EE 113 DA Digital Signal Processing Design

Lab 2: EXTERNAL INPUTS AND VOICE TRANSFORMATION

Professor Mike Briggs, Fall 2018

Brian Tehrani UID: 604715464

Attiano Purpura-Pontoniere UID:504847318

Milad Nourian UID: 004854226

**Objective:**

The purpose of this lab is to manipulate sound waveforms with the LCDK. In the first part of the lab we are tasked to create a delay between output pulses of the LCDK and function generator using a FIFO buffer. For the last part of the lab we are tasked with manipulating a sound wave into several different speeds.

**Part 1: Manipulating and analyzing sine waves**

For the first part of the lab we are to use the code form Lab 2 which reads in the input for the function generator and stores it into the LCDK. We then chart the time delay of one pulse (sine wave, 0.5 Vpp) between the function generator output and the LCDK output and compare the two. For the last part of the lab, we program a 3 ms time delay and observe the output.

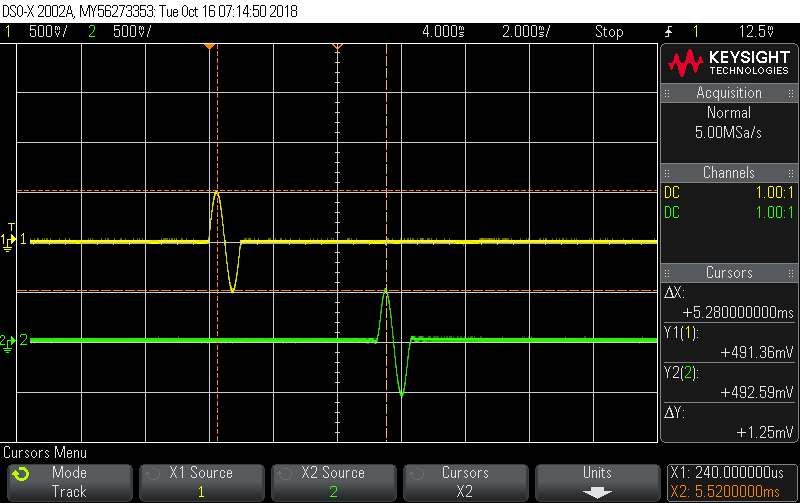
****

Figure 1: 8Ksps burst signal of sine wave

The graph in Figure 1 shows the output of both the function generator (yellow) and LCDK (green) one pulse sinusoidal signals. The difference between the two waves is that Gibbs phenomenon is shown on the LCDK waveform. Gibbs phenomenon is shows as “ripples” as the wave behaves sinc-like compared to the sinusoidal-like output from the function generator. This phenomenon occurs due to the sharp transition of the waveform and the sampling frequency being small such that only a few odd harmonics will be read. The sampling rate is 8Ksps and the oscilloscope reads the output form -4KHz to 4 KHz, thus only the odd harmonics of 1KHz and 3KHz will be read, since the output frequency is 1KHz.

Creating a 3ms delay:

Figure 1 shows a 5.3ms delay between outputs. The next part of Part 1 requires a delay in the output of 8.3ms, a 3ms delay. In order to create the delay, a FIFO buffer array is created (code shown below).

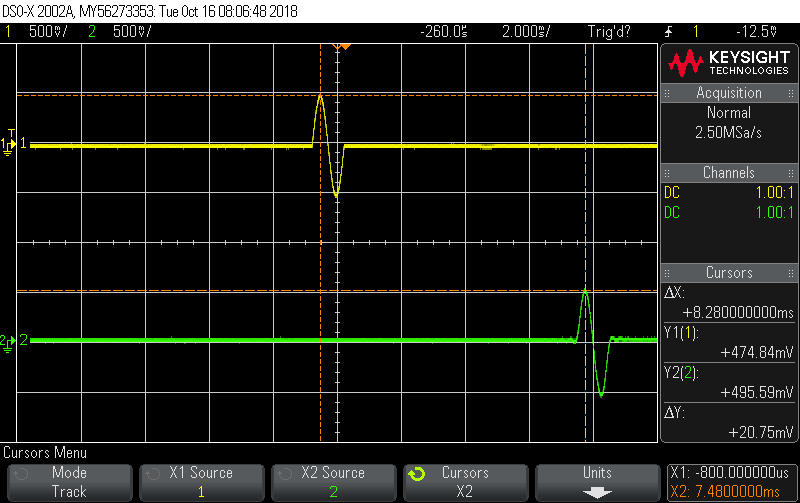


Figure 2: 3ms delay of Fig. 1

The buffer code works as follows, an array of side 24 is created to store the output values of the function generator. The size of the array is determined via dimensional analysis of the sampling frequency multiplied by the desired delay shown in the below equations.

\*

The buffer array is first filled with 24 zeros, then as the first value of the array is outputted (value 0) the last value of the array will fill with the output of the function generator red into the LCDK. The result is a 3ms delay shown in Fig 2.

Relevant Code:

*#include "L138\_LCDK\_aic3106\_init.h"*

*#include "evmomapl138\_gpio.h"*

*#include <stdint.h>*

*#include <math.h>*

*#include <ti/dsplib/dsplib.h>*

*// Global Definitions and Variables*

*int fifo[24];*

*int i=0;*

*interrupt void interrupt4(void) // interrupt service routine*

*{*

*if(i==24){*

*i=0;*

*}*

*int16\_t left\_sample;*

*// Input from ADC (Line IN)*

*left\_sample = input\_left\_sample();*

*output\_left\_sample(fifo[i]);*

*fifo[i]=left\_sample;*

*i++;*

*// Output to DAC (Line OUT)*

*return;*

*}*

*int main(void)*

*{*

*int j=0;*

*for(j=0;j<24;j++){*

*fifo[j]=0;*

*}*

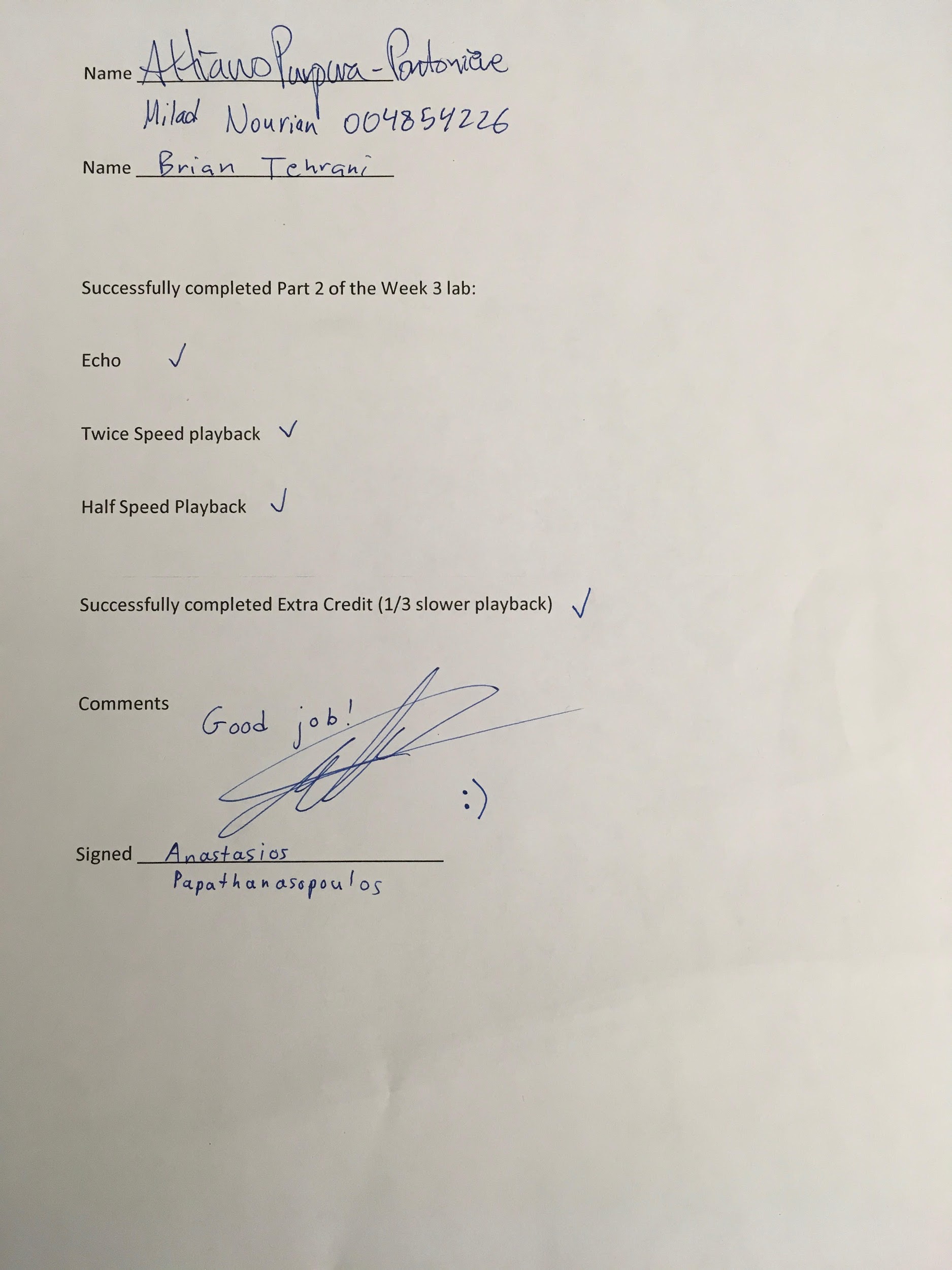
*L138\_initialise\_intr(FS\_8000\_HZ,ADC\_GAIN\_0DB,DAC\_ATTEN\_0DB,LCDK\_LINE\_INPUT);*

*// SAMPLE CODE: USE OF FFT ROUTINES*

*while (1);*

*}*

**Part 2:**



For part 2 of the lab we are tasked with recording one second of sound using a microphone connected to the LCDK and output the sound into several distinct speeds. The strategy to determine how to yield the correct speeds are shown below.

Echo:

To generate the echo we need to create a delay of 70ms. To figure out the number of samples that equates to this time delay we use dimensional analysis:

\*

The echo is then coded without using a buffer as in part 1 but via a different method. The first 560 samples were output straight to the speaker. Then after that the original signal is then added to the current and both this added signal and original will be played at the same time.

Twice Speed Playback:

To create twice speed playback we just skipped every other point in the input\_buffer, effectively cutting down the number of samples in half.

Half Speed Playback:

To create half-speed playback we interpolated between every two points an extra point that is the average value of the current and following point. we accomplished this by using a new counter that goes from 0-15999 and on every even value of the counter, we just output the current value of the input buffer, and on every odd value of the counter, we just output the average of the current value and next value of the input buffer.

⅓ Slower Playback (Extra Credit):

To create ⅓ slower playback we interpolated between every two points an extra two points, one point is one-third of the difference between the original two points, and the other point is two-thirds of the difference between the original two points. To accomplish this we found the difference of the values of the two points, and then added ⅓ and ⅔ of the difference to the first original point for both new points respectively.

Certain issues that could arise is when the values of the input soundwave exceed the upper frequency limits of the LCDK. To accommodate for this we can normalize the values and multiply the values by an adequate gain as to capture the entire input waveform.

Code:

#include "L138\_LCDK\_aic3106\_init.h"

#include "evmomapl138\_gpio.h"

#include "L138\_LCDK\_switch\_led.h"

#include <stdint.h>

#include <math.h>

#include <stdio.h>

// Global Definitions and Variables

int fifo[24];

int i=0;

int total=0;

int all\_light\_counter=0;

int testicle\_count=0;

int16\_t input\_buffer[8000];

int buffer\_index=0;

int temp\_max=0;

int state\_6\_counter=0;

int state\_7\_count=0;

int state\_8\_count=0;

int16\_t output;

int counttt=0;

void find\_max\_in\_8000(int\* p){

int j=0;

for(j=0;j<8000;j++){

if(abs(p[j])>temp\_max){

temp\_max=abs(p[j]);

}

}

}

interrupt void interrupt4(void) // interrupt service routine

{

total = LCDK\_SWITCH\_state(5)+LCDK\_SWITCH\_state(6)+LCDK\_SWITCH\_state(7)+LCDK\_SWITCH\_state(8);

//if(i==24){

// i=0;

//}

int16\_t left\_sample;

left\_sample = input\_left\_sample();

if(total==0){

testicle\_count=0;

buffer\_index=0;

LCDK\_LED\_off(4);

LCDK\_LED\_off(6);

LCDK\_LED\_off(5);

LCDK\_LED\_off(7);

output\_left\_sample(0);

}

else if(total>1){

testicle\_count=0;

buffer\_index=0;

all\_light\_counter++;

//error more than two switches turned

//flash every half second

if(all\_light\_counter<4000 || all\_light\_counter == 4000){

LCDK\_LED\_on(4);

LCDK\_LED\_on(5);

LCDK\_LED\_on(6);

LCDK\_LED\_on(7);

}

else if(all\_light\_counter>4000){

if(all\_light\_counter>8000){

all\_light\_counter=0;

}

LCDK\_LED\_off(4);

LCDK\_LED\_off(5);

LCDK\_LED\_off(6);

LCDK\_LED\_off(7);

}

output\_left\_sample(0);

}

else if (total==1){

if(LCDK\_SWITCH\_state(5)){

testicle\_count++;

if(testicle\_count<8000){

LCDK\_LED\_off(5);

LCDK\_LED\_off(6);

LCDK\_LED\_off(7);

LCDK\_LED\_on(4);

}

else if(testicle\_count > 8000 && testicle\_count<16000){

LCDK\_LED\_off(4);

LCDK\_LED\_off(6);

LCDK\_LED\_off(7);

LCDK\_LED\_on(5);

}

else if(testicle\_count > 16000 && testicle\_count<24000){

LCDK\_LED\_off(5);

LCDK\_LED\_off(4);

LCDK\_LED\_off(7);

LCDK\_LED\_on(6);

}

else if(testicle\_count > 24000 && testicle\_count<32001){

LCDK\_LED\_off(4);

LCDK\_LED\_off(6);

LCDK\_LED\_off(5);

LCDK\_LED\_on(7);

input\_buffer[buffer\_index]=left\_sample;

buffer\_index++;

}

else if(testicle\_count > 32000){

LCDK\_LED\_off(7);

LCDK\_LED\_off(6);

LCDK\_LED\_off(5);

LCDK\_LED\_on(4);

}

output\_left\_sample(0);

}

else if(LCDK\_SWITCH\_state(6)){

LCDK\_LED\_off(4);

LCDK\_LED\_off(7);

LCDK\_LED\_off(6);

LCDK\_LED\_on(5);

testicle\_count=0;

buffer\_index=0;

state\_6\_counter++;

//560 samples between first and second output

if(state\_6\_counter<560){

output=input\_buffer[state\_6\_counter];

}

else{

//find\_max\_in\_8000(input\_buffer);

output=(input\_buffer[state\_6\_counter]+(2.0/3.0)\*input\_buffer[state\_6\_counter-560]);

}

if(state\_6\_counter==7999){

state\_6\_counter=0;

}

output\_left\_sample(output);

}

else if(LCDK\_SWITCH\_state(7)){

LCDK\_LED\_off(4);

LCDK\_LED\_off(7);

LCDK\_LED\_off(5);

LCDK\_LED\_on(6);

testicle\_count=0;

buffer\_index=0;

state\_7\_count++;

if(state\_7\_count==3999){

state\_7\_count=0;

}

output\_left\_sample(input\_buffer[state\_7\_count\*2]);

}

else if(LCDK\_SWITCH\_state(8)){

LCDK\_LED\_off(4);

LCDK\_LED\_off(6);

LCDK\_LED\_off(5);

LCDK\_LED\_on(7);

testicle\_count=0;

buffer\_index=0;

state\_8\_count++;

state\_8\_count=state\_8\_count%24000;

if(state\_8\_count%3==0){

counttt++;

}

counttt=counttt%8000;

int difference = abs(input\_buffer[counttt]-input\_buffer[counttt+1]);

if(state\_8\_count%3==0){

output=input\_buffer[counttt];

}

else if(state\_8\_count%3==1){

output=input\_buffer[counttt]+(1.0/3.0)\*(float)(input\_buffer[counttt+1]-input\_buffer[counttt]);

}

else if (state\_8\_count%3==2){

output=input\_buffer[counttt]+(2.0/3.0)\*(float)(input\_buffer[counttt+1]-input\_buffer[counttt]);

}

output\_left\_sample(output);

}

}

// Input from ADC (Line IN)

//left\_sample = input\_left\_sample();

//output\_left\_sample(fifo[i]);

//fifo[i]=left\_sample;

//i++;

// Output to DAC (Line OUT)

return;

}

int main(void)

{

int j=0;

for(j=0;j<24;j++){

fifo[j]=0;

}

L138\_initialise\_intr(FS\_8000\_HZ,ADC\_GAIN\_24DB,DAC\_ATTEN\_0DB,LCDK\_MIC\_INPUT);

LCDK\_GPIO\_init();

LCDK\_SWITCH\_init();

LCDK\_LED\_init();

// SAMPLE CODE: USE OF FFT ROUTINES

while(1);

}