

Lab 4:
Design, implementation and testing of a Non-
pipelined 8 Point FFT algorithm using
butterfly structures.

Katya Thorup
Lab Partner(s): Adam Lipson, Michael Eve

Date: 3/9, 3/23, 3/30

EE 26 - Digital Logic Systems

Lab Section: Monday 4:30 pm - 6:30 pm

Tufts University School of Engineering

Introduction

Disclaimer: Introduction copied from lab 3 report because the content is the same for both labs

Abbreviations:

DFT: Discrete Fourier Transform

FFT: Fast Fourier Transform

DIT: Decimation in Time

DFT is a process that transform a finite set of points observed in time domain into a set of point in frequency domain. For instance, in time domain the sine wave is shown to change over time and is written as $\sin(t)$. In frequency domain, the sin function will be represented in the form of its amplitude and phase. It might not seem such a big deal to find the DFT of a sin wave. That is because it is very easy to see the frequency and amplitude of the sine wave in time domain. However, what if the analysis had to performed on a piece of audio recording instead of a simple wave. Most likely the frequency and amplitude of the signal will be indistinguishable. On the other hand, taking the DFT of the audio signal will decompose it into its components, showing the frequencies and their amplitudes present in the signal. This is very important when attempting to process an analog signal on a digital device. Digital devices cannot process analog signals because of their encoding. Instead, they must convert it into a set of discrete points which are then analyzed by software. In order to obtain an accurate representation of our analog signal, the digital devices must sample the input at the frequency that is at least twice the highest frequency seen in the original signal. This is called the Nyquist-Shannon sampling theorem. The FFT is a mathematical tool that is used to analyze the sample points to determine the DFT.

There are several ways of implementing the FFT, one of which is DIT. DIT is the process of breaking down the sample date obtained in the time domain into its smaller and smaller parts until the DFT is obtained. DIT is implemented with the butterfly structure. The mathematical representation of the butterfly structure is:

$$A = x + jX \text{ and } B = y + jY$$

$$W_N^k = \cos\left(\frac{2\pi k}{N}\right) - j\sin\left(\frac{2\pi k}{N}\right)$$

$$A' = A + B * W_N^k \text{ and } B' = A - B * W_N^k$$

j is an imaginary unit, N represents the number of obtained time domain sample points, and $k = 0 \dots \log_2(N)$

In order to determine the complete FFT, a set of $N/2$ butterfly structures are implemented for $\log_2 N$ stages. The output of the 1st stage is passed to the 2nd stage and so on. The output propagates through the stages until the final output is produced. The diagram of the single butterfly structure and the full FFT structure are shown below. In addition, the output is displayed through a 7 Segment-Display, whose diagram is also shown below.

