Lab 5 Applications Using the DSP56858EVM CODEC

-Hemanth Balaji Dandi

1. Lab Objectives

The objective of this lab is to develop real-time audio applications using the available onboard codec and get familiar with the on-board codec features and settings, and the C and assembly fixed point implementation of Echo and Finite Impulse Response (FIR) digital filters.

2. Results and Analysis

Q: E*xamine the generated code in "Codec_LoopBack.c", in particular the section shown in Figure 4. Why are "Codec_Read" and "Codec_Write" used within a while loop?*

A: Because DSP keeps receiving samples. As long as DSP is receiving, the codec must read the samples and then write the samples in the same loop.

Q: *For each of three cases, debug and check the output. Answer the following questions:*

1) In equation (1) we notice that for n < D, y(n-D) has a negative index. You cannot have a negative index into an array in C. How does the C code handle this?

2) Which variable in the C code should you modify to vary the delay D?

3) Why do we need to perform Modulo division here: cnt=cnt%BUFSIZE;

4) Compare the program memory size for the three versions and comment.

5) Describe the program flow.

A: 1) Since we use circular buffers, y(n-D)=y(n+T-D), which has a positive index. The C code use this way to handle this problem.

2) We should modify BUFSIZE to vary the delay.

3) Because we use circular buffers, so cnt represents the delay in a positive way after we perform the Modulo division. Since we won't be having an infinite BUFSIZE length, we can control the overall length through circular convolution, thereby maintaining the size while varying the corresponding index through the modulas operation.

4) The program memory size for fixed point, floating point and intrinsic is 4KB. All of them are the same.

5) First we do internal initialization and set up LEDs. Then we use a while loop. If we are receiving samples, read the input samples. By using modulo division, we can calculate the delay. Then perform the formula of the Echo to produce the output samples and use Codec_Write to generate the output.

Q: *Pick one of the proposed versions. Three parameters can be varied- Attenuation, Delay and Sampling frequency. You will be varying one out of the three parameters at a time while keeping the other two constant. Notice changes in the output and tabulate your results as shown. To change the sampling frequency of the CODEC use switch 5 on the board (description of the switch operation is described in the DSP56858EVM interactive power point presentation).*

A: We picked fixed point version.
Case 1: Constant Attenuation and Delay, varying sampling frequency
Attenuation=0.5 D=0.125

Sampling Frequency	Observation
8kHz	The echo occurred faster and less echo could be heard as sampling frequency becomes larger
9.6kHz	
12kHz	

Case 2: Constant sampling frequency and Delay, varying Attenuation
Sampling Frequency =8kHz D=0.125

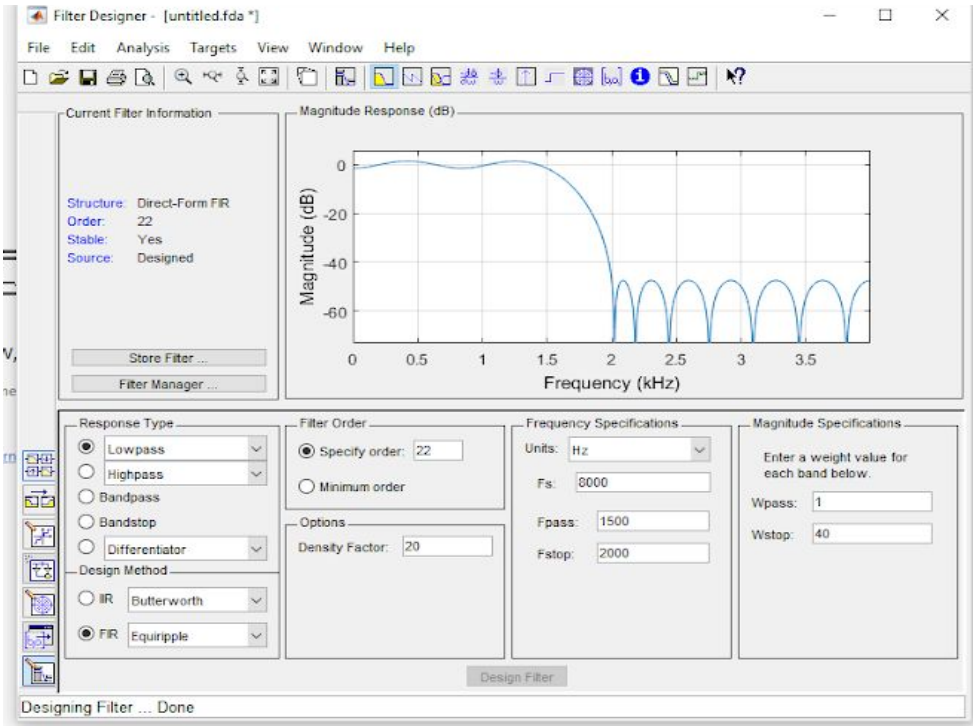
Attenuation	Observation
0.5	As attenuation became smaller, we could hear less echo and it became louder.
0.25	
0.125	

Case 3: Constant Attenuation and sampling frequency, varying Delay
Attenuation=0.5 Sampling Frequency=8kHz

D	Observation
0.125	The echo occurred slower and we could hear more echo as the delay became larger.
0.25	
0.5	

