Signoffs and Grade:

Name:_____

Component	Signoff	Date	Time
Low Pass Filter Graph Matches provided example			
High Pass Filter graph Matches provided example			

==========

Component	Received	Possible
Signoffs		100
Penalties • after the first 15 minutes of lab session 9: -10 • after the first 15 minutes of "open session": -25 no signoffs after 5/5 at 4:25pm	-	
Total		100

Educational Objective

The educational objective of this lab is to investigate the creation of digital signal processing circuitry using VHDL and the implementation of these algorithms in the resources of the Cyclone V FPGA.

Technical Objective

The technical objective of this laboratory is to design and verify both a high pass and a low pass FIR filter in VHDL. These filters will be used to process an audio signal in the next lab.

Background

Filtering is a process of adjusting a signal – for example, removing noise. Noise in a sound waveform is represented by small, but frequent changes to the amplitude of the signal. A simple logic circuit that achieves the task of noise-filtering is an averaging Finite Impulse Response (FIR) filter. A schematic diagram of the filters for this lab is given in figure 1.

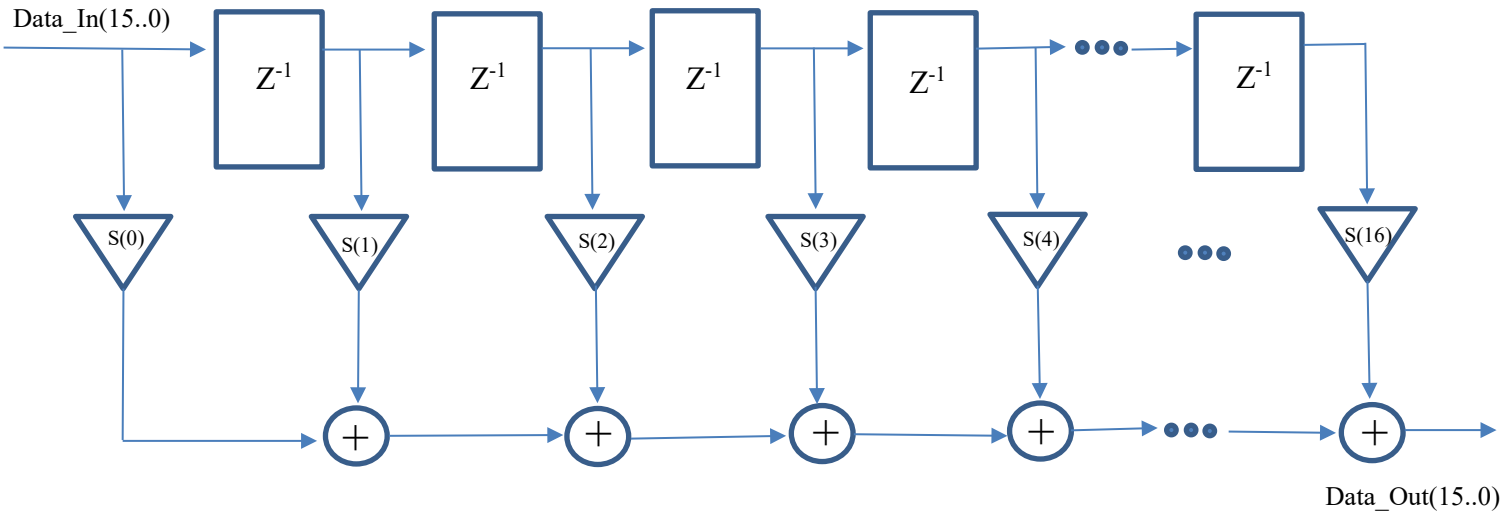


Figure 1: Schematic for simple averaging FIR filter

- Each Z^{-1} block represents an enabled register that only latches its input data when its enable signal is active
- Each triangle represents a multiplier that multiplies the input signal with the corresponding coefficient (see table below).
- Each circle represents an adder

For this filter, the input and output data signals are signed 16 bit fixed-point numbers

The filter design will be the same for both the high-pass and the low-pass filter. The difference between the two is in the coefficients, as seen in table 1.

	Low Pass			High Pass	
Coefficient	Value	16-bit fixed point		Value	16-bit fixed point
S(0)	0.0025			0.0019	
S(1)	0.0057			-0.0031	
S(2)	0.0147			-0.0108	
S(3)	0.0315			0.0	
S(4)	0.0555			0.0407	
S(5)	0.0834			0.0445	
S(6)	0.1099			-0.0807	
S(7)	0.1289			-0.2913	
S(8)	0.1358			0.5982	
S(9)	0.1289			-0.2913	
S(10)	0.1099			-0.0807	
S(11)	0.0834			0.0445	
S(12)	0.0555			0.0407	
S(13)	0.0315			0.0	
S(14)	0.0147			-0.0108	
S(15)	0.0057			-0.0031	
S(16)	0.0025			0.0019	

Table 1: Coefficients for high and low pass filter in decimal

The entity for both filters will be the following:

```
entity low_pass_filter is
Port (clk, reset_n          : in std_logic;
      filter_en             : in std_logic;
      data_in               : in std_logic_vector(15 downto 0);
      data_out              : out std_logic_vector(15 downto 0));
end filter;
```

Below is a brief description of the required ports.

