Lab 6

Dr. Ryan Gerdes

ECE 3710

By: Jonathan Tousley

Partner: Aaron Kunz

**Overview:**

The microcontroller outputs a digital sine wave to the DAC, which converts the values to analog and outputs them to the speaker. The speaker also has a potentiometer which sends a signal to the microcontroller that allows the user to alter the frequency of the sine wave to the speaker. This project requires:

- A 12-bit DAC

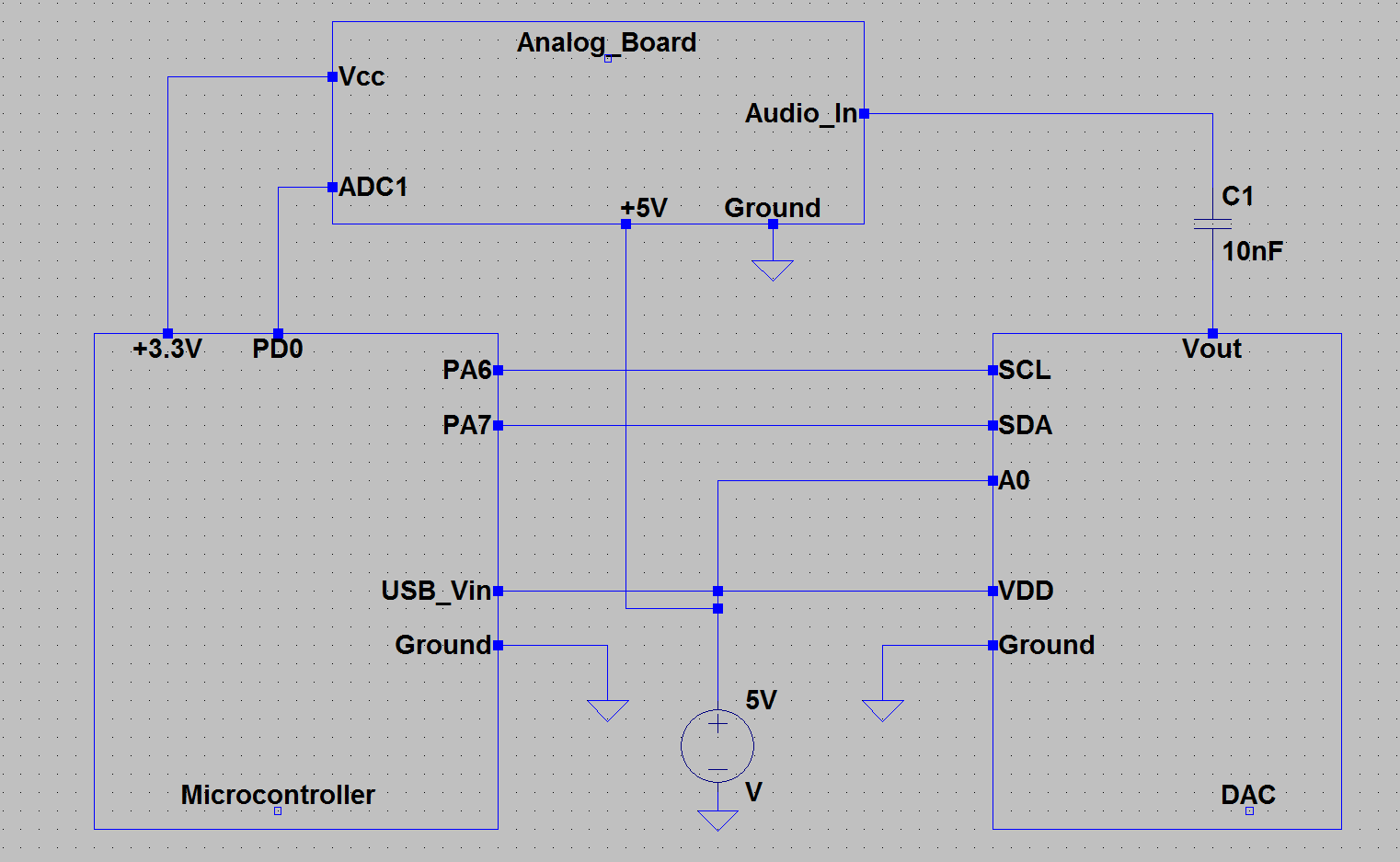
- A 5V voltage source

- A 10nF capacitor

- An analog board with both a potentiometer and speaker

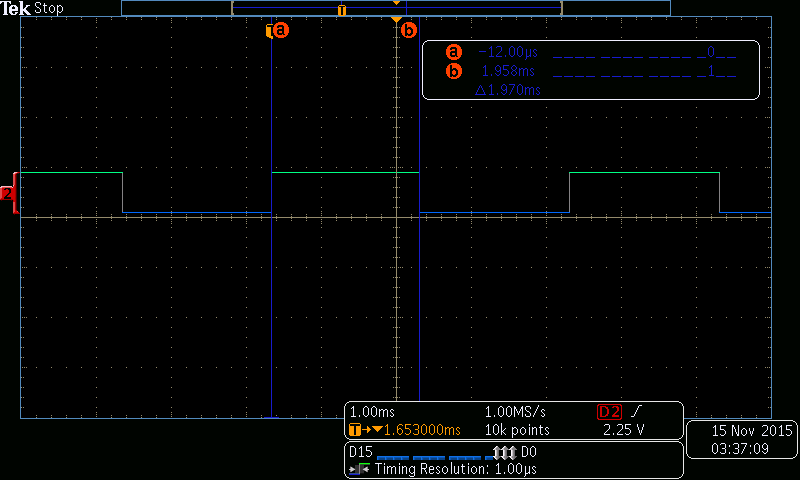
**Hardware Details:**

The microcontroller communicates with the DAC via I2C, requiring only 2 pins. For this, the I2C1 module was used on Port A. A0 was wired to power, making the address of the DAC 0x63 (instead of 0x62). The Vout pin of the DAC was the audio\_in