**REQUIREMENTS NOT MET**

Part 3 and 4: Unable to achieve the desired frequency of 1760Hz.

Part 5: Incomplete.

**PROBLEMS ENCOUNTERED**

Some problems encountered with part 2 of the lab includes the pre-lab question portion where it had to be researched heavily from the manuals and lecture resources to be completed. The problems encountered with part 2 was with figuring out the configuration needed to be done for the tcc0 timer and DAC interrupt. For part 3, the major issue was figuring out how to setup the DMA and get the appropriate frequency which was not met. This required the assistance of PI in explaining the necessary steps to be done. For part 4, it was completed with relatively little issues, but more problems arose in part 5 as I could not get the music from the backpack to work, requiring me to redesign the necessary code for it to function property.

**FUTURE WORK/APPLICATIONS**

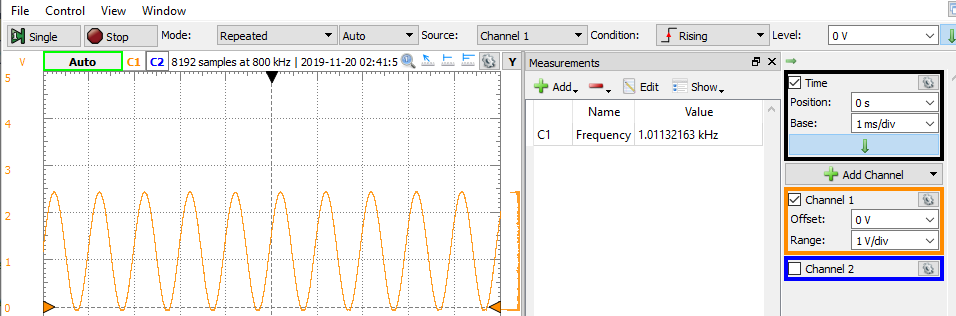
This lab was a good introduction into the implementation and function of DAC and the difference of direct and alternating voltages. This lab is to be the expansion of more complex assembly programs, able to give the users’ another way to interact with the microprocessor, enabling new hardware running on a different power source to be combined. With the power source type are no longer restricting the type of hardware being used, the user can now implement a new functions and techniques that can simplify user’s task or create new uses. Like the subroutine, the way I program is now changed to be inclusive of DAC for more capability of my programs. Having the ability to use DAC allows for me to expand my hardware access to devices. If given more time, the code of the lab could have been more organized and have a much neater layout to further reduce the likelihood of mistakes and further enhance the understanding of the program. With more time a more compacted or efficient communication can be implemented. Additionally, I could have used better instructions to make the code run more efficiently or learn to write more complex programs.

**PRE-LAB EXERCISES**

Part 2: GENERATING A WAVEFORM WITH A LOOKUP TABLE

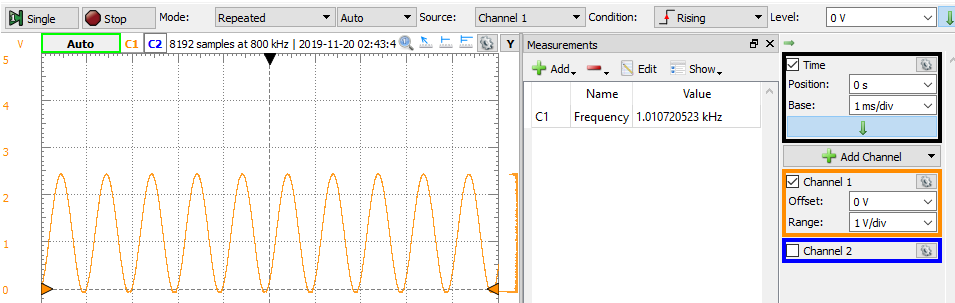
i. Why might you be unable to generate a desired frequency with this method of using an interrupt? Refer to the disassembly of the interrupt service routine. Additionally, temporarily change the optimization level of your compiler to -O1. Are the results any different? Why or why not?

Interrupt halts the current task of the program to increment the sine wave value, resulting in a varying number of lines of codes that may delay the program from generating the desired frequency. The change in optimization level of the compiler to -O1 results in no different frequency as the number of instructions does not effect the timing of the interrupt and thus not change the incrementation of the sine wave value.



ii. Would a method of synchronous polling (i.e., a method with no interrupts) result in the same issue identified in the previous exercise? In other words, would the desired frequency not initially met now be achieved? Alter your program to check your answer, and then take a screenshot of the waveform generated.

A method of synchronous polling result in the same issue identified in the previous exercise. The desired frequency not initially met now be achieved.



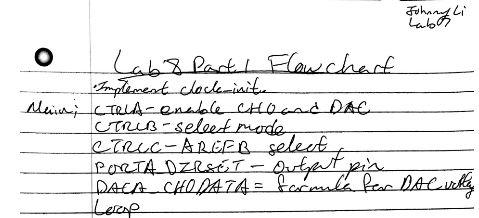
iii. What is the correlation between the amount of data points used to recreate the waveform and the overall quality of the waveform?

Increasing the amount of data points used to recreate the waveform, improved the overall quality of the waveform. This essentially mimics the sampling frequency of the sine wave which would make the sine wave look smoother and more accurate.

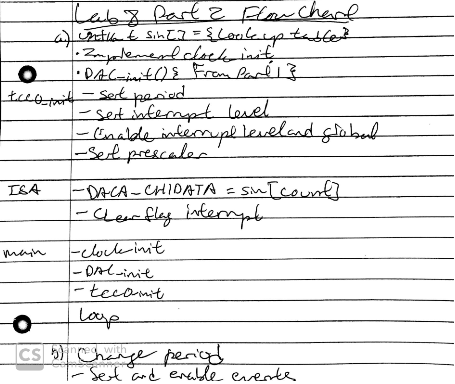
**PSEUDOCODE/FLOWCHARTS**

**SECTION X (1, 2, etc.)**

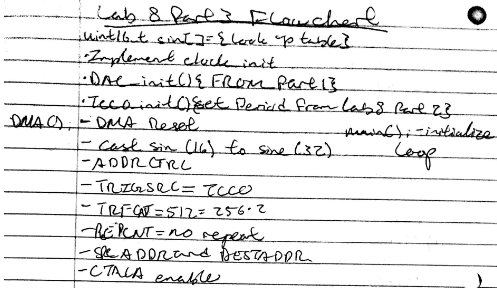
Part 1: INTRODUCTION TO DAC



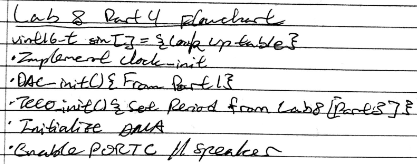
Part 2a,b: GENERATING A WAVEFORM WITH A LOOKUP TABLE



