Lab 7
Jineidy Mak
Secton 7F31

Pre-Lab Questions:
1. If the NAD/DAD board were not available, nor any conventional desktop measuring tools, how could you verify that your DAC were functioning properly?

Sending the output to the input of the ADC internally and checking with code using the previous labs LCD. Can check for on and off voltage by using an LED.

2. You probably noticed that your sine wave does not look very good. While at least meeting specifications, how could you increase the quality of your sine wave?

More data points. Using float type for arithmetic. More resolution for both timer and dac.

3. How many DMA channels are available on the XMEGA?

3 channels.

4. How many different options are there to trigger the DMA?

(Too many) 0xB3 sources.

5. While you were not asked to implement this on your board, explain how one might vary the amplitude of an output wave, much like the frequency was varied.

Wave can be varied by simply using a LuT that had values of a different amplitude. We could have easily added a multiplier to the conversion process of each point. If we wanted large amplitudes we would need a gain and that could be added through external circuit.

Problems Encountered:

Understanding that the voltage is suppose to peak at 2.5 V helps.

Future Work/Applications:

Can use it to transfer large amounts of data through UART, ADC and DAC while the processor is doing something else. Could be useful for real-time data monitoring.

Schematics:

None for this lab.

Decoding Logic:

```
library ieee;
use ieee.std_logic_1164.all;

entity lab3 is
  port (

    CS0_L   : in  std_logic;
    CS1_L   : in  std_logic;
    RE_L      : in  std_logic;
    WE_L      : in  std_logic;
    A15                     : in std_logic;
    A14                     : in std_logic;
    A13                     : in std_logic;
    A12                     : in std_logic;
    LCD_En  : out std_logic

    );

end lab3;


architecture BHV of lab3 is

begin

process(CS0_L, CS1_L, RE_L, WE_L, A15, A14, A13, A12)

                variable CS0_H : std_logic;
                variable CS1_H : std_logic;
                variable A15_L : std_logic;
                variable A14_L : std_logic;
                variable A13_L : std_logic;
                variable RE_H : std_logic;
                variable WE_H : std_logic;

begin

  CS0_H := not(CS0_L);
  CS1_H := not(CS1_L);
  RE_H := not(RE_L);
  WE_H := not(WE_L);
  A15_L := not(A15);
  A14_L := not(A14);
  A13_L := not(A13);

  LCD_En <= CS1_H and (RE_H or WE_H) and ((A15_L and A14_L and A13) or (A15_L and A14));

 end process;

end BHV;
```

Pseudocode/Flowcharts:

PartA:
Initialize DAC.
Use equation to figure out DATA input to DAC using a 2.5V Vref.
Send DATA to DAC continuously using a while loop while polling for busy flag.

PartB:
Initialize DAC
Initialize TC
Convert all table values to DATA values.
Use for loop to send each value to DAC as TC overflows. TC overflows will trigger the DAC to convert a point at a time. This allows us to use TC as control for frequency.

PartC:
Initialize DAC
Initialize TC
Initialize DMA
All parts are same as PartB, the exception is instead of using a for loop to send DATA values to DAC, use the DMA which is also triggered by the TC.

PartD:

Same as PartC for each part.
Use a switch case for each keypad.
Poll keypad and find the value to select the case.
Each case should be a frequency multiple of 50.
Use a while loop and continuous keypad polling to change the selection in real-time.

Program Code:

Part A:

```
/*
 * lab7a.c
 *
 * Created: 7/25/2016 3:18:40 PM
 * Author : James Mak
 */

#include <avr/io.h>

void DAC_INIT()
{
		PORTB_DIRSET = 0x04; //Pin 2 to output.
		PORTB_OUT = 0x00;
		DACB.CTRLA = DAC_CH0EN_bm | DAC_ENABLE_bm;
		DACB.CTRLB = 0x00;
		DACB.CTRLC = DAC_REFSEL_AREFB_gc;

}

void DAC_WRITE(int number)
{
		   while(!(DACB_STATUS & DAC_CH0DRE_bm));
				DACB_CH0DATA = number;

}


int main(void)
{
    DAC_INIT();

		float value = 0;
		int value_int = 0;

		value = ((1.3 * 2047) / 2.5) - 4095;
		value_int = 2048 - (int)value;

		PORTB_PIN2CTRL = 0x40;
    while (1)
    {

				DAC_WRITE(value_int);

    }
}
```

