Assignment 5 Deliverables

1) YouTube Demo of Triangle Wave:
   https://youtu.be/_77ImkgVqT4

2) Screenshots of Oscilloscope:
   Figures 1 and 2 display the screenshots for the square and triangle waves, respectively.

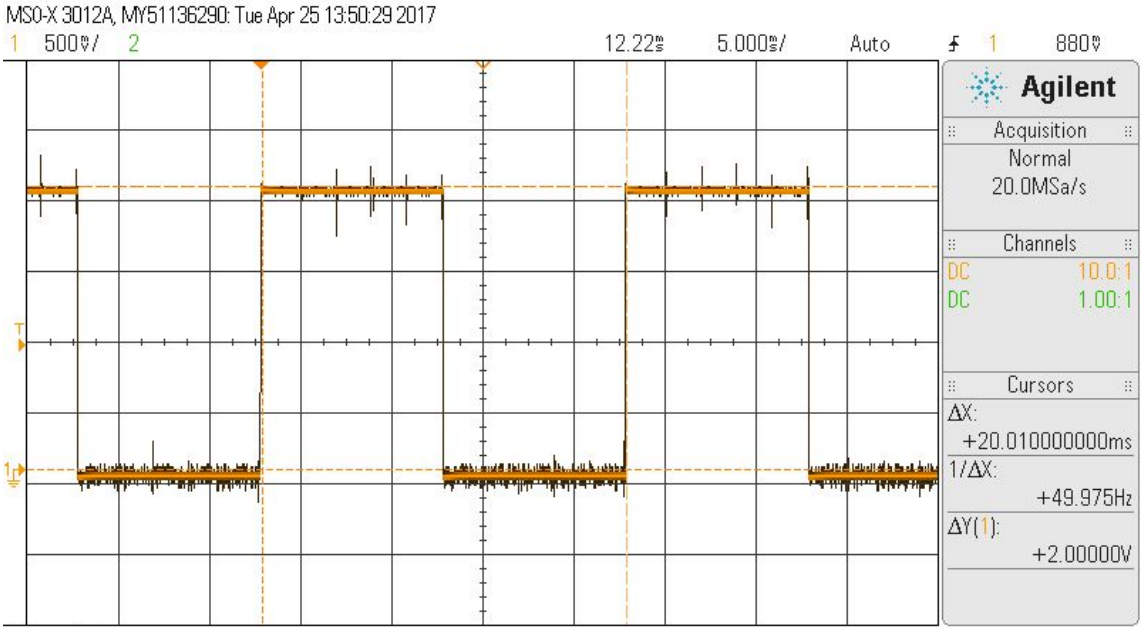

Figure 1: Square Wave -- 2VPP, 20ms period

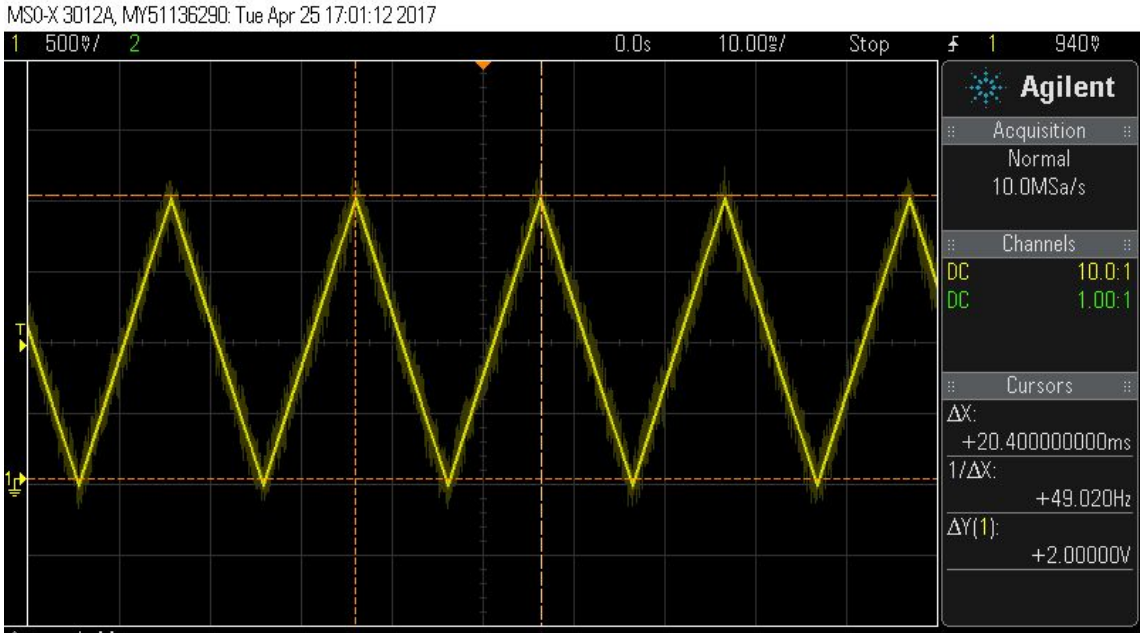

Figure 1: Triangle Wave -- 2VPP, 20ms period

Derek Nola & Bevin Tang
CPE329, Spring 2017
Professor: Paul Hummel

3) Our code:

Main.c

```
//*****************************************************************************
****
// Assignment 5
// Derek Nola and Bevin Tang
//
//*****************************************************************************
***

#include "msp.h"
#include "delay.h"
#include "timers.h"

void Drive_DAC(unsigned int level);

volatile unsigned int TempDAC_Value = 0;
volatile unsigned int delay = 0;

#define TRI_STEPS 64
int DAC_COUNT = 0;
int DAC_SHIFT = 32;

int main(void) {

  WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;   // Stop watchdog timer


  set_DC0(FREQ_24_MHz);

  // Configure port bits for SPI
  P4->DIR |= BIT1;                    // Will use BIT4 to activate /CE on
the DAC
  P1SEL0 |= BIT6 + BIT5;              // Configure P1.6 and P1.5 for
UCB0SIMO and UCB0CLK
  P1SEL1 &= ~(BIT6 + BIT5);           //

  // SPI Setup
  EUSCI_B0->CTLW0 |= EUSCI_B_CTLW0_SWRST;   // Put eUSCI state machine in
reset
```

```
  EUSCI_B0->CTLW0 = EUSCI_B_CTLW0_SWRST |   // Remain eUSCI state machine
in reset
                    EUSCI_B_CTLW0_MST   |   // Set as SPI master
                    EUSCI_B_CTLW0_SYNC  |   // Set as synchronous mode
                    EUSCI_B_CTLW0_CKPL  |   // Set clock polarity high
                    EUSCI_B_CTLW0_MSB;      // MSB first

  EUSCI_B0->CTLW0 |= EUSCI_B_CTLW0_SSEL__SMCLK; // SMCLK
  EUSCI_B0->BRW = 0x01;                         // divide by 16, clock =
fBRCLK/(UCBRx)
  EUSCI_B0->CTLW0 &= ~EUSCI_B_CTLW0_SWRST;      // Initialize USCI state
machine, SPI
                                                // now waiting for
something to
                                                // be placed in TXBUF

  EUSCI_B0->IFG |= EUSCI_B_IFG_TXIFG;  // Clear TXIFG flag

 //Setup timer and interrupts
 delay =  0xDB23; //5ms base

 TIMER_A0->CCTL[0] = TIMER_A_CCTLN_CCIE; // TACCR0 interrupt enabled
 TIMER_A0->CCR[0] = delay;

 TIMER_A0->CTL = TIMER_A_CTL_SSEL__SMCLK | // SMCLK, continuous mode
         TIMER_A_CTL_MC__CONTINUOUS;

 SCB->SCR |= SCB_SCR_SLEEPONEXIT_Msk;    // Enable sleep on exit from ISR

     // Enable global interrupt
 __enable_irq();
 NVIC->ISER[0] = 1 << ((TA0_0_IRQn) & 31);

 while (1){
     __sleep();
     __no_operation();                   // For debugger
 }

} // end of main
```

Derek Nola & Bevin Tang
CPE329, Spring 2017
Professor: Paul Hummel

```
//Square Timer A0 interrupt service routine

void TA0_0_IRQHandler(void) {
 static uint8_t DAC_STATE = 0;
 static uint8_t Delay_Loop = 0;

 TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG;
  if(Delay_Loop < 4){
      TIMER_A0->CCR[0] += delay;
      Delay_Loop++;
  }
  else if(0 == DAC_STATE){
      DAC_STATE = 1;
      TempDAC_Value = 2048; //Set output to 2V
      Delay_Loop = 0;
  }
  else if(1 == DAC_STATE){
      DAC_STATE = 0;
      TempDAC_Value = 0; //Set output to 0V
      Delay_Loop = 0;
  }
  Drive_DAC(TempDAC_Value);
  TIMER_A0->CCR[0] += delay;
}

//Triangle Timer A0 interrupt service routine
void TA0_0_IRQHandler(void) {
    TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG;

    if (DAC_COUNT >= TRI_STEPS) {
        DAC_SHIFT *= -1; //invert shift once count hits 2V limit
        DAC_COUNT = 0;
    }
    Drive_DAC(TempDAC_Value);
    TempDAC_Value += DAC_SHIFT; //change voltage
    TIMER_A0->CCR[0] += 0x0F00; //reset timer
    DAC_COUNT++;
}

void Drive_DAC(unsigned int level){
  unsigned int DAC_Word = 0;
```

Derek Nola & Bevin Tang
CPE329, Spring 2017
Professor: Paul Hummel

```
 int i;
  DAC_Word = (0x1000) | (2*level/5 & 0x0FFF);   // 0x1000 sets DAC for
Write
                                              // to DAC, Gain = 2, /SHDN = 1
                                              // and put 12-bit level value
                                              // in low 12 bits.

  P4->OUT &= ~BIT1;                                    // Clear P4.1 (drive
/CS low on DAC)
                                                       // Using a port
output to do this for now

  EUSCI_B0->TXBUF = (unsigned char) (DAC_Word >> 8);  // Shift upper byte
of DAC_Word
                                                       // 8-bits to right

  while (!(EUSCI_B0->IFG & EUSCI_B_IFG_TXIFG));              // USCI_A0 TX
buffer ready?
     EUSCI_B0->TXBUF = (unsigned char) (DAC_Word & 0x00FF);  // Transmit
lower byte to DAC

  while (!(EUSCI_B0->IFG & EUSCI_B_IFG_TXIFG));      // Poll the TX flag to
wait for completion

  for(i = 200; i > 0; i--);                                  // Delay
200 16 MHz SMCLK periods
                                                     //to ensure TX is
complete by SIMO

  P4->OUT |= BIT1;                                   // Set P4.1   (drive
/CS high on DAC)

  return;
}
```