for the speaker, after the value went through the capacitor (which helped even out the rough edges of the waveform). ADC1 was used for the potentiometer and was input to the microcontroller on PD0. See schematic below:

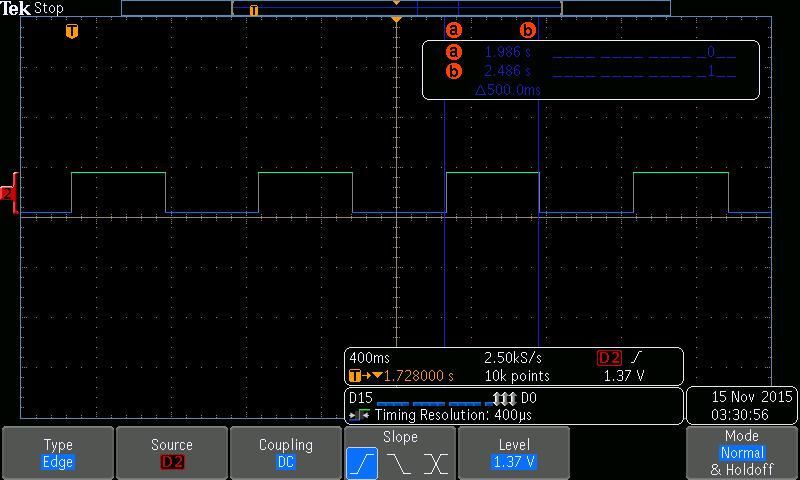


**Software Details:**

The sine wave was generated by creating a 40-element array. Each time timer C expired, the next value (circular) of the array was output to the DAC via I2C. Every 2ms, timer A expired and the current potentiometer value was read into the global value “VOLTAGE”. The logic analyzer verified that delay in this screenshot:

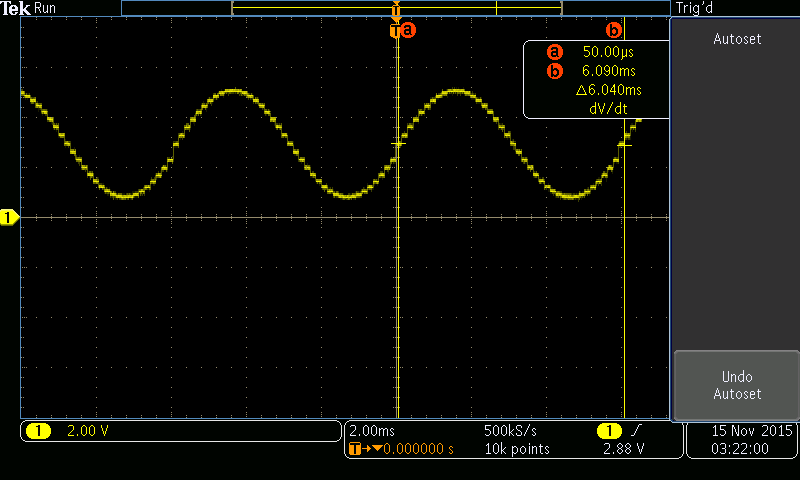
****

Every 500ms, the frequency of Timer C's expiration (the frequency of the sine wave) was checked to make sure it coincided with the value of the potentiometer. The logic analyzer verified that value below:

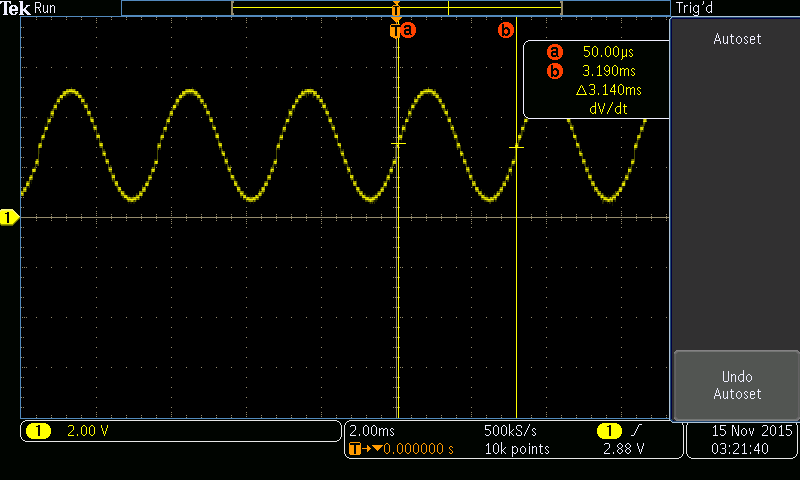


To determine the min and max frequencies of the sine wave, we looked at the frequency for middle C and made that our middle value. Using a slope of 1 (to prevent interger multiplication), that gave a minimum of 0x7D0 at 312.5 Hz (period of 1/312.5 = 3.2ms) and a max of 0x137F at 165.6 Hz (period of 1/165.6 = 6.03ms). Lower values gave higher frequencies.

The logic analyzer verified the lowest frequency:

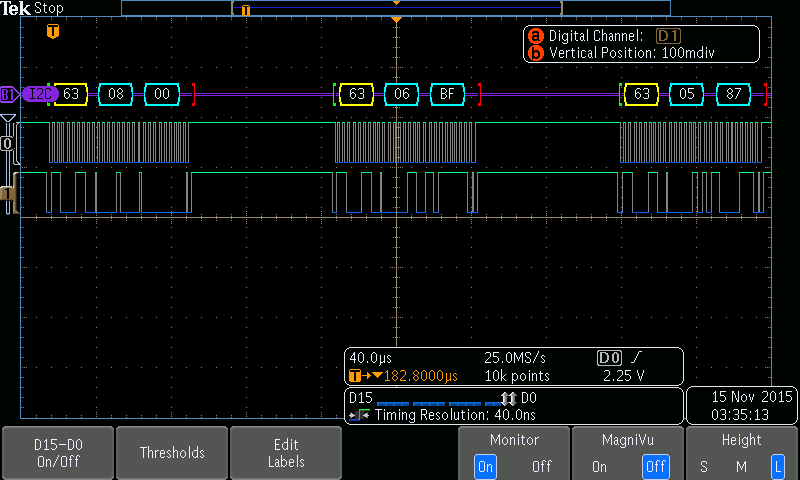


And the highest frequency:



There is a little bit of error between the theoretical and the exact from the logic analyzer, which is okay considering the level of precision we used on the logic analyzer (low). Overall, it is clear to see everything is working properly and as expected within a very close margin.

And here is a screenshot of the I2C module sending 3 consecutive values of the sine table to the DAC (0x800,0x6bf,0x587):



**Timers:**

Although there were more efficient methods of working with the timers, we used a separate timers for each task. Timer 0 acted as timer A and was set to expire every 2ms, which triggered an interrupt to read the ADC module. Timer 1 acted as timer B and expired every 500ms, which triggered an interrupt to check the value read by timer A and configure the expire time for timer C. Timer 2 acted as timer C and expired with the min and max times specified above. During the interrupt, it transmitted the next value of the sine wave to the DAC.

**Requirements:**

1. The input voltage range for the DAC (Vss) is 2.7-5.5V (from datasheet).

2. The function of the A0 pin is the address of the DAC for I2C. We will use A0 = 1, so the I2C address for the DAC will be 0110 0011 (0x63).

