PROJECT REPORT
ON

ECL302

HARDWARE DESCRIPTION LANGUAGE

"Fourier Transform Calculation"

Submitted to
IIIT NAGPUR

Under the Guidance of
Dr. VIPIN KAMBLE

DEPARTMENT OF
ELECTRONICS AND COMMUNICATION ENGINEERING
IIIT NAGPUR
2019-2020

BY

BT18ECE015 KODIGUDLA GAYATHRI
BT18ECE026 ANUJ LATHI
BT18ECE030 SHRIJEET KHARCHE

Table of Contents

1 INTRODUCTION

1.1 OBJECTIVE

The Objective of the project is to compute the Discrete Fourier Transform (DFT) using the Decimation in Time Radix 2 FFT algorithm with the help of Hardware Description Language.

1.2 ABSTRACT

The Discrete Fourier Transform (DFT) can be executed exceptionally quick utilizing Fast Fourier Transform (FFT). It is probably the best activity in the territory of Digital Signal & Image Processing. There are many different algorithms available in digital signal processing. Among them the Cooley–Tukey algorithm the Radix-2 decimation-in-time Fast Fourier Transform is the simplest form and with least complexity O(N log(N)). This algorithm breaks DFT into smaller DFTs, it can be combined arbitrarily with the help of butterfly stages. DFT is used to convert a time-domain signal into its frequency spectrum domain. FFT algorithms use many applications, for example, OFDM, Noise reduction, Digital audio broadcasting, Digital video broadcasting. It's used to design butterflies for different point FFT.

In this project, we present the implementation of fast algorithms for the DFT for repesenting time domain discrete samples to frequency domain. The simulation of this algorithm will take place by implementing them on the Xilinx ISE 14.7 Spartan 3E by developing our own FFT processor architecture.

2 METHODOLOGY

The Discrete Fourier Transform allows to develop a large number of applications applied to images, among which are: filtering (using convolution in the frequency domain), image compression and encryption, and object or people recognition (with the correlation operation in the domain frequency), etc. The possibility of implementation of the above applications are due to the existence of a fast algorithm for calculating the DFT, this algorithm is known as the Fast Fourier Transform (FFT), The first FFT algorithms were published by Cooley and Tukey in 1965.

In this project we are going to implement the Cooley-Tukey Radix-2 Algorithm in Hardware Description Language. Due to radix-2, FFT can achieve in less time delay, beat down the area complication and, also reach cost dominant execution with minimum grow up time.

2.1 ALGORITHM

The analysis equation for $N-$point DFT are given as,

$$X(k) = \sum_{n=0}^{N-1} x[n] W_N^{kn} \tag{1}$$

In order to find X(1), we require N multiplications and $N-1$ additions. For example, if $x[n] = \{1, 2, 3, 4\}$, and $N = 4$, then,

$$X(1) = 1 \times W_N^{1\cdot 0} + 2 \times W_N^{1\cdot 2} + 3 \times W_N^{1\cdot 2} + 4 \times W_N^{1\cdot 3}$$

In this we have $N = 4$ multiplications and $N - 1 = 3$ additions. Similar thing will be observed for calculating the values of $X(0),\ X(1),\ X(2), \ldots,\ X(N-1)$, i.e. N values of the DFT. Therefore in total,

- Number of multiplications $= N \times N = N^2$
- Number of additions $= N \times (N-1)$

Therefore, in order to find this using computational tools, we would require two for loops each of them runs N times. Therefore, complexity of this implementation is of $O(N^2)$ which is very high. This gives rise to the need for development of an optimized, or a linear or logarithmic order implementation.
Consider the analysis equation. We can split it in two parts:

- where n is odd, and
- where n is even.

