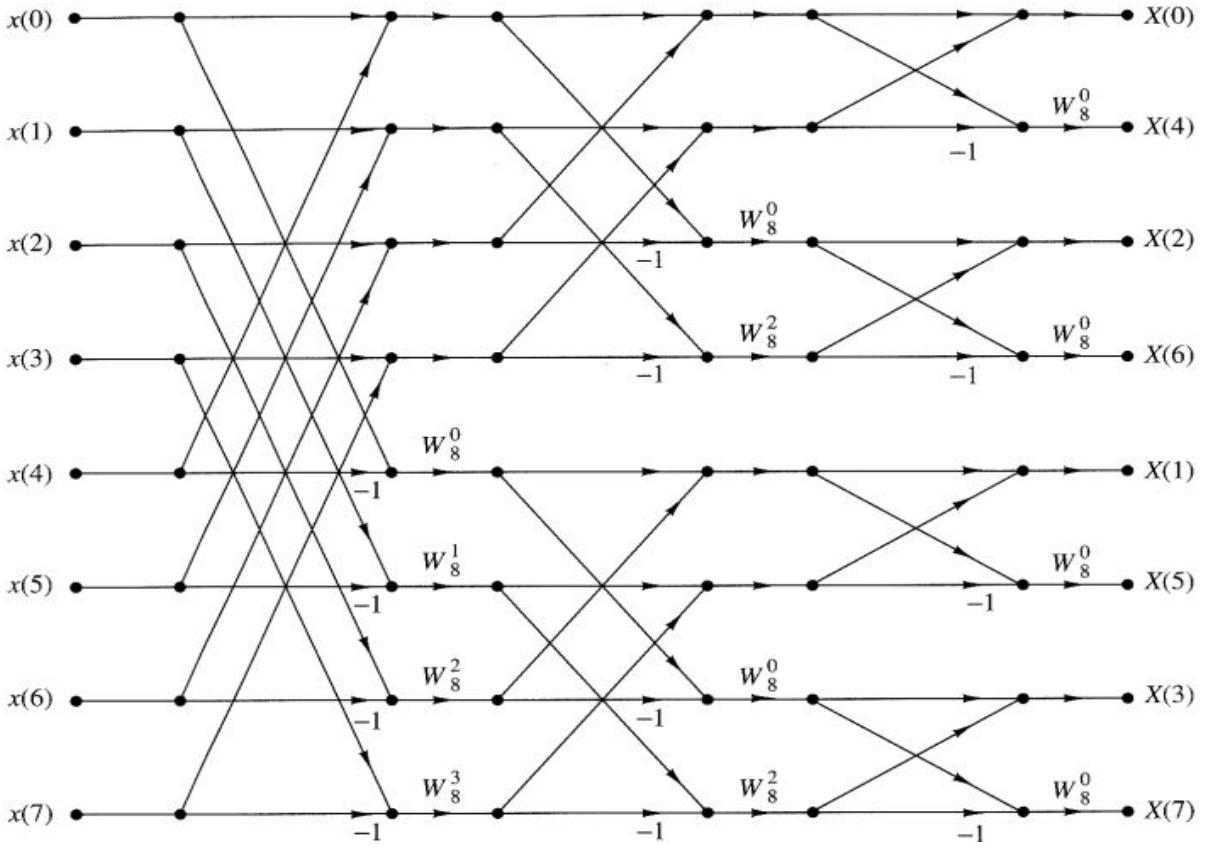
$$
\begin{aligned}
X(2r+1) &= \sum_{n=0}^{N-1} x(n) W_N^{(2r+1)n} \\
&= \sum_{n=0}^{\frac{N}{2}-1} \left(x(n) + W_N^{\frac{N}{2}} x\left(n+\frac{N}{2}\right)\right) W_N^{(2r+1)n} \\
&= \sum_{n=0}^{\frac{N}{2}-1} \left(\left(x(n) - x\left(n+\frac{N}{2}\right)\right) W_N^{n}\right) W_{\frac{N}{2}}^{rn} \\
&= \mathrm{DFT}_{\frac{N}{2}} \left[\left(x(n) - x\left(n+\frac{N}{2}\right)\right) W_N^{n}\right]
\end{aligned}
$$

First stage of decimation in frequency N = 8 point FFT algorithm





Basic butterfly computation in decimation in frequency FFT algorithm.



N = 8 point decimation in frequency FFT algorithm

Implementing FFT in verilog

Research paper

We have based this implementation on the following paper on single-path delay feedback pipelined FFT.

"Analysis and implementation of SDF Radix-2 FFT processor using VERILOG Hardware Description Language"

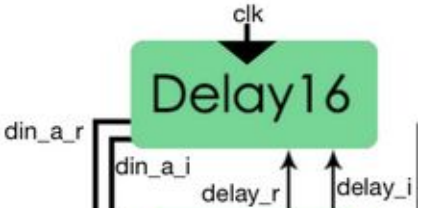
Phuong H. Lai[1], Manh Hoang[2], Viet Q. Tran[2], Tung V. Nguyen[2], Thien V. Truong[2], and Phong H. Nguyen

- Features:
  - FFT length = 32 and 256
  - FFT algorithm used: Decimation in Frequency
  - Processor Architecture: Single-path delay-feedback (SDF) pipeline processor architechture

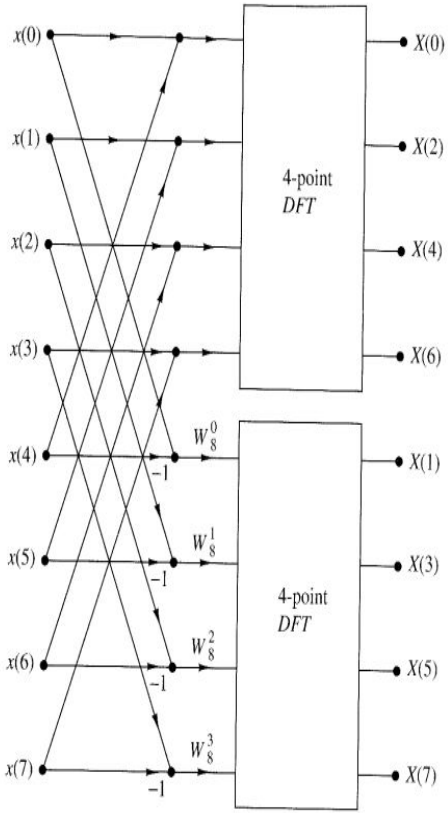
How our Algorithm works?

- We have used single-path delay feedback pipeline FFT processor.
- We have 4 blocks namely radix-2 block , delay block, ROM block , sort module.
- The radix, delay and ROMare repeated logN times and are used to calculate different lengths .
- The sorting block finally rearranges all the outputs. Since this is decimation in frequency, the bit-reversal permutation is required at output of FFT module.

Delay module



- The delay module is simply a FIFO shift register. It will shift 24 bits every cycle.
- The delay module has 2 functions. First is to load the input while we are in waiting state and the other is to multiply the twiddle factors stored in ROM with the subtracted input.
- First function : In waiting state, we cannot do any calculation.
- Second function: Here we multiply the delay signal, which is the signal we output to the delay module in the first half state, with our twiddle factor.



Radix-2 butterfly module

While entering the first radix-2 butterfly, din is extended to 24bits to match the twiddle factor. The radix-2 butterfly is the core processor of this circuit. It contains three states :

- Waiting: In waiting state, we cannot do any calculation. For instance, we have to wait for x[16] in the first state to do x[0]+x[16], so x[0]~x[15] will be in waiting state.
- First Half: In the first half, the output will be the summation of two index, e.g. x[0]+x[16]. We will output x[0]-x[16] to delay module simultaneously.
- Second Half: In the last state, we multiply the delay signal, which is the signal we output to the delay module in the first half state, with our twiddle factor. Same as above, the input (din_a_r and din_b_r) will be output to the delay module.

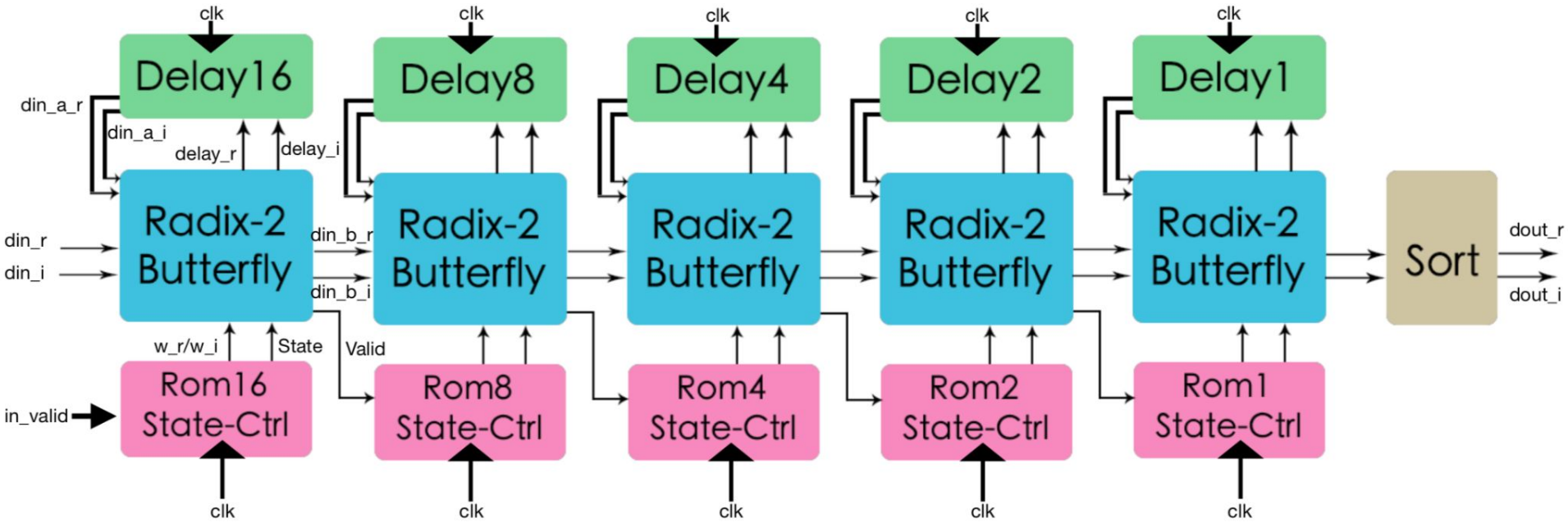
ROM and state control Module

- The ROM is where the twiddle factors are stored. When it receives the valid signal from the front stage, it sets a counter.
- Based on the counter, it will output a state control output signal to the radix2 butterfly module.
- For the second half state where multiplication takes place, it will provide the needed twiddle factor.

Sort module

- Since this is decimation by frequency the output will come in scrambled order we have to correct it.
- Since we know the output signal order, we can simply control the order by directing the signal to a 2D array and place the value in the right place.
- For instance, the 2nd output is X[15], we can store it into result[15].
- It will take 32 cycles to sort. To add on, the input of sort module is the most significant 16 bits of the output of the last radix-2 butterfly.

Overall algorithm



Based on DIF radix-2 butterfly FFT algorithm, the suitable

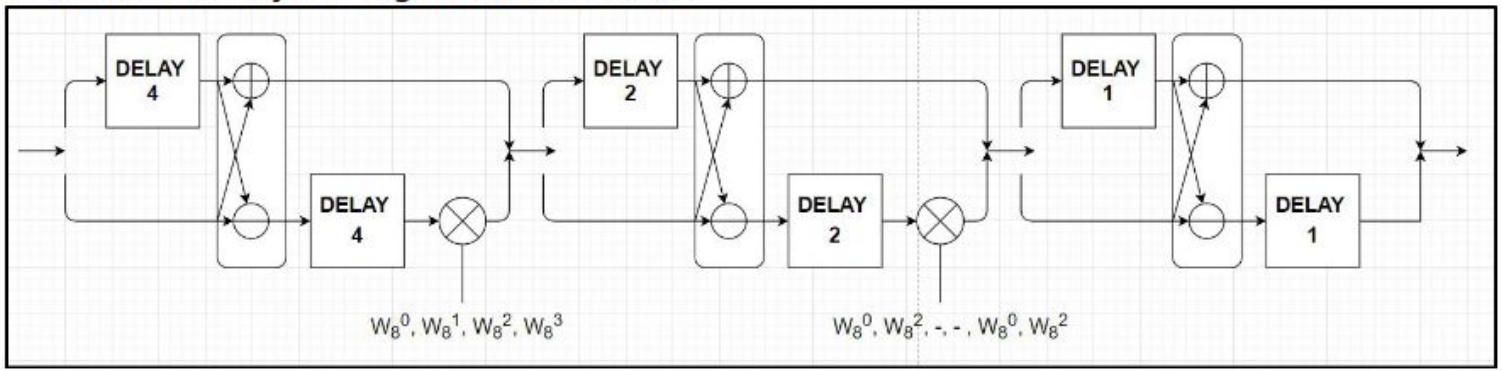
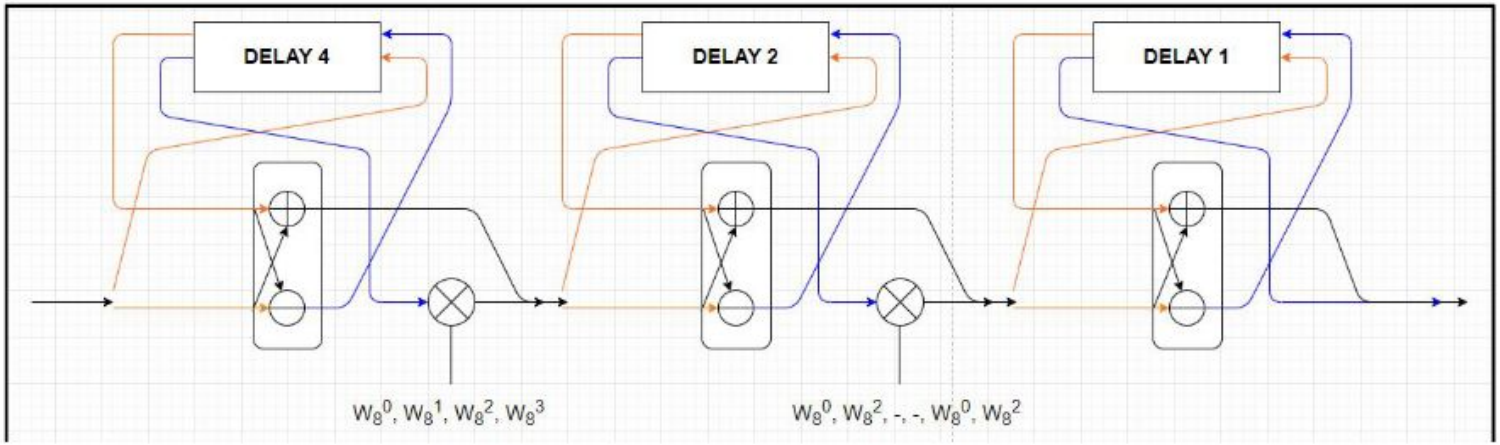
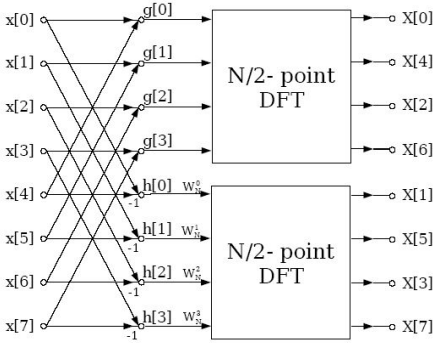


Figure 6: System model for simple 8-points **FFT** processor.





Overall algorithm

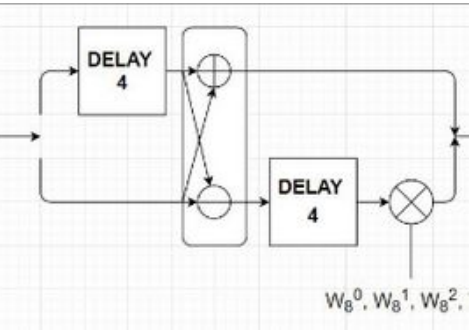
We give the input serially at negative clock edges. The delay block now loads all the 128 inputs. Then it changes state and calculates sum and difference of the in[128],in[129]... and in[0],in[1]...

It sets the input as valid for the next block and sends the sum and sends the difference to the delay to be multiplied with the twiddle factors.

This delayed difference comes after 128 clock cycles and then is multiplied with the twiddle factor that is stored in the ROM.

This process is carried out by each step of the algorithm where each step calculated the corresponding factor of 2

Overall algorithm

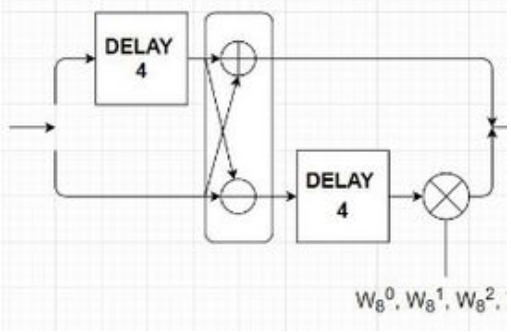


The first Radix-2 module runs only once as in[0] is added and subtracted with in[129], in[1] with in[130].. and so forth.

But the next Radix-2 module runs twice (And 2X as we go further every stage) as we need to compute FFT of two sets of 128 inputs from the previous Radix module and we again send forth the sums and differences computed in the current Radix module.

Note that computations in this module starts only in the 128 + 65 = 193rd cycle, as the delay block of the second radix module waits to receive 64 inputs, and starts computing addition and subtractions only once the 65th input is received.

Similar to the previous block, here too we send the sum; and the difference to be multiplied by appropriate twiddle factors to the delay of the next block.

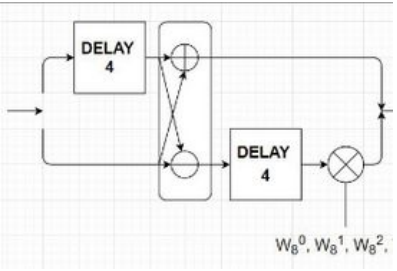


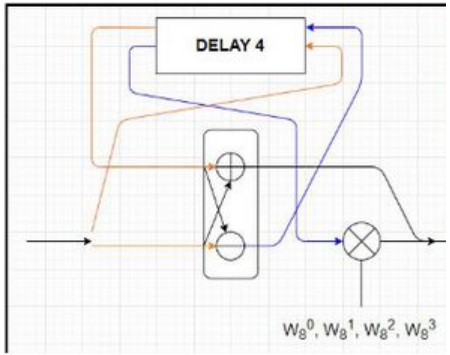
Overall algorithm

For 32 point FFT, we need five stages and for 256 point FFT we need 8 stages. Each stage consists of a Radix-2 module. Only difference comes in the delay and ram blocks in each stage. For a **simple pipeline FFT processor,** in each stage there are two delay modules and one multiplication with twiddle value.