Part B:

```
/*
 * lab7b.c
 *
 * Created: 7/25/2016 7:25:32 PM
 * Author : James Mak
 */

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

float triLuT[] = {4,8,12,16,20,23,27,31,
		35,39,43,47,51,55,59,63,
		66,70,74,78,82,86,90,94,
		98,102,105,109,113,117,121,125,
		129,133,137,141,145,148,152,156,
		160,164,168,172,176,180,184,188,
		191,195,199,203,207,211,215,219,
		223,227,230,234,238,242,246,250,
		246,242,238,234,230,227,223,219,
		215,211,207,203,199,195,191,188,
		184,180,176,172,168,164,160,156,
		152,148,145,141,137,133,129,125,
		121,117,113,109,105,102,98,94,
		90,86,82,78,74,70,66,63,
		59,55,51,47,43,39,35,31,
		27,23,20,16,12,8,4,0};

float sineLuT[] = {125,131,137,143,149,155,161,167,
		173,178,184,189,194,199,204,209,
		213,218,222,225,229,232,235,238,
		240,243,245,246,248,249,249,250,
		250,250,249,249,248,246,245,243,
		240,238,235,232,229,225,222,218,
		213,209,204,199,194,189,184,178,
		173,167,161,155,149,143,137,131,
		125,119,113,107,101,95,89,83,
		77,72,66,61,56,51,46,41,
		37,32,28,25,21,18,15,12,
		10,7,5,4,2,1,1,0,
		0,0,1,1,2,4,5,7,
		10,12,15,18,21,25,28,32,
		37,41,46,51,56,61,66,72,
		77,83,89,95,101,107,113,119};

int sineLuT_conv[128];

void Delay_XuS(int a) // Delay a certain amount of uS.
{
		for(int i = 0; i < a; i++);
}

void DAC_INIT()
{
		PORTB_DIRSET = 0x04; //Pin 2 to output.
	PORTB_OUT = 0x00;
	DACB.CTRLA = DAC_CH0EN_bm | DAC_ENABLE_bm; //Enable Ch0 and enable the entire DAC.
	DACB.CTRLB = 0x01; //Auto triggered ch0. When the event system is triggered the DAC converts the next value.
	DACB.CTRLC = DAC_REFSEL_AREFB_gc;
	DACB.EVCTRL = 0x00; //CH0 event triggers DAC.
}

void DAC_POLL() //Polls the busy flag of the DAC.
{
	int bf = DACB_STATUS;
	Delay_XuS(400);

	while((bf & 0xFD) == 0x00)
	{
		bf = DACB_STATUS;
	}

}
void DAC_WRITE(int number)
{
	DAC_POLL();
	Delay_XuS(400);
	DACB_CH0DATA = number;

}

void Convert()
{
	float value = 0;
	int value_int = 0;

	for(int i = 0; i < 128; i++)
	{
		value = ((sineLuT[i]/100) * 4095) / 2.5;
		sineLuT_conv[i] = (int)value;
	}
}

void DAC_WRITE_WAVE()
{
	for(int i = 0; i < 128; i++)
	{
		DAC_WRITE(sineLuT_conv[i]);
		Delay_XuS(400);
	}
}

void TC0_INIT(void)
{
	//We are using Port E TC because it is the only port available that isn't taken.
	TCE0.CTRLA = TC_CLKSEL_DIV1_gc;  //Turn on the Counter 2MHZ.
	TCE0.CTRLB = TC_WGMODE_FRQ_gc | TC0_CCAEN_bm;  //Waveform FRQ generation mode and Compare
enable A set.
	TCE0.CTRLE = TC_BYTEM_NORMAL_gc; //Normal mode TC.
	TCE0.CTRLFCLR = TC0_DIR_bm;  //Incrementing Counter.
	TCE0.CNT = 0; //Start count at zero.
	TCE0_CCA = 78; //Controls the TOP of the counter effectively controls frequency.
	EVSYS_CH0MUX = EVSYS_CHMUX_TCE0_OVF_gc; //When counter overflows then the event system is triggered
(see DAC_INIT).

}

ISR(TCE0_OVF_vect)
{
	TCE0_INTFLAGS = 0x01; //Clear the flag.
}

int main(void)
{
   DAC_INIT();
	TC0_INIT();
	Convert(); //Converts the look up table to the proper values.
	PORTB_PIN2CTRL = (PORT_OPC_TOTEM_gc | PORT_ISC_INPUT_DISABLE_gc); //These are set by default
anyways. No need to set.

   while(1)
   {
			DAC_WRITE_WAVE();
			Delay_XuS(400);
   }
}
```

