1. Introduction

This document describes the design for a sinusoidal waveform generator prototype. For this generator, an ADC is used to vary the voltage source level and the DAC converts that voltage to a sinusoidal waveform that is emitted through a speaker. Practical application of this technology is frequently utilized across the entire demographic spectrum of consumers, businesses, schools, and government institutions. Most often the described waveform generator is used for audio recording and playback or as a means to drive a mechanical oscillator.

2. Overview

This document discusses, in general, the electrical and software design for the LCD touch screen prototype. It includes the requirements, design details and commentary. A hardware diagram, hardware details, and logic analyzer results are included to reinforce the results and conclusions of the lab. The complete code and look-up table is located in the Appendix.

3. Requirements

The following are the given requirements for the sinusoidal waveform generator prototype:

1. Use the external 16 MHz crystal with the PLL to set the system clock (configure the PLL to produce whichever frequency is most convenient). This is necessary to produce consistent output on the DAC.
2. Connect one of the analog board potentiometers to `ADC0`. Connect the output of the DAC to the analog test board (`AIN` or `AUDIO_IN`), you will be able to hear as the frequency changes. It is not a pleasant noise.
3. Use a timer to configure the ADC to automatically sample the input line (i.e. initiate a conversion) every 2 ms.
4. Make use of an interrupt (no polling) to copy the converted voltage value from the ADC's registers.
5. Produce a sine wave using a look-up table with 40 entries. Appendix – 6.2
6. Configure I^2C to communicate with the DAC in fast mode (400 kHz).
7. Update the DAC as a result of a timer (auto-reload) using an interrupt.
8. Set the DAC output as a result of an interrupt.
9. Use the capacitor to filter the square edges of the DAC waveform.

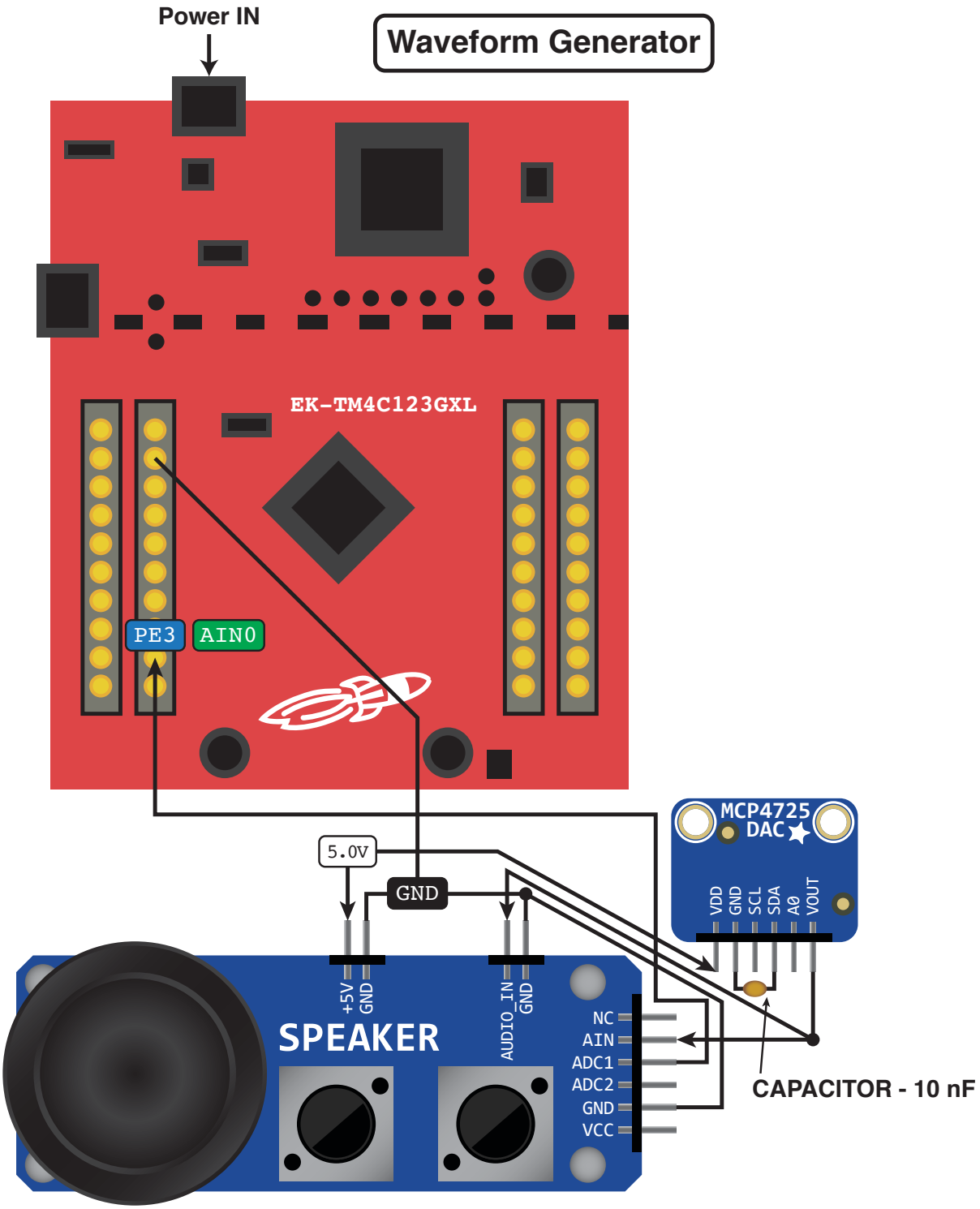
You only need to check to see if the user has adjusted the voltage every 500 ms.