Therefore we have,

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x[n] W_N^{kn} \\ &= \sum_{n\ \text{even}} x[n] W_N^{kn} + \sum_{n\ \text{odd}} x[n] W_N^{kn} \end{aligned}$$

We can represent the even number as $2n$ and odd numbers as $2n+1$ such that n varies from 0 to $\frac{N}{2} - 1$. Therefore we have,

$$\begin{aligned} X(k) &= \sum_{n=0}^{\frac{N}{2}-1} x[2n] W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} x[2n+1] W_N^{(2n+1)k} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x[2n] W_N^{2nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x[2n+1] W_N^{2nk} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x[2n] W_{\frac{N}{2}}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x[2n+1] W_{\frac{N}{2}}^{nk} \\ &= X^e(k) + W_N^k X^o(k) \end{aligned}$$

where,

$$X^e(k) = \sum_{n=0}^{\frac{N}{2}-1} x[2n] W_{\frac{N}{2}}^{nk}$$
$$X^o(k) = \sum_{n=0}^{\frac{N}{2}-1} x[2n+1] W_{\frac{N}{2}}^{nk}$$

We can observe that, both $X^e(k)$ and $X^o(k)$ are $N/2$-point DFT (Observe the subscript of W), i.e., k varies from 0 to $\frac{N}{2}-1$. Moreover, we can say that $X^o(k)$ and $X^e(k)$ are periodic with periodic with period $\frac{N}{2}$. Therefore, $X^o\left(k+\frac{N}{2}\right) = X^o(k)$ and $X^e\left(k+\frac{N}{2}\right) = X^e(k)$. Therefore,

$$\begin{aligned} X\left(k+\frac{N}{2}\right) &= X^e\left(k+\frac{N}{2}\right) + W_N^{k+\frac{N}{2}} X^o\left(k+\frac{N}{2}\right) \\ &= X^e(k) - W_N^k X^o(k) \end{aligned}$$

Therefore, we have,

$$X(k) = X^e(k) + W_N^k X^o(k) \tag{2}$$
$$X\left(k+\frac{N}{2}\right) = X^e(k) - W_N^k X^o(k) \tag{3}$$

were, $k = \{0, 1, \ldots \frac{N}{2}-1\}$. Using above two equations, we can calculate the $N-$point DFT of the sequence. If we observe the equations of $X^o(k)$ and $X^e(k)$, we have the term $x[2n]$ and not $x[n]$. For example if $x[n] = \{1, 2, 3, 4\}$, then $x[2n] = \{1, 3\}$. i.e. we consider alternate values only. This is known as *decimation by a factor of 2*. As we are decimating the signal in time domain, this method pf finding the DFT is known as *Decimation in Time-DFT* or *DIT-DFT* of *DIT-FFT* of radix 2.

Example : Consider the input signal $x[n] = \{\underset{\uparrow}{1}, 2, 3, 4, 5, 6, 7, 8\}$

Sol^n **:-**

1. Rearranging the signal values using bit-reversal form as shown in Table 1.

Table 1: Bit reversed representation

n	Original Sequence	Bit Representation of n	Reversed Bit	Bit reversed sequence
0	$x[0]$	000	000	$x[0]$
1	$x[1]$	001	100	$x[4]$
2	$x[2]$	010	010	$x[2]$
3	$x[3]$	011	110	$x[6]$
4	$x[4]$	100	001	$x[1]$
5	$x[5]$	101	101	$x[5]$
6	$x[6]$	110	011	$x[3]$
7	$x[7]$	111	111	$x[7]$

2. First consider two consecutive values and find the 2-point FFT. The output is $\{6, -4, 10, -4, 8, -4, 12, -4\}$.
3. Again consider two values of two consecutive values obtained earlier and find the 2-point FFT. The output is $\{16, -4+4j, -4, -4-4j, 20, -4+4j, -4, -4-4j\}$.
4. Finally, calculate the 2-point FFT in order to get final output, i.e. 8-point FFT. The final butterfly structure is as shown in Fig. Therefore, $X(k) = \{\underset{\uparrow}{36}, -4+j9.6569, -4+j4, -4+j1.6569, -4, -4-j1.6569, -4-j4, -4-j9.6569\}$.