<u>**Part C:**</u>

```
/*
 * lab7c.c
 *
 * Created: 7/26/2016 12:19:36 AM
 * Author : James Mak
 */

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

uint32_t table = 0;

float triLuT[] = {4,8,12,16,20,23,27,31,
	35,39,43,47,51,55,59,63,
	66,70,74,78,82,86,90,94,
	98,102,105,109,113,117,121,125,
	129,133,137,141,145,148,152,156,
	160,164,168,172,176,180,184,188,
	191,195,199,203,207,211,215,219,
	223,227,230,234,238,242,246,250,
	246,242,238,234,230,227,223,219,
	215,211,207,203,199,195,191,188,
	184,180,176,172,168,164,160,156,
	152,148,145,141,137,133,129,125,
	121,117,113,109,105,102,98,94,
        90,86,82,78,74,70,66,63,
        59,55,51,47,43,39,35,31,
        27,23,20,16,12,8,4,0};

float sineLuT[] = {125,131,137,143,149,155,161,167,
        173,178,184,189,194,199,204,209,
        213,218,222,225,229,232,235,238,
        240,243,245,246,248,249,249,250,
        250,250,249,249,248,246,245,243,
        240,238,235,232,229,225,222,218,
        213,209,204,199,194,189,184,178,
        173,167,161,155,149,143,137,131,
        125,119,113,107,101,95,89,83,
        77,72,66,61,56,51,46,41,
        37,32,28,25,21,18,15,12,
        10,7,5,4,2,1,1,0,
        0,0,1,1,2,4,5,7,
        10,12,15,18,21,25,28,32,
        37,41,46,51,56,61,66,72,
        77,83,89,95,101,107,113,119};

int sineLuT_conv[128];

void Delay_XuS(int a) // Delay a certain amount of uS.
{
        for(int i = 0; i < a; i++);
}

void DAC_INIT()
{
        PORTB_DIRSET = 0x04; //Pin 2 to output.
        PORTB_OUT = 0x00;
        DACB.CTRLA = DAC_CH0EN_bm | DAC_ENABLE_bm; //Enable Ch0 and enable the entire DAC.
        DACB.CTRLB = 0x01; //Auto triggered ch0. When the event system is triggered the DAC converts the next value.
        DACB.CTRLC = DAC_REFSEL_AREFB_gc;
        DACB.EVCTRL = 0x00; //CH0 event triggers DAC.
}

void DAC_POLL() //Polls the busy flag of the DAC.
{
        int bf = DACB_STATUS;
        Delay_XuS(400);

        while((bf & 0xFD) == 0x00)
        {
                bf = DACB_STATUS;
        }

}
void DAC_WRITE(int number)
{
        DAC_POLL();
        Delay_XuS(400);
        DACB_CH0DATA = number;

}
```

```
void Convert()
{
        float value = 0;
        int value_int = 0;

        for(int i = 0; i < 128; i++)
        {
                value = ((sineLuT[i]/100) * 4095) / 2.5;
                sineLuT_conv[i] = (int)value;
        }
}

void DAC_WRITE_WAVE()
{
        for(int i = 0; i < 128; i++)
        {
                DAC_WRITE(sineLuT_conv[i]);
                Delay_XuS(400);
        }
}

void DMA_INIT(uint32_t table_addr)
{
        DMA.CTRL = DMA_ENABLE_bm;
        DMA.CH0.CTRLA = DMA_CH_REPEAT_bm | DMA_CH_BURSTLEN_2BYTE_gc | DMA_CH_SINGLE_bm;
        DMA.CH0.REPCNT = 0;
        DMA.CH0.ADDRCTRL = DMA_CH_SRCRELOAD_TRANSACTION_gc | DMA_CH_SRCDIR_INC_gc |
DMA_CH_DESTRELOAD_BURST_gc | DMA_CH_DESTDIR_INC_gc;
        DMA.CH0.TRIGSRC = DMA_CH_TRIGSRC_EVSYS_CH0_gc;
        DMA.CH0.TRFCNT = 256;

        DMA.CH0.SRCADDR0 = table_addr >> 0;
        DMA.CH0.SRCADDR1 = table_addr >> 8;
        DMA.CH0.SRCADDR2 = table_addr >> 16;

        DMA.CH0.DESTADDR0 = (uint32_t)(&DACB_CH0DATA) >> 0;
        DMA.CH0.DESTADDR1 = (uint32_t)(&DACB_CH0DATA) >> 8;
        DMA.CH0.DESTADDR2 = (uint32_t)(&DACB_CH0DATA) >> 16;

        DMA_CH0_CTRLA = 0xA5;
}

void TC0_INIT(void)
{
        //We are using Port E TC because it is the only port available that isn't taken.
        TCE0.CTRLA = TC_CLKSEL_DIV1_gc;  //Turn on the Counter 2MHZ.
        TCE0.CTRLB = TC_WGMODE_FRQ_gc | TC0_CCAEN_bm;  //Waveform FRQ generation mode and Compare
enable A set.
        TCE0.CTRLE = TC_BYTEM_NORMAL_gc; //Normal mode TC.
        TCE0.CTRLFCLR = TC0_DIR_bm;  //Incrementing Counter.
        TCE0.CNT = 0; //Start count at zero.
        TCE0_CCA = 78; //Controls the TOP of the counter effectively controls frequency.
        EVSYS_CH0MUX = EVSYS_CHMUX_TCE0_OVF_gc; //When counter overflows then the event system is triggered
(see DAC_INIT).

}
```

```
ISR(TCE0_OVF_vect)
{
		TCE0_INTFLAGS = 0x01; //Clear the flag.
}

int main(void)
{
    DAC_INIT();
		TC0_INIT();
		Convert(); //Converts the look up table to the proper values.
		table = (uint32_t)&(sineLuT_conv);
		DMA_INIT(table);

		PORTB_PIN2CTRL = (PORT_OPC_TOTEM_gc | PORT_ISC_INPUT_DISABLE_gc); //These are set by default
anyways. No need to set.


    while(1)
    {
				Delay_XuS(400);
    }
}
```