Figure 1: Single Butterfly Structure

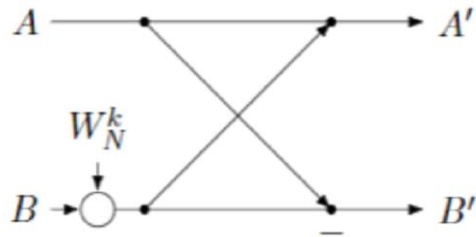


Figure 2: 8-Point FFT Structure

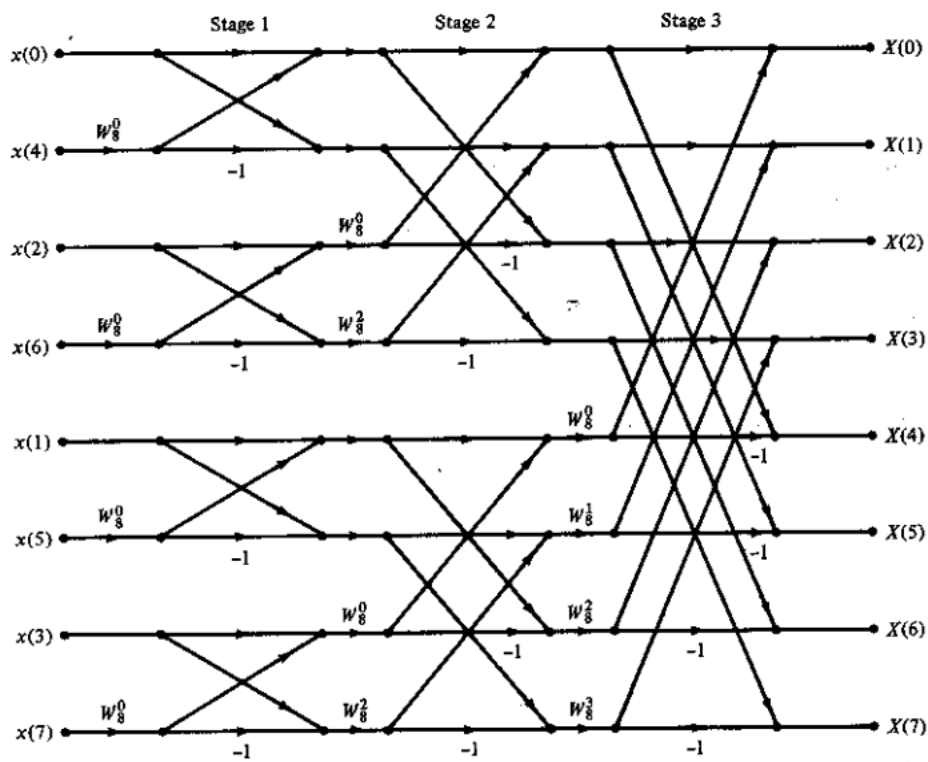


Figure 3: Seven-Segment Display

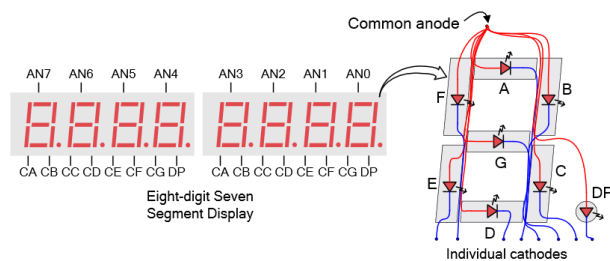
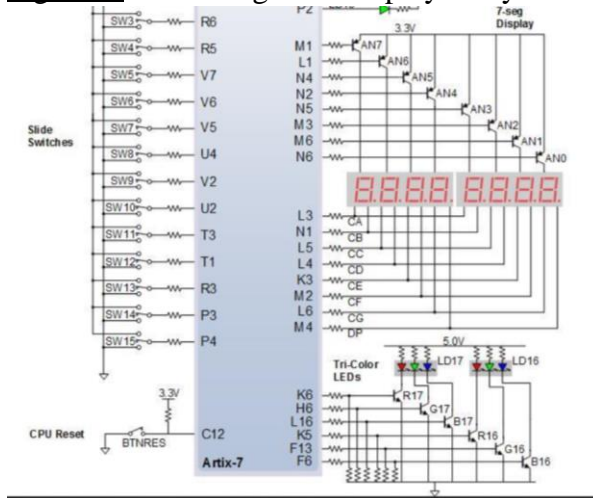


Figure 18. Common anode circuit node

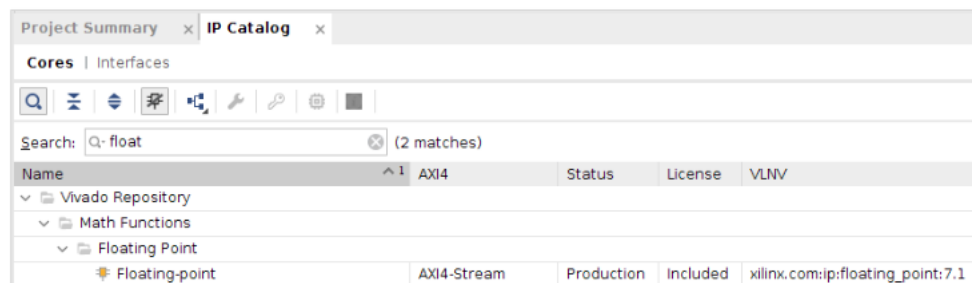
Figure 4: Seven-Segment Display Nexys 4 Board Connections



