Lab 7 Summary

Schuster, Mark

Section 1540

11/22/2017

B) Pre-Lab Answers:

1. The greater the number of discrete points or “samples” the greater the quality or “integrity” of the signal.
2. There are 4 DMA channels.
3. 26 different base trigger values.

C) Problems encountered:

I was struggling to get the DMA to work for several hours. I read through the entire manual entry and the additional documentation provided by Atmel. After much trial and error, I realized that the DMA channel must be enabled before the rest of the settings can be set.

D) Future Work/Applications:

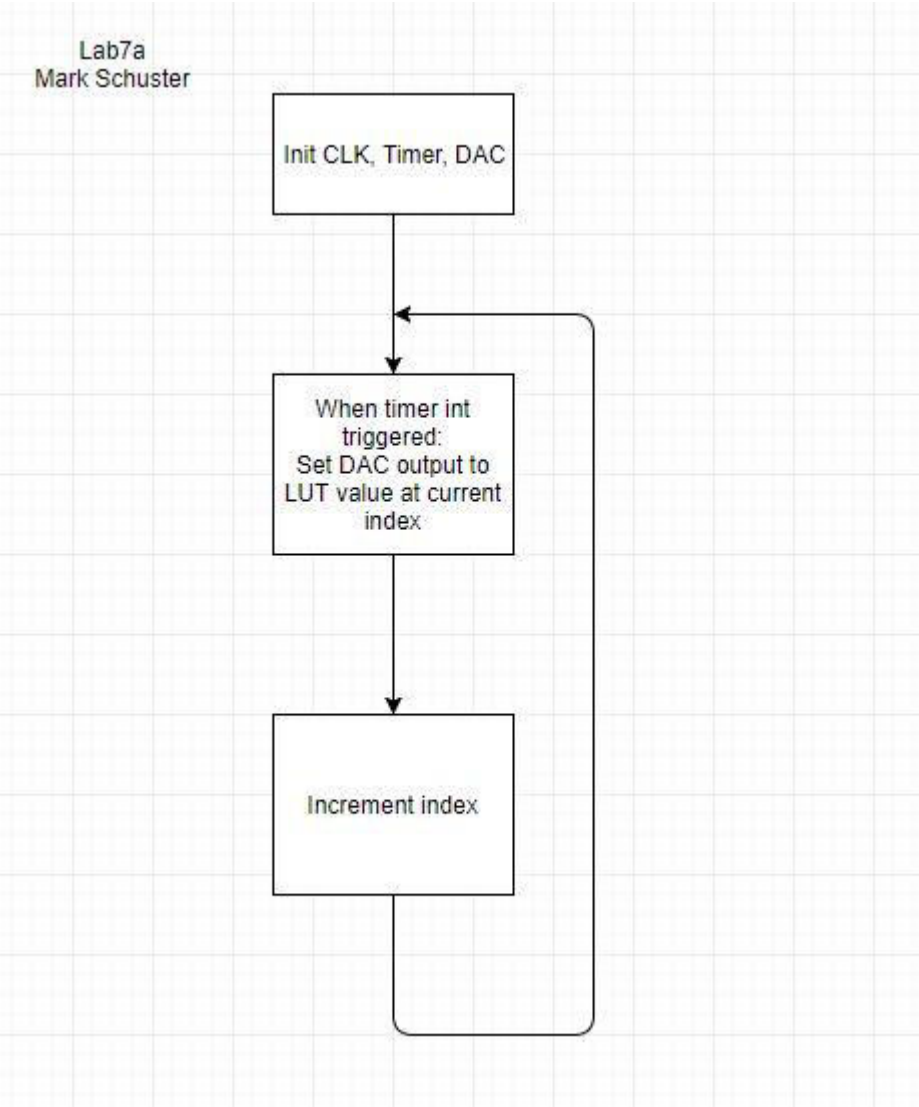
I enjoyed working with the DMA module and its use has parallels some topics in my operating systems class.

E) Schematics:

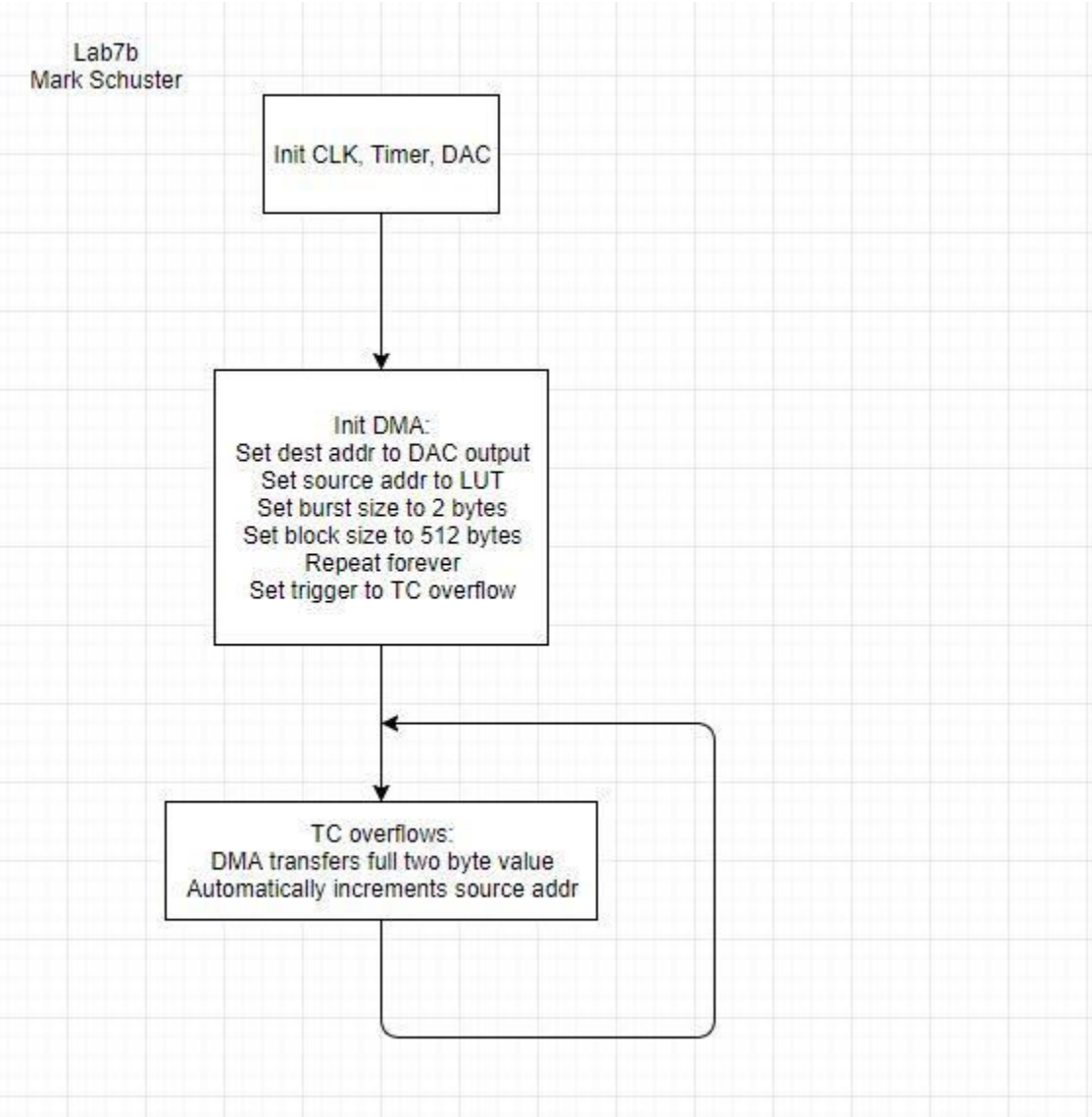
Not applicable for this lab.

F) Pseudocode/Flowcharts:

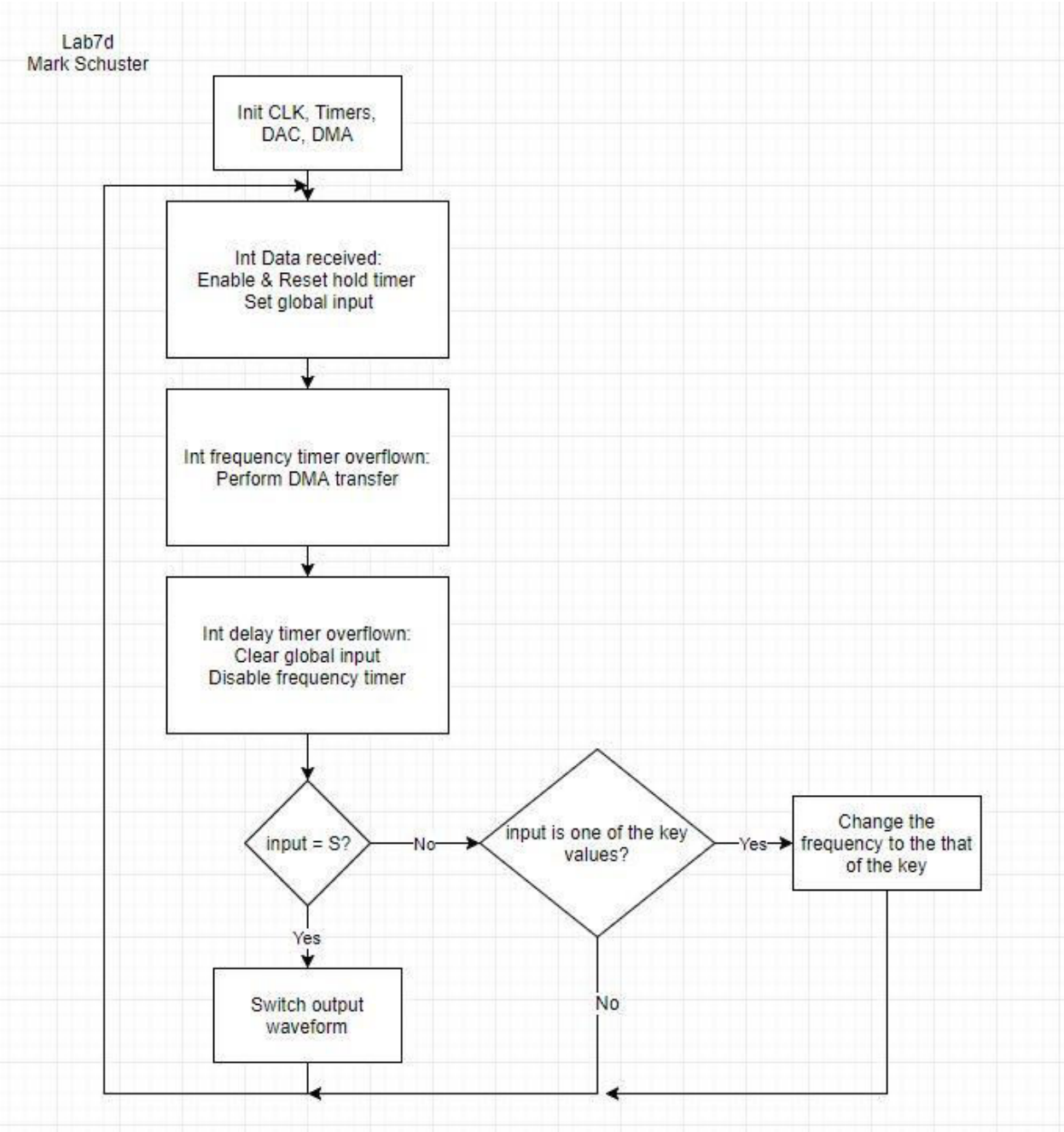
Lab7a flow diagram:



Lab7b flow diagram:



Lab7d flow diagram:



G) Program Code:

Lab7a.c:

```
// Lab 7 part A
// Name:         Mark L. Schuster
// Section #:    1540
// TA Name:      Christopher Crary
// Description:  Output a sine wave from a LUT.

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

#define CLK_PRESCALER       CLK_PSADIV_1_gc

void INIT_CLK(void);
void INIT_INTS(void);
void INIT_DAC(void);
void setTimers(void);

uint16_t sineLUTData[256] ={
          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)
{
          // Init CLK, inturrpts, DAC, and timers.
          INIT_CLK();
          INIT_INTS();
          INIT_DAC();
          setTimers();

          while (1);

          return 0;
}


/*********************FUNCTIONS*************************************/
// Subroutine Name: INIT_CLK
// Init the CLK to 32MHz.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_CLK(void){
          // Enable 32Mhz CLK.
```

```
        OSC_CTRL = OSC_RC32MEN_bm;

        // Wait for 32Mhz flag to be set.
        while( !(OSC_STATUS & OSC_RC32MRDY_bm) );

        // Write to restriction register to allow writing
        // to the CLK CTRL, then sel the 32MHz CLK.
        CPU_CCP = CCP_IOREG_gc;
        CLK_CTRL = CLK_SCLKSEL_RC32M_gc;

        // Write to restriction register to allow writing
        // to the CLK PSCTRL, then set the prescaler.
        CPU_CCP = CCP_IOREG_gc;
        CLK_PSCTRL = CLK_PRESCALER;

        return;
}

/*********************FUNCTIONS*************************************/
// Subroutine Name: INIT_INTS
// Init interrupts.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_INTS(void){
        // Set the PMIC to enable low level interrupts.
        PMIC_CTRL = PMIC_LOLVLEN_bm;

        // Set the interrupt enable bit.
        CPU_SREG |= 0x80;

        return;
}

/*********************FUNCTIONS*************************************/
// Subroutine Name: INIT_DAC
// Init the DAC.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_DAC(void){
        // Set the direction of the DAC output.
        PORTA_DIRSET = 0x08;

        // Set to single channel mode.
        DACA.CTRLB = DAC_CHSEL0_bm; // Header conflicts with doc3881. This value enables channel 1.

        // Set PortB as the external reference.
        DACA.CTRLC = 0b00011000;

        // Set the persistent data for 0.7V out.
        DACA.CH1DATA= 0x0000;

        // Enable channel 0 and the DAC.
        DACA.CTRLA = (DAC_CH1EN_bm | DAC_ENABLE_bm);

        return;
}


/*********************FUNCTIONS*************************************/
// Subroutine Name: setTimers
// Inits the timers.
// Inputs: None
// Outputs: None
// Affected: None
void setTimers(void){

        //Set the period.
        TCC0_PER = 214;

        // Enable the interrupt and then the timer.
        TCC0_INTCTRLA = TC_OVFINTLVL_LO_gc;

        TCC0_CTRLA = TC_CLKSEL_DIV2_gc;

        return;
}
```

```
/********************INTERUPT*************************************/
// Sets up an interrupt to be triggered by TC0 being overflown.
// Inputs: None
// Outputs: None
// Affected: None
ISR(TCC0_OVF_vect)
{
	static uint8_t index = 0;

	DACA.CH1DATA = sineLUTData[index++];

	return;
}
```