<u>Part D:</u>

```
/*
 * lab7d.c
 *
 * Created: 7/26/2016 2:25:51 AM
 * Author : James Mak
 */

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

uint32_t table = 0;

float triLuT[] = {4,8,12,16,20,23,27,31,
		35,39,43,47,51,55,59,63,
		66,70,74,78,82,86,90,94,
		98,102,105,109,113,117,121,125,
		129,133,137,141,145,148,152,156,
		160,164,168,172,176,180,184,188,
		191,195,199,203,207,211,215,219,
		223,227,230,234,238,242,246,250,
		246,242,238,234,230,227,223,219,
		215,211,207,203,199,195,191,188,
		184,180,176,172,168,164,160,156,
		152,148,145,141,137,133,129,125,
		121,117,113,109,105,102,98,94,
		90,86,82,78,74,70,66,63,
		59,55,51,47,43,39,35,31,
          27,23,20,16,12,8,4,0};

float sineLuT[] = {125,131,137,143,149,155,161,167,
          173,178,184,189,194,199,204,209,
          213,218,222,225,229,232,235,238,
          240,243,245,246,248,249,249,250,
          250,250,249,249,248,246,245,243,
          240,238,235,232,229,225,222,218,
          213,209,204,199,194,189,184,178,
          173,167,161,155,149,143,137,131,
          125,119,113,107,101,95,89,83,
          77,72,66,61,56,51,46,41,
          37,32,28,25,21,18,15,12,
          10,7,5,4,2,1,1,0,
          0,0,1,1,2,4,5,7,
          10,12,15,18,21,25,28,32,
          37,41,46,51,56,61,66,72,
          77,83,89,95,101,107,113,119};

int triLuT_conv[128];

int sineLuT_conv[128];

void Delay_XuS(int a) // Delay a certain amount of uS.
{
          for(int i = 0; i < a; i++);
}

void DAC_INIT()
{
          PORTB_DIRSET = 0x04; //Pin 2 to output.
          PORTB_OUT = 0x00;
          DACB.CTRLA = DAC_CH0EN_bm | DAC_ENABLE_bm; //Enable Ch0 and enable the entire DAC.
          DACB.CTRLB = 0x01; //Auto triggered ch0. When the event system is triggered the DAC converts the next value.
          DACB.CTRLC = DAC_REFSEL_AREFB_gc;
          DACB.EVCTRL = 0x00; //CH0 event triggers DAC.
}

void DAC_POLL() //Polls the busy flag of the DAC.
{
          int bf = DACB_STATUS;
          Delay_XuS(400);

          while((bf & 0xFD) == 0x00)
          {
                    bf = DACB_STATUS;
          }

}
void DAC_WRITE(int number)
{
          DAC_POLL();
          Delay_XuS(400);
          DACB_CH0DATA = number;

}
```

```
void Convert()
{
	float value = 0;
	int value_int = 0;

	for(int i = 0; i < 128; i++)
	{
		value = ((sineLuT[i]/100) * 4095) / 2.5;
		sineLuT_conv[i] = (int)value;

		value = ((triLuT[i]/100) * 4095) / 2.5;
		triLuT_conv[i] = (int)value;

	}
}

void DAC_WRITE_WAVE()
{
	for(int i = 0; i < 128; i++)
	{
		DAC_WRITE(sineLuT_conv[i]);
		Delay_XuS(400);
	}
}

void DMA_INIT(uint32_t table_addr)
{
	DMA.CTRL = DMA_ENABLE_bm;
	DMA.CH0.CTRLA = DMA_CH_REPEAT_bm | DMA_CH_BURSTLEN_2BYTE_gc | DMA_CH_SINGLE_bm;
	DMA.CH0.REPCNT = 0;
	DMA.CH0.ADDRCTRL = DMA_CH_SRCRELOAD_TRANSACTION_gc | DMA_CH_SRCDIR_INC_gc |
DMA_CH_DESTRELOAD_BURST_gc | DMA_CH_DESTDIR_INC_gc;
	DMA.CH0.TRIGSRC = DMA_CH_TRIGSRC_EVSYS_CH0_gc;
	DMA.CH0.TRFCNT = 256;

	DMA.CH0.SRCADDR0 = table_addr >> 0;
	DMA.CH0.SRCADDR1 = table_addr >> 8;
	DMA.CH0.SRCADDR2 = table_addr >> 16;

	DMA.CH0.DESTADDR0 = (uint32_t)(&DACB_CH0DATA) >> 0;
	DMA.CH0.DESTADDR1 = (uint32_t)(&DACB_CH0DATA) >> 8;
	DMA.CH0.DESTADDR2 = (uint32_t)(&DACB_CH0DATA) >> 16;

}

void TC0_INIT(void)
{
	//We are using Port E TC because it is the only port available that isn't taken.
	TCE0.CTRLA = TC_CLKSEL_DIV1_gc;  //Turn on the Counter 2MHZ.
	TCE0.CTRLB = TC_WGMODE_FRQ_gc | TC0_CCAEN_bm;  //Waveform FRQ generation mode and Compare
enable A set.
	TCE0.CTRLE = TC_BYTEM_NORMAL_gc; //Normal mode TC.
	TCE0.CTRLFCLR = TC0_DIR_bm;  //Incrementing Counter.
	TCE0.CNT = 0; //Start count at zero.
	TCE0.CCA = 0;
	EVSYS_CH0MUX = EVSYS_CHMUX_TCE0_OVF_gc; //When counter overflows then the event system is triggered
(see DAC_INIT).
```

