ECE 4755: Digital Signal Processing Lab

Project 5: Echo Cancellation

Angel Rodriguez

Utah Valley University

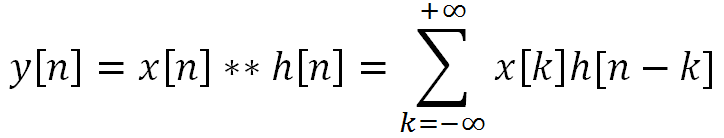
# Introduction

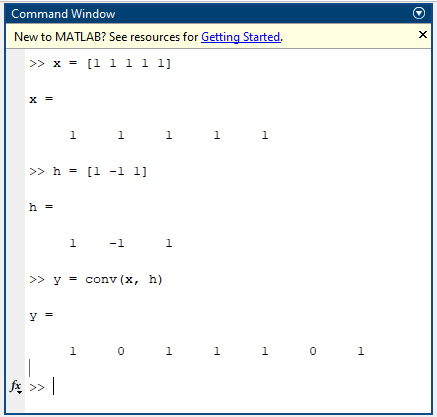
The objective of this lab is to learn how to use Code Composer Studio (CCS) to implement convolution in C. This project is focused on the basics of CCS and how to load the code from the computer into a TI DSP memory. Voice recordings of the input signal x[n] will be convolved with the impulse response y[n] to derive an output y[n] = x[n] \* h[n]. X[n] will include the recordings of our own voice within an area without an echo while h[n] shall be a recording of a clap within an area to create an echoing impulse response. The goal of the convolution in C is to import these two sounds files as arrays of whatever types necessary (i.e., uint16, int32), output y[n] as another array of the same type and convert that array back into the same sound file type x[n] and h[n] originated from. The expected resulting sound is the voice with an echo.

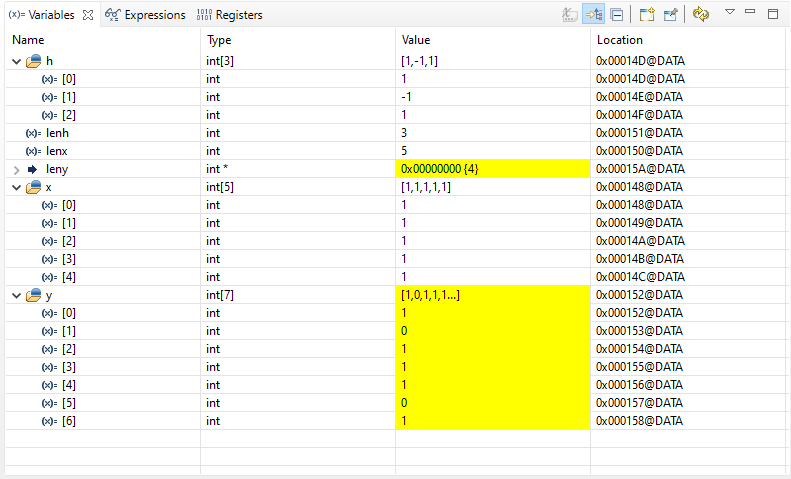
# Tasks

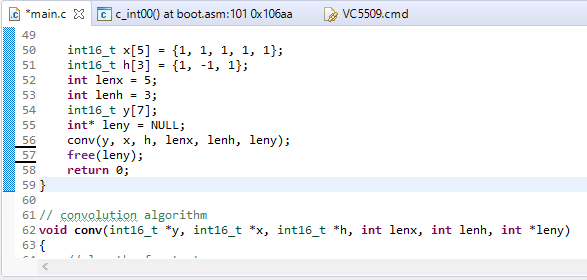
1. Work with the Signal
   1. e) How do you find the exact duration of this signal in seconds?  
      The exact duration is of the signal in seconds is the length of the signal divided by the sampling frequency. The sampling frequency is how many samples are taken per second (Hz).  
      Explain why it works?  
      >> plot(0:1/Fs:1/Fs\*(length(x)-1),x) works because you are plotting the signal x from 0 to 1/Fs\*length(x)-1 with 1/Fs being the increment. Where Fs is the sampling frequency (samples per second), 1/Fs is the sampling period (seconds per sample). The end of the signal is the sampling period multiplied by the length of x minus 1. Length x is equivalent to the sampling frequency multiplied by the number of seconds the signal lasts so length(x)\*(1/Fs) is approximately the duration of the signal in seconds.
2. Create the Echo-Corrupted Signal
   1. The delay parameter D to in seconds is N/Fs where Fs is the sampling frequency. (Can you explain why?)  
      N is in samples while Fs is samples per second. N/Fs yields a value whose units are in seconds. Therefore, D and N both represent a delay in which the difference is the unit used (samples versus seconds).
   2. What is the impulse response of this system?

