Software Studio - Project 2: FIR Filter

Anastasia Riana 邱慧莉 - 109006234

Things to know:

We use the online tool https://fiiir.com/ to generate the filter coefficients. We set the sampling rate to be 44100Hz, cutoff frequency to be 10000Hz and the transition bandwidth to be 630Hz, in Rectangular window type. The tool results in the filter yielding 65 coefficients.

Part 1: Read/Write a WAV File

First, we need to create a few structs for the RIFF chunk, FMT sub-chunk and the data subchuck. Inside the WAV header, we need to declare RIFF ID, size of RIFF and RIFF type. Next, in the FMT chunk, we need to declare FMT ID, size of FMT, format type, channels, sample rate, byte rate, block align and bit per sample. Lastly, in the data chunk struct, we declare the data ID and size. Then, we combine those structs to be the WAV header. Not only that, we also need to declare a few structs to store additional information. In this struct, we need to store the number of samples, sample sizes, duration and bytes that are in the channels. Then, we write the additional information into a struct. Lastly, we prepare the struct to output the result of the filtering.

In the main function, we start opening the WAV file using the fopen function using ‘rb’ mode. We should use "rb" because we are opening non-text files. Using the fread function, we are going to read the WAV file, then using the struct to write additional information about the WAV file that we have created before, we write the information to the WAV header.

Next, we allocate memory to the input and output samples using malloc. We also create an error variable to keep track of what went wrong with the code. Then, we convert the coefficients from float to int16 and call the FIR filter initialization. Then, we do the filtering by assigning the results of the FIR filter update function to output samples while iterating through the number of samples available in the input file additional information. Details on how the conversion was implemented and how signals are processed are in **Part 3** and **Part 2** respectively.

Onto the output part, we first declare the output struct where we have stored all the information. Then, we open the output WAV file using fopen ‘wb’ mode. In the directory, we can put any type of WAV file and rename it to **out.wav.** Using the fopen function, the file will be overwritten to be the real output file that is the processed audio. Then we get to check if there’s any error. If the value of error is equal to 1, then we successfully write the WAV header. Other than that, we also get to check if the number of samples are equal to the error value. If so, we successfully write the samples of the output audio. Lastly, we close both the input and output file and free all resources like the input and output samples.

Part 2: Input/Output Signal with Your Filter

In **Part 1**, we get to call the FIRFilter_init function that is already declared in the header file. The implementation of FIRFilter.c and FIRFilter.h are based on this resource given from the TAs https://www.youtube.com/watch?v=uNNNj9AZisM&t=1128s. In the header file, we declare a FIRFilter struct and the contents of the struct will be output, coefficient, buffer, converted filter coefficients, buffer size and count.

In the FIRFilter.c file, the initialization part consists of assigning the filter count to 263, as it's the filter coefficients, filter output to zero, filter buffer size to be 1000 and filter buffer to be the memory of the int16 times 1000. Then, in a for-loop, we assign the filter's buffer values to be zero. Next, we assign the converted filter coefficients to the filter coefficients.

Onto the update part, we first reset the filter's output value to be zero and declare the result to be 2^{14} as a rounding value. Next, we assign the input to be put into the buffer. Then, we proceed to the optimization part which is explained in **Part 4.** After the optimization part is completed, we move the buffer overflow forward. Lastly, we complete the saturation part and then we are outputting the output WAV file.

Part 3: Process Coefficients to Fixed-Point

In this case, we are converting float data type to int16 type. We take the current float array of the coefficients and the empty array of int16 coefficients. Then, we iterate through the for-loop, as the boundary for float point is -32768 <= f <= 32767, we check the int16 value of the coefficients when multiplied with 32768. If it's bigger than 32768, then the value becomes 32768. If it's less than -32768, then the value becomes -32768. Else, we convert the value to int16.

Part 4: FIR Filter RVP Optimization

The optimization happens in the FIRFilter.c file. RVP Optimization that we did is based on this resource from the TA, https://thewolfsound.com/fir-filter-with-simd/. We use the Inner Loop Vectorization and in this case, the loop will be incremented 5 times for every iteration. Here, we do the inner product of the two 5-element vectors and recreate this equation into code:

$$y[n] = (x[n] * h[n])[n]$$
$$= \sum_{k=0}^{N_h-1} x[n+k]c[k], \quad n = 0, \ldots, N_x - 1. \quad (3)$$

The implementation should be much more efficient because SIMD uses multiple registers at the same time which enables the processor to handle processes much more efficiently. In this assignment, RVP optimization is not done.

Part 5: Conclusion

The difficulties we encountered is the fact that it's quite difficult to find resources for this assignment and the programming language that we use is not the language that was taught by the instructor. Maybe, next time the TAs can provide a half-finished source code that we can complete, so doing this assignment would take that much time to research. Nevertheless, it is quite interesting to learn how you can manipulate audio using code.

Resources:

https://stackoverflow.com/questions/23030980/creating-a-stereo-wav-file-using-c
https://sestevenson.wordpress.com/implementation-of-fir-filtering-in-c-part-1/
https://sestevenson.wordpress.com/implementation-of-fir-filtering-in-c-part-2/
https://sestevenson.wordpress.com/implementation-of-fir-filtering-in-c-part-3/
https://topic.alibabacloud.com/a/c-language-parsing-wav-audio-files_1_31_30000716.html
https://thewolfsound.com/fir-filter-with-simd/