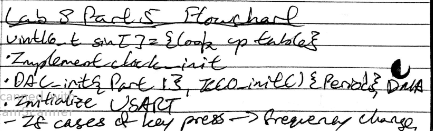
Part 3: INTRODUCTION TO DMA



Part 4: INTRODUCTION TO MUSIC



Part 5: MAKING A MUSICAL INSTRUMENT



**PROGRAM CODE**

**SECTION X (1, 2, etc.)**

Part 1: INTRODUCTION TO DAC

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;Lab 8 Part 1

;Section #: 1823

;Name: Johnny Li

;Class #: 12378

;PI Name: Jared Holley

;Description: INTRODUCTION TO DAC

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INCLUDES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <avr/io.h>

//Change clock.s

extern void clock\_init(void);

int main(void){

//Default 1V

//CH0 enable and DAC enable

DACA\_CTRLA = DAC\_CH0EN\_bm | DAC\_ENABLE\_bm;

//Single mode channel 0

DACA\_CTRLB = DAC\_CHSEL\_SINGLE\_gc;

//AREFB

DACA\_CTRLC = DAC\_REFSEL\_AREFB\_gc;

PORTA\_DIRSET = PIN5\_bm; //Output

DACA\_CH0DATA = (1.1/2.5)\*4095;

while (1) {}

return 0;

}

Part 2a: GENERATING A WAVEFORM WITH A LOOKUP TABLE

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;Lab 8 Part 2a

;Section #: 1823

;Name: Johnny Li

;Class #: 12378

;PI Name: Jared Holley

;Description: GENERATING A WAVEFORM WITH A LOOKUP TABLE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INCLUDES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <avr/io.h>

#include <avr/interrupt.h>

//Change clock.s

extern void clock\_init(void);

#define period ((32000000/256/1) / 1000)

//1000Hz -> (32MHz/256)/1000Hz

//Global Variable

volatile *uint8\_t* count = 0; //Goes through the sin

const *uint16\_t* sin[] = { //Sin wave

0x800,0x832,0x864,0x896,0x8c8,0x8fa,0x92c,0x95e,

0x98f,0x9c0,0x9f1,0xa22,0xa52,0xa82,0xab1,0xae0,

0xb0f,0xb3d,0xb6b,0xb98,0xbc5,0xbf1,0xc1c,0xc47,

0xc71,0xc9a,0xcc3,0xceb,0xd12,0xd39,0xd5f,0xd83,

0xda7,0xdca,0xded,0xe0e,0xe2e,0xe4e,0xe6c,0xe8a,

0xea6,0xec1,0xedc,0xef5,0xf0d,0xf24,0xf3a,0xf4f,

0xf63,0xf76,0xf87,0xf98,0xfa7,0xfb5,0xfc2,0xfcd,

0xfd8,0xfe1,0xfe9,0xff0,0xff5,0xff9,0xffd,0xffe,

0xfff,0xffe,0xffd,0xff9,0xff5,0xff0,0xfe9,0xfe1,

0xfd8,0xfcd,0xfc2,0xfb5,0xfa7,0xf98,0xf87,0xf76,

0xf63,0xf4f,0xf3a,0xf24,0xf0d,0xef5,0xedc,0xec1,

0xea6,0xe8a,0xe6c,0xe4e,0xe2e,0xe0e,0xded,0xdca,

0xda7,0xd83,0xd5f,0xd39,0xd12,0xceb,0xcc3,0xc9a,

0xc71,0xc47,0xc1c,0xbf1,0xbc5,0xb98,0xb6b,0xb3d,

0xb0f,0xae0,0xab1,0xa82,0xa52,0xa22,0x9f1,0x9c0,

0x98f,0x95e,0x92c,0x8fa,0x8c8,0x896,0x864,0x832,

0x800,0x7cd,0x79b,0x769,0x737,0x705,0x6d3,0x6a1,

0x670,0x63f,0x60e,0x5dd,0x5ad,0x57d,0x54e,0x51f,

0x4f0,0x4c2,0x494,0x467,0x43a,0x40e,0x3e3,0x3b8,

0x38e,0x365,0x33c,0x314,0x2ed,0x2c6,0x2a0,0x27c,

0x258,0x235,0x212,0x1f1,0x1d1,0x1b1,0x193,0x175,

0x159,0x13e,0x123,0x10a,0xf2,0xdb,0xc5,0xb0,

0x9c,0x89,0x78,0x67,0x58,0x4a,0x3d,0x32,

0x27,0x1e,0x16,0xf,0xa,0x6,0x2,0x1,

0x0,0x1,0x2,0x6,0xa,0xf,0x16,0x1e,

0x27,0x32,0x3d,0x4a,0x58,0x67,0x78,0x89,

0x9c,0xb0,0xc5,0xdb,0xf2,0x10a,0x123,0x13e,

0x159,0x175,0x193,0x1b1,0x1d1,0x1f1,0x212,0x235,

0x258,0x27c,0x2a0,0x2c6,0x2ed,0x314,0x33c,0x365,

0x38e,0x3b8,0x3e3,0x40e,0x43a,0x467,0x494,0x4c2,

0x4f0,0x51f,0x54e,0x57d,0x5ad,0x5dd,0x60e,0x63f,

0x670,0x6a1,0x6d3,0x705,0x737,0x769,0x79b,0x7cd

};

int main(void){

//Initialization

clock\_init();

DAC\_init();

tcc0\_init();

//Polling test

while (1) {

/\*if((TCC0.INTFLAGS & 0x01)==0x01){

//Sin value in data

DACA\_CH1DATA = sin[count++];

//Clear flag

TCC0.INTFLAGS = 0x01;