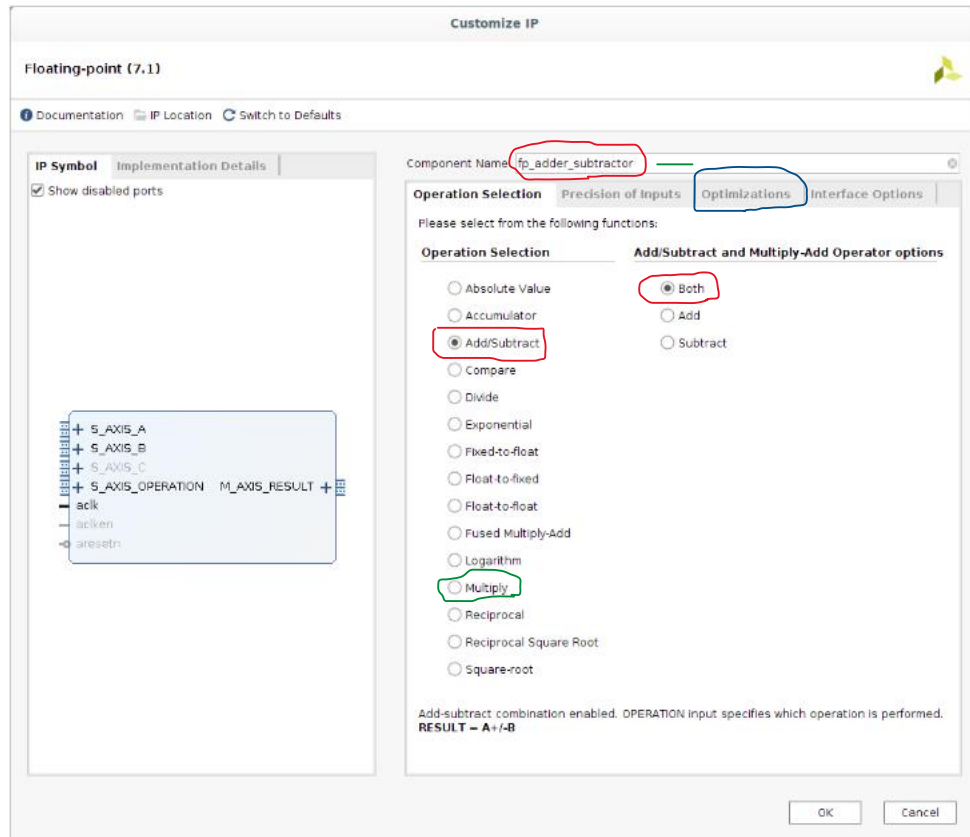
Experimental Setup

- 1) Create project in Xilinx Vivado desktop application
 - a. Open Vivado, go to File>Create New Project. Click Next.
 - b. Choose a project name and project location. Click Next.
 - c. Choose RTL Project and click Next.
 - d. Change both Target language and Simulator language to VHDL. Click Next.
 - e. Continue clicking Next until you see the Default Part dialog.
 - f. Choose xc7a100tcsg324-1 as your device type.
 - g. Click Finish to finish creating your new project.
- 2) Create a Design Source file for the 8-point FFT module
 - a. In Project Manager on the left pane, click Add Source.
 - b. Choose Add or create design sources. Click Next.
 - c. Click Create File and name it. Then click Finish.
 - d. In the Define Module dialog, choose your entity name and architecture name (Behavioral). Then click OK.
 - e. Double click the 8-point FFT file in the Sources pane and write the code to implement the FFT
 - i. Inputs are: 3-bit selector for which computed FFT point to display, 1-bit selector to display either imaginary or real component, and 1-bit clock because the 7-Segment display and the floating-point IP need the clock
 - ii. Outputs are: 1-bit to indicate whether the point on 7-segment display is on or off, 8-bit vector for the anodes of the 7-segment display, and 7-bit vector for the cathodes of the 7-segment display
 - iii. Include fft_butterfly and display_butterfly from lab 3 as components
 - iv. Create an array type of length 8 holding complex values (sample_array). Define 3 different signals with that type.

- v. Create an array type of length 4 holding complex values (w_array). Define 2 different signals with that type.
 - vi. Initialize 1 of the 3 sample_array signals with the input data points. Initialize 1 of the 2 w_array signals with values of W
 - vii. Define 3 different generate statements (1 for each stage) that will use 4 butterfly structures each. The last stage will produce the desired output Y_i
 - viii. Map the signals between the generate statements as shown in figure 2.
 - ix. Display the final output on the 7-Segment Display based on the selector values
- 3) From lab 3 add as a design source the fft_butterfly file, the complex_pkg file, and the display_butterfly file.
- a. Modify the fft_butterfly file to work with the floating-point IP, and also to accept W and clock as input
 - b. Modify the display_butterfly file to accept 1 complex number as input and to map to 7-segment display either the imaginary or real part of that number as indicated by the selector (also input to display_butterfly file)
 - c. Modify the complex_pkg file to have the real and imaginary component as a standard logic vector of length 32
- 4) Add Floating Point IP (steps given by Zengxu Yang)
- a. In the Vivado application menu, click Window → IP Catalog to open the IP Catalog panel.
 - b. Enter float in the **search** field to find the **Floating-Point IP**



