ELEC2221 D1 Report Form 2021/22

Name	Tzu-Yun, Chang	**Email**	tyc1g20@soton.ac.uk

Abstract (2 marks)

Finite Impulse Response (FIR) low-pass and band-pass filters were both designed and verified its functioning in the lab. They were verified by examining the simulated output waveform with different input frequencies in ModelSim and by downloading the design onto FPGA to listen with a headphone. A parameterised version of the codes with two parameters, `SIZE` and `STAGE`, was completed as instructed in extended section.

Design and Verification (3 marks)

The objective of this lab was to design a FIR filter. In the frequency domain, a Fourier transformed input wave would be multiplied by a function which only passes the desired frequency. This is a multiplication of two functions in Fourier transform. According to the convolution theorem, it states that the inverse Fourier transform of a product of functions is the convolution of the inverse transforms of the individual factors. The convolution of individual functions can be thought of as a smoothing or averaging of a signal with respect to a 'windowing' function. Its mathematical form is $(f*g)(x) = \int_{-\infty}^{\infty} f(p)g(x-p)\,dp$. In our case, the array which stores all the coefficients is the windowing function and the incoming signal is the function to be filtered. Therefore, by multiplying each sample with different coefficients and summing the product, we are essentially implementing the convolution of the input waveform with the coefficients in time domain.

In the combinational logic block of the state machine, it contains 4 stages, waiting, loading, processing and saving. When the state is at waiting stage, the `reset_accumulator` is assigned 1 and the accumulator register is reset. On the next clock cycle, if `input_ready` is set to 1, the state machine enters the next stage, loading, whereas if 0, it returns back to waiting. In the loading stage, two variables, `load` and `reset_accumulator` are assigned 1 and the input signal will be loaded to a 16-bit array called `samples`. On the next clock cycle, the state machine goes into processing stage where the `count` is set to 1. The value of the `address` continues to increase by 1 as long as the `count` is set and `address` does not exceed 15. An `always_ff` block was written to implement the increment of the `address` while `count` is 1 or otherwise reset it. When address is equal to 15. the state machine will enter saving stage on the next cycle. In this state, the `output_ready` goes high and the top 16 bits excluding the most significant bit is clocked onto a 16 bits array named `out`. Additionally, an `always_ff` block which updates the `present_state` to `next_state` was coded up. An asynchronous active high reset is present in this sequential logic block to reset the `present_state` to waiting. As it is an asynchronous and active high signal, it is considered as a `posedge` triggered parameter for the `always_ff` block.

After running the simulation for the testbench in ModelSim and verify the output was a staircase-like waveform, the file was uploaded to the online checker. The first attempt returned an error as I wrote a blocking assignment in `always_ff` block. However, after replacing it with a non-blocking assignment, the code passed.

In order to pass the input signals through the designed filter, two filters one for the left channel and the other for the right, are instantiated in the D1.sv file. Port mapping technique was used. The inputs and outputs of the filters are mapped to the top 16-bit of readdata_left, readdata_right, writedata_left, and writedata_right, respectively. Both the filters are

connected to CLOCK_50, a 50MHz clock signal, reset ,a signal triggered by Push-button[0], and read_ready, a signal produced by audio codec to indicate that the input signal is ready to be analysed. The left_write_ready and the right_write_ready signals are logically AND together to account for the write signal to cue the codec to input the signal to its system.

Quartus was used for synthesis, compilation, and downloading the code on FPGA board. The D1.qsf file was modified to include all the pin assignments for clock generators and audio CODEC interface and the design was downloaded on FPGA board. To test the design, a tone generator was used to produce the input signals at different frequencies and the output was connected to a headphone. The volume of the signals at higher frequencies were certainly reduced but could still be heard.

Extended Design (3 marks)

In the extended design, two parameters, `SIZE` and `STAGE`, are declared in the module header. The `SIZE` parameter refers to the width of the input as well as output and the `STAGE` refers to the number of multiplications that shall be carried out. To parameterise the code, some modifications were required. For example, the size of `sum` array is double the value of `SIZE` and only the top `SIZE` bits excluding the topmost bit are passed to the output of the module. Apart from this, the size of `coefficients` array is the value of `STAGE` and the size of `address` is log base 2 of `STAGE` parameter. Last change was made on the `samples` array which is affected by both parameters. It is an unpacked array with size STAGE that each contains in total `SIZE` number of packed elements. Moreover, instead of implementing a low-pass filter, I replaced it with a band-pass filter having a band-pass frequency between 2.2kHz to 6.6kHz. The coefficients for the band-pass filter were generated by this command in Matlab, `round(fir1(32,[0.1 0.3],"bandpass")*32768)`. A final change was made on the D1.sv file to input a number for the parameters.

I verified the filter in two ways. The first way, I modified the input frequency in the testbench and ran it on ModelSim. As predicted, the maximum amplitude and the trace of the output waveform were distorted when the input frequencies were around 2.2kHz and 6.6kHz, whereas frequencies within the range were operating correctly. The other way that I verified the signal was by inputting signals at different frequencies from an online tone generator to the FPGA board. Its output was connected to a headphone so that I could confirm whether the filter works. It was cleared that the filter worked because the sound was completely cutoff after certain frequency, about 10kHz.

Reflection (2 marks)

This exercise taught me the that, by attenuating incoming signals sampled at different time and summing them up, it is effectively filtering the signals at certain frequencies. The second thing that I learn from this lab is that even though ModelSim won't display an error for placing blocking assignment in a sequential block, the code will certainly fail Verilator check.

A major thing that I would like to change is the verifying methods that I used for checking the functioning of the filter. Using human ears to verify the result is undeniably unreliable as they might be more subjected to environmental factors such as noises in the room, dampness and/or the natural limitation on the frequency range of human ears. Therefore, it would be a more dependable means for verification if the output is connected directly to an oscillator where the function of the filter can be confirmed by comparing the output waveform to input waveform to see if there is a reduction in amplitude. Furthermore, using a signal generator provided in the laboratory will be a more reliable source to generate incoming signal than phone as it is well-designed for this task.