}\*/

}

return 0;

}

Part 2b: GENERATING A WAVEFORM WITH A LOOKUP TABLE

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;Lab 8 Part 2

;Section #: 1823

;Name: Johnny Li

;Class #: 12378

;PI Name: Jared Holley

;Description: GENERATING A WAVEFORM WITH A LOOKUP TABLE (b)

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INCLUDES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <avr/io.h>

#include <avr/interrupt.h>

//Change clock.s

extern void clock\_init(void);

#define period ((32000000/256/1) / 1760)

//1000Hz -> (32MHz/256/1)/1000Hz

//Global Variable

volatile *uint8\_t* count = 0; //Goes through the sin

const *uint16\_t* sin[] = { //Sin wave

0x800,0x832,0x864,0x896,0x8c8,0x8fa,0x92c,0x95e,

0x98f,0x9c0,0x9f1,0xa22,0xa52,0xa82,0xab1,0xae0,

0xb0f,0xb3d,0xb6b,0xb98,0xbc5,0xbf1,0xc1c,0xc47,

0xc71,0xc9a,0xcc3,0xceb,0xd12,0xd39,0xd5f,0xd83,

0xda7,0xdca,0xded,0xe0e,0xe2e,0xe4e,0xe6c,0xe8a,

0xea6,0xec1,0xedc,0xef5,0xf0d,0xf24,0xf3a,0xf4f,

0xf63,0xf76,0xf87,0xf98,0xfa7,0xfb5,0xfc2,0xfcd,

0xfd8,0xfe1,0xfe9,0xff0,0xff5,0xff9,0xffd,0xffe,

0xfff,0xffe,0xffd,0xff9,0xff5,0xff0,0xfe9,0xfe1,

0xfd8,0xfcd,0xfc2,0xfb5,0xfa7,0xf98,0xf87,0xf76,

0xf63,0xf4f,0xf3a,0xf24,0xf0d,0xef5,0xedc,0xec1,

0xea6,0xe8a,0xe6c,0xe4e,0xe2e,0xe0e,0xded,0xdca,

0xda7,0xd83,0xd5f,0xd39,0xd12,0xceb,0xcc3,0xc9a,

0xc71,0xc47,0xc1c,0xbf1,0xbc5,0xb98,0xb6b,0xb3d,

0xb0f,0xae0,0xab1,0xa82,0xa52,0xa22,0x9f1,0x9c0,

0x98f,0x95e,0x92c,0x8fa,0x8c8,0x896,0x864,0x832,

0x800,0x7cd,0x79b,0x769,0x737,0x705,0x6d3,0x6a1,

0x670,0x63f,0x60e,0x5dd,0x5ad,0x57d,0x54e,0x51f,

0x4f0,0x4c2,0x494,0x467,0x43a,0x40e,0x3e3,0x3b8,

0x38e,0x365,0x33c,0x314,0x2ed,0x2c6,0x2a0,0x27c,

0x258,0x235,0x212,0x1f1,0x1d1,0x1b1,0x193,0x175,

0x159,0x13e,0x123,0x10a,0xf2,0xdb,0xc5,0xb0,

0x9c,0x89,0x78,0x67,0x58,0x4a,0x3d,0x32,

0x27,0x1e,0x16,0xf,0xa,0x6,0x2,0x1,

0x0,0x1,0x2,0x6,0xa,0xf,0x16,0x1e,

0x27,0x32,0x3d,0x4a,0x58,0x67,0x78,0x89,

0x9c,0xb0,0xc5,0xdb,0xf2,0x10a,0x123,0x13e,

0x159,0x175,0x193,0x1b1,0x1d1,0x1f1,0x212,0x235,

0x258,0x27c,0x2a0,0x2c6,0x2ed,0x314,0x33c,0x365,

0x38e,0x3b8,0x3e3,0x40e,0x43a,0x467,0x494,0x4c2,

0x4f0,0x51f,0x54e,0x57d,0x5ad,0x5dd,0x60e,0x63f,

0x670,0x6a1,0x6d3,0x705,0x737,0x769,0x79b,0x7cd

};

int main(void){

//Initialization

clock\_init();

DAC\_init();

tcc0\_init();

while (1) {

//Stall till flag occurs

if((TCC0.INTFLAGS & 0x01)==0x01){

//Sin value in data

DACA\_CH1DATA = sin[count++];

//Clear flag

TCC0.INTFLAGS = 0x01;

}

}

return 0;

}

Part 3: INTRODUCTION TO DMA

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;Lab 8 Part 3

;Section #: 1823

;Name: Johnny Li

;Class #: 12378

;PI Name: Jared Holley

;Description: INTRODUCTION TO DMA

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INCLUDES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <avr/io.h>

#include <avr/interrupt.h>

//Change clock.s

extern void clock\_init(void);

#define period ((32000000/256/1) / 1760)

//1000Hz -> (32MHz/256)/1000Hz | ((32000000/256) / 1760)