Table 1: List of Timers and Interrupts

Timer / Interrupt Needed	Function	Frequency
Timer A	Initiate ADC conversion.	2 ms
ADC Interrupt	Copy voltage from ADC to appropriate place.	N/A
Timer B	Update the Timer C frequency using data from the ADC interrupt.	500 ms
Timer C + Interrupt	Send to I2C to update the output voltage of the DAC.	Varies

4. Design Details

4.1 Hardware Diagram



4.2 Hardware Details

a. What is the input voltage (V_{DD}) range for the DAC?
   - 2.7 V to 5.5 V (pg. 3 – MCP4725)

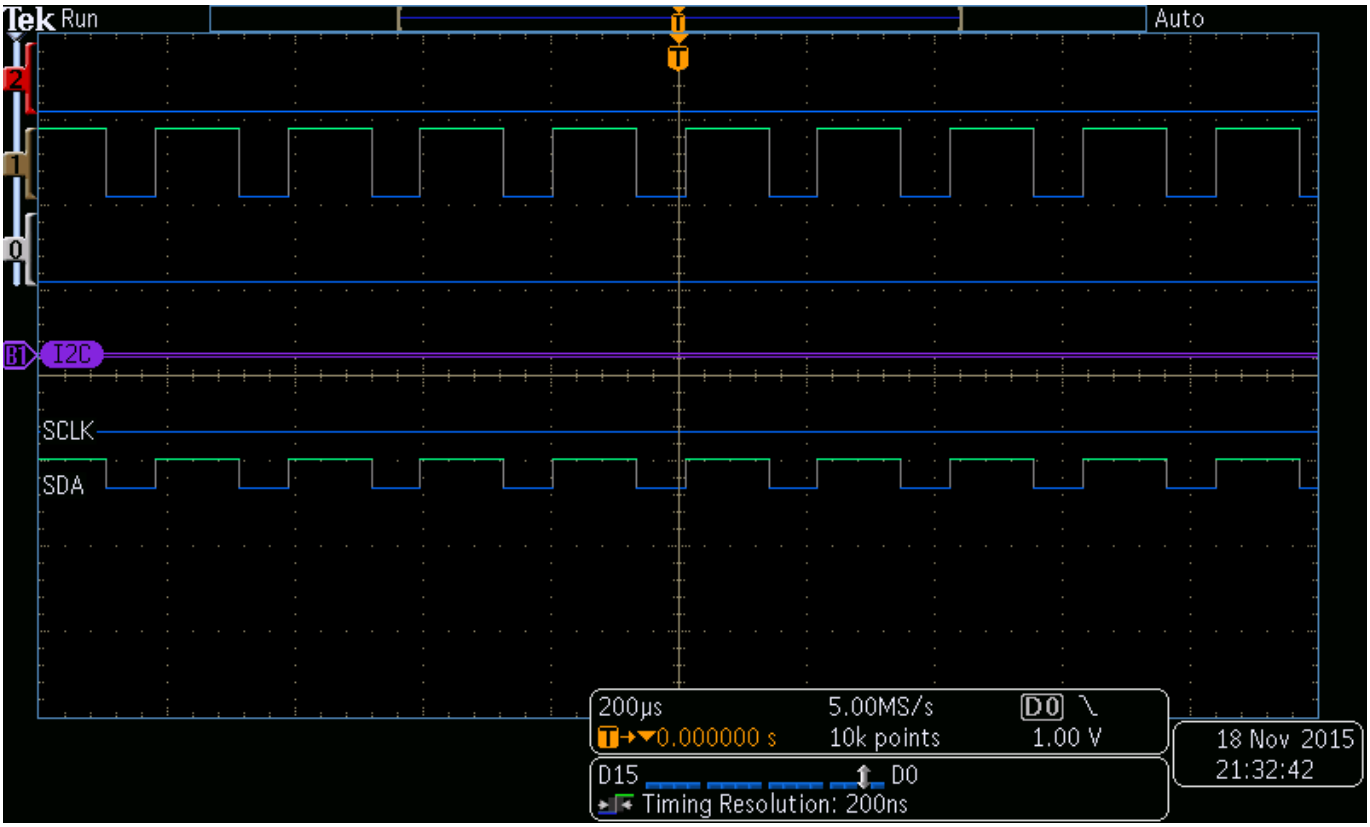
b. What is the function of the `A0` pin of the DAC? Select a value for `A0`. Using your value of `A0`, what is the I^2C address of the DAC? (A2=0, A1=1).
   - "I^2C Address Bit Selection pin (A0 bit). This pin can be tied to VSS or VDD, or can be actively driven by the digital logic levels. The logic state of this pin determines what the A0 bit of the I^2C address bits should be" (pg. 13 – MCP4725).

