Today's plan:

- Announcements: midterm, projects, scheduling
- Solution to Activity 3
- Lab 5/6:
  - Python and GUI's.
  - Timers on the MSP430
- Sensors [next time]
- Activity 4

Announcements:

- Midterm coming on **Feb 9**, will be similar to in-class activities we've been having, but **individual**, and longer. Will need to write simple programs in C and/or assembler for MSP430, you will need to extract information from data sheets (which I will provide), and will need to analyze/explain code samples.

Announcements:

• Projects: should be fleshing out. You will need to provide a short written description (eg 1/2 page) of what you plan to build. In it, describe: 1) the function of the project, 2) an outline of 'how it will work,' both in terms of user interface and in terms of what hardware does what. 3) A list of what kinds of parts you will require. 4) How you plan to acquire parts. [things like op-amps, comparators, transistors, resistors can be supplied by the lab, no need to worry about those.]

• Bring written description to turn in during your lab section Jan 31 -Feb 6 (Lab 5)**.**

• Project Scope: Your project **must** use the MSP430 as a central component. Your project should incorporate *at least one* non-trivial external hardware component (sensor, motor, display etc). Your project may (but is not required) to communicate with a host computer for display or user interaction.

Announcements:

- Computer set-up for labs 5/6: do it ahead of time! Should be able to test at home with Serial Port Example. This part is unlikely to work reliably on Windows with USB 3 ports.

- Holidays: Monday Feb 13 is a holiday (Family day). Monday people: we don't want to get a week behind. If at all possible, please make up the time on T or Th Feb 2, 7, 9, 14, 16. Lab 5/6 due for everyone on Friday Feb 17.

- Feb 20-24 is spring break, the lab will be closed

- Lab time to work on Projects begins Feb 14

Activity 3

1) Find the values of x after each of the following C commands (run on an MSP430). Answer in hexadecimal.

unsigned int x = 6;
a) x %= 4;

b) x = 96 >> 2;

c) x = ~65280; Hint: 65280 = 0xFF00

d) x = 32769 << 4; Hint: 32769 = 0x8001

2) Write the lines of C code needed to:

a) set P1.0 to P1.3 to be outputs and P1.4 to P1.7 to be inputs

b) starts a for loop that continuously copies the values on the inputs to the values of the outputs (ie P1.4 -> P1.0, P1.5 -> P1.1 etc).

c) if all four of the inputs are 0, exit the loop.

Activity 3

1) Find the values of x after each of the following C commands (run on an MSP430). Answer in hexadecimal.

unsigned int x = 6;

a) x %= 4; x = 0x2

b) x = 96 >> 2; x = 0x18 (= 96/4)

c) x = ~65280; Hint: 65280 = 0xFF00 x = 0x00FF

d) x = 32769 << 4; Hint: 32769 = 0x8001 x = 0x0010

2) Write the lines of C code needed to:

a) set P1.0 to P1.3 to be outputs and P1.4 to P1.7 to be inputs

b) starts a for loop that continuously copies the values on the inputs to the values of the outputs (ie P1.4 -> P1.0, P1.5 -> P1.1 etc).

c) if all four of the inputs are 0, exit the loop.

```
P1DIR = 0x0F;
P1OUT = P1IN >>4;
for(; (P1IN >> 4) != 0  ; P1OUT = P1IN >>4)
```

```
for(P1DIR = 0x0F; (P1IN >> 4) != 0  ; P1OUT = P1IN >>4)
```

Lab 3/4, ADC:

```
ADC10CTL0 |= ENC + ADC10SC;
while (ADC10CTL1 & ADC10BUSY);
```

```
while (a == b){                    meaning of while (a & b);


}
```

vs

```
while(a == b);
{

}
```

Labs 5/6 Distance measurement with real-time display.

1. Trigger the ultrasonic distance measurement sensor to begin a measurement.

2. Do nothing till the “echo” pin goes high

3. Time the interval while the echo pin is high

4. Transmit the time interval through the serial port to the host computer where a python program will receive it and plot it.

Python and GUI's

- Python is a “high-level language” (complicated things are often easier)
- Interpreted, not compiled (speed/efficiency/memory footprint)
- cross-platform (windows/mac/linux)
- whitespace/indenting matters (unlike C where it's just for readability)

Do you need to learn Python? (no python on midterm)

Do you need to install Python, yes – before next week's lab, or use lab computers!