//Global Variable

const *uint16\_t* sin[] = { //Sin wave

0x800,0x832,0x864,0x896,0x8c8,0x8fa,0x92c,0x95e,

0x98f,0x9c0,0x9f1,0xa22,0xa52,0xa82,0xab1,0xae0,

0xb0f,0xb3d,0xb6b,0xb98,0xbc5,0xbf1,0xc1c,0xc47,

0xc71,0xc9a,0xcc3,0xceb,0xd12,0xd39,0xd5f,0xd83,

0xda7,0xdca,0xded,0xe0e,0xe2e,0xe4e,0xe6c,0xe8a,

0xea6,0xec1,0xedc,0xef5,0xf0d,0xf24,0xf3a,0xf4f,

0xf63,0xf76,0xf87,0xf98,0xfa7,0xfb5,0xfc2,0xfcd,

0xfd8,0xfe1,0xfe9,0xff0,0xff5,0xff9,0xffd,0xffe,

0xfff,0xffe,0xffd,0xff9,0xff5,0xff0,0xfe9,0xfe1,

0xfd8,0xfcd,0xfc2,0xfb5,0xfa7,0xf98,0xf87,0xf76,

0xf63,0xf4f,0xf3a,0xf24,0xf0d,0xef5,0xedc,0xec1,

0xea6,0xe8a,0xe6c,0xe4e,0xe2e,0xe0e,0xded,0xdca,

0xda7,0xd83,0xd5f,0xd39,0xd12,0xceb,0xcc3,0xc9a,

0xc71,0xc47,0xc1c,0xbf1,0xbc5,0xb98,0xb6b,0xb3d,

0xb0f,0xae0,0xab1,0xa82,0xa52,0xa22,0x9f1,0x9c0,

0x98f,0x95e,0x92c,0x8fa,0x8c8,0x896,0x864,0x832,

0x800,0x7cd,0x79b,0x769,0x737,0x705,0x6d3,0x6a1,

0x670,0x63f,0x60e,0x5dd,0x5ad,0x57d,0x54e,0x51f,

0x4f0,0x4c2,0x494,0x467,0x43a,0x40e,0x3e3,0x3b8,

0x38e,0x365,0x33c,0x314,0x2ed,0x2c6,0x2a0,0x27c,

0x258,0x235,0x212,0x1f1,0x1d1,0x1b1,0x193,0x175,

0x159,0x13e,0x123,0x10a,0xf2,0xdb,0xc5,0xb0,

0x9c,0x89,0x78,0x67,0x58,0x4a,0x3d,0x32,

0x27,0x1e,0x16,0xf,0xa,0x6,0x2,0x1,

0x0,0x1,0x2,0x6,0xa,0xf,0x16,0x1e,

0x27,0x32,0x3d,0x4a,0x58,0x67,0x78,0x89,

0x9c,0xb0,0xc5,0xdb,0xf2,0x10a,0x123,0x13e,

0x159,0x175,0x193,0x1b1,0x1d1,0x1f1,0x212,0x235,

0x258,0x27c,0x2a0,0x2c6,0x2ed,0x314,0x33c,0x365,

0x38e,0x3b8,0x3e3,0x40e,0x43a,0x467,0x494,0x4c2,

0x4f0,0x51f,0x54e,0x57d,0x5ad,0x5dd,0x60e,0x63f,

0x670,0x6a1,0x6d3,0x705,0x737,0x769,0x79b,0x7cd

};

int main(void){

//Initialization

clock\_init();

DAC\_init();

tcc0\_init();

DMA\_init();

//Enabled interrupts

PMIC\_CTRL = PMIC\_LOLVLEN\_bm; //Low level interrupts

sei(); //global interrupt

while (1) {}

return 0;

}

Part 4: INTRODUCTION TO MUSIC

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;Lab 8 Part 4

;Section #: 1823

;Name: Johnny Li

;Class #: 12378

;PI Name: Jared Holley

;Description: INTRODUCTION TO MUSIC

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INCLUDES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <avr/io.h>

#include <avr/interrupt.h>

//Change clock.s

extern void clock\_init(void);

#define period ((32000000/256/1) / 1760)

//1000Hz -> (32MHz/256)/1000Hz | ((32000000/256) / 1760)

//((int)((1000000)/(x))+10)

//Global Variable

const *uint16\_t* sin[] = { //Sin wave

0x800,0x832,0x864,0x896,0x8c8,0x8fa,0x92c,0x95e,

0x98f,0x9c0,0x9f1,0xa22,0xa52,0xa82,0xab1,0xae0,

0xb0f,0xb3d,0xb6b,0xb98,0xbc5,0xbf1,0xc1c,0xc47,

0xc71,0xc9a,0xcc3,0xceb,0xd12,0xd39,0xd5f,0xd83,

0xda7,0xdca,0xded,0xe0e,0xe2e,0xe4e,0xe6c,0xe8a,

0xea6,0xec1,0xedc,0xef5,0xf0d,0xf24,0xf3a,0xf4f,

0xf63,0xf76,0xf87,0xf98,0xfa7,0xfb5,0xfc2,0xfcd,

0xfd8,0xfe1,0xfe9,0xff0,0xff5,0xff9,0xffd,0xffe,

0xfff,0xffe,0xffd,0xff9,0xff5,0xff0,0xfe9,0xfe1,

0xfd8,0xfcd,0xfc2,0xfb5,0xfa7,0xf98,0xf87,0xf76,

0xf63,0xf4f,0xf3a,0xf24,0xf0d,0xef5,0xedc,0xec1,

0xea6,0xe8a,0xe6c,0xe4e,0xe2e,0xe0e,0xded,0xdca,

0xda7,0xd83,0xd5f,0xd39,0xd12,0xceb,0xcc3,0xc9a,

0xc71,0xc47,0xc1c,0xbf1,0xbc5,0xb98,0xb6b,0xb3d,

0xb0f,0xae0,0xab1,0xa82,0xa52,0xa22,0x9f1,0x9c0,

0x98f,0x95e,0x92c,0x8fa,0x8c8,0x896,0x864,0x832,

0x800,0x7cd,0x79b,0x769,0x737,0x705,0x6d3,0x6a1,

0x670,0x63f,0x60e,0x5dd,0x5ad,0x57d,0x54e,0x51f,

0x4f0,0x4c2,0x494,0x467,0x43a,0x40e,0x3e3,0x3b8,

0x38e,0x365,0x33c,0x314,0x2ed,0x2c6,0x2a0,0x27c,

0x258,0x235,0x212,0x1f1,0x1d1,0x1b1,0x193,0x175,

0x159,0x13e,0x123,0x10a,0xf2,0xdb,0xc5,0xb0,

0x9c,0x89,0x78,0x67,0x58,0x4a,0x3d,0x32,

0x27,0x1e,0x16,0xf,0xa,0x6,0x2,0x1,

0x0,0x1,0x2,0x6,0xa,0xf,0x16,0x1e,

0x27,0x32,0x3d,0x4a,0x58,0x67,0x78,0x89,

0x9c,0xb0,0xc5,0xdb,0xf2,0x10a,0x123,0x13e,

0x159,0x175,0x193,0x1b1,0x1d1,0x1f1,0x212,0x235,

0x258,0x27c,0x2a0,0x2c6,0x2ed,0x314,0x33c,0x365,

0x38e,0x3b8,0x3e3,0x40e,0x43a,0x467,0x494,0x4c2,

0x4f0,0x51f,0x54e,0x57d,0x5ad,0x5dd,0x60e,0x63f,

0x670,0x6a1,0x6d3,0x705,0x737,0x769,0x79b,0x7cd

};

int main(void){

//Initialization

clock\_init();