Lab7b.c:

```
// Lab 7 part B
// Name:         Mark L. Schuster
// Section #:    1540
// TA Name:      Christopher Crary
// Description:  Output a sine wave via DAC using 256 samples using DMA.

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

#define CLK_PRESCALER       CLK_PSADIV_1_gc

void INIT_CLK(void);
void INIT_INTS(void);
void INIT_ADC(void);
void INIT_DAC(void);
void setTimers(void);
void INIT_DMA(void);

uint16_t sineLUTData[256] ={
0x10,0x20,0x30,0x40,0x50,0x60,0x70,0x80,
0x90,0xa0,0xb0,0xc0,0xd0,0xe0,0xf0,0x100,
0x110,0x120,0x130,0x140,0x150,0x160,0x170,0x180,
0x190,0x1a0,0x1b0,0x1c0,0x1d0,0x1e0,0x1f0,0x200,
0x210,0x220,0x230,0x240,0x250,0x260,0x270,0x280,
0x290,0x2a0,0x2b0,0x2c0,0x2d0,0x2e0,0x2f0,0x300,
0x310,0x320,0x330,0x340,0x350,0x360,0x370,0x380,
0x390,0x3a0,0x3b0,0x3c0,0x3d0,0x3e0,0x3f0,0x400,
0x410,0x420,0x430,0x440,0x450,0x460,0x470,0x480,
0x490,0x4a0,0x4b0,0x4c0,0x4d0,0x4e0,0x4f0,0x500,
0x510,0x520,0x530,0x540,0x550,0x560,0x570,0x580,
0x590,0x5a0,0x5b0,0x5c0,0x5d0,0x5e0,0x5f0,0x600,
0x610,0x620,0x630,0x640,0x650,0x660,0x670,0x680,
0x690,0x6a0,0x6b0,0x6c0,0x6d0,0x6e0,0x6f0,0x700,
0x710,0x720,0x730,0x740,0x750,0x760,0x770,0x780,
0x790,0x7a0,0x7b0,0x7c0,0x7d0,0x7e0,0x7f0,0x800,
0x80f,0x81f,0x82f,0x83f,0x84f,0x85f,0x86f,0x87f,
0x88f,0x89f,0x8af,0x8bf,0x8cf,0x8df,0x8ef,0x8ff,
0x90f,0x91f,0x92f,0x93f,0x94f,0x95f,0x96f,0x97f,
0x98f,0x99f,0x9af,0x9bf,0x9cf,0x9df,0x9ef,0x9ff,
0xa0f,0xa1f,0xa2f,0xa3f,0xa4f,0xa5f,0xa6f,0xa7f,
0xa8f,0xa9f,0xaaf,0xabf,0xacf,0xadf,0xaef,0xaff,
0xb0f,0xb1f,0xb2f,0xb3f,0xb4f,0xb5f,0xb6f,0xb7f,
0xb8f,0xb9f,0xbaf,0xbbf,0xbcf,0xbdf,0xbef,0xbff,
0xc0f,0xc1f,0xc2f,0xc3f,0xc4f,0xc5f,0xc6f,0xc7f,
0xc8f,0xc9f,0xcaf,0xcbf,0xccf,0xcdf,0xcef,0xcff,
0xd0f,0xd1f,0xd2f,0xd3f,0xd4f,0xd5f,0xd6f,0xd7f,
0xd8f,0xd9f,0xdaf,0xdbf,0xdcf,0xddf,0xdef,0xdff,
0xe0f,0xe1f,0xe2f,0xe3f,0xe4f,0xe5f,0xe6f,0xe7f,
0xe8f,0xe9f,0xeaf,0xebf,0xecf,0xedf,0xeef,0xeff,
0xf0f,0xf1f,0xf2f,0xf3f,0xf4f,0xf5f,0xf6f,0xf7f,
0xf8f,0xf9f,0xfaf,0xfbf,0xfcf,0xfdf,0xfef,0xfff
};

int main(void)
{
        // Init CLK, inturrpts, DAC, DMA, and timers.
        INIT_CLK();
        INIT_DAC();
        INIT_DMA();
        setTimers();
        INIT_INTS();

        while (1);

        return 0;
}

/*********************FUNCTIONS*************************************/
// Subroutine Name: INIT_CLK
// Init the CLK to 32MHz.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_CLK(void){
        // Enable 32Mhz CLK.
        OSC_CTRL = OSC_RC32MEN_bm;
```

```
        // Wait for 32Mhz flag to be set.
        while( !(OSC_STATUS & OSC_RC32MRDY_bm) );

        // Write to restriction register to allow writing
        // to the CLK CTRL, then sel the 32MHz CLK.
        CPU_CCP = CCP_IOREG_gc;
        CLK_CTRL = CLK_SCLKSEL_RC32M_gc;

        // Write to restriction register to allow writing
        // to the CLK PSCTRL, then set the prescaler.
        CPU_CCP = CCP_IOREG_gc;
        CLK_PSCTRL = CLK_PRESCALER;

        return;
}

/*********************FUNCTIONS*************************************/
// Subroutine Name: INIT_INTS
// Init interrupts.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_INTS(void){
        // Set the PMIC to enable low level interrupts.
        PMIC_CTRL = PMIC_LOLVLEN_bm;

        // Set the interrupt enable bit.
        CPU_SREG |= 0x80;

        return;
}


/*********************FUNCTIONS*************************************/
// Subroutine Name: INIT_DAC
// Init the DAC.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_DAC(void){
        // Set the direction of the DAC output.
        PORTA_DIRSET = 0x08;

        // Set to single channel mode.
        DACA.CTRLB = DAC_CHSEL0_bm; // Header conflicts with doc3881. This value enables channel 1.

        // Set PortB as the external reference.
        DACA.CTRLC = 0b00011000;

        // Set the persistent data for 0.7V out.
        DACA.CH1DATA= 0x0000;

        // Enable channel 0 and the DAC.
        DACA.CTRLA = (DAC_CH1EN_bm | DAC_ENABLE_bm);

        return;
}


/*********************FUNCTIONS*************************************/
// Subroutine Name: INIT_DMA
// Init DMA.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_DMA(void){

        DMA.CTRL = 0;
        DMA.CTRL = DMA_RESET_bm;
        while ((DMA.CTRL & DMA_RESET_bm) != 0);

        DMA.CTRL                    = DMA_ENABLE_bm;
        DMA.CH0.CTRLA               = DMA_CH_ENABLE_bm | DMA_CH_BURSTLEN0_bm | DMA_CH_REPEAT_bm | DMA_CH_SINGLE_bm
        ;
        DMA.CH0.ADDRCTRL   = DMA_CH_SRCRELOAD_BLOCK_gc | DMA_CH_SRCDIR_INC_gc | DMA_CH_DESTRELOAD_BURST_gc |
DMA_CH_DESTDIR_INC_gc;
        DMA.CH0.TRIGSRC             = DMA_CH_TRIGSRC_TCC0_OVF_gc;
        DMA.CH0.TRFCNT              = 512;    // Sample num * bytes per smaple.
```

```
        DMA.CH0.REPCNT              = 0x00; // Repeat endlessly.
        DMA.CH0.SRCADDR0   = ( (uint16_t) &sineLUTData[0]) >> 0;
        DMA.CH0.SRCADDR1   = ( (uint16_t) &sineLUTData[0]) >> 8;
        DMA.CH0.SRCADDR2   = 0x00;
        DMA.CH0.DESTADDR0  = (( (uint16_t) &DACA.CH1DATA) >> 0) &0xFF;
        DMA.CH0.DESTADDR1  =  ( (uint16_t) &DACA.CH1DATA) >> 8;
        DMA.CH0.DESTADDR2  = 0x00;

        return;
}


/*********************FUNCTIONS*************************************/
// Subroutine Name: setTimers
// Used to created a delay.
// Inputs: None
// Outputs: None
// Affected: None
void setTimers(void){

        //Set the period.
        TCC0_PER = 213;

        // Enable the interrupt and then the timer.
        TCC0_INTCTRLA = TC_OVFINTLVL_LO_gc;

        // Set a prescaler of 2 for greater accuracy.
        TCC0_CTRLA = TC_CLKSEL_DIV2_gc;

        return;
}


/*********************INTERUPT*************************************/
// Sets up an interrupt to be triggered by TCC0 being overflown.
// Inputs: None
// Outputs: None
// Affected: None
ISR(TCC0_OVF_vect)
{
        return;
}
```