C vs Python:

Python programs generally consume much more memory and execute much more slowly than an equivalent program in C.

For a well-written python program the speed penalty is often tolerable.

Why are we using python?

- It is much easier to make a non-trivial program (with gui) work cross-platform in python than C.

- Interacting with system hardware (eg serial port) is very straightforward in python.

To make your python programs run tolerably fast:

If you are manipulating arrays of numbers,
1) always use numpy arrays
2) never iterate over the array if you can avoid it (and you can almost always avoid it!)

eg:

```
import numpy as np   # similar to an include file, but way more powerful.
a = np.arange(0,50,1)  # a is an array of 50 elements: 0, 1, 2 ... 49
b=a * 0.2                    # b is an array 0, 0.2, 0.4, ... 0.98
```

vs:

```
a =  range(0,50,1)
b=[]
for i in range(50):
      b[i] = a[i]*0.2
```

```
import numpy as np
...
ser = serial.Serial(port,9600,timeout = 0.05)
ser.baudrate=9600 #sometimes pyserial has trouble unless you change the
#  baud rate
# with timeout=0, read returns immediately, even if no data
# with timeout=.05, ser.read will wait for up to 50 ms for a byte to appear
# from the serial port, if there isn't one waiting.
...

yvals = np.zeros(50) #array to hold last 50 measurements

...

while(1):                       # loop forever
    data = ser.read(1)          # look for a character from serial port
    if len(data) > 0:           #was there a byte to read?
        yvals = np.roll(yvals,-1) # shift the values in the array
        yvals[49] = ord(data)  # take the value of the byte
        outFile.write(str(time()-start_time)+" "+str(yvals[49])+"\n") #write to file
        line.set_ydata(yvals)  # draw the line
        fig.canvas.draw()       # update the canvas
        win.set_title("Temp: "+str(yvals[49])+" deg F")
    while gtk.events_pending():  #makes sure the GUI updates
        gtk.main_iteration()
```

Python resources:

There are tons of python resources on the web.

Some useful starting points:

Phys 409 has a couple of nice introductory slides sets:
http://www.phas.ubc.ca/~phys409

Beginners guide to python:
https://wiki.python.org/moin/BeginnersGuide

numpy:
http://www.numpy.org/

Matplotlib:
http://matplotlib.org/contents.html

Event Timing

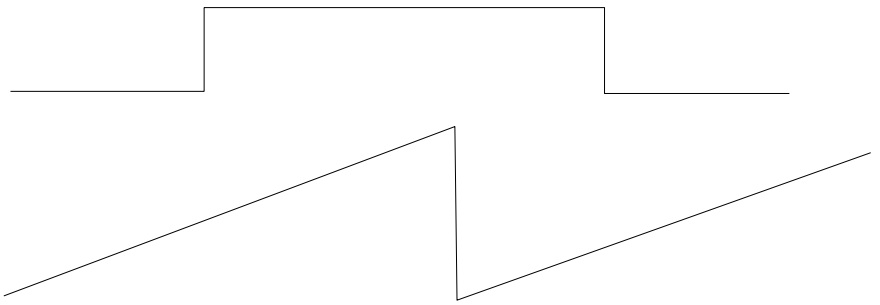
The MSP430G2553 has two timer units (Timer A0 and Timer A1) that count without CPU intervention. See Chapter 12 in the family reference manual

We used one of these timers to generate PWM signals on an output in lab 4

Several strategies are possible to use the timer, eg:

A) 1) start the event to time
   2) read (or reset) the timer value
   3) enter a loop that continuously checks to see if the event is finished
   4) read the new timer value and subtract the initial value (overflows?)

B) 1) configure an interrupt to trigger when the event is complete
   2) start the event
   3) read (or reset) the timer value
   4) go to sleep and wait for the interrupt to trigger
   5) read the timer value and subtract the initial value (overflows?)



Event Timing – Dealing with overflows

Two strategies:

1) Guarantee that the event to be timed is never longer than one timer period (2^16 counts). How? Hardware prescale so that the counter counts slowly enough (set clock divider with IDx bits in TACTL)

PRO: might be easiest solution.
CON: might limit resolution.

2) Keep track of overflows (interrupts, or just by checking that a new value is less than the previous value).

PRO: get the full resolution possible
CON: more complicated code needed to track overflows

Configuring the Timer

12.3.1 TACTL, Timer_A Control Register

15	14	13	12	11	10	9	8
Unused						TASSELx	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
IDx		MCx		Unused	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Unused Bits 15-10 Unused

TASSELx Bits 9-8 Timer_A clock source select
- 00 TACLK
- 01 ACLK
- 10 SMCLK
- 11 INCLK (INCLK is device-specific and is often assigned to the inverted TBCLK) (see the device-specific data sheet)

IDx Bits 7-6 Input divider. These bits select the divider for the input clock.
- 00 /1
- 01 /2
- 10 /4
- 11 /8

MCx Bits 5-4 Mode control. Setting MCx = 00h when Timer_A is not in use conserves power.
- 00 Stop mode: the timer is halted.
- 01 Up mode: the timer counts up to TACCR0.
- 10 Continuous mode: the timer counts up to 0FFFFh.
- 11 Up/down mode: the timer counts up to TACCR0 then down to 0000h.

probably →

Unused Bit 3 Unused

TACLR Bit 2 Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero.

TAIE Bit 1 Timer_A interrupt enable. This bit enables the TAIFG interrupt request.
- 0 Interrupt disabled
- 1 Interrupt enabled

TAIFG Bit 0 Timer_A interrupt flag
- 0 No interrupt pending
- 1 Interrupt pending

SMCLK runs at ~1 MHz by default.

Then you can read/write TAR to set the counter to some specific value or find out what its value is.

Other registers are available to configure timer values to be stored (known as INPUT CAPTURE) and an interrupt triggered on external events (see TACCTLx in 12.3.4).

eg:

```
unsigned int start,end,difference;
start = TAR;

... stuff happens

end = TAR;

difference = end-start;
// as long as there is at most one overflow, this gives the right answer
// timer must be configured to count all the way to 0xFFFF (and not reset to 0 at CCR0)
```

General Advice:

Get pieces of a complicated system working one by one.

Ensure that each new piece doesn't break any old pieces.

Start with the simplest way, then if you have time and/or a good reason, move to a better way.

The debugger and oscilloscope can be very helpful!

12.3.1 TACTL, Timer_A Control Register

15	14	13	12	11	10	9	8
Unused						TASSELx	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
IDx		MCx		Unused	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

TASSEL_0 = 0
TASSEL_1 = 0x0100
TASSEL_2 = 0x0200
TASSEL_3 = 0x0300

Defined in msp430g2553.h

Unused	Bits 15-10	Unused
TASSELx	Bits 9-8	Timer_A clock source select
	00	TACLK
	01	ACLK
	10	SMCLK
	11	INCLK (INCLK is device-specific and is often assigned to the inverted TBCLK) (see the device-specific data sheet)
IDx	Bits 7-6	Input divider. These bits select the divider for the input clock.
	00	/1
	01	/2
	10	/4
	11	/8
MCx	Bits 5-4	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power.
	00	Stop mode: the timer is halted.
	01	Up mode: the timer counts up to TACCR0.
	10	Continuous mode: the timer counts up to 0FFFFh.
	11	Up/down mode: the timer counts up to TACCR0 then down to 0000h.
Unused	Bit 3	Unused
TACLR	Bit 2	Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero.
TAIE	Bit 1	Timer_A interrupt enable. This bit enables the TAIFG interrupt request.
	0	Interrupt disabled
	1	Interrupt enabled

Sensors

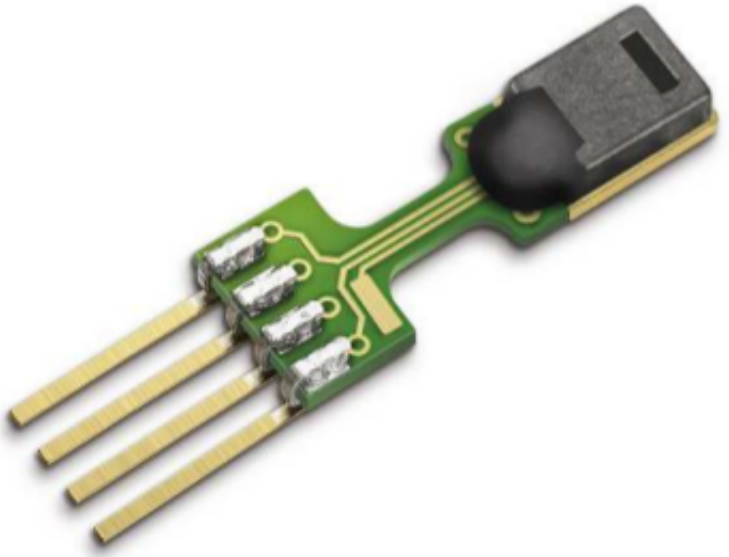
Piles of different kinds of sensors available:

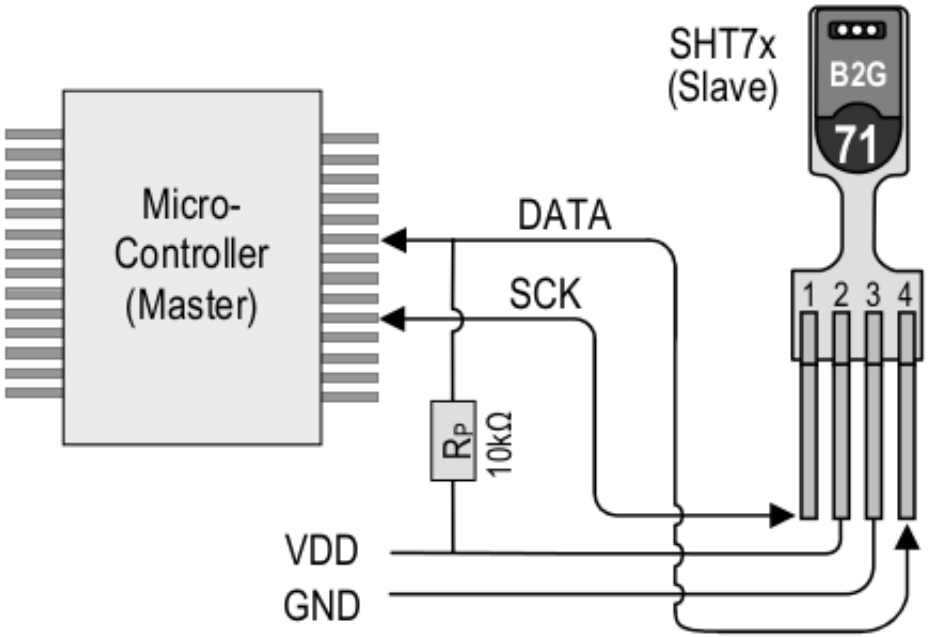
- optical sensors
- temperature
- humidity
- magnetic field
- pressure
- distance
- position and bend
- accelerometer
- gyroscope
- GPS

- servo motors, stepper motors, relays

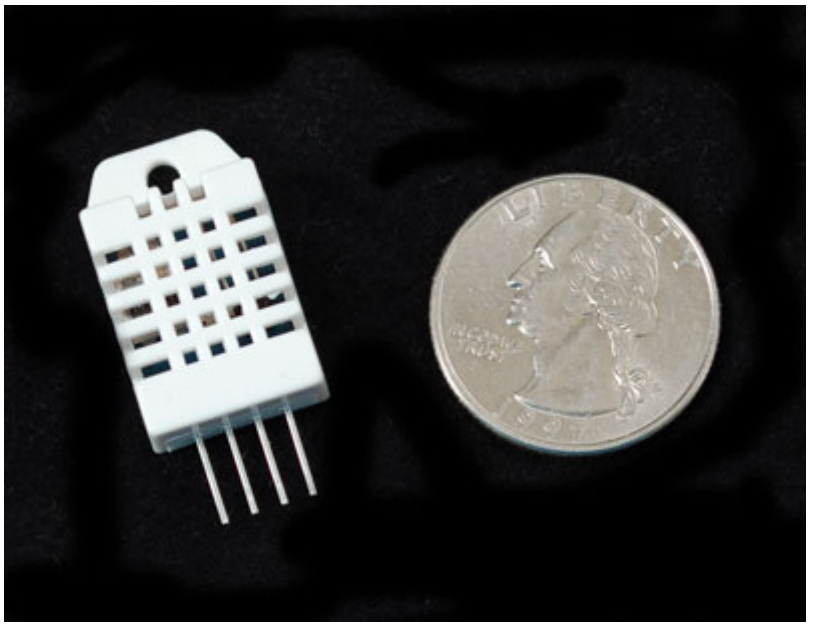
SHT75 Temperature and Humidity Sensor

- Fully Calibrated
- Digital output
- Low power consumption
- Excellent long term stability
- two-wire serial interface.
- Accuracy +/- 0.5C





