Seth Pregler
ECEN 489-508
02.25.22

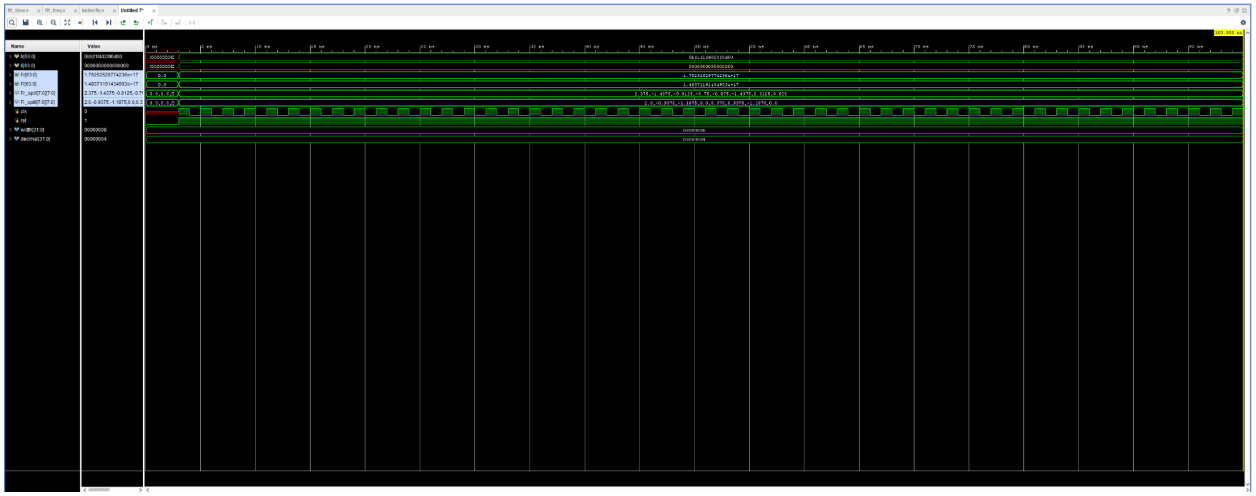
Lab 4 Post-lab Submission

Screenshots of the terminal output of the radix-2 decimation in time FFT design:

```
zynq> ./lab4_fft_time_test
send data to FPGA
-------time domain---------
signal 0
:0: 0.000000+0.000000i
1: 0.841471+0.000000i
2: 0.909297+0.000000i
3: 0.141120+0.000000i
4: -0.756802+0.000000i
5: -0.958924+0.000000i
6: -0.279415+0.000000i
7: 0.656987+0.000000i
signal 1
:0: 1.000000+0.000000i
1: 1.000000+0.000000i
2: 1.000000+0.000000i
3: 1.000000+0.000000i
4: 0.000000+0.000000i
5: 0.000000+0.000000i
6: 0.000000+0.000000i
7: 0.000000+0.000000i
signal 2
:0: 0.000000+0.000000i
1: 0.250000+0.000000i
2: 0.500000+0.000000i
3: 0.750000+0.000000i
4: 1.000000+0.000000i
5: 0.750000+0.000000i
6: 0.500000+0.000000i
7: 0.250000+0.000000i
```

