Lab Report 5

Nathan Tipton, Partner: Tarrin Rasmussen

November 1, 2016

Contents

1 Objectives

The purpose of this lab is to use an analog to digital converter (ADC) and a digital converter (DAC) on the microcontroller to produce a varying frequency sine wave.

2 Overview

For this lab, we will be writing a C program to generate a sinusoid using a DAC. User will be able to select the frequency of the sine wave by changing the voltage level with a potentiometer connected to the ADC.

2.1 Requirements

1. Use the external 16 MHz crystal with the PLL to set the system clock.
2. Connect the output of the DAC to the analog test board.
3. Use a timer to configure the ADC to automatically sample the input line every 2 ms.
4. Make use of an interrupt to copy the converted voltage value from the ADC's registers.
5. Produce a sine wave using a look-up table with 40 entries.
6. Configure I2c to communicate with the DAC in fast mode.
7. Update the DAC as a result of a timer using an interrupt. The frequency range is 100 HZ to 1000 Hz.
8. Set the DAC output as a result of an interrupt.
9. Use the capacitor to filter the square edges of the DAC waveform.
10. Check if user has adjusted voltage every 500 ms.

3 Procedure

3.1 DAC

The DAC converts digital signals into analog.

1. The input voltage range for the DAC is 2.7 V to 5.5 V.
2. The AO pin of the DAC sets the last value of the I2C address. The value of A0 will be 1 or 0 depending on if it is connected to VDD. 1 if it is. This allows you to connect two DAC boards to the same I2C bus pins. Using our value of 0 for the DAC gives us an I2C address of 0x62.

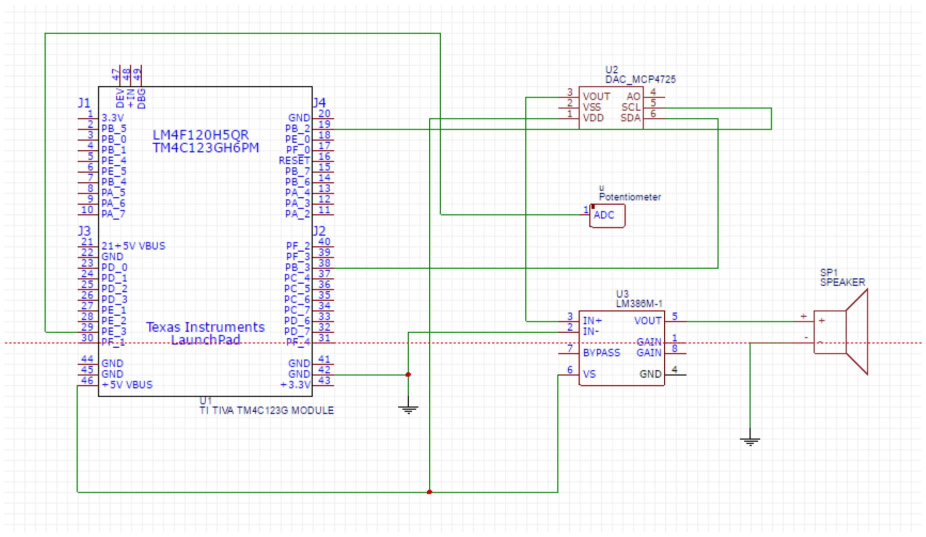


Figure 1: Schematic

3.2 I2C

The microcontroller needs to be connected to the data and clock lines of the DAC. These correspond with pins PB [2:3] on the microcontroller. With I2c, devices are either masters or slaves. The masters drive the clock line. Slaves respond to the master. Only masters can initiate transfers. Multiple slaves can be on one I2C bus. The master uses an unique address for each slave. When a master wants to talk to a slave it first sends a start sequence. Next the master will send the address of the desired slave. Only this slave will pay attention to the data now. The master will now use the data line to transfer data. After sending the desired data the master will send a stop sequence. The following steps are used to configure I2C:

1. Enable Alternate function for the GPIO pins. In this case PB [2:3].
2. Enable open drain for the data line.
3. Enable Digital function for the two pins.
4. Using the mux control register enable the pins for I2C functionality.
5. Initialize the microcontroller for I2C master.
6. Set the SCL clock to Fast- Mode plus.
7. Set the Timer period. This is calculated with equation 1:

$$TPR = (SystemClock/(2*(SCL\_P + SCL\_HP)*SCL\_CLK)) - 1 \quad (1)$$

8. Set your slave address. In this case 0x62 for our DAC.

3.3 ADC

To configure the ADC:

1. Set up alternate function for the AIN0 pin, PE3.
2. Disable the digital mode for this pin, making it Analog.
3. Select Analog mode for the pin.
4. Disable the desired sequencer you are using for you ADC module. We used sequencer 3.
5. Set the trigger event for you sequencer. In this case a timer.
6. Set input channel. AIN0
7. Set the interrupt mask. This enables the sequencer to interract with the interrupt controller.
8. Enable your sequencer.
9. Enable the interrupt for your timer that triggers the sequencer.