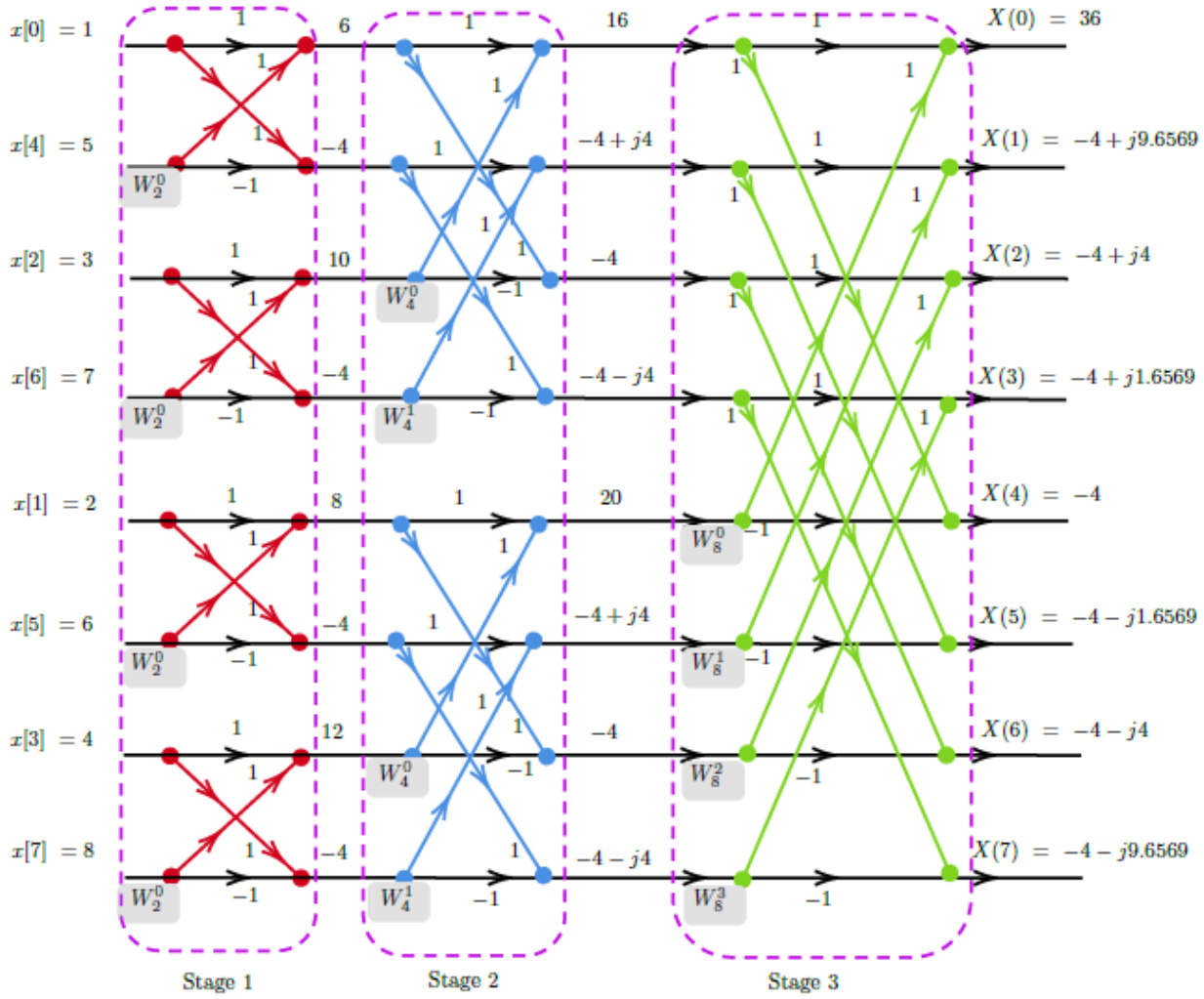

Figure 1: 8 Pt. FFT

3 CODES

Complex_Package.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4  package Complex_Package is
5
6  -- Defining a structure of complex datatype.
7    type complex is
8      record
9      Re : real;
10     Im : real;
11     end record;
12
13 -- Defining a collection of the structure as an array of datatype 'complex'.
14   type CMPLX_Array16 is array (0 to 15) of complex;
15   type CMPLX_Array8 is array (0 to 7) of complex;
16   type CMPLX_Array4 is array (0 to 3) of complex;
17   type CMPLX_Array2 is array (0 to 1) of complex;
18
19 -- Defining functions for aithmetic operations of complex numbers.
20   function Add(a, b: complex) return complex;
21   function Sub(a, b: complex) return complex;
22   function Mul(a, b: complex) return complex;
23
24 end Complex_Package;
25
26 package body Complex_Package is
27
28 -- Addition of Complex Numbers.
29   function Add(a, b: complex) return complex is
30     variable Sum : complex;
31
32     begin
33       Sum.Re := a.Re + b.Re;
34       Sum.Im := a.Im + b.Im;
35     return Sum;
36   end Add;
37
38 -- Subtraction of Complex Numbers.
39   function Sub(a, b: complex) return complex is
40     variable Diff : complex;
41
42     begin
43       Diff.Re := a.Re - b.Re;
44       Diff.Im := a.Im - b.Im;
45     return Diff;
46   end Sub;
47
48 -- Multiplication of Complex Numbers.
49   function Mul(a, b: complex) return complex is
50     variable Product : complex;
51
52     begin
```

```
53        Product.Re := (a.Re * b.Re) - (a.Im * b.Im);
54        Product.Im := (a.Re * b.Im) + (a.Im * b.Re);
55      return Product;
56    end Mul;
57
58 end Complex_Package;
```

Butterfly_Stage.vhd

```
1  library IEEE;
2  library work;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use work.Complex_Package.ALL;
5
6  entity Butterfly_Stage is
7      port(
8          p, q : in complex;  -- Inputs.
