Visvesvaraya National Institute of Technology (VNIT), Nagpur

Digital Hardware Design (ECP313)

Lab Report

Submitted by :
Sushant Jha (BT19ECE108)
Omkar Gupta (BT19ECE036)
Shah Sushlok Ajay (BT19ECE101)
Sourav Pungati (BT19ECE090)
Semester VI
Group No (19)

Submitted to :
Dr. Anamika Singh
(Lab Instructor)
Department of Electronics and Communication Engineering,
VNIT Nagpur

Contents

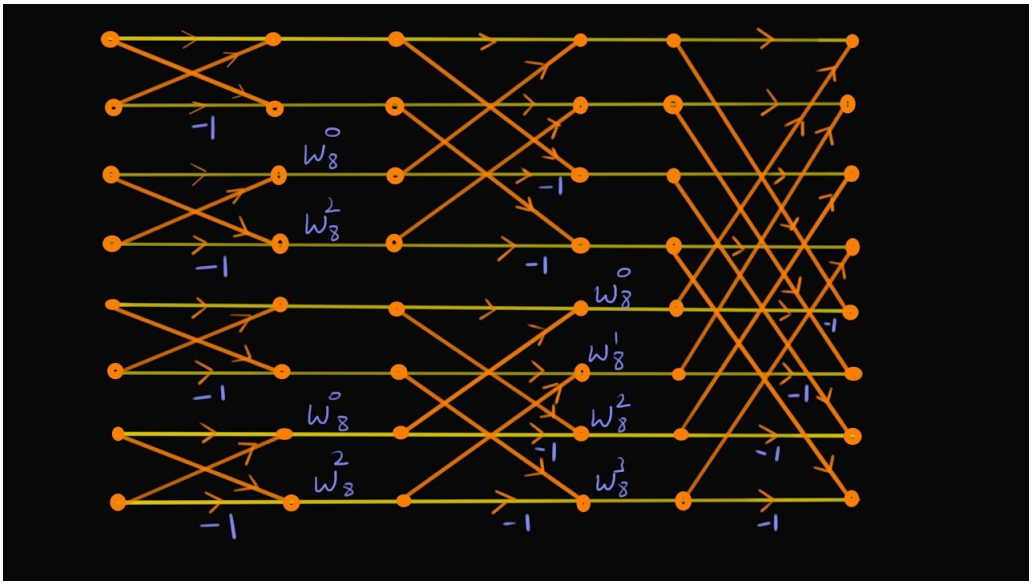
1 **8 point FFT using VHDL** 2
- 1.1 Theory 2
- 1.2 Code details 2
- 1.3 Codes 3
- 1.4 Testbench 6
- 1.5 RTL Schematic 7
- 1.6 Simulation Waveform 7
- 1.7 Applications 8
- 1.8 Conclusion 8

8 point FFT using VHDL

1.1 Theory: A fast Fourier transform (FFT) is an algorithm that calculates the discrete Fourier transform (DFT) of the given sequence – the discrete Fourier transform is a tool to convert specific types of sequences of functions into other types of representations. Another way to explain discrete Fourier transform is that it transforms the structure of the cycle of a waveform into sine components.

A fast Fourier transform can be used in various types of signal processing. It may be useful in reading things like sound waves, or for any image-processing technologies. A fast Fourier transform can be used to solve various types of equations, or show various types of frequency activity in useful ways.

As an extremely mathematical part of both computing and electrical engineering, fast Fourier transform and the DFT are largely the province of engineers and mathematicians looking to change or develop elements of various technologies. For example, fast Fourier transform might be helpful in sound engineering, seismology or in voltage measurements.



1.2 Code details: We have used 'real' data type for taking inputs and presenting output hence this code is not synthesizable directly on the hardware however, this gives us a model which can be extended to work on the hardware. We have used 'math-real' library to perform the calculations.

The code have been divided into three pieces:

1. fft-pkg: A package containing all the functions for complex related calculations.
2. butterfly: Contains the butterfly structure which is the heart of the fft.
3. fft: Main code which includes all others and performs I/O operations.
4. Testbench: For simulating the code.