DAC\_init();

tcc0\_init();

DMA\_init();

//Enabled interrupts

//PMIC\_CTRL = PMIC\_LOLVLEN\_bm; //Low level interrupts

//sei(); //global interrupt

PORTC\_DIRSET |= PIN7\_bm; //enable\_speaker

PORTC\_OUTSET = PIN7\_bm;

while (1){}

return 0;

}

Part 5: MAKING A MUSICAL INSTRUMENT

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;Lab 8 Part 5

;Section #: 1823

;Name: Johnny Li

;Class #: 12378

;PI Name: Jared Holley

;Description: MAKING A MUSICAL INSTRUMENT

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INCLUDES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <avr/io.h>

#include <avr/interrupt.h>

#include "USART.h"

//Change clock.s

extern void clock\_init(void);

#define period ((32000000/256/1) / 1760)

//1000Hz -> (32MHz/256)/1000Hz | ((32000000/256) / 1760)

//((int)((1000000)/(x))+10)

//Global Variable

const *uint16\_t* sin[] = { //Sin wave

0x800,0x832,0x864,0x896,0x8c8,0x8fa,0x92c,0x95e,

0x98f,0x9c0,0x9f1,0xa22,0xa52,0xa82,0xab1,0xae0,

0xb0f,0xb3d,0xb6b,0xb98,0xbc5,0xbf1,0xc1c,0xc47,

0xc71,0xc9a,0xcc3,0xceb,0xd12,0xd39,0xd5f,0xd83,

0xda7,0xdca,0xded,0xe0e,0xe2e,0xe4e,0xe6c,0xe8a,

0xea6,0xec1,0xedc,0xef5,0xf0d,0xf24,0xf3a,0xf4f,

0xf63,0xf76,0xf87,0xf98,0xfa7,0xfb5,0xfc2,0xfcd,

0xfd8,0xfe1,0xfe9,0xff0,0xff5,0xff9,0xffd,0xffe,

0xfff,0xffe,0xffd,0xff9,0xff5,0xff0,0xfe9,0xfe1,

0xfd8,0xfcd,0xfc2,0xfb5,0xfa7,0xf98,0xf87,0xf76,

0xf63,0xf4f,0xf3a,0xf24,0xf0d,0xef5,0xedc,0xec1,

0xea6,0xe8a,0xe6c,0xe4e,0xe2e,0xe0e,0xded,0xdca,

0xda7,0xd83,0xd5f,0xd39,0xd12,0xceb,0xcc3,0xc9a,

0xc71,0xc47,0xc1c,0xbf1,0xbc5,0xb98,0xb6b,0xb3d,

0xb0f,0xae0,0xab1,0xa82,0xa52,0xa22,0x9f1,0x9c0,

0x98f,0x95e,0x92c,0x8fa,0x8c8,0x896,0x864,0x832,

0x800,0x7cd,0x79b,0x769,0x737,0x705,0x6d3,0x6a1,

0x670,0x63f,0x60e,0x5dd,0x5ad,0x57d,0x54e,0x51f,

0x4f0,0x4c2,0x494,0x467,0x43a,0x40e,0x3e3,0x3b8,

0x38e,0x365,0x33c,0x314,0x2ed,0x2c6,0x2a0,0x27c,

0x258,0x235,0x212,0x1f1,0x1d1,0x1b1,0x193,0x175,

0x159,0x13e,0x123,0x10a,0xf2,0xdb,0xc5,0xb0,

0x9c,0x89,0x78,0x67,0x58,0x4a,0x3d,0x32,

0x27,0x1e,0x16,0xf,0xa,0x6,0x2,0x1,

0x0,0x1,0x2,0x6,0xa,0xf,0x16,0x1e,

0x27,0x32,0x3d,0x4a,0x58,0x67,0x78,0x89,

0x9c,0xb0,0xc5,0xdb,0xf2,0x10a,0x123,0x13e,

0x159,0x175,0x193,0x1b1,0x1d1,0x1f1,0x212,0x235,

0x258,0x27c,0x2a0,0x2c6,0x2ed,0x314,0x33c,0x365,

0x38e,0x3b8,0x3e3,0x40e,0x43a,0x467,0x494,0x4c2,

0x4f0,0x51f,0x54e,0x57d,0x5ad,0x5dd,0x60e,0x63f,

0x670,0x6a1,0x6d3,0x705,0x737,0x769,0x79b,0x7cd

};

volatile int rflag = 0; //Receiver flag

volatile char c;

int main(void){

//Initialization

clock\_init();

DAC\_init();

tcc0\_init(period);

usartd0\_init();

DMA\_init();

//Enabled interrupts

PMIC\_CTRL = PMIC\_LOLVLEN\_bm; //Low level interrupts

sei(); //global interrupt

PORTC\_DIRSET |= PIN7\_bm; //enable\_speaker

PORTC\_OUTSET = PIN7\_bm;

while (1){

//Get input

if (rflag){

//Reset

rflag = 0;

//Output Note

switch (c)

{

case 'W': tcc0\_init((int) ((32000000/256/1) / 1046.50)); //C6

break;

case '3': tcc0\_init((int) ((32000000/256/1) / 1108.73)); //C#6/Db6

break;

case 'E': tcc0\_init((int) ((32000000/256/1) / 1174.66)); //D6

break;

case '4': tcc0\_init((int) ((32000000/256/1) / 1244.51)); //D#6/Eb6

break;

case 'R': tcc0\_init((int) ((32000000/256/1) / 1318.51)); //E6

break;

case 'T': tcc0\_init((int) ((32000000/256/1) / 1396.91)); //F6

break;

case '6': tcc0\_init((int) ((32000000/256/1) / 1479.98)); //F#6/Gb6

break;

case 'Y': tcc0\_init((int) ((32000000/256/1) / 1567.98)); //G6

break;

case '7': tcc0\_init((int) ((32000000/256/1) / 1661.22)); //G#6/Ab6

break;

case 'U': tcc0\_init((int) ((32000000/256/1) / 1760.00)); //A6

break;

case '8': tcc0\_init((int) ((32000000/256/1) / 1864.66)); //A$6/Bb6

break;

case 'I': tcc0\_init((int) ((32000000/256/1) / 1975.53)); //B6

break;