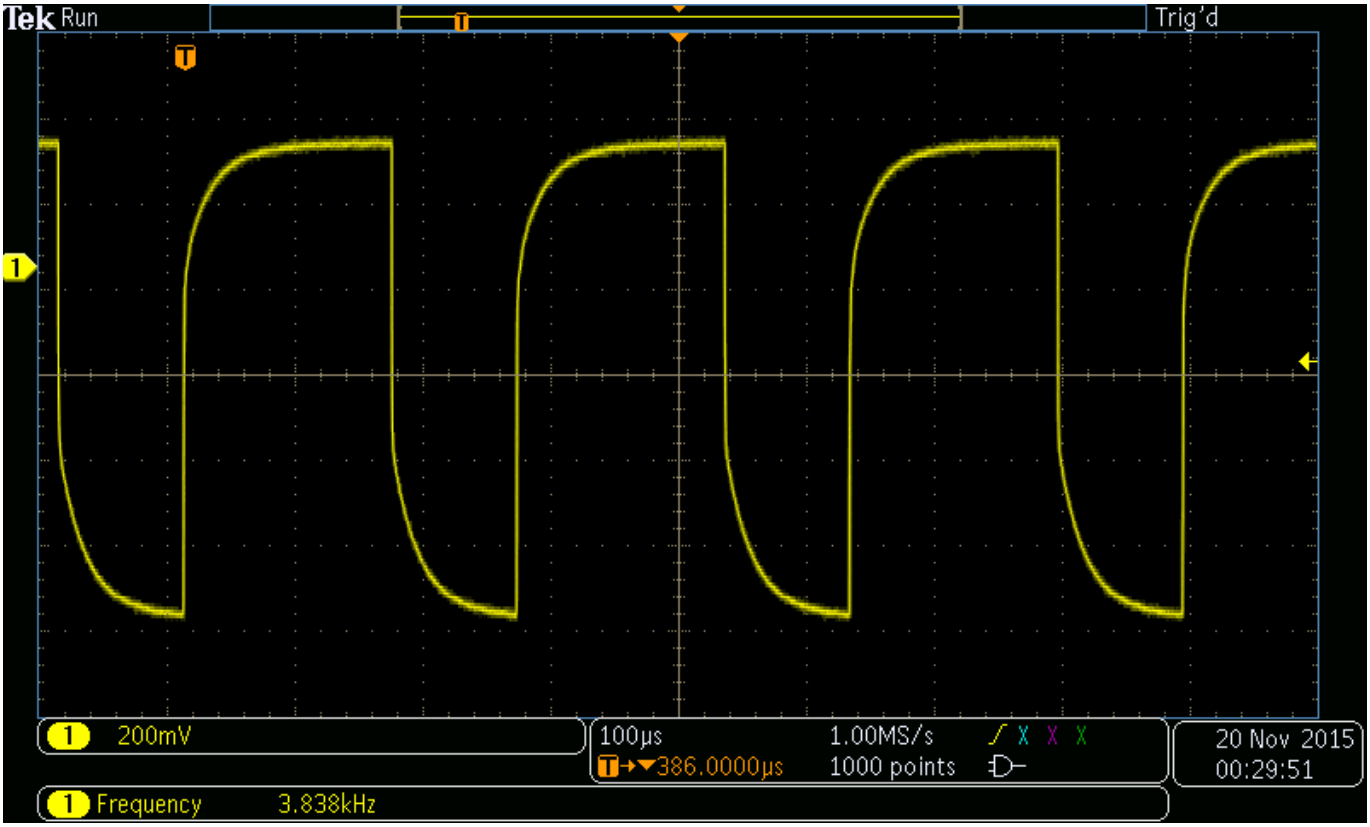
Slave Address for **MCP4725**

Device Code				Address Bits			
1	1	0	0	0 A2	1 A1	0,1 A0	= 0x62,0x63

(pg. 27 – MCP4725)

