EE 113 DA\_ Digital Signal Processing Design

Lab 1: Waveform Generator and Measurement

Professor Mike Briggs, Fall 2018

Brian Tehrani UID: 604715464

Milad Nourian UID: 004854226

Attiano Purpura-Pontoniere UID:504847318

**Objective:**

The objective for Laboratory 1 is to become familiar with Computer Composer Studio software, the LCDK programmable board, and oscilloscope features. The lab requires us to generate specific sine waves, being coded in the programming language C, and output them on the oscilloscope. An additional purpose of this lab is to review various digital signal processing concepts such as nyquist’s theorem and aliasing.

**Step 1:**

The first part of the lab is to generate a 250 Hz sine wave with a sampling rate of 8 kHz. We are to familiarize ourselves with the Lab1.c code and become comfortable changing the code to output sine waves at different frequencies to the oscilloscope.

Report Requirements:

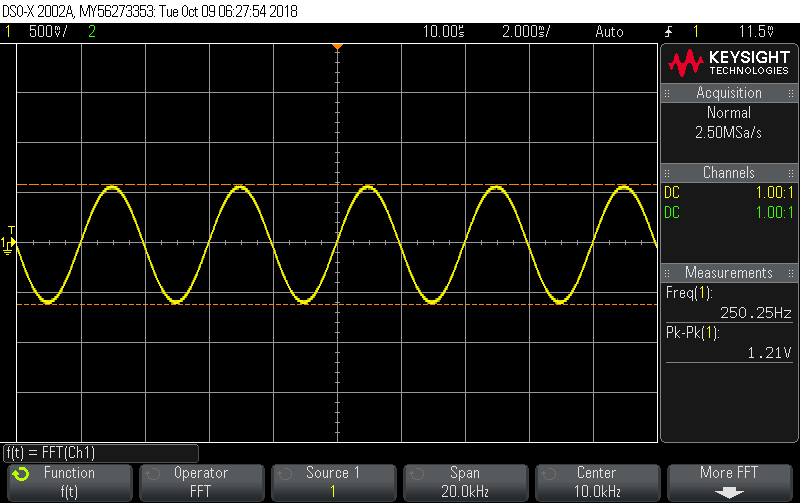


Figure 1a: Original 250 Hz sinusoid - the Frequency needs to be shown

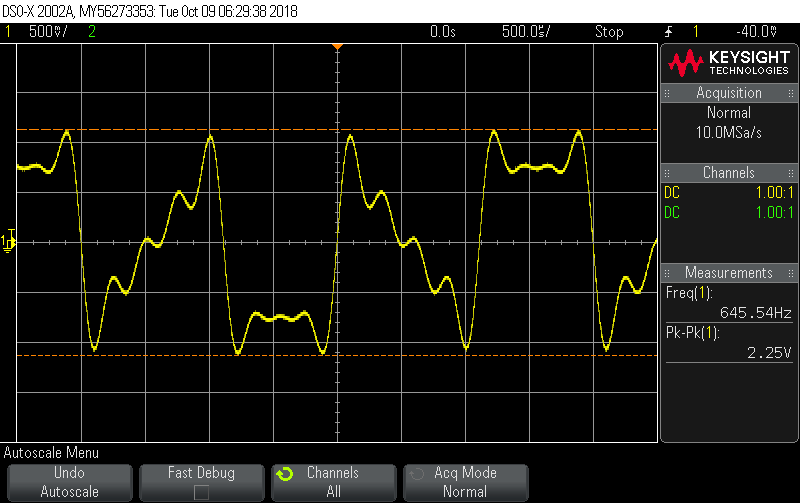


Figure 1b: Gain change form 20 to 40

The sinusoid in Figure 1a shows the output when running the Lab1.c code which has a gain of 20. In FIgure 1b the output is of the same sinusoid, but with the gain changed from 20 to 40.

**Analysis:**

1. T

**Code Snippet:**

int16\_t sine\_table[LOOPLENGTH]={0, 195, 383, 556, 707, 831, 924, 981,

1000, 981, 924, 831, 707, 556, 383, 195, 0,

-195, -383, -556, -707, -831, -924, -981,

-1000, -981, -924, -831, -707, -556, -383, -195};

**int16\_t gain = 20; // Change this to 40 for Figure 1b**

The doubled output is no longer a sinusoid because the sine table used in Lab1.c is not between the values -1 and 1, but -32765 to 32765. When the 40 gain is multiplied by the sine wave with the value of 1000 within the sine table the value becomes 40,000 which is beyond the sine table limit and thus have values that “roll over” in the table. In other words, this multiplication by 40 causes an overflow that causes the signal to behave in an unusual manner. The accounting for this “roll over” distorts the sinusoid.

**Step 2:**

The second part of the lab requires the change in the frequency of the sinusoid from 250 Hz to 1 kHz and 1.4 kHz. The 1 kHz wave can be achieved by either changing the index of the lookup table. The 1.4 kHz can be achieved by manipulating the sample rate, number of cycles, and the ratio of element arrays to the number of cycles.

Report Requirements:

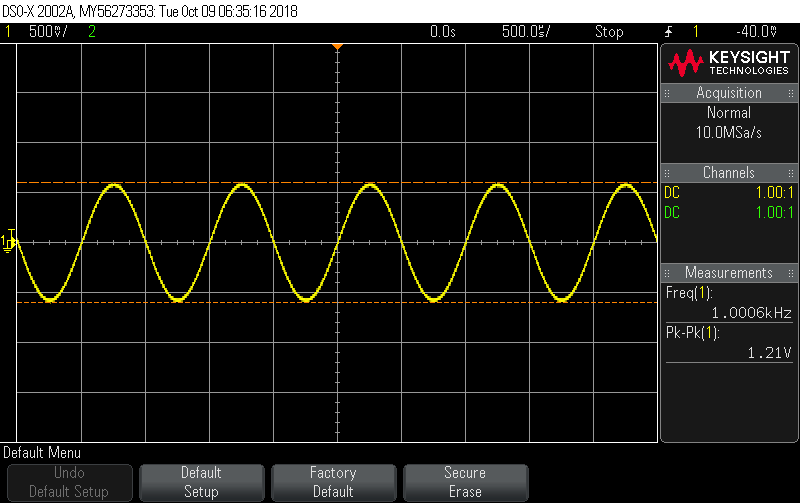


Figure 2a: Sine wave of 1 kHz

**the code snippet for the second part is the following:**

#include "L138\_LCDK\_aic3106\_init.h"   
#include "evmomapl138\_gpio.h"  
  
#define LOOPLENGTH 32  
  
int16\_t sine\_ptr = 0;  
int16\_t sine\_table[LOOPLENGTH]={0, 195, 383, 556, 707, 831, 924, 981,  
 1000, 981, 924, 831, 707, 556, 383, 195, 0,   
 -195, -383, -556, -707, -831, -924, -981,  
 -1000, -981, -924, -831, -707, -556, -383, -195};  
  
int16\_t gain = 20;  
  
interrupt void interrupt4(void) // interrupt service routine  
{  
 int16\_t left\_sample;  
  
 left\_sample = (sine\_table[sine\_ptr]\*gain);   
 sine\_ptr = sine\_ptr + 4;  
 sine\_ptr = sine\_ptr % LOOPLENGTH;  
  
 output\_left\_sample(left\_sample);  
  
 return;   
}  
int main(void)  
{   
L138\_initialise\_intr(FS\_8000\_HZ,ADC\_GAIN\_0DB,DAC\_ATTEN\_0DB,LCDK\_LINE\_INPUT);  
while (1);   
}

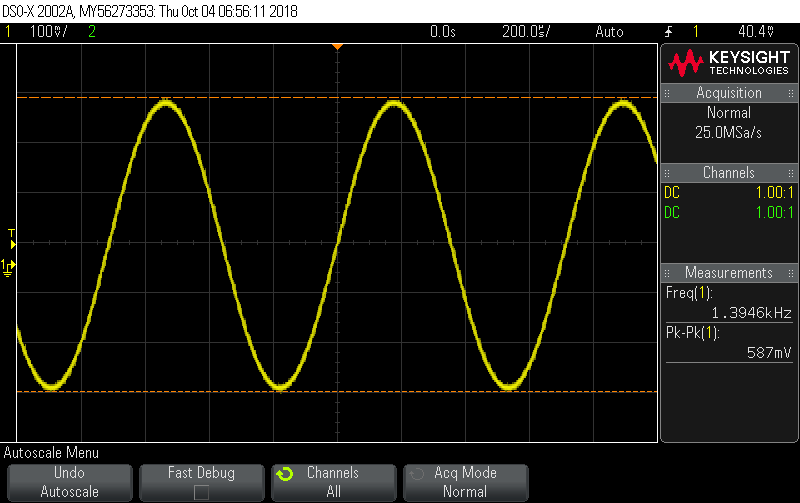


Figure 2b: Sine wave of 1.4 kHz

**Code Snippet:**

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

#include "evmomapl138\_gpio.h"

#include "math.h"

#include "stdio.h"

#define LOOPLENGTH 40

#define M\_PI 3.14159265358979323846

int16\_t sine\_ptr = 0;

int16\_t sine\_table[LOOPLENGTH]={891,809,-156,-951,-707, 309,988,588,-454,

-1000,-454,588,988,309,-707,-951,-156,809,891,

0,-891,-809,156,951,707,-309,-988,-588,454,1000,

454,-588,-988,-309,707,951,156,-809,-891,0};

int16\_t gain = 20;

interrupt void interrupt4(void) // interrupt service routine

{

int16\_t left\_sample;

//left\_sample = (int16\_t) sin((2\*M\_PI\*7\*sine\_ptr)/40)\*gain; //new line

left\_sample = (sine\_table[sine\_ptr]\*gain);

sine\_ptr = sine\_ptr +1;

sine\_ptr = sine\_ptr % LOOPLENGTH;

output\_left\_sample(left\_sample);

return;

}

int main(void)

{

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

while (1);

To create the 1 kHz sine wave the index of the array, known as sine\_ptr, must be changed using dimensional analysis. With a sampling frequency of 8 kHz and to find the desired 1 kHz frequency sine wave we need to determine the number of samples per cycle. This is achieved by dividing the two values to get 1/8 samples per cycle.

As asked in the lab, we now derive an expression for fgen (fsample, #samples/#cycles):

The index must skip every 8th value and thus instead of sampling all 32 values of the table we sample every 4.

To create the 1.4 kHz is more involved in that when dividing the sample rate with the frequency of our desired output sine wave we achieve 40 cycles per 7 samples. The sine table size must be increased to accommodate this change from 32 to 40. The sine equation must change accordingly:

The values of each point were calculated on a seperate excel spreadsheet and placed in the sine table.

**Step 3:**

The third part of the lab is to generate a 5 kHz sinusoid using a difference equation.

Report Requirements:

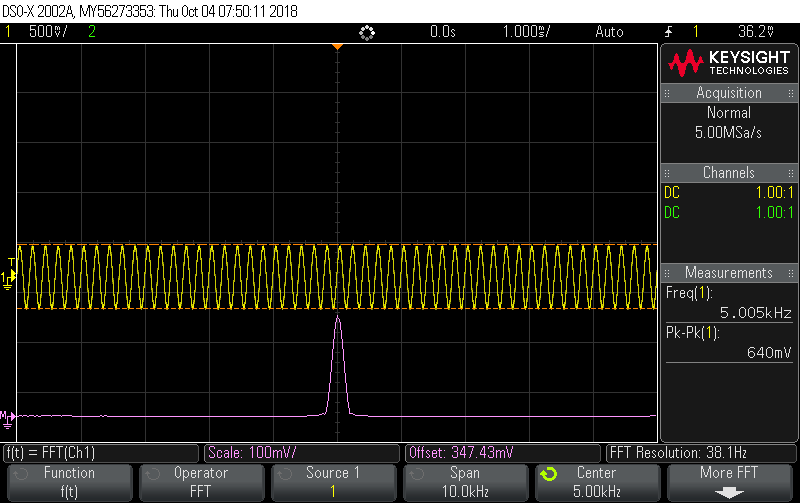


Figure 3: 5 kHz sine wave and FFT

**Code Snippet:**

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

#include "evmomapl138\_gpio.h"

#include "math.h"

#include "stdio.h"

#define LOOPLENGTH 40

#define PI 3.14159265358979323846

int16\_t sine\_ptr = 0;

int16\_t gain = 20;

float y\_negativetwo = -0.7071;

float y\_negativeone = 0;

float y = 0;

float frequency = 5000;

float theta;

interrupt void interrupt4(void) // interrupt service routine

{

// int16\_t left\_sample;

theta = (2\*PI\*(frequency/48000));

y = 2\*cos(theta)\*y\_negativeone - y\_negativetwo;

y\_negativetwo = y\_negativeone;

y\_negativeone = y;

//left\_sample = (int16\_t) sin((2\*M\_PI\*7\*sine\_ptr)/40)\*gain; //new line

// left\_sample = (sine\_table[sine\_ptr]\*gain);

// sine\_ptr = sine\_ptr +1;

// sine\_ptr = sine\_ptr % LOOPLENGTH;

output\_left\_sample(20000.0\*y);

return;

}

int main(void)

{

L138\_initialise\_intr(FS\_48000\_HZ,ADC\_GAIN\_0DB,DAC\_ATTEN\_0DB,LCDK\_LINE\_INPUT);

while (1);

}

The effect of changing the Time Base of the FFT affects the horizontal scale, turning it up gives more cycles in time, whereas turning it down gives less cycles in time. More cycless mean that the FFT is narrower and easier to notice and with less cycles the FFT is wider. This is because FFT is able to become closer to the actual FFT of the 5kHz sinusoid because raising the time base essentially increases the FFT’s resolution, and a low resolution FFT of a sinusoid has more side lobes appear because its approximated. Also worth noting is that decreasing the time resolution increase the frequency resolution. In other words, the samples rate varies with time base settings. By lowering the frequency rate, the FFT uses less points of the sequence to construct the frequency response which in turn, causes the frequency response to not have sharp edges at critical frequencies. Increasing sampling rate by time base on the oscilloscope produces a frequency response with sharper peaks.

**Step 4:**

In the fourth part of the lab we are to construct a waveform consisting of three sinusoids: 2.5 kHz, 3.5 kHz, and 5.8 kHz. The we will add a fourth sinusoid, 19.5 kHz, as noise to determiner the differences between the two waves. The FFT will be taken, then the sampling rate will change to 24 ksps and both the time and FFT will be observed in the resulting signal.

Report Requirements:

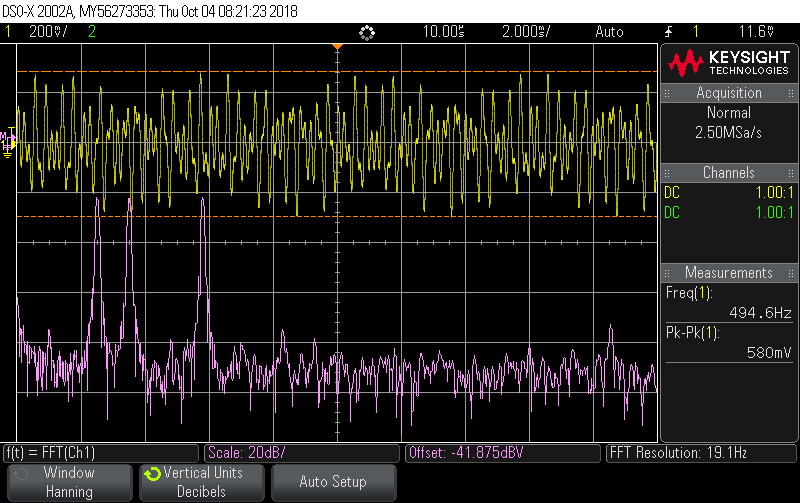


Figure 4a: Original three-frequency signal and its FFT



Figure 4b: Waveform and FFT result of generating the 4 sine waves at 48 Ksps

Figure 4c: waveform and FFT result of generating the 4 sine waves at 24 Ksps

Code Snippets:

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

#include "evmomapl138\_gpio.h"

#include "math.h"

#include "stdio.h"

#define LOOPLENGTH 40

#define PI 3.14159265358979323846

int16\_t sine\_ptr = 0;

int16\_t gain = 20;

float frequency1 = 2500;

float frequency2 = 3500;

float frequency3 = 5800;

float frequency4 = 19500;

float theta1;

float theta2;

float theta3;

float theta4;

float y\_negativetwo = .3214;

float y\_negativeone = 0;

float x\_negativetwo = 0.442288;

float x\_negativeone = 0;

float z\_negativetwo = .68835;

float z\_negativeone = 0;

float xyz\_negativetwo = .60876;

float xyz\_negativeone = 0;

interrupt void interrupt4(void) // interrupt service routine

{

// int16\_t left\_sample;

theta1 = (2\*PI\*(frequency1/24000));

theta2 = (2\*PI\*(frequency2/24000));

theta3 = (2\*PI\*(frequency3/24000));

theta4 = (2\*PI\*(frequency4/24000));

float y = 2\*cos(theta1)\*y\_negativeone - y\_negativetwo;

y\_negativetwo = y\_negativeone;

y\_negativeone = y;

//left\_sample = (int16\_t) sin((2\*M\_PI\*7\*sine\_ptr)/40)\*gain; //new line

float x = 2\*cos(theta2)\*x\_negativeone - x\_negativetwo;

x\_negativetwo = x\_negativeone;

x\_negativeone = x;

float z = 2\*cos(theta3)\*z\_negativeone - z\_negativetwo;

z\_negativetwo = z\_negativeone;

z\_negativeone = z;

float xyz = 2\*cos(theta4)\*xyz\_negativeone - xyz\_negativetwo;

xyz\_negativetwo = xyz\_negativeone;

xyz\_negativeone = xyz;

// left\_sample = (sine\_table[sine\_ptr]\*gain);

// sine\_ptr = sine\_ptr +1;

// sine\_ptr = sine\_ptr % LOOPLENGTH;

output\_left\_sample(20000.0/4\*(y+z+x+xyz));

return;

}

int main(void)

{

L138\_initialise\_intr(FS\_24000\_HZ,ADC\_GAIN\_0DB,DAC\_ATTEN\_0DB,LCDK\_LINE\_INPUT);

while (1);

}

The first screenshot has only three different frequency sinusoids ( 2.5 KHz, 3.5 KHz, and 5.8 KHz), this isnt so clear in the time display as it is just a periodic somewhat sinusoidal shaped function, however in the frequency display it is clear there are three distinct frequencies in the sinusoid. The second screenshot is taken with a sampling rate of 48ksps so there is no aliasing from the 19khz sinusoid added to the other three sinusoids, and shows a similarly periodic sinusoidal looking signal in the time domain, and four distinct peaks at the corresponding frequencies in the frequency domain ( 2.5 KHz, 3.5 KHz, 5.8 KHz and 19.5KHz). In the last screenshot, the sampling rate is 24ksps so the 19KHz sinusoid aliases down to F\_a = closest integer multiple of sampling rate - F => F\_a = 24 - 19.5Khz = 4.5Khz, the time display shows a strange looking sinusoidally shaped periodic function, and the frequency domain graph has peaks at 2.5 KHz, 3.5 KHz, 5.8 KHz and 4.5KHz.

The reason the first waveform and FFT is different from the other two is because another sinusoid is added to the other two, 19.5KHz sinusoid specifically. The reason the second waveform and its FFT is different from the third waveform and its FFT is because the sampling rate is halved from 48ksps to 24ksps, which results in the 19.5KHz sinusoid aliasing down to 4.5Khz because by Nyquist’s theorem the largest frequency able to be represented by a given sampling frequency is half the sampling frequency, or 12KHz in this scenario.

The conclusion then regarding the choice of sample rate vs highest frequency in the input signal is that the sampling rate must be greater than or equal to 2\*F\_highest or in other words F\_highest = F\_s/2.