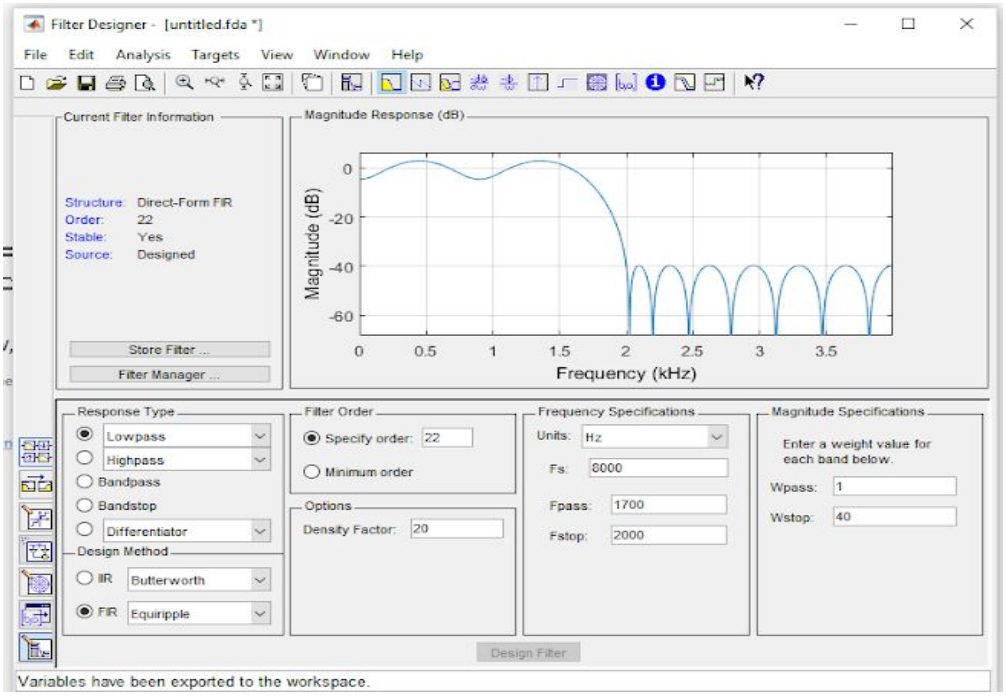
Q: *Implement the echo equation using a hybrid function called HEcho(). The HEcho() function should accept three parameters- The current sample value, the attenuation and*
the past output value. It should return the new calculated output. Use the fixed point version. Rewrite the line
buffer[cnt+i] = pSamples[i]+(int)(((long)attenuation(long)buffer[cnt+i])>>15);*
with a call to your implementation as follows:
buffer[cnt+i] = HEcho(pSamples[i],attenuation,buffer[cnt+i]);
A: This is included in “Code” folder.

Q: *Perform the following test:*
Generate a sinewave of 200mV peak-to-peak using the function generator and feed that
to the codec input.
Connect the codec output to the oscilloscope.
Start increasing the frequency from 200 Hz to around 4000 Hz. What do you observe? At what point does the sine wave amplitude start decreasing?
Change the passband frequency and get the coefficients using FDATOOL. Incorporate the new coefficients into the C code and run the test again. Comment.



A: The first decreasing occurs at 440Hz, then the second decreasing occurrs at 1240Hz. And it continues to decrease and goes to 0 at 2000Hz and remains 0 throughout.

Then we changed the passband to 1700Hz, the stopband is 2000Hz, and we observed the following results-



The first decreasing occurs at 600Hz, then the second decreasing occurrs at 1600Hz. And it continues to decrease and goes to 0 at 2000Hz and remains 0 throughout.

Frequency (in Hz)	Amplitude (in mv)
200	550
400	580
600	590
800	512
1000	540
1200	590
1700	280
1900	100
2000	0

Q: *Instead of a sine input, try to feed an audio stream from the PC and connect the output to the speakers. Compare the sound quality before and after filtering. Add two "ExtInt" beans for IRQA and IRQB. Perform filtering when IRQA is pressed and loopback (no filtering) when IRQB is pressed.*
A: This is included in "Code" folder.

Q: *Examine the C code in Filter_C_fixed.c and describe the flow of the program*
A: First use Codec_Read to read the input samples. Then use two for-loop to do MAC operation to get the output array. Use Codec_Write to generate the output.

Q: *Implement the entire C code (including loop) shown in Figure 11 using a hybrid function called HFilter(). It should accept three parameters- The pointer to the sample array, the pointer to the impulse response and the size of the filter. It should return the new calculated output. Replace the code in Figure 11 with a call to your implementation as follows:*
pSamples[samples]= HFilter(sample_array,h,SIZE);
A: This is included in "Code" folder.

3. Lab Evaluation

Difficulties: We had trouble modifying hybrid functions. Besides, we took a little bit long to figure out how to use external interrupt to control the filter.
Errors or unclear statements: Checking the program memory size should be done by checking sdm.elf in "bin" folder, not ldm.elf in "Output" folder.
Suggestions: There should be more detailed explanation about the codes of echo.

4. Conclusion

In this lab, we learned how to use codec to do loop back. Further more, we learned how to generate echo and design filter. And we got more familiar with hybrid functions and external interrupt.
What we like the most is using external interrupt to control the filter. What we like the least is the echo part.