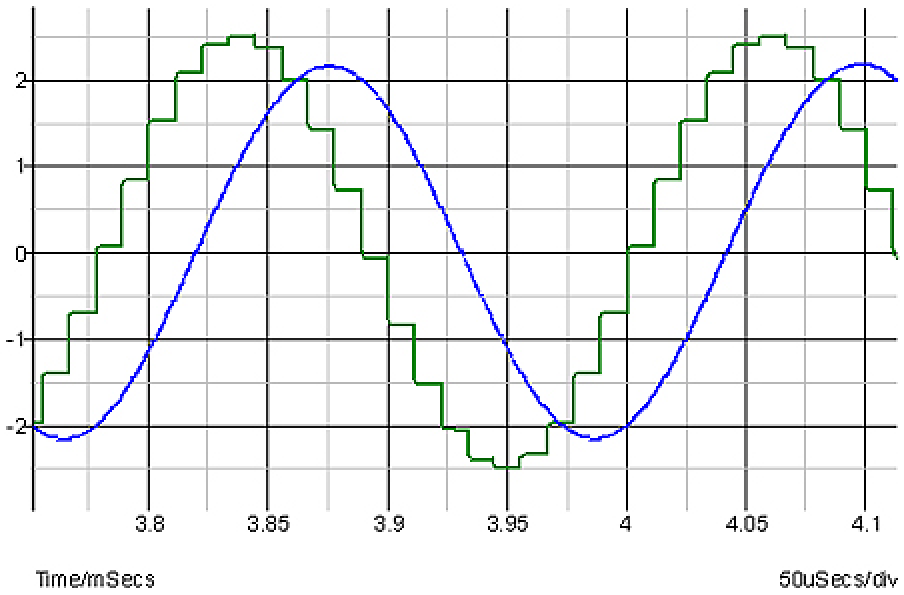
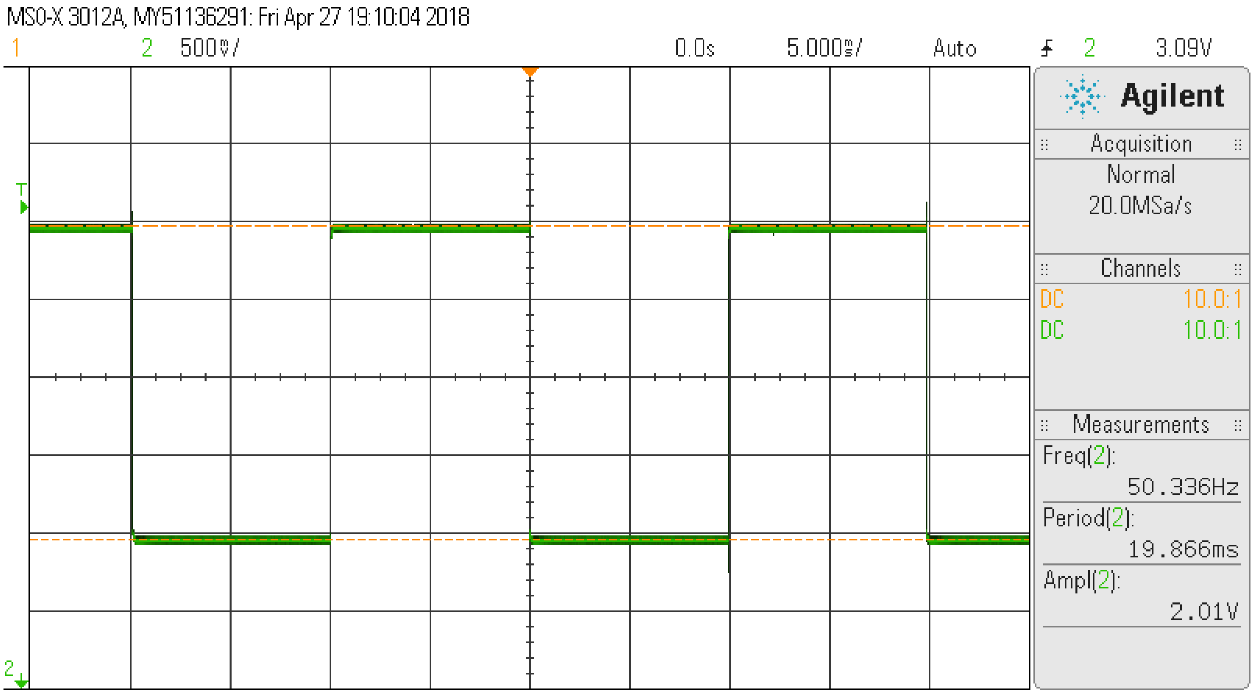