}

}

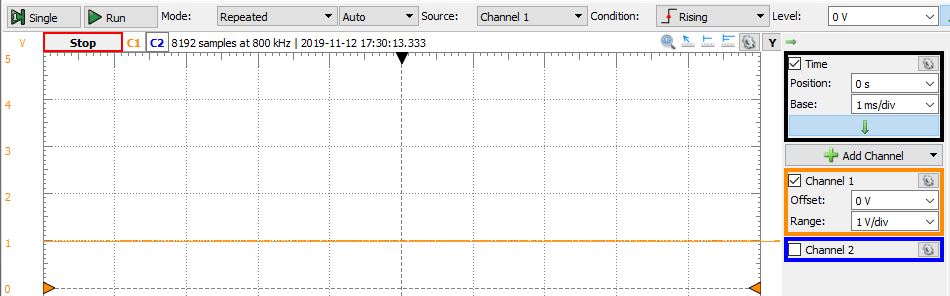
}

return 0;

}

**APPENDIX**

Part 1: INTRODUCTION TO DAC



Screenshot 1: Waveform with a constant voltage of 1 V, using a DAC module.

Part 2a: GENERATING A WAVEFORM WITH A LOOKUP TABLE

Support Code: Lab8\_2a.c

//Initialize DAC

void DAC\_init(void){

//Default 1V

//CH1 enable and DAC enable

DACA\_CTRLA = DAC\_CH1EN\_bm | DAC\_ENABLE\_bm;

//DUAL mode

DACA\_CTRLB = DAC\_CHSEL\_DUAL\_gc;

//AREFB

DACA\_CTRLC = DAC\_REFSEL\_AREFB\_gc;

PORTA\_DIRSET = PIN3\_bm; //Output

}

//Initialize TCC0 timer/counter

void tcc0\_init(void){

TCC0\_PERL = (*uint8\_t*) period; //Period

TCC0\_PERH = (*uint8\_t*) (period>>8);

TCC0.INTCTRLA = TC\_OVFINTLVL\_LO\_gc; //Low level interrupt

//Enabled interrupts

PMIC\_CTRL = PMIC\_LOLVLEN\_bm; //Low level interrupts

sei(); //global interrupt

TCC0\_CTRLA = 0x01; //Prescaler = 1

//0x02;

}

//Overflow

ISR(TCC0\_OVF\_vect){

//Sin value in data

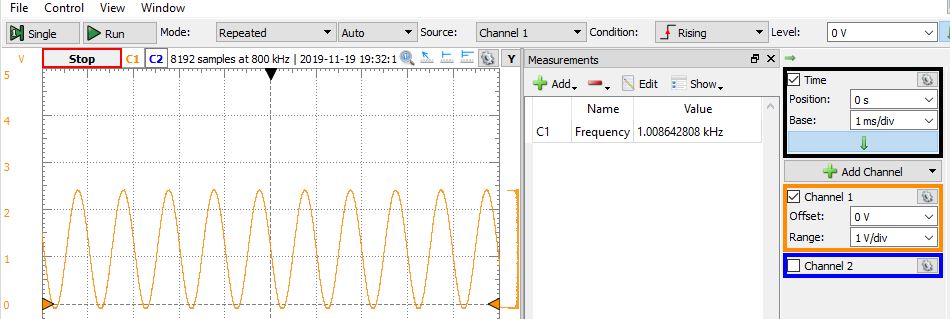
DACA\_CH1DATA = sin[count++];

//Clear flag

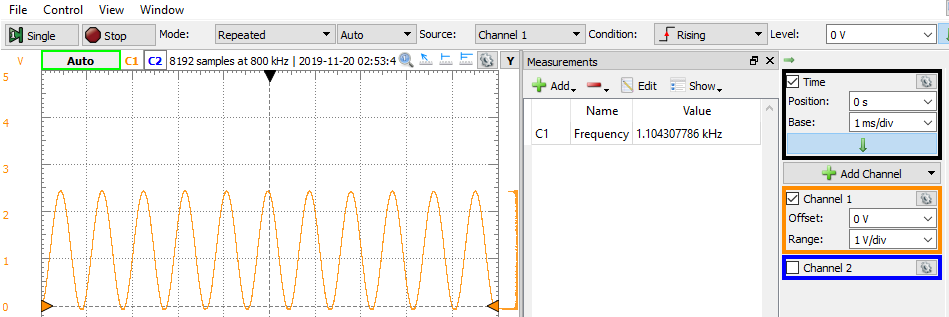
TCC0.INTFLAGS = 0x01;

return;

}



Screenshot 2: Generate a sine wave of at least 256 data points, with voltages ranging between 0V and AREFB, and with a frequency of 1000 Hz.



Screenshot 3: Generate a sine waveform with a frequency of 1140 Hz, while still meeting all other requirements listed above. In increments of 10 Hz.

Part 2b: GENERATING A WAVEFORM WITH A LOOKUP TABLE

Support Code: Lab8\_2b.c

//Initialize DAC

void DAC\_init(void){

//Default 1V

//CH1 enable and DAC enable

DACA\_CTRLA = DAC\_CH1EN\_bm | DAC\_ENABLE\_bm;

//DUAL mode

DACA\_CTRLB = DAC\_CHSEL\_DUAL\_gc;

//AREFB

DACA\_CTRLC = DAC\_REFSEL\_AREFB\_gc;

//DACB conversion start when Event Channel 0 is triggered

DACA\_EVCTRL = DAC\_EVSEL\_1\_gc;