Lab7d.c:

```
// Lab 7 part D
// Name:         Mark L. Schuster
// Section #:    1540
// TA Name:      Christopher Crary
// Description:  Make a basic piano keyboard with input via UART.

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

#define CLK_PRESCALER       CLK_PSADIV_1_gc
#define PERIOD_300HZ        426.0
enum{
        SINE,
        SAW
};

void INIT_CLK(void);
void INIT_INTS(void);
void INIT_ADC(void);
void INIT_DAC(void);
void setTimers(void);
void INIT_DMA(void);
void INIT_USART(void);
void switchWaveform(void);
void playFrequency(double);
uint16_t roundDouble(double);

uint16_t sineLUTData[256] ={
        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
};

uint16_t sawLUTData[256] = {
        0x10,0x20,0x30,0x40,0x50,0x60,0x70,0x80,
        0x90,0xa0,0xb0,0xc0,0xd0,0xe0,0xf0,0x100,
        0x110,0x120,0x130,0x140,0x150,0x160,0x170,0x180,
        0x190,0x1a0,0x1b0,0x1c0,0x1d0,0x1e0,0x1f0,0x200,
        0x210,0x220,0x230,0x240,0x250,0x260,0x270,0x280,
        0x290,0x2a0,0x2b0,0x2c0,0x2d0,0x2e0,0x2f0,0x300,
        0x310,0x320,0x330,0x340,0x350,0x360,0x370,0x380,
        0x390,0x3a0,0x3b0,0x3c0,0x3d0,0x3e0,0x3f0,0x400,
        0x410,0x420,0x430,0x440,0x450,0x460,0x470,0x480,
        0x490,0x4a0,0x4b0,0x4c0,0x4d0,0x4e0,0x4f0,0x500,
        0x510,0x520,0x530,0x540,0x550,0x560,0x570,0x580,
        0x590,0x5a0,0x5b0,0x5c0,0x5d0,0x5e0,0x5f0,0x600,
        0x610,0x620,0x630,0x640,0x650,0x660,0x670,0x680,
```

```
        0x690,0x6a0,0x6b0,0x6c0,0x6d0,0x6e0,0x6f0,0x700,
        0x710,0x720,0x730,0x740,0x750,0x760,0x770,0x780,
        0x790,0x7a0,0x7b0,0x7c0,0x7d0,0x7e0,0x7f0,0x800,
        0x80f,0x81f,0x82f,0x83f,0x84f,0x85f,0x86f,0x87f,
        0x88f,0x89f,0x8af,0x8bf,0x8cf,0x8df,0x8ef,0x8ff,
        0x90f,0x91f,0x92f,0x93f,0x94f,0x95f,0x96f,0x97f,
        0x98f,0x99f,0x9af,0x9bf,0x9cf,0x9df,0x9ef,0x9ff,
        0xa0f,0xa1f,0xa2f,0xa3f,0xa4f,0xa5f,0xa6f,0xa7f,
        0xa8f,0xa9f,0xaaf,0xabf,0xacf,0xadf,0xaef,0xaff,
        0xb0f,0xb1f,0xb2f,0xb3f,0xb4f,0xb5f,0xb6f,0xb7f,
        0xb8f,0xb9f,0xbaf,0xbbf,0xbcf,0xbdf,0xbef,0xbff,
        0xc0f,0xc1f,0xc2f,0xc3f,0xc4f,0xc5f,0xc6f,0xc7f,
        0xc8f,0xc9f,0xcaf,0xcbf,0xccf,0xcdf,0xcef,0xcff,
        0xd0f,0xd1f,0xd2f,0xd3f,0xd4f,0xd5f,0xd6f,0xd7f,
        0xd8f,0xd9f,0xdaf,0xdbf,0xdcf,0xddf,0xdef,0xdff,
        0xe0f,0xe1f,0xe2f,0xe3f,0xe4f,0xe5f,0xe6f,0xe7f,
        0xe8f,0xe9f,0xeaf,0xebf,0xecf,0xedf,0xeef,0xeff,
        0xf0f,0xf1f,0xf2f,0xf3f,0xf4f,0xf5f,0xf6f,0xf7f,
        0xf8f,0xf9f,0xfaf,0xfbf,0xfcf,0xfdf,0xfef,0xfff
};


static char input = '0';
int main(void)
{
        // Init CLK, pin dir, and ADC.
        INIT_CLK();
        INIT_USART();
        INIT_DAC();
        INIT_DMA();
        setTimers();
        INIT_INTS();
        PORTC_DIRSET = 0x80;
        PORTC_OUT = 0x80;

        while (1){

                // Manage the input.
                if( (input == 'S') | (input == 's') ) {
                        switchWaveform();
                        input =0;
                }else if( (input == 'W') | (input == 'w') ){
                        TCC0_CTRLA = TC_CLKSEL_DIV1_gc;
                        playFrequency(1046.5);

                }else if( (input == '3') ){
                        TCC0_CTRLA = TC_CLKSEL_DIV1_gc;
                        playFrequency(1108.73);

                }else if( (input == 'E') | (input == 'e') ){
                        TCC0_CTRLA = TC_CLKSEL_DIV1_gc;
                        playFrequency(1174.66);

                }else if( (input == '4') ){
                        TCC0_CTRLA = TC_CLKSEL_DIV1_gc;
                        playFrequency(1244.51);

                }else if( (input == 'R') | (input == 'r') ){
                        TCC0_CTRLA = TC_CLKSEL_DIV1_gc;
                        playFrequency(1318.51);

                }else if( (input == 'T') | (input == 't') ){
                        TCC0_CTRLA = TC_CLKSEL_DIV1_gc;
                        playFrequency(1396.91);