4.3 Logic Analyzer Output





5. Commentary

5.1 Testing

We see that the voltage determines the frequency output due to the analog board potentiometer which varies the timer on the microcontroller. This in turn determines the output frequency produced by the DAC and emitted through the speaker. Otherwise, due to time constraints that concludes the testing commentary.

5.2 Conclusions

Lab 6 demonstrates the function of interrupts on the Tiva C microcontroller and the I^2C module such that the microcontroller is able to interface with the DAC which in turn is connected to the speaker. The ADC takes input from the variable voltage source and the DAC takes the code generated by the ADC to send the desired frequency to the speaker.

6. Appendix

6.1 Program Code

```
This code outputs a sine wave to the speaker which makes noise I will also attach the sine wave I got on
the oscilloscope
//#include "GPIO.h"
unsigned char *CP = (unsigned char *) 0xE000E000;
unsigned int *count = (unsigned int *) 0x20002000;
unsigned int *PD_int = (unsigned int *) 0x40007000;
unsigned char *SYSCTL = (unsigned char *) 0x400FE000;            //System Control base address
unsigned int *ADC1 = (unsigned int *) 0x40039000;
unsigned int *ADC_CLK = (unsigned int *) 0x400FE000; //offset 638
unsigned int *PB = (unsigned int *) 0x40005000;
unsigned int *I2C0 = (unsigned int *) 0x40020000;
unsigned int *RCGC2_R = (unsigned int *) 0x400FE108;
unsigned char *GPIOA = (unsigned char *) 0x40004000;             //GPIO A base address
unsigned char *GPIOB = (unsigned char *) 0x40005000;             //GPIO B base address
unsigned char *GPIOC = (unsigned char *) 0x40006000;             //GPIO C base address
unsigned char *GPIOD = (unsigned char *) 0x40007000;             //GPIO D base address
unsigned char *GPIOE = (unsigned char *) 0x40024000;             //GPIO E base address
unsigned char *GPIOF = (unsigned char *) 0x40025000;             //GPIO F base address

volatile int Voltage = 0;

void I2C0_POLL()
{
       int temp = I2C0[0x4];
       if((temp &= 0x1) != 1)
              I2C0_POLL();
}

//setup systick
void configSystick()
{
       CP[0x10] = 0;                        //stop timer
       CP[0x14] = 0x40;             //set initial value to last for 2 ms
       CP[0x18] = 0;                        //clear current timer value
       CP[0x10] = 0x5;                      //CLK_SRC,INTEN,ENABLE
}

void I2C_TX()
{
       //1. set slave addr and direction
       I2C0[0x0] = (0x2A << 1);

       //2. tx byte
       I2C0[0x8] = 0x55;

       //3. set master to single tx mode
       I2C0[0x4] = 0x7;

       //4. poll busy bit (busy=1 then still tx)
       I2C0_POLL();

       //5. we're done: restore LR and return to main
}
```

```
void I2C_RX()
{
	int temp = 0x54;

	//1. set slave addr and direction
	I2C0[0x8] = 0x0;
	temp |= 0x1;
	I2C0[0x0] = temp;

	//2. set master to single rx mode
	I2C0[0x4] = 0x7;

	//3. poll busy bit (busy=1 then still tx)
	I2C0_POLL();

	//4. rx byte


	//5. we're done: restore LR and return to main
}

//configure I2C
void configI2C()
{
	//1. enable I2C0 clock: bit twelve of RCGC1
	SYSCTL[0x620] = 1;

	//2. enable GPIO clock (port b): bit one of RCGC2
	SYSCTL[0x608] |= 0x2;

	//3. pin config: alt func, open drain, and digital enable
  //   port b, pins 2 (scl) and 3 (sda)

	GPIOB[0x400] |= 0xFF;
	GPIOB[0x510] |= 0xFF;
	GPIOB[0x420] |= 0xC;
	GPIOB[0x50C] |= 0xC;
	GPIOB[0x51C] |= 0xC;

	//4. I2C0 as master
	I2C0[0x20] = 0x10;

	//5. set I2C0 clock rate (100Kbps or 400Kbps)
	I2C0[0xC] = 0x2;//0x5;

	//write 0x55 to slave 0x2A
	I2C_TX();

	//clear out rx/tx buffer
	I2C0[0x8] = 0x0;

	//read from slave 0x2A

}

//configure port d for an interrupt
void PD_init()
{
	*RCGC2_R |= 0x8; // Enable clock for port D
```

```
	PD_int[0x520/4] = 0x4C4F434B; // Unlock
	PD_int[0x420/4] = 0x1; //Alternative functionality
	PD_int[0x400/4] = 0x0;  //set direction to input
	PD_int[0x51C/4] = 0x0; // enable
	PD_int[0x528/4] = 0xF;
	//analog mode select GPIOAMSEL
}


void delay500ms()
{
	//check to see if user has changed voltage
}

void delay2ms()
{
	int i = 1;
	*count += i;
	if(*count == 250)
	{
		delay500ms();
	}
}

void ADC_init()
{
	ADC_CLK[0x638/4] = 0x2;
	//turns on clock to adc
	ADC1[0x14] = 0x5000;
	//event mux,
	ADC1[0x40] = 0x7;
	//ADSSSMUX
	ADC1[0x8] = 0x8;
	//ADCIM
	CP[0x106] = 0x8;
	//turning NVIC at ADC
	ADC1[0x0] = 0x8;
	//turn on ADC
	//setup adc to use timer and interrupt
}


void ADC1_Handler()
{
	//read from adc fifo queue
	Voltage = ADC1[0x48/4];
	//acknowledge the interrupt
	ADC1[0xC] = 0x8;
}

int main()
{
	int sinArray[] = { 2048, 2377, 2697, 3000, 3279, 3526, 3734, 3899, 4016, 4082
4095, 4055, 3963, 3822, 3635, 3407, 3143, 2851, 2539, 2213
1884, 1558, 1246, 954, 690, 462, 275, 134, 42, 2
15, 81, 198, 363, 571, 818, 1097, 1400, 1720, 2048};
	int i=0;
	configI2C();
	ADC_init();
	configSystick();
```

```
        *count = 0;
        while(1)
        {
                for(;i<50;i++)
                {
                        I2C0[0x0] = 0xC4;
                        while( I2C0[0x4] & 0x40){};
                        I2C0[0x8] = sinArray[i];
                        delay2ms();
                }
                if((CP[0x10]>>16) == 1)
                {
                        delay2ms();
                }
                i=0;
        }
        return 0;
}
```