DHT22 Temperature and humidity sensor



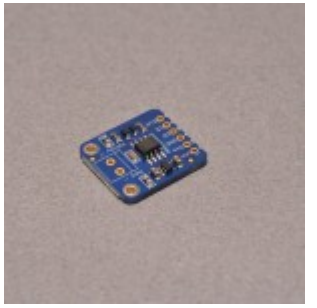
Accuracy	humidity +/-2%RH(Max +-5%RH);	temperature +/-0.2C
Resolution or sensitivity	humidity +/-0.1%RH;	temperature +/-0.1C
Repeatability	humidity +/-1%RH;	temperature +/-0.2C
Humidity hysteresis	+/-0.3%RH	
Long-term Stability	+/-0.5%RH/year	
Sensing period	Average 2s	
Interchangeability	fully interchangeable	

DS18B20 Temperature only

-Inexpensive,
- somewhat complicated interface
- high resolution 0.0625 degrees
- accuracy (+/- 0.5C)
- easy to multiplex many sensors

Also:

- Thermocouples
- Thermistors
- IR no contact sensor



MLX90614:

Distance Sensors:

- Optical:
  short range QRD1114,
  medium range GP2D12

- Ultrasonic

Magnetic Field Sensors:

- Hall effect
  on-off vs field measurement
- magneto-resistive
- magneto-inductive

eg: Devantech CMPS03 compass module

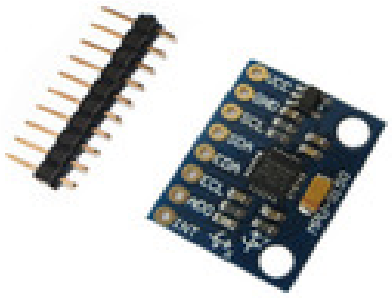
Position Sensor:

- Potentiometers (3/4 turn, 10 turn)
- Linear potentiometers

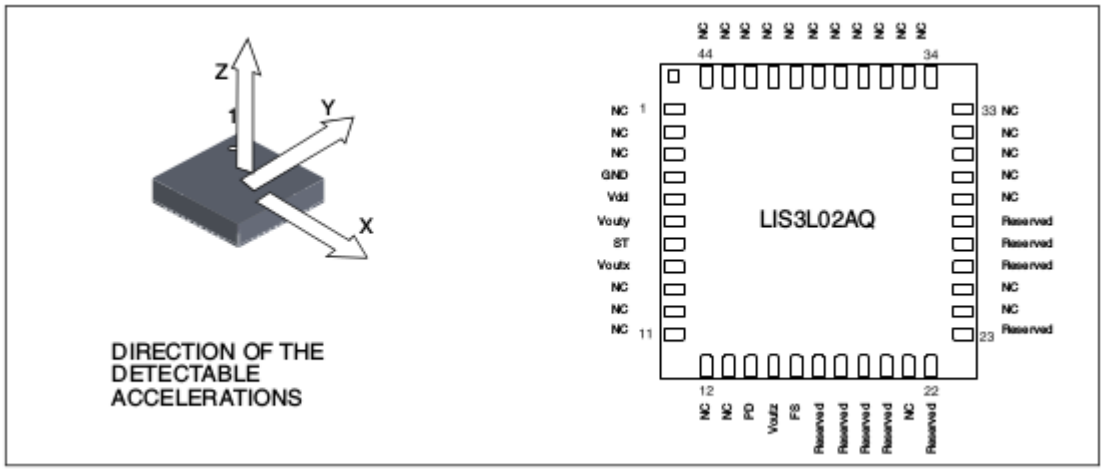
Accelerometers, Gyroscopes:

- 1 axis, 2 axis, 3 axis

EG: MPU9150: 3 axis gyroscope,
3 axis accelerometer+ 3 axis magnetic field
I^2C interface for ~$10.



EG: LIS3L02AQ – 3 axis accelerometer with analog outputs.