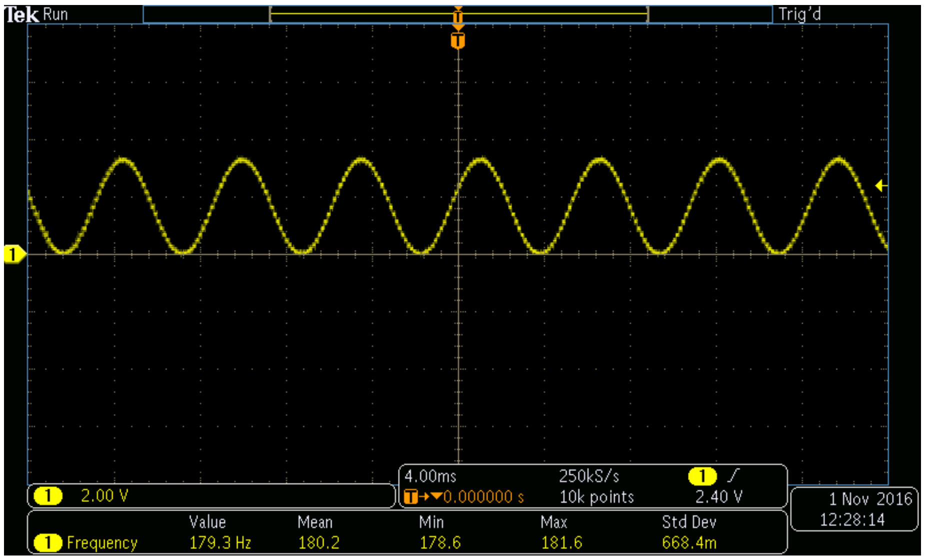
3.4 Music

In order to create music, the frequency of all the desired notes had to be found. This is easily found with a quick internet search. Equation 2 was used to find the delay values needed to produce the desired frequencies. An interrupt triggered with an onboard switch is used to start or stop the music.

$$Frequency = -177.7ln(delay) + 1449.5 \tag{2}$$

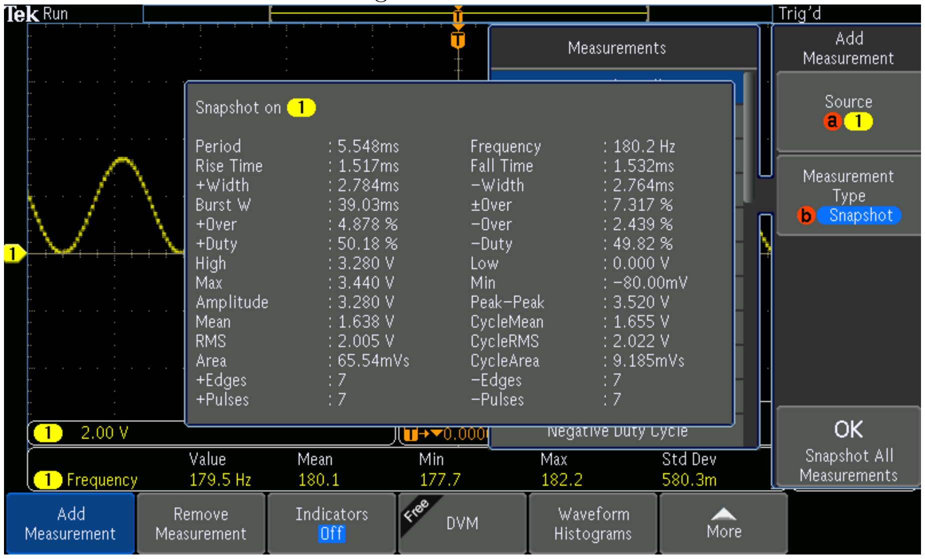
3.5 Testing

Using the Logic Analyzer, we verified slave address and data were being sent via I2C. We used the oscilloscope to capture the sine wave being produced as seen in figure 2.



	Value	Mean	Min	Max	Std Dev
1 Frequency	179.3 Hz	180.2	178.6	181.6	668.4m

Figure 2: sine wave



Snapshot on 1

Period	: 5.548ms	Frequency	: 180.2 Hz
Rise Time	: 1.517ms	Fall Time	: 1.532ms
+Width	: 2.784ms	−Width	: 2.764ms
Burst W	: 39.03ms	±Over	: 7.317 %
+Over	: 4.878 %	−Over	: 2.439 %
+Duty	: 50.18 %	−Duty	: 49.82 %
High	: 3.280 V	Low	: 0.000 V
Max	: 3.440 V	Min	: −80.00mV
Amplitude	: 3.280 V	Peak−Peak	: 3.520 V
Mean	: 1.638 V	CycleMean	: 1.655 V
RMS	: 2.005 V	CycleRMS	: 2.022 V
Area	: 65.54mVs	CycleArea	: 9.185mVs
+Edges	: 7	−Edges	: 7
+Pulses	: 7	−Pulses	: 7

	Value	Mean	Min	Max	Std Dev
1 Frequency	179.5 Hz	180.1	177.7	182.2	580.3m

Figure 3: Measurement snapshot

4 Conclusion

Initally, we were using the 16Mhz clock and the DAC in fast mode. However we realized this did not provide us with the range of frequencies that we wanted to produce. In order to get our desired range we changed the clock to 80 Mhz and set the DAC to fast plus mode (1 Mbps). Doing this greatly increased the range of frequencies we could produce. We had a difficult time getting the I2c to work. We realized we were sending the data wrong. We were trying to send 2 bytes and getting overflow. We also were not sending the start, stop, run sequences correctly. This lab was diifficult for us. Errors and bug were constantly coming up. We were able to create the sine wave and vary the frequency using the potentiometer. Using delays we were also able to create the desired frequencies to play Mary had a little lamb. The delays are calculated using the voltages. However in trying to implement interrupts for the program, everything would stop working. We have been unable to debug this problem before the lab due date. However our current implementation does meet many of the requirements. If given more time I am confident we could implement the required interrupts.

