Physics 319 Spring 2025:

Introduction to the Programming and Use of Microprocessors

Sing Chow, Andrzej Kotlicki, Carl Michal and Ryan Wicks

This lab is going to introduce you to the world of microprocessors. As you probably know, most modern equipment, from streetlights, to cell phones, to cars and to planes are controlled by one or many microprocessors. In this lab you are going to learn how to build and program microprocessor based devices.

You will work with a MSP430 microprocessor, powered by a USB port, which will also be used to communicate with a computer.

You will learn how to interface the MSP430 with sensors, actuators, displays, wireless links and other devices.

Remember these safety rules while working with the MSP430! Any mistake involving violation of these rules might damage the board. To continue, you will have to purchase another one!

- Static charge from your hands, wires or tools can damage the microprocessor; always touch some metal ground object like the BNC connector on the oscilloscope before touching the board.
- None of the microprocessor leads should ever be directly connected to voltages below ground or above 3.6V! We will connect the 5V outputs from sensors, but only with large series resistors to protect the board.
- Turn off the power before working on the circuit.
- Do not connect any logic pins (any pins except power supply) directly to the power supply.
- Open inputs should be either grounded or pulled up (either internally or externally). They should never be left floating.
- Do not connect two or more logic outputs together.
- Discharge any electrolytic capacitors before inserting them into the circuit

You will program the microprocessor from your computers.

Lab Notes

As you work through the activities in the manual, you must keep notes on what you do. You should keep all your notes electronically. Your notes should contain things like: clear statements of what you are trying to do (your objective), circuit diagrams, notes of problems you encounter and how you solve them, file names of programs and data, links to documentation consulted, explanations to of things you've read about, explanations of help you've received from other students or course staff. You will need to have drawings of things like circuit diagrams that you may wish to draw quickly by hand – these can be photographed or scanned and included in your lab notebook. This inclusion should be done immediately as otherwise it is easily forgotten. Your programs should be well commented so they are readable, and should include pointers to dates and locations in your lab notes where related circuit diagrams can be found. You can use any editor or Jupiter notebooks to create your notes but you have to be able to convert them to pdf. After each two-week block of labs, you will need to turn in your notes and programs to your TA.

As this is an upper level lab course, you are expected to show initiative. Performing the tasks recommended in the lab manual will not get you full marks for the labs, we expect you to go beyond the printed instructions and explore the equipment and software on your own.

Circuit Tidyness:

As your circuits become increasing complicated, the tidyness of the circuits you build will become more important for two reasons: the first reason is that tidier circuits are much easier to debug when things go wrong. The second reason is that *tidier circuits actually work better*. With high speed logic circuits, tidier circuits tend to produce far less interference between nearby signal lines, improving the reliability of the circuit. Some general advice is to avoid using extra wire wherever practical – don't have huge loops of wire to connect nearby components. Lay out the components to minimize the distances between connections. Use the power rails on the breadboard for power supply and ground lines, but not for any sort of signal lines.

1. Setting up the students number by hand

The Breadboard: Begin by reminding yourself which holes are electrically connected to which other holes on the solderless breadboard.

Manually setting a display:

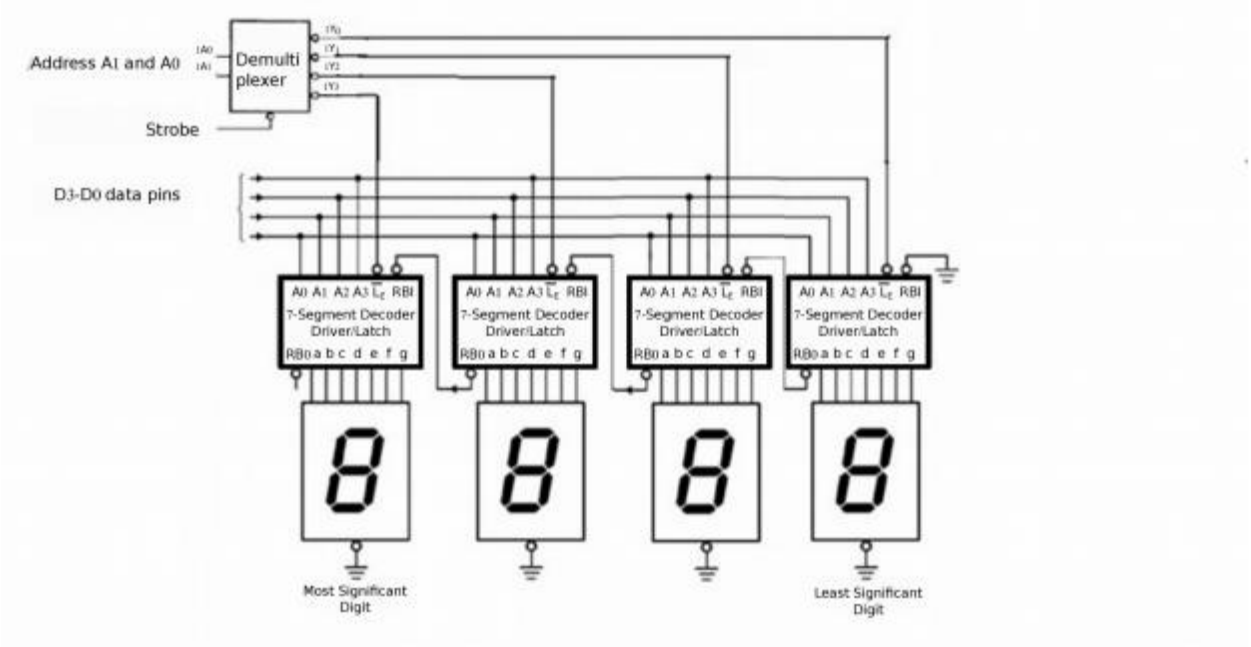


Fig.1. Schematic of lab breadboard display.

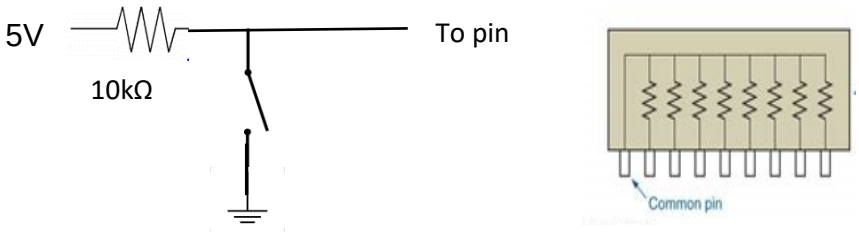


Fig.2. Each input pin (D0-D3, A0,A1 and strobe) should be controlled by a set up like this. You will need an integrated block of 10kΩ resistors (the diagram shown on the right) and the block of 8 switches to set it up.

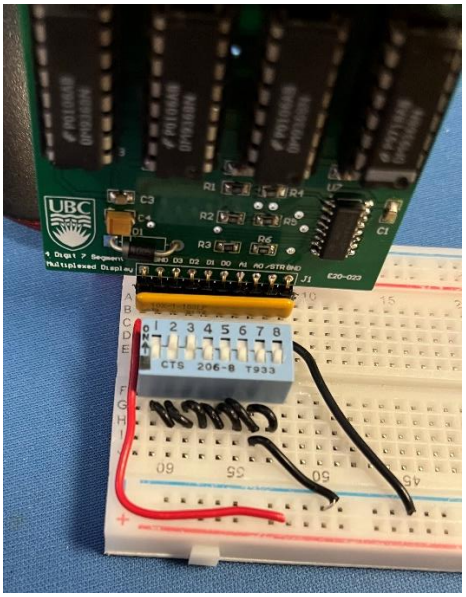
You have a multiplexed version of the seven segment display , shown schematically in Fig .1. Plug it into the breadboard . You will display 4 digit number on this display : initially by hand , and then with the microcontroller. The purpose of this exercise is to introduce you to the microprocessor input /output at its most basic level and learn a little about the MSP430's architecture.