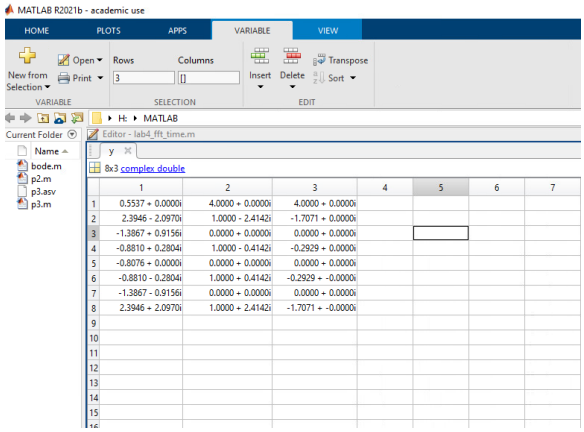
```
trigger FPGA
wait for FPGA
read back the results
-------frequency domain---------
signal 0:
0: 0.500000+0.000000i
1: 2.312500-2.062500i
2: -1.375000+0.875000i
3: -0.812500+0.187500i
4: -0.750000+0.000000i
5: -0.812500-0.187500i
6: -1.375000-0.875000i
7: 2.312500+2.062500i
signal 1:
0: 4.000000+0.000000i
1: 1.000000-2.375000i
2: 0.000000+0.000000i
3: 1.000000-0.375000i
4: 0.000000+0.000000i
5: 1.000000+0.375000i
6: 0.000000+0.000000i
7: 1.000000+2.375000i
signal 2:
0: 4.000000+0.000000i
1: -1.687500-0.062500i
2: 0.000000+0.000000i
3: -0.312500-0.062500i
4: 0.000000+0.000000i
5: -0.312500+0.062500i
6: 0.000000+0.000000i
7: -1.687500+0.062500i
```

Screenshot of the behavioral simulation of the radix-2 decimation in freq FFT design:



The figure shown above, displays the behavioral simulation for the radix-2 decimation in frequency FFT. In the testbench, we pass as input a series of points from a discrete Sine wave. The result is a series of Fourier coefficients with both a real and imaginary part.

Lastly, I verified the results from the behavioral simulation with the outputs generated from the Matlab test file. The image below displays this output; the first column is the Fourier coefficients and begins with F_0 at the top and ends with F_7 at the bottom. Our results are within the error threshold of 0.1 so we can conclude that the implementation was a success. Note that this error is due to the limited number of bits used in our computation.



Output of Matlab Test

Appendix

Terminal output of the radix-2 decimation in freq FFT design

```
The physical address is 0x43c00000
The virtual address is 0x608e0000
Registered a device with dynamic Major number of 245
Create a device file for this device with this command:
'mknod /dev/transfpga c 245 0'.
zynq> mknod /dev/transfpga c 245 0
zynq> ./mem_test
send data to FPGA
read back the results
0->1
1->3
2->5
3->7
4->9
5->11
6->13
7->15
8->17
9->19
zynq> ./lab4_fft_time_test
-/bin/ash: ./lab4_fft_time_test: not found
zynq> ./lab4_fft_freq_test
send data to FPGA
-------time domain---------
signal 0
:0: 0.000000+0.000000i
1: 0.841471+0.000000i
2: 0.909297+0.000000i
3: 0.141120+0.000000i
4: -0.756802+0.000000i
5: -0.958924+0.000000i
6: -0.279415+0.000000i
7: 0.656987+0.000000i
signal 1
:0: 1.000000+0.000000i
1: 1.000000+0.000000i
2: 1.000000+0.000000i
3: 1.000000+0.000000i
4: 0.000000+0.000000i
5: 0.000000+0.000000i
6: 0.000000+0.000000i
7: 0.000000+0.000000i
signal 2
:0: 0.000000+0.000000i
1: 0.250000+0.000000i
2: 0.500000+0.000000i
3: 0.750000+0.000000i
4: 1.000000+0.000000i
5: 0.750000+0.000000i
6: 0.500000+0.000000i
7: 0.250000+0.000000i
trigger FPGA
wait for FPGA
read back the results
-------frequency domain---------
signal 0:
0: 0.500000+0.000000i
1: 2.250000-1.125000i
2: -1.375000+0.875000i
3: -0.812500+0.250000i
4: -0.750000+0.000000i
5: -0.750000-1.125000i
6: -1.375000-0.875000i
7: 2.312500+2.000000i
signal 1:
0: 4.000000+0.000000i
1: 1.000000-1.000000i
2: 0.000000+0.000000i
3: 1.000000-0.375000i
4: 0.000000+0.000000i
5: 1.000000-1.000000i
6: 0.000000+0.000000i
7: 1.000000+2.375000i
signal 2:
0: 4.000000+0.000000i
1: -1.750000+0.000000i
2: 0.000000+0.000000i
3: -0.312500+0.000000i
4: 0.000000+0.000000i
5: -0.250000+0.000000i
6: 0.000000+0.000000i
7: -1.687500+0.000000i
zynq>
```