1.3 Codes:

```
--fft_pkg.vhd
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.MATH_REAL.ALL;

package fft_pkg is

type complex is
    record
        r : real;
        i : real;
    end record;

type comp_array is array (0 to 7) of complex;
type comp_array2 is array (0 to 3) of complex;

function add (n1,n2 : complex) return complex;
function sub (n1,n2 : complex) return complex;
function mult (n1,n2 : complex) return complex;

end fft_pkg;

package body fft_pkg is

--addition of complex numbers
function add (n1,n2 : complex) return complex is

variable sum : complex;
```

```
begin
sum.r:=n1.r + n2.r;
sum.i:=n1.i + n2.i;
return sum;
end add;

--subtraction of complex numbers.
function sub (n1,n2 : complex) return complex is

variable diff : complex;

begin
diff.r:=n1.r - n2.r;
diff.i:=n1.i - n2.i;
return diff;
end sub;

--multiplication of complex numbers.
function mult (n1,n2 : complex) return complex is

variable prod : complex;

begin
prod.r:=(n1.r * n2.r) - (n1.i * n2.i);
prod.i:=(n1.r * n2.i) + (n1.i * n2.r);
return prod;
end mult;

end fft_pkg;

------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.math_real.all;
library work;
use work.fft_pkg.all;

entity fft8 is
port(s: in comp_array; --input signal in time domain
       y : out comp_array --output signal in F domain
```

```
        );
end fft8;

architecture Behavioral of fft8 is
component butterfly is
        port(
        s1,s2 : in complex; -- inputs
        w : in complex; --phase factor
        g1, g2 : out complex --output
        );
end component;

signal g1, g2: comp_array:= (others=> (0.0,0.0));
--phase factor, w_N = e^(-j*2*pi*i/N); and i has range from o to 7;
constant w : comp_array2 := ((1.0,0.0), (0.7071, -0.7071), (0.0,-1.0),(-0.7071,-0.7
begin
--first stage of butterfly's
bf11 : butterfly port map(s(0), s(4), w(0), g1(0), g1(1));
bf12 : butterfly port map(s(2), s(6), w(0), g1(2), g1(3));
bf13 : butterfly port map(s(1), s(5), w(0), g1(4), g1(5));
bf14 : butterfly port map(s(3), s(7), w(0), g1(6), g1(7));

--2nd stage
bf21 : butterfly port map(g1(0), g1(2), w(0), g2(0), g2(2));
bf22 : butterfly port map(g1(1), g1(3), w(2), g2(1), g2(3));
bf23 : butterfly port map(g1(4), g1(6), w(0), g2(4), g2(6));
bf24 : butterfly port map(g1(5), g1(7), w(2), g2(5), g2(7));

--3rd stage
bf31 : butterfly port map(g2(0), g2(4), w(0), y(0), y(4));
bf32 : butterfly port map(g2(1), g2(5), w(1), y(1), y(5));
bf33 : butterfly port map(g2(2), g2(6), w(2), y(2), y(6));
bf34 : butterfly port map(g2(3), g2(7), w(3), y(3), y(7));

end Behavioral;

----------------------------------------------------------
--butterfly.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
library work;
use work.fft_pkg.ALL;

entity butterfly is
   port(
      s1,s2 : in complex;       --inputs
      w :in complex;       -- phase factor
      g1,g2 :out complex       -- outputs
   );
end butterfly;

architecture Behavioral of butterfly is

begin

--butterfly equations.
g1 <= add(s1,mult(s2,w));
g2 <= sub(s1,mult(s2,w));

end Behavioral;

----------------------------------------------------------
```

1.4 Testbench:

```
--testbench for running FFT
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
library work;
use work.fft_pkg.all;

ENTITY tb IS
END tb;

ARCHITECTURE behavior OF tb IS
   signal s,y : comp_array;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: entity work.fft8 PORT MAP (
```

```
            s => s,
               y => y
         );

   -- Stimulus process
   stim_proc: process
   begin
    --sample inputs in time domain.
      s(0) <= (-1.8,0.2);
        s(1) <= (1.9,1.0);
        s(2) <= (0.4,3.0);
        s(3) <= (-1.0,-1.2);
        s(4) <= (4.4,-1.4);
        s(5) <= (-0.2,0.2);
        s(6) <= (0.7,1.0);
        s(7) <= (-4.0,-1.1);
      wait;
   end process;

END;
```

