National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering

Faculty Member: Dr. Wajahat Hussain

Dated: 28-3-2107

Course/Section: BEE6-B

Semester: 6th Semester

EE-330 Digital Signal Processing

Lab #6 Sampling, Quantization, Aliasing using DSK

Name	Reg. no.	Report Marks / 10	Lab Quiz-Viva Marks / 5	Total / 15
Saad Iqbal	111394			
Usman Iqbal	111393			
Abdullah Bin Asif	111596			

Lab Task 1

```
#define D 16000 // represents 2 sec at fs=8kHz
short d=0; // move these before main()interrupt
void isr(){
read_inputs(&xL, &xR);
yL=(d<D) *xL;
yR=(d>=D)*xR;
if (++d >= 2*D) d=0;
write_outputs(yL,yR);
return;
}
```

Rebuild your program with these changes and play a song. In your lab write-up explain why and how this code works.

Answer:

This code read left and right input samples from codec and tests a condition, according to which it write left and right output samples to codec. When the value of "d" is smaller than "D" we can hear the sound only in the left speaker. Then the sound was audible in the right speaker for 2 seconds. This code does not work once, but continues till the sound ends.

Lab Task 2

Aliasing

This part demonstrates aliasing effects. The smallest sampling rate that can be defined is 8 kHz with a Nyquist interval of [−4,4] kHz. Thus, if a sinusoidal signal is generated (e.g. with MATLAB) with frequency outside this interval, e.g., f = 5 kHz, and played into the line-input of the DSK, one might expect that it would be aliased with fa =f−fs =5−8 =−3kHz. However, this will not work because the antialiasing oversampling decimation filters of the codec filter out any such out-of-band components before they are sent to the processor.

An alternative is to decimate the signal by a factor of 2, i.e., dropping every other sample. If the codec sampling rate is set to 8 kHz and every other sample is dropped, the effective sampling rate will be 4 kHz, with a Nyquist interval of [−2,2]kHz. A sinusoid whose frequency is outside the decimated Nyquist interval [−2,2]kHz, but inside the true Nyquist interval [−4,4]kHz, will not be cut off by the antialiasing filter and will be aliased. For example, if f =3kHz, the decimated sinusoid will be aliased with fa =3−4 =−1kHz.

Copy the template programs to your working directory. Set the sampling rate to 8 kHz and select line-input. Modify the template program to output every other sample, with zero values in-between. This can be accomplished in different ways, but a simple one is to define a "sampling pulse" periodic signal whose values alternate between 1 and 0, i.e., the sequence [1,0,1,0,1,0,...] and multiply the input samples by that sequence. The following simple code segment implements this idea:

```
yL = pulse * xL;
yR = pulse * xR;
pulse = (pulse==0);
```

Where pulse must be globally initialized to 1 before main() and isr(). Why does this work? Next, rebuild the new program with CCS.

Answer:

This code works because it simply multiply your sound signal to a series of 1 & 0 pulse, thus losing half of the samples of original sound, hence causing down sampling.

Open MATLAB and generate three sinusoids of frequencies f1 = 1kHz, f2 = 3kHz, and f3 = 1kHz, each of duration of 1 second, and concatenate them to form a 3-second signal. Then play this out of the PCs sound card using the sound() function. For example, the following MATLAB code will do this:

```
fs = 8000; f1 = 1000; f2 = 3000; f3 = 1000;
L = 8000; n = (0:L-1);A = 1/5; % adjust playback volume
x1=A* cos(2*pi*n*f1/fs);
x2=A* cos(2*pi*n*f2/fs);
x3=A* cos(2*pi*n*f3/fs);
sound([x1,x2,x3], fs);
```

a) Connect the sound card's audio output to the line-input of the DSK and rebuild/run the CCS down-sampling program after commenting out the line:

pulse = (pulse==0);

This disables the down sampling operation. Send the above concatenated sinusoids to the DSK input and you should hear three distinct 1-sec segments, with the middle one having a higher frequency.

b) Next, uncomment the above line so that down sampling takes place and rebuild/run the program. Send the concatenated sinusoids to the DSK and you should hear all three segments as though they have the same frequency (because the middle 3 kHz one is aliased with other ones at 1 kHz). You may also play your favorite song to hear the aliasing distortions, e.g., out of tune vocals.

Answer:

Due to the aliasing of 3 kHz the same sound was heard over a period of 3 seconds.

c) Set the codec sampling rate to 44 kHz and repeat the previous two steps. What do you expect to hear in this case?

Answer:

In both the cases aliasing won't occur. In the first one due to the antialiasing filters and in the second one because the Nyquist interval of [22,-22] kHz

d) To confirm the antialiasing pre-filtering action of the codec, replace the first two lines of the above MATLAB code by the following two:

fs = 16000; f1 = 1000; f2 = 5000; f3 = 1000;L = 16000; n = (0:L-1);

Now, the middle sinusoid has frequency of 5 kHz and it should be cutoff by the antialiasing pre filter. Set the sampling rate to 8 kHz, turn off the down sampling operation, rebuild and run your program, and send this signal through the DSK, and describe what you hear.

Answer:

First a 1 KHz tone was heard, and then a pause and then again a 1KHz tone. 5 KHz frequency got filtered out. Because when the sampling rate was set to 8000 Hz aliasing occur as Nyquist criteria wasn't satisfied. But when the sampling rate was set to 16000 Hz and 44000 Hz aliasing didn't occur as Nyquist criteria was satisfied.

Lab Task 3:

Quantization:

The DSK's codec is a 16-bit ADC/DAC with each sample represented by a two's complement integer. Given the 16-bit representation of a sample, [b1b2 ⋯b16], the corresponding 16-bit integer is given by:

$$x = -b12^{-1} + b22^{-2} + b32^{-3} + \cdots + b162^{-16} \quad (1.1)$$

The MSB bit b1 is the sign bit. The range of representable integers is: $-32768 \leq x \leq 32767$. As discussed in Ch. 2 of Ref. [1], for high-fidelity audio at least 16 bits are required to match the dynamic range of human hearing; for speech, 8 bits are sufficient. If the audio or speech samples are quantized to less than 8 bits, quantization noise will become audible. The 16-bit samples can be re-quantized to fewer bits by a right/left bit-shifting operation. For example, right shifting by 3 bits will knock out the last 3 bits, then left shifting by 3 bits will result in a 16-bit number whose last three bits are zero, that is, a 13-bit integer. These operations are illustrated below:

$$[b1,b2, \ldots, b13,b14,b15,b16] \Rightarrow [0,0,0,b1,b2, \ldots, b13] \Rightarrow [b1,b2, \ldots, b13,0,0,0]$$

a) Modify the basic template program so that the output samples are re-quantized to B bits, where $1 \leq B \leq 16$. This requires right/left shifting by L= 16 – B bits, and can be implemented very simply in C as follows:

```
yL = (xL >> L) << L;

yR = (xR >> L) << L;
```

Start with B = 16, set the sampling rate to 8 kHz, and rebuild/run the program. Send a wave file as input and listen to the output.

Answer:

Firstly with B=16, we could hear the sound clearly because there was no shifting of bits hence no noise was added.

b) Repeat with the following values: B= 8,6,4,2,1, and listen to the gradual increase in the quantization noise.

Answer:

As B was decreased the number of bits that dropped off increases hence noise increased. The number of quantization level decreased and the error increased.

Conclusion:

From this lab we learnt how to process the audio signals using the DSP starter kit and we observed the effects of aliasing and quantization in the audio signal.