- c. Double click on the Floating-point IP to open Customize IP dialog.
- d. In Operation Selection, choose Add/Subtract (in red)







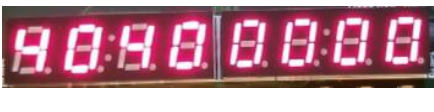



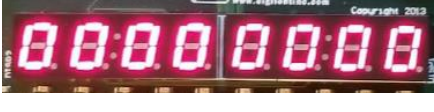

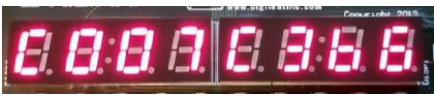
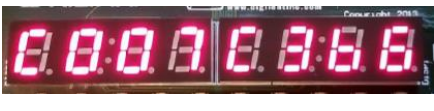




- e. In Add/Subtract and Multiply-Add Operator options choose Both (In red)
- f. In Component Name, change the name to be fp_adder_subtractor. (in red) In Optimizations options (in blue) change the usage to medium. Click OK, Generate to add the fp_adder_subtractor IP to your project design sources
- g. Repeat the last step, but change Operation Selection to Multiply and Component Name to fp_multiplier. (in green)

























Additional Notes:

1. The Optimization is set to medium because otherwise each butterfly structure will use 24 DSP cells. There are 3 stages, each using 4 butterflies. Therefore, in total the implementation would need 288 DSP cells, but the board only has 240 cells. Without the medium optimization setting, the program will run out of resources and the implementation will fail. Initially, the program was not optimized to medium. To avoid running out of resources the last stage only calculated 2 output points (based on the selector value). This cut out the use of 3 butterflies, which equates to 72 DSP cells. Thus, the board was able to handle a fully optimized program. In order to expend the use of the FFT beyond 8 points, the optimization needs to be set to medium.
2. To simulate the program using the Xilinx Behavioral Simulation a testbench file was created. This file connected to the 8-point FFT program. 2 output signals were added to the 8-point FFT file. These output signals were mapped to the real and imaginary part of

one of the points generated in the last FFT stage. The point was selected using a 3-bit selector signal. In addition to making the testbench file, the Simulation Reset Clock IP had to be added to the program in order to provide a clock signal from the testbench to the 8-point FFT program. The IP was added in the same fashion as the Floating-Point IP. It was then added as a component to the testbench file.

Results

7-Segment Display Results				
	Input	Output		
		Expected	Real	Imaginary
x_0/X_0	1	$2 + j3$		
x_1/X_1	$j2$	$2.121 + j2.121$		
x_2/X_2	0	$3 + j0$		
x_3/X_3	0	$2.707 - j0.707$		
x_4/X_4	0	$0 - j1$		
x_5/X_5	0	$-2.121 - j2.121$		
x_6/X_6	$1j$	$-1 - j2$		
x_7/X_7	1	$1.29 + j0.707$		

Behavioral Simulation					
X ₀	<div> <div>>  valueR_test[31:0]</div> <div>40000000</div> <div>40000000</div> </div> <div> <div>>  valueI_test[31:0]</div> <div>40400000</div> <div>40400000</div> </div> <div> <div>>  an_test[7:0]</div> <div>fe</div> <div>fe</div> </div> <div> <div>>  c_test[6:0]</div> <div>01</div> <div>01</div> </div>				
X ₁	<div> <div>>  valueR_test[31:0]</div> <div>00000000</div> <div>00000000</div> </div> <div> <div>>  valueI_test[31:0]</div> <div>00000000</div> <div>00000000</div> </div> <div> <div>>  an_test[7:0]</div> <div>fe</div> <div>fe</div> </div> <div> <div>>  c_test[6:0]</div> <div>01</div> <div>01</div> </div>				
X ₂	<div> <div>>  valueR_test[31:0]</div> <div>3f800000</div> <div>3f800000</div> </div> <div> <div>>  valueI_test[31:0]</div> <div>bf800000</div> <div>bf800000</div> </div> <div> <div>>  an_test[7:0]</div> <div>fe</div> <div>fe</div> </div> <div> <div>>  c_test[6:0]</div> <div>01</div> <div>01</div> </div>				
X ₃	<div> <div>>  valueR_test[31:0]</div> <div>402d413d</div> <div>402d413d</div> </div> <div> <div>>  valueI_test[31:0]</div> <div>bf3504f3</div> <div>bf3504f3</div> </div> <div> <div>>  an_test[7:0]</div> <div>fe</div> <div>fe</div> </div> <div> <div>>  c_test[6:0]</div> <div>42</div> <div>42</div> </div>				
X ₄	<div> <div>>  valueR_test[31:0]</div> <div>00000000</div> <div>00000000</div> </div> <div> <div>>  valueI_test[31:0]</div> <div>bf800000</div> <div>bf800000</div> </div> <div> <div>>  an_test[7:0]</div> <div>fe</div> <div>fe</div> </div> <div> <div>>  c_test[6:0]</div> <div>01</div> <div>01</div> </div>				
X ₅	<div> <div>>  valueR_test[31:0]</div> <div>00000000</div> <div>00000000</div> </div> <div> <div>>  valueI_test[31:0]</div> <div>00000000</div> <div>00000000</div> </div> <div> <div>>  an_test[7:0]</div> <div>fe</div> <div>fe</div> </div> <div> <div>>  c_test[6:0]</div> <div>01</div> <div>01</div> </div>				