fft_freq.v

```
`timescale 1ns / 1ps

module fft_freq(
clk,
rst,
fr,     // Real part of the 8 inputs.
fi,     // Imag part of the 8 inputs.
Fr,     // Real part of the 8 outputs.
Fi      // Imag part of the 8 outputs.
    );

parameter width=8;
parameter decimal=4;

input clk,rst;

// fr: the real part of input
// fi: the imag part of input
// inputReal[i]=fr[(i+1)*width-1:i*width]
// inputImag[i]=fi[(i+1)*width-1:i*width]
input [8*width-1:0] fr,fi;

// Fr: the real part of output
// Fi: the imag part of output
// outputReal[i]=Fr[(i+1)*width-1:i*width]
// outputImag[i]=Fi[(i+1)*width-1:i*width]
output reg [8*width-1:0] Fr,Fi;

// Wires for the outputs.
wire[8*width-1:0] Fwr,Fwi;

// Outputs of level 1
wire[width-1:0] o0r[7:0];
wire[width-1:0] o0i[7:0];

// Outputs of level 2
wire[width-1:0] o1r[7:0];
wire[width-1:0] o1i[7:0];


// Start of your code
// (1<<decimal) --> real part of weight input
//------------------------------level 1--------------------------//
butterfly_freq #(.width(width),.decimal(decimal)) bt0(fr[1*width-1:0*width],fi[1*width-1:0*width],fr[5*width-1:4*width],fi[5*width-1:4*width],
(1<<decimal),0,o0r[0],o0i[0],o0r[4],o0i[4]);
butterfly_freq #(.width(width),.decimal(decimal)) bt1(fr[2*width-1:1*width],fi[2*width-1:1*width],fr[6*width-1:5*width],fi[6*width-1:5*width], 11,
-11,o0r[1],o0i[1],o0r[5],o0i[5]);
butterfly_freq #(.width(width),.decimal(decimal)) bt2(fr[3*width-1:2*width],fi[3*width-1:2*width],fr[7*width-1:6*width],fi[7*width-1:6*width], 0, -16
,o0r[2],o0i[2],o0r[6],o0i[6]);
butterfly_freq #(.width(width),.decimal(decimal)) bt3(fr[4*width-1:3*width],fi[4*width-1:3*width],fr[8*width-1:7*width],fi[8*width-1:7*width], -11,
-11,o0r[3],o0i[3],o0r[7],o0i[7]);

//------------------------------level 2--------------------------//
butterfly_freq #(.width(width),.decimal(decimal)) bt4(o0r[0],o0i[0],o0r[2],o0i[2],(1<<decimal),0,o1r[0],o1i[0],o1r[2],o1i[2]);
butterfly_freq #(.width(width),.decimal(decimal)) bt5(o0r[1],o0i[1],o0r[3],o0i[3],0,-16,o1r[1],o1i[1],o1r[3],o1i[3]);
butterfly_freq #(.width(width),.decimal(decimal)) bt6(o0r[4],o0i[4],o0r[6],o0i[6],(1<<decimal),0,o1r[4],o1i[4],o1r[6],o1i[6]);
butterfly_freq #(.width(width),.decimal(decimal)) bt7(o0r[5],o0i[5],o0r[7],o0i[7],0,-16,o1r[5],o1i[5],o1r[7],o1i[7]);

//------------------------------level 3--------------------------//
butterfly_freq #(.width(width),.decimal(decimal))
bt8(o1r[0],o1i[0],o1r[1],o1i[1],(1<<decimal),0,Fwr[1*width-1:0*width],Fwi[1*width-1:0*width],Fwr[5*width-1:4*width],Fwi[5*width-1:4*width]);
butterfly_freq #(.width(width),.decimal(decimal))
bt9(o1r[2],o1i[2],o1r[3],o1i[3],(1<<decimal),0,Fwr[3*width-1:2*width],Fwi[3*width-1:2*width],Fwr[7*width-1:6*width],Fwi[7*width-1:6*width]);
butterfly_freq #(.width(width),.decimal(decimal))
bt10(o1r[4],o1i[4],o1r[5],o1i[65],(1<<decimal),0,Fwr[2*width-1:1*width],Fwi[2*width-1:1*width],Fwr[6*width-1:5*width],Fwi[6*width-1:5*width]);
butterfly_freq #(.width(width),.decimal(decimal))
bt11(o1r[6],o1i[6],o1r[7],o1i[7],(1<<decimal),0,Fwr[4*width-1:3*width],Fwi[4*width-1:3*width],Fwr[8*width-1:7*width],Fwi[8*width-1:7*width]);

always@(posedge clk or negedge rst)begin
    if(~rst)begin
        Fr<=0;
        Fi<=0;
    end
    else begin
        Fr<=Fwr;
        Fi<=Fwi;
    end
end
// End of your code

endmodule
```