9          W :in complex;     -- Twiddle factor i.e. W_N = e^(-j*2*pi/N)
10                            --                  Where, N = Number of input points
11         r, s :out complex   -- Outputs.
12       );
13 end Butterfly_Stage;
14
15 architecture BEH of Butterfly_Stage is
16
17 begin
18
19 -- Butterfly Stage Calculation.
20   r <= Add(p, Mul(q, W));
21   s <= Sub(p, Mul(q, W));
22
23 end BEH;
```

2 pts. FFT

```
1  library IEEE;
2  library work;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.MATH_REAL.ALL;
5  use work.Complex_Package.ALL;
6
7  entity FFT2 is
8    port(
9      -- Input flag in the time area.
10       x2 : in CMPLX_Array2;
11     -- Yield flag in the recurrence area.
12       y2 : out CMPLX_Array2
13   );
14 end FFT2;
15
16 architecture STR of FFT2 is
17
18   component Butterfly_Stage is
```

```
    port(
        p, q : in complex;  -- Inputs.
        W : in complex;     -- Twiddle factor i.e. WN = e^(-j*2*pi/N)
                            --              Where, N = Number of input points
        r, s : out complex  -- Outputs.
      );
  end component;

  constant W2 : complex   := (1.0, 0.0) ;

begin

  -- 2 point DFT
    BS211 : Butterfly_Stage port map(x2(0), x2(1), W2, y2(0), y2(1));

end STR;
```

4 pts. FFT

```
library IEEE;
library work;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.MATH_REAL.ALL;
use work.Complex_Package.ALL;

entity FFT4 is
  port(
    -- Input flag in the time area.
      x4 : in CMPLX_Array4;
    -- Yield flag in the recurrence area.
      y4 : out CMPLX_Array4
  );
end FFT4;

architecture STR of FFT4 is

  component Butterfly_Stage is
    port(
        p, q : in complex;  -- Inputs.
        W : in complex;     -- Twiddle factor i.e. WN = e^(-j*2*pi/N)
                            --              Where, N = Number of input points
        r, s : out complex  -- Outputs.
      );
  end component;