X ₆	<div> <div>> valueR_test[31:0]</div> <div>3f800000</div> </div> <div> <div>> valueI_test[31:0]</div> <div>bf800000</div> </div> <div> <div>> an_test[7:0]</div> <div>fe</div> </div> <div> <div>> c_test[6:0]</div> <div>01</div> </div>	<div>3f800000</div> <div>bf800000</div> <div>fe</div> <div>01</div>		
X ₇	<div> <div>> valueR_test[31:0]</div> <div>3fa57d86</div> </div> <div> <div>> valueI_test[31:0]</div> <div>3f3504f3</div> </div> <div> <div>> /fft_testbench/an_test[7:0]</div> <div>fe</div> </div> <div> <div>> c_test[6:0]</div> <div>20</div> </div>	<div>3fa57d86</div> <div>3f3504f3</div> <div>fe</div> <div>20</div>		

Analysis

The results from the simulation were obtained after the results from the Seven-Segment Display. The results from the 7-Segment display are correct, while the results from the Behavioral Simulation are not. valueR_test represents the real part of the produced FFT point, and valueI_test represents the imaginary part of the produced FFT point. This can be attributed to the fact that the Behavioral Simulation uses a different clock than the 8-point FFT does when displaying onto the 7-Segment display. The clock in the testbench, which is a clock from the Simulation Reset Clock IP, must not provide enough time for the 8-point FFT to produce the correct results. That is why the Behavioral Simulation results do not match the expected values. Each value is produced by generating only 1 output at the time. Therefore, technically, the testbench can take as much time as it needs to generate the results. However, it seems that the testbench might grab the results from the 8-point FFT file before the results are produced. Another indication of this is that the values of an_test (7-Segment display anode output of 8-point FFT) and c_test (7-Segment display cathode output of 8-point FFT) never change even though the values valueR and valueI are not produced until the display_butterfly is called.

Besides understanding how to create generate statements, to map the same component in a loop, big part of this laboratory was also using IP's. The previous labs did not require the use of an IP. Therefore, for this lab additional time had to be taken to read about the IPs, understand how to initiate them and how to incorporate them into design source files. In addition, this lab required an understanding of DSP cells to know how to resolve the board resource problem.

Conclusion

The purpose of this lab was met, which was to properly implement the 8-point FFT structure. Although the Behavioral Simulation results does not confirm this, the discrepancy can be attributed to the used of a very different clock. The 7-Segment display produces results that match the expected output. Since the 7-segment display uses the FPGA board clock, versus an IP clock, these results must be the correct results.

APPENDIX

8-point FFT

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

library IEEE_Proposed;
use IEEE_Proposed.float_pkg.all;

use work.complex_record.all;

entity fft_8point is
    Port ( num : in STD_LOGIC_VECTOR (2 downto 0);
          part_fft : in STD_LOGIC;
          clk : in STD_LOGIC;
          dp_fft : out STD_LOGIC;
          valueR : out STD_LOGIC_VECTOR(31 downto 0);
          valueI : out STD_LOGIC_VECTOR(31 downto 0);
          an_fft : out STD_LOGIC_VECTOR (7 downto 0);
          c_fft : out STD_LOGIC_VECTOR (6 downto 0));
end fft_8point;