To begin, read the data sheet for the DM9368 (7 segment decoder, driver and latch) and the 74HC139 (2 to 4 line decoder). Find out (and explain in your notes) the function of the each of these chips, and how you use them to reduce the 32 wires needed to drive 4 individual 7 segment displays (with seven segments each) down to 7 wires.

Now you will try to display the last 4 digits of your student number on the 7-segment displays, by hand. To achieve a logical one on the input, you will connect it to 5V through a 10 kΩ resistor or connect directly to GND (using the block of 8 switches) for a logical 0. The resistor acts as a current limit. When the switch is open 5 V is delivered to the pin, when it is closed the pin is connected to the ground and so is the resistor (See Fig.2). The 5V has to come from the board power supply not from the microprocessor. [1]

Remember the information appears on the output pins when the strobe (enable) input goes from high to low.

The photos below show the easiest way to set it all up. The writing on the integrated resistor block should face the switches. Notice that open switch corresponds to logical 1 on the input pin and closed switch corresponds to 0.



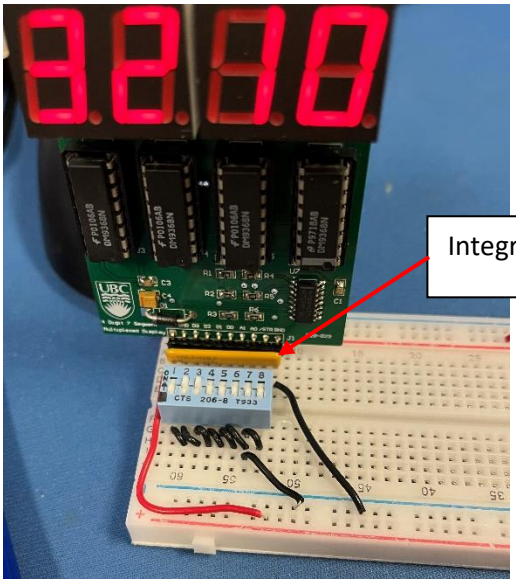


Fig.2.

No pins should be connected directly to 5V except the one that powers the display.

Write down the steps you followed to complete this task.

Notice: The display might not show a leading 0. If the most significant digit is 0 it might show blank.

Show the program and display with your number to the TA

2. Using the Microprocessor to set the display

The next task is to do the same thing with a microprocessor. Now, instead of changing the inputs manually, you connect them to your microprocessor, and write a program to set the value on each display. Before we get into the details of how the microprocessor executes the program, let's consider the connections we have to make.

Make the following connections (notice that you do not need the integrated resistor block any more):

LaunchPad	P6.3	P6.2	P6.1	P6.0		P4.1	P4.0	P2.0
display	D3	D2	D1	D0	-	A1	A0	strobe

Now you have to write software to control the outputs of the microcontroller. At the most basic level (and you can't get more basic than microcontroller programming), software is simply a list of instructions that the microprocessor steps through and executes. You write the software on a computer in assembly, or in a high level language like C, compile it into machine code, and then load it into a non-volatile section of memory on the microcontroller (normally Flash or EEPROM nowadays). On initial power up or on a reset, the MSP430 microcontroller reads the contents of a particular spot in memory, called the reset vector. This reset vector is loaded with the memory location of the start of your program. It takes this memory location, and loads it into a special memory register called the program counter (PC). This register contains the memory location of the next instruction to be executed by the microprocessor. It is incremented as each instruction is evaluated, and this process continues forever. Conditional branching and looping are achieved by changing the value of the program counter to point to an arbitrary spot in memory. We will get into conditional looping and branching in a later lab.

This is the function of any microprocessor, to execute a list of instructions. For the MSP430, there are a total of 27 different instructions available, plus a few derived instructions. You can find a listing of all of these instructions in the reference document. When you write program in C and build it

in Code Composer it get translated to a set of these instructions. We are going to restrict our programing to C so we will only have to figure out how to tell the microprocessor to do in C language.

Before you attempt to write the program, take some time to understand how the microcontroller communicates with the outside world and its own peripherals by writing to special locations in memory.

General-purpose I/O registers overview:

Our MSP430 launchpad microcontroller is a full system on a chip, containing a microprocessor and all the extra peripherals you need to run it, like memory, storage, clocks, counters and a variety of other modules that are useful in certain situations (like analog to digital converters, serial ports and PWM - pulse width modulators). The microprocessor inside the microcontroller remains the same over an entire range of devices offered (just look at the number of MSP430 devices offered by TI). It is unnecessary to create a new microprocessor with new instructions for each iteration of the design. Therefore, the approach taken by all manufacturers is to make peripheral devices accessible by writing or reading information from special locations in memory (known as special function registers (SFR's). Beyond just controlling peripherals, these specific memory locations also control the settings of the microcontroller. In this section, we will introduce you to the memory locations used to control the function of the external microcontroller general purpose input/output (GPIO) pins.

The digital inputs and outputs are organized into 8-bit ports called parallel-input-output ports (I/O). There are 7 available I/O ports in MSP430F5529 but not all the pins are available. There are specific registers controlling these I/O ports. For port P1 for example there are following registers (for other ports just replace 1 with an appropriate number):

P1DIR – sets the pin directions. Bit = 0 = input, Bit = 1 = output.

P1IN – input register. When configured for input, this register contains the digital input values

P1OUT – output register. When configured for output, writing to this register sets the outputs. When configured as input sets the pullup (1) or down (0)

P1REN – pullup/pulldown enable. Bit = 1, enable resistor - P1OUT sets whether pullup(1) or down(0).

P1SEL – alternate function enable

P1IE – Enable interrupt on some input port pins

P1IES – interrupt occurs on raising (0) or falling (1) edge

Each of these registers controls the behaviour of each pin individually. When the bits of the register PxSEL (SEL = select) are set to 0 (which is the default, which means that they are set to 0

when the microprocessor is switched on) the port is set as an input/output port (as opposed to an ADC input, for instance). When PxSEL=0, the values of the bits in register PxDIR (DIR = direction) defines if the appropriate port pins are set as inputs (0) or outputs (1). If a bit in PxSEL = 1 then the pin is connected to some alternate function. Which function depends on the specific chip and pin – reading the datasheet is necessary.

An input pin should not be left open when the PxDIR=0 because electric noise can switch it up or down. One can prevent this by setting the bit corresponding to the open input in the register PxREN=1 (REN = resistor enable) to enable the pull down or up resistors. The pull up (bringing the input to high) or down (bringing the input to ground) is determined by the setting of register PxOUT. The pull up – pull down function is very useful when the pin is used as an input. For example if we pull the input pin up we could control its state with only one switch to ground (switch on will correspond to logical 0 and switch off to logical 1, as the pin will be pulled up). For input pins, PxIN can be examined to read in the state of signals applied to them.

Your program for this part of the lab needs to deal with registers P1DIR, P1REN and P1OUT, while the register P1SEL can be left as they are, because they are set to zero on power up. The use of the interrupt registers P1IES (IES = interrupt edge set), P1IE (IE = interrupt enable) and P1IFG (IFG = interrupt flag) will be demonstrated later.

Before we can start writing the code we need some software, which can translate C program to the machine language. We are going to use the integrated development environment (IDE) called Code-Composer Studio (CCS).provided by Texas Instruments, It works well both on Windows computers and Macs.