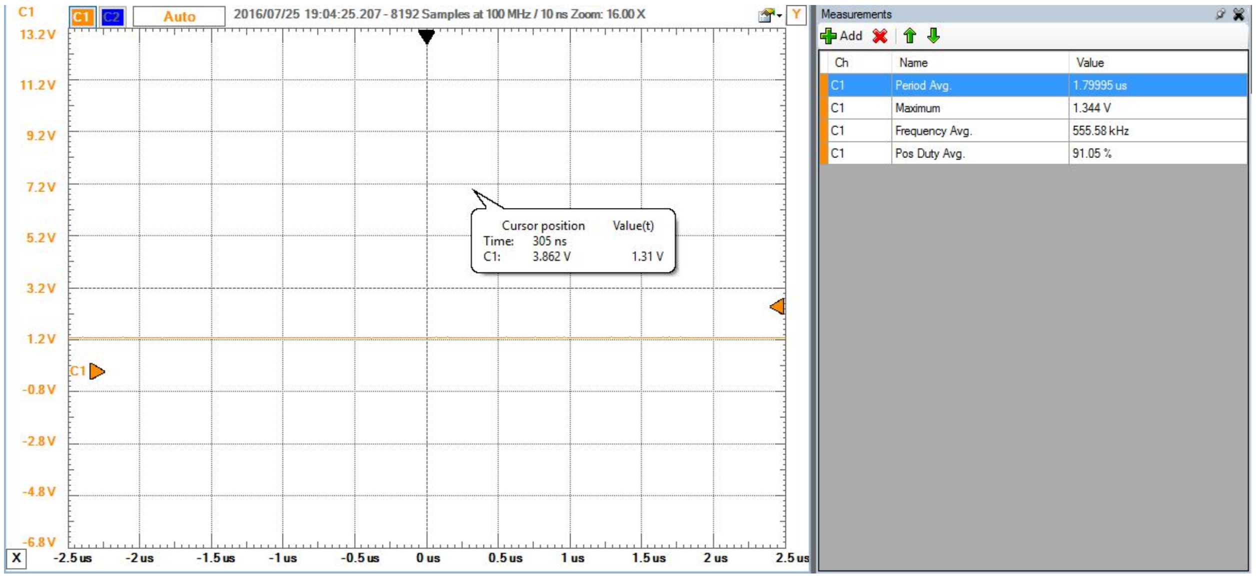
```
}

ISR(TCE0_OVF_vect)
{
        TCE0_INTFLAGS = 0x01; //Clear the flag.
}

void PORTF_INIT(void)
{

        PORTF.DIRSET = 0x0F;        //Configure the I/O pins of Port F.

        PORTF.PIN7CTRL = 0x18;        //Set pull-up resistor to pin 7.
        PORTF.PIN6CTRL = 0x18;        //Set pull-up resistor to pin 6.
        PORTF.PIN5CTRL = 0x18;        //Set pull-up resistor to pin 5.
        PORTF.PIN4CTRL = 0x18;                //Set pull-up resistor to pin 4.
}

char keypad_decode()
{
        switch(PORTF_IN)
        {
                Delay_XuS(400);
                case 0x77: return '1';
                break;
                case 0xB7: return '2';
                break;
                case 0xD7: return '3';
                break;
                case 0xE7: return 'A';
                break;
                case 0x7B: return '4';
                break;
                case 0xBB: return '5';
                break;
                case 0xDB: return '6';
                break;
                case 0xEB: return 'B';
                break;
                case 0x7D: return '7';
                break;
                case 0xBD: return '8';
                break;
                case 0xDD: return '9';
                break;
                case 0xED: return 'C';
                break;
                case 0x7E: return '*';
                break;
                case 0xBE: return '0';
                break;
                case 0xDE: return '#';
                break;
                case 0xEE: return 'D';
                break;
                default: return '\0';
                break;
        }
}

char keypad_poll()
{

        char key = '\0';

        while(key == '\0')
        {

                PORTF.OUT = 0x07;
                key = keypad_decode();
                if(key != '\0')
                {
                        return key;
                }
                PORTF.OUT = 0x0B;
                key = keypad_decode();
                if(key != '\0')
                {
                        return key;
                }
                PORTF.OUT = 0x0D;
                key = keypad_decode();
                if(key != '\0')
                {
                        return key;
                }
                PORTF.OUT = 0x0E;
                key = keypad_decode();
                if(key != '\0')
                {
                        return key;
                }
        }

        return key;
}


int main(void)
{
        PORTF_INIT();
  DAC_INIT();
        TC0_INIT();
        Convert(); //Converts the look up tables to the proper values.
        table = (uint32_t)&(sineLuT_conv);
        DMA_INIT(table);

        PORTB_PIN2CTRL = (PORT_OPC_TOTEM_gc | PORT_ISC_INPUT_DISABLE_gc); //These are set by default
anyways. No need to set.

        char key_pad = '0';
        int frequency = 3; //(50 Hz);
        int multiplier = 1;
```

```
uint8_t control = 0x25;

while(1)
{
        key_pad = keypad_poll();

        switch(key_pad)
        {
                case '0':
                                control = 0x25;
                                multiplier = -90+16;
                break;
                case '1':
                                control = 0xA5;
                                multiplier = 295+16;
                break;
                case '2':
                                control = 0xA5;
                                multiplier = 138+16;
                break;
                case '3':
                                control = 0xA5;
                                multiplier = 86+16;
                break;
                case '4':
                                control = 0xA5;
                                multiplier = 59+16;
                break;
                case '5':
                                control = 0xA5;
                                multiplier = 43+16;
                break;
                case '6':
                                control = 0xA5;
                                multiplier = 33+16;
                break;
                case '7':
                                control = 0xA5;
                                multiplier = 25+16;
                break;
                case '8':
                                control = 0xA5;
                                multiplier = 20+16;
                break;
                case '9':
                                control = 0xA5;
                                multiplier = 15+16;
                break;
                case 'A':
                                control = 0xA5;
                                multiplier = 12+16;
                break;
                case 'B':
                                control = 0xA5;
                                multiplier = 9+16;
                break;
                case 'C':
```

```
                                        control = 0xA5;
                                        multiplier = 6+16;
                        break;
                        case 'D':
                                        control = 0xA5;
                                        multiplier = 4+16;
                        break;
                        case '*':
                                        //control = 0x25;
                                        table = (uint32_t) &triLuT_conv;
                        break;
                        case '#':
                                        //control = 0x25;
                                        table = (uint32_t) &sineLuT_conv;
                        break;

                        default:
                        break;

                }

                        DMA.CH0.SRCADDR0 = table >> 0;
                        DMA.CH0.SRCADDR1 = table >> 8;
                        DMA.CH0.SRCADDR2 = table >> 16;
                        DMA_CH0_CTRLA = control;
                        TCE0_CCA = frequency + multiplier; //Controls the TOP of the counter effectively controls
frequency (16 is an offset for calibration purposes).
        }
}
```

Appendix:

Part A - 1.3V PortB Pin2.



Auto

2016/07/25 19:04:25.207 - 8192 Samples at 100 MHz / 10 ns Zoom: 16.00 X

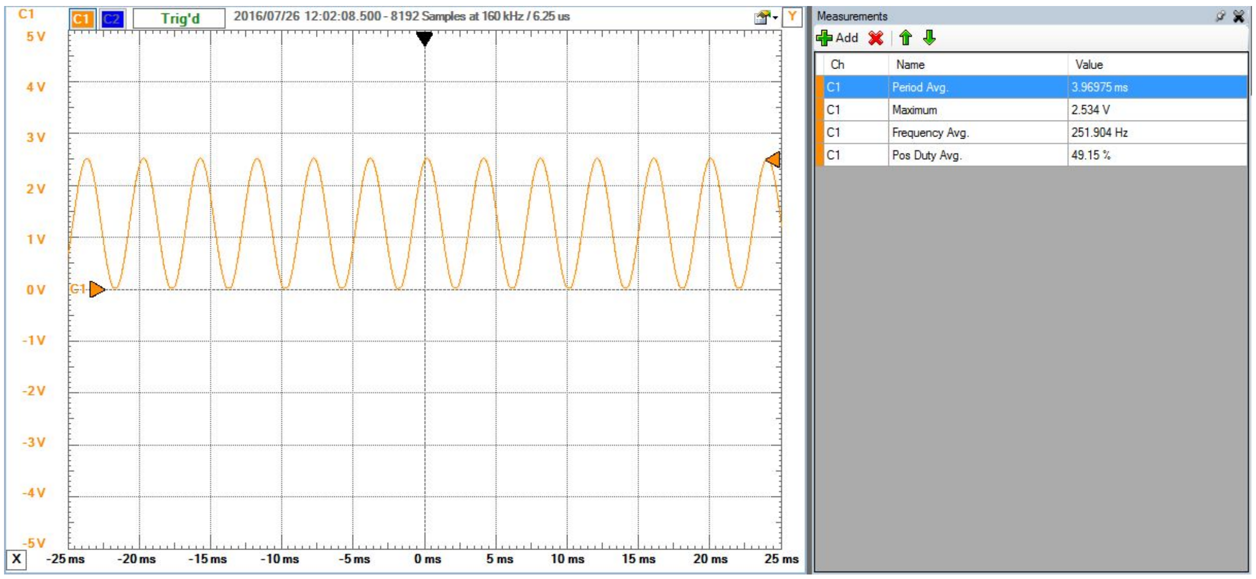
```
Cursor position        Value(t)
Time:    305 ns
C1:      3.862 V       1.31 V
```

Measurements

Add

Ch	Name	Value
C1	Period Avg.	1.79995 us
C1	Maximum	1.344 V
C1	Frequency Avg.	555.58 kHz
C1	Pos Duty Avg.	91.05 %

Part B - SineWave using TC at 250Hz.



Trig'd

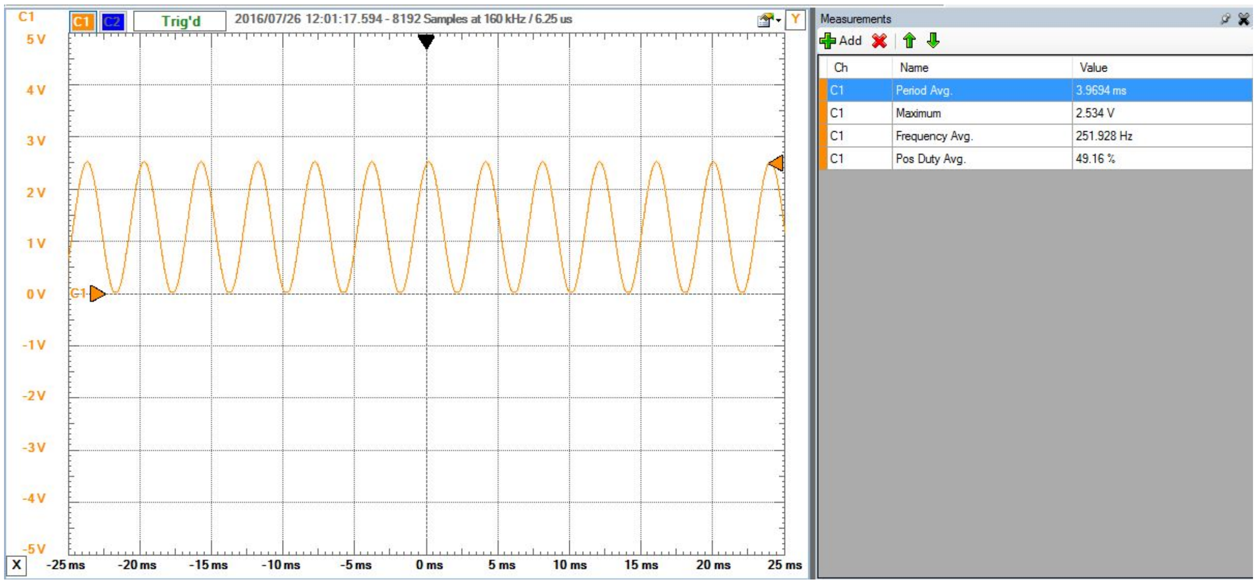
2016/07/26 12:02:08.500 - 8192 Samples at 160 kHz / 6.25 us

Measurements

Add

Ch	Name	Value
C1	Period Avg.	3.96975 ms
C1	Maximum	2.534 V
C1	Frequency Avg.	251.904 Hz
C1	Pos Duty Avg.	49.15 %

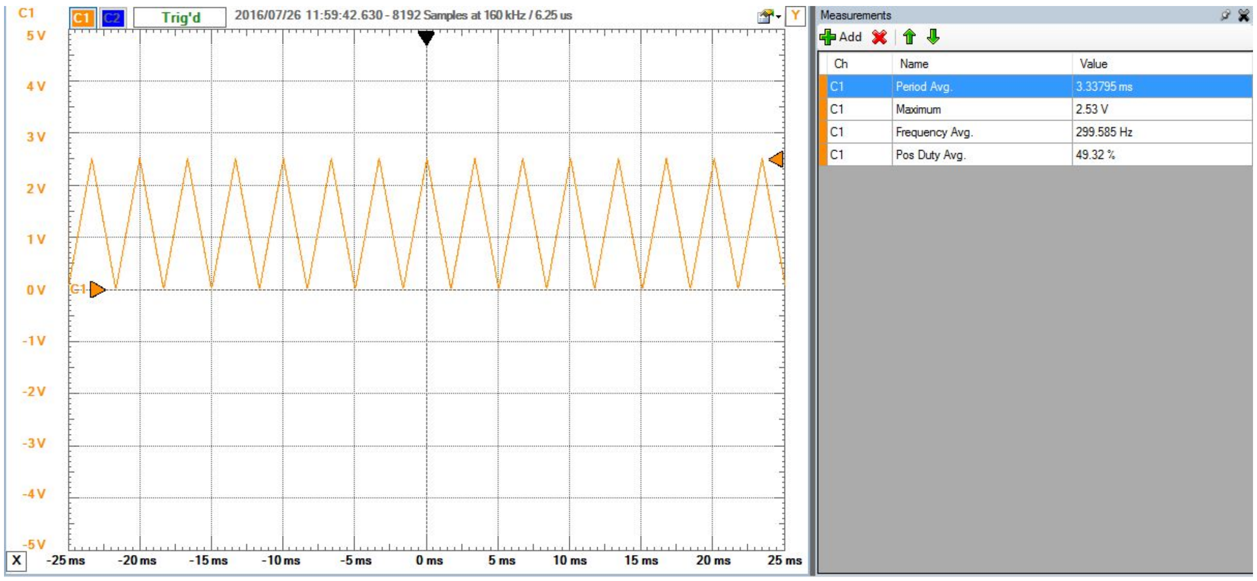
Part C - SineWave using DMA/TC at 250Hz.



2016/07/26 12:01:17.594 - 8192 Samples at 160 kHz / 6.25 us

Ch	Name	Value
C1	Period Avg.	3.9694 ms
C1	Maximum	2.534 V
C1	Frequency Avg.	251.928 Hz
C1	Pos Duty Avg.	49.16 %

Part D - TriangleWave at 300Hz.



2016/07/26 11:59:42.630 - 8192 Samples at 160 kHz / 6.25 us

Ch	Name	Value
C1	Period Avg.	3.33795 ms
C1	Maximum	2.53 V
C1	Frequency Avg.	299.585 Hz
C1	Pos Duty Avg.	49.32 %