architecture Behavioral of fft_8point is

    component fft_butterfly is
        Port ( A : in complex;
              B : in complex;
              W : in complex;
              clk : in STD_LOGIC;
              A_comp : out complex;
              B_comp : out complex);
    end component;

    component display_butterfly is
        Port ( numIn : in complex;
              part : in STD_LOGIC;
              clk : in STD_LOGIC;
              C : out STD_LOGIC_VECTOR (6 downto 0);
              DP : out STD_LOGIC;
              AN : out STD_LOGIC_VECTOR (7 downto 0));
    end component;

    type w_array is array(3 downto 0) of complex;
    signal W : w_array;
    signal w_stage3 : w_array;

    type sample_array is array(7 downto 0) of complex;
    signal x_in : sample_array;
    signal stage1, stage2, x_out: sample_array;

    signal value: complex;

begin
    -- Initializing the input signal
    x_in(0).r <= std_logic_vector(to_float(1, 8, 23)); x_in(0).i <=
    "00000000000000000000000000000000";
```

```

x_in(1).r <= "00000000000000000000000000000000"; x_in(1).i <= std_logic_vector(to_float(2, 8,
23));
x_in(2).r <= "00000000000000000000000000000000"; x_in(2).i <= "00000000000000000000000000000000";
x_in(3).r <= "00000000000000000000000000000000"; x_in(3).i <= "00000000000000000000000000000000";
x_in(4).r <= "00000000000000000000000000000000"; x_in(4).i <= "00000000000000000000000000000000";
x_in(5).r <= "00000000000000000000000000000000"; x_in(5).i <= "00000000000000000000000000000000";
x_in(6).r <= "00000000000000000000000000000000"; x_in(6).i <= std_logic_vector(to_float(1, 8,
23));
x_in(7).r <= std_logic_vector(to_float(1, 8, 23)); x_in(7).i <=
"00000000000000000000000000000000";

W(0).r <= std_logic_vector(to_float(1, 8, 23)); W(0).i <= "00000000000000000000000000000000";
W(1).r <= std_logic_vector(to_float(0.7071067812, 8, 23)); W(1).i <= std_logic_vector(to_float(-
0.7071067812, 8, 23));
W(2).r <= "00000000000000000000000000000000"; W(2).i <= std_logic_vector(to_float(-1, 8, 23));
W(3).r <= std_logic_vector(to_float(-0.7071067812, 8, 23)); W(3).i <= std_logic_vector(to_float(-
0.7071067812, 8, 23));

w_stage3 <= W;
w_stage3(1) <= w(2);
w_stage3(2) <= w(1);

stage1_gen:
for i in 0 to 3 generate
    butterfly1: fft_butterfly port map(A => x_in(i),
        B => x_in(i + 4),
        W => W(0),
        clk => clk,
        A_comp => stage1(i),
        B_comp => stage1(i + 4));
end generate stage1_gen;

stage2_gen1:
for i in 0 to 1 generate
    butterfly2: fft_butterfly port map(A => stage1(i),
        B => stage1(i + 2),
        W => W(0),
        clk => clk,
        A_comp => stage2(i),
        B_comp => stage2(i + 2));
end generate stage2_gen1;

stage2_gen2:
for i in 4 to 5 generate
    butterfly2: fft_butterfly port map(A => stage1(i),
        B => stage1(i + 2),
        W => W(2),
        clk => clk,
        A_comp => stage2(i),
        B_comp => stage2(i + 2));
end generate stage2_gen2;

stage3_gen:
for i in 0 to 3 generate
    butterfly3: fft_butterfly port map(A => stage2(2 * i),
        B => stage2((2 * i) + 1),
        W => w_stage3(i),
        clk => clk,
        A_comp => x_out(2 * i),
        B_comp => x_out((2 * i) + 1));
end generate stage3_gen;

value <= x_out(0) when num = "000" else
    x_out(1) when num = "001" else
    x_out(2) when num = "010" else
    x_out(3) when num = "011" else

```

