Analog-to-Digital and Digital-to-Analog Conversion
8th Laboratory Report for ECE 383 Microcomputers

Submitted by
Patrick Brooks
11650957
Lauren Brackin
1186460

The University of Alabama
Tuscaloosa, AL 35487

4/23/21

Deliverables:

Deliverable 1:

Patrick Brooks : 11650957

$V_{RA} = 3.3v \frac{2.33}{2.33+910.52} = 3.3v \frac{2.33}{912.33}$

$= .0084127$

Deliverable 2:

https://www.youtube.com/playlist?list=PLLXkv2lvJPoMbN19zv9Gmnz35RzuGJmz9

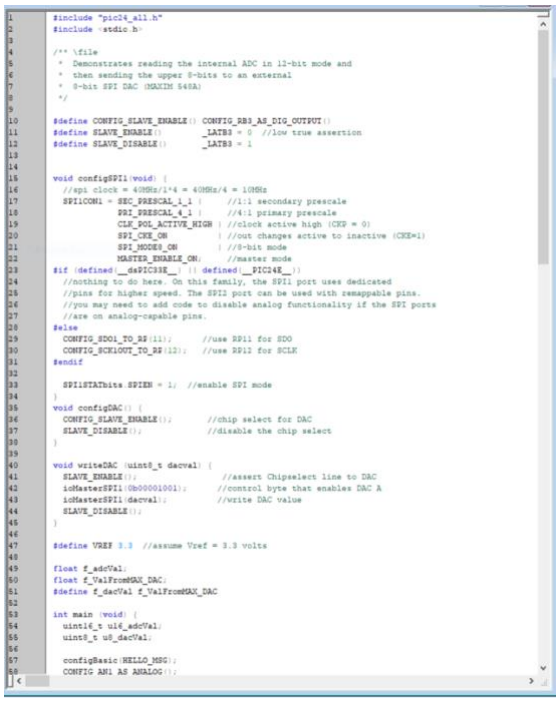
Deliverable 3:

```
#include "pic24_all.h"
#include <stdio.h>

/** \file
 *  Demonstrates reading the internal ADC in 12-bit mode and
 *  then sending the upper 8-bits to an external
 *  8-bit SPI DAC (MAXIM 548A)
 */

#define CONFIG_SLAVE_ENABLE()  CONFIG_RB3_AS_DIG_OUTPUT()
#define SLAVE_ENABLE()         _LATB3 = 0  //low true assertion
#define SLAVE_DISABLE()        _LATB3 = 1


void configSPI1(void) {
  //spi clock = 40MHz/1*4 = 40MHz/4 = 10MHz
  SPI1CON1 = SEC_PRESCAL_1_1 |      //1:1 secondary prescale
             PRI_PRESCAL_4_1 |      //4:1 primary prescale
             CLK_POL_ACTIVE_HIGH |  //clock active high (CKP = 0)
             SPI_CKE_ON          |  //out changes active to inactive (CKE=1)
             SPI_MODE8_ON        |  //8-bit mode
             MASTER_ENABLE_ON;      //master mode
#if (defined(__dsPIC33E__) || defined(__PIC24E__))
  //nothing to do here. On this family, the SPI1 port uses dedicated
  //pins for higher speed. The SPI2 port can be used with remappable pins.
  //you may need to add code to disable analog functionality if the SPI ports
  //are on analog-capable pins.
#else
  CONFIG_SDO1_TO_RP(11);      //use RP11 for SDO
  CONFIG_SCK1OUT_TO_RP(12);   //use RP12 for SCLK
#endif

  SPI1STATbits.SPIEN = 1;  //enable SPI mode
}
void configDAC() {
  CONFIG_SLAVE_ENABLE();       //chip select for DAC
  SLAVE_DISABLE();             //disable the chip select
}

void writeDAC (uint8_t dacval) {
  SLAVE_ENABLE();              //assert Chipselect line to DAC
  ioMasterSPI1(0b00001001);    //control byte that enables DAC A
  ioMasterSPI1(dacval);        //write DAC value
  SLAVE_DISABLE();
}

#define VREF 3.3  //assume Vref = 3.3 volts

float f_adcVal;

int main (void) {
  uint16_t ui6_adcVal;
  uint8_t u8_dacVal;
  float f_dacVal;

  configBasic(HELLO_MSG);
  CONFIG_AN1_AS_ANALOG();
  CONFIG_AN10_AS_ANALOG();
    CONFIG_RB14_AS_DIG_OUTPUT();
```