**Conclusion:**

Although not a pleasant noise, the speaker does output a sound at apparently the correct frequencies, thanks to some very precise timing and a lot of frequent interrupts. Undoubtedly, this system has a very practical and useful purpose, but at this level, it succeeded in only being very annoying to the other people in the lab. Using I2C was much easier than SPI and it is easy to see why it is so popular.

**Code:**

The rest of this document contains only code. Here it is:

GPIO.h

typedef struct{

volatile unsigned int \_0:1,\_1:1,\_2:1,\_3:1,\_4:1,\_5:1,\_6:1,\_7:1,

\_8:1,\_9:1,\_10:1,\_11:1,\_12:1,\_13:1,\_14:1,\_15:1,

\_16:1,\_17:1,\_18:1,\_19:1,\_20:1,\_21:1,\_22:1,\_23:1,

\_24:1,\_25:1,\_26:1,\_27:1,\_28:1,\_29:1,\_30:1,\_31:1;

}bitReg;

typedef struct{

volatile unsigned char \_0,\_1,\_2,\_3;

}byteReg;

typedef struct{

volatile unsigned short \_0,\_1;

}halfwordReg;

//access register via bit, byte, halfword, and word

typedef union{

bitReg b;

byteReg B;

halfwordReg H;

volatile unsigned int W;

}REG;

typedef struct{

volatile unsigned char DATA\_[1019]; //TI bit addresses

REG DATA; //0x3FC

REG DIR; //0x400

REG IS; //0x404

REG IBE; //0x408

REG IEV; //0x40C

REG IM; //0x410

REG RIS; //0x414

REG MIS; //0x418

REG ICR; //0x41C

REG AFSEL; //0x420

volatile unsigned char RES[220]; //reserved

REG DR2R; //0x500

REG DR4R; //0x504

REG DR8R; //0x508

REG ODR; //0x50C

REG PUR; //0x510

REG PDR; //0x514

REG SLR; //0x518

REG DEN; //0x51C

REG LOCK; //0x520

REG CR; //0x524

REG AMSEL; //0x528

REG PCTL; //0x52C

}GPIO;

Main.c

#include "GPIO.h"

GPIO \*PA = (GPIO \*) 0x40004000;

GPIO \*PB = (GPIO \*) 0x40005000;

GPIO \*PC = (GPIO \*) 0x40006000;

GPIO \*PD = (GPIO \*) 0x40007000;

GPIO \*PE = (GPIO \*) 0x40024000;

GPIO \*PF = (GPIO \*) 0x40025000;

unsigned char \*SYSCTL = (unsigned char \*) 0x400FE000;

unsigned char \*CORE = (unsigned char \*) 0xE000E000;

unsigned char \*ADC0 = (unsigned char \*) 0x40038000;

unsigned char \*TM0 = (unsigned char \*) 0x40030000;

unsigned char \*TM1 = (unsigned char \*) 0x40031000;

unsigned char \*TM2 = (unsigned char \*) 0x40032000;

unsigned char \*I2C1 = (unsigned char \*) 0x40021000;

unsigned char tabAdd = 0;

unsigned short SINTABLE[40] =

{

0x800,0x940,0xa78,0xba1,0xcb3,

0xda7,0xe78,0xf20,0xf9b,0xfe6,

0xfff,0xfe6,0xf9b,0xf20,0xe78,

0xda7,0xcb3,0xba1,0xa78,0x940,

0x800,0x6bf,0x587,0x45e,0x34c,

0x258,0x187,0xdf,0x64,0x19,

0x0,0x19,0x64,0xdf,0x187,

0x258,0x34c,0x45e,0x587,0x6bf

};

void enableClock(void);

void enablePortA(void);

void enablePortB(void);

void enablePortC(void);

void enablePortD(void);

void enablePortE(void);

void enablePortF(void);

void enableADC0(void);

void enableTM0(void);

void enableTM1(void);

void enableTM2(void);

void enableI2C1(void);

void configurePorts(void);

volatile unsigned int VOLTAGE = 0;

void ADC0SS0\_Handler(void)

{

ADC0[0x00C] |= 0x01; //ack interrupt

VOLTAGE = \*(int\*)&ADC0[0x48]; //Result FIFO

}

void TIMER1A\_Handler(void)

{

TM1[0x024] |= 0x1; //ack

TM2[0x00C] &= 0x00; //GPTMCTL

\*(int\*)&TM2[0x028] = VOLTAGE + 0x7D0; // change frequency

TM2[0x00C] |= 0x1; //start timer

}

void TIMER2A\_Handler(void)