5 Appendix

5.0.1 Code

```
#include "TM4C123.h"



unsigned int adcc;
unsigned int adcc5;
unsigned char i = 0;
unsigned short j;
unsigned short l;
unsigned char m = 0;
char en = 0;
unsigned short lut[40] = {
        2048,2369,2681,2978,3252,3497,3705,3873,3996,4071,
        4095,4071,3996,3873,3705,3497,3252,2978,2681,2369,
        2048,1728,1416,1119,845,600,392,224,101,26,
        0,26,101,224,392,600,845,1119,1416,1728
        };


unsigned short music[26] = { 545, 667, 800, 667, 545, 545, 545, 667, 667, 667,
        545, 385, 385, 545, 667, 800, 667, 545, 545, 545,
        545, 667, 667, 545, 667, 800 };
```

```
void ADC0SS3_Handler() {







        adcc = ADC0−>SSFIFO3;
        ADC0−>ISC = (1 << 3);
}




void GPIOF_Handler() {
        int z;
 for (z = 0; z < 2500000; z++) {}
         GPIOF−>ICR = 0x10;
if(en==0){en=1;
        m=0;}
        else en=0;




}
//void TIMER1A_Handler(){
//// adcc5 = adcc/250;
//// adcc=0;
// TIMER1−>ICR = (1 << 3);
//}






int main(){


        //
        // Clock init
        //

        //RCC 16 MHz PLL
        SYSCTL−>RCC = 0x7CE1540;
        SYSCTL−>RCC2 = (0xC10 << 20);

        //Enable clock to GPIO modules: Port E, B, F (338)
        SYSCTL−>RCGCGPIO |= 0x32;

        //Enable ADC clock for ADC module 0 (350)
        SYSCTL−>RCGCADC |= 1;

        //Enable Timer0 and 1 Clock
        SYSCTL−>RCGCTIMER |= 0x1;
        SYSCTL−>RCGCTIMER |= (1<<1);

        //Enable I2C clock (346)
        SYSCTL−>RCGCI2C |= 0x1;
        //
        //ADC initialization (814)
        //Register Map (815)
        //

        //Enable alternate function PE3 (668, 1337)
        GPIOE->AFSEL |= (1 << 3);

        //Analog Enable (680)
        GPIOE->DEN &= ~(1 << 3);

        //GPIO Analog mode select (684)
        GPIOE->AMSEL |= (1 << 3);


        //
        // Sequencer3 init
        //






        //Disable Sequencer
        ADC0->ACTSS &= ~(1 << 3);

        //Set Trigger Event
        ADC0->EMUX = (0x5 << (3 * 4));

        //Set Input Channel
        ADC0->SSMUX0 = 0;

        //Configure Sample
        ADC0->SSCTL3 = 0x6;

        //Set Mask
        ADC0->IM = (1 << 3);

        //Enable Sequencer
        ADC0->ACTSS |= (1 << 3);

        //Enable Interrupt
        NVIC->ISER[0] |= (1 << 17);


//TIMER1->CTL=0;


//TIMER1->TAMR=0x22;


//TIMER1->TAILR=0x1312d00;


//TIMER1->IMR=1;


////Enable Timer
//
// TIMER1->CTL |=1;
// NVIC_EnableIRQ(TIMER1A_IRQn);
```

```
        //
        //Timer 0 init
        //

        //Disable Timer
        TIMER0−>CTL &= ~1;

        //Set Config Register
        TIMER0−>CFG = 0;

        //Set Periodic
        TIMER0−>TAMR |= 0x2;

        //Set Initial Value
        TIMER0−>TAILR = 32000;

        //Set ADC Trigger
        TIMER0−>CTL |= (1 << 5);

        //Enable Timer
        TIMER0−>CTL |= 1;

// //timer B init
// //Disable Timer
// TIMER1−>CTL &= ~1;
//
// //Set Config Register
// TIMER1−>CFG = 0;
//
// //Set Periodic
// TIMER1−>TAMR |= 0x22;
//
// //Set Initial Value
// TIMER1−>TAILR = 32000;
//
// //enable interrupt
// TIMER1−>IMR |=1;
//  //Enable Interrupt
// NVIC−>ISER[0] |= (1 << 21);
//
// //Enable Timer
// TIMER1−>CTL |= 1;


        //
        //I2C Module 0 Config (1010)
        //

        // Enable Alternate function PB2,PB3
        GPIOB−>AFSEL |= 0xC;

        //Enable Open drain for data (673)
        GPIOB−>ODR |= (1 << 3);

        //Pullup for two lines
        //GPIOB−>PUR |= 0xC;

        //Digital Enable
        GPIOB−>DEN |= 0xC;

        //Enable I2c for pins, mux control (686,1344)
        GPIOB−>PCTL = 0x3300;

        //Initialize I2C master (1026)
        //Write 0x00000010
        I2C0−>MCR |= (1 << 4) ;
        //Set SCL clock speed (1021)
        //(16Mhz/(2*(6+4*400000))−1=1=TPR
        I2C0−>MTPR = 0x1;

        //Set Slave Address
        I2C0−>MSA = 0xC4;

//
//Setup GPIO Port F Pin 4
//



//Enable GPIO Clock
//Done above



//Unlock Port
        GPIOF−>LOCK = 0x4C4F434B;

//Unlock All Pins
        GPIOF−>CR = 0xFF;

//Set Pin Direction
        GPIOF−>DIR &= 0xFFFFFFEF; //Pin 4 input

//Disable ALT Function
        GPIOF−>AFSEL &= 0xFFFFFFEF; //Pin 4 GPIO

//Set Drive Strength
        GPIOF−>DR8R |= 0x10; //Pin 4 8mA

//Set Pin Function
        GPIOF−>PUR |= 0x10; //Pin 4 pull up

//Digital Enable
        GPIOF−>DEN |= 0x10; //Pin 4 enable

//Set Interrupt Sense
        GPIOF−>IS &= 0xFFFFFFEF; //Pin 4 edge

//Set Interrupt Both Edges
        GPIOF−>IBE &= 0xFFFFFFEF; //Pin 4 controlled by IEV

//Set Interrupt Event
        GPIOF−>IEV &= 0xFFFFFFEF; //Pin 4 falling edge

//Set Interrupt Mask
        GPIOF−>IM |= 0x10; //Pin 2

                //Enable Interrupt
        NVIC−>ISER[0] |= (1 << 30);
        while(1) {
                while(en==1){
                if (m == 26) m = 0;

                for (l = 0; l < 10000; l++) {

                if (i == 40) i = 0;

                I2C0->MDR = lut[i] >> 8;
                I2C0->MCS = 0x3;
                while(I2C0->MCS & (1 << 0)) {}
                if (I2C0->MCS & (1 << 1)) while(1) {}
                I2C0->MDR = lut[i];
                I2C0->MCS = 0x5;
                while(I2C0->MCS & (1 << 6)) {}
                if (I2C0->MCS & (1 << 1)) while(1) {}
                i++;

                for (j = 0; j < music[m]/10; j++) {}

                }

                m++;

        }



                if (m == 26) m = 0;

                for (l = 0; l < 10000; l++) {

                if (i == 40) i = 0;

                I2C0->MDR = lut[i] >> 8;
                I2C0->MCS = 0x3;
                while(I2C0->MCS & (1 << 0)) {}
                if (I2C0->MCS & (1 << 1)) while(1) {}
                I2C0->MDR = lut[i];
                I2C0->MCS = 0x5;
                while(I2C0->MCS & (1 << 6)) {}
                if (I2C0->MCS & (1 << 1)) while(1) {}
                i++;

                for (j = 0; j < adcc/10; j++) {}

                }

                m++;

        }



                }
```