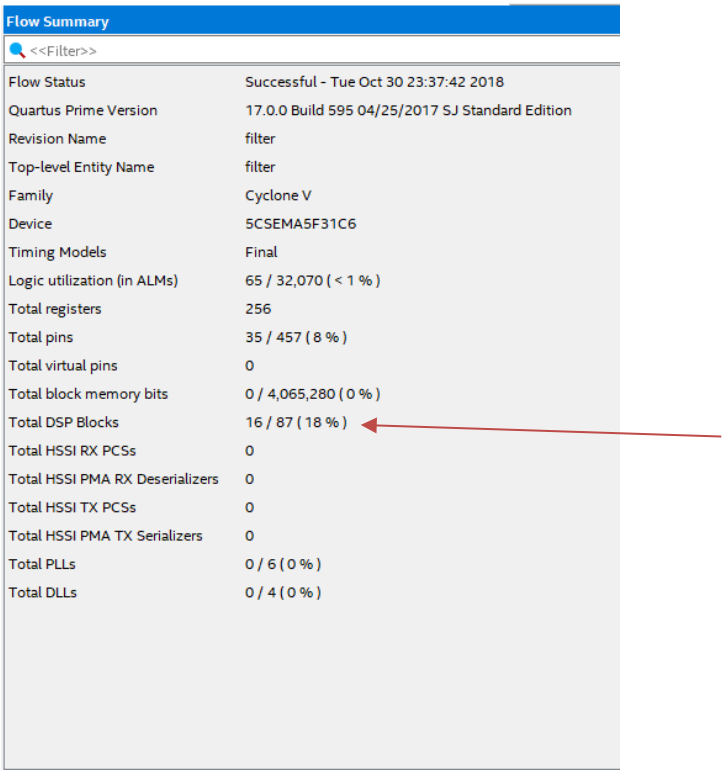
- **Data_in** is the audio sample. It is a 16 bit signed fixed point number with 15 bits after the implied decimal point. This data will come from a .wav file.
- **Filter_en** is a positive going, one 50 MHz clock wide pulse that comes from the Digital to Analog Converter on every sample period. The filter calculation is done once for every pulse on the Filter_en signal. This signal will enable the registers

- **Data_out** is the output of your digital filter. This signal is a 16 bit signed fixed point number with 15 bits after the implied decimal point.

Additional notes on designing the filter are provided in the PowerPoint presentation titled “Lab 8 Guide” found in MyCourses. It is suggested that you follow this guide carefully.

Procedure

1. Create a new Quartus Prime Project for the 5CSEMA5F31C6 FPGA and the DE1-SoC board.
2. Write the VHDL for the low pass filter
3. Follow the instructions in the Lab 8 Guide for creating the multiplier component. Use the LPM_MULT instead of * for multiplication
4. Write the VHDL for the high pass filter
5. Compile each filter. Inspect the compilation report and verify that the synthesizer maps your code to the DSP blocks.



Flow Summary

<<Filter>>

Flow Status	Successful - Tue Oct 30 23:37:42 2018
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Standard Edition
Revision Name	filter
Top-level Entity Name	filter
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	65 / 32,070 (< 1 %)
Total registers	256
Total pins	35 / 457 (8 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	16 / 87 (18 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

6. Choose Tools > Netlist Viewers > RTL Viewer and verify the RTL schematic looks similar to figure 1 on page 1.
7. Write a testbench for Modelsim to verify your design. Your testbench will read a waveform from a file, process it through your filter, and output the filtered waveform to a file. The file containing the input waveform is called **one_cycle_200_8k**. A template for the testbench is given in Appendix A.
8. In order to verify functionality, you will plot the output using Excel. Remember that you will need to divide the integer output of your filter by 2^15 in Excel, in order to convert it to a floating point number.
9. Plot the filtered data in your output file and verify it against the expected results provided for a sign off.

Appendix A

```
stimulus : process is
  file read_file : text open read_mode is "./src/verification_src/one_cycle_integer.csv";
  file results_file : text open write_mode is "./src/verification_src/output_waveform.csv";
  variable lineIn : line;
  variable lineOut : line;
  variable readValue : integer;
  variable writeValue : integer;
begin
  wait for 100 ns;
  reset <= '1';
  -- Read data from file into an array
  for i in 0 to 39 loop
    readline(read_file, lineIn);
    read(lineIn, readValue);
    audioSampleArray(i) <= to_signed(readValue, 16);
    wait for 50 ns;
  end loop;
  file_close(read_file);

  -- Apply the test data and put the result into an output file
  for i in 1 to 10 loop
    for j in 0 to 39 loop

      -- Your code here...
      -- Read the data from the array and apply it to Data_In
      -- Remember to provide an enable pulse with each new sample

      -- Write filter output to file
      writeValue := to_integer(data_out);
      write(lineOut, writeValue);
      writeline(results_file, lineOut);

      -- Your code here...

    end loop;
  end loop;

  file_close(results_file);

  -- end simulation
  sim_done <= true;
  wait for 100 ns;

  -- last wait statement needs to be here to prevent the process
  -- sequence from restarting at the beginning
  wait;

end process stimulus;
```