ECE 260C Spring 2024 Lab2

Objectives:

1. Implement a resampling Farrow filter for a Xilinx or Intel FPGA.

2. Maximize achievable clock rate.

3. Simulate and verify all designs for arithmetic accuracy.

4. Write up briefly summarizes your results and conclusions.

Assignment:

1. Starting with the architecture shown below and the corresponding Verilog starter code and/or Simulink model provided, generate Verilog by hand, by using Vivado HLS, and/or by using Matlab/Simulink HDL Coder. Be sure to register the data and mu input ports and the data output port. These Dflipflops are in addition to the ones depicted in the diagram.

Assume 16-bit two's comp. precision at the data input port dat_i (input signed[15:0] dat_i) and at the final data output port dat_o (output signed logic[15:0] dat_o).

Fractional displacement mu (input signed[15:0] mu) is in 16-bit signed notation, with two integer bits before the binary point and 14 fractional bits after. We shall restrict mu in our model to positive values only, from 16'b00.00000000000000 = 0 to 16'b01.11111111111111 = +2.0-(2**-14). All internal products are computed to 32-bit two's complement and rounded to 16 bits.

2. Verify this initial design by simulation using the test bench provided. (Screen shot of waveform also provided.)

3. Build this base design using your choice of Quartus or Vivado, and report your maximum clock rate.

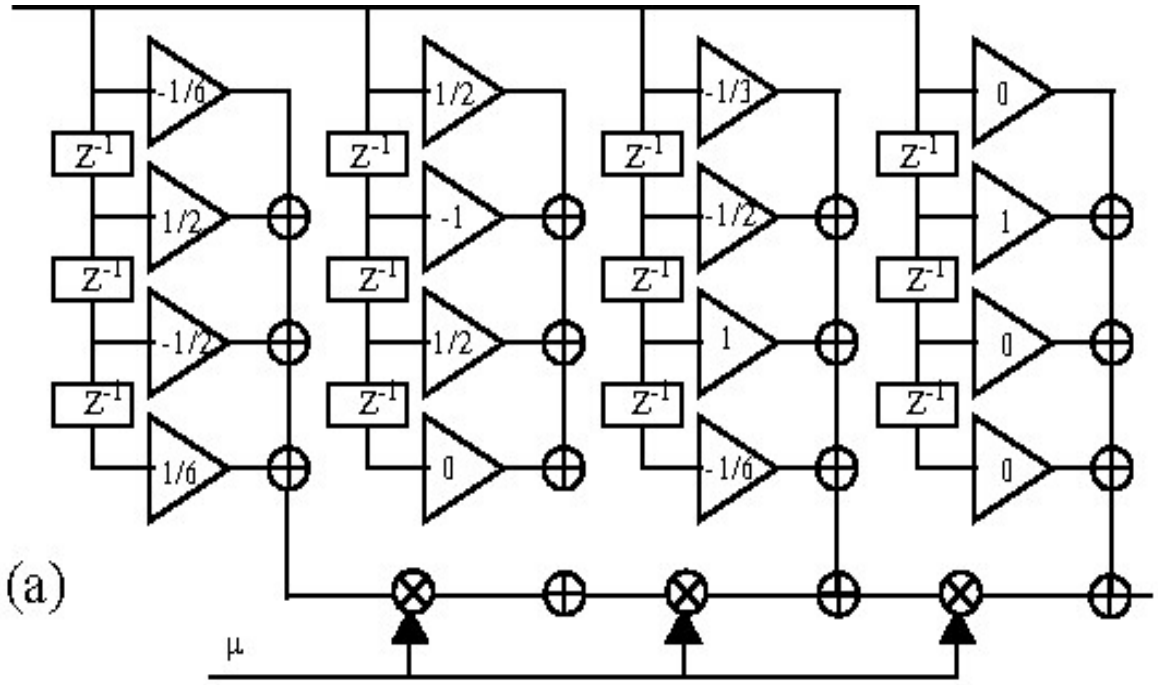
4. Now optimize the design for speed, through pipelining (strongly recommended) and/or other techniques, including transposed FIR architecture, if helpful in this case. Also consider replacing the top row multipliers by parallel adders, since all have fixed coefficients.