-- Defining Intermediate Signals.
  signal t1 : CMPLX_Array4 := (others => (0.0, 0.0));

  constant W4 : CMPLX_Array2  := ( (1.0, 0.0), (0.0, -1.0) );

begin

  -- 4 point DFT
  -- First Butterfly Stage.
```

```
    BS411 : Butterfly_Stage port map(x4(0), x4(2), W4(0), t1(0), t1(1));
    BS412 : Butterfly_Stage port map(x4(1), x4(3), W4(0), t1(2), t1(3));
  -- Second Butterfly Stage.
    BS421 : Butterfly_Stage port map(t1(0), t1(2), W4(0), y4(0), y4(2));
    BS422 : Butterfly_Stage port map(t1(1), t1(3), W4(1), y4(1), y4(3));

end STR;
```

8 pts. FFT

```
library IEEE;
library work;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.MATH_REAL.ALL;
use work.Complex_Package.ALL;

entity FFT8 is
  port(
    -- Input flag in the time area.
      x8 : in CMPLX_Array8;
    -- Yield flag in the recurrence area.
      y8 : out CMPLX_Array8
  );
end FFT8;

architecture STR of FFT8 is

  component Butterfly_Stage is
    port(
        p, q : in complex;  -- Inputs.
        W : in complex;     -- Twiddle factor i.e. W_N = e^(-j*2*pi/N)
                    --              Where, N = Number of input points
        r, s : out complex  -- Outputs.
      );
  end component;


-- Defining Intermediate Signals.
  signal t2,t3 : CMPLX_Array8 := (others => (0.0, 0.0));

  constant W8 : CMPLX_Array4  := ( (1.0, 0.0), (0.7071, -0.7071), (0.0, -1.0),
     (-0.7071, -0.7071) );

begin

  -- 8 point DFT
  -- First Butterfly Stage.
    BS811 : Butterfly_Stage port map(x8(0), x8(4), W8(0), t2(0), t2(1));
    BS812 : Butterfly_Stage port map(x8(2), x8(6), W8(0), t2(2), t2(3));
    BS813 : Butterfly_Stage port map(x8(1), x8(5), W8(0), t2(4), t2(5));
    BS814 : Butterfly_Stage port map(x8(3), x8(7), W8(0), t2(6), t2(7));

  -- Second Butterfly Stage.
    BS821 : Butterfly_Stage port map(t2(0), t2(2), W8(0), t3(0), t3(2));
    BS822 : Butterfly_Stage port map(t2(1), t2(3), W8(2), t3(1), t3(3));
```

```
45      BS823 : Butterfly_Stage port map(t2(4), t2(6), W8(0), t3(4), t3(6));
46      BS824 : Butterfly_Stage port map(t2(5), t2(7), W8(2), t3(5), t3(7));
47
48    -- Third Butterfly Stage.
49      BS831 : Butterfly_Stage port map(t3(0), t3(4), W8(0), y8(0), y8(4));
50      BS832 : Butterfly_Stage port map(t3(1), t3(5), W8(1), y8(1), y8(5));
51      BS833 : Butterfly_Stage port map(t3(2), t3(6), W8(2), y8(2), y8(6));
52      BS834 : Butterfly_Stage port map(t3(3), t3(7), W8(3), y8(3), y8(7));
53
54
55  end STR;
```

16 pts. FFT