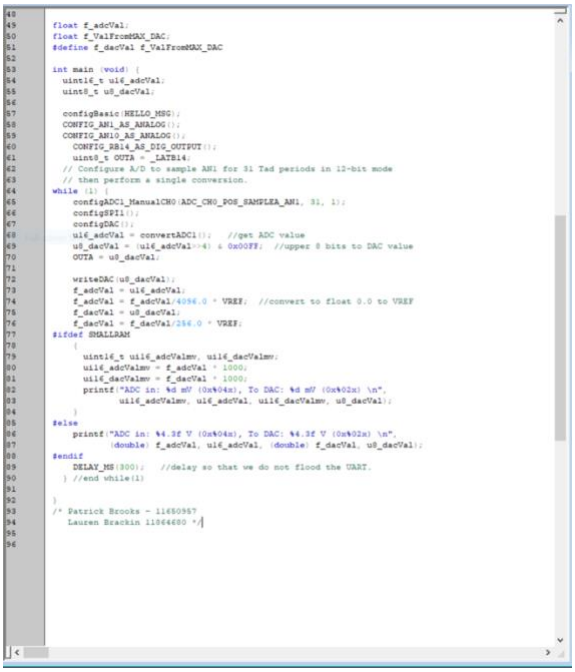
```
#define VREF 3.3  //assume Vref = 3.3 volts

float f_adcVal;

int main (void) {
  uint16_t ui6_adcVal;
  uint8_t u8_dacVal;
  float f_dacVal;

  configBasic(HELLO_MSG);
  CONFIG_AN1_AS_ANALOG();
  CONFIG_AN10_AS_ANALOG();
    CONFIG_RB14_AS_DIG_OUTPUT();
    uint8_t OUTA = _LATB14;
  // Configure A/D to sample AN1 for 31 Tad periods in 12-bit mode
  // then perform a single conversion.
while (1) {
    configADC1_ManualCH0(ADC_CH0_POS_SAMPLEA_AN1, 31, 1);
    configSPI1();
    configDAC();
    ui6_adcVal = convertADC1();   //get ADC value
    u8_dacVal = (ui6_adcVal>>4) & 0x00FF;  //upper 8 bits to DAC value
    OUTA = u8_dacVal;

    writeDAC(u8_dacVal);
    f_adcVal = ui6_adcVal;
    f_adcVal = f_adcVal/4096.0 * VREF;  //convert to float 0.0 to VREF
    f_dacVal = u8_dacVal;
    f_dacVal = f_dacVal/256.0 * VREF;
#ifdef SMALLRAM
    {
      uint16_t ui6_adcValmv, ui16_dacValmv;
      ui16_adcValmv = f_adcVal * 1000;
      ui16_dacValmv = f_dacVal * 1000;
      printf("ADC in: %d mV (0x%04x), To DAC: %d mV (0x%02x) \n",
             ui16_adcValmv, ui6_adcVal, ui16_dacValmv, u8_dacVal);
    }
#else
    printf("ADC in: %4.3f V (0x%04x), To DAC: %4.3f V (0x%02x) \n",
           (double) f_adcVal, ui6_adcVal, (double) f_dacVal, u8_dacVal);
#endif
    DELAY_MS(300);   //delay so that we do not flood the UART.
  } //end while(1)

}
/* Patrick Brooks - 11650957
   Lauren Brackin 11864680 */
```

Deliverable 4:
https://www.youtube.com/playlist?list=PLLXkv2lvJPoMbN19zv9Gmnz35RzuGJmz9

Deliverable 5:



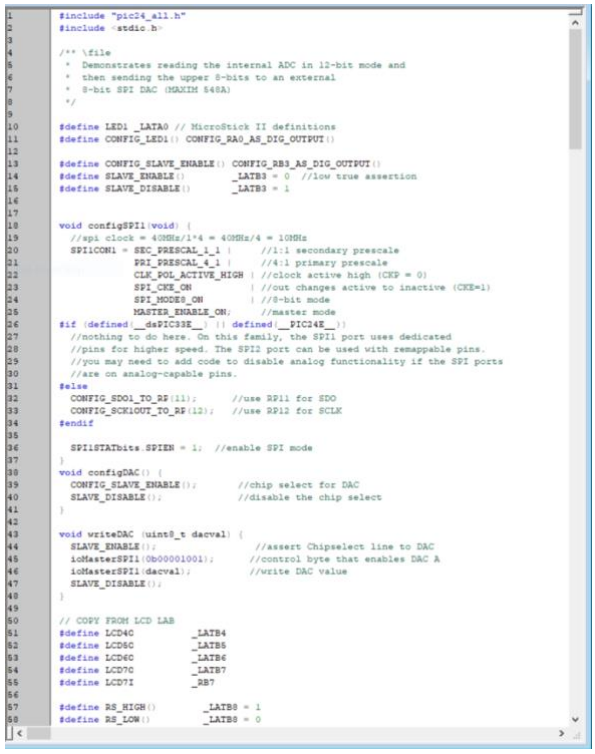
```
#include "pic24_all.h"
#include <stdio.h>

/** \file
 *  Demonstrates reading the internal ADC in 12-bit mode and
 *  then sending the upper 8-bits to an external
 *  8-bit SPI DAC (MAXIM 548A)
 */

#define CONFIG_SLAVE_ENABLE()  CONFIG_RB3_AS_DIG_OUTPUT()
#define SLAVE_ENABLE()         _LATB3 = 0  //low true assertion
#define SLAVE_DISABLE()        _LATB3 = 1


void configSPI1(void) {
  //spi clock = 40MHz/1*4 = 40MHz/4 = 10MHz
  SPI1CON1 = SEC_PRESCAL_1_1 |      //1:1 secondary prescale
             PRI_PRESCAL_4_1 |      //4:1 primary prescale
             CLK_POL_ACTIVE_HIGH |  //clock active high (CKP = 0)
             SPI_CKE_ON          |  //out changes active to inactive (CKE=1)
             SPI_MODE8_ON        |  //8-bit mode
             MASTER_ENABLE_ON;      //master mode
#if (defined(__dsPIC33E__) || defined(__PIC24E__))
  //nothing to do here. On this family, the SPI1 port uses dedicated
  //pins for higher speed. The SPI2 port can be used with remappable pins.
  //you may need to add code to disable analog functionality if the SPI ports
  //are on analog-capable pins.
#else
  CONFIG_SDO1_TO_RP(11);      //use RP11 for SDO
  CONFIG_SCK1OUT_TO_RP(12);   //use RP12 for SCLK
#endif

  SPI1STATbits.SPIEN = 1;  //enable SPI mode
}
void configDAC() {
  CONFIG_SLAVE_ENABLE();       //chip select for DAC
  SLAVE_DISABLE();             //disable the chip select
}

void writeDAC (uint8_t dacval) {
  SLAVE_ENABLE();              //assert Chipselect line to DAC
  ioMasterSPI1(0b00001001);    //control byte that enables DAC A
  ioMasterSPI1(dacval);        //write DAC value
  SLAVE_DISABLE();
}

#define VREF 3.3  //assume Vref = 3.3 volts

float f_adcVal;
float f_ValFromMAX_DAC;
#define f_dacVal f_ValFromMAX_DAC

int main (void) {
  uint16_t ui6_adcVal;
  uint8_t u8_dacVal;

  configBasic(HELLO_MSG);
  CONFIG_AN1_AS_ANALOG();
```



