Lab 6
Pengzhao Zhu
Section: 112D

B) Prelab Questions

1. What is the main benefit of using an ADC system with 12-bit results, over an ADC system with 8-bit results? Would there be any reason to use 8-bit results instead of 12-bit results? If so, explain.
   The main benefit of a 12-bit ADC is that we will have a more accurate digital representation of the analog signal. This will give us more reliable result regarding the analog signal.
   One reason to use a 8 bit system is that a 8 bit conversion is faster than 12 bit. Even though 12 bit system is more accurate, we would want to use a 8 bit system if we want shorter conversion time (assuming 8 bit is enough for our system).

2. What is the resolution of a 12-bit signed ADC system, with a voltage range from -1V to 3V? What about the accuracy of the system?
   -1 to 3 V= 4V.
   Resolution (Δ) =smallest change in input that will produce a change at output=(4V/(2^12))=(1/1024) V

   Accuracy=closeness of a measurement to its actual value. It depends on measured value according to Dr. Schwartz's lecture slides.
   If Δ=(1/1024)V=.00097 V and we measured .00200 V, then accuracy = (.00097/.00200)* 100 = 48.5 %. This is a really bad value for accuracy.

3. What voltage references can your XMEGA be configured to use, taking the µPAD into account? For each possible voltage reference, describe a situation in which you would want to use that specific reference.
   Taking the uPad into account. The XMEGA can be configured to use 4 or 5 different voltage references (5 if you include the fact that we can use put external voltage at Port A, then we can use AREF pin on PORT A as reference).
   - **10/11 of bandgap (1.0 V)= This is an internal voltage. We need to use it when there is no external voltage available to use. We can decide which internal source to use depending on the range of analog voltage signal.**
   - **Vcc/1.6=This is an internal voltage, we need to use it when there is no external voltage available to use. We can decide which internal source to use depending on the range of analog voltage signal.**
   - **Vcc/2=This is an internal voltage, we need to use it when there is no external voltage available to use. We can decide which internal source to use depending on the range of analog voltage signal.**
   - **AREFA=External voltage reference on Port A. We can use it when we need bigger range of external voltage. We need to use it when we need external reference voltage (i.e. need to use PORT B ADC, use pin on PORT A as reference).**

- **AREFB= External voltage reference on Port B. We can use it when we need bigger range of external voltage. We need to use it when we need external reference voltage (i.e. need to use PORT A ADC, use pin on PORT B as reference).**

4. What is the correlation between the amount of data points used to recreate the waveform and the overall quality of the waveform?
   The more data points, the more accurate we will be able to create a sine waveform (a better sine wave).

C) Problems Encountered

The first problem I ran into in this lab was getting a decimal number for voltage when I calculate it using the measured ADC value (Part B). The result would round off and I would always just get an integer. To fix the problem, I had to search online for C coding resources and realized I had to set the number type as float when I do the math.

The second problem I ran into in this lab was getting the speaker to work when I was trying to output the note in Part E. I forgot to set the 'Power Down' pin high to prevent shutdown and I couldn't get it to work. After realizing what happened, I set the low true 'Power Down' pin high to prevent the speaker from shutting down.

D) Future Work/Application

I can take the ADC and DAC concepts I learned in this lab and apply it to many different applications in the real world. The world is based on analog signals and continuous with time, so we need to convert analog signals to digital values when we work with continuous signals. By finishing this lab, I can utilize the ADC concepts I learned in this lab to work with real time signals in my future jobs (for examples, digital/analog filters and general DSP). DAC would also be very useful when I need to create waveforms that are not available on a regular waveform generator. In my opinion, this lab is the most useful lab since this is the session where I learn how to manipulate real-time signals.

E) Schematics

N/A

F) Pseudocode/Flowcharts

Part A

Call function to setup 32 Mhz clock

Call function to set up ADC system

While (1) {

while((ADCA_CH0_INTFLAGS & 0x01)!= 0x01);

Read data from ADCA_CH0_RES;

Clear interrupt flag

}

*Function to set up ADC with AREF on Port B as reference. Prescaler of 128. 8 bit signed mode free run.

Enable ADC and ADC Channel 0. Use differential mode with gain of 1.

*Function to configure the microprocessor to run at 32 MHZ

Part B

Call function to setup 32 Mhz clock

Call function to set up ADC system

Call function to set up USART

Call function to set up timer

While(1) {

while((TCC0_INTFLAGS & 0x01) != 0x01);

Set TCC0 CNT to zero.

Clear TCC0 overflow interrupt flag

while((ADCA_CH0_INTFLAGS & 0x01) != 0x01);

Read ADC value from ADCA_CH0_RES;

Clear the ADC Interrupt flag.

```
if (adc < 0) {
        sign='-';
    } else if (adc > 0) {
```

sign='+';

} else if (adc==0) {

sign=' ';

}

Transmit the sign.

Get voltage value.

Transit voltage value.

Transit the necessary parenthesis, letters, and numbers.

Transmit the hex value of the ADC value.

Reset TCC0_CNT to 0

Return 0;

}

*Function to set up ADC with AREF on Port B as reference. Prescaler of 512. 8 bit signed mode free run.

Enable ADC and ADC Channel 0. Use differential mode with gain of 1.

*Function to configure the USART system

*Function to configure the timer system to output every 100 ms

*Write the OUT_CHAR function

*Function to configure the microprocessor to run at 32 MHZ

Part C

Call 32MHZ function

Call function to initialize DAC

Set PA2 as DAC0 output

Set DACA_CH0DATA value to a value corresponding to 1 V

While(1);

*Function to initialize the DAC system. Enable Channel 0, single-channel operation, and use AREF on

PORTB as reference

*Function to configure the microprocessor to run at 32 Mhz

Part D

Initialize sine lookup table with 256 data points.

Call function to set up 32MHZ clock

Call function to initialize timer to output a 1760 sine wave.

Call function to initialize DAC.

Set PA1 as DAC0 output

```
While(1) {
    for (int i=0; i< 256;i++) {    //go through the 512 samples

        while((TCC0_INTFLAGS & 0x01) != 0x01);   //wait for interrupt flag of sample rate to set

        TCC0_INTFLAGS=0x01;   //clears the interrupt flag


        DACA_CH0DATA=Table[i];   //DAC output value according to the formula


        TCC0_CNT=0x00;    //reset TCC0_CNT to 0

        }

}
```

*Function to initialize the DAC system. Enable Channel 0, single-channel operation, and use AREF on PORTB as reference

*Function to configure the microprocessor to run at 32 Mhz

*Function to initialize timer to output sine wave at 1760 Hz

Part E

Initialize sine lookup table with 256 data points.