{

unsigned short Tx = SINTABLE[tabAdd++];

unsigned char B1 = (Tx >> 8);

unsigned char B2 = (Tx & 0xFF);

if(tabAdd == 39){

tabAdd = 0;

}

TM2[0x024] |= 0x1; //ack

while((I2C1[0x4] & 0x1) == 0x1); //wait MCS BUSY

I2C1[0x008] = B1; //MDR

while((I2C1[0x4] & 0x40) == 0x40); // wait MCS BUSBSY

I2C1[0x004] = 0x03; //MCS

while((I2C1[0x4] & 0x1) == 0x1); //wait MCS BUSY

I2C1[0x008] = B2; //MDR

I2C1[0x004] = 0x05; //MCS

while((I2C1[0x4] & 0x1) == 0x1); //wait MCS BUSY

}

int main(void)

{

configurePorts();

while(1);

}

void configurePorts()

{

enableClock();

enablePortA();

enablePortD();

enableADC0();

enableTM0();

enableI2C1();

enableTM1();

enableTM2();

}

void enableClock(){

unsigned int i;

// \*(int\*)&SYSCTL[0x060] = 0x8E3D40; //sysclk 16MHz

\*(int\*)&SYSCTL[0x060] = 0x24E1540; //sysclk 40 MHz

for(i = 0; i < 0xFF; i++);

SYSCTL[0x608] = 0x09; //enable PA PD PF

SYSCTL[0x638] |= 0x01; //enable ADC0

SYSCTL[0x604] |= 0x07; //TM0 TM1 TM2

SYSCTL[0x620] |= 0x02; // I2C1

\*(int\*)&CORE[0x100] = 0xA04000; //NVIC interrupt ADC0SS0 TM1A TM2A

}

void enablePortA(void)

{

PA->AFSEL.B.\_0 = 0xC0;

PA->DEN.B.\_0 = 0xC0;

PA->DIR.B.\_0 = 0xC0;

PA->PCTL.W = 0x33000000;

PA->ODR.b.\_7 = 0x1;

PA->PUR.B.\_0 |= 0xC0;

}

void enablePortD(void)

{

PD->LOCK.W = 0x4C4F434B; //GPIO Unlcok

PD->CR.B.\_0 = 0x80;

PD->DEN.B.\_0 = 0x00;

PD->DIR.B.\_0 = 0x00;

PD->AMSEL.b.\_0 = 0x1;

}

void enableADC0(void)

{

ADC0[0x0] = 0x00; //ADC0 disable

ADC0[0x8] |= 0x01; //SS0 interrupt enable

ADC0[0x14] = 0x05; //ADCEMUX trigger SS0 set timer

ADC0[0x040] = 0x7; //ADCSSMUX0 select AIN7

ADC0[0x44] |= 0x06; //ADCSSCTL0 interrupt enable, 1st sample is end

ADC0[0x0] |= 0x01; //ADC SS0 enable

}

//Timer A

void enableTM0(void)

{

TM0[0x00C] &= 0x00; //GPTMCTL

TM0[0x0] &= 0x00; //GPTMCFG

TM0[0x4] = 0x02; //periodic

\*(int\*)&TM0[0x028] = 0x13880; // 2ms = INITIAL / 40 MHz

TM0[0x00C] |= 0x21; //start timer with ADC

}

//Timer B

void enableTM1(void)

{

TM1[0x00C] &= 0x00; //GPTMCTL

TM1[0x0] &= 0x00; //GPTMCFG

TM1[0x4] = 0x02; //periodic

TM1[0x018] |= 0x1; //enable interrupts GPTMIMR

\*(int\*)&TM1[0x028] = 0x1312D00; // 500ms = INITIAL / 40 MHz

TM1[0x00C] |= 0x1; //start timer

}

//Timer C

void enableTM2(void)

{

TM2[0x00C] &= 0x00; //GPTMCTL

TM2[0x0] &= 0x00; //GPTMCFG

TM2[0x4] = 0x02; //periodic

TM2[0x018] |= 0x1; //enable interrupts GPTMIMR

\*(int\*)&TM2[0x028] = 0xFCF; // .1ms = INITIAL / 40 MHz

TM2[0x00C] |= 0x1; //start timer

}

void enableI2C1(void)

{

I2C1[0x004] = 0x00; //I2C disable

I2C1[0x020] |= 0x10; //master function enable

I2C1[0x00C] = 0x4; //TPR = 4

I2C1[0x000] = (0x63 << 1); // slave address = 0x63

}