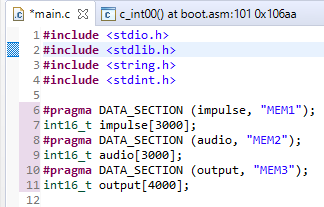
# Figures

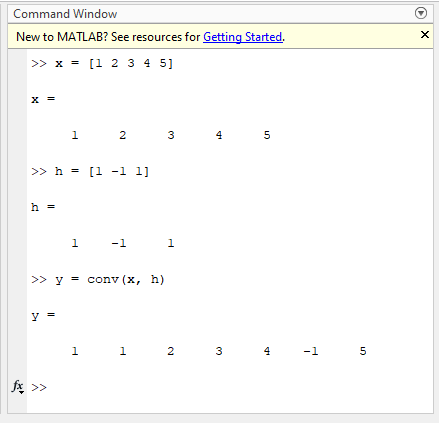
  
Figure 1:  
Convolution summation equation used for C program. The sigma is implemented by a for loop where negative infinity is 0 and positive infinity is the length of the longer array between x[n] and h[n].

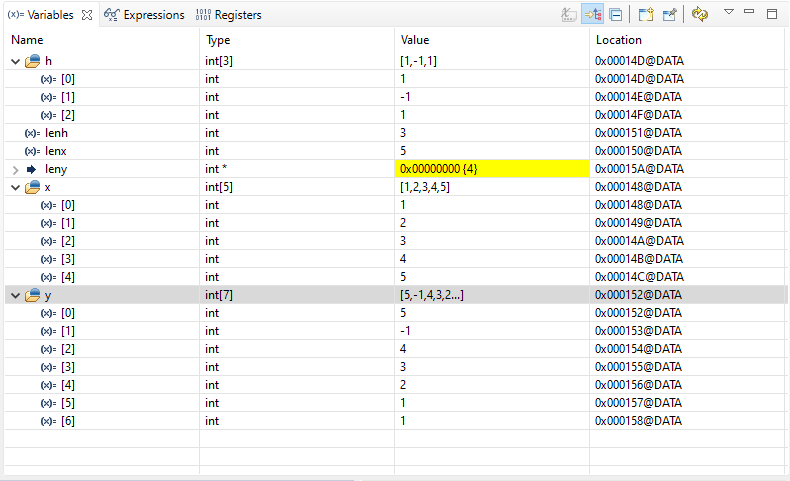
  
Figure 2:  
Command Window of MATLAB used to find the correct output of convolution using small testing arrays to also be used in CCS.

  
Figure 3:  
Expressions for arrays of x, h, and y depicting values which match the results of MATLAB showing the convolution function written in C works adequately.

  
Figure 4:  
C code of CCS implementing convolution for small testing arrays

  
Figure 5:  
Dedicating specific amounts of RAM (”MEM1“, ”MEM2“, ”MEM3”) to the large arrays which are used for impulse array h[n], audio array x[n], and output array y[n].

  
Figure 6:  
The second MATLAB test shows results of y[n] which is not equivalent to itself reversed.

  
Figure 7:  
Expressions for arrays of x, h, and y depicting values which when compared to the second MATLAB test, shows that y is reversed of what it should be and, therefore, the convolution function written in C does not work as previously thought.

# Conclusion

The lab ends with learning the manuals regards to memory and working with DSPLab unit to implement convolution. Concepts learned include connecting dedicating RAM to arrays, converting .WAV files to int16\_t arrays in C, passing arrays to a function as a pointer, using Python to shorten the length of audio values to put into C and then convert back to audio files, and better understanding of arrays/pointers/memory allocation in CCS. A concept learned is that variables assigned to arrays are pointers which point to the 0th index of the array. That way, when the convolution function created has a parameter which is an int16\_t pointer, the array variable name can be used as the parameter to the function call since that array variable name points to a memory address holding an int16\_t value (the first index of the array). When demanding large amounts of memory for a program array, the .cmd file in CCS must be manually edited to allocate RAM necessary to hold the large array(s). This issue was the greatest and most time-consuming obstacle of the lab and required much assistance and reading from the DSP books. Sound files cannot be imported simply to CCS as easily as in MATLAB, so the solution was to convert sound files (.WAV) to binary (using Python) to be used in CCS, and then when CCS gives an output, the output was converted back to .WAV form using a Python program for the sound to be played.

# Attachments

## CCS Code of main.c: Implements convolution

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <stdint.h>  
  
#pragma DATA\_SECTION (impulse, "MEM1");  
int16\_t impulse[3000];  
#pragma DATA\_SECTION (audio, "MEM2");  
int16\_t audio[3000];  
#pragma DATA\_SECTION (output, "MEM3");  
int16\_t output[4000];  
  
void conv(int16\_t \*y, int16\_t \*x, int16\_t \*h, int lenx, int lenh, int \*leny);  
  
/\*\*  
 \* main.c  
 \*/  
int main(void)  
{  
  
 // create pointers to audio files  
 FILE \* impulse\_file;  
 FILE \* audio\_file;  
  
 // temporary values for the file reading  
 int16\_t tmp\_read;  
 int tmp\_pointer = 0;  
  
