CS 288 Group 19 Project Documentation

Goal of the experiment:

The goal of this experiment is to implement an 8-point radix 2 Decimation in frequency(DIF) Fast Fourier Transform in VHDL

Procedure:

--Number representation (32 total bits) : 16 bits for real part and 16 bits for imaginary part.
--Now in the 16 bit: 1 sign bit, 7 bits to represent the integral part, and 8 to represent the fractional part. Choosing this allows us precision of 0.00390625

To implement the Fast Fourier Transform we need to first build some fundamental units efficiently which are the building blocks of FFT at its core.

1) 16 bit Adder

   To implement this we have used the inbuilt libraries ieee.std_logic_arith.all and ieee.std_logic_unsigned.all. From them we use the inbuilt + and – operators. Considering the four cases formed by the various combination of the sign bits of the two inputs we perform the appropriate operations and output the result as a 16 bit number.

2) 32 bit Adder

   The 32 bit adder simply uses two 16 bit adder to perform computations corresponding to the real and imaginary part of the 32 bit complex number. This is done using components and their proper port mapping with the right bits in the 32 bit adder.

3) 16 bit Subtractor

To implement this we have used the inbuilt libraries ieee.std_logic_arith.all and ieee.std_logic_unsigned.all. From them we use the inbuilt + and – operators.
Here also due to the combinations of the two sign bits from the two inputs we encounter 4 cases. Using a process and proper if-else statements we have performed the required operations and output the result as a 16 bit binary number.

4) 32 bit Subtractor

To implement the 32 bit adder we have simply used two 16 bit adders to perform the computations corresponding to the real and imaginary parts of the complex number.
This is done using components and their proper port mapping with the right bits in the 32 bit subtractor.

5) 16 bit Multiplier

To implement the 16 bit multiplier we have used the above mentioned inbuilt libraries. From them we have used the * operator to multiply two 15 bit unsigned binary numbers. Using the process statement and the if-else statement we have taken care of the cases arising through the various combinations of the sign bits of the two inputs. Note that the output of the 15X15 bit multiplier would be a 32 bit number. But we have to still maintain the standard and output the result in a sixteen bit format. The only time we have to multiply is the case when we are multiplying by the twiddle factor. Now the twiddle factor has magnitude always less that 1 so the integral part of the product would have the same size or less as that of the other number in the product. So when truncating the 32 bit output into a 16 bit output we ignore the extra decimal bits after 8 bits thus losing some precision but without losing sight of the accuracy.

6) 32 bit multiplier

   To implement the 32 bit multiplier we have used

   1) 2 16 bit multiplier
   2) 1 16 bit adder
   3) 1 16 bit subtractor

   Consider the following two complex numbers. a+bi and c+di . When multiplying the real part of the output will be ac – bd and imaginary part would be ad + bc. So as is apparent from the value we'll need two 16 bit multipliers and one each of 16 bit adder and subtractor to obtain the final value of the complex product. This is done using the component statement along with their proper initialisation connecting the right output to the correct bits of the 32 bit complex product.

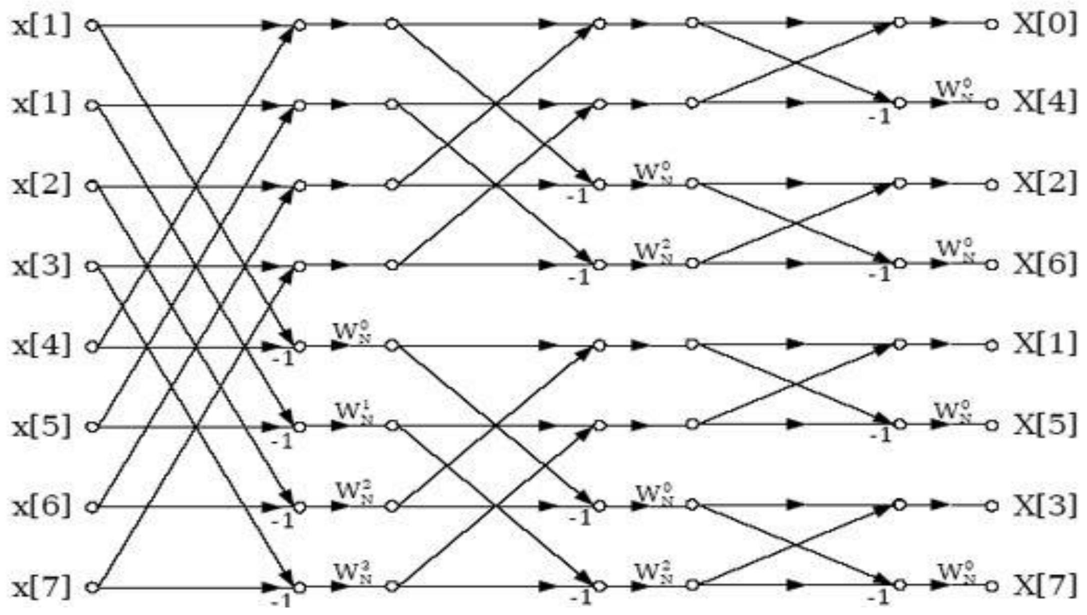
Now that we are finished with describing the basic computation unit, lets move on to the real thing. The Fast Fourier Transform.