                }else if( (input == '6') ){
                        TCC0_CTRLA = TC_CLKSEL_DIV1_gc;
                        playFrequency(1479.98);

                }else if( (input == 'Y') | (input == 'y') ){
                        TCC0_CTRLA = TC_CLKSEL_DIV1_gc;
                        playFrequency(1567.98);

                }else if( (input == '7') ){
                        TCC0_CTRLA = TC_CLKSEL_DIV1_gc;
                        playFrequency(1661.22);

                }else if( (input == 'U') | (input == 'u') ){
                        TCC0_CTRLA = TC_CLKSEL_DIV1_gc;
```

```
                playFrequency(1760.00);

        }else if( (input == '8') ){
                TCC0_CTRLA = TC_CLKSEL_DIV1_gc;
                playFrequency(1864.66);

        }else if( (input == 'I') | (input == 'i') ){
                TCC0_CTRLA = TC_CLKSEL_DIV1_gc;
                playFrequency(1975.53);

        }else if( (input == 'O') | (input == 'o') ){
                TCC0_CTRLA = TC_CLKSEL_DIV1_gc;
                playFrequency(2093.00);

        }else if( (input == '9') ){
                TCC0_CTRLA = TC_CLKSEL_DIV1_gc;
                playFrequency(2217.46);

        }else
                // If the input is not one of the defined above output nothing.
                DACA.CH1DATA = 0x00;
    }

    return 0;
}

/*********************FUNCTIONS***************************************/
// Subroutine Name: INIT_CLK
// Init the CLK to 32MHz.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_CLK(void){
    // Enable 32Mhz CLK.
    OSC_CTRL = OSC_RC32MEN_bm;

    // Wait for 32Mhz flag to be set.
    while( !(OSC_STATUS & OSC_RC32MRDY_bm) );

    // Write to restriction register to allow writing
    // to the CLK CTRL, then sel the 32MHz CLK.
    CPU_CCP = CCP_IOREG_gc;
    CLK_CTRL = CLK_SCLKSEL_RC32M_gc;

    // Write to restriction register to allow writing
    // to the CLK PSCTRL, then set the prescaler.
    CPU_CCP = CCP_IOREG_gc;
    CLK_PSCTRL = CLK_PRESCALER;

    return;
}


/*********************FUNCTIONS***************************************/
// Subroutine Name: INIT_USART
// Init the USART regs.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_USART(void){
    // Set the direction of the Tx & Rx pins.
    PORTD_DIRSET = USART_TXEN_bm;
    PORTD_DIRCLR = USART_RXEN_bm;

    // Set the baud rate.
    USARTD0_BAUDCTRLA = 1;
    USARTD0_BAUDCTRLB = 0;

    // Set the data size and the mode.
    USARTD0_CTRLC = USART_CHSIZE_8BIT_gc;
    USARTD0_CTRLB = 0b00011000;

    USARTD0_CTRLA = USART_RXCINTLVL_LO_gc;

    return;
}


/*********************FUNCTIONS***************************************/
```

```
// Subroutine Name: INIT_INTS
// Init interrupts.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_INTS(void){
	// Set the PMIC to enable low level interrupts.
	PMIC_CTRL = PMIC_LOLVLEN_bm;

	// Set the interrupt enable bit.
	CPU_SREG |= 0x80;

	return;
}


/*********************FUNCTIONS***********************************/
// Subroutine Name: INIT_DAC
// Init the DAC.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_DAC(void){
	// Set the direction of the DAC output.
	PORTA_DIRSET = 0x08;

	// Set to single channel mode.
	DACA.CTRLB = DAC_CHSEL0_bm; // Header conflicts with doc3881. This value enables channel 1.

	// Set PortB as the external reference.
	DACA.CTRLC = 0b00011000;

	// Set the persistent data for 0.7V out.
	DACA.CH1DATA= 0x0000;

	// Enable channel 0 and the DAC.
	DACA.CTRLA = (DAC_CH1EN_bm | DAC_ENABLE_bm);

}


/*********************FUNCTIONS***********************************/
// Subroutine Name: INIT_DMA
// Init DMA.
// Inputs: None
// Outputs: None
// Affected: None
void INIT_DMA(void){

	DMA.CTRL = 0;
	DMA.CTRL = DMA_RESET_bm;
	while ((DMA.CTRL & DMA_RESET_bm) != 0);

	DMA.CTRL                    = DMA_ENABLE_bm;
	DMA.CH0.CTRLA               = DMA_CH_ENABLE_bm | DMA_CH_BURSTLEN0_bm | DMA_CH_REPEAT_bm | DMA_CH_SINGLE_bm
	;