```

        x_out(4) when num = "100" else
        x_out(5) when num = "101" else
        x_out(6) when num = "110" else
        x_out(7) when num = "111";

display : display_butterfly port map(numIn => value,
                                   part => part_fft,
                                   clk => clk,
                                   C => c_fft,
                                   DP => dp_fft,
                                   AN => an_fft);

valueR <= value.r; valueI <= value.i;
end Behavioral;

```

Display_butterfly

```

library IEEE_Proposed;
use IEEE_Proposed.float_pkg.all;

use work.complex_record.all;

entity display_butterfly is
    Port ( numIn : in complex;
          part : in STD_LOGIC;
          clk : in STD_LOGIC;
          C : out STD_LOGIC_VECTOR (6 downto 0);
          DP : out STD_LOGIC;
          AN : out STD_LOGIC_VECTOR (7 downto 0));
end display_butterfly;

architecture Behavioral of display_butterfly is

    component sevenseg is
        Port ( num : in STD_LOGIC_VECTOR (31 downto 0);
              clk : in STD_LOGIC;
              point : in STD_LOGIC_VECTOR (7 downto 0);
              C_out : out STD_LOGIC_VECTOR (6 downto 0);
              DP : out STD_LOGIC;
              AN_out : out STD_LOGIC_VECTOR (7 downto 0));
    end component;

    signal to_display : STD_LOGIC_VECTOR (31 downto 0);

    begin
        to_display <= (numIn.r) when part = '0' else
                      (numIn.i) when part = '1';

        display : sevenseg port map(num => to_display,
                                   clk => clk,
                                   point => "00000000",
                                   C_out => C,
                                   DP => DP,
                                   AN_out => AN);

    end Behavioral;

```

fft_butterfly

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- Uncomment the following library declaration if using

```

```

-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
library IEEE_Proposed;
use IEEE_Proposed.float_pkg.all;
use work.complex_record.all;

entity fft_butterfly is
    Port ( A : in complex;
           B : in complex;
           W : in complex;
           clk : std_logic;
           A_comp : out complex;
           B_comp : out complex);
end fft_butterfly;

architecture Behavioral of fft_butterfly is

type num_array is array (3 downto 0) of std_logic_vector(31 downto 0);
signal add_in1, add_in2, add_out, sub_in1, sub_in2, sub_out, mul_in1, mul_in2,
mul_out : num_array;
component fp_adder_subtractor IS
    PORT (
        aclk : IN STD_LOGIC;
        s_axis_a_tvalid : IN STD_LOGIC;
        s_axis_a_tready : OUT STD_LOGIC;
        s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        s_axis_b_tvalid : IN STD_LOGIC;
        s_axis_b_tready : OUT STD_LOGIC;
        s_axis_b_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        s_axis_operation_tvalid : IN STD_LOGIC;
        s_axis_operation_tready : OUT STD_LOGIC;
        s_axis_operation_tdata : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        m_axis_result_tvalid : OUT STD_LOGIC;
        m_axis_result_tready : IN STD_LOGIC;
        m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
END component;

component fp_multiplier IS
    PORT (
        aclk : IN STD_LOGIC;
        s_axis_a_tvalid : IN STD_LOGIC;
        s_axis_a_tready : OUT STD_LOGIC;
        s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        s_axis_b_tvalid : IN STD_LOGIC;
        s_axis_b_tready : OUT STD_LOGIC;
        s_axis_b_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        m_axis_result_tvalid : OUT STD_LOGIC;
        m_axis_result_tready : IN STD_LOGIC;
        m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
END component;

begin
addition_gen:
for i in 0 to 3 generate
    addition: fp_adder_subtractor port map (aclk => clk,
        s_axis_a_tvalid => '0',
        s_axis_a_tready => open,
        s_axis_a_tdata => add_in1(i),
        s_axis_b_tvalid => '0',
        s_axis_b_tready => open,

```

```

s_axis_b_tdata => add_in2(i),
s_axis_operation_tvalid => '0',
s_axis_operation_tready => open,
s_axis_operation_tdata => "00000000",
m_axis_result_tvalid => open,
m_axis_result_tready => '0',
m_axis_result_tdata => add_out(i));
end generate addition_gen;

subtraction_gen:
for i in 0 to 3 generate
    subtraction: fp_adder_subtractor port map (aclk => clk,
s_axis_a_tvalid => '0',
s_axis_a_tready => open,
s_axis_a_tdata => sub_in1(i),
s_axis_b_tvalid => '0',
s_axis_b_tready => open,
s_axis_b_tdata => sub_in2(i),
s_axis_operation_tvalid => '0',
s_axis_operation_tready => open,
s_axis_operation_tdata => "00000001",
m_axis_result_tvalid => open,
m_axis_result_tready => '0',
m_axis_result_tdata => sub_out(i));
end generate subtraction_gen;

multiplication_gen:
for i in 0 to 3 generate
    multiplication: fp_multiplier port map (aclk => clk,
s_axis_a_tvalid => '0',
s_axis_a_tready => open,
s_axis_a_tdata => mul_in1(i),
s_axis_b_tvalid => '0',
s_axis_b_tready => open,
s_axis_b_tdata => mul_in2(i),
m_axis_result_tvalid => open,
m_axis_result_tready => '0',
m_axis_result_tdata => mul_out(i));
end generate multiplication_gen;
--A_comp.r <= A.r + (B.r * W.r) - (B.i * W.i);
--A_comp.i <= A.i + (B.i * W.r) + (B.r * W.i);
--B_comp.r <= A.r - (B.r * W.r) + (B.i * W.i);
--B_comp.i <= A.i - (B.i * W.r) - (B.r * W.i);
A_comp.r <= add_out(0);
add_in1(0) <= A.r;
add_in2(0) <= sub_out(0); -- A_comp.r <= A.r + sub_out(0)
sub_in1(0) <= mul_out(0);
mul_in1(0) <= B.r;
mul_in2(0) <= W.r; -- sub_in1(0) <= B.r * W.r;
sub_in2(0) <= mul_out(1);
mul_in1(1) <= B.i;
mul_in2(1) <= W.i; -- sub_in2(0) <= B.i * W.i;
A_comp.i <= add_out(1);
add_in1(1) <= A.i;
add_in2(1) <= add_out(2); -- A_comp.i <= A.i + add_out(2)
add_in1(2) <= mul_out(2);
mul_in1(2) <= B.i;
mul_in2(2) <= W.r; -- add_in1(2) <= B.i * W.r;
add_in2(2) <= mul_out(3);
mul_in1(3) <= B.r;
mul_in2(3) <= W.i; -- add_in2(2) <= B.r * W.i;
B_comp.r <= sub_out(1);
sub_in1(1) <= A.r;
sub_in2(1) <= sub_out(2); -- B_comp.r <= A.r - sub_out(2)

sub_in1(2) <= mul_out(0); -- sub_in1(2) <= B.r * W.r;

```

```

sub_in2(2) <= mul_out(1); -- sub_in2(0) <= B.i * W..i;
B_comp.i <= sub_out(3);
sub_in1(3) <= A.i;
sub_in2(3) <= add_out(3); -- B_comp.i <= A.i - add_out(3)
add_in1(3) <= mul_out(2); -- add_in1(3) <= B.i * W.r;
add_in2(3) <= mul_out(3); -- add_in2(3) <= B.r * W..i;
end Behavioral;

```

fft_testbench

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

library IEEE_Proposed;
use IEEE_Proposed.float_pkg.all;

use work.complex_record.all;

entity fft_testbench is
-- Port ( );
end fft_testbench;

architecture Behavioral of fft_testbench is

component fft_8point is
    Port ( num : in STD_LOGIC_VECTOR (2 downto 0);
          part_fft : in STD_LOGIC;
          clk : in STD_LOGIC;
          dp_fft : out STD_LOGIC;
          valueR : out STD_LOGIC_VECTOR(31 downto 0);
          valueI : out STD_LOGIC_VECTOR(31 downto 0);
          an_fft : out STD_LOGIC_VECTOR (7 downto 0);
          c_fft : out STD_LOGIC_VECTOR (6 downto 0));
end component;

COMPONENT clk_gen_sim_0
    PORT (
        axi_clk_in_0 : IN STD_LOGIC;
        axi_rst_in_0_n : IN STD_LOGIC;
        axi_clk_0 : OUT STD_LOGIC;
        axi_rst_0_n : OUT STD_LOGIC
    );
END COMPONENT;

signal clk_test : STD_LOGIC;
signal dp_test : STD_LOGIC;
signal valueR_test : STD_LOGIC_VECTOR(31 downto 0);
signal valueI_test : STD_LOGIC_VECTOR(31 downto 0);
signal an_test : STD_LOGIC_VECTOR(7 downto 0);
signal c_test : STD_LOGIC_VECTOR(6 downto 0);

begin

dut_clk : clk_gen_sim_0
    PORT MAP (

```



```
    axi_clk_in_0 => '1',
    axi_rst_in_0_n => '0',
    axi_clk_0 => clk_test
);

dut : fft_8point port map(num => "111",
    part_fft => '0',
    dp_fft => dp_test,
    clk => clk_test,
    valueR => valueR_test,
    valueI => valueI_test,
    an_fft => an_test,
    c_fft => c_test);

end Behavioral;
```