EECS 598 – 002 VLSI for Wireless Communication and Machine Learning

Homework 1

Due: 9/24 Tuesday 11:59pm, Online Canvas submission only.

Download hw1.mat from Canvas and load data using 'load hw1.mat' in Matlab. Vector 'my_fir_h' and 'my_signal' contains double-precision floating point FIR coefficients and input signal, respectively, for problem 1 – 3.

Problem 1 (20 pts)

Identify a uniform quantization with 32 levels that (with your best effort) maximizes the SQNR of the quantized input signal ('my_signal'). Explain your strategy to determine the quantized levels. Report the corresponding 32 quantization levels and the best SQNR that you achieved. Generate a figure that plots the original and quantized signal on top of each other to visualize the difference. Also plot the PDF/histogram of quantization error.

Problem 2 (25 pts)

Make your own fixed point quantization function for parameterized (N, R) representation (i.e., N and R are input parameters for your function). The input of that function is a vector/matrix of floating point values and the output of that function is a vector/matrix of integer mantissa values of the fixed point (N,R) representation.

And make your own fixed point arithmetic functions; +, -, and multiplication that take two input (integer) mantissa vectors in (N1, R1), (N2, R2) fixed point and produce the output (integer) mantissa vector in (N3, R3) fixed point. N1, N2, N3, R1, R2 and R3 are parameters for the functions.

Make sure your fixed arithmetic operations include proper saturation and truncation. Rounding is optional (floor instead rounding is fine). **Do NOT use Matlab/Python fixed point library/toolbox functions. Make your own functions.** Note that all mantissa values must be integer. Submit your code (as a file attachment).

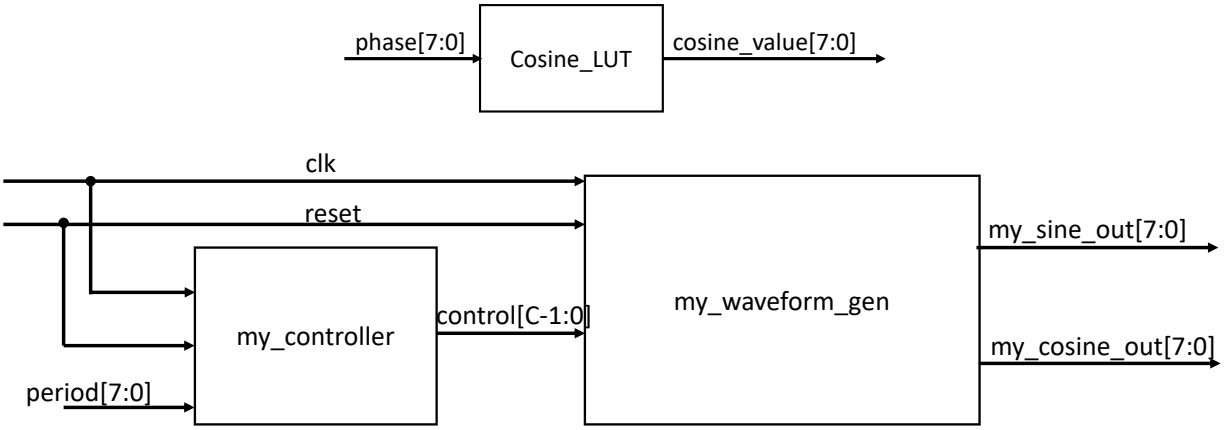
Problem 3 (20 pts)

Load hw1.mat to obtain input signals for this problem. Using the functions from Problem 2, quantize the floating point input signal to 11-bit fixed point mantissa with (11,X) representation, and quantize the FIR tabs to mantissa of 7-bit fixed point (7,Y) representation. Perform FIR filtering in fixed point using your functions from Problem 2 to obtain 10-bit filter output mantissa values for in (10,Z) representation. You are free to choose fixed point representations for intermediate results (e.g., multiplication output / adder input) but the final output should be in (10,Z). Select reasonable X,Y, and Z parameters to obtain a good SQNR. Show a plot that compares floating point (i.e., input, FIR taps and operations are all in floating point) vs. fixed point FIR results after converting the output mantissa values to real numbers

(back to floating point in Matlab). Plot the histogram of the error (i.e., the difference between the floating point output and fixed point output). Report the output SNR in dB. That is, (power of the floating point output) / (power of the error) in dB.

Problem 4 (35 pts)

Design a digital frequency synthesis logic in Verilog. The following diagram specifies the interface of the block. Design a sine lookup table (LUT) module with 256 entries that stores cos(θ_k) for $k = 0,1, \ldots, 255$ and $\theta_k = k2\pi/256$ for the phase in the range of $(0, 2\pi)$. This LUT is a combinatorial logic module (without any clock or reset) with the interface shown below. cos(θ_k) values stored in the LUT should be fixed point quantized in (8,7) representation (i.e., mantissa values are stored in the LUT). Note that the maximum positive value you can represent in this format is slightly less than 1.0. So you will need to saturate the cos($\pi/2$) to the maximum allowed mantissa value before storing it in the LUT.



Note that because sine and cosine have a certain phase relationship (cosine waveform is 90-degree phase shifted version of sine waveform), you can use the cosine LUT to generate the sine output for the same time index. Internal design of the logic is up to you as long as it is synthesizable (e.g., you can't use 'real' type variables / signal in the logic). But you design should only use the cosine LUT (i.e., use two instances of the same LUT), not a separate sine LUT. Run your logic with a 100MHz clk.

a) Write a testbench for 'my_waveform_gen' to show in simulation that your logic can generate sine and cosine waveforms with the target frequency of 1) 3.29 MHz, 2) -1.35 MHz, and 3) 0.43 MHz by providing a proper 'control' input. The design should be able to produce a negative frequency signal when the control input is negative. Make the generated frequency as close as possible to the target. Note that the proper value and bit width of the control for each frequency depends on your internal design. Explain with a mathematical expression the relationship between the control value and the actual generated frequency of your design in the homework report. Show the simulation waveform capture plots showing both sine and cosine to prove that you obtained the (close-to) desired frequency.

b) Design 'my_controller' to produce a modulated output signal that alternates the frequency between 2.3 MHz and 0.3 MHz with a period given by the input 'period', which is an 8 bit unsigned integer. period = k means that the period is k us. Show that your controller work with a period of 16 us (i.e., 2.3 MHz for 8 us and 0.3 MHz for 8 us, and then repeat).

c) Store the output of problem 4.b) above and load the samples in Matlab/Python. Note that you will need to store sine and cosine parts separately (e.g., using two columns in a text file), and then combine them into complex valued samples in Matlab/Python by assigning sine values to the real part and cosine values to the imaginary part before running FFT/DFT. Run DFT (or FFT) and plot the absolute values of the DFT/FFT result (x-axis is frequency index, y-axis is absolute value of the DFT/FFT output). Confirm that you observe two frequency tones with the correct frequency index.

Submit your HDL and Matlab/Python codes as a zip file attachment.

Note regarding Verilog simulation results plotting

Note that Verdi simulator allows plotting multi-bit signed/unsigned wire and register signals as 'analog' waveforms instead of hex or decimal values. It is helpful to visualize the waveform coming out of your design. Refer to the 'Verdi tutorial.pdf' posted with homework on Canvas. One optional method is to store the output of your simulation in a text file and then load it in Matlab / Python to analyze the signal frequency.