Physics 3150, Laboratory 10
Microcontrollers and Embedded Control

April 4 and 6, 2016

Last revised April 2, 2016, by Ed Eyler

Notes:

(1) This is the last organized lab. The rest of the semester will be devoted to your final projects.

(2) You may wish to leave your microcontroller wired on a breadboard after the lab ends, if you are considering a final project that might use it.

Purpose:

1. To build a very simple but complete single-chip computer system, interfaced to a serial LCD display.
2. To learn the process of compiling and loading a program, making a few simple changes in a provided program to display values from a 12-bit ADC built into the microcontroller. We keep this simple, so you don't need to learn the full programming language (standard C, with device-specific extensions) unless you want to use it for a later project.
3. To add an external 12-bit DAC using a fast "SPI" serial interface, which uses only three wires.

References:

Documentation from Microchip, Inc. The basic data sheet is available on the course web page, and much additional information can be found at www.microchip.com.

Equipment:

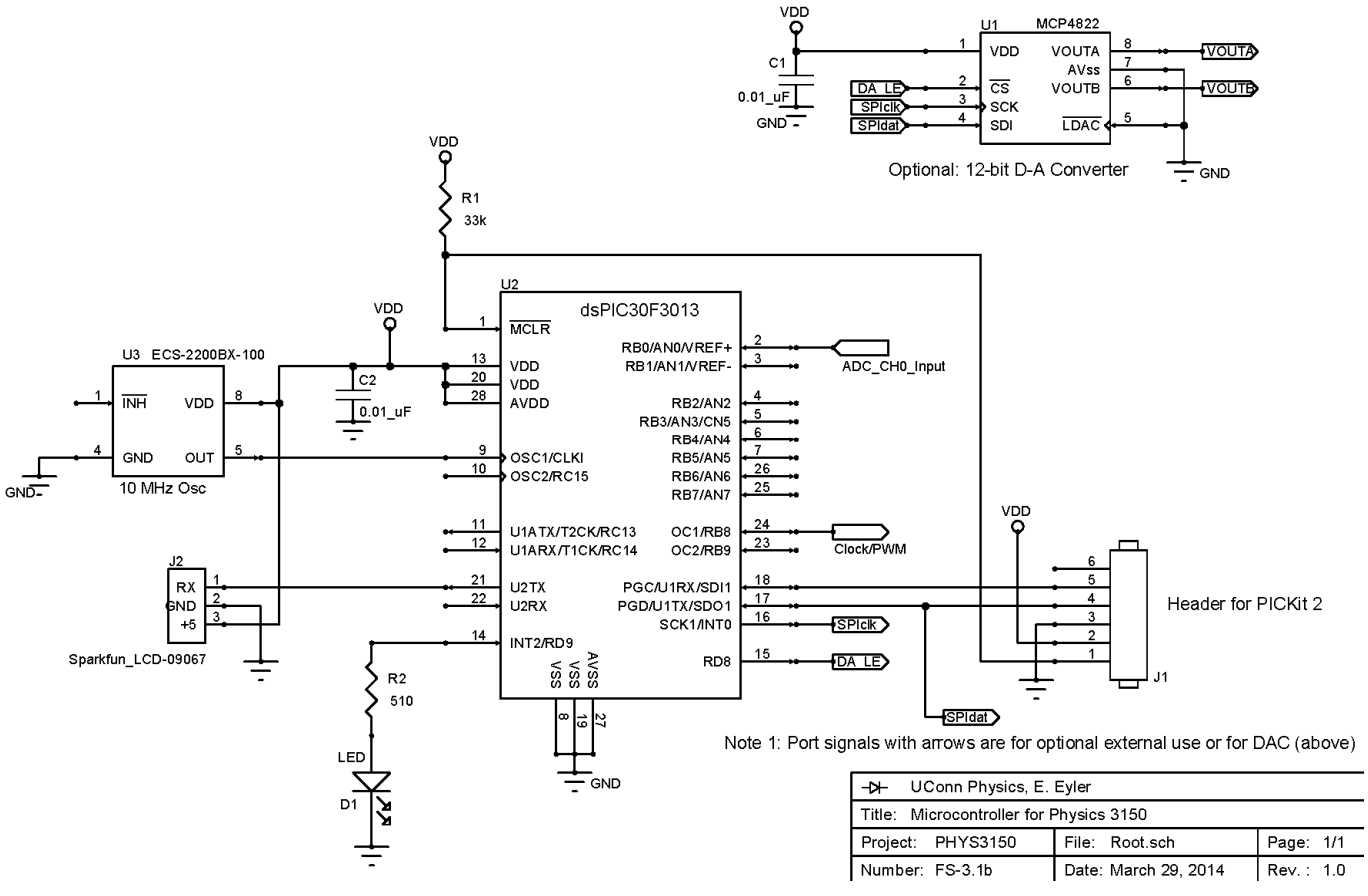
1. Digital oscilloscope.
2. Breadboard with logic switches, logic indicator LEDs, and 5V supplies.
3. 16-bit general-purpose microcontroller in 28-pin package, Microchip dsPIC30F3013.
4. 10 MHz crystal oscillator module, ECS-2200BX-100.
5. Two-line LCD display (5V), Sparkfun LCD-09395.
6. Decoupling capacitors, 0.01 μF to 0.1 μF.
7. Dual 12-bit DAC with SPI serial interface, Microchip MCP4822.
8. PICkit3 programmer from Microchip, Inc., and a 6-pin right-angle header to allow breadboard operation.
9. Desktop PC running Microchip MPLAB with XC-16 compiler, for program development and microcontroller programming.

I. Basic Microcontroller operation with an LCD display

The dsPIC30F3013 is an inexpensive (about $5) self-contained computer containing not only a 16-bit processor and a small random-access memory (2 kB), but also a non-volatile program memory and numerous internal interface modules, including serial interfaces, counter-timers, digital interface lines, and a multiplexed 12-bit ADC. Most of its 28 pins can be assigned to any of several different input/output functions, making the chip quite versatile. The processor is designed to execute 16-bit instructions at up to 30 MHz, and is optimized for programming in the C language. Here we will operate the controller with a 20 MHz instruction cycle frequency. A major advantage of this particular processor series is that it can operate from a 5 V supply, with TTL-compatible inputs and outputs. Closely related 32-bit microcontrollers are available in the Physics 3150 lab that are both more powerful and less expensive, but they generally operate with 3.3 V supplies and incorporate only a handful of 5V-tolerant input pins.

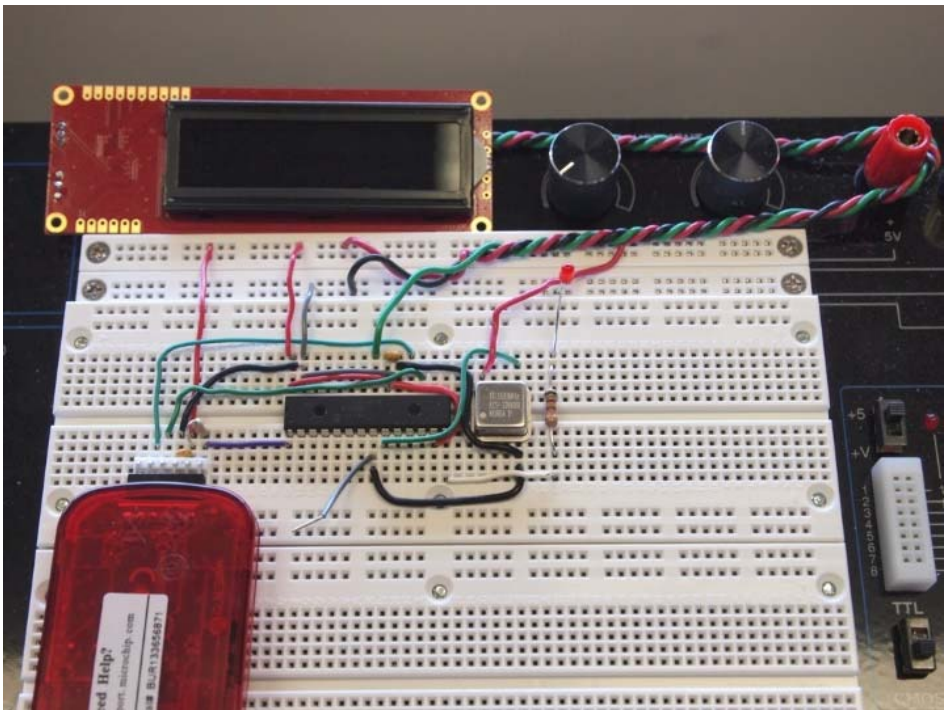
In this portion of the lab, you will connect the microcontroller in a minimal configuration, and by running a pre-stored program, use it to display a running count on an external LCD display panel. This circuit could easily be included as part of a larger device — this is the key idea of *imbedded control*, in which a tiny computer is included as a routine component in a circuit design.

All that's necessary is to wire up the lower portion of the circuit shown below, without the optional 12-bit D-A converter. The four-pin crystal oscillator is used to provide a stable time base The chips are somewhat static-sensitive, so you should try to avoid static electrical discharges on dry days. For the programming header, use a 6-pin right-angle header, and orient it as shown in the photo on the next page. The program will start automatically when power is applied, and can be



restarted by grounding the MCLR line on pin 1. The program listing appears at the end of this writeup. Basically, it does three things:

1. Initialize the microcontroller by specifying how the clock input is to be utilized, and which pins are to be used as digital inputs, digital outputs, or analog inputs.
2. Set an internal 16-bit timer, together with a built-in "Output Compare" device, to generate a clock with a period of 256 ms on the OC1 line. The timer/comparator is actually configured as a programmable pulse-width modulator, which could be used for control of external devices such as heaters or dc motors. For present purposes it is set to a duty cycle of 25%. The timing on OC1 is determined by hardware, not by program execution, so it is extremely precise. The period should be accurate to about 10 parts per million!
3. Using an internal serial interface (a *universal asynchronous receiver/transmitter*, or UART), display the present count value on an external display. The display can also show arbitrary alphanumerical characters, with two lines of 16 characters each. The computer code to provide a buffered serial output stream is fairly complicated, and is in a separately compiled module titled UART30.c, which you will not need to modify.



II. Programming the microcontroller to display ADC data

The provided program also includes a function that can digitize analog input data using the built-in 12-bit A/D converter in the microcontroller. It's intentially set up so that by making a trivial change to the C code, you can substitute the ADC values for the counter values that are displayed by default. The ADC will then be sampled and displayed every 256 ms, which is very convenient for human observation. The ADC itself is capable of operation at up to 100–200 kHz.

Open the program in the MPLAB programming environment by double-clicking on "Phys3150.c" under "Source Files" in the project window at the upper left. In the editing window, look for the primary execution loop that appears on the top of page 3, starting with `while (!Done) {}`.

All that's needed is to change "count" to "adcVal" in the argument of the function call to sprintf(). The edited line should read `sprintf(outstr, "%Value: %-5u", adcVal);`.

To compile and load your new program, you will need to attach a PICkit3 programmer to your microcontroller. It attaches to the right-angle header, but note that the connector must be oriented so that the programmer is upside-down, since it is otherwise too thick to fit without pulling the header out of the breadboard.

Once your programmer is attached to the breadboard and to a USB connector on the PC, you can compile your modified program and load the resulting machine-language code onto the controller by selecting "Run" from the main menu at the top of the MPLAB user interface window. The display window at the bottom of the page will indicate the progress as your program compiles, then loads into binary form, then is programmed onto the chip. If anything goes amiss, ask your TA for assistance.

Attach a variable dc voltage to the ADC_CH0_Input pin (pin 2 of the dsPIC30F3013), and watch the corresponding 12-bit DAC output on your display. **Be sure not to exceed 5.1 V at the input, nor to apply negative voltages, since either condition will likely cause damage to the microcontroller.** If you use, say, a 0–20 V supply, add a voltage divider to make sure that you don't exceed 5 V.

III. Adding an external 12-bit DAC with an SPI interface

One feature not included in the dsPIC30F3013 is a D/A converter. However, the MCP4822 is a simple 8-pin chip that provides a pair of 12-bit DACs, designed for easy interfacing with a high-speed SPI interface. Unlike the old-fashioned serial data conventions, the SPI interface includes both a data line and a clock signal, indicating exactly when the data bits can be read. This allows data transfer rates up to 50 megabits/s.

Your program already includes an initialization routine for the SPI interface, accomplished using library functions provided by Microchip. All you need to do to use a DAC is:

1. Wire the DAC into your circuit, as indicated at the upper-right-hand corner of the schematic diagram. We will need only DAC A, so the output signal will appear at the VOUTA port on pin 8. Three wires are needed for an SPI interface: a data line, a clock line, and a "chip select" line $\overline{\text{CS}}$ that informs the chip that it is the one presently being addressed by the interface. It also serves as a data latch enable for the DAC.
2. Make a further modification to the program by taking the count that was originally sent to the display panel, and instead sending it to the DAC. This can be done by inserting a call to the function Write_DAC(0,value). The code for this function is already provided in the program. In addition, it would be nice to use a faster clock to generate a more rapid ramp. You can decrease the Timer 3 period by changing tmrPeriod from 20000 to something like 6–10. However, you will also need to "comment out" the code that writes to the LCD display via UART2, because it will otherwise cause an overflow that will hold up program execution.
3. Now repeat the compilation and programming cycle. **The SPI data output is shared with the programming intput, so the PICkit3 programmer must be disconnected before the DAC will operate properly.** Do so, and observe the ramp output. Only the lowest-order 12 bits of the count are actually used, so the output should be a repetitive sawtooth ranging from output code zero (0 V) to output code 4095 decimal (4.096 V, derived from an internal voltage reference).

```
/************************************************
* Phys3150.c
*
*  Written for the dsPIC30F3013 16-bit microcontroller from Microchip,using
*  the Microchip MPLAB XC-16 Compiler in its free configuration.
*
*  The program demonstrates the timer, output compare register, A/D converter, UART serial
*    interface, and SPI cserial interface.  It is written to be easily expandable for
*    use in student projects --- for example, the timer/compare setup implements a pulse-width
*    modulator that could easily be modified to control a heater or motor.
*
*   Hardware interrupts are used only in the "hidden" code of
*    the UART management routines, which are in a separately compiled file,
*    to make the flow of execution easy to follow.
*
*  In the main program (function main()), the primary execution loop is the
*    code segment beginning with while (!Done).  It executes indefinitely.
*
*  Last modified by E. Eyler, University of Connecticut,
*  on 4/2/14
************************************************/

/* Specify some standard header files to be included for compilation */
#include <p30fxxxx.h>
#include <libpic30.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <timer.h>
#include <outcompare.h>
#include <spi.h>

/* Define some numerical constants and mnemonics */
#define F_CY 20000000// Frequency of instruction clock (=Fosc/4)
#define CYCLES_MSEC (F_CY/1000)// Number of cycles per msec
#define DAC_CS   PORTDbits.RD8
#define LED  PORTDbits.RD9

/* The following configuration data is for special registers on the dsPIC30F3013 chip */
_FOSC(CSW_FSCM_OFF & ECIO_PLL8);// Use external oscillator x8, no switching
_FWDT(WDT_OFF);// Disable watchdog timer
_FBORPOR(PBOR_OFF & PWRT_16 & MCLR_EN); // 16 ms timer, enable MCLR reset
_FGS(CODE_PROT_OFF);// Code is not protected

/* Declare prototypes for functions in this source file */
int Write_DAC(unsigned int DAC, unsigned int val);
void Setup_ADC(void);

/* THese are prototypes for functions in separately compiled source files*/
extern void __delay32(unsigned long cycles);
void delaymsec (unsigned long duration);
void itoafw(int m, int fwidth, char s[]);
int UART2_CharIn(void);
void UART2_Init (void);
void UART2_StringOut(char* outstr);
int UART2_CharOut(char outchar);

/* Define global variables */
unsigned long F_Cy;//Microcontroller cycle frequency
```

```
/*********************************************
main();
*********************************************/

int main (void)
{
unsigned int count;
char outstr[40];
char Done;
unsigned int tmrPeriod, ocPeriod; //Periods for Timer 2 and output comparator
int adcVal;       //Latest data value from the A/D converter

  F_Cy = F_CY;// Initialize clock frequency for external functions

/* Start initializing: Set up the processor for output on Port B */
  PORTB = 0;        // Reset PORTB latch
  ADPCFG = 0xFFFE;// Set ADC pins to digital except AN0
  TRISB = 0x03;     // Set PORTB as outputs except for RB0, RB1
  /* Also set up Port D, with bits 8 and 9 as outputs */
  TRISD = 0x0FF;
  /* Initialize bits D8 (DAC select) and D9 (LED), both high */
  DAC_CS = 1;
  LED = 1;

/* Wait a while for external hardware initialization/warmup */
  delaymsec(1400);
/* Now set up serial transmission on UART2 for the external Sparkfun LCD display */
  UART2_Init();
  UART2_StringOut("\xFE\x01");    //Clear the display
  UART2_StringOut("Hello");
  delaymsec(1000);                //Hold the welcome message for a second

/* Initialize the SPI serial interface for optional use with an MCP4822 dual 12-bit DAC.
   Disable interrupts and use a 20 MHz clock. */
  ConfigIntSPI1(SPI_INT_DIS);
  OpenSPI1(FRAME_ENABLE_OFF & ENABLE_SDO_PIN & SPI_MODE16_ON &
    SPI_CKE_ON & SLAVE_ENABLE_OFF & CLK_POL_ACTIVE_HIGH &
    MASTER_ENABLE_ON & SEC_PRESCAL_1_1 & PRI_PRESCAL_1_1,
    SPI_ENABLE & SPI_IDLE_CON & SPI_RX_OVFLOW_CLR);

/* Initialize the built-in ADC for acquisition on pin AN0, triggered by Timer 3 */
  Setup_ADC();

/* Initialze Timer 3 and output comparator OC1 to provide a PWM output on pin OC1.
 * The periods are measured in units of 12.8 microseconds for a 20 MHz instruction clock,
 * and will be accurate within 10-20 parts per million. */
  tmrPeriod = 20000;    //Set the timer period using a Microchip-supplied library routine
  OpenTimer3(T3_ON & T3_GATE_OFF & T3_PS_1_256 & T3_SOURCE_INT,
    tmrPeriod-1); //Note that actual period is count-1, so we adjust accordingly
  ocPeriod = tmrPeriod/4;   //Set PWM 'on' time to 25% of the clock cycle
  //Call another library routine to set up and start the output comparator unit
  OpenOC1(OC_TIMER3_SRC & OC_PWM_FAULT_PIN_DISABLE,ocPeriod,ocPeriod);

/* Now enter an infinite loop and just keep running things. */
  Done = 0;
  count = 0;
```

```
  while (!Done) {
    // Check for a Timer 3 "tick" by looking at its interrupt flag, and update everything
    //   if a timer reset event has just occurred
    if (IFS0bits.T3IF) {              //The timer 3 interrupt flag has been set
      count++;                        //Increment the count
      IFS0bits.T3IF = 0;              //Reset the timer flag to await the next tick
      LED = ~LED;                     //Toggle the LED as a quick visual check that we are here

      //Read the ADC, which is triggered by the timer
      while (!ADCON1bits.DONE) {};    //Wait if the conversion is presently in progress
      adcVal = ADCBUF0;               //Read the 12-bit value, storing it as adcVal

      // Write to the LCD display.  For data rates > 50 Hz, comment out this section.
      UART2_StringOut("\xFE\x80");    //Reset the Sparkfun display cursor to row 0, column 0
      //Write an unsigned integer value to text string outstr, in a 5-character field width
      sprintf(outstr, "Value: %-5u", count);
      UART2_StringOut(outstr);        //Send the text string to the display
    }

  }// end of while (!Done)

  /* We should never reach this point, but if we do, shut down */
  while (1) ;// Wait forever for a reset
}

/*****************************************
*void Setup_ADC(void);
*
*  This function sets up ADC channel 0.
*  New conversions start automatically whenever a compare event occurs
*  for Timer 3.
*****************************************/
void Setup_ADC(void)
{
  ADCON1 = 0;               //Start with all configuration bits 0
  ADCON2 = 0;
  ADCON3 = 0;
  ADCON1bits.SSRC = 2;      //Timer 3 starts the conversion
  ADCON1bits.ASAM = 1;      //Start sampling after each conversion
  ADCON2bits.SMPI = 0;      //Set interrrupt flag after each conversion
  ADCON3bits.ADCS = 0x3F;   //Conversion clock at minimum, 32*Tcy
  ADCHS = 0;                //Use pin AN0 for channel 0 positive input
  ADCON1bits.ADON = 1;      //Enable the ADC
  IFS0bits.ADIF = 0;        //Clear ADC interrupt flag
  return;
}

/*****************************************
*int Write_DAC(unsigned int DAC, unsigned int val);
*
*  This function writes a 12-bit value to an MCP4822 DAC via SPI 1.
*  DAC=0 selects DAC A, 1 selects DAC B.
*  Returns 0 if normal, -1 if a fault occurred.
*****************************************/
int Write_DAC(unsigned int DAC, unsigned int val)
{

  DAC_CS = 0;                   //Set DAC chip select/latch enable low
  SPI1STATbits.SPIROV = 0;//Clear receive overflow flag, if it's set
```

```
  SPI1BUF = (val & 0xFFF) | 0x1000 | DAC<<15;
  while (!IFS0bits.SPI1IF) {}    // Wait for the transfer to finish
  IFS0bits.SPI1IF = 0;           //Clear the flag for next time
  DAC_CS = 1;                    //All done---Set DAC latch enable high

  return(0);
}
```