Call function to set up 32MHZ clock

Call function to initialize timer to output a 1760 sine wave.

Call function to initialize DAC.

Set PA3 as DAC1 output

Set ‘POWER DOWN’ pin as output

Set ‘POWER DOWN’ pin always high to prevent shut down

```
While(1) {
        for (int i=0; i< 256;i++) {     //go through the 512 samples
                while((TCC0_INTFLAGS & 0x01) != 0x01);   //wait for interrupt flag of sample rate to set
                TCC0_INTFLAGS=0x01;   //clears the interrupt flag

                DACA_CH0DATA=Table[i];   //DAC output value according to the formula

                TCC0_CNT=0x00;    //reset TCC0_CNT to 0
                }
}
```

*Function to initialize the DAC system. Enable Channel 1, single-channel operation, and use AREF on PORTB as reference

*Function to configure the microprocessor to run at 32 Mhz

*Function to initialize timer to output sine wave at 1760 Hz

Part F

Initialize sine lookup table with 256 data points.

Call function to set up 32MHZ clock

Call function to initialize timer to output a 1760 sine wave.

Call function to initialize DAC.

Initialize global variable ‘change’ to 2.

Set PA3 as DAC1 output

Set ‘POWER DOWN’ pin as output

Set ‘POWER DOWN’ pin always high to prevent shut down

While(1) {

CHECK:;

Call IN_CHAR

Call OUT_CHAR

If it is not one of the keyboard options. Go back to ‘CHECK’.

If if (input=='S') {

change=change *(-1);   //2 means sine, -2 means sawtooth

goto CHECK;

}

Change PER value depending on the keyboard input (if statements)

Set TCC0_CNT to zero.
if (change==2) {

for(int i=0; i< 175;i++){

for (int i=0; i< 256;i++) {   //go through the 512 samples

while((TCC0_INTFLAGS & 0x01) != 0x01); //wait for interrupt flag of sample rate to be set

TCC0_INTFLAGS=0x01; //clears the interrupt flag

DACA_CH1DATA=Table[i]; //DAC output value according to the formula

TCC0_CNT=0x00;   //reset TCC0_CNT to 0

}

i++;

}

}

if(change==-2) {

for(int i=0; i< 175;i++){

for (int i=0; i< 256;i++) {   //go through the 512 samples

while((TCC0_INTFLAGS & 0x01) != 0x01); //wait for interrupt flag of sample rate to be set

TCC0_INTFLAGS=0x01; //clears the interrupt flag

float sawtooth=i*(273/17);

DACA_CH1DATA=(int) sawtooth; //DAC output value according to the formula

TCC0_CNT=0x00;   //reset TCC0_CNT to 0

}

i++;

}

*Function to initialize the DAC system. Enable Channel 1, single-channel operation, and use AREF on

PORTB as reference

*Function to configure the microprocessor to run at 32 Mhz

*Function to initialize timer. PER value will be decided in the main code

*Function to initialize USART system

*IN_CHAR function

*OUT_CHAR function

G) Program Code

Part A

```
/* Lab 6 Part A
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This program configures an ADC system (8 bit signed, differential with
gain of 1) and ADC channel 0.
                            This program will continuously read the ADC conversion value.
                            I will then measure the signal with DAD at the output.

 */


#include <avr/io.h>
#include <avr/interrupt.h>
void CLK_32MHZ(void);

void ADC(void);
volatile int16_t adc;

int main(void)
{
       CLK_32MHZ();
       ADC();

   //int16_t adc;    //8 bit adc. y=(adc/51)+.009804.   .5 V= adc of


              while(1) {
              while((ADCA_CH0_INTFLAGS & 0x01)!= 0x01);
              adc=ADCA_CH0_RES;
              ADCA_CH0_INTFLAGS=0x01;
              }

    return 0;
}

void ADC(void) {

       /*
       PORTA_DIRCLR=0b01000010; //PA1 as positive input, PA6 as negative input. used
later for cds cell
       ADCA_CTRLA=0x01; //enable ADC
       ADCA_CTRLB= 0b00010100; //signed mode, free running, and 8 bit right adjusted

       ADCA_REFCTRL=0b00110000; //arefb are the voltage reference of 2.5
       ADCA_PRESCALER=0b00000000; //adc prescaler of 512
       ADCA_CH0_CTRL=0b00000011; //start channel 0 conversion, 1x gain, differential
input signal with gain
       ADCA_CH0_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative

       */
```

```
	/*
	PORTA_DIRCLR=0b01000010; //PA1 as positive input, PA6 as negative input. used
later for cds cell
	ADCA_CTRLA=0x01; //enable ADC
	ADCA_CTRLB= 0b00011100; //signed mode, free running, and 8 bit right adjusted

	ADCA_REFCTRL=0b00110000; //arefb are the voltage reference of 2.5
	ADCA_PRESCALER=0b00000111; //adc prescaler of 512
	ADCA_CH0_CTRL=0b10000011; //start channel 0 conversion, 1x gain, differential
input signal with gain
	ADCA_CH0_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative
	*/



	ADCA_REFCTRL=ADC_REFSEL_AREFB_gc;	//adc reference as PORTB aref. start
scanning on channel 0
	ADCA_PRESCALER=ADC_PRESCALER_DIV128_gc;			//512 prescaler or adc
clock
	ADCA_CTRLB=ADC_CONMODE_bm | ADC_RESOLUTION_8BIT_gc | ADC_FREERUN_bm;	//signed
mode, 12 bit resolution, free run
	PORTA_DIRCLR= 0b01000010; //PA1 as positive input, PA6 as negative input. used
later for cds cell
	ADCA_CH0_CTRL=ADC_CH_GAIN_1X_gc | ADC_CH_INPUTMODE_DIFFWGAIN_gc;
	ADCA_CH0_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative

	ADCA_CTRLA=ADC_ENABLE_bm|ADC_CH0START_bm;



}


void CLK_32MHZ(void)
{

	OSC_CTRL=0x02;	//select the 32Mhz osciliator
	while ( ((OSC_STATUS) & 0x02) != 0x02 );	//check if 32Mhz oscillator is stable
	//if not stable. keep looping

	CPU_CCP= 0xD8;				//write IOREG to CPU_CCP to enable change
	CLK_CTRL= 0x01;						//select the 32Mhz
oscillator
	CPU_CCP= 0xD8;						//write IOREG to CPU_CCP to
enable change
	CLK_PSCTRL= 0x00;					//0x00 for the prescaler

}
```