butterfly.v

```
`timescale 1ns / 1ps

module butterfly_time(
Fer,    // Real part of the even input.
Fei,    // Imag part of the even input.
For,    // Real part of the odd input.
Foi,    // Imag part of the odd input.
Wr,     // Real part of the weight input.
Wi,     // Imag part of the weight input.
o0r,    // Real part of output 0.
o0i,    // Imag part of output 0.
o1r,    // Real part of output 1.
o1i     // Imag part of output 1.
    );

parameter width=8;
parameter decimal=4;

input[width-1:0] Fer,Fei,For,Foi,Wr,Wi;
output[width-1:0] o0r,o0i,o1r,o1i;

wire[7:0] m0,m1,m2,m3,mr,mi;
// multiplication of complex numbers: m = Fo*W
multiply #(.width(width),.decimal(decimal)) mp0(For,Wr,m0);
multiply #(.width(width),.decimal(decimal)) mp1(For,Wi,m1);
multiply #(.width(width),.decimal(decimal)) mp2(Foi,Wr,m2);
multiply #(.width(width),.decimal(decimal)) mp3(Foi,Wi,m3);

assign mr=m0-m3;
assign mi=m1+m2;

assign o0r=Fer+mr;
assign o0i=Fei+mi;

assign o1r=Fer-mr;
assign o1i=Fei-mi;

endmodule


module butterfly_freq(
f0r,    // Real part of input 0.
f0i,    // Imag part of input 0.
f1r,    // Real part of input 1.
f1i,    // Imag part of input 1.
Wr,     // Real part of the weight input.
Wi,     // Imag part of the weight input.
o0r,    // Real part of output 0.
o0i,    // Imag part of output 0.
o1r,    // Real part of output 1.
o1i     // Imag part of output 1.
    );

parameter width=8;
parameter decimal=4;

input[width-1:0] f0r,f0i,f1r,f1i,Wr,Wi;
output[width-1:0] o0r,o0i,o1r,o1i;

// Start of your code
assign o0r = f0r + f1r;
assign o0i = f0i + f1i;

wire[7:0]sub01r, sub01i;
// (X + jY) term in multiplication with W_Ni
assign sub01r = f0r - f1r;
assign sub01i = f0i - f1i;

// ****** Implementation of Eff. Complex Mult. ******
// (X + jY) (C + jS) = XC - YS (real), YC + XS (imag)
// G-H = (X + jY), sub01r = X, sub01i = Y
// W_Ni = (C + jS), Wr = C, Wi = S

wire[7:0] XC, YS, YC, XS;
// multiplication of complex numbers: m = (G-H)*W
multiply #(.width(width),.decimal(decimal)) mp0(sub01r, Wr ,XC);
multiply #(.width(width),.decimal(decimal)) mp1(sub01i, Wi, YS);
multiply #(.width(width),.decimal(decimal)) mp2(sub01i, Wr, YC);
multiply #(.width(width),.decimal(decimal)) mp3(sub01r, Wi, XS);
assign o1r = XC - YS;
assign o1i = YC + XS;
endmodule
```