Taking example of the 32 point FFT, a delay-16 module is used to set 16 signal go to the upper input of Radix-2 module and remaining 16 inputs in lower input of Radix-2. The upper output of Radix-2 will directly go to next stage, in contrast the lower outputs will come to another delay-4 module and take the multiplication with corresponding twiddle values which was stored in ROM.

Overall algorithm



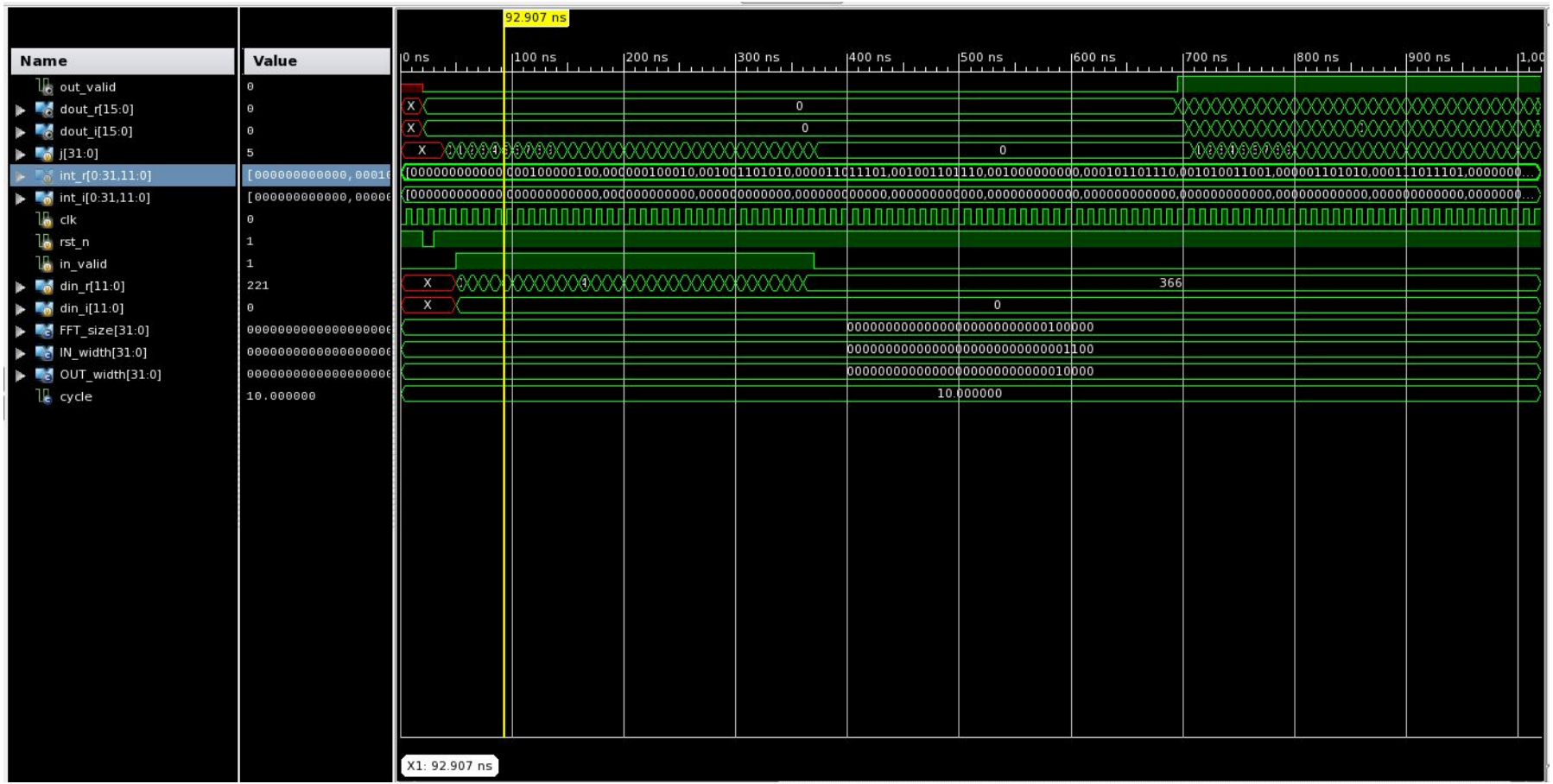


The further stages of FFT work similarly, except for the fact that we use smaller delays like delay-8, delay-4 and so on. Finally we obtain the output values in frequency domain, but in permutation order (Because we used DIF). We use a sort module to correct the order.

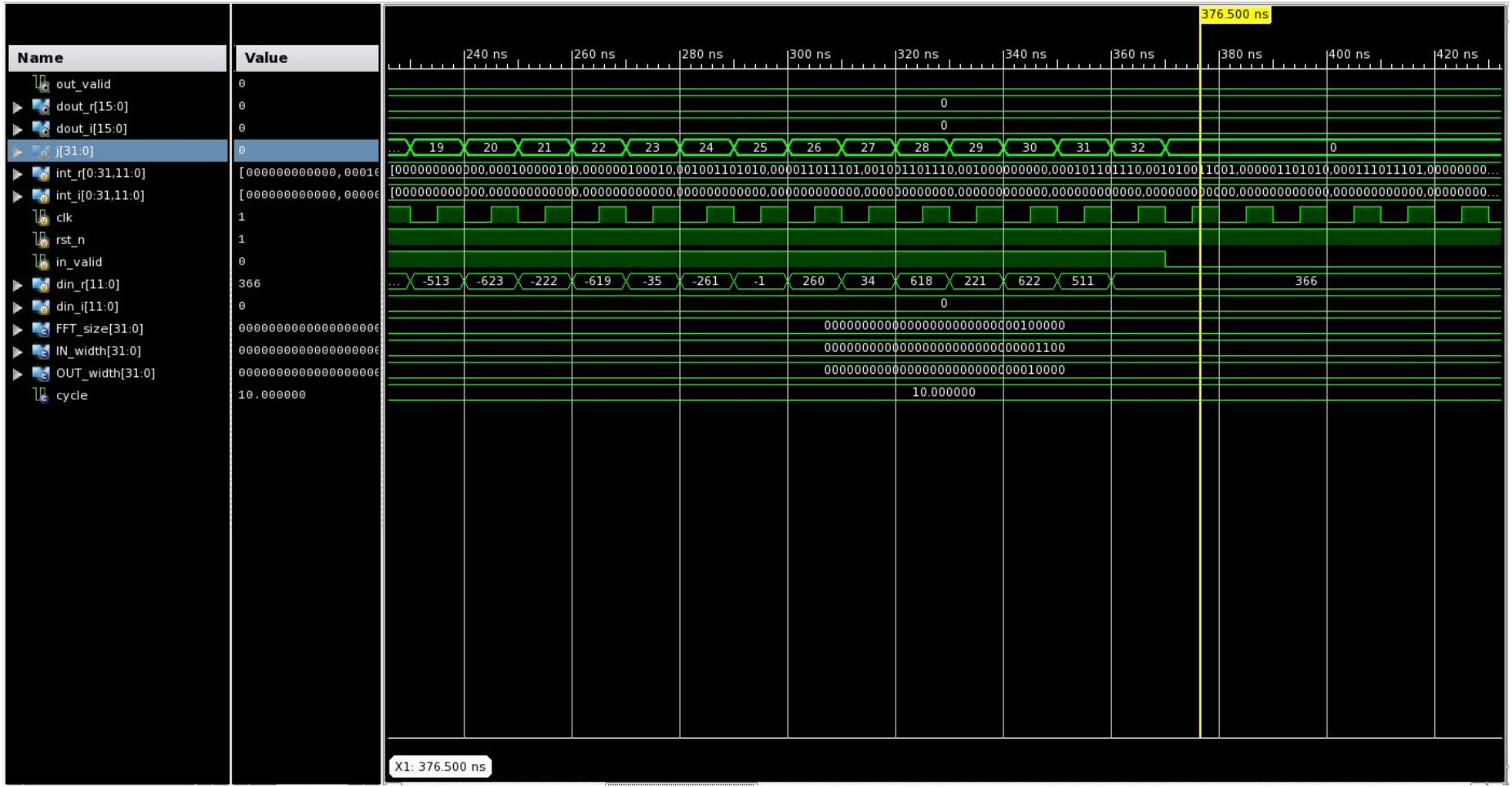
In order to save resources, instead of the two delay system for each stage, we use a one delay module with the help of delay feedback. This is called the single-path delay feedback pipeline FFT processor.