Part B

```
/* Lab 6 Part B
   Name: Pengzhao Zhu
   Section#: 112D
```

```
   TA Name: Chris Crary
   Description: This program creates a voltmeter than will measure the drop across the
CDS cell every 100 ms. It will then output
                            to Putty.
 */



#include <avr/io.h>
#include <avr/interrupt.h>


void CLK_32MHZ(void);

void ADC(void);
void USART_INIT(void);
void TIMER_INIT(void);
void OUT_CHAR(uint8_t data);

#define BSELHIGH (((4)*((32000000/(16*57600))-1))>>8)   //bscale of -2
#define BSEL ((4)*((32000000/(16*57600))-1))                 //bscale of -2


#define timer_100 (32000000*.1)/1024

int16_t adc;
uint8_t sign;  //for +, -, or neither


float voltage;
float voltage2;
float voltage3;
int int1;
int int2;
int int3;
int int1_send;
int int2_send;
int int3_send;
uint8_t hex1_send;
uint8_t hex2_send;

uint8_t adc_send;



int main(void)
{
       CLK_32MHZ();
       ADC();
       USART_INIT();
       TIMER_INIT();

      //8 bit adc. y=(adc/51)+(1/102).   .5 V= adc of
         //y=(1/819)x + (1/1638) for 12 bit


              while(1) {       //uncomment for full part b code
```

```
		while((TCC0_INTFLAGS & 0x01) != 0x01);   //wait for interrup flag of 100 ms
for TCC0

		TCC0_CNT=0x00;    //reset TCC0_CNT to 0
		TCC0_INTFLAGS=0x01;   //clears the interrupt flag


		while((ADCA_CH0_INTFLAGS & 0x01)!= 0x01);   //wait for adc conversion to be
completed
		adc=ADCA_CH0_RES;      //take adc value
		ADCA_CH0_INTFLAGS=0x01;    //clear adc interrupt flag

		if (adc < 0) {
			sign='-';
		} else if (adc > 0) {
			sign='+';
		} else if (adc==0) {
			sign=' ';
		}

		OUT_CHAR(sign);  //transmit positive or negative sign



	      voltage = ( (((float)adc)/51)+(1/102));      //get floating point voltage
value

		if (voltage<0) {
			voltage=voltage*(-1);                 //so voltage value will always
be positive when i am doing math later
		}

		int1 = (int) voltage;                         //transmit the tenth place
		int1_send = int1+48;                                //from number to ascii
according to the ascii table
		OUT_CHAR(int1_send);

		OUT_CHAR('.');

		voltage2=10*(((float)voltage)-int1);         //transmit the first decimal
place
		int2= (int) voltage2;
		int2_send= int2+48;                                               //from
number to ascii according to the ascii table
		OUT_CHAR(int2_send);

		voltage3=10*(((float)voltage2)-int2);           //transmit the second
decimal place
		int3= (int) voltage3;
		int3_send=int3+48;                                                //from
number to ascii according to the ascii table
		OUT_CHAR(int3_send);

		OUT_CHAR(' ');
		OUT_CHAR('V');
		OUT_CHAR(' ');
		OUT_CHAR('(');
		OUT_CHAR('0');
```

```
		OUT_CHAR('x');

		adc_send= adc>>4;				//take the upper byte of the
8 bit
		adc_send=adc_send & 0x0F;
		if ( adc_send >= 10) {				//if it is a character, add
55 (ascii table)
			hex1_send=adc_send+55;
		} else if (adc_send < 10) {			//if it is a number, add 48
(ascii table)
			hex1_send=adc_send +48;
		}
		OUT_CHAR(hex1_send);

		adc_send= adc;					//take the lower byte of the
8 bit
		adc_send=adc_send & 0x0F;
		if ( adc_send >= 10) {				//if it is a character, add
55 (ascii table)
			hex2_send=adc_send+55;
			} else if (adc_send < 10) {			//if it is a number,
add 48 (ascii table)
			hex2_send=adc_send +48;
		}
		OUT_CHAR(hex2_send);

		OUT_CHAR(')');
		OUT_CHAR(' ');
		OUT_CHAR(' ');
		OUT_CHAR(' ');

		TCC0_CNT=0x00;	//reset TCC0_CNT to 0

		}				//uncomment for full part B code

    return 0;
}



void ADC(void) {

	/*
	PORTA_DIRCLR=0b01000010; //PA1 as positive input, PA6 as negative input. used
later for cds cell
	ADCA_CTRLA=0x01; //enable ADC
	ADCA_CTRLB= 0b00011100; //signed mode, free running, and 8 bit right adjusted

	ADCA_REFCTRL=0b00110000; //arefb are the voltage reference of 2.5
	ADCA_PRESCALER=0b00000111; //adc prescaler of 512
	ADCA_CH0_CTRL=0b10000011; //start channel 0 conversion, 1x gain, differential
input signal with gain
	ADCA_CH0_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative
	*/
```

```c
	ADCA_REFCTRL=ADC_REFSEL_AREFB_gc;	//adc reference as PORTB aref. start
scanning on channel 0
	ADCA_PRESCALER=ADC_PRESCALER_DIV512_gc;			//512 prescaler or adc
clock
	ADCA_CTRLB=ADC_CONMODE_bm | ADC_RESOLUTION_8BIT_gc | ADC_FREERUN_bm;	//signed
mode, 12 bit resolution, free run
	PORTA_DIRCLR= 0b01000010; //PA1 as positive input, PA6 as negative input. used
later for cds cell
	ADCA_CH0_CTRL=ADC_CH_GAIN_1X_gc | ADC_CH_INPUTMODE_DIFFWGAIN_gc;
	ADCA_CH0_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative

	ADCA_CTRLA=ADC_ENABLE_bm|ADC_CH0START_bm;

	}

void USART_INIT(void)
{
	PORTD_DIRSET=0x08;	//set transmitter as output
	PORTD_DIRCLR=0X04;	//set receiver as input

	USARTD0_CTRLB=0x18;	//enable receiver and transmitter
	USARTD0_CTRLC= 0X33; //USART asynchronous, 8 data bit, odd parity, 1 stop bit

	USARTD0_BAUDCTRLA= (uint8_t) BSEL;	//load lowest 8 bits of BSEL
	USARTD0_BAUDCTRLB= (((uint8_t) BSELHIGH) | 0xE0); //load BSCALE and upper 4 bits
of BSEL. bitwise OR them

	PORTD_OUTSET= 0x08;	//set transit pin idle
}

void TIMER_INIT(void) {

	TCC0_CNT=0x0000;	//set CNT to zero
	TCC0_PER=(uint16_t) timer_100;	//timer per value to 100 ms
	TCC0_CTRLA=0b00000111; //timer prescaler of 1024

}


void OUT_CHAR(uint8_t data) {

	while( ((USARTD0_STATUS) & 0x20) != 0x20);			//keep looping if
DREIF flag is not set

	USARTD0_DATA= (uint8_t) data;

}



void CLK_32MHZ(void)
{

	OSC_CTRL=0x02;	//select the 32Mhz osciliator
	while ( ((OSC_STATUS) & 0x02) != 0x02 );	//check if 32Mhz oscillator is stable
	//if not stable. keep looping
```

```
	CPU_CCP= 0xD8;				//write IOREG to CPU_CCP to enable change
	CLK_CTRL= 0x01;						//select the 32Mhz 
oscillator
	CPU_CCP= 0xD8;						//write IOREG to CPU_CCP to 
enable change
	CLK_PSCTRL= 0x00;					//0x00 for the prescaler

}
```