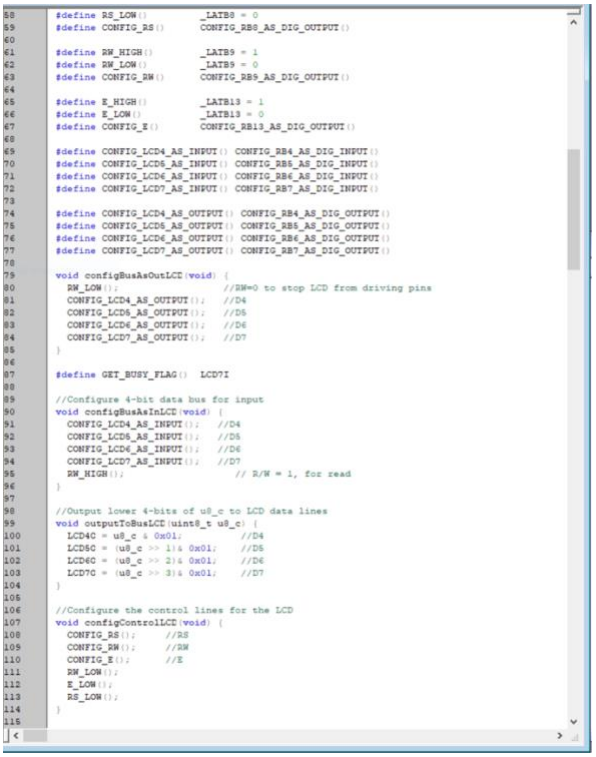
```
float f_adcVal;
float f_ValFromMAX_DAC;
#define f_dacVal f_ValFromMAX_DAC

int main (void) {
  uint16_t ui6_adcVal;
  uint8_t u8_dacVal;

  configBasic(HELLO_MSG);
  CONFIG_AN1_AS_ANALOG();
  CONFIG_AN10_AS_ANALOG();
    CONFIG_RB14_AS_DIG_OUTPUT();
    uint8_t OUTA = _LATB14;
  // Configure A/D to sample AN1 for 31 Tad periods in 12-bit mode
  // then perform a single conversion.
while (1) {
    configADC1_ManualCH0(ADC_CH0_POS_SAMPLEA_AN1, 31, 1);
    configSPI1();
    configDAC();
    ui6_adcVal = convertADC1();   //get ADC value
    u8_dacVal = (ui6_adcVal>>4) & 0x00FF;  //upper 8 bits to DAC value
    OUTA = u8_dacVal;

    writeDAC(u8_dacVal);
    f_adcVal = ui6_adcVal;
    f_adcVal = f_adcVal/4096.0 * VREF;  //convert to float 0.0 to VREF
    f_dacVal = u8_dacVal;
    f_dacVal = f_dacVal/256.0 * VREF;
#ifdef SMALLRAM
      {
        uint16_t ui6_adcValmv, ui16_dacValmv;
        ui16_adcValmv = f_adcVal * 1000;
        ui16_dacValmv = f_dacVal * 1000;
        printf("ADC in: %d mV (0x%04x), To DAC: %d mV (0x%02x) \n",
               ui16_adcValmv, ui6_adcVal, ui16_dacValmv, u8_dacVal);
      }
#else
    printf("ADC in: %4.3f V (0x%04x), To DAC: %4.3f V (0x%02x) \n",
           (double) f_adcVal, ui6_adcVal, (double) f_dacVal, u8_dacVal);
#endif
    DELAY_MS(300);   //delay so that we do not flood the UART.
  } //end while(1)

}
/* Patrick Brooks - 11650957
   Lauren Brackin 11964680 */
```

Deliverable 6:
https://www.youtube.com/playlist?list=PLLXkv2lvJPoMbN19zv9Gmnz35RzuGJmz9

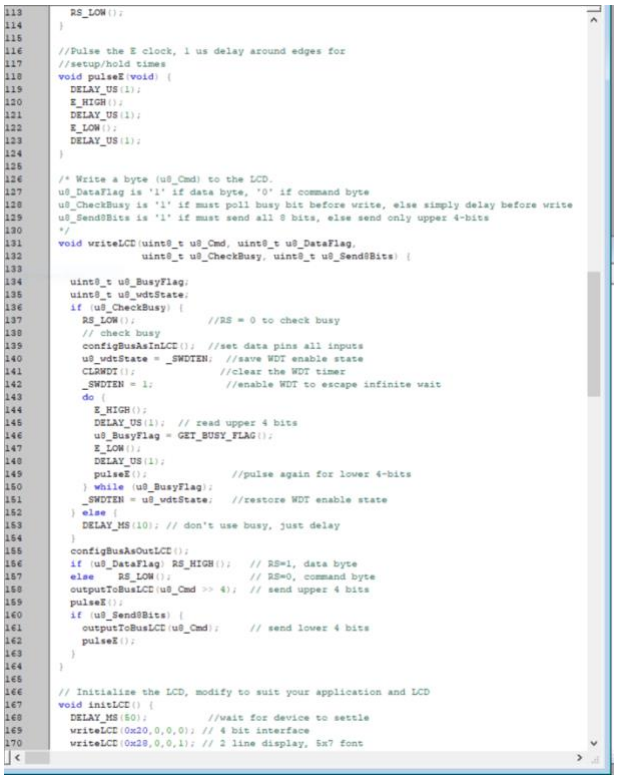
Deliverable 7:



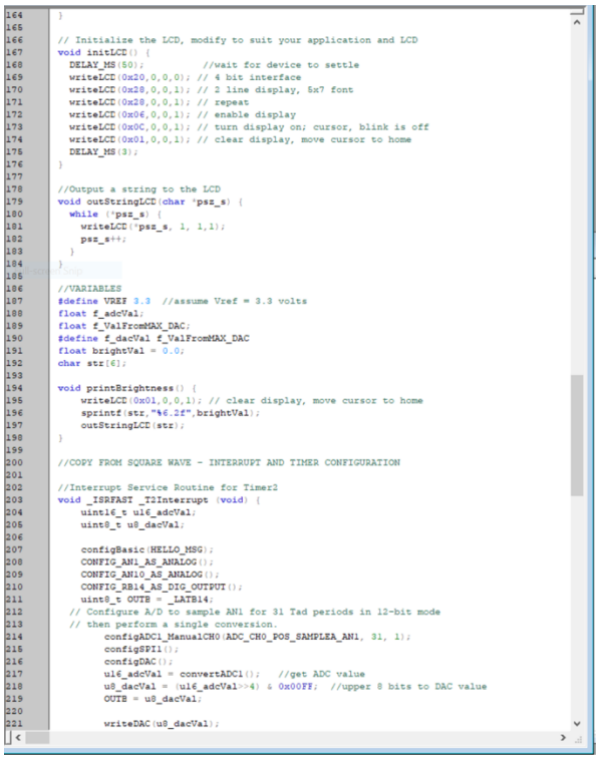
```
#include "pic24_all.h"
#include <stdio.h>

/** \file
 *  Demonstrates reading the internal ADC in 12-bit mode and
 *  then sending the upper 8-bits to an external
 *  8-bit SPI DAC (MAXIM 548A)
 */

#define LED1 _LATA0 // MicroStick II definitions
#define CONFIG_LED1() CONFIG_RA0_AS_DIG_OUTPUT()

#define CONFIG_SLAVE_ENABLE() CONFIG_RB3_AS_DIG_OUTPUT()
#define SLAVE_ENABLE()        _LATB3 = 0  //low true assertion
#define SLAVE_DISABLE()       _LATB3 = 1


void configSPI1(void) {
  //spi clock = 40MHz/1*4 = 40MHz/4 = 10MHz
  SPI1CON1 = SEC_PRESCAL_1_1 |      //1:1 secondary prescale
             PRI_PRESCAL_4_1 |      //4:1 primary prescale
             CLK_POL_ACTIVE_HIGH | //clock active high (CKP = 0)
             SPI_CKE_ON          | //out changes active to inactive (CKE=1)
             SPI_MODE8_ON        | //8-bit mode
             MASTER_ENABLE_ON;     //master mode
#if (defined(__dsPIC33E__) || defined(__PIC24E__))
  //nothing to do here. On this family, the SPI1 port uses dedicated
  //pins for higher speed. The SPI2 port can be used with remappable pins.
  //you may need to add code to disable analog functionality if the SPI ports
  //are on analog-capable pins.
#else
  CONFIG_SDO1_TO_RP(11);      //use RP11 for SDO
  CONFIG_SCK1OUT_TO_RP(12);   //use RP12 for SCLK
#endif

  SPI1STATbits.SPIEN = 1;  //enable SPI mode
}
void configDAC() {
  CONFIG_SLAVE_ENABLE();       //chip select for DAC
  SLAVE_DISABLE();             //disable the chip select
}

void writeDAC (uint8_t dacval) {
  SLAVE_ENABLE();              //assert Chipselect line to DAC
  ioMasterSPI1(0b00001001);    //control byte that enables DAC A
  ioMasterSPI1(dacval);        //write DAC value
  SLAVE_DISABLE();
}

// COPY FROM LCD LAB
#define LCD4C          _LATB4
#define LCD5C          _LATB5
#define LCD6C          _LATB6
#define LCD7C          _LATB7
#define LCD7I          _RB7

#define RS_HIGH()       _LATB8 = 1
#define RS_LOW()        _LATB8 = 0
```



```
#define RS_LOW()        _LATB8 = 0
#define CONFIG_RS()     CONFIG_RB8_AS_DIG_OUTPUT()

#define RW_HIGH()       _LATB9 = 1
#define RW_LOW()        _LATB9 = 0
#define CONFIG_RW()     CONFIG_RB9_AS_DIG_OUTPUT()

#define E_HIGH()        _LATB13 = 1
#define E_LOW()         _LATB13 = 0
#define CONFIG_E()      CONFIG_RB13_AS_DIG_OUTPUT()

#define CONFIG_LCD4_AS_INPUT() CONFIG_RB4_AS_DIG_INPUT()
#define CONFIG_LCD5_AS_INPUT() CONFIG_RB5_AS_DIG_INPUT()
#define CONFIG_LCD6_AS_INPUT() CONFIG_RB6_AS_DIG_INPUT()
#define CONFIG_LCD7_AS_INPUT() CONFIG_RB7_AS_DIG_INPUT()

#define CONFIG_LCD4_AS_OUTPUT() CONFIG_RB4_AS_DIG_OUTPUT()
#define CONFIG_LCD5_AS_OUTPUT() CONFIG_RB5_AS_DIG_OUTPUT()
#define CONFIG_LCD6_AS_OUTPUT() CONFIG_RB6_AS_DIG_OUTPUT()
#define CONFIG_LCD7_AS_OUTPUT() CONFIG_RB7_AS_DIG_OUTPUT()

void configBusAsOutLCD(void) {
  RW_LOW();                 //RW=0 to stop LCD from driving pins
  CONFIG_LCD4_AS_OUTPUT();  //D4
  CONFIG_LCD5_AS_OUTPUT();  //D5
  CONFIG_LCD6_AS_OUTPUT();  //D6
  CONFIG_LCD7_AS_OUTPUT();  //D7
}

#define GET_BUSY_FLAG()  LCD7I

//Configure 4-bit data bus for input
void configBusAsInLCD(void) {
  CONFIG_LCD4_AS_INPUT();   //D4
  CONFIG_LCD5_AS_INPUT();   //D5
  CONFIG_LCD6_AS_INPUT();   //D6
  CONFIG_LCD7_AS_INPUT();   //D7
  RW_HIGH();                  // R/W = 1, for read
}

//Output lower 4-bits of u8_c to LCD data lines
void outputToBusLCD(uint8_t u8_c) {
  LCD4C = u8_c & 0x01;          //D4
  LCD5C = (u8_c >> 1)& 0x01;    //D5
  LCD6C = (u8_c >> 2)& 0x01;    //D6
  LCD7C = (u8_c >> 3)& 0x01;    //D7
}

//Configure the control lines for the LCD
void configControlLCD(void) {
  CONFIG_RS();      //RS
  CONFIG_RW();      //RW
  CONFIG_E();       //E
  RW_LOW();
  E_LOW();
  RS_LOW();
}
```



```
  RS_LOW();
}

//Pulse the E clock, 1 us delay around edges for
//setup/hold times
void pulseE(void) {
  DELAY_US(1);
  E_HIGH();
  DELAY_US(1);
  E_LOW();
  DELAY_US(1);
}

/* Write a byte (u8_Cmd) to the LCD.
u8_DataFlag is '1' if data byte, '0' if command byte
u8_CheckBusy is '1' if must poll busy bit before write, else simply delay before write
u8_Send8Bits is '1' if must send all 8 bits, else send only upper 4-bits
*/
void writeLCD(uint8_t u8_Cmd, uint8_t u8_DataFlag,
              uint8_t u8_CheckBusy, uint8_t u8_Send8Bits) {

  uint8_t u8_BusyFlag;
  uint8_t u8_wdtState;
  if (u8_CheckBusy) {
    RS_LOW();               //RS = 0 to check busy
    // check busy
    configBusAsInLCD();  //set data pins all inputs
    u8_wdtState = _SWDTEN;  //save WDT enable state
    CLRWDT();               //clear the WDT timer
    _SWDTEN = 1;             //enable WDT to escape infinite wait
    do {
      E_HIGH();
      DELAY_US(1);  // read upper 4 bits
      u8_BusyFlag = GET_BUSY_FLAG();
      E_LOW();
      DELAY_US(1);
      pulseE();                 //pulse again for lower 4-bits
    } while (u8_BusyFlag);
    _SWDTEN = u8_wdtState;   //restore WDT enable state
  } else {
    DELAY_MS(10); // don't use busy, just delay
  }
  configBusAsOutLCD();
  if (u8_DataFlag) RS_HIGH();   // RS=1, data byte
  else     RS_LOW();            // RS=0, command byte
  outputToBusLCD(u8_Cmd >> 4);  // send upper 4 bits
  pulseE();
  if (u8_Send8Bits) {
    outputToBusLCD(u8_Cmd);     // send lower 4 bits
    pulseE();
  }
}

// Initialize the LCD, modify to suit your application and LCD
void initLCD() {
  DELAY_MS(50);          //wait for device to settle
  writeLCD(0x20,0,0,0); // 4 bit interface
  writeLCD(0x28,0,0,1); // 2 line display, 5x7 font
```



```
}

// Initialize the LCD, modify to suit your application and LCD
void initLCD() {
  DELAY_MS(50);          //wait for device to settle
  writeLCD(0x20,0,0,0); // 4 bit interface
  writeLCD(0x28,0,0,1); // 2 line display, 5x7 font
  writeLCD(0x28,0,0,1); // repeat
  writeLCD(0x06,0,0,1); // enable display
  writeLCD(0x0C,0,0,1); // turn display on; cursor, blink is off
  writeLCD(0x01,0,0,1); // clear display, move cursor to home
  DELAY_MS(3);
}

//Output a string to the LCD
void outStringLCD(char *psz_s) {
  while (*psz_s) {
    writeLCD(*psz_s, 1, 1,1);
    psz_s++;
  }
}

//VARIABLES
#define VREF 3.3  //assume Vref = 3.3 volts
float f_adcVal;
float f_ValFromMAX_DAC;
#define f_dacVal f_ValFromMAX_DAC
float brightVal = 0.0;
char str[6];

void printBrightness() {
    writeLCD(0x01,0,0,1); // clear display, move cursor to home
    sprintf(str,"%6.2f",brightVal);
    outStringLCD(str);
}

//COPY FROM SQUARE WAVE - INTERRUPT AND TIMER CONFIGURATION

//Interrupt Service Routine for Timer2
void _ISRFAST _T2Interrupt (void) {
    uint16_t u16_adcVal;
    uint8_t u8_dacVal;

    configBasic(HELLO_MSG);
    CONFIG_AN1_AS_ANALOG();
    CONFIG_AN10_AS_ANALOG();
    CONFIG_RB14_AS_DIG_OUTPUT();
    uint8_t OUTB = _LATB14;
  // Configure A/D to sample AN1 for 31 Tad periods in 12-bit mode
  // then perform a single conversion.
        configADC1_ManualCH0(ADC_CH0_POS_SAMPLEA_AN1, 31, 1);
        configSPI1();
        configDAC();
        u16_adcVal = convertADC1();   //get ADC value
        u8_dacVal = (u16_adcVal>>4) & 0x00FF;  //upper 8 bits to DAC value
        OUTB = u8_dacVal;

        writeDAC(u8_dacVal);
```

```
        configDAC();
        u16_adcVal = convertADC1();   //get ADC value
        u8_dacVal = (u16_adcVal>>4) & 0x00FF;  //upper 8 bits to DAC value
        OUTB = u8_dacVal;

        writeDAC(u8_dacVal);
        f_adcVal = u16_adcVal;
        f_adcVal = f_adcVal/4096.0 * VREF;  //convert to float 0.0 to VREF
        f_dacVal = u8_dacVal;
        f_dacVal = f_dacVal/256.0 * VREF;

    //CONVERT DAC FLOAT TO 0-255 RANGE
    brightVal = 255 - (255 * (f_dacVal / 3.3));
    printBrightness();
    _T2IF = 0;                  //clear the timer interrupt bit
}

#define ISR_PERIOD  10    // in ms

void  configTimer2(void) {
  //T2CON set like this for documentation purposes.
  //could be replaced by T2CON = 0x0020
  T2CON = T2_OFF | T2_IDLE_CON | T2_GATE_OFF
          | T2_32BIT_MODE_OFF
          | T2_SOURCE_INT
          | T2_PS_1_64 ;  //results in T2CON = 0x0020
  //subtract 1 from ticks value assigned to PR2 because period is PRx + 1
  PR2 = msToU16Ticks (ISR_PERIOD, getTimerPrescale(T2CONbits)) - 1;
  TMR2  = 0;                        //clear timer2 value
  _T2IF = 0;                        //clear interrupt flag
  _T2IP = 1;                        //choose a priority
  _T2IE = 1;                        //enable the interrupt
  T2CONbits.TON = 1;                //turn on the timer
}


int main (void) {
  CONFIG_LED1();
  configBasic(HELLO_MSG);       // Set up heartbeat, UART, print hello message and diags
  configControlLCD();       //configure the LCD control lines
  initLCD();                //initialize the LCD
  configTimer2();
    while(1){
        while(f_adcVal > 1.7){
        LED1 = !LED1;
        DELAY_MS(100);
        }
    LED1 = 1;
    }
}
/* Patrick Brooks - 11650957
    Lauren Brackin - 11864680 */
```