	DMA.CH0.ADDRCTRL   = DMA_CH_SRCRELOAD_BLOCK_gc | DMA_CH_SRCDIR_INC_gc | DMA_CH_DESTRELOAD_BURST_gc |
DMA_CH_DESTDIR_INC_gc;
	DMA.CH0.TRIGSRC             = DMA_CH_TRIGSRC_TCC0_OVF_gc;
	DMA.CH0.TRFCNT              = 512;    // Sample num * bytes per smaple.
	DMA.CH0.REPCNT              = 0x00; // Repeat endlessly.
	DMA.CH0.SRCADDR0   = ( (uint16_t) &sineLUTData[0]) >> 0;
	DMA.CH0.SRCADDR1   = ( (uint16_t) &sineLUTData[0]) >> 8;
	DMA.CH0.SRCADDR2   = 0x00;
	DMA.CH0.DESTADDR0  = (( (uint16_t) &DACA.CH1DATA) >> 0) &0xFF;
	DMA.CH0.DESTADDR1  =  ( (uint16_t) &DACA.CH1DATA) >> 8;
	DMA.CH0.DESTADDR2  = 0x00;

	return;
}


/*********************FUNCTIONS***********************************/
// Subroutine Name: setTimers
// Inits the two timers used.
// Inputs: None
// Outputs: None
// Affected: TCC0_PER, TCC1_PER, TCC0_INTCTRLA, TCC1_INTCTRLA, TCC0_INTCTRLA,
//                           TCC1_INTCTRLA
```

```
void setTimers(void){

	//Set the period.
	TCC0_PER = PERIOD_300HZ;

	// Enable the interrupt and then the timer.
	TCC0_INTCTRLA = TC_OVFINTLVL_LO_gc;

	// Set a prescaler of 2 for greater accuracy.
	TCC0_CTRLA = TC_CLKSEL_OFF_gc;

	//Set the period.
	TCC1_PER = 0x1000;

	// Enable the interrupt and then the timer.
	TCC1_INTCTRLA = TC_OVFINTLVL_LO_gc;

	// Set a prescaler of 2 for greater accuracy.
	TCC1_CTRLA = TC_CLKSEL_DIV1024_gc;

	return;
}


/*********************FUNCTIONS**************************************/
// Subroutine Name: switchWaveform
// Used to swtich the waveform to be outputted.
// Inputs: None
// Outputs: None
// Affected: DMA.CH0.CTRLA, DMA.CH0.SRCADDR(0/1/2), DMA.CH0.TRFCNT
void switchWaveform(void){
	// Set state to determine the current waveform.
	static uint8_t waveState = SAW;

	// Disable the DMA.
	DMA.CH0.CTRLA &= ~DMA_CH_ENABLE_bm;
	while(!(DMA.CH0.CTRLA | DMA_CH_ENABLE_bm));

	// Update the DMA source depending on the state.
	if(waveState == SINE){
		DMA.CH0.SRCADDR0   = ( (uint16_t) sineLUTData) >> 0;
		DMA.CH0.SRCADDR1   = ( (uint16_t) sineLUTData) >> 8;
		DMA.CH0.SRCADDR2   = 0x00;
		waveState = SAW;
	}else if(waveState == SAW){
		DMA.CH0.SRCADDR0   = ( (uint16_t) sawLUTData) >> 0;
		DMA.CH0.SRCADDR1   = ( (uint16_t) sawLUTData) >> 8;
		DMA.CH0.SRCADDR2   = 0x00;
		waveState = SINE;
	}

	// Re-enable the DMA and reset its block size.
	DMA.CH0.CTRLA |= DMA_CH_ENABLE_bm;
	DMA.CH0.TRFCNT              = 512;

	return;
}


/*********************FUNCTIONS**************************************/
// Subroutine Name: playFrequency
// Change the output frequncy to the passed frequency.
// Inputs: double frequency - Frequency to be outputted.
// Outputs: None
// Affected: TCC0_PER
void playFrequency(double frequency){

	double newPeriod = (PERIOD_300HZ * 300.0) / frequency;
	uint16_t newIntPeriod = roundDouble(newPeriod);
	TCC0_PER = newIntPeriod;

	return;
}


/*********************FUNCTIONS**************************************/
// Subroutine Name: roundDouble
// Change the output frequncy to the passed frequency.
// Inputs: double d - Double to be rounded.
```

```
// Outputs: uint16_t - Rounded integer value of the passed double.
// Affected: None
uint16_t roundDouble(double d){
	return d < 0 ? d - 0.5 : d + 0.5;
}


/********************INTERUPT*****************************************/
// Sets up an interrupt to be triggered by TCC0 being overflown.
// Inputs: None
// Outputs: None
// Affected: None
ISR(TCC0_OVF_vect)
{
	return;
}


/********************INTERUPT*****************************************/
// Sets up an interrupt to be triggered by TCC1 being overflown.
// Inputs: None
// Outputs: None
// Affected: TCC0_CTRLA, input
ISR(TCC1_OVF_vect){
	input = 0;
	TCC0_CTRLA = TC_CLKSEL_OFF_gc;

	return;
}


/********************INTERUPT*****************************************/
// Sets up an interrupt to be triggered by a character being
// over UART.
// Inputs: None
// Outputs: None
// Affected: TCC1_CNT, input
ISR(USARTD0_RXC_vect){
	TCC1_CNT = 0;
	input = USARTD0_DATA;

	return;
}
```