Motors, Servos, etc

- Servo motors
- AC and DC motors
- Stepper motors
- Solenoids
- Relays
- Solid-state relays

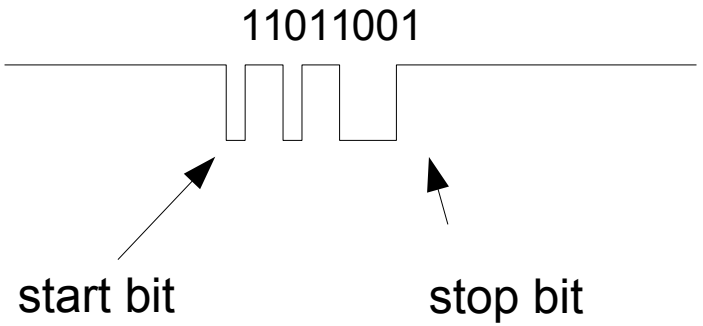
Pressure Sensors

- Gas or liquid (MPX5100)
- Mechanical (IESF-R-5L)

Many sensors use “standard” interfaces such as I^2C (inter-integrated circuit) or SPI (serial peripheral interface) to talk to the microcontroller.

The MSP430 has a module that can ease using these interfaces (Universal Serial Communication Interface, USCI, which can speak: I^2C, SPI and UART).

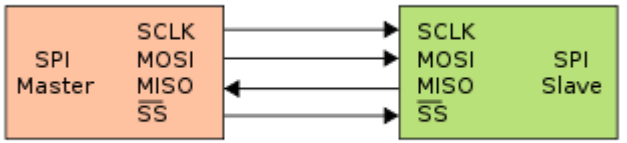
UART: Universal Asynchronous Receiver/Transmitter



- 3 lines for bi-directional communication: ground, transmit, receive
- start bit is always low, stop bit is always high.
- usually have 8 data bits in between, (but sometimes 5 or 6 or 7)
- least significant bit first, most significant bit last
- sometimes there is parity bit after the data

The USCI can output bytes and decode incoming bytes. Our example programs don't use the USCI, because the pins aren't compatible with the layout of the Launchpad board(!)

SPI: Serial Peripheral Interface

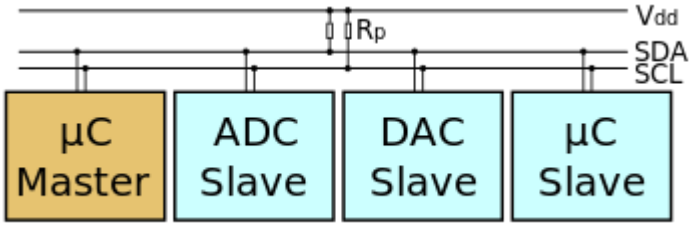


SCLK: clock from master
MOSI: Master out, slave in
MISO: Master in, slave out.
Slave Select

On every toggle of the clock, bits are transmitted in both directions, though not always useful. Communications controlled completely by the master.

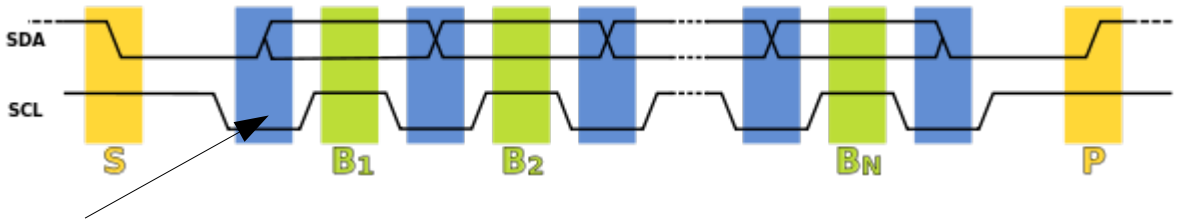
point – point, one master, one slave.

I2C: Inter-integrated Circuit



SDA – serial data
SCL – serial clock

Both lines are open-drain, pulled up with pull-up resistors



data line changes when clock is held low.

I2C is a bus: can be multiple masters, multiple slaves on the bus.

Activity 4

Complete the C program below so that it will:
1) configure pin P1.0 as an output and P1.3 as an input.
2) then enter a loop that continuously reads the P1.3 value. Each time the program sees a change from Low to High, it should toggle the P1.0 output.

```
#include <msp430.h>
int main(void){

    WDTCTL = WDTPW + WDTHOLD;


    while(1){




    }
}
```