Part C

```
/* Lab 6 Part C
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This program initializes the DAC system and generates a waveform with a 
constant voltage of 1 V.
				I will then measure the signal with the DAD oscilloscope at 
the output.

 */



#include <avr/io.h>
#include <avr/interrupt.h>


void CLK_32MHZ(void);
void DAC(void);


int main(void) {
	CLK_32MHZ();
	DAC();		//initialize DAC

	//VDAC=(CHDATA/0xFFF) x VREF
	PORTA_DIRSET=0x04;	//set PA2 as DAC0 output

	DACA_CH0DATA=1638;	//DAC output value according to the formula

	while(1);

	return 0;
}

void DAC(void) {
	DACA_CTRLA= DAC_ENABLE_bm | DAC_CH0EN_bm ;	//enable DAC, enable channel 0 
output
	DACA_CTRLB=DAC_CHSEL_SINGLE_gc;	//single-channel operation on channel 0
	DACA_CTRLC=DAC_REFSEL_AREFB_gc;	//AREF on PORTB as reference


	/*
```

```
	DACA_CTRLA= 0b00000101;		//enable DAC, enable channel 0 output
	DACA_CTRLB= 0x00;				//single-channel operation on channel 0
	DACA_CTRLC= 0b00011000;			//AREF on PORTB as reference
	*/
}


void CLK_32MHZ(void)
{

	OSC_CTRL=0x02;		//select the 32Mhz osciliator
	while ( ((OSC_STATUS) & 0x02) != 0x02 );	//check if 32Mhz oscillator is stable
	//if not stable. keep looping

	CPU_CCP= 0xD8;				//write IOREG to CPU_CCP to enable change
	CLK_CTRL= 0x01;						//select the 32Mhz
oscillator
	CPU_CCP= 0xD8;						//write IOREG to CPU_CCP to
enable change
	CLK_PSCTRL= 0x00;					//0x00 for the prescaler

}
```

Part D

```
/* Lab 6 Part D
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This program generates a 1760 Hz sine waveform using a look-up data of
256 data points.
				I will then measure the signal at the output using the DAD
oscilloscope.

 */



#include <avr/io.h>
#include <avr/interrupt.h>


void CLK_32MHZ(void);
void DAC(void);
void ADC(void);
void TIMER_INIT(void);

#define timer_freq ((32000000)*(1/450560))

//#define timer_freq ((32000000)*.1)/1024
//double decimal (1/901120);
//double timer=((32000000)*decimal);

const uint16_t Table[]= {
2048,2098,2148,2198,2248,2298,2348,2398,
```

```
2447,2496,2545,2594,2642,2690,2737,2784,
2831,2877,2923,2968,3013,3057,3100,3143,
3185,3226,3267,3307,3346,3385,3423,3459,
3495,3530,3565,3598,3630,3662,3692,3722,
3750,3777,3804,3829,3853,3876,3898,3919,
3939,3958,3975,3992,4007,4021,4034,4045,
4056,4065,4073,4080,4085,4089,4093,4094,
4095,4094,4093,4089,4085,4080,4073,4065,
4056,4045,4034,4021,4007,3992,3975,3958,
3939,3919,3898,3876,3853,3829,3804,3777,
3750,3722,3692,3662,3630,3598,3565,3530,
3495,3459,3423,3385,3346,3307,3267,3226,
3185,3143,3100,3057,3013,2968,2923,2877,
2831,2784,2737,2690,2642,2594,2545,2496,
2447,2398,2348,2298,2248,2198,2148,2098,
2048,1997,1947,1897,1847,1797,1747,1697,
1648,1599,1550,1501,1453,1405,1358,1311,
1264,1218,1172,1127,1082,1038,995,952,
910,869,828,788,749,710,672,636,
600,565,530,497,465,433,403,373,
345,318,291,266,242,219,197,176,
156,137,120,103,88,74,61,50,
39,30,22,15,10,6,2,1,
0,1,2,6,10,15,22,30,
39,50,61,74,88,103,120,137,
156,176,197,219,242,266,291,318,
345,373,403,433,465,497,530,565,
600,636,672,710,749,788,828,869,
910,952,995,1038,1082,1127,1172,1218,
1264,1311,1358,1405,1453,1501,1550,1599,
1648,1697,1747,1797,1847,1897,1947,1997,
};

int main(void) {
	//output frequency=sample rate(Hz)/ size of table
	//how fast you need to sample 512 to get (1/1760) when you finished the whole
table
	//(1/1760)=512(1/x).   x is the number in Hz

	//sample rate(Hz)=output frequency x No. samples


	CLK_32MHZ();
	TIMER_INIT();
	DAC();

	//  int arr[100]={1,2,3,4,5};
	//int size = sizeof(arr)/sizeof(arr[0]);
	// to find number of elements in an array


	PORTA_DIRSET=0x04;   //set PA2 as DAC0 output

	while(1) {

	for (int i=0; i< 256;i++) {      //go through the 512 samples
		while((TCC0_INTFLAGS & 0x01) != 0x01);   //wait for interrupt flag of
sample rate to be set
```

```
			TCC0_INTFLAGS=0x01;   //clears the interrupt flag

			DACA_CH0DATA=Table[i];   //DAC output value according to the formula

			TCC0_CNT=0x00;    //reset TCC0_CNT to 0
			}

	}


	return 0;
}

void DAC(void) {
	DACA_CTRLA= DAC_ENABLE_bm | DAC_CH0EN_bm ;        //enable DAC, enable channel 0
output
	DACA_CTRLB=DAC_CHSEL_SINGLE_gc;   //single-channel operation on channel 0
	DACA_CTRLC=DAC_REFSEL_AREFB_gc;  //AREF on PORTB as reference

}

void TIMER_INIT(void) {

	TCC0_CNT=0x0000;   //set CNT to zero
	TCC0_PER=54;    //timer per value to output 1760 Hz sine wave
	TCC0_CTRLA=TC_CLKSEL_DIV1_gc; //timer prescaler of 1
	//TCC0_CTRLA=TC_CLKSEL_DIV1024_gc;

}



void CLK_32MHZ(void)
{

	OSC_CTRL=0x02;     //select the 32Mhz osciliator
	while ( ((OSC_STATUS) & 0x02) != 0x02 );   //check if 32Mhz oscillator is stable
	//if not stable. keep looping

	CPU_CCP= 0xD8;                       //write IOREG to CPU_CCP to enable change
	CLK_CTRL= 0x01;                                 //select the 32Mhz
oscillator
	CPU_CCP= 0xD8;                                  //write IOREG to CPU_CCP to
enable change
	CLK_PSCTRL= 0x00;                           //0x00 for the prescaler

}
```

Part E

```
/* Lab 6 Part E
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
```

```
   Description: This program generates a 1760 Hz sine waveform using a look-up data of
256 data points.
                             It will then output the signal to the speaker on the analog
backpack continuously.
 */



#include <avr/io.h>
#include <avr/interrupt.h>


void CLK_32MHZ(void);
void DAC(void);
void ADC(void);
void TIMER_INIT(void);

#define timer_freq ((32000000)*(1/450560))

//#define timer_freq ((32000000)*.1)/1024
//double decimal (1/901120);
//double timer=((32000000)*decimal);

const uint16_t Table[]= {
2048,2098,2148,2198,2248,2298,2348,2398,
2447,2496,2545,2594,2642,2690,2737,2784,
2831,2877,2923,2968,3013,3057,3100,3143,
3185,3226,3267,3307,3346,3385,3423,3459,
3495,3530,3565,3598,3630,3662,3692,3722,
3750,3777,3804,3829,3853,3876,3898,3919,
3939,3958,3975,3992,4007,4021,4034,4045,
4056,4065,4073,4080,4085,4089,4093,4094,
4095,4094,4093,4089,4085,4080,4073,4065,
4056,4045,4034,4021,4007,3992,3975,3958,
3939,3919,3898,3876,3853,3829,3804,3777,
3750,3722,3692,3662,3630,3598,3565,3530,
3495,3459,3423,3385,3346,3307,3267,3226,
3185,3143,3100,3057,3013,2968,2923,2877,
2831,2784,2737,2690,2642,2594,2545,2496,
2447,2398,2348,2298,2248,2198,2148,2098,
2048,1997,1947,1897,1847,1797,1747,1697,
1648,1599,1550,1501,1453,1405,1358,1311,
1264,1218,1172,1127,1082,1038,995,952,
910,869,828,788,749,710,672,636,
600,565,530,497,465,433,403,373,
345,318,291,266,242,219,197,176,
156,137,120,103,88,74,61,50,
39,30,22,15,10,6,2,1,
0,1,2,6,10,15,22,30,
39,50,61,74,88,103,120,137,
156,176,197,219,242,266,291,318,
345,373,403,433,465,497,530,565,
600,636,672,710,749,788,828,869,
910,952,995,1038,1082,1127,1172,1218,
1264,1311,1358,1405,1453,1501,1550,1599,
1648,1697,1747,1797,1847,1897,1947,1997
};
```

```
int main(void) {
	//output frequency=sample rate(Hz)/ size of table
	//how fast you need to sample 512 to get (1/1760) when you finished the whole 
table
	//(1/1760)=512(1/x).   x is the number in Hz

	//sample rate(Hz)=output frequency x No. samples


	CLK_32MHZ();
	TIMER_INIT();
	DAC();




	PORTA_DIRSET=PIN3_bm; //set PA3 as DAC1 output
	PORTC_DIRSET=PIN7_bm; //set POWER DOWN pin as output
	PORTC_OUTSET=PIN7_bm; //set POWER DOWN pin always high to prevent shutdown

	while(1) {


		for (int i=0; i< 256;i++) {	    //go through the 512 samples
			while((TCC0_INTFLAGS & 0x01) != 0x01);   //wait for interrupt flag 
of sample rate to be set
			TCC0_INTFLAGS=0x01;   //clears the interrupt flag

			DACA_CH1DATA=Table[i];   //DAC output value according to the formula


			TCC0_CNT=0x00;    //reset TCC0_CNT to 0
		}

	}


	return 0;
}

void DAC(void) {
	DACA_CTRLA= DAC_ENABLE_bm | DAC_CH1EN_bm ;	    //enable DAC, enable channel 1 
output
	DACA_CTRLB=DAC_CHSEL_SINGLE1_gc;   //single-channel operation on channel 1
	DACA_CTRLC=DAC_REFSEL_AREFB_gc;  //AREF on PORTB as reference

}

void TIMER_INIT(void) {

	TCC0_CNT=0x0000;   //set CNT to zero
	TCC0_PER=54;    //timer per value to output 1760 Hz sine wave
	TCC0_CTRLA=TC_CLKSEL_DIV1_gc; //timer prescaler of 1
	//TCC0_CTRLA=TC_CLKSEL_DIV1024_gc;

}
```

```
void CLK_32MHZ(void)
{

	OSC_CTRL=0x02;     //select the 32Mhz osciliator
	while ( ((OSC_STATUS) & 0x02) != 0x02 );   //check if 32Mhz oscillator is stable
	//if not stable. keep looping

	CPU_CCP= 0xD8;                       //write IOREG to CPU_CCP to enable change
	CLK_CTRL= 0x01;                                  //select the 32Mhz
oscillator
	CPU_CCP= 0xD8;                                   //write IOREG to CPU_CCP to
enable change
	CLK_PSCTRL= 0x00;                            //0x00 for the prescaler

}
```

Part F

```
/* Lab 6 Part F
   Name: Pengzhao Zhu
   Section#: 112D
   TA Name: Chris Crary
   Description: This program creates keyboard piano synthesizer with notes in the 6th
octave.
                            It has both a sine and sawtooth mode.
 */


#include <avr/io.h>
#include <avr/interrupt.h>


void CLK_32MHZ(void);
void DAC(void);
void ADC(void);
void TIMER_INIT(void);
void USARTD0_init(void);
uint8_t IN_CHAR(void);
void OUT_CHAR(uint8_t data);

#define BSELHIGH (((4)*((32000000/(16*57600))-1))>>8)   //bscale of -2
#define BSEL ((4)*((32000000/(16*57600))-1))                  //bscale of -2

#define timer_freq ((32000000)*(1/450560))

//#define timer_freq ((32000000)*.1)/1024
//double decimal (1/901120);
//double timer=((32000000)*decimal);

uint8_t input;
int change=2;
```

```
volatile int receive;

const uint16_t Table[]= {
	2048,2098,2148,2198,2248,2298,2348,2398,
	2447,2496,2545,2594,2642,2690,2737,2784,
	2831,2877,2923,2968,3013,3057,3100,3143,
	3185,3226,3267,3307,3346,3385,3423,3459,
	3495,3530,3565,3598,3630,3662,3692,3722,
	3750,3777,3804,3829,3853,3876,3898,3919,
	3939,3958,3975,3992,4007,4021,4034,4045,
	4056,4065,4073,4080,4085,4089,4093,4094,
	4095,4094,4093,4089,4085,4080,4073,4065,
	4056,4045,4034,4021,4007,3992,3975,3958,
	3939,3919,3898,3876,3853,3829,3804,3777,
	3750,3722,3692,3662,3630,3598,3565,3530,
	3495,3459,3423,3385,3346,3307,3267,3226,
	3185,3143,3100,3057,3013,2968,2923,2877,
	2831,2784,2737,2690,2642,2594,2545,2496,
	2447,2398,2348,2298,2248,2198,2148,2098,
	2048,1997,1947,1897,1847,1797,1747,1697,
	1648,1599,1550,1501,1453,1405,1358,1311,
	1264,1218,1172,1127,1082,1038,995,952,
	910,869,828,788,749,710,672,636,
	600,565,530,497,465,433,403,373,
	345,318,291,266,242,219,197,176,
	156,137,120,103,88,74,61,50,
	39,30,22,15,10,6,2,1,
	0,1,2,6,10,15,22,30,
	39,50,61,74,88,103,120,137,
	156,176,197,219,242,266,291,318,
	345,373,403,433,465,497,530,565,
	600,636,672,710,749,788,828,869,
	910,952,995,1038,1082,1127,1172,1218,
	1264,1311,1358,1405,1453,1501,1550,1599,
	1648,1697,1747,1797,1847,1897,1947,1997,
};

int main(void) {

	//output frequency=sample rate(Hz)/ size of table
	//how fast you need to sample 512 to get (1/1760) when you finished the whole
table
	//(1/1760)=512(1/x).   x is the number in Hz

	//sample rate(Hz)=output frequency x No. samples


	CLK_32MHZ();
	TIMER_INIT();
	DAC();
	USARTD0_init();

	PORTA_DIRSET=PIN3_bm; //set PA3 as DAC1 output
	PORTC_DIRSET=PIN7_bm; //set POWER DOWN pin as output
	PORTC_OUTSET=PIN7_bm; //set POWER DOWN pin always high to prevent shutdown

	while(1) {
```

```
		CHECK:;
		input=IN_CHAR();
		OUT_CHAR(input);

		if ((input != 'S') && (input != 'W') && (input != '3') && (input != 'E') &&
(input != '4') && (input != 'R')
		&& (input != 'T') && (input !='6') && (input !='Y') && (input != '7') &&
(input != 'U') && (input != '8') && (input != 'I')) {
			goto CHECK;

		}

		if (input=='S') {
			change=change *(-1);		//2 means sine, -2 means sawtooth
			goto CHECK;
		}

		if ((input=='W') && (change==2)) {
			TCC0_PER=103;
			} else if ((input=='W') && (change==-2)) {
			TCC0_PER=112;
		}

		if ((input=='3') && (change==2)) {
			TCC0_PER=95;
			} else if ((input=='3') && (change==-2)) {
			TCC0_PER=103;
		}

		if ((input=='E') && (change==2)) {
			TCC0_PER=91;
			} else if ((input=='E') && (change==-2)) {
			TCC0_PER=97;
		}

		if ((input=='4') && (change==2)) {
			TCC0_PER=85;
			} else if ((input=='4') && (change==-2)) {
			TCC0_PER=90;
		}

		if ((input=='R') && (change==2)) {
			TCC0_PER=77;
			} else if ((input=='R') && (change==-2)) {
			TCC0_PER=85;
		}

		if ((input=='T') && (change==2)) {
			TCC0_PER=72;
			} else if ((input=='T') && (change==-2)) {
			TCC0_PER=79;
		}

		if ((input=='6') && (change==2)) {
			TCC0_PER=69;
			} else if ((input=='6') && (change==-2)) {
			TCC0_PER=75;
		}
```

```
		if ((input=='Y') && (change==2)) {
			TCC0_PER=61;
			} else if ((input=='Y') && (change==-2)) {
			TCC0_PER=71;
		}

		if ((input=='7') && (change==2)) {
			TCC0_PER=57;
			} else if ((input=='7') && (change==-2)) {
			TCC0_PER=66;
		}

		if ((input=='U') && (change==2)) {
			TCC0_PER=54;
			} else if ((input=='U') && (change==-2)) {
			TCC0_PER=62;
		}

		if ((input=='8') && (change==2)) {
			TCC0_PER=50;
			} else if ((input=='8') && (change==-2)) {
			TCC0_PER=58;
		}

		if ((input=='I') && (change==2)) {
			TCC0_PER=46;
			} else if ((input=='I') && (change==-2)) {
			TCC0_PER=54;
		}

		TCC0_CNT=0x00;



		if (change==2) {
			for(int i=0; i< 150;i++){
				for (int i=0; i< 256;i++) {	//go through the 512 samples
					while((TCC0_INTFLAGS & 0x01) != 0x01);	//wait for
interrupt flag of sample rate to be set
					TCC0_INTFLAGS=0x01;	//clears the interrupt flag

					DACA_CH1DATA=Table[i];	//DAC output value according
to the formula

					TCC0_CNT=0x00;	//reset TCC0_CNT to 0
				}
				i++;

			}
		}




		if(change==-2) {
			for(int i=0; i< 150;i++){
				for (int i=0; i< 256;i++) {	//go through the 512 samples
```

```
                                    while((TCC0_INTFLAGS & 0x01) != 0x01);   //wait for
interrupt flag of sample rate to be set
                                    TCC0_INTFLAGS=0x01;   //clears the interrupt flag

                                    float sawtooth=i*(273/17);
                                    DACA_CH1DATA=(int) sawtooth;   //DAC output value
according to the formula

                                    TCC0_CNT=0x00;    //reset TCC0_CNT to 0


                             }
                             i++;
                      }
               }

       }


       return 0;
}

void DAC(void) {
       DACA_CTRLA= DAC_ENABLE_bm | DAC_CH1EN_bm ;        //enable DAC, enable channel 1
output
       DACA_CTRLB=DAC_CHSEL_SINGLE1_gc;   //single-channel operation on channel 1
       DACA_CTRLC=DAC_REFSEL_AREFB_gc;  //AREF on PORTB as reference

}

void TIMER_INIT(void) {

       TCC0_CNT=0x0000;   //set CNT to zero
       TCC0_PER=0;    //timer per value to output 1760 Hz sine wave
       TCC0_CTRLA=TC_CLKSEL_DIV1_gc; //timer prescaler of 1
       //TCC0_CTRLA=TC_CLKSEL_DIV1024_gc;

}

void USARTD0_init(void)
{
       PORTD_DIRSET=0x08;   //set transmitter as output
       PORTD_DIRCLR=0X04;    //set receiver as input

       USARTD0_CTRLB=0x18;  //enable receiver and transmitter
       USARTD0_CTRLC= 0X33; //USART asynchronous, 8 data bit, odd parity, 1 stop bit

       USARTD0_BAUDCTRLA= (uint8_t) BSEL;    //load lowest 8 bits of BSEL
       USARTD0_BAUDCTRLB= (((uint8_t) BSELHIGH) | 0xE0); //load BSCALE and upper 4 bits
of BSEL. bitwise OR them

       PORTD_OUTSET= 0x08;   //set transit pin idle


}


uint8_t IN_CHAR(void) {
```

```c
	while( (USARTD0_STATUS & 0x80) != 0x80);		//keep looping if DREIF flag is not set

	return USARTD0_DATA;

}

void OUT_CHAR(uint8_t data) {

	while( ((USARTD0_STATUS) & 0x20) != 0x20);			//keep looping if DREIF flag is not set

	USARTD0_DATA= (uint8_t) data;

}

void CLK_32MHZ(void)
{

	OSC_CTRL=0x02;	//select the 32Mhz osciliator
	while ( ((OSC_STATUS) & 0x02) != 0x02 );	//check if 32Mhz oscillator is stable
	//if not stable. keep looping

	CPU_CCP= 0xD8;			//write IOREG to CPU_CCP to enable change
	CLK_CTRL= 0x01;				//select the 32Mhz oscillator
	CPU_CCP= 0xD8;				//write IOREG to CPU_CCP to enable change
	CLK_PSCTRL= 0x00;			//0x00 for the prescaler

}
```

H) Appendix

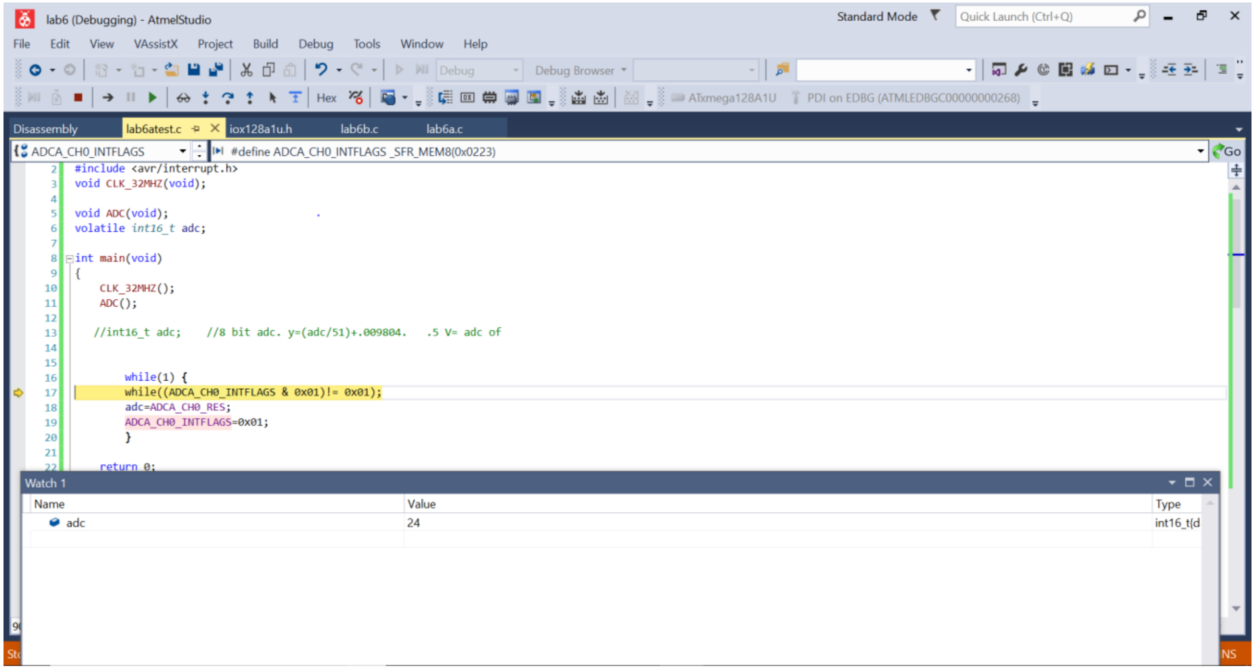

Figure 1: .5 V in differential input (Part A). Could be slightly different due to a Non-DSP ADC

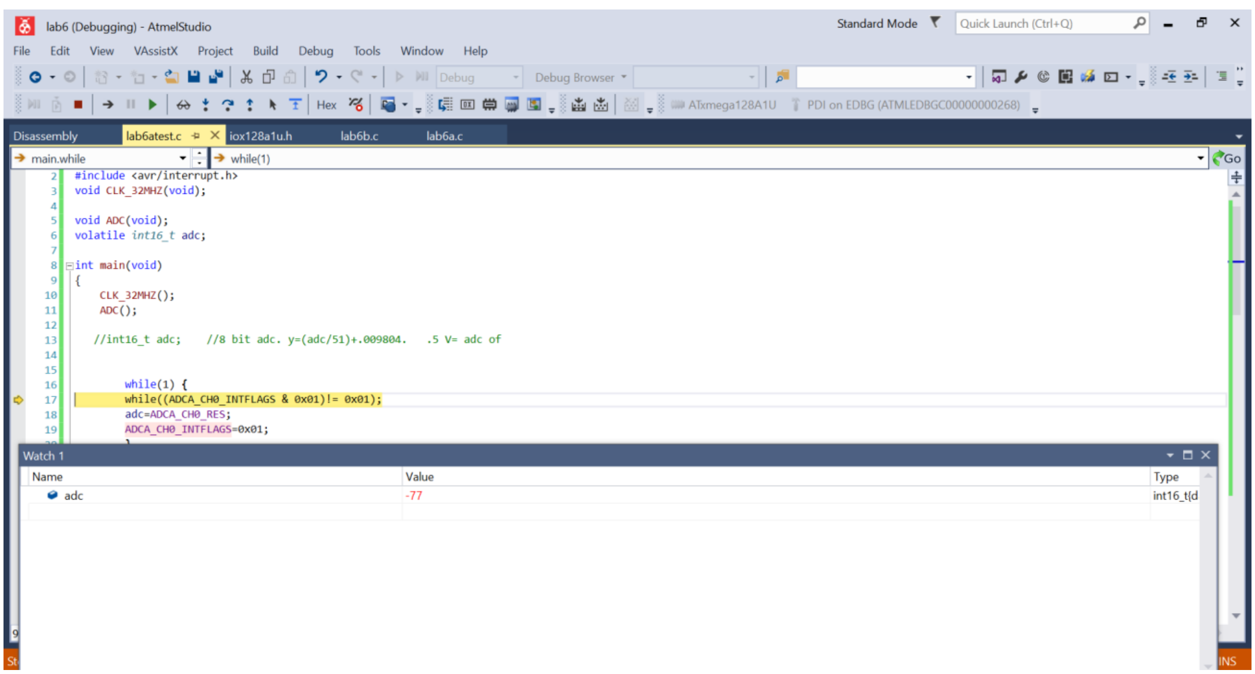

Figure 2: Negative ADC corresponding to -1.5 V in differential input (Part A). Could be slightly different due to a Non-DSP ADC

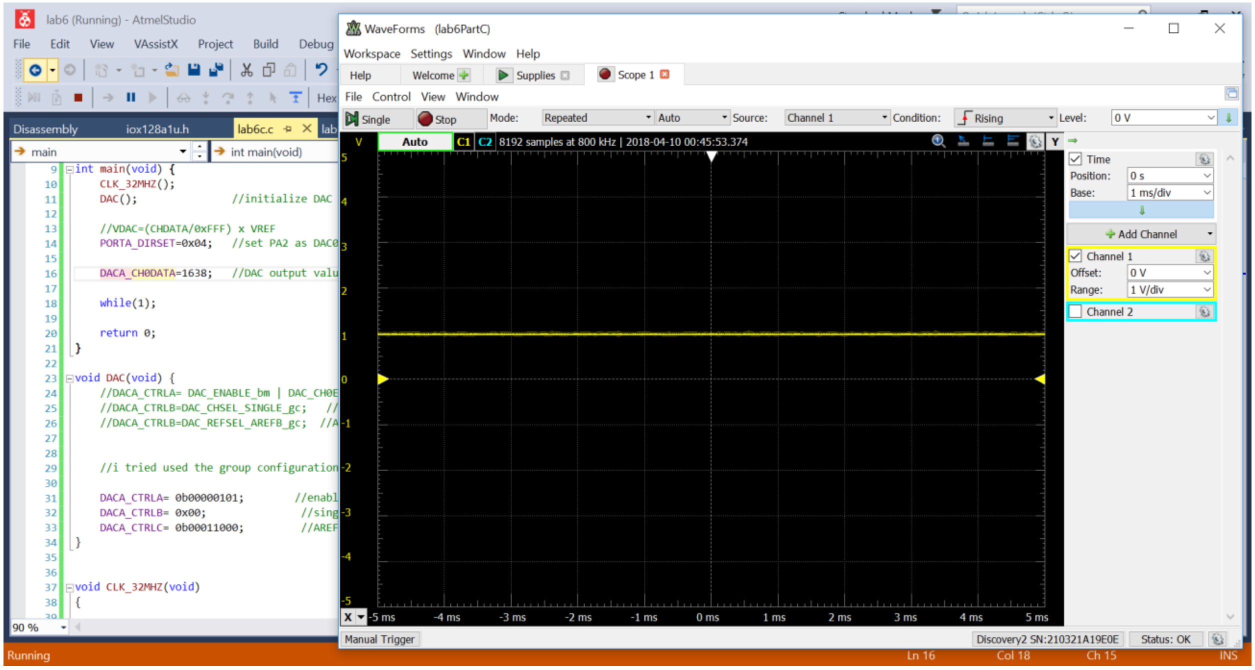

Figure 3: DAC generation of 1 V waveform (Part C)

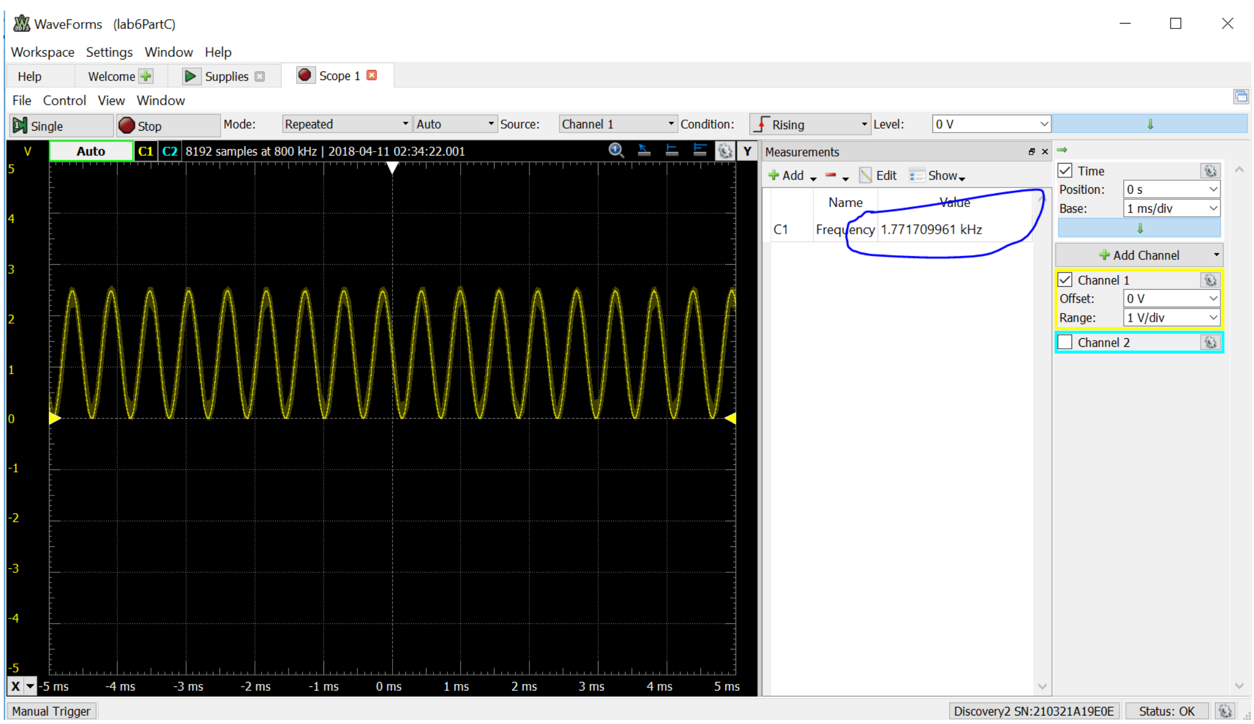

Figure 4: DAC generation of 1760 Sine Wave (Part D)