Results and Analysis

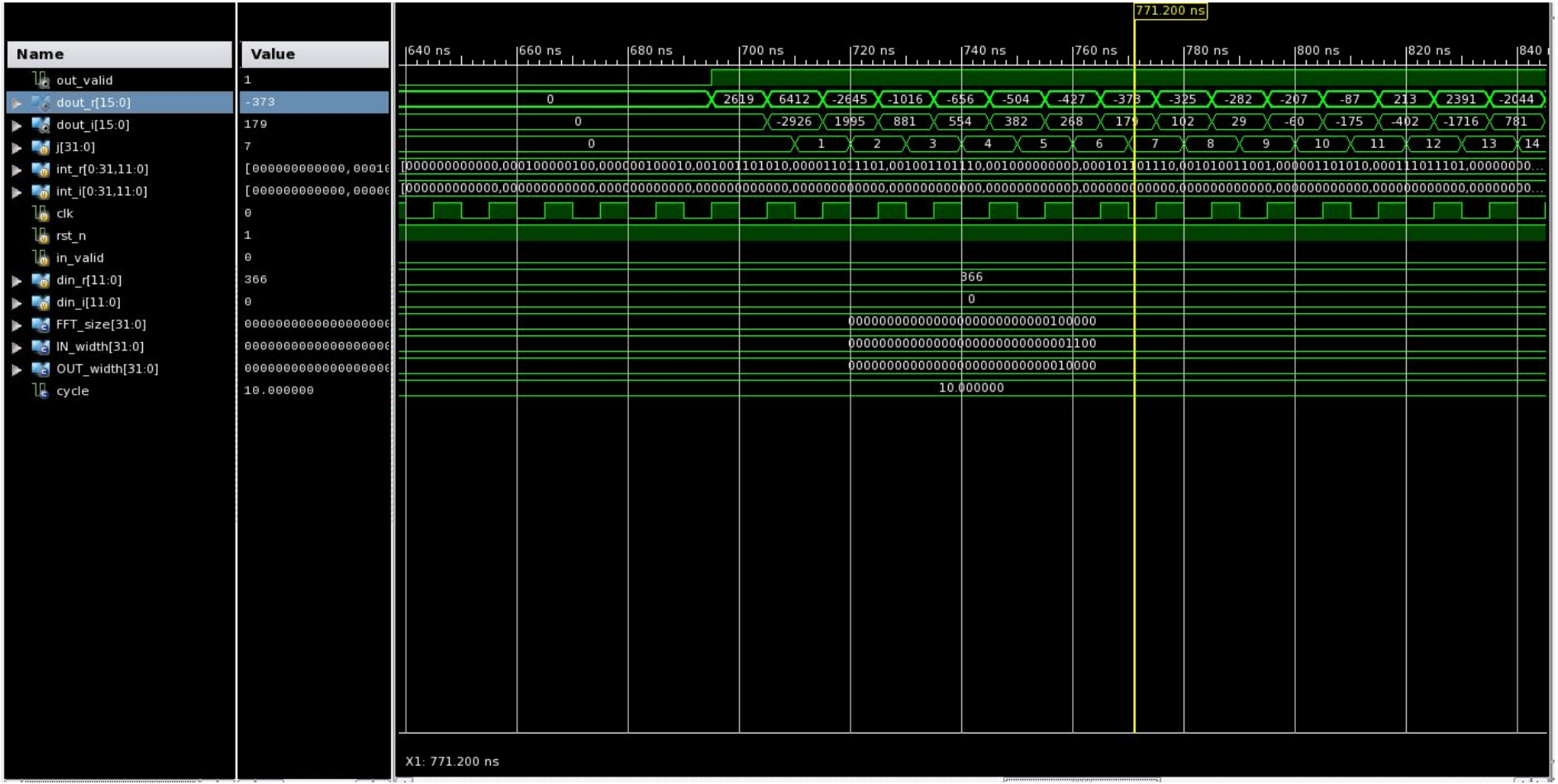
Output of the complete simulation for 32 FFT



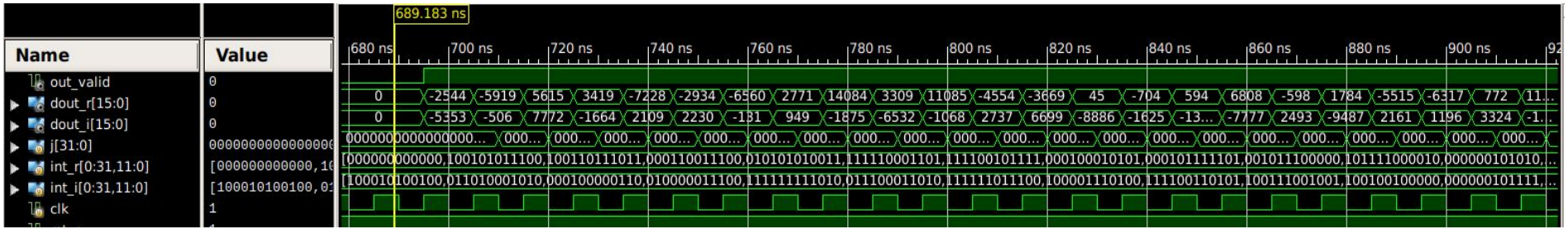
Loading Phase for 32 FFT



Output Phase for 32 FFT



Comparing results from our code with online tool



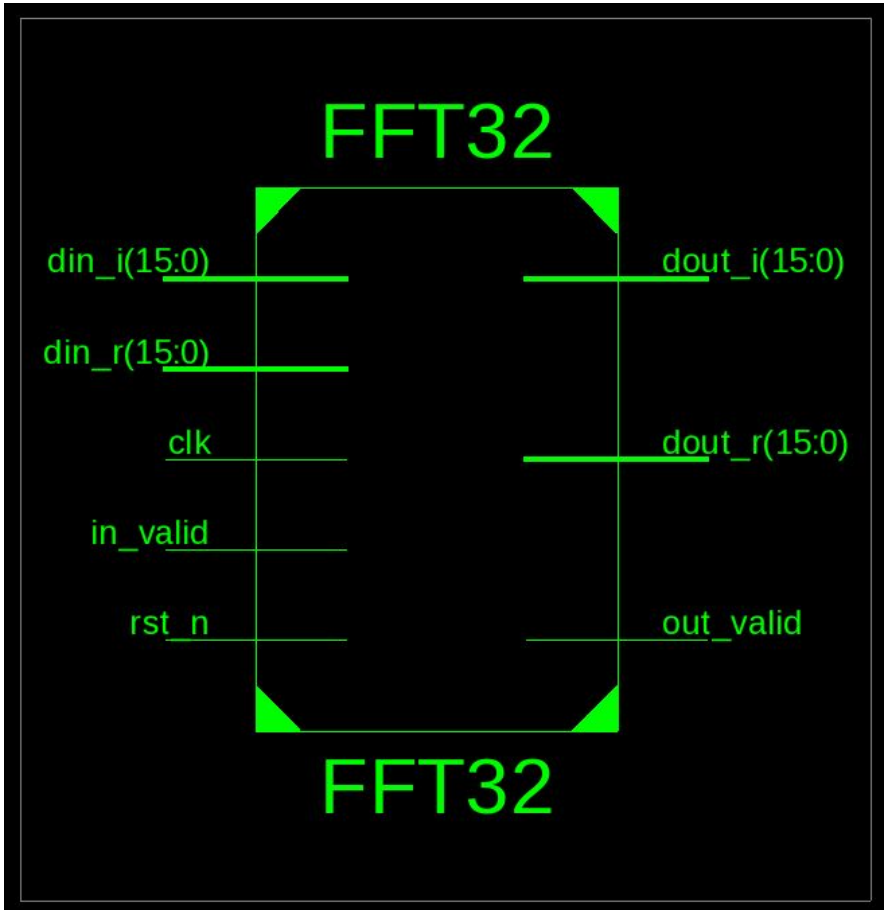
-2544.000000,-5353.000000
-5910.183418,-516.529404
5614.581902,7755.270135
3413.802375,-1661.005573
-7227.376767,2109.790981
-2932.550750,2218.902292
-6550.902187,-150.209627
2767.691386,948.416873
14084.000000,-1875.000000
3301.062390,-6524.053992
11075.139564,-1068.576929

-4548.429407,2729.322718
3668.096829,6700.639313
43.715827,-8872.559691
-706.020618,-1622.089366
596.212179,-13330.544300
6808.000000,-7777.000000
-590.529453,2489.560824
1785.171781,-9469.321359
-5504.882075,2161.927227
-6316.623233,1196.209019
775.997601,3308.722230
1102.813008,-18956.820949

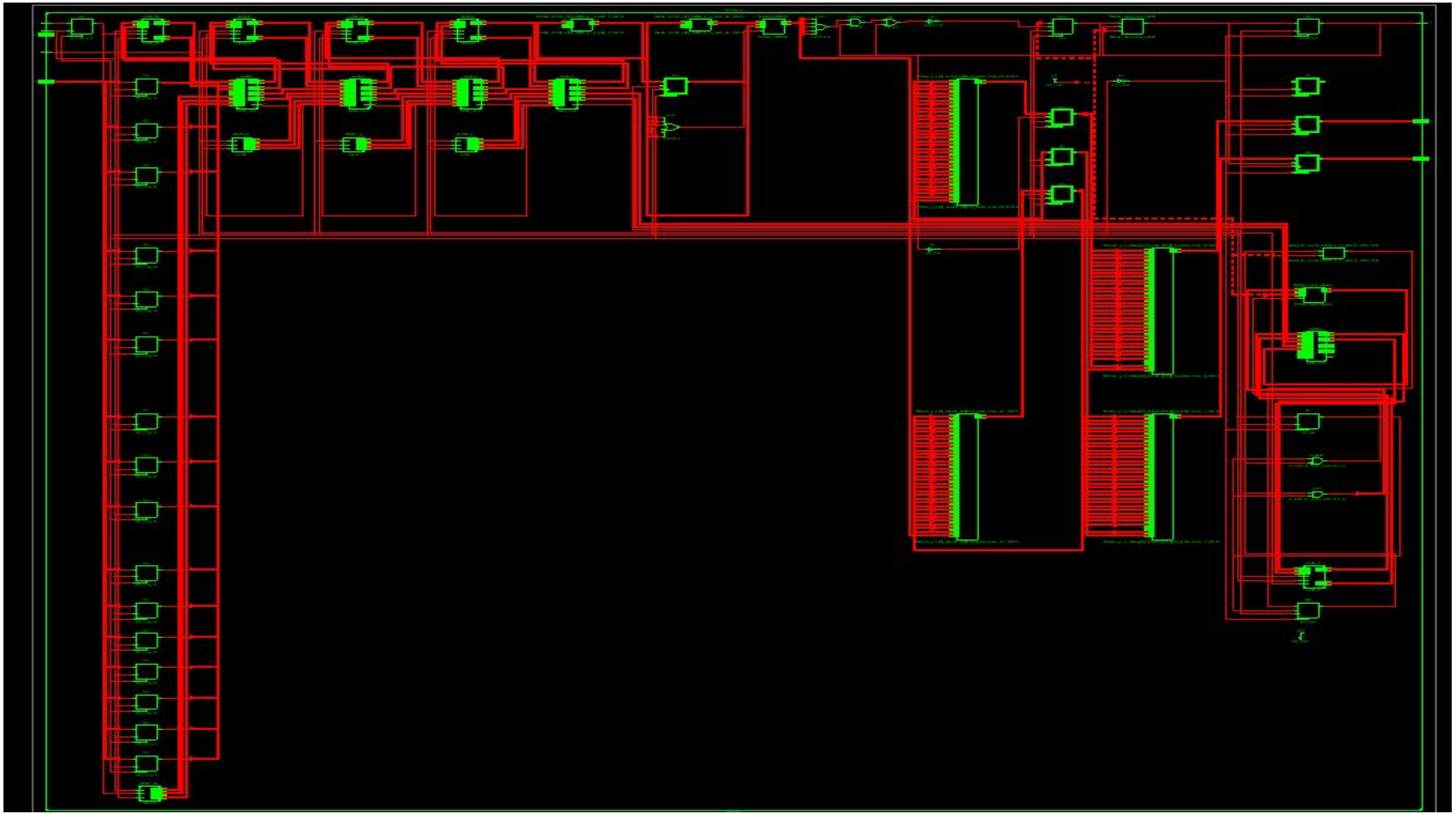
[Link to the online fft calculator]

-

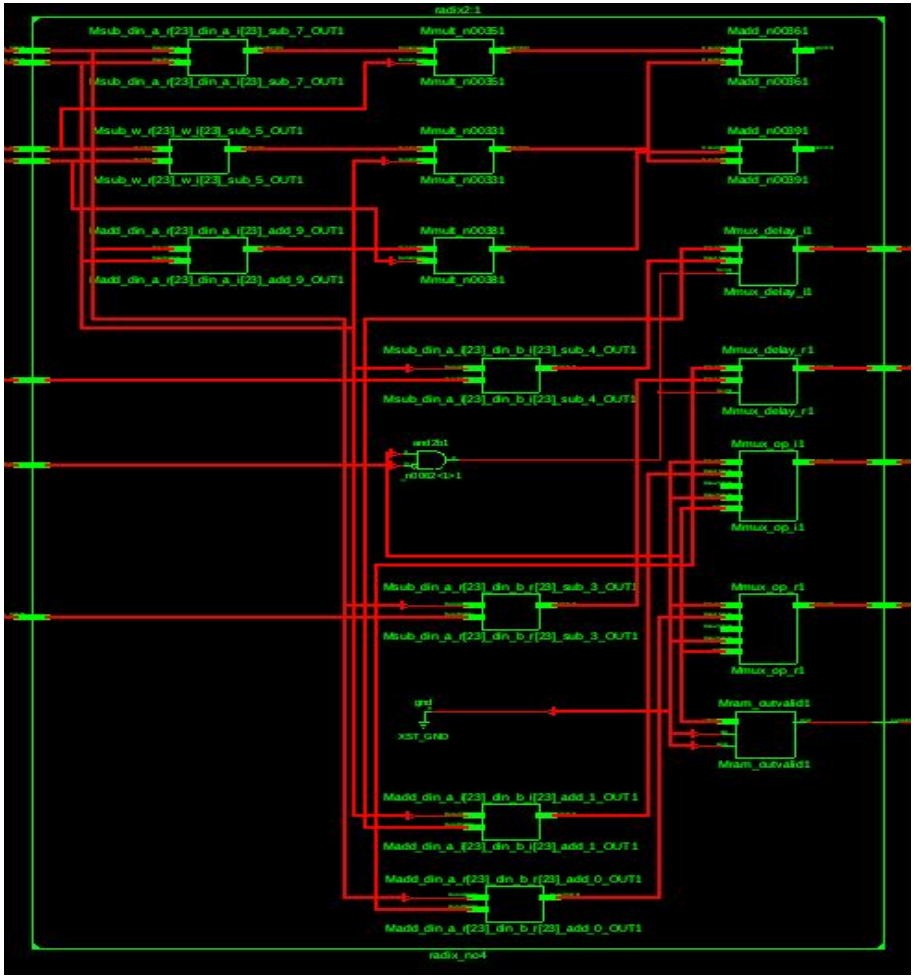
RTL Schematic of 32 FFT



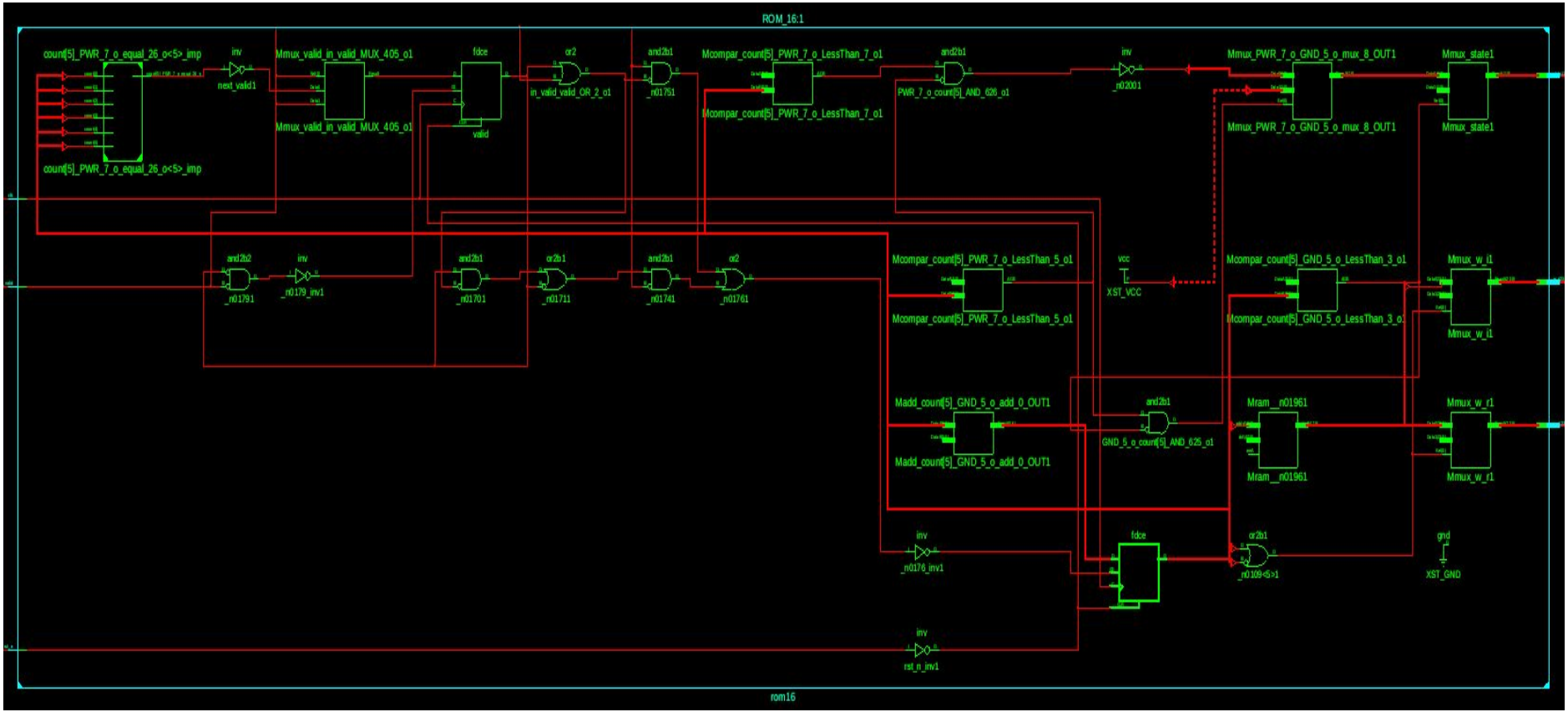
Complete RTL Schematic of 32 FFT



Schematic of the Radix-2 butterfly module in FFT



Schematic of ROM-16 module used in 32 FFT



Timing Details of 32 FFT

```
==========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 41.856ns (frequency: 23.891MHz)
  Total number of paths / destination ports: 25077439511543 / 3676
--------------------------------------------------------------------------
Delay:               41.856ns (Levels of Logic = 48)
  Source:            rom16/count_0 (FF)
  Destination:       result_r_31_15 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising
```

Device utilization Summary of 32 FFT

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	1,727	33,280	5%	
Number used as Flip Flops	1,723			
Number used as Latches	4			
Number of 4 input LUTs	2,384	33,280	7%	
Number of occupied Slices	1,864	16,640	11%	
Number of Slices containing only related logic	1,864	1,864	100%	
Number of Slices containing unrelated logic	0	1,864	0%	
Total Number of 4 input LUTs	2,384	33,280	7%	
Number used as logic	2,288			
Number used as Shift registers	96			
Number of bonded IOBs	60	519	11%	
IOB Flip Flops	58			
Number of BUFGMUXs	1	24	4%	
Number of DSP48As	41	84	48%	
Average Fanout of Non-Clock Nets	2.65			

Analysis of power distribution from XPower Analyser

Device	
Family	Spartan3adsp
Part	xc3sd1800a
Package	fg676
Temp Grade	Commercial
Process	Typical
Speed Grade	-4

Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	0

Characterization	
PRODUCTION	v1.1,06-26-09

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.001	1	---	---
Logic	0.000	2382	33280	7
Signals	0.000	5785	---	---
DSPs	0.000	41	84	49
IOs	0.000	60	519	12
Leakage	0.114			
Total	0.115			

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	15.9	83.2	26.8

Supply Source	Summary Voltage	Total Current (A)	Dynamic Current (A)	Quiescent Current (A)
Vccint	1.200	0.043	0.001	0.042
Vccaux	2.500	0.025	0.000	0.025
Vcco25	2.500	0.000	0.000	0.000

Supply	Power (W)	Total	Dynamic	Quiescent
		0.115	0.001	0.114

Power Summary from the Synthesis Report

2. Summary

2.1. On-Chip Power Summary

On-Chip Power Summary

On-Chip	Power (mW)	Used	Available	Utilization (%)
Clocks	0.88	1	---	---
Logic	0.00	2382	33280	7
Signals	0.00	5785	---	---
IOs	0.00	60	519	12
DSPs	0.00	41	84	49
Quiescent	114.09			
Total	114.97			

2.2. Thermal Summary

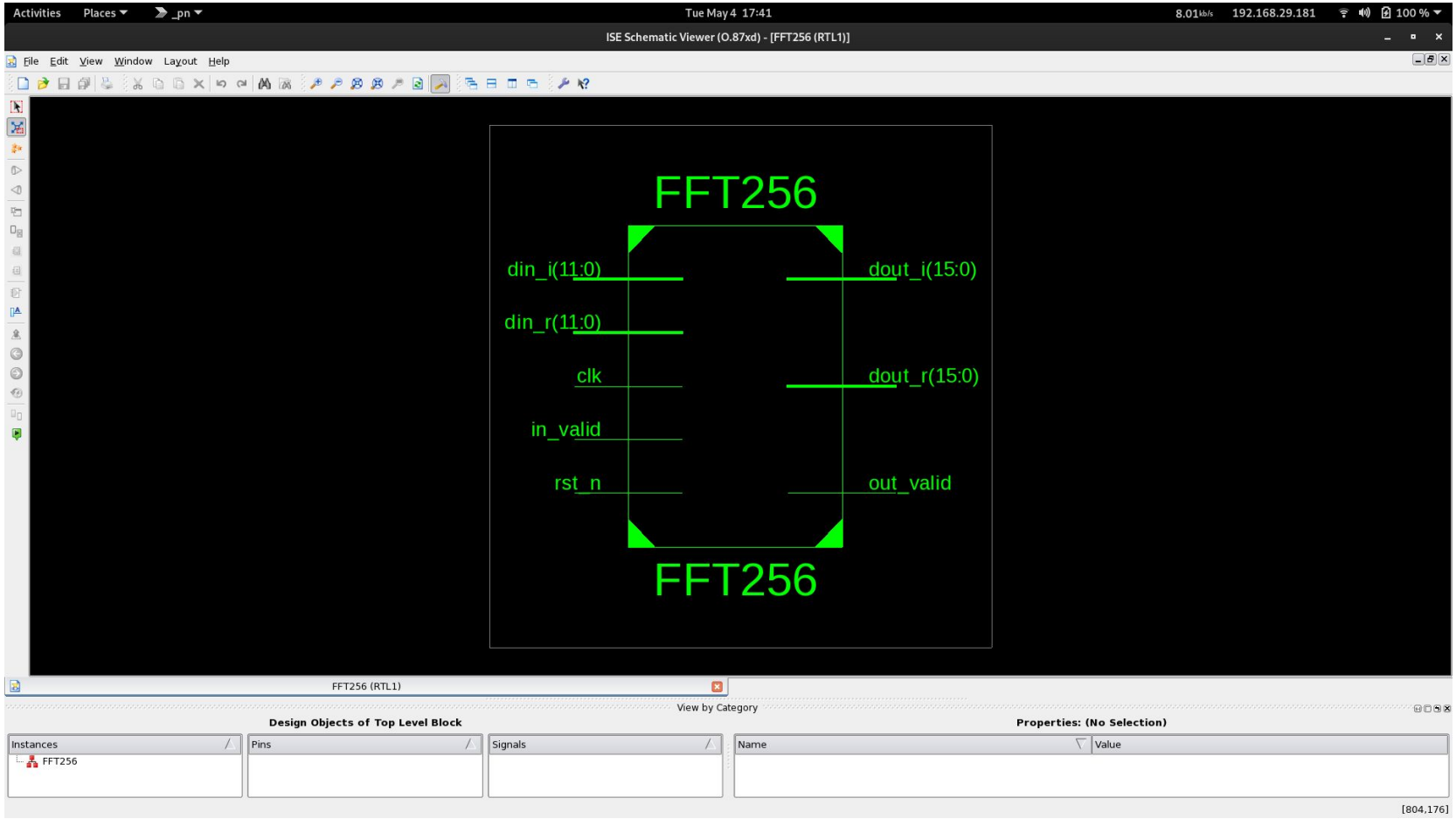
Thermal Summary	
Effective TJA (C/W)	15.9
Max Ambient (C)	83.2
Junction Temp (C)	26.8

2.3. Power Supply Summary

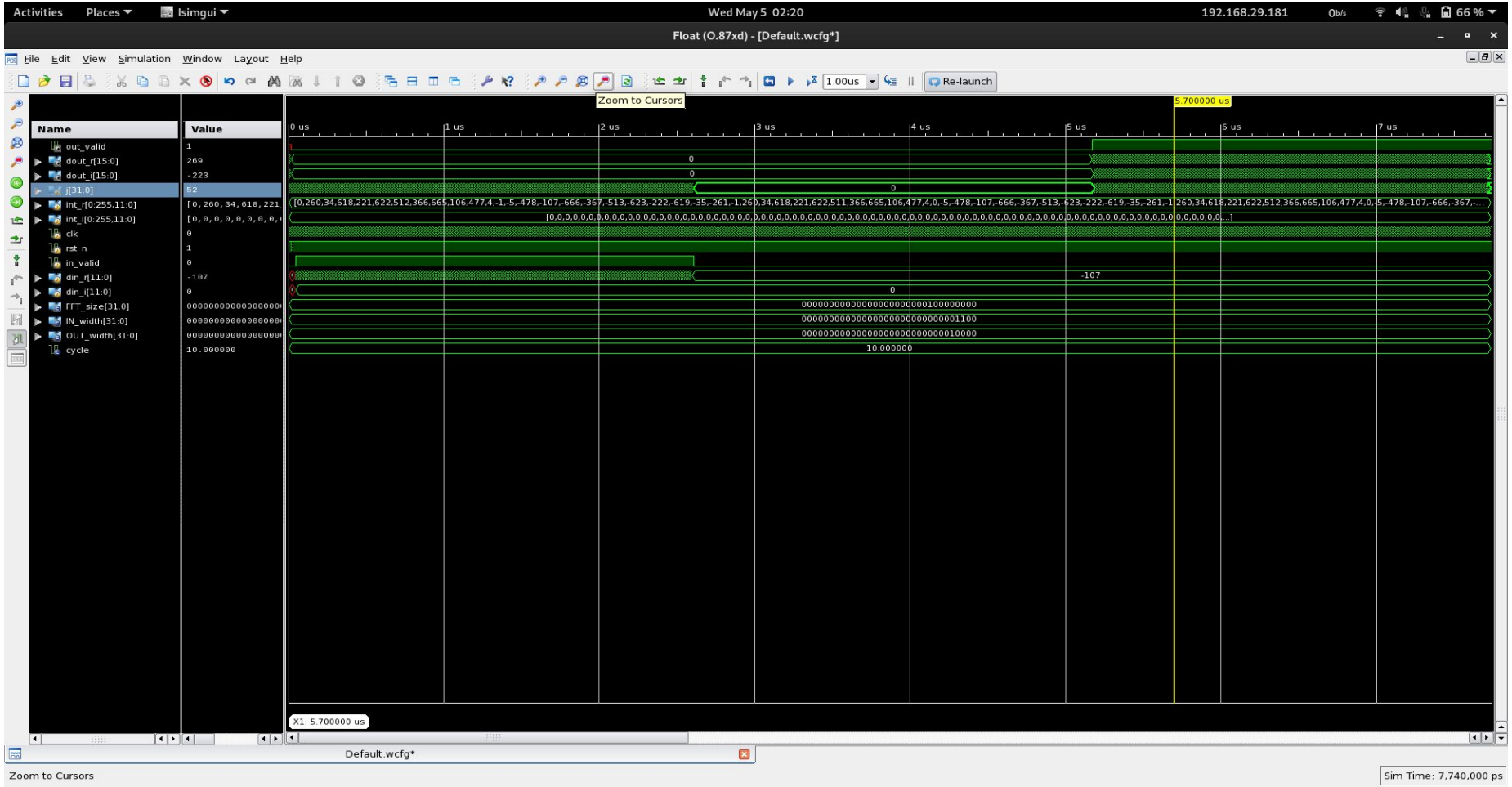
Power Supply Summary

	Total	Dynamic	Quiescent
Supply Power (mW)	114.97	0.88	114.09

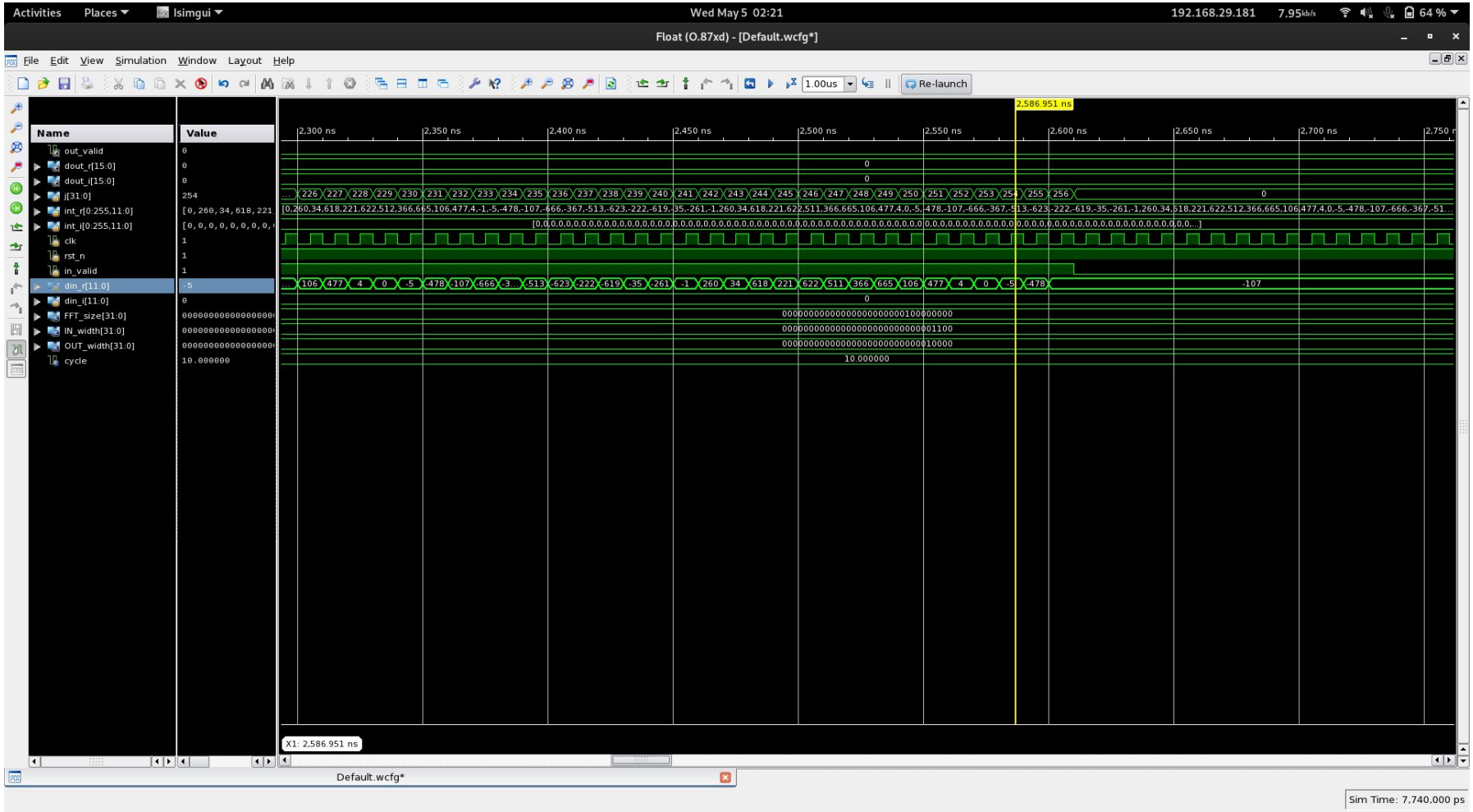
256 FFT RTL schematic



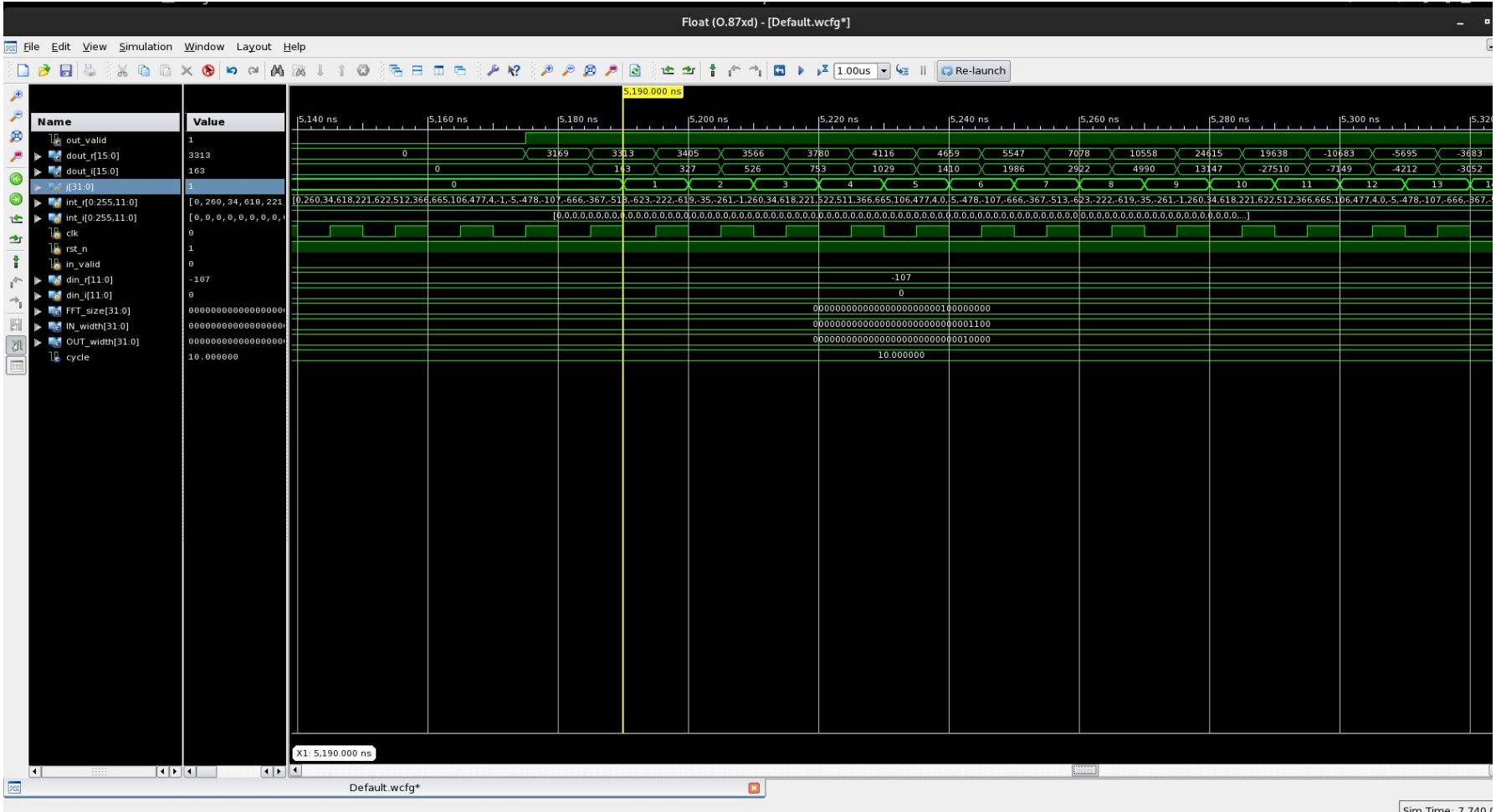
Output of the complete simulation for 256 FFT



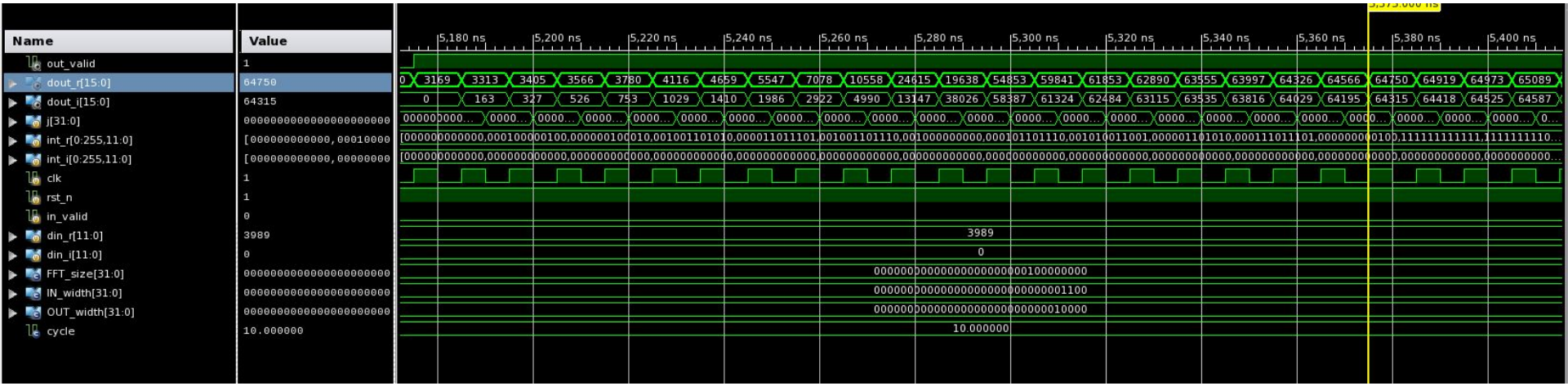
Loading Phase for 256 FFT



Output Phase for 256 FFT



Comparing results from our code with online tool



3169.000000,0.000000
3325.260415,160.079226
3410.162279,333.460743
3556.147457,526.251365
3782.969592,751.208822
4130.316782,1027.677615
4660.332352,1406.969313
5525.740736,1969.674471

7075.974434,2921.279827
10557.371795,4983.544600
24601.874342,13135.813791
-45858.491164,-27493.551722

[Link to the online fft calculator]

2D FFT using our Verilog module for Image Processing

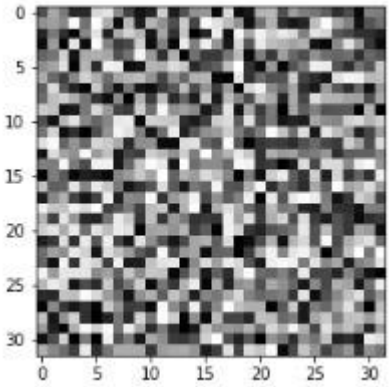
Since images are 2d type objects and here we are calculating only 1d-fft we need a way to use 1d-fft to perform 2d fft. The algorithm is:

1. do 1D FFT on each row and store the output. (real to complex)
2. do 1D FFT on each column resulting from (1) (complex to complex)

This is not directly applicable because verilog cannot load images. So we use python and cv2 to generate a random image save it in .bmp format. Then use a matlab script to convert it to .hex format. Then we read image (32x32) as a flattened array (1x1024) in verilog. And perform the above algorithm in that.

Output of 2D FFT using a sample image as input in the testbench

```
ISim>
# run 100.00us
-4024 0
-1235 -71
698 -1286
3868 171
1231 -1283
1242 -1150
-2171 2020
1317 1529
-1737 -3375
-133 1214
2858 -957
-1426 -1018
-1690 1261
-2964 -939
534 -430
-1965 -214
4886 0
-1965 211
537 430
-2970 940
-1690 -1262
-1423 1016
2857 955
-136 -1218
-1737 3375
1314 -1527
-2171 -2020
```

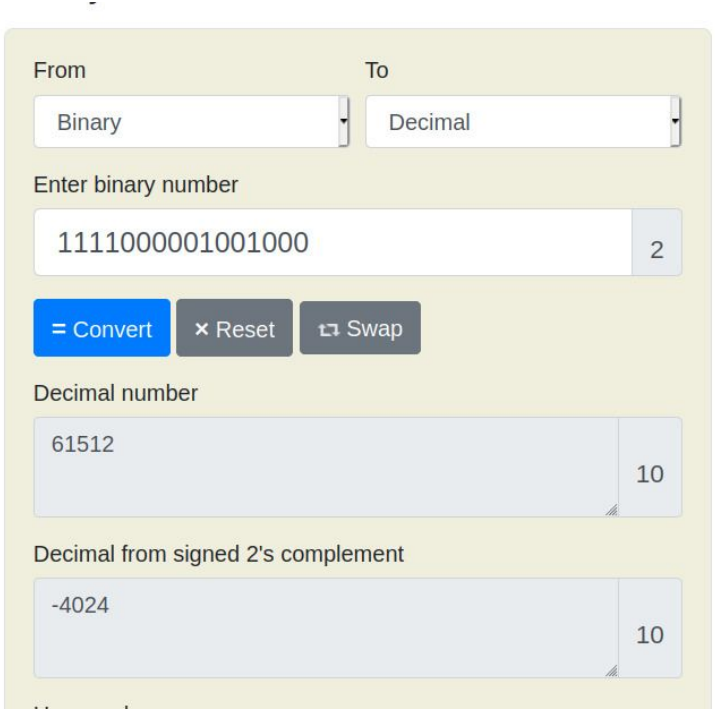


This shows the FFT output for the same image computed using python.

```
1 from numpy.fft import fft2, ifft2,fft
2 #  fft(tem[0,:])
3  fft2(tem)[:,0]
```

```
array([127048.          +0.j        ,  -1231.88531888  -72.04908869j,
         699.85139534-1283.62876995j,   3863.07771124 +172.53904917j,
        1231.17550315-1282.60660172j,   1246.10762721-1149.40460618j,
       -2165.97045459+2019.30978146j,   1313.67256913+1526.70728926j,
       -1737.        -3375.j        ,   -135.73476256+1215.19882978j,
        2853.61302349 -957.35039032j,  -1423.89581308-1015.33218231j,
       -1689.17550315+1261.39339828j,  -2964.33079699 -938.78968066j,
         536.50603576 -428.28894173j,  -1963.01121606 -212.95870187j,
        4886.          +0.j        ,  -1963.01121606 +212.95870187j,
         536.50603576 +428.28894173j,  -2964.33079699 +938.78968066j,
       -1689.17550315-1261.39339828j,  -1423.89581308+1015.33218231j,
        2853.61302349 +957.35039032j,   -135.73476256-1215.19882978j,
       -1737.        +3375.j        ,   1313.67256913-1526.70728926j,
       -2165.97045459-2019.30978146j,   1246.10762721+1149.40460618j,
        1231.17550315+1282.60660172j,   3863.07771124 -172.53904917j,
         699.85139534+1283.62876995j,  -1231.88531888  +72.04908869j])
```

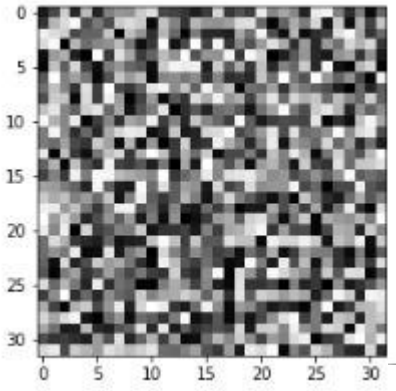
Here, we observe that the output values are similar to the expected values when computed in python.



In the output shown in previous slide, all the elements are same except the first element. It is different because 127048 is a 17 bit number and there was an overflow of bits which caused the output to go to -4024.

The following is the output for an image with smaller pixel range given as input from the testbench (performing 2D FFT)

```
Console
Finished circuit initialization process.
ISim>
# run 100.00us
25232 0
342 -57
-80 -260
-89 -9
274 78
-18 92
171 138
516 -177
312 56
63 196
383 -80
-208 -728
243 312
361 -285
-155 62
-339 -686
-636 0
-338 684
-155 -63
361 285
243 -313
-209 727
383 78
63 -197
312 -56
516 176
171 -139
-18 -92
274 -79
-88 8
-80 259
343 57
292 179
```



This is the output of FFT computed using python for the same image. The values are similar.

```
[25232.          +0.j           342.94216322 -56.96727809j
  -79.96552472-259.22549754j   -88.15485208  -8.84980339j
  274.55634919 +78.16147161j   -17.84631983 +91.79175446j
  171.51218372+138.63196387j   516.11510854-176.82911033j
  312.          +56.j           63.40706742+196.71248368j
  383.05340019 -79.04213432j  -207.56682615-726.69308378j
  243.44365081+312.16147161j   360.86742121-284.43723873j
 -154.60005918 +63.10040426j  -337.76376232-684.52828118j
 -636.          +0.j          -337.76376232+684.52828118j
 -154.60005918 -63.10040426j   360.86742121+284.43723873j
  243.44365081-312.16147161j  -207.56682615+726.69308378j
  383.05340019 +79.04213432j    63.40706742-196.71248368j
  312.          -56.j          516.11510854+176.82911033j
  171.51218372-138.63196387j   -17.84631983 -91.79175446j
  274.55634919 -78.16147161j   -88.15485208  +8.84980339j
  -79.96552472+259.22549754j   342.94216322 +56.96727809j]
```

Thankyou!