Fast Fourier Transform(FFT) is a divide-and-conquer algorithm to compute the Discrete Fourier Transform of n sampling points(in this case 8).
Given inputs x_0 to x_N, the DFT(X_0 to X_N) is calculated as follows
$X_k = \sum_{n=0}^{n=N-1} x(n)(e^{-i2\pi kn/N})$ k=0,1,2…..N-1. In our case N=8.
FFT is an algorithm to compute the DFT in O(*NlogN)* time instead of O(N^2*)* time.
It essentially uses this formula to compute each X_k .

$$X_k = \underbrace{\sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2}mk}}_{\text{DFT of even}-\text{indexed part of } x_m} + e^{-\frac{2\pi i}{N}k} \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2}mk}}_{\text{DFT of odd}-\text{indexed part of } x_m} = E_k + e^{-\frac{2\pi i}{N}k} O_k.$$

(courtesy: Wikipedia)

At the heart of the FFT algorithm lie the Butterfly operations. Each butterfly operation takes 2 inputs, computes their sum(which is one output) and their difference. The difference is then multiplied with an appropriate Twiddle factor(W_8^k). 8-point radix 2 FFT can be broken down into 3 blocks of 4 butterfly operations each as is evident from the figure.



This is exactly the procedure used for this project. We first take in the 8 inputs, perform 4 butterfly operations with each appropriate pair and store them results in 8 temporary signals(temp01 to temp07).

In this first block, the Twiddle factors used are W_8^0 to W_8^3. These signals are then passed as inputs to the "so-called" second block of butterfly operations.

Since this is corresponding to two 4 point FFTs, the twiddle factors used are only W_8^0 and W_8^2. The corresponding 8 outputs of the second block are then stored in 8 signals(temp10 to temp17). These signals are then passed as inputs to the final block of butterfly operations.

This corresponds to four 2-point FFTs and, therefore, the Twiddle factor W_8^0 is only used.

The reason for choosing this procedure is that it is essentially an iterative procedure for computing the FFT of 8 inputs and is easy to code up in VHDL.

Entities used:

1. Butterfly entity : Takes two inputs(input0 and input1) along with W_8^k and returns two outputs(output0 and output1) .

```
entity Butterfly is
    Port ( input0 : in  STD_LOGIC_VECTOR (31 downto 0);
             input1 : in  STD_LOGIC_VECTOR (31 downto 0);
            wk : in STD_LOGIC_VECTOR(31 downto 0);
             output0 : out  STD_LOGIC_VECTOR (31 downto 0);
             output1 : out  STD_LOGIC_VECTOR (31 downto 0)
             );
end Butterfly;
```

2. EightPointFFT: Takes 8 inputs (32 bits each representing a complex number in our representation) and carries out the 3 blocks of butterfly operations to return the 8 FFT outputs(output0 to output7)

```
entity EightPointFFT is
    Port ( input0 : in  STD_LOGIC_VECTOR (31 downto 0);
           input1 : in  STD_LOGIC_VECTOR (31 downto 0);
           input2 : in  STD_LOGIC_VECTOR (31 downto 0);
           input3 : in  STD_LOGIC_VECTOR (31 downto 0);
           input4 : in  STD_LOGIC_VECTOR (31 downto 0);
           input5 : in  STD_LOGIC_VECTOR (31 downto 0);
           input6 : in  STD_LOGIC_VECTOR (31 downto 0);
           input7 : in  STD_LOGIC_VECTOR (31 downto 0);
           output0 : out  STD_LOGIC_VECTOR (31 downto 0);
           output1 : out  STD_LOGIC_VECTOR (31 downto 0);
           output2 : out  STD_LOGIC_VECTOR (31 downto 0);
           output3 : out  STD_LOGIC_VECTOR (31 downto 0);
           output4 : out  STD_LOGIC_VECTOR (31 downto 0);
           output5 : out  STD_LOGIC_VECTOR (31 downto 0);
           output6 : out  STD_LOGIC_VECTOR (31 downto 0);
           output7 : out  STD_LOGIC_VECTOR (31 downto 0)
);
end EightPointFFT;
```

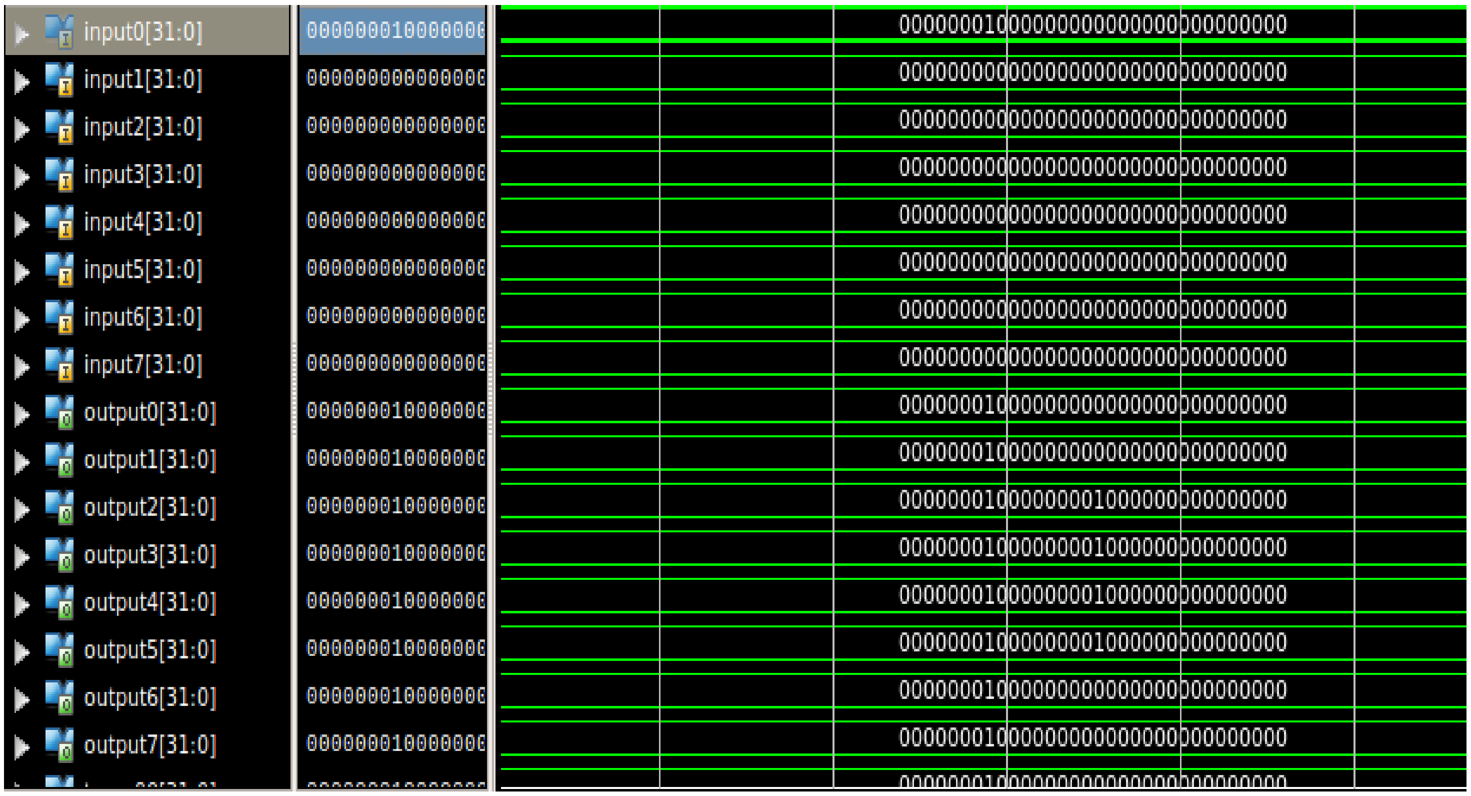
3. //////////For Ved to fill up(brief description should suffice if properly covered at the top)

Results: We tested our code using the simulator on some sample inputs. The screenshots of the same are given below:

Screenshot 1

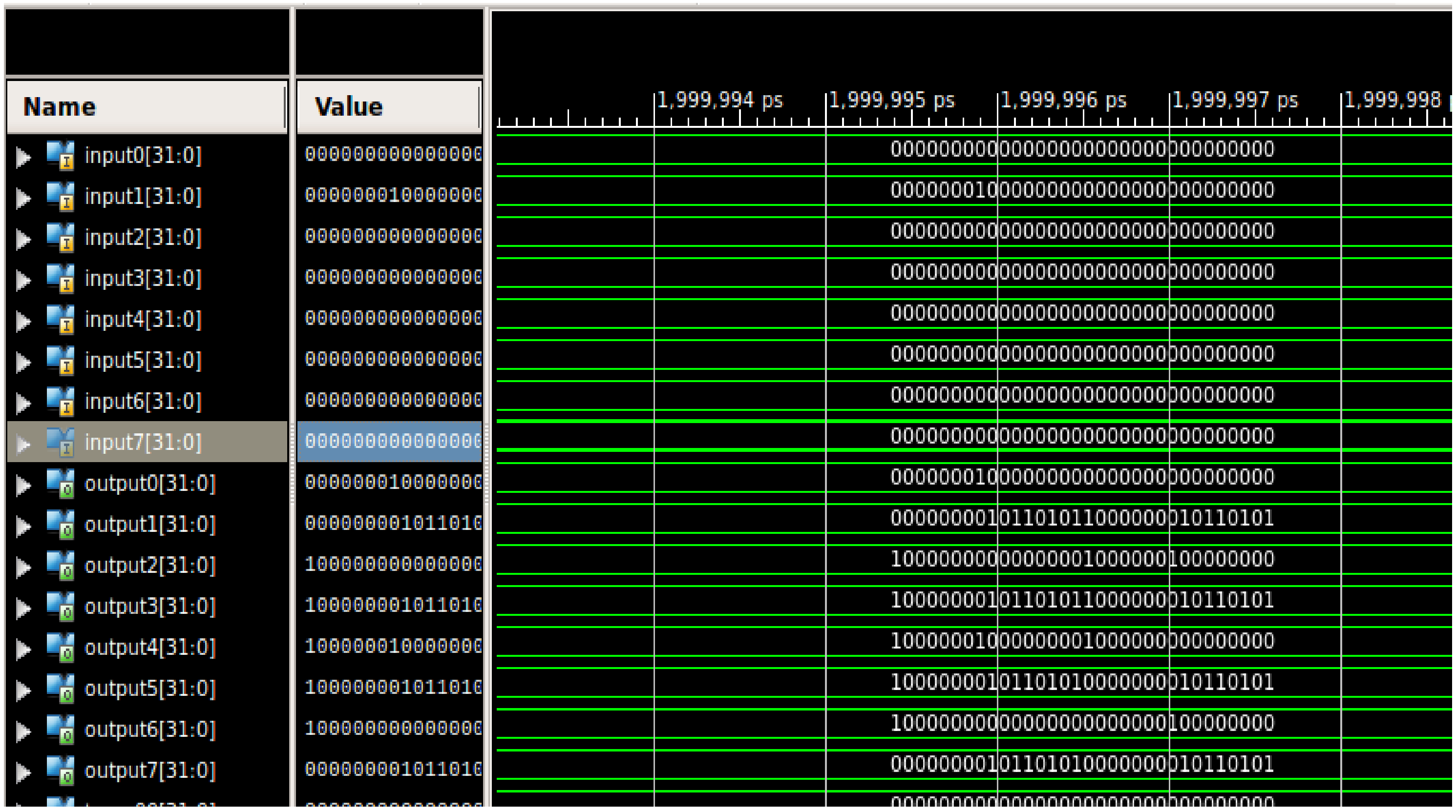
input0 : 1 + 0i **all outputs: 1**

other inputs : 0 + 0i



```
input0[31:0]   0000000100000000   00000001000000000000000000000000
input1[31:0]   0000000000000000   00000000000000000000000000000000
input2[31:0]   0000000000000000   00000000000000000000000000000000
input3[31:0]   0000000000000000   00000000000000000000000000000000
input4[31:0]   0000000000000000   00000000000000000000000000000000
input5[31:0]   0000000000000000   00000000000000000000000000000000
input6[31:0]   0000000000000000   00000000000000000000000000000000
input7[31:0]   0000000000000000   00000000000000000000000000000000
output0[31:0]  0000000100000000   00000001000000000000000000000000
output1[31:0]  0000000100000000   00000001000000000000000000000000
output2[31:0]  0000000100000000   00000001000000001000000000000000
output3[31:0]  0000000100000000   00000001000000001000000000000000
output4[31:0]  0000000100000000   00000001000000001000000000000000
output5[31:0]  0000000100000000   00000001000000001000000000000000
output6[31:0]  0000000100000000   00000001000000000000000000000000
output7[31:0]  0000000100000000   00000001000000000000000000000000
```

Screenshot 2

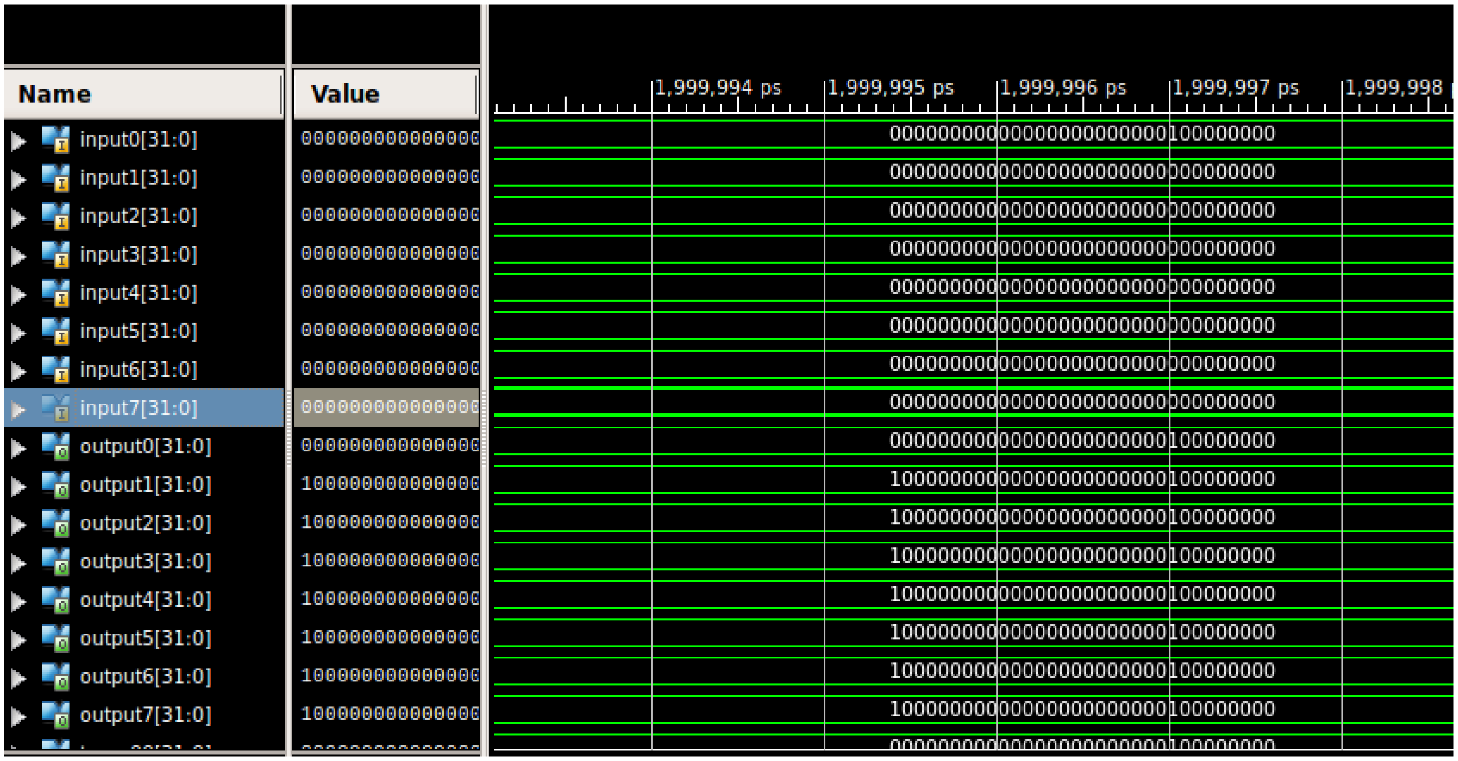


```
Name            Value             1,999,994 ps   1,999,995 ps   1,999,996 ps   1,999,997 ps   1,999,998
input0[31:0]    000000000000000   00000000000000000000000000000000
input1[31:0]    000000010000000   00000001000000000000000000000000
input2[31:0]    000000000000000   00000000000000000000000000000000
input3[31:0]    000000000000000   00000000000000000000000000000000
input4[31:0]    000000000000000   00000000000000000000000000000000
input5[31:0]    000000000000000   00000000000000000000000000000000
input6[31:0]    000000000000000   00000000000000000000000000000000
input7[31:0]    000000000000000   00000000000000000000000000000000
output0[31:0]   000000010000000   00000001000000000000000000000000
output1[31:0]   000000001011010   00000000101101011000000010110101
output2[31:0]   100000000000000   10000000000000001000000100000000
output3[31:0]   100000001011010   10000000101101011000000010110101
output4[31:0]   100000010000000   10000001000000001000000000000000
output5[31:0]   100000001011010   10000000101101010000000010110101
output6[31:0]   100000000000000   10000000000000000000000100000000
output7[31:0]   000000001011010   00000000101101010000000010110101
```

input1 : 1 + 0i
other inputs : 0+0i

Screenshot 3

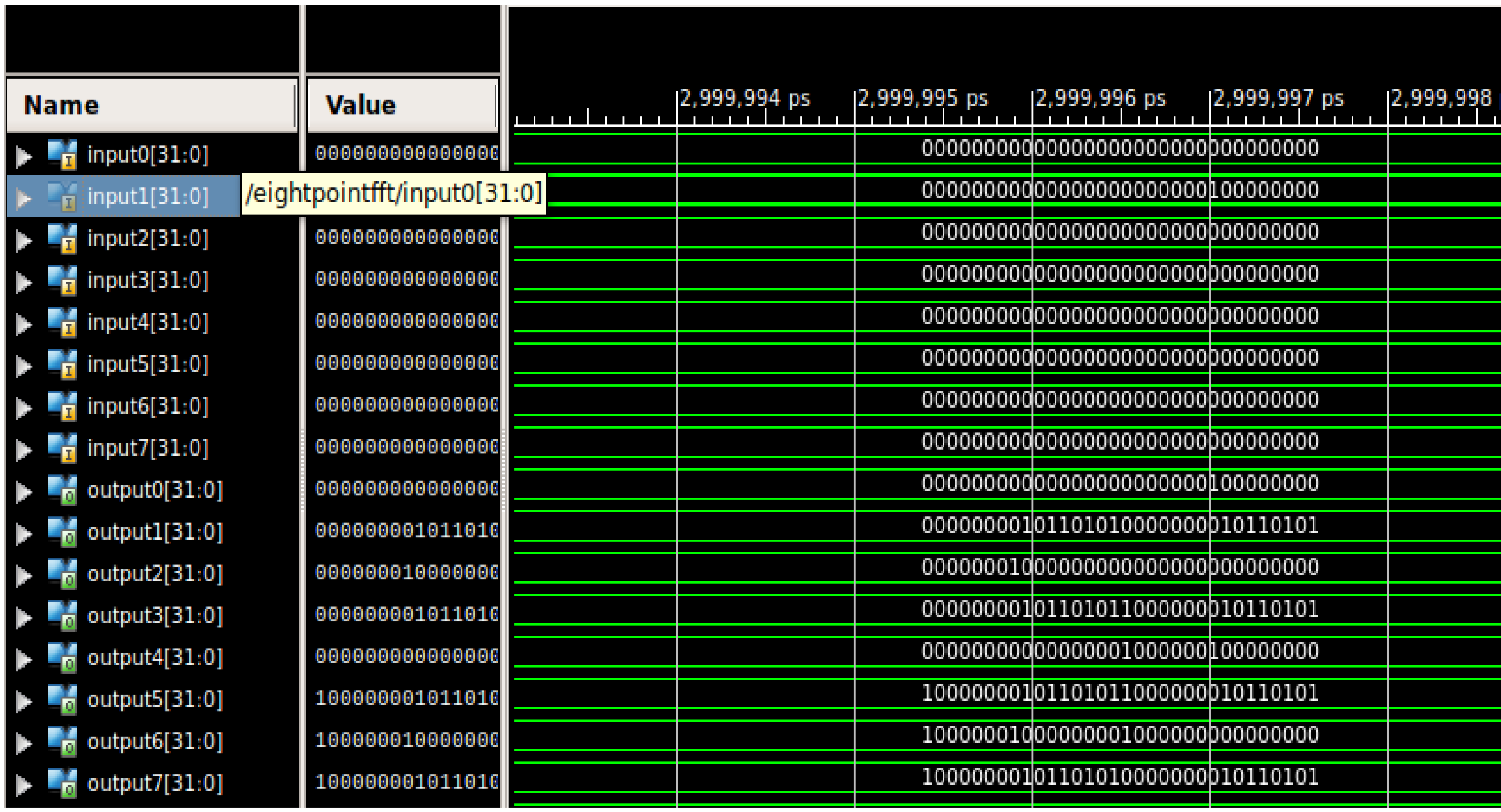
input0 : 0 + 1i   other inputs : 0+0i   output(all) : 0 + 1i



```
Name             Value              |1,999,994 ps  |1,999,995 ps  |1,999,996 ps  |1,999,997 ps  |1,999,998
input0[31:0]     000000000000000    00000000000000000000000100000000
input1[31:0]     000000000000000    00000000000000000000000000000000
input2[31:0]     000000000000000    00000000000000000000000000000000
input3[31:0]     000000000000000    00000000000000000000000000000000
input4[31:0]     000000000000000    00000000000000000000000000000000
input5[31:0]     000000000000000    00000000000000000000000000000000
input6[31:0]     000000000000000    00000000000000000000000000000000
input7[31:0]     000000000000000    00000000000000000000000000000000
output0[31:0]    000000000000000    00000000000000000000000100000000
output1[31:0]    100000000000000    10000000000000000000000100000000
output2[31:0]    100000000000000    10000000000000000000000100000000
output3[31:0]    100000000000000    10000000000000000000000100000000
output4[31:0]    100000000000000    10000000000000000000000100000000
output5[31:0]    100000000000000    10000000000000000000000100000000
output6[31:0]    100000000000000    10000000000000000000000100000000
output7[31:0]    100000000000000    10000000000000000000000100000000
```

Screenshot 4

input1 : 0 + 1i



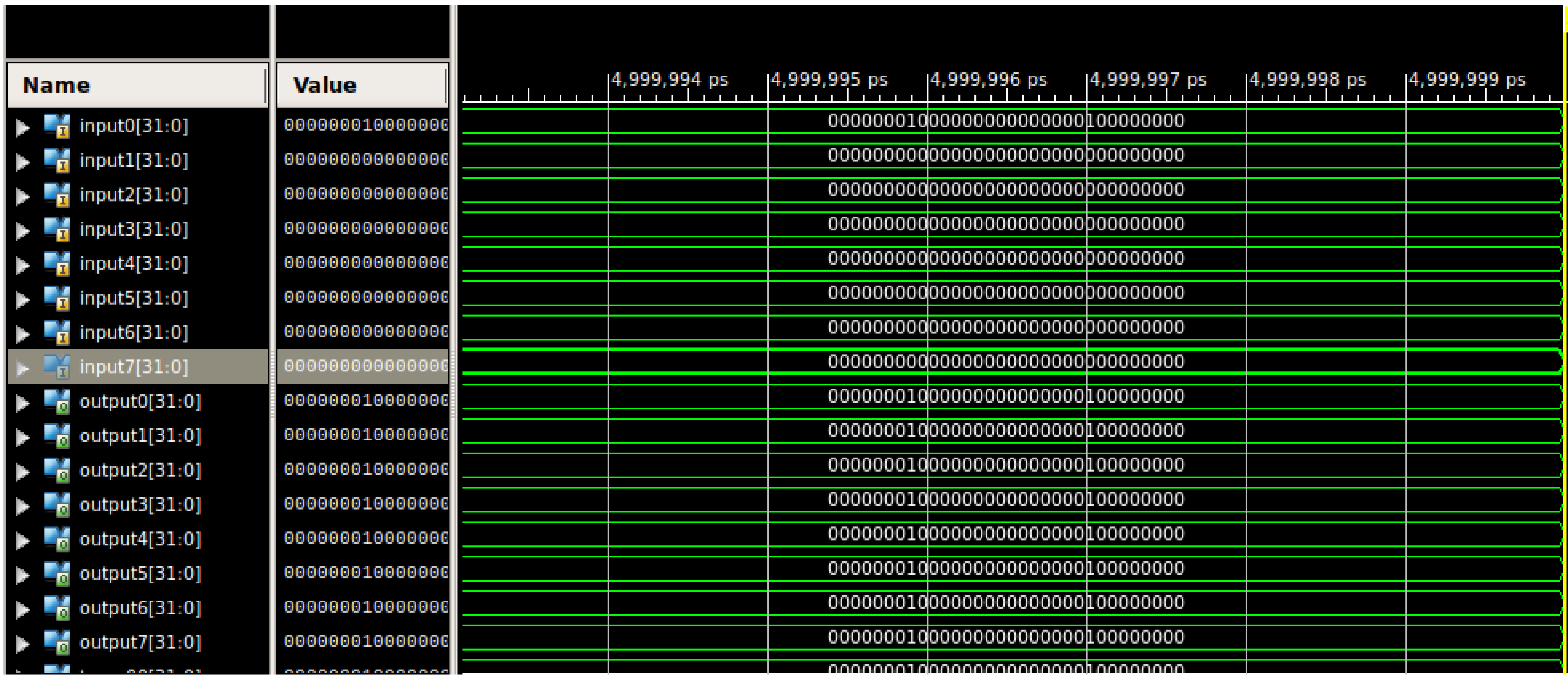
```
Name             Value               |2,999,994 ps  |2,999,995 ps  |2,999,996 ps  |2,999,997 ps  |2,999,998
input0[31:0]     000000000000000     00000000000000000000000000000000
input1[31:0]     /eightpointfft/input0[31:0]   00000000000000000000000100000000
input2[31:0]     000000000000000     00000000000000000000000000000000
input3[31:0]     000000000000000     00000000000000000000000000000000
input4[31:0]     000000000000000     00000000000000000000000000000000
input5[31:0]     000000000000000     00000000000000000000000000000000
input6[31:0]     000000000000000     00000000000000000000000000000000
input7[31:0]     000000000000000     00000000000000000000000000000000
output0[31:0]    000000000000000     00000000000000000000000100000000
output1[31:0]    000000001011010     00000000101101010000000010110101
output2[31:0]    000000010000000     00000001000000000000000000000000
output3[31:0]    000000001011010     00000000101101011000000010110101
output4[31:0]    000000000000000     00000000000000001000000100000000
output5[31:0]    100000001011010     10000000101101011000000010110101
output6[31:0]    100000010000000     10000001000000001000000000000000
output7[31:0]    100000001011010     10000000101101010000000010110101
```

others : 0+0i

Screenshot 5

input1 : 1 + 1i
others : 0+0i

output : 1 + 1i



```
Name              Value               4,999,994 ps  4,999,995 ps  4,999,996 ps  4,999,997 ps  4,999,998 ps  4,999,999 ps
input0[31:0]      0000000100000000    00000001000000000000000100000000
input1[31:0]      0000000000000000    00000000000000000000000000000000
input2[31:0]      0000000000000000    00000000000000000000000000000000
input3[31:0]      0000000000000000    00000000000000000000000000000000
input4[31:0]      0000000000000000    00000000000000000000000000000000
input5[31:0]      0000000000000000    00000000000000000000000000000000
input6[31:0]      0000000000000000    00000000000000000000000000000000
input7[31:0]      0000000000000000    00000000000000000000000000000000
output0[31:0]     0000000100000000    00000001000000000000000100000000
output1[31:0]     0000000100000000    00000001000000000000000100000000
output2[31:0]     0000000100000000    00000001000000000000000100000000
output3[31:0]     0000000100000000    00000001000000000000000100000000
output4[31:0]     0000000100000000    00000001000000000000000100000000
output5[31:0]     0000000100000000    00000001000000000000000100000000
output6[31:0]     0000000100000000    00000001000000000000000100000000
output7[31:0]     0000000100000000    00000001000000000000000100000000
```