PORTA\_DIRSET = PIN3\_bm; //Output

}

//Initialize TCC0 timer/counter

void tcc0\_init(void){

TCC0\_PERL = (*uint8\_t*) period; //Period

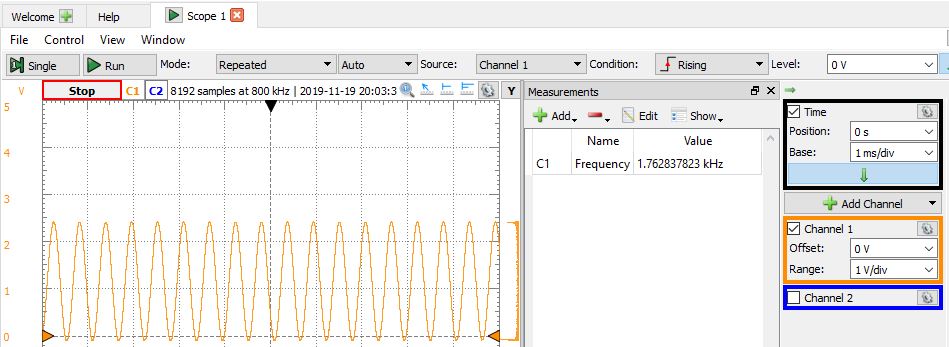
TCC0\_PERH = (*uint8\_t*) (period>>8);

//Trigger an event on Event Channel 0

EVSYS\_CH1MUX = EVSYS\_CHMUX\_TCC0\_OVF\_gc;

TCC0\_CTRLA = 0x01; //Prescaler = 1

}



Screenshot 4: Generate a sine waveform with a frequency of 1760 Hz, while still meeting all other requirements listed above using events.

Part 3: INTRODUCTION TO DMA

Support Code: Lab8\_3.c

//Initialize DAC

void DAC\_init(void){

//Default 1V

//CH1 enable and DAC enable

DACA\_CTRLA = DAC\_CH1EN\_bm | DAC\_ENABLE\_bm;

//AREFB

DACA\_CTRLC = DAC\_REFSEL\_AREFB\_gc;

PORTA\_DIRSET = PIN3\_bm; //Output

}

//Initialize TCC0 timer/counter

void tcc0\_init(void){

TCC0.CNT = 0; //Initial value 0

TCC0\_PERL = (*uint8\_t*) period; //Period

TCC0\_PERH = (*uint8\_t*) (period>>8);

TCC0.INTCTRLA = TC\_OVFINTLVL\_LO\_gc; //Low level interrupt

TCC0\_CTRLA = 0x01; //Prescaler = 1

}

ISR(TCC0\_OVF\_vect){

//Preserve Status Reg

*uint8\_t* temp = CPU\_SREG;

//Clear interrupt flags

TCC0.INTFLAGS = 0x01;

//Restore Status Reg

CPU\_SREG = temp;

//Return from ISR

return;

}

//Initialize DMA

void DMA\_init(){

*uint32\_t* sine = (*uint32\_t*)sin;

//Reset DMA

DMA.CTRL |= DMA\_RESET\_bm;

//Increment the source after transfer

//Increment the destination after received

DMA.CH0.ADDRCTRL = DMA\_CH\_SRCRELOAD\_BLOCK\_gc|DMA\_CH\_SRCDIR\_INC\_gc|DMA\_CH\_DESTRELOAD\_BURST\_gc|DMA\_CH\_DESTDIR\_INC\_gc;

//Transfer data when TCE0 overflows

DMA.CH0.TRIGSRC = DMA\_CH\_TRIGSRC\_TCC0\_OVF\_gc;

//Byte transfers in a block transfer

DMA.CH0.TRFCNT =512; //512=256(8bit)\*2(16bit)

//Unlimited repeats

DMA.CH0.REPCNT = 0x00;

//Starting address of the source

DMA.CH0.SRCADDR0 = (*uint8\_t*)(sine>>0);

DMA.CH0.SRCADDR1 = (*uint8\_t*)(sine>>8);

DMA.CH0.SRCADDR2 = (*uint8\_t*)(sine>>16);

*uint8\_t*\* dac\_ptr = &DACA.CH1DATA;

*uint32\_t* dac\_address = (*uint32\_t*)dac\_ptr;

//Store data to the DAC

DMA.CH0.DESTADDR0 = (*uint8\_t*) (dac\_address>>0);

DMA.CH0.DESTADDR1 = (*uint8\_t*) (dac\_address>>8);

DMA.CH0.DESTADDR2 = (*uint8\_t*) (dac\_address>>16);

//Enable CHO

//Unlimited repeat, Data is sent in burst where each burst is 2 bytes long

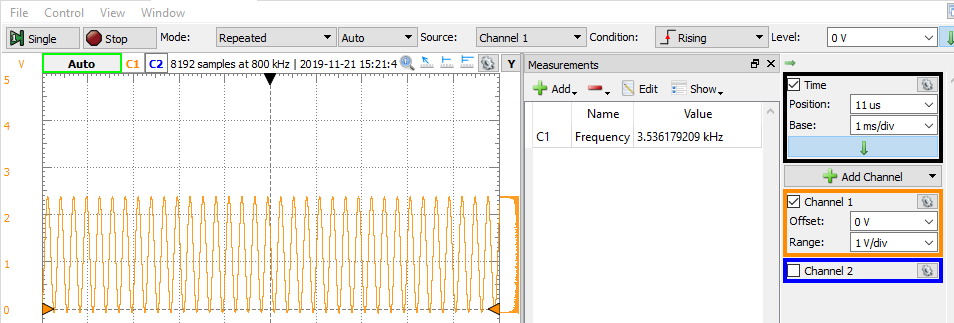
DMA.CH0.CTRLA = DMA\_CH\_REPEAT\_bm|DMA\_CH\_SINGLE\_bm|DMA\_CH\_BURSTLEN\_2BYTE\_gc;

DMA.CH0.CTRLA |= DMA\_CH\_ENABLE\_bm;

//Enable DMA

DMA.CTRL |= DMA\_ENABLE\_bm;

}



Screenshot 5: Generate a sine waveform with a frequency of 1760 Hz\*2, unable to reach 1760 Hz, while still meeting all other requirements listed above using events.