A6 - Assignment DAC Waveform Generation

Students: Zach Bunce & Garrett Maxon
Class: CPE 329-03
Professor: Gerfen, Jeffrey

Link to YouTube demo of triangle tave running on oscilloscope.

https://youtu.be/E8oNNBWtFtk



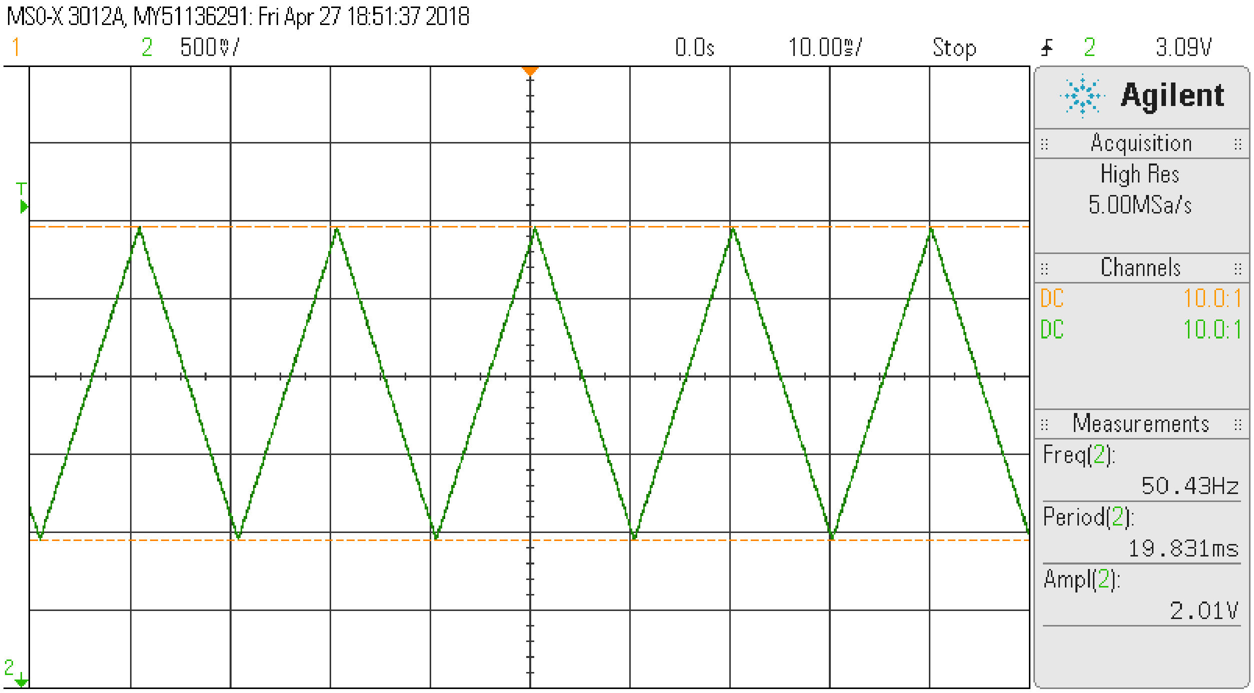
```
MSO-X 3012A, MY51136291: Fri Apr 27 19:10:04 2018
1        2   500mV/          0.0s     5.000ms/     Auto     ⨍  2    3.09V

Acquisition
Normal
20.0MSa/s

Channels
DC        10.0:1
DC        10.0:1

Measurements
Freq(2):
      50.336Hz
Period(2):
      19.866ms
Ampl(2):
      2.01V
```

Figure 1: 2 V_{pp}, 1 V DC Offset, 50 Hz Square Wave



```
MSO-X 3012A, MY51136291: Fri Apr 27 18:51:37 2018
1        2   500mV/          0.0s     10.00ms/     Stop     ⨍  2    3.09V

Acquisition
High Res
5.00MSa/s

Channels
DC        10.0:1
DC        10.0:1

Measurements
Freq(2):
      50.43Hz
Period(2):
      19.831ms
Ampl(2):
      2.01V
```

Figure 2: 2 V_{pp}, 1 V DC Offset, 50 Hz Triangle Wave

Main C Code

```
/**
 * main.c
 *
 * Sets up SPI and generates specified wave
 *
 * Date: April 25 2018
 * Authors: Zach Bunce, Garrett Maxon
 */

//TIE LDAC LOW

//P1.5 SCLK
//P1.6 MOSI
//P1.7 MISO

#include "msp.h"
#include <stdint.h>
#include "set_DCO.h"
#include "delays.h"
#include "SPI.h"
#include "DAC.h"

int main(void) {
    int CLK = 120;
    set_DCO(CLK);

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

    SPI_INIT();                                 //Initializes SPI comms

    while(1) {
        makeWave(square, 2, 1, 50, CLK);        //Generates the specified wave
    }
}
```

SPI Library

```
/**
 * SPI.h
 *
 * Header for SPI communication
 *
 * Date: April 25 2018
 * Authors: Zach Bunce, Garrett Maxon
 */
#ifndef SPI_H_
#define SPI_H_
void SPI_INIT();
void sendByte_SPI(uint8_t);
#endif /* SPI_H_ */
```

```
/**
 * SPI.c
 *
 * Contains functions for both SPI setup and transmission
 * P1.5 SCLK, P1.6 MOSI, P1.7 MISO
 *
 * Date: April 25 2018
 * Authors: Zach Bunce, Garrett Maxon
 */
#include "msp.h"
#include <stdint.h>
#include "SPI.h"

void SPI_INIT() {
    P5 -> DIR  |= BIT5;                         //CS Output
    P5 -> OUT  |= BIT5;                         //Initializes CS high
    P1 -> SEL0 |= (BIT5 | BIT6 | BIT7);         //SPI Line pinout setup
    P1 -> SEL1 &= ~(BIT5 | BIT6);
    EUSCI_B0 -> CTLW0 |= EUSCI_B_CTLW0_SWRST;   //Reset
    EUSCI_B0 -> CTLW0  = EUSCI_B_CTLW0_SWRST |
                         EUSCI_B_CTLW0_MST   |
                         EUSCI_B_CTLW0_SYNC  |
                         EUSCI_B_CTLW0_CKPL  |
                         EUSCI_B_CTLW0_MSB;
    EUSCI_B0 -> CTLW0 |= EUSCI_B_CTLW0_SSEL__SMCLK; //SMCLK
    EUSCI_B0 -> BRW = 0x01;                     //BRW = 1; SDICLK = SMCLK
    EUSCI_B0 -> CTLW0 &= ~EUSCI_B_CTLW0_SWRST;  //Start FSM
    //SPI ready to go as soon as TX data dropped in buffer
    EUSCI_B0 -> IFG |= EUSCI_B_IFG_TXIFG;
}

void sendByte_SPI(uint8_t data) {
    EUSCI_B0 -> TXBUF = data;                       //Drops data into buffer
    while (!(EUSCI_B0->IFG & EUSCI_B_IFG_TXIFG));   //Waits for TX flag to go low
}
```

DAC Library

```
/**
 * DAC.h
 *
 * Header for running the DAC
 *
 * Date: April 25 2018
 * Authors: Zach Bunce, Garrett Maxon
 */

#ifndef DAC_H_
#define DAC_H_

#define square      1       //Wave type definitions
#define triangle    2
#define sine        3
#define Vref        33      //Reference voltage * 10
#define SDOFF       BIT4    //Shutdown mode OFF
#define GAIN1       BIT5    //Unity gain mode
#define HIGH        0xFF    //Definitions for square wave states
#define LOW         0x00

void write_DAC(uint16_t);
void makeDC(int);
void makeWave(int, int, int, int, int);

#endif /* DAC_H_ */
```

```
/**
 * DAC.c
 *
 * Contains functions for running the DAC
 * Allows for DC voltages and various wave types to be automatically produced
 *
 * Date: April 25 2018
 * Authors: Zach Bunce, Garrett Maxon
 */

#include "msp.h"
#include <stdint.h>
#include "SPI.h"
#include "DAC.h"

static int z = 0;       //Timer division variable
static int waveType;    //Wave property variables for ISR
static int Vpp;
static int Voff;
static int freq;
static int CLK;

static uint8_t sqw_ST;  //Square wave state variable for ISR

static int UD;          //Triangle wave ISR variables
static int16_t DN_Point;
static int incDiv;

void makeDC(int volt)
{
    uint16_t DN = (((4096*volt)/(Vref))*10);    //Maps output voltage to 12-bit DAC value
    write_DAC(DN);                              //Writes value to the DAC
}

void makeWave(int waveT, int pp, int offset, int frequency, int clock)
{
    waveType = waveT;   //Links outside specified parameters to ISR globals
    Vpp = pp;
    Voff = offset;
    freq = frequency;
    CLK = clock;

    TIMER_A0->CCTL[0] = TIMER_A_CCTLN_CCIE;     //Enables TACCR0 interrupt
    //Runs Timer A on SMCLK and in continuous mode
    TIMER_A0->CTL = TIMER_A_CTL_SSEL__SMCLK | TIMER_A_CTL_MC__CONTINUOUS;
    SCB->SCR |= SCB_SCR_SLEEPONEXIT_Msk;        //Enables sleep on exit from ISR

    __enable_irq();                             //Enables global interrupts
    NVIC->ISER[0] = 1 << ((TA0_0_IRQn) & 31);   //Links ISR to NVIC
    int top = Vpp + Voff;                       //Calculates peak voltage
    int bot = Voff;                             //Calculates trough voltage

    if (waveType == square)                     //Square wave
    {
        TIMER_A0->CCR[0] = 60000;               //Initializes first high time count
        while(1);                               //Allows interrupt to control generation
    }
    else if (waveType == triangle)              //Triangle wave
    {
        DN_Point = (((4096*Voff)/(Vref))*10);   //Sets the initial DAC input to the trough
        int16_t DN_Top     = (((4096*top)/(Vref))*10);  //Finds the DAC input for the peak
        int16_t DN_Bottom  = (((4096*bot)/(Vref))*10);  //Finds the DAC input for the trough
        //Finds the counter amount needed
        incDiv = ((CLK * 1000000) / (2 * (DN_Top - DN_Bottom) * freq));
        TIMER_A0->CCR[0] = incDiv;              //Initializes first increment count
        while(1);                               //Allows interrupt to control generation
    }
}

void write_DAC(uint16_t data)
{
    uint8_t up_Byte;
    uint8_t low_Byte;

    up_Byte  =  ((data & 0x0F00) >> 8); //Masks top 4 data bits & shifts into lower nibble
    up_Byte &= 0x0F;                    //Redundantly masks the data bits after shifting
    up_Byte |= (GAIN1 | SDOFF);         //Appends control bits onto upper nibble
    low_Byte =   (data & 0x00FF);       //Masks bottom 8 data bits

    P5 -> OUT &= ~BIT5;                 //Lowers chip select
    sendByte_SPI(up_Byte);              //Transmits upper byte on SPI line
    sendByte_SPI(low_Byte);             //Transmits lower byte on SPI line
    P5 -> OUT |= BIT5;                  //Sets chip select
}
```

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

    if (waveType == square) {
        int delCyc  = (CLK * 100000) / freq;    //Calculates frequency division needed
        int timeDiv = delCyc / (60000*2);       //Maps division to usable clock increments

        if (z == timeDiv) {
            sqw_ST = ~sqw_ST;                   //Inverts square wave level
            if (sqw_ST == HIGH) {
                makeDC((Vpp + Voff));           //Sets output voltage to high value
            }
            else if (sqw_ST == LOW) {
                makeDC(Voff);                   //Sets output voltage to low value
            }
            z = 0;                              //Clears ISR entry counter
        }
        z++;                                    //Increments ISR entry counter
        TIMER_A0->CCR[0] += 60000;              //Adds next offset to TACCR0
    }
    else if (waveType == triangle) {
        int16_t DN_Top     = (((4096*(Vpp + Voff))/(Vref))*10); //Finds DAC input for the peak
        int16_t DN_Bottom  = (((4096*Voff)/(Vref))*10); //Finds the DAC input for the trough
        if (DN_Point >= DN_Top) {
            UD = -10;               //Sets the wave to decrement
            DN_Point = DN_Top;      //Ensures output is at peak value
        }
        else if (DN_Point <= DN_Bottom) {
            UD = 10;                //Sets the wave to increment
            DN_Point = DN_Bottom;   //Ensures output is at trough value
        }
        write_DAC(DN_Point);        //Sends the DAC value to the DAC
        DN_Point += UD;             //Calculates increment or decrement for DAC value
        TIMER_A0->CCR[0] += incDiv; //Adds 5ms offset to TACCR0
    }
}
```