```
1   library IEEE;
2   library work;
3   use IEEE.STD_LOGIC_1164.ALL;
4   use IEEE.MATH_REAL.ALL;
5   use work.Complex_Package.ALL;
6
7   entity FFT16 is
8     port(
9       -- Input flag in the time area.
10        x16 : in CMPLX_Array16;
11      -- Yield flag in the recurrence area.
12        y16 : out CMPLX_Array16
13    );
14  end FFT16;
15
16  architecture STR of FFT16 is
17
18    component Butterfly_Stage is
19      port(
20          p, q : in complex;  -- Inputs.
21          W : in complex;     -- Twiddle factor i.e. W_N = e^(-j*2*pi/N)
22                              --              Where, N = Number of input points
23          r, s : out complex  -- Outputs.
24        );
25    end component;
26
27
28  -- Defining Intermediate Signals.
29    signal t4,t5,t6 : CMPLX_Array16 := (others => (0.0, 0.0));
30
31    constant W16 : CMPLX_Array8 := ( (1.0, 0.0), (0.7071, -0.7071), (0.0, -1.0),
          (-0.7071, -0.7071),
32                          (0.9239, -0.3827), (0.3827, -0.9239), (-0.3827, -0.9239),
                              (-0.9239, -0.3827) );
33
34  begin
35
36    -- 16 point DFT
37    -- First Butterfly Stage.
38      BS1611 : Butterfly_Stage port map(x16(0), x16(8), W16(0), t4(0), t4(1));
39      BS1612 : Butterfly_Stage port map(x16(4), x16(12), W16(0), t4(2), t4(3));
```

```
    BS1613 : Butterfly_Stage port map(x16(2), x16(10), W16(0), t4(4), t4(5));
    BS1614 : Butterfly_Stage port map(x16(6), x16(14), W16(0), t4(6), t4(7));
    BS1615 : Butterfly_Stage port map(x16(1), x16(9), W16(0), t4(8), t4(9));
    BS1616 : Butterfly_Stage port map(x16(5), x16(13), W16(0), t4(10), t4(11));
    BS1617 : Butterfly_Stage port map(x16(3), x16(11), W16(0), t4(12), t4(13));
    BS1618 : Butterfly_Stage port map(x16(7), x16(15), W16(0), t4(14), t4(15));

  -- Second Butterfly Stage.
    BS1621 : Butterfly_Stage port map(t4(0), t4(2), W16(0), t5(0), t5(2));
    BS1622 : Butterfly_Stage port map(t4(1), t4(3), W16(2), t5(1), t5(3));
    BS1623 : Butterfly_Stage port map(t4(4), t4(6), W16(0), t5(4), t5(6));
    BS1624 : Butterfly_Stage port map(t4(5), t4(7), W16(2), t5(5), t5(7));
    BS1625 : Butterfly_Stage port map(t4(8), t4(10), W16(0), t5(8), t5(10));
    BS1626 : Butterfly_Stage port map(t4(9), t4(11), W16(2), t5(9), t5(11));
    BS1627 : Butterfly_Stage port map(t4(12), t4(14), W16(0), t5(12), t5(14));
    BS1628 : Butterfly_Stage port map(t4(13), t4(15), W16(2), t5(13), t5(15));

  -- Third Butterfly Stage.
    BS1631 : Butterfly_Stage port map(t5(0), t5(4), W16(0), t6(0), t6(4));
    BS1632 : Butterfly_Stage port map(t5(1), t5(5), W16(1), t6(1), t6(5));
    BS1633 : Butterfly_Stage port map(t5(2), t5(6), W16(2), t6(2), t6(6));
    BS1634 : Butterfly_Stage port map(t5(3), t5(7), W16(3), t6(3), t6(7));
    BS1635 : Butterfly_Stage port map(t5(8), t5(12), W16(0), t6(8), t6(12));
    BS1636 : Butterfly_Stage port map(t5(9), t5(13), W16(1), t6(9), t6(13));
    BS1637 : Butterfly_Stage port map(t5(10), t5(14), W16(2), t6(10), t6(14));
    BS1638 : Butterfly_Stage port map(t5(11), t5(15), W16(3), t6(11), t6(15));

  -- Fourth Butterfly Stage.
    BS1641 : Butterfly_Stage port map(t6(0), t6(8), W16(0), y16(0), y16(8));
    BS1642 : Butterfly_Stage port map(t6(1), t6(9), W16(4), y16(1), y16(9));
    BS1643 : Butterfly_Stage port map(t6(2), t6(10), W16(1), y16(2), y16(10));
    BS1644 : Butterfly_Stage port map(t6(3), t6(11), W16(5), y16(3), y16(11));
    BS1645 : Butterfly_Stage port map(t6(4), t6(12), W16(2), y16(4), y16(12));
    BS1646 : Butterfly_Stage port map(t6(5), t6(13), W16(6), y16(5), y16(13));
    BS1647 : Butterfly_Stage port map(t6(6), t6(14), W16(3), y16(6), y16(14));
    BS1648 : Butterfly_Stage port map(t6(7), t6(15), W16(7), y16(7), y16(15));

end STR;
```

Testbench Code

```
LIBRARY ieee;
LIBRARY work;
USE ieee.std_logic_1164.ALL;
USE work.Complex_Package.ALL;