 // code to read the impulse file and store it into the impulse array  
 impulse\_file = fopen("impulse.bin", "r");  
 if(impulse\_file == NULL) {  
 perror("Error opening impulse file.");  
 return(-1);  
 }  
 // pointer, size in bytes of each element, number of elements, file stream  
 fread(impulse, 2, 1826, impulse\_file);  
 fclose(impulse\_file);  
  
 // code to read the audio file and store it into the audio array  
 tmp\_pointer = 0;  
 audio\_file = fopen("audio.bin", "r");  
 if(audio\_file == NULL) {  
 perror("Error opening audio file.");  
 return(-1);  
 }  
 // pointer, size in bytes of each element, number of elements, file stream  
 fread(audio, 2, 1718, audio\_file);  
 fclose(audio\_file);  
  
 int\* leny = NULL;  
  
  
  
 free(leny);  
  
 return 0;  
}  
  
// convolution algorithm  
void conv(int16\_t \*y, int16\_t \*x, int16\_t \*h, int lenx, int lenh, int \*leny)  
{  
 // length of output  
 int conv\_length;  
 // iterator for x, h, and inverted x respectively  
 int i, j, i1;  
 // temporary result of each output  
 int16\_t tmp;  
  
 // allocated convolution array  
 conv\_length = lenx+lenh-1;  
  
 // convolution process  
 // outside loop loops through output  
 for (i=0; i < conv\_length; i++)  
 {  
 // this would point to the end of x  
 i1 = lenx - i - 1;  
 // temporary storage value  
 tmp = 0;  
 // inner loop loops through h  
 for (j=0; j<lenh; j++)  
 {  
 // this if loop checks to make sure the parts (x) are valid  
 if(i1>=0 && i1<lenx)  
 // this is the summation of each multiplication for each x[-n]\*h[m]  
 tmp = tmp + (x[i1]\*h[j]);  
  
 // increment so that we pass through each value of x where an h exists  
 i1 = i1+1;  
 // assign our working value to the output signal  
 y[i] = tmp;  
 }  
 }  
  
 //get length of convolution array  
 (\*leny) = conv\_length;  
}

Python Code of [modifydata.py:](https://github.com/austinphill6/ECE4755/blob/main/modifydata.py) Script to edit waveform  
from scipy.io import wavfile  
import numpy as np  
  
impulse = wavfile.read('/home/snekmaster/Documents/impulse\_response2.wav')  
  
impulse\_response = np.array([])  
size = int(impulse[1].shape[0]/4)  
  
with open('impulse\_response.c', 'w') as file:  
 file.write(f'uint8\_t impulse\_response [{size}] = ' + '{')  
 for i in impulse[1]:  
 if i % 4 == 0:  
 file.write(f'{i},\n')  
 impulse\_response = np.append(impulse\_response, i)  
 file.write('};')  
  
fs = int(impulse[0]/4)  
print(fs)  
print(impulse\_response.shape)  
wavfile.write("new\_impulse\_response.wav", fs, impulse\_response.astype(np.uint8))  
  
echo = wavfile.read('/home/snekmaster/Documents/echo.wav')  
  
echo2 = np.array([])  
size = int(echo[1].shape[0]/4)  
with open('echo.c', 'w') as file:  
 file.write(f'uint8\_t echo [{size}] = ' + '{')  
 for i in echo[1]:  
 if i % 4 == 0:  
 file.write(f'{i},\n')  
 echo2 = np.append(echo2, i)  
 file.write('};')  
  
fs = int(echo[0]/4)  
print(fs)  
print(echo2.shape)  
wavfile.write("new\_echo.wav", fs, echo2.astype(np.uint8))

Python Code of [wavtobinary.py](https://github.com/austinphill6/ECE4755/blob/main/wavtobinary.py)[:](https://github.com/austinphill6/ECE4755/blob/main/modifydata.py) Script to convert .WAV file to binary values  
import wave  
  
divider = 4  
  
counter = 0  
with wave.open('impulse.wav', 'r') as impulse:  
 with open('data\_files/impulse.bin', 'bw') as f:  
 while True:  
 frame = impulse.readframes(1)  
 if frame == b'':  
 break  
 elif counter % divider == 0:  
 f.write(frame)  
 counter = counter + 1  
  
counter = 0  
with wave.open('audio.wav', 'r') as audio:  
 with open('data\_files/audio.bin', 'bw') as f:  
 while True:  
 frame = audio.readframes(1)  
 if frame == b'':  
 break  
 elif counter % divider == 0:  
 f.write(frame)  
 counter = counter + 1

## Python Code of [binarytowav.py](https://github.com/austinphill6/ECE4755/blob/main/binarytowav.py)[:](https://github.com/austinphill6/ECE4755/blob/main/modifydata.py) Script to convert binary back to .WAV file

import wave  
  
divider = 4  
  
counter = 0  
with wave.open('output.wav', 'w') as output:  
 with open('data\_files/output.bin', 'br') as f:  
 output.setnchannels(1)  
 output.setsampwidth(2)  
 output.setframerate(2756)  
 output.setnframes(3543)  
 output.writeframes(f.read(3543\*2))  
 output.close()