1.5 RTL Schematic: The code doesn't simulate because it is not synthesizable and hence we get this error.

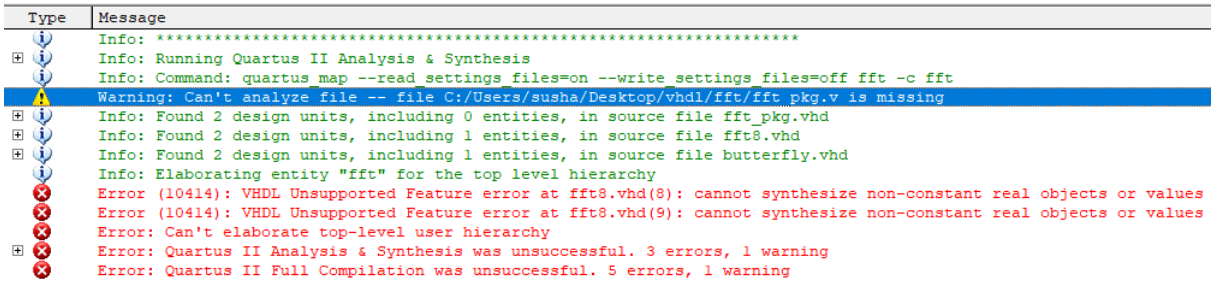
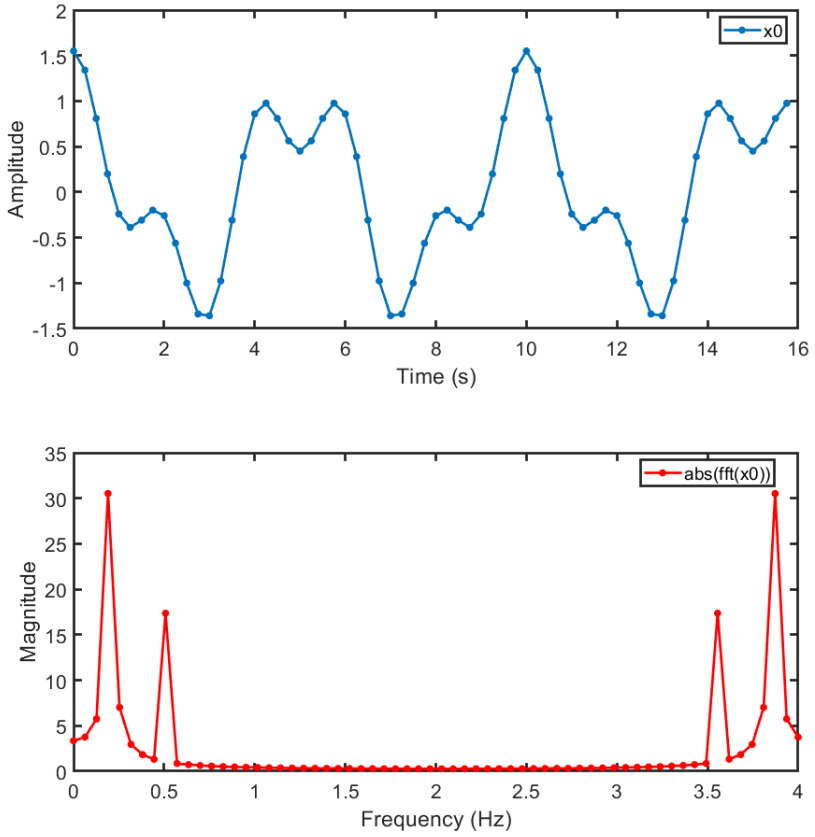


Figure 1: Error on compiling the vhdl code.

1.6 Simulation Waveform: Since the code doesn't compile we do not get a waveform. But we have written an appropriate Testbench in case we can somehow be able to run this unsynthesizable code.

This is the desired FFT waveform:



1.7 Applications: The FFT is used in , sampling, additive synthesis and pitch correction software.

- digital recording,
- sampling,
- additive synthesis and,
- pitch correction software.

1.8 Conclusion: Simulation of 8 point FFT was done on VHDL. Advantage of an FFT is speed, which it gets by decreasing the number of calculations needed to analyze a waveform.