ENTITY tb_FFT IS
END tb_FFT;

ARCHITECTURE behavior OF tb_FFT IS

   -- Inputs
    signal x2 :  CMPLX_Array2;
    signal x4 :  CMPLX_Array4;
    signal x8 :  CMPLX_Array8;
```

```
15      signal x16 :  CMPLX_Array16;
16
17
18    -- Outputs
19      signal y2 :  CMPLX_Array2;
20      signal y4 :  CMPLX_Array4;
21      signal y8 :  CMPLX_Array8;
22      signal y16 :  CMPLX_Array16;
23
24  BEGIN
25
26    -- Instantiate the Unit Under Test (UUT)
27     uut2: entity  work.FFT2 PORT MAP (
28                           x2 => x2,
29                           y2 => y2
30                          );
31
32    uut4: entity  work.FFT4 PORT MAP (
33                           x4 => x4,
34                           y4 => y4
35                          );
36
37    uut8: entity  work.FFT8 PORT MAP (
38                           x8 => x8,
39                           y8 => y8
40                          );
41
42    uut16: entity  work.FFT16 PORT MAP (
43                           x16 => x16,
44                           y16 => y16
45                          );
46     -- Stimulus process
47     stim_proc2: process
48     begin
49    -- Sample inputs in time domain.
50      x2(0) <= (1.0,0.0);
51      x2(1) <= (2.0,0.0);
52
53      x4(0) <= (4.0,0.0);
54      x4(1) <= (3.0,0.0);
55      x4(2) <= (2.0,0.0);
56      x4(3) <= (1.0,0.0);
57
58      x8(0) <= (1.0,0.0);
59      x8(1) <= (2.0,0.0);
60      x8(2) <= (3.0,0.0);
61      x8(3) <= (4.0,0.0);
62      x8(4) <= (5.0,0.0);
63      x8(5) <= (6.0,0.0);
64      x8(6) <= (7.0,0.0);
65      x8(7) <= (8.0,0.0);
66
67      x16(0) <= (1.0,0.0);
68      x16(1) <= (2.0,0.0);
69      x16(2) <= (3.0,0.0);
70      x16(3) <= (4.0,0.0);
71      x16(4) <= (5.0,0.0);
```

```
72      x16(5) <= (6.0,0.0);
73      x16(6) <= (7.0,0.0);
74      x16(7) <= (8.0,0.0);
75      x16(8) <= (9.0,0.0);
76      x16(9) <= (10.0,0.0);
77      x16(10) <= (11.0,0.0);
78      x16(11) <= (12.0,0.0);
79      x16(12) <= (13.0,0.0);
80      x16(13) <= (14.0,0.0);
81      x16(14) <= (15.0,0.0);
82      x16(15) <= (16.0,0.0);
83        wait;
84    end process;
85
86
87
88 END;
```

4 OUTPUT / RESULT

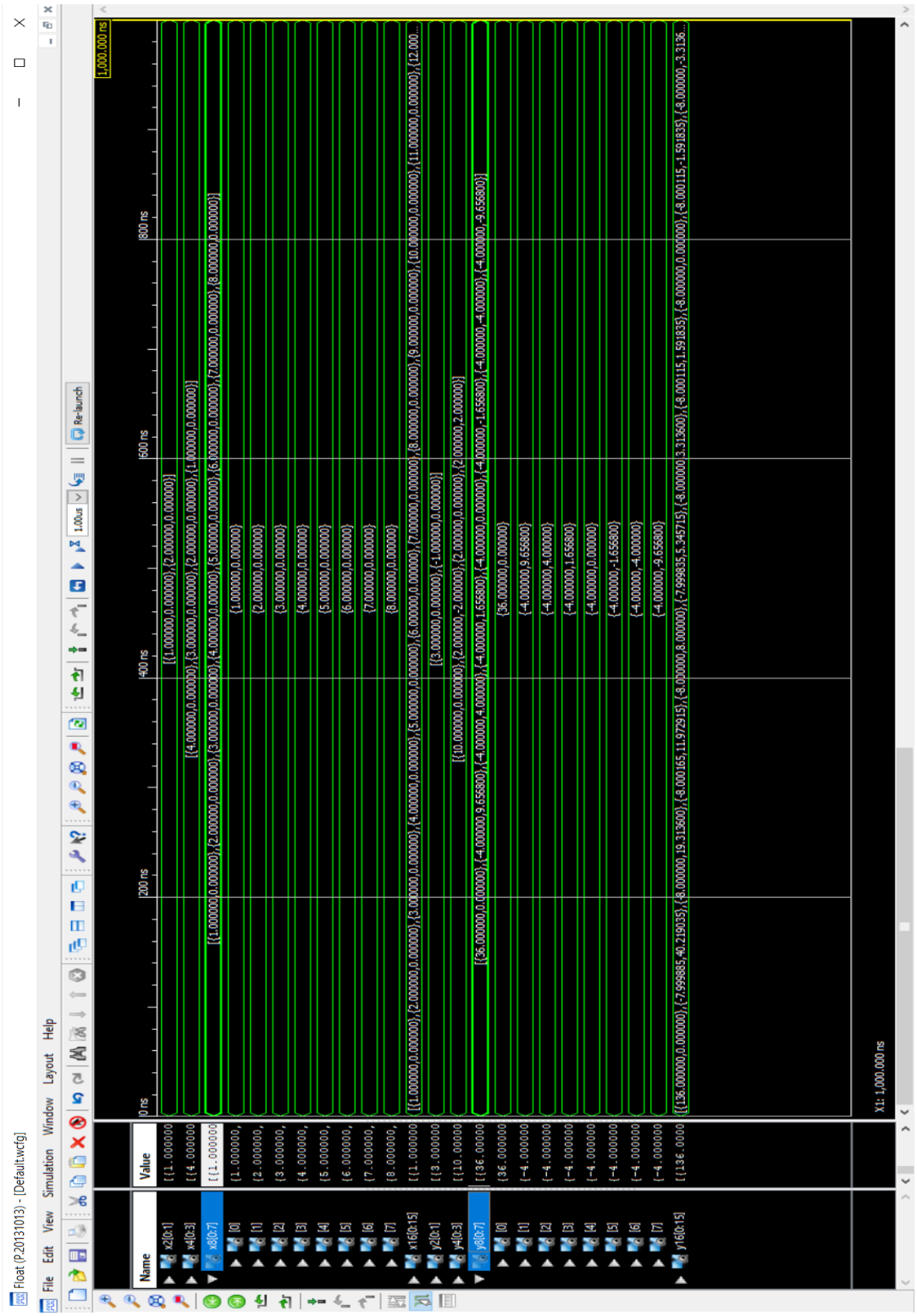

Figure 2: OUTPUT 1.1 (8-point FFT)

5 CONCLUSION

The simulation of the Cooley-Tukey Decimation in Time Radix 2 FFT algorithm is successfully observed on Xilinx ISE 14.7 Software. The simulation is carried out for 2, 4, 8 and 16 points bit reversed input. Though the VHDL code is not synthesis able as we had used the MATH_REAL package to define the complex number structure.

6 REFERENCE

- VHDL Documentation
  `https://www.csee.umbc.edu/portal/help/VHDL/`
- VHDL Language Reference Guide
  `http://vhdl.renerta.com/`
- Comlex Numbers in VHDL (Forum Disscusion)
  `https://www.edaboard.com/showthread.php?330113-Complex-number-in-vhdl`