Part 4: INTRODUCTION TO MUSIC

Support Code: Lab8\_4.c

//Initialize DAC

void DAC\_init(void){

PORTA\_DIRSET = PIN3\_bm; //Output

//Default 1V

//CH1 enable and DAC enable

DACA\_CTRLA = DAC\_CH1EN\_bm | DAC\_ENABLE\_bm;

DACA\_CTRLB = 0x20;

//AREFB

DACA\_CTRLC = DAC\_REFSEL\_AREFB\_gc;

}

//Initialize TCC0 timer/counter

void tcc0\_init(void){

TCC0.CNT = 0; //Initial value 0

TCC0\_PERL = (*uint8\_t*) period; //Period

TCC0\_PERH = (*uint8\_t*) (period>>8);

//TCC0.INTCTRLA = TC\_OVFINTLVL\_LO\_gc; //Low level interrupt

EVSYS\_CH1MUX = EVSYS\_CHMUX\_TCC0\_OVF\_gc;

TCC0\_CTRLA = 0x01; //Prescaler = 1

}

//Initialize DMA

void DMA\_init(){

*uint32\_t* sine = (*uint32\_t*)sin;

//Reset DMA

DMA.CTRL |= DMA\_RESET\_bm;

//Increment the source after transfer

//Increment the destination after received

DMA.CH0.ADDRCTRL = DMA\_CH\_SRCRELOAD\_BLOCK\_gc|DMA\_CH\_SRCDIR\_INC\_gc|DMA\_CH\_DESTRELOAD\_BURST\_gc|DMA\_CH\_DESTDIR\_INC\_gc;

//Transfer data when TCE0 overflows

DMA.CH0.TRIGSRC = DMA\_CH\_TRIGSRC\_EVSYS\_CH1\_gc;

//Byte transfers in a block transfer

DMA.CH0.TRFCNT =512; //512=256(8bit)\*2(16bit)

//Unlimited repeats

DMA.CH0.REPCNT = 0x00;

//Starting address of the source

DMA.CH0.SRCADDR0 = (*uint8\_t*)(sine>>0);

DMA.CH0.SRCADDR1 = (*uint8\_t*)(sine>>8);

DMA.CH0.SRCADDR2 = (*uint8\_t*)(sine>>16);

*uint8\_t*\* dac\_ptr = &DACA.CH1DATA;

*uint32\_t* dac\_address = (*uint32\_t*)dac\_ptr;

//Store data to the DAC

DMA.CH0.DESTADDR0 = (*uint8\_t*) (dac\_address>>0);

DMA.CH0.DESTADDR1 = (*uint8\_t*) (dac\_address>>8);

DMA.CH0.DESTADDR2 = (*uint8\_t*) (dac\_address>>16);

//Enable CHO

//Unlimited repeat, Data is sent in burst where each burst is 2 bytes long

DMA.CH0.CTRLA = DMA\_CH\_REPEAT\_bm|DMA\_CH\_SINGLE\_bm|DMA\_CH\_BURSTLEN\_2BYTE\_gc;

DMA.CH0.CTRLA |= DMA\_CH\_ENABLE\_bm;

//Enable DMA

DMA.CTRL |= DMA\_ENABLE\_bm;

}

Part 5: MAKING A MUSICAL INSTRUMENT

Support Code: Lab8\_5.c

//Initialize DAC

void DAC\_init(void){

PORTA\_DIRSET = PIN3\_bm; //Output

//Default 1V

//CH1 enable and DAC enable

DACA\_CTRLA = DAC\_CH0EN\_bm | DAC\_ENABLE\_bm;

//AREFB

DACA\_CTRLC = DAC\_REFSEL\_AREFB\_gc;

}

//Initialize TCC0 timer/counter

void tcc0\_init(int p){

TCC0.CNT = 0; //Initial value 0

TCC0\_PERL = (*uint8\_t*) p; //Period

TCC0\_PERH = (*uint8\_t*) (p>>8);

TCC0.INTCTRLA = TC\_OVFINTLVL\_LO\_gc; //Low level interrupt

TCC0\_CTRLA = 0x01; //Prescaler = 1

}

ISR(TCC0\_OVF\_vect){

//Preserve Status Reg

*uint8\_t* temp = CPU\_SREG;

//Clear interrupt flags

TCC0.INTFLAGS = 0x01;

//Restore Status Reg

CPU\_SREG = temp;

//Return from ISR

return;

}

//Initialize DMA

void DMA\_init(){

*uint32\_t* sine = (*uint32\_t*)sin;

//Reset DMA

DMA.CTRL |= DMA\_RESET\_bm;

//Increment the source after transfer

//Increment the destination after received

DMA.CH0.ADDRCTRL = DMA\_CH\_SRCRELOAD\_BLOCK\_gc|DMA\_CH\_SRCDIR\_INC\_gc|DMA\_CH\_DESTRELOAD\_BURST\_gc|DMA\_CH\_DESTDIR\_INC\_gc;

//Transfer data when TCE0 overflows

DMA.CH0.TRIGSRC = DMA\_CH\_TRIGSRC\_TCC0\_OVF\_gc;

//Byte transfers in a block transfer

DMA.CH0.TRFCNT =512; //512=256(8bit)\*2(16bit)

//Unlimited repeats

DMA.CH0.REPCNT = 0x00;

//Starting address of the source

DMA.CH0.SRCADDR0 = (*uint8\_t*)(sine>>0);

DMA.CH0.SRCADDR1 = (*uint8\_t*)(sine>>8);

DMA.CH0.SRCADDR2 = (*uint8\_t*)(sine>>16);

*uint8\_t*\* dac\_ptr = &DACA.CH0DATA;

*uint32\_t* dac\_address = (*uint32\_t*)dac\_ptr;

//Store data to the DAC

DMA.CH0.DESTADDR0 = (*uint8\_t*) (dac\_address>>0);

DMA.CH0.DESTADDR1 = (*uint8\_t*) (dac\_address>>8);

DMA.CH0.DESTADDR2 = (*uint8\_t*) (dac\_address>>16);

//Enable CHO

//Unlimited repeat, Data is sent in burst where each burst is 2 bytes long

DMA.CH0.CTRLA = DMA\_CH\_REPEAT\_bm|DMA\_CH\_SINGLE\_bm|DMA\_CH\_BURSTLEN\_2BYTE\_gc;

DMA.CH0.CTRLA |= DMA\_CH\_ENABLE\_bm;

//Enable DMA

DMA.CTRL |= DMA\_ENABLE\_bm;

}

//Receiver Handler

ISR (USARTD0\_RXC\_vect){

//Get input

c = USARTD0.DATA;

//Set receiver flag

rflag = 1;

}