5. Re-verify, adjusting the test bench as needed to accommodate any added pipeline delays in your design.

6. Build your pipelined version in Quartus or Vivado and report your maximum clock rate.

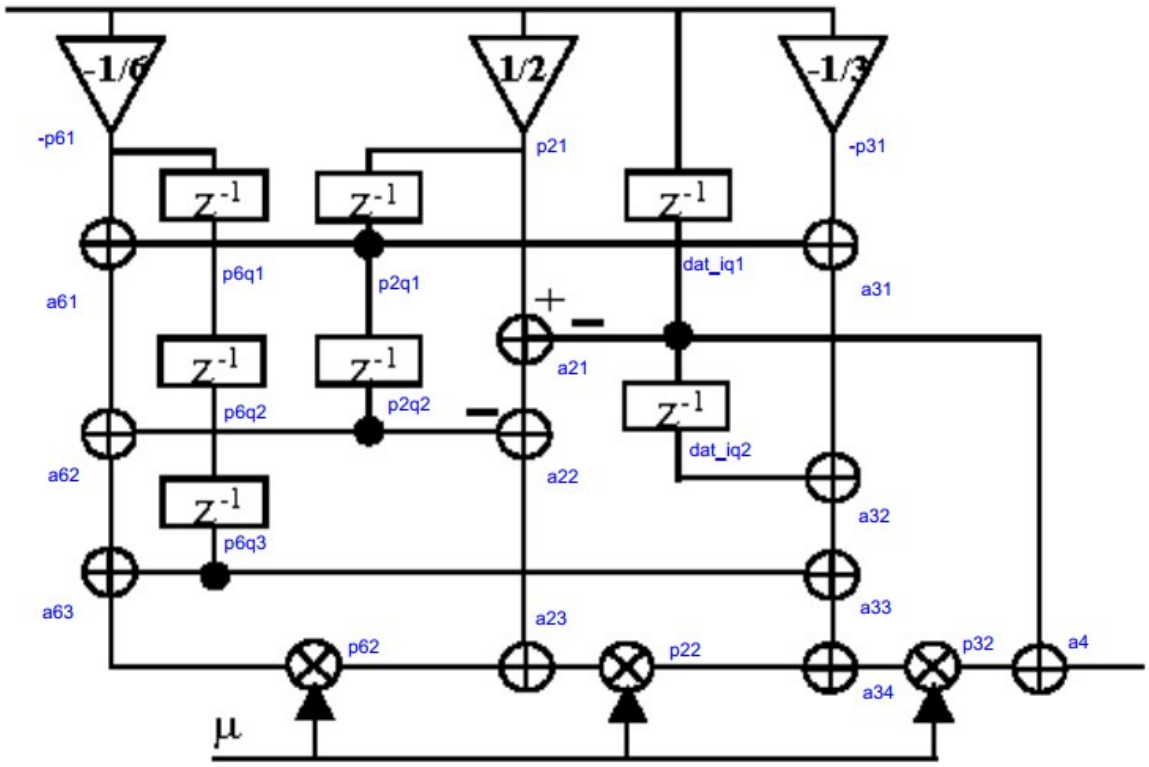
7. Briefly summarize your results and findings. What worked, what didn't? How much difference were you able to make in the clock rate?

Starting Point Block diagram. Don't forget to register all inputs and outputs, including dat_i, mu, and dat_o. Also provide synchronous reset for the various registers. You may use my Verilog code on Canvas as a starting point.



I have also generated Verilog code for a diagram which purported to perform the same function, but in operation looks more like a gain adjuster than a phase/delay adjuster. (Proof that you can't believe everything you read.) You may use my Verilog as sample code when writing your own based on the correct architecture, which is shown above.

(Aside: I declared the following internal variables in my design:)



I have also provided this Verilog code, in case anyone wants to repair this algorithm and adjust the code accordingly. You can easily write an equation for each of the four downward feeds to the bottom row, then compare notes to see where the designer of this block diagram went astray. (This is strictly optional for this course.)

Here is my Simulink of the correct architecture: