Lab 1: Signal Generation and I/O Using Dedicated Hardware

Objective

The purpose of this lab is to become familiar with the I/O capabilities of the hardware system by generating a sinusoidal output of arbitrary frequency using the 'C67.

Laboratory Experiment Background

A simple digital IIR system for creating a sinusoid can be designed using two poles and appropriate filter weights. With a small modification, the filter can produce two sinusoids that are 90° out of phase. Modulation using quadrature phase signals is very common in communication systems.

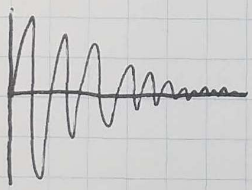
Procedure

1. Write a C program that implements a digital oscillator. (be sure to include the C code you wrote in your write-up.) The equations for this system are given by (*Proakis* text, p. 349):

$$\begin{bmatrix} y_c(n) \\ y_s(n) \end{bmatrix} = \begin{bmatrix} \cos\omega_0 & -\sin\omega_0 \\ \sin\omega_0 & \cos\omega_0 \end{bmatrix} \begin{bmatrix} y_c(n-1) \\ y_s(n-1) \end{bmatrix}$$

(c) Use a sampling rate of 24 kHz and an oscillator frequency of 3 kHz.

I initially thought the $\cos W_0$ was really $\cos W_0 n$ to cause the sinusoid to appear. When I did this, the wave form looked more like an impulse response and ~~to~~ the signal ~~went~~ went to zero after about ~~3 seconds~~ + less than 1 second.



running at 3kHzs caused no issues, but at 2kHz, the wave form amplitude would slowly decrease. This was due to me computing and storing values y_c and y_s as int16_t instead of float. y_c and y_s was cast to int16_t when writing out instead.

1/23/19

[signature]

Sinewave.c

```
int main(void)
{
	float ycData = AMPLITUDE;	// Sine Sample
	float oldYcData;		// Sine Sample delayed
	float ysData = AMPLITUDE;	// Sine Sample
	float oldYsData;		// Sine Sample delayed
	float sinVal = sin(OMEGA_0);	  // sin value at predetermined w0
	float cosVal = cos(OMEGA_0);	  // cos value at predetermined w0

	Init();

	// Infinite loop:  Each loop reads/writes one sample to the left and right
channels.
	while (true){

		//store y[n-1]
		oldYcData = ycData;
		oldYsData = ysData;

		// compute output as shown in Proakis text, p. 349
		ycData =  (oldYcData*cosVal - oldYsData*sinVal);
		ysData =  (oldYcData*sinVal + oldYsData*cosVal);


			//output samples
		// wait for xmit ready and send a sample to the left channel.
		while (!CHKBIT(MCASP->SRCTL11, XRDY)) {}
		MCASP->XBUF11 = (int16_t) ycData;

		// wait for xmit ready and send a sample to the right channel.
		while (!CHKBIT(MCASP->SRCTL11, XRDY)) {}
		MCASP->XBUF11 = (int16_t) ysData;

	}
}
```

Sinewave.h

```
// Set the sampling frequency at which the DAC will be operating.
#define SAMP_FREQ   (24000)		// Default

// The frequency of the sine wave that will be output to the line out port.
//#define SIN_FREQ    (2000)	// in Hz
#define SIN_FREQ    (3000)		// in Hz
//#define SIN_FREQ    (6000)	// in Hz
//#define SIN_FREQ    (9000)	// in Hz

#define PI (3.14159265358979)
// Amplitude of the sinewave
#define AMPLITUDE   (20000)
// Frequency of the sinewave
#define OMEGA_0 (2.0*PI*SIN_FREQ/SAMP_FREQ)
```

1/23/19
Am Pz

2. Using *Code Composer Studio*, compile and run the C program.

3. Using the two-channel oscilloscope, look at the time waveform of the oscillator outputs. Capture the waveform from the Analog Discovery software and paste in your notebook. Also, measure the frequency of the generated sinusoid.

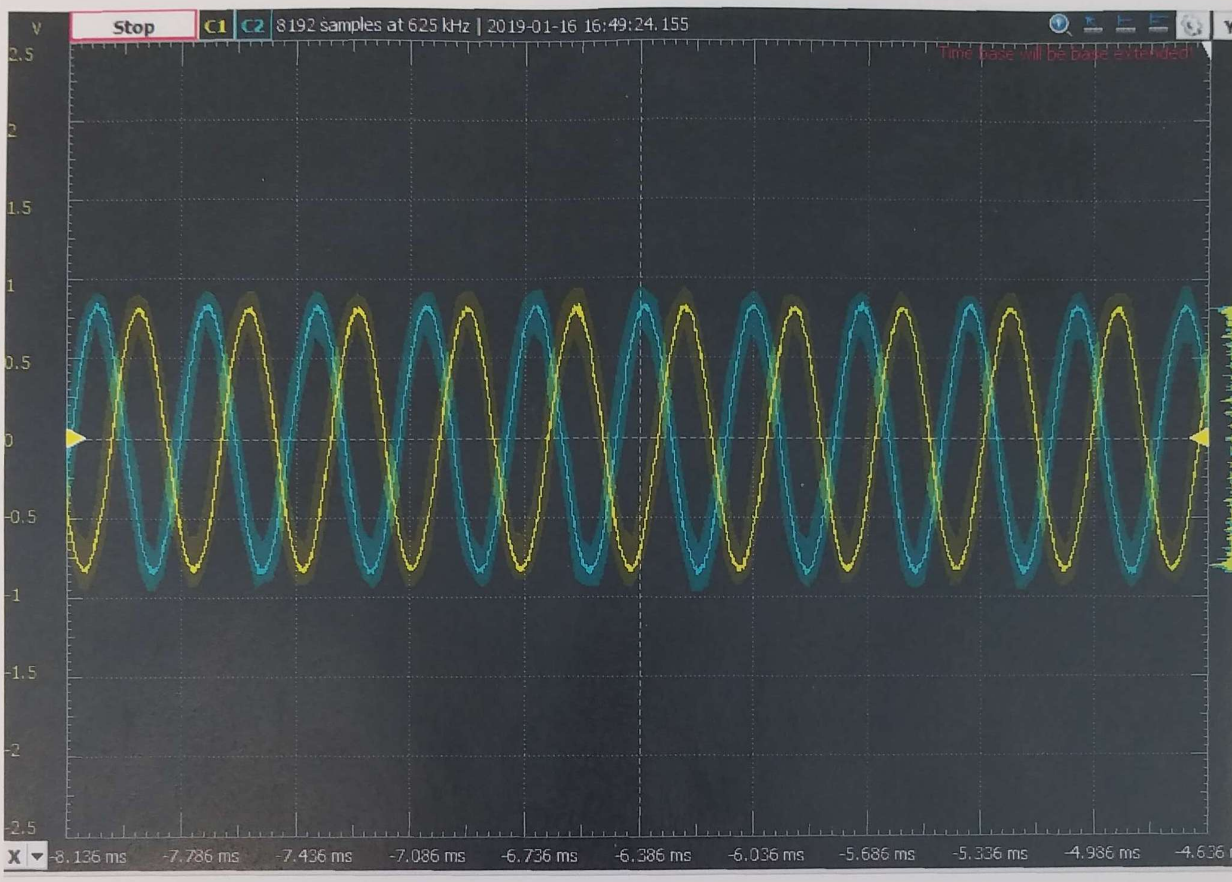


Figure 1 3kHz waveform

ok

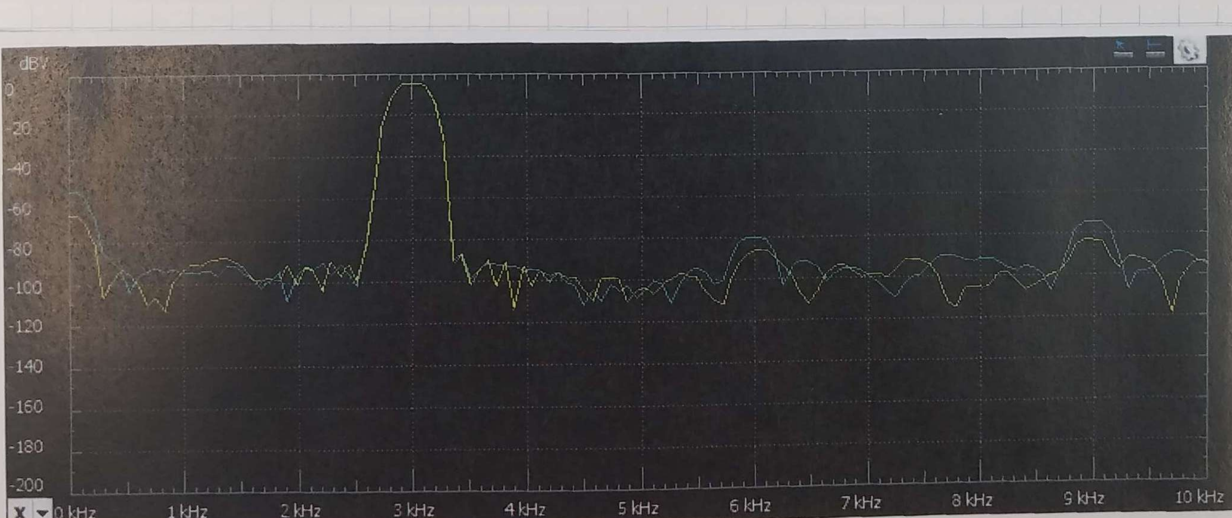


Figure 2 3kHz FFT

ok

1/23/19

4

4. Repeat the above steps with the oscillator frequency set to 2 kHz, 6 kHz, and 9 kHz. (You do not need to sketch the waveforms, but plot spectra for these frequencies.)

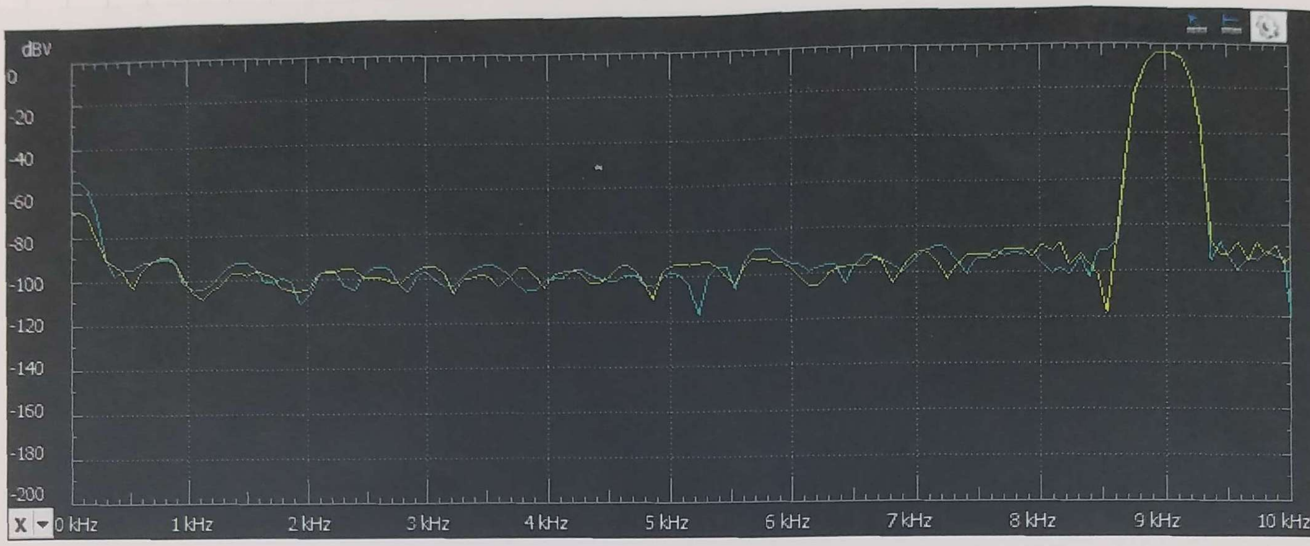


Figure 3 9kHz FFT

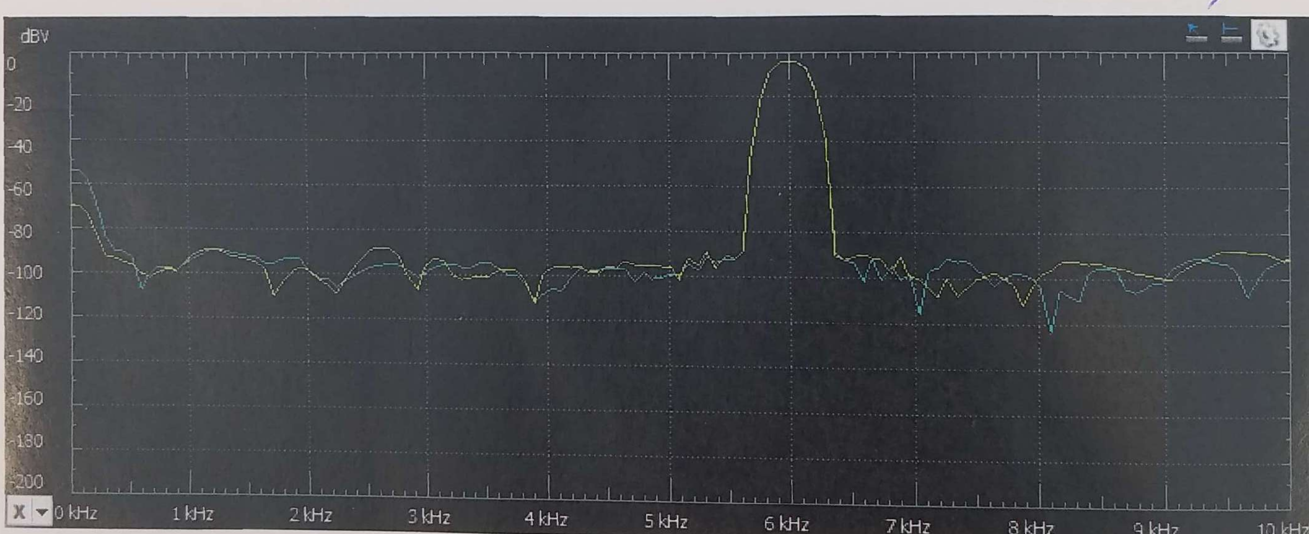


Figure 4 6kHz FFT

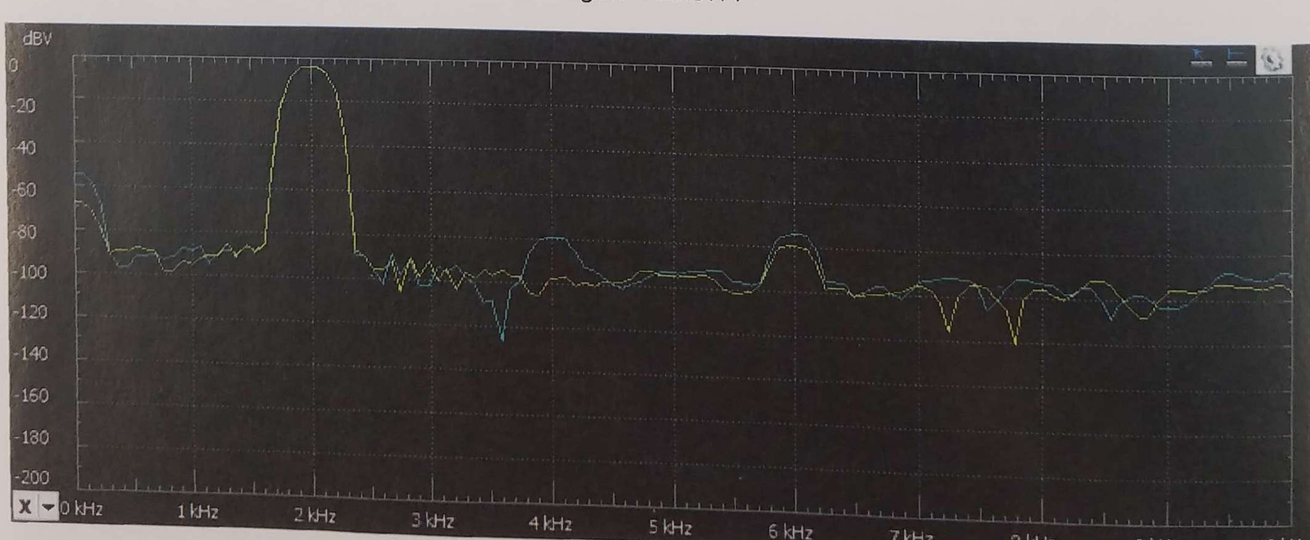


Figure 5 2kHz FFT

1/23/19

5. Using the 3 kHz oscillator, let the oscillator run for 5-10 minutes. Has the output changed?

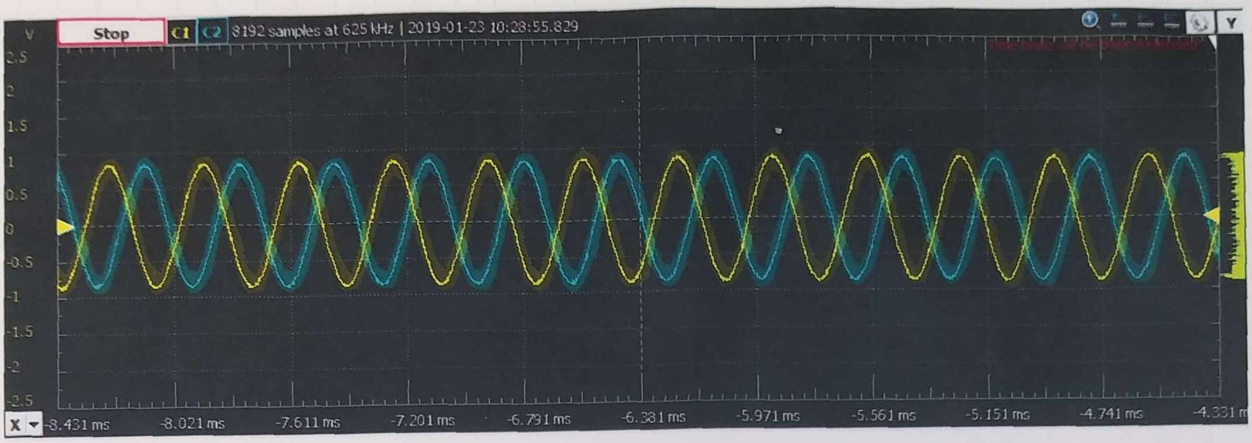


Figure 6 3kHz initial waveform

ok

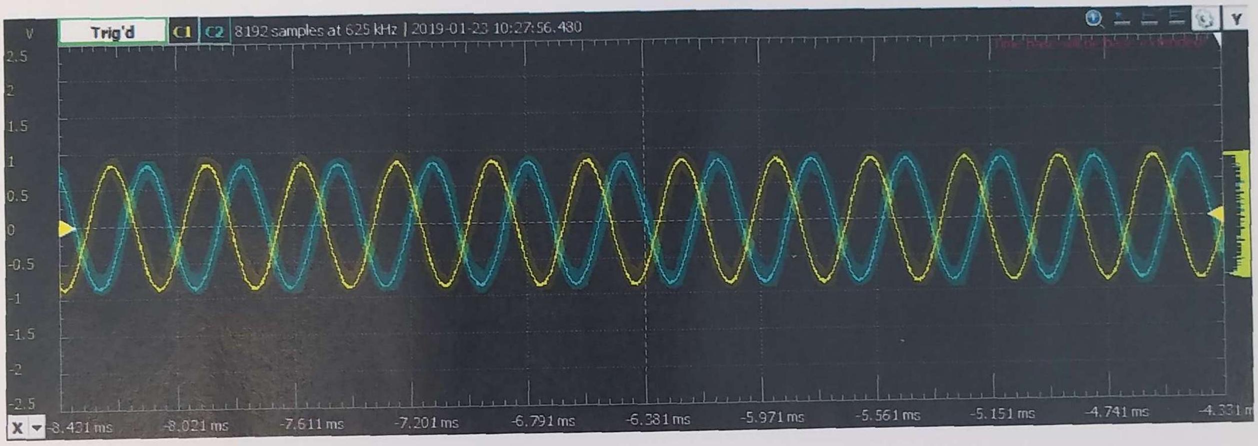


Figure 7 3kHz waveform after 10 minutes

~~No~~ Cannot tell any difference between the waveforms. Even after letting the system run for more than 4 hours, the waveform (not pictured) looks the same as Figures 6 and 7.

Questions

1. Are the measured output frequencies the same as the design values?

As shown in Figures 2-5, the FFT bins are centered around the designed frequencies. When zooming out on the wave form, the FFT ~~bins get skinnier~~ ~~as the~~ bin width decreases and converges on the designed frequency

1/23/19
NWP

2. Are the outputs of the oscillator 90° out of phase?

The left and right channels are not exactly 90° apart. There is a slight delay as the left channel is written to, then the right. The channels are not written in parallel so there will be a slight delay.

3. Explain the changes in the output after several minutes of operation.

I could not see any difference after letting the system run for several ~~min~~ minutes. Even when I came back over an hour later, I could not percieve any change.

4. Look at the assembly code. Do you think you could optimize it to run more efficiently? How?

Looks like there are several NOP between operations where some other process could be done.

Conclusion

The hardest part of the lab was setting everything up. There were some issues with the libraries for the board and I was unable to compile. The ~~t~~ correct libraries and an older compilier were installed allowing the board to compile and run the template program.

One optimization step ~~tt~~ I could do in C without going into assembly is comput Yc, output Yc, and while waiting for the DAC to output the signal, Ys can be computed.

1/23/19