6.2 Look-Up Table

Entry	Angle	Code	Voltage
0	0.00°	2048	1.650
1	9.23°	2377	1.916
2	18.46°	2697	2.174
3	27.69°	3000	2.419
4	36.92°	3279	2.644
5	46.15°	3526	2.844
6	55.38°	3734	3.013
7	64.62°	3899	3.147
8	73.85°	4016	3.242
9	83.08°	4082	3.296
10	92.31°	4095	3.307
11	101.54°	4055	3.276
12	110.77°	3963	3.202
13	120.00°	3822	3.089
14	129.23°	3635	2.939
15	138.46°	3407	2.755
16	147.69°	3143	2.542
17	156.92°	2851	2.307
18	166.15°	2539	2.055
19	175.38°	2213	1.791
20	184.62°	1884	1.525
21	193.85°	1558	1.262
22	203.08°	1246	1.009
23	212.31°	954	0.773
24	221.54°	690	0.559
25	230.77°	462	0.375

$$\text{Entry}_\Sigma = 40 \quad \text{Angle} = \frac{360°}{\text{Entry}_\Sigma - 1} \times \text{Entry}$$

$$\text{Code} = \left\lceil \left(\frac{V_{\text{REF}}}{2^{12}} \right)^{-1} \left(\frac{V_{\text{REF}}}{2} + \frac{V_{\text{REF}}}{2} \sin(\text{Angle}) \right) \right\rceil$$

$$V_{\text{REF}} = +5.5\text{V} - 2.7\text{V} \quad \text{Voltage} = \frac{\text{Code}}{2^{12} - \text{Entry}}$$

26	240.00°	275	0.223
27	249.23°	134	0.109
28	258.46°	42	0.034
29	267.69°	2	0.002
30	276.92°	15	0.012
31	286.15°	81	0.066
32	295.38°	198	0.161
33	304.62°	363	0.295
34	313.85°	571	0.464
35	323.08°	818	0.665
36	332.31°	1097	0.892
37	341.54°	1400	1.138
38	350.77°	1720	1.399
39	360.00°	2048	1.666