We strongly recommend using a browser other than a Microsoft browser for the instalation steps. Use Firefox or Chrome.
To install the CCS go to:
https://software-dl.ti.com/ccs/esd/documents/ccs_downloads.html
and download Version 10. The off line installer works best. After downloading install it with custom installation using only the parts mentioning MSP430.
To start programming open CCS and press “file, “new”, “project”, “Code Composer Studio”, “CCS project”, set the family to MSP430x5xx and processor to MSP430F5529. Name your project and choose empty project with main. Write or copy and paste the program you want to run.
Once the code is ready, press "project" then "build”. It will compile it into a machine language code. Afterwards press "run" and "Debug" it will load the program to the Launchpad, then press "run" and "resume", which will start it. You will try and modify all your C programs this way. You should try it with a program from part 3 of the manual. This is all software you need for parts 2 to 7 of the manual.

Back to Lab 2:

Now that you understand how to change the state of the output pins and have connected up the circuit, you can start writing your code. There are some things you have to include in your code:

```
#include <msp430.h>   //this file provides the code with all the relevant names
                       //of the registers and their corresponding codes
```

```
void main(void) {     //start of the main program

          WDTCTL = WDTPW + WDTHOLD;      // stop the watchdog timer (the watchdog timer is a
                                         //feature that allows the cpu to detect and recover
                                          //from some kinds of software
                                           //bugs. We just want to disable it for now. )

   P6DIR = 0b00001111;     //Set pins 0,1,2,3 of port 6 to output

   P4DIR =  0b00000011;     //Set pins 0,1 of port 4 to output

   P4OUT = 0b00000000;       //Set pins 0,1 of port 4 to low, this will correspond

                     //to choosing a least significand digit in a 7 segment display


   }

   //To decide which pin is output or input we set it to 1 or 0. It means
   //sending a binary number to the port. Binary numbers are represented with the
   //prefix 0b as shown above. Notice that this binary representation might not
   //work in all C compilers.
```

Now continue with this program sending the high or low to the pins corresponding to your student number using P1OUT = xxx commands after issuing all the needed the P1DIR commands for all the ports you are going to use. Notice that you need to repeat what you did manualy:

Set the data and address and keeping them bring the strobe down and up again. How many commands does it requare per digit?

Notice: The display might not show a leading 0. If the most significant digit is 0 it might show blank.

Show the program and display with your number to the TA

3. Example of program in C to introduce timing

(https://www.phas.ubc.ca/~kotlicki/Physics_319/Program1.C)

Make sure that you understand all the lines in the C programs and comment them or add to the existing comments.

```
/*
* PHYS319 timing Example in C
* Written by Ryan Wicks
* Modified by A.K,
*/
#include <msp430.h>
void main(void) {
WDTCTL = WDTPW + WDTHOLD; //Stop WDT
P1DIR =0b00000001; //Set P1.0  to output;
P4DIR =0b10000000; //Set P4.7  to output;
P1OUT = 0b00000000; //Set the output Pin P1.0 to low
P4OUT = 0b10000000; //Set the output Pin  P4.7 to high
```

```
while (1) { // Loop forever
 delay_cycles (500000); //This function introduces 0.5 s delay
P1OUT = P1OUT ^ 0b00000001; //bitwise xor the output with 00000001
P4OUT = P4OUT ^ 0b10000000; //bitwise xor the output with 10000000
}
}
```

Change the timing to make the blinking in Program 1 twice as fast and make it blink both on both off. Show the working modifications to your TA

4. Example of program in C to introduce interrupts

Program 2 in C (https://www.phas.ubc.ca/~kotlicki/Physics_319/Program2.C)

```
/*
* PHYS319 Interrupt Example in C
*
* Written by Ryan Wicks
* Modified by A.K,
*
*
*/
#include <msp430.h>
void main(void)
{
WDTCTL = WDTPW + WDTHOLD;     // Stop watchdog timer
P1DIR = 0b00000001;           //set  P1.0 pin for output rest including  P1.1 are inputs
P4DIR = 0b10000000;           //set  P4.7 pin for output
P1OUT = 0b00000011;           // set Pin P1.0  to high and P1.1 to pullup
P4OUT = 0b10000000;           // set Pin P4.7  to high
P1REN = 0b00000010;           //enable pull up/down resistor on P1.1
P1IE =  0b00000010;           //Enable input at P1.1 as an interrupt
P1IES=  0b00000010;           //Interrupt occures when input voltage goes from High to Low
_BIS_SR (LPM4_bits + GIE);    //Turn on interrupts and go into the lowest
                          //power mode (the program stops here)
                          //Notice the strange format of the function, it is an "intrinsic"
                          //ie. not part of C; it is specific to this microprocessor
}
      //Port 1 interrupt service routine starts below
void __attribute__ ((interrupt(PORT1_VECTOR))) PORT1_ISR(void)    {
                                  //code of the interrupt routine goes here
        P1OUT ^=  0b00000001;
        P4OUT ^=  0b10000000;       // toggle the LEDS  is
        P1IFG &= ~0b00000010;       // Clear P1.1 IFG. If you don't, it just happens again.
    }
```

Modify the program in such a way that both LEDs are off at the beginning, than each push of the button brings it through the sequence: red on, green on, both on, both off.

5. Analog to digital conversion (ADC) with Launchpad

The MSP430 microprocessor you are using has a 12 bit ADC that can sample the input of one or more of the IO pins. The example below will introduce you to its use.

You are going to build a CMOS level tester. Your instrument will illuminate the red built-in LED when the voltage connected to the input pin level is High (above 2.4V) , the built-in green LED should indicate Low (below 0.5V) and an external yellow LED should light up when the voltage is in between these two levels.

The program below (by D. Dang from Texas Instruments and modified by A.K. shows you how to set up the ADC. It just samples the voltage connected to input pin P6.0 and sets the red LED on when voltage exeeds some value. You will need to modify it to obtain the functionality described in the previous paragraph.

For the 12-bit ADC full scale is $2^{12} - 1 = 4095$. The red LED should work with the example program, all you have to do is to get the green and yellow LEDs to indicate the respective logic levels. The test voltage should come from the potentiometer on your breadboard: connect one side to ground, the other side to 3.3V, and connect the wiper (center pin), **through a 330Ω resistor,** to P6.0. Connect P1.2 to an external LED's.

You can download the example program shown below from:
https://www.phas.ubc.ca/~kotlicki/Physics_319/ adc.c

```
//******************************************************************************
//  MSP430G2x31 Demo - ADC10, Sample A0, Vcc Ref, Set P1.0 if > 0.75*Vcc
//
//  Description: A single sample is made on A0 with reference to Vcc.
//  Software sets ADC12SC to start sample and conversion - ADC12SC
//  automatically cleared at end of conversion. ADC12 internal oscillator sets the
// sample time (16x) and conversion.
//
//               MSP430F5529
//            -----------------
//
// input  >---|P6.0/A0      P1.0|--> red Led onboard BIT0
//            |                 |
//            |             P1.2|--> yellow Led external
//            |             P4.7|--> green Led onboard BIT7
//
//
//  D. Dang, modified by AK.
```

```
//  Texas Instruments Inc.
//*****************************************************************************




#include <msp430f5529.h>
#include<stdio.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;
  ADC12CTL0 = ADC12SHT02 + ADC12ON;         // Sampling time, ADC12 on
  ADC12CTL1 = ADC12SHP;                     // sampling timer
  ADC12CTL0 |= ADC12ENC;                    // ADC enable
  P6SEL |= 0b00000001;                      // P6.0 allow ADC on pin 6.0
ADC12MCTL0 = ADC12INCH_0  //selects which input results are
          //stored in memory ADC12MEM0. Input
          //one is selected on reset so this line is not needed
  P1DIR |= 0b00000001;                      // set pin P1.0 as output


  while (1)
  {

    ADC12CTL0 |= ADC12SC;                   // Start sampling
    while (ADC12CTL1 & ADC12BUSY);//while bit ADC12BUSY in register
ADC12CTL1 is high wait

    if(ADC12MEM0>=3072)    //This value depends on the input voltage
       P1OUT |= BIT0;
    else
      P1OUT &= ~BIT0;

  }

}
```

Test the system and show the working instrument to your TA.

6. Pulse Width Modulation (PWM) waveform with Launchpad

PWM waveforms are frequently used to control power delivered to motors, lights, heaters and other devices without wasting energy in series resistors or voltage dividers. Our microcontroller has timer peripherals that allow creation of PWM waveforms without requiring the CPU to be continually setting output pins on and off – you configure the timer to do the work without cpu intervention. You are going to analyze a sample program producing a PWM waveform on output P 1.2. You can download the program from

https://www.phas.ubc.ca/~kotlicki/Physics_319/pwm.c

```
#include <msp430.h>
void main(void){
 WDTCTL = WDTPW|WDTHOLD; // Stop WDT
 P1DIR |= BIT2; // Output on Pin 1.2
 P1SEL |= BIT2; // Pin 1.2 selected as PWM
 TA0CCR0 = 512; // PWM period (512/1.048)  microseconds

 TA0CCR1 = 50; // PWM duty cycle
 TA0CCTL1 = OUTMOD_7; // TA0CCR1 reset/set-high voltage
                      // below count, low voltage when past

 TA0CTL = TASSEL_2 + MC_1 + TAIE +ID_0;
                       // Timer A control set to SMCLK, 1.048 MHz
                       // and count up mode MC_1
 _bis_SR_register(LPM0_bits); // Enter Low power mode 0
   }
```

Observe the resulting waveform on the oscilloscope. Connect the piezoelectric “beeper” to listen to the tone produced. What should be the ration of duty cycle to period to get most pure tone?

The internal crystal oscillator has the frequency of 1.048 MHz.

Modify the program to create an other tone and show it to a TA.

7. LED dimmer – combining the functionality of ADC and PMW.

Create a system which reads the voltage coming from a voltage divider and sets the duty cycle of the PWM between 0 and full period. 0 voltage should correspond to 0 duty cycle and 3.3 V should correspond to 100% duty cycle. The period should be constant and less then 10 ms to avoid flickering of the LED. Connect the PMW output to the LED. You just created an LED dimmer!

Show the working dimmer to a TA.

8. Data acquisition from Serial Port Interface and computer display

Your will be using the MSP430 to read values from a sensor and report these values to the host computer where they are displayed and plotted. We'll begin by setting up some demonstration code that measures the temperature of the MSP430 from an internal sensor, and transmits this data to the host computer.

To get the data displayed on the computer you will need another program. We will use the python language on the host computer to talk to the Launchpad.

You should install python and the other required libraries **before** coming to the lab. The msp430 program we'll use was originally supplied by an engineer at TI (and then changed locally). The python programs can be found in the same download.

The python program uses a graphical user interface called gtk to draw windows on the screen, and a plotting library called matplotlib to make graphs. We will need to install these components.

Components needed are: Anaconda 3 with Python 3.7 which you most likely have from other courses.

After you install Anaconda with Python make sure that you add PySerial to Python. To do it:

-open Anaconda prompt

-type "pip install pyserial"

Download:

https://phas.ubc.ca/~kotlicki/Physics_319/temperature_demo.zip

and unzip it

Build the program in main.c and flash into the Launchpad.

Make sure that all the include commands, used in the previous projects are included. They might be different from the ones showing in this version of the program, depending on the CodeComposer version.

The C code for the Launchpad program is longer than the programs we've used so far, but you can get a lot of insight into Launchpad C programming by studying it. For now just download it, compile it, and load it to your microprocessor.

You will need to edit the python programs to contain the correct serial port name.
You can find which port is connected to the Launchpad from the Windows device manager. It this might be COM3, COM5 or COM7 on Windows and "`/dev/tty.uart-XXXX" for Mac (XXXX is a number)`.
If you have difficulty finding which com port is connected to the Launchpad use the python script, which you can download from our webpage:
List_All_coms.py
Once you know which port you use you can edit the python program from the temperature_demo, paste it into Spyder environment and press RUN.
You should start it with: python-serial-print-new.py
After this python program starts, push button s2 on the Launchpad, and temperature measurements will be delivered from the Launchpad to the python program and printed on the screen.

The program will start in an idle mode where it blinks the LED's. When the P1.1 (s2) button is pushed, it will begin taking temperature measurements and sending them to the USB port. Don't push the switch till you've got the python program started up.

To graph the temperature you will use python-serial-plot-new.py

Once the python program is up and running, press the P1.1 (s2) button and the temperature will be updated on the window titlebar, and a graph of the temperature as a function of time should show on the screen (in Celsius).

In this microprocessor program the ADC readings are taken from the built in temperature sensor.

The result is converted to the temperature in Celsius and transmitted in RS232 data format via

USB port.

Your task is to create a system that uses the MSP430 to make distance measurements using an SRF04 ultrasonic distance measurement sensor, and plot them in real time. You can find details of the sensor at:

https://www.phas.ubc.ca/~kotlicki/Physics_319/hc-sr04_ultrasonic_module_user_guide and

here https://www.phas.ubc.ca/~kotlicki/Physics_319/Ultra-Sonic_Ranger_SRF04.htlm

You are free to use pieces of the demo programs. A complete report will include some

demonstration of the accuracy of the device, including a graph of distances measured with a metre stick versus those measured with your sensor. Your lab notes should fully describe how the programs work, and all the relevant communications protocols.

You should read about the sensor, but the important bits of how to use it are:

– the microcontroller (ie your program) triggers a measurement with a ~ 100 μs long pulse on the trigger pin.

– the sensor will initiate a measurement and then raise the echo pin

– when an echo returns (or the sensor gets tired of waiting, ~36 ms), the sensor lowers the echo pin.

– wait 10 ms or more before repeating.

You need to measure how long the echo pin is held high by the sensor. That time, along with the speed of sound will give you a distance.

You will need to connect the trigger pin to a port pin that is configured as an output, and connect the echo pin, through a 1 or 10 kΩ resistor (its a 5 V device!) to a different pin that is configured as an input.

Some advice: Start simple. Get distance measurements working, and communicate single bytes (distances in cm) to the host computer. If you have time, revise the communications protocol to communicate multiple bytes to allow higher precision measurements (in mm) to be communicated. There are simple ways of measuring the time, and very elegant, high precision ways. Start simple. When you get a simple method working, save your code and if you have time, feel free to work on an elegant, higher precision version. Show the working program to your TA.

9. Using a Heart Beat Monitor and Oxygenetion Sensor MAX 30102

Download the the C program main.c from the Blood sensor.zip on our web page, build it and load it to lunchpad.

Connect MAX30102 sensor to the lunchpad directly using the female – female cables as per table below:

Lunchpad	MAX30102
VCC	VIN
P4.2	SCL
P4.1	SOA
GND	GND

When you start the program or press reset on lunchpad the red LED on lunchpad should blink.

Download the phyton program python-serial-plot-heartbeat.py from the Blood sensor.zip on our web page, change the line:

```
csv_savename = "C:\\Users\\User\\Downloads...\\heartdata.csv"
```

so it corresponds to your directory, choose the appropriate port and run the program.

Some numbers should be printed on the console and a plot starts. If it does not start disconnect the MSP430power and reconnect.

Analyse both MSP430 and Python programs and describe how the lunchpad communicates with the sensor, what setting are set on it (what mode and speed it operates on) and what data is send to the computer console.

The datasheet of the sensor, MAX30102.pdf is in the zip file as well.

What is the content of the heartdata.csv file?

Use the sensor to analise your heartbeat. What was your heart rate? How did you calculate it?

Show the working sensor to your TA. You should be able to show how you accuire new data and analyse it.

Appendix.

Pin allocation