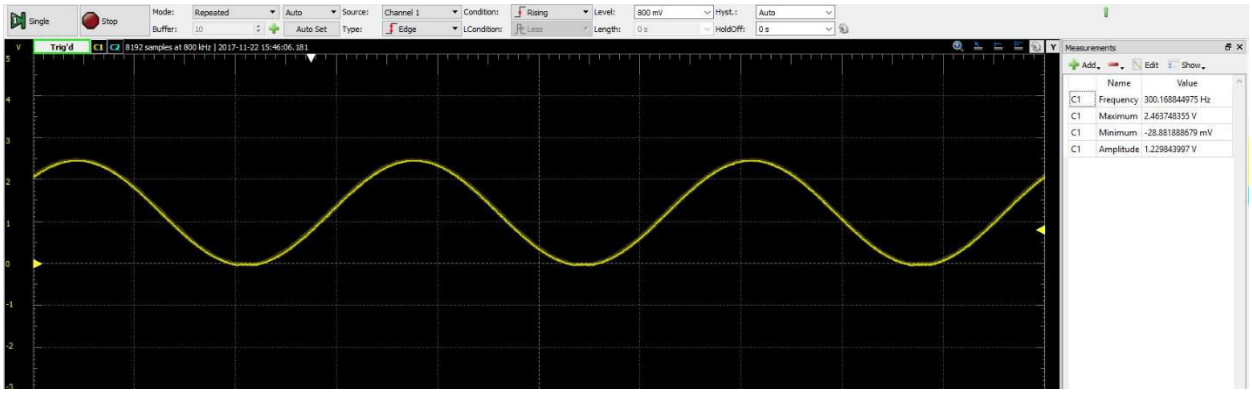
H) Appendix:

Files:

- Lab7.pdf
- Lab7a.c
- Lab7b.c
- Lab7d.c

Screenshots:

Non-DMA Sinusoid:



DMA Sinusoid: