Date: _____ Section: _____

Names: _____ and _____

LABORATORY CHECK OFFS	
Section 1 Pin 11 LED on for 2 seconds, off 50msec. with `#define`: Bytes program storage=_____, Bytes of dynamic memory=_____ with `int`: Bytes program storage=_____, Bytes of dynamic memory=_____	☐
Section 2 LED turns on only when switch is pressed. Approximate kohm value of the ATMEGA328's internal pullup resistor =_____ (datasheet)	☐
Section 3 Demo that switch decoding isSwChange, isSWPressed, etc. works as expected.	☐
Section 4 Fast serial print has correct values when finger moves close to sensor	☐
Section 5 Simple state machine to print analog sensor value when isSwPressed. Exit housekeeping is done.	☐
Section 6 Serial.println values of STOP, LED_ON, LED_OFF are _____,_____,_____ Demo behavior if (isSwJustReleased) changed to if(isSwPressed).	☐
Section 8 The sequence of states {LED_OFF, BLINK_G, BLINK_R, BLINK_GR, BLINK_RATE}; BLINK_R prints correct statename.	☐
Section 9 Microseconds for high pulse using `digitalWrite()` = _____ Microseconds for high pulse using `PORTB` = _____	☐
Points	☐

Prelab: (80 minutes)

1) Read the lab handout with a focus on the sections on state machines, entry and exit conditions, and the switch case statement. (40 minutes)

2) Write the code for section 1 and section 2. You will test the code in lab. (20 minutes)

3) Prepare the hardware: (20 minutes)

a) Solder the header pins to the QTR-1A infrared reflectance sensor chip. See images in the handout below,

b) cut the wire leads on the LEDs and make jumper wires as shown in the protoboard section.

4) Bring your kit to lab - bread board shield, switches, leds, Arduino board and USB programming cable.

Learning Outcomes for this lab:

At the end of this lab:

1) You will be able to configure a microcontroller's pins to act as inputs or outputs using the `pinMode()` command to match the pin direction (input/output) to the externally connected circuitry.
2) You will be able to configure a microcontroller's pin be an input and to have an internal pullup resistor (internal to the microcontroller), using the command `pinMode(PIN_NAME, INPUT_PULLUP);`
3) You will be able to configure an I/O pin as a power supply source to a sensor by setting the pin as an output that is HIGH or as a ground for a sensor by setting the pin to an output that is LOW.
4) You will be able to read a digital input to see whether it is at logic level HIGH or LOW, using the command `digitalRead(SW1_PIN);`
5) You will be able to determine the state of a switch (pressed, not pressed) or transitions of a switch from one state to another (just pressed, just released, change).
6) You will be able to change the sampling interval for checking the status of a switch by changing the delay in the main loop using `delay(MSEC_SAMPLE);`.
7) You will be able to define a variable representing the state of a switch (i.e. whether it is pressed or not), where the variable is HIGH/true when the switch is pressed, even if the switch closure causes the voltage on the pin itself to have a LOW logic value. For example: `isSwPressed = !digitalRead(SW1_PIN),`
8) You will be able to use a state machine structure to organize your code and to change how the microcontroller system behaves, depending upon what state it is in.
9) You will be able to create an enumerated datatype to define all the possible states in a state machine.
10) You will be able to use a switch case structure for state machines instead of an if-then-else structure.

11) You will be able to define the behavior of each state in a state machine, where each state uses the same code structure/pattern of state entry housekeeping, state business, state exit housekeeping.
12) You will be able to define a state machine where the values of analog and digital inputs are read in one common code location in the code (input scan and signal conditioning)
13) You will be able to read an analog voltage using the software command `analogRead().`
14) You will be able to display analog value measurements on the serial monitor using `Serial.println();` and control update rate to the serial monitor is readable by humans by adding a `delay()` statement to the main loop.
15) You will be able to reduce the amount of RAM and FLASH memory used by a program by choosing to define constants using `#define` or `const int` instead of `int`.
16) You will be able to directly change the output level (HIGH/LOW) of a digital pin using the `PORT` command, which executes much faster than a `digitalWrite();` command.
17) You will be able to use direct bit manipulation using bit-wise AND and OR operations and a bit mask e.g.

```
PORTB = PORTB | 0b00000001; // set pin HIGH
PORTB = PORTB & 0b11111110; // set pin LOW
```

Where both `0b00000001;` and `0b11111110;` are bit masks.

18) You will be able to use an oscilloscope to measure how fast the output level on a digital pin can be changed using different software commands in code (`PORTB=` vs `digitalWrite();`).

How to Succeed With This Lab:

Have the protoboard circuit ready. Read the lab handout. Refer back to the previous lab handout for sample code. Utilize the Arduino.cc website for reference information on Arduino pin definitions and language. Work together with your partner.

The number one problem students face doing this lab:

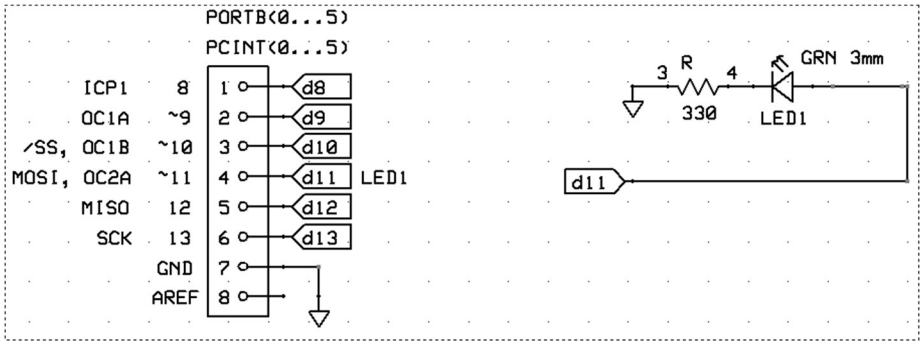
Mis-wiring the circuit, either by putting the wires into an adjacent pin on the Arduino board or on the protoboard.

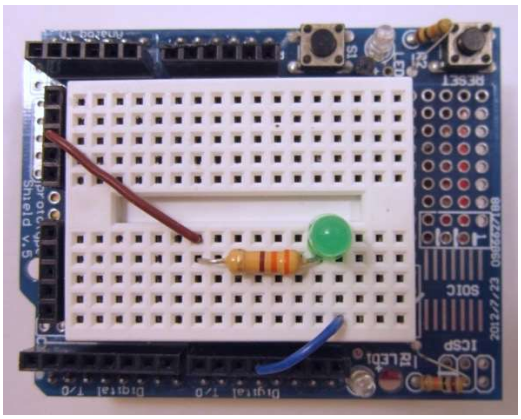
SECTION 1 - Writing Code to Connect an External LED

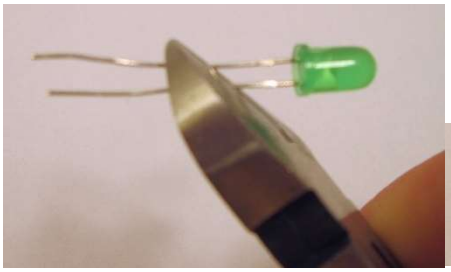
Procedure:

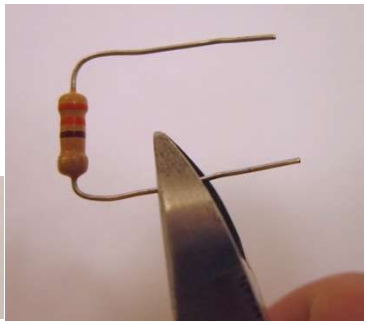
1) Connect a green LED and 330 ohm current limiting resistor to pin 11 of the Arduino and to Ground as shown in the schematic below. Note the orientation of the LED i.e. which way current must flow to turn on the LED. Mount the LED and resistor in the small protoboard shield exactly as shown in the photos below.
    a. First, clip the leads on the green LED shorter but cut the leads at a diagonal as shown in the photo below so that the longer lead remains longer. This will make it clear which lead should have positive voltage on it (the longer lead) to turn the LED on.

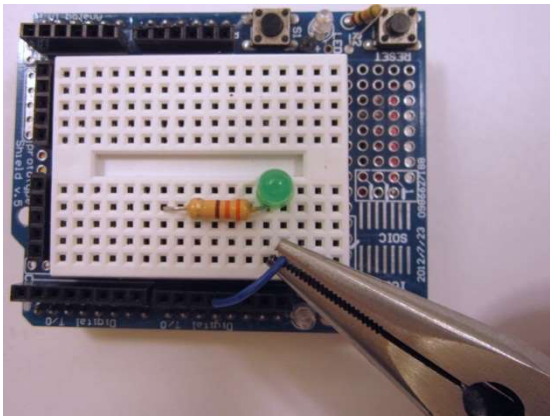
b. Clip the leads on the resistor short so that all the components and wires are low, flat, and tight to the protoboard. You may find that it is helpful to use a needlenose plier to insert the wires into the protoboard.











2) Write a program that turns the LED on for 2 seconds then off for 50 milliseconds repeatedly.

```
// Incomplete code snippet – modify as needed

//#define LED1_PIN 11
//int LED1_PIN=11;

void setup() {
  pinMode(LED1_PIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_PIN, HIGH);
  delay(2000);
  digitalWrite(LED_PIN, LOW);
  delay(50);
  LED1_PIN=11; //leave this line of code in when testing "int" declaration
               // comment this line out when using #define
}
```

3)

4) In your code, you may have used a `#define` statement or an `int` statement to declare the LED pin. When you compile your code, write down the amount of FLASH memory your code takes (look in the black status window at the bottom of the IDE screen). Now switch from `#define` to `int` or vice versa. Recompile and record how many bytes of RAM are used (dynamic memory) and how much program storage memory is used. What is the difference in bytes of RAM and which version of code version used less flash memory?

5) Demonstrate the operation of your program to the lab instructor and get a sign-off.

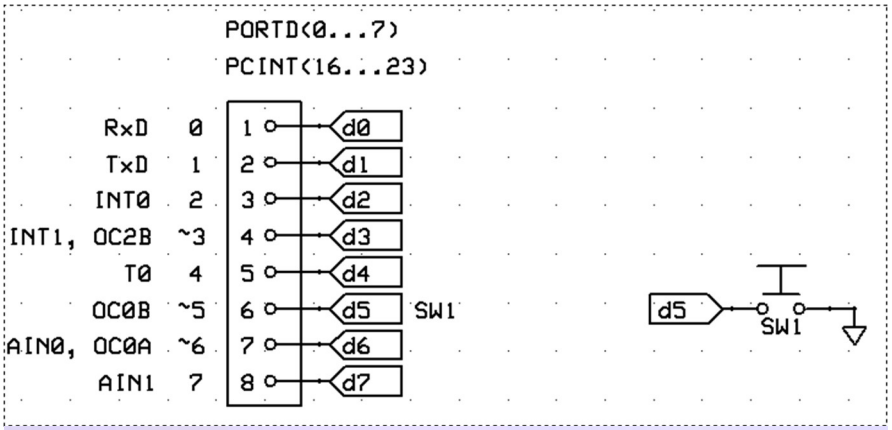
SECTION 2 - Adding a Pushbutton to Turn on LED

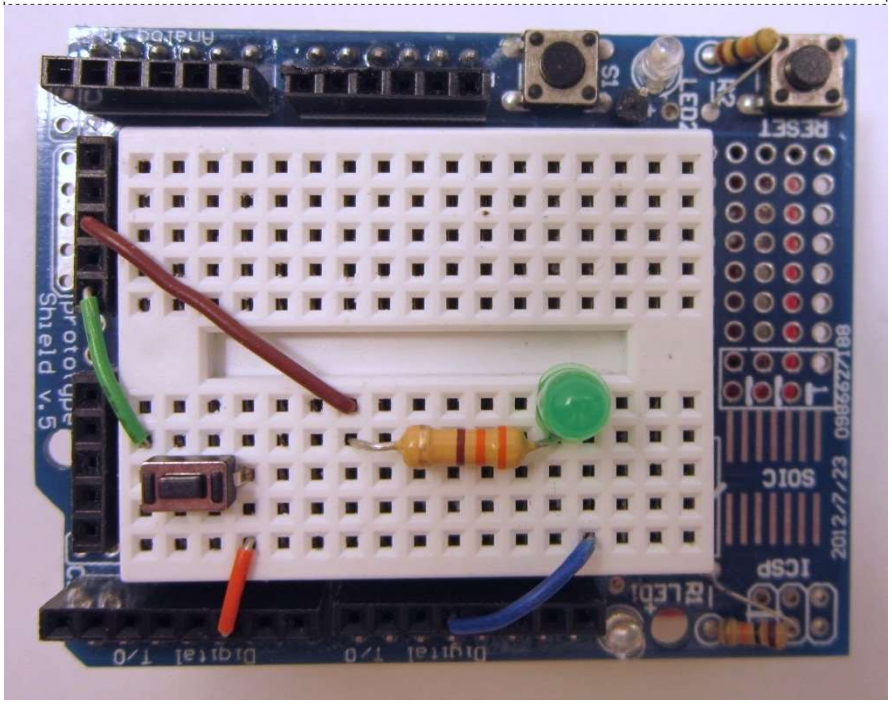
So far, you have controlled the LED using a timer and in the previous lab by input from the PC via the serial monitor window. You can also control the operation of the LED using a push button switch. The microcontroller software will have to be changed to tell the microcontroller where and how the switch is connected.

Procedure:

1) Install the switch in the protoboard, exactly as shown, and connect the jumper wire from one side of the switch to GND on the proto board. Connect the other side of the switch to PIN 5 using a jumper.

2)





6) Modify your code so that the microcontroller knows where the switch is connected. You will need to put the following code snippets into the correct location in the code for it to work (either declarations in, `setup ()` or in `loop()`).

```
#define SW_PIN 5
```

7) To read whether the switch is pressed or not, we need to add a pullup resistor. When the switch is open the pullup resistor makes the digital input value HIGH. When the switch is pressed, the switch connects the digital input to ground creating a digital input value of LOW. **The ATMEGA328, like many microcontrollers, has internal pullup resistors that are under software control.** To turn on the pullup resistor, insert the following code.

```
pinMode(SW1_PIN, INPUT_PULLUP);
```

8) Now change the main code, removing the blink delay statements and turn on the LED only when the switch is pressed. The digital input can be read using the command digitalread e.g.

```
isSwPressed = !digitalRead(SW1_PIN);
```

Tip: Read the online Arduino reference documentation about `digitalWrite()` and `digitalRead()`.

9) What is the value, in kOhms, of the pull-up resistor?

Tip: You will need to look up the value of the internal pull-up resistor in the ATMEGA 328p datasheet, posted on MyCourses. The value of the pull-up resistor can be found in section 31 of the datasheet. *Note that to figure out the approximate value of the resistor, you will have to read a graph and use Ohms law.* R= $\Delta V / \Delta I$

10) Demonstrate the operation of your program to the lab instructor and get a sign-off.

SECTION 3 - Examining the State of a Pushbutton

Beyond telling if the switch is pressed, there are times when you want to know if the switch has just been pressed or if it has just been released, or whether the switch has changed state. To do this, the code has to keep track of the switch's previous state (pressed or not pressed).

It becomes useful to not simply read the switch state, but to define a set of true/false (or boolean) variables to describe the switch state.

The first state could be, is the switch pressed? `isSwPressed`

Then was the switch previously pressed? `prevIsSwPressed`

Has the switch state changed ? `isSwChange`

Has the switch state just been pressed? `isSwJustReleased`

Has the switch state just been released? `isSwJustPressed`

These can be declared as boolean (true/false) variables as follows:

```
boolean isSwPressed, prevIsSwPressed, isSwJustReleased, isSwJustPressed,
isSwChange;
```

It makes sense to sample the switch state at a regular interval of time and to keep the LED on for long enough to be able to see what switch event has occurred. To do this, the code defines a sample period of 100 milliseconds. This is how long the code waits, with the LED on or off, until it checks again for the switch status. The time value is declared as follows.

```
#define MSEC_SAMPLE 100
```

And then delay is written as:

```
delay(MSEC_SAMPLE);
```

Code Snippet (Incomplete)

```
#define SW1_PIN 5
#define LED1_PIN 11

void setup() {
  pinMode(SW1_PIN, INPUT_PULLUP);
  pinMode(LED1_PIN, OUTPUT);  digitalWrite(LED1_PIN, LOW);

  Serial.begin(9600);
  Serial.println(F("Lab 2: Switch State Decoding v0.0\n"));
}

void loop() {
  prevIsSwPressed = isSwPressed;
  isSwPressed = !digitalRead(SW1_PIN);
  // When the switch is pressed, the SW_PIN is low, so !low is high or true
  isSwJustPressed = (isSwPressed && !prevIsSwPressed);  // switch edge detection
  isSwJustReleased = (!isSwPressed && prevIsSwPressed);
  isSwChange = (isSwJustReleased || isSwJustPressed);

// uncomment just one line below at time to see how each input
//  condition is detected. (enable just one output function at a time)

//  digitalWrite(LED_PIN, isSwPressed);
  digitalWrite(LED1_PIN, isSwJustReleased);
//  digitalWrite(LED_PIN, isSwJustPressed);
//  digitalWrite(LED_PIN, isSwChange);
//  digitalWrite(LED_PIN, !digitalRead(LED_PIN)); // see the sample rate

  delay(MSEC_SAMPLE);
}
```

Procedure:

1) Starting from the code in the box above, add declarations for the boolean variables and the sample period. Upload the code (CTRL+U) and fix any errors. You can copy the code from the box above and paste it directly into the Arduino IDE.
2) Slowly press and release the switch and observe the how the LED behaves relative to the switch state (pressed or released).
3) Modify the code by uncommenting only one of the `digitalWrite()` commands at a time and observe how detecting the just pressed, just released, changing, or pressed state of the switch gives a different LED behavior.
4) Demonstrate the operation of your program to the lab instructor and get a sign-off.

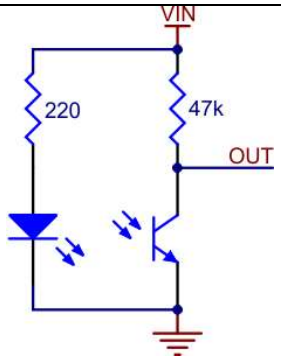
SECTION 4 - Using Digital I/O Pins to Supply Power and Ground to a Sensor and Reading an Analog Sensor Input

In this section, an analog reflectance sensor is added to the microcontroller. The reflectance sensor can be used to detect an object, as well as to indicate the distance to an object.

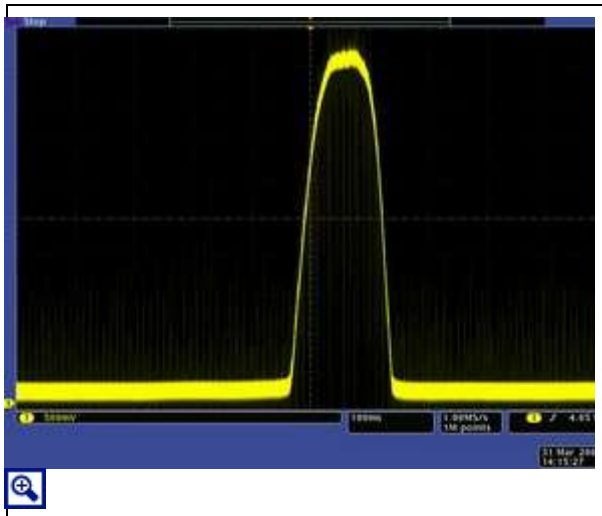
Description of sensor:

The reflectance sensor has an infrared light emitting diode paired with a photo transistor that is sensitive to infrared light. The LED sends out a beam of light and if there is an object nearby, the light is reflected off the object, back into the photo transistor which detects the amount of reflected light. In a distance sensing application, more reflected light means the object is closer. The figures below show the sensor, its schematic, its operating parameters, and scope traces of the sensor in operation. Note that sensor output is an analog voltage. An object that is white reflects infrared, while an object that is black absorbs infrared (less reflected light).

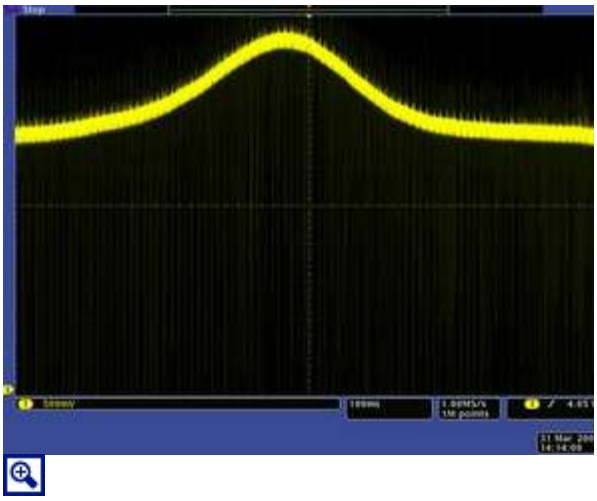
REFLECTING LIGHT = LOW VOLTAGE AT PIN, NO REFLECTION = HIGH VOLTAGE AT PIN



Operating voltage: 5.0 V
Supply current: 17 mA
Output format: analog voltage
Output voltage range: 0 to supplied voltage
Optimal sensing distance: 0.125" (3 mm)
Maximum recommended sensing distance: 0.25" (6 mm)



QTR-1A output 1/8" away from a spinning white disk with a black line on it.

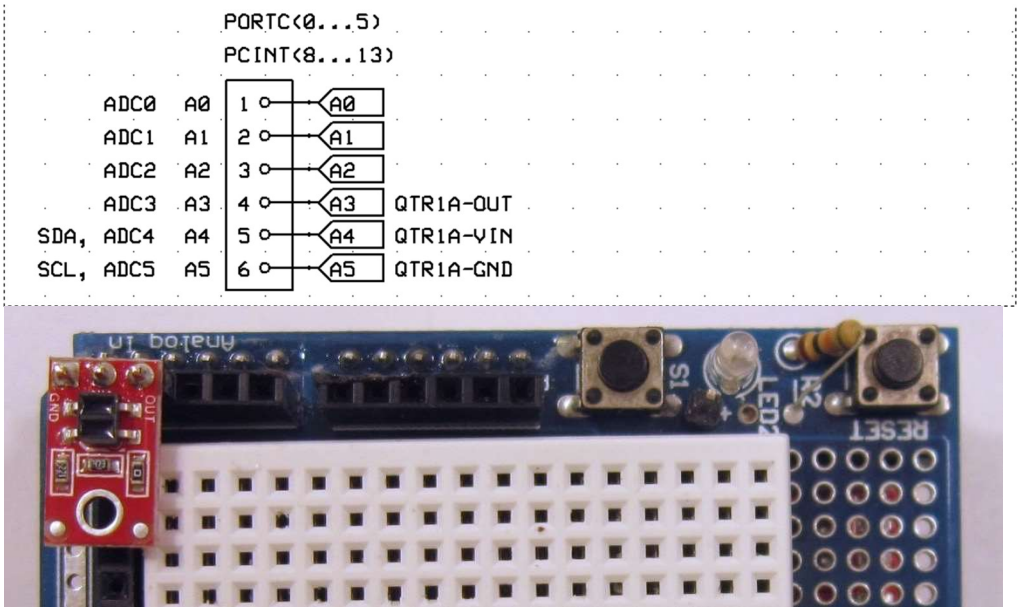


QTR-1A output 3/8" away from a spinning white disk with a black line on it.

Note that in the specifications, the QTR sensor only requires 17 ma of current to run. That is low enough that a digital pin on the microcontroller can power it. In fact, that is what we will do in this section. The +5V power for the sensor will come from a digital pin. The ground for the sensor will come from a second digital pin and the analog output voltage will be read from the built in analog to digital converter in the ATMEGA328.

Procedure:

1) First install the QTR reflectance sensor on the Arduino board as shown



2) The output pin of the QTR sensor is connected to PIN A3 on the Arduino board. This will be the analog input signal to the ATMEGA 328 analog to digital converter. The ATMEGA 328 has a 10-bit ADC that converts an analog voltage ranging from 0 volts to +5 volts into a digital number from 0 to 1023. (A 10-bit converter means there are 2^10 = 1024 possible output values.) An input voltage of 2.5 volts dc to the ADC would read in the code as a digital value of around 512. A value of 1.25 volts dc at the ADC would be a reading of 256 (`adcQTR = analogRead(QTR_SIG_PIN);`)

3) The code to tell the microcontroller that there is an analog voltage on this pin is: `pinMode(QTR_SIG_PIN, INPUT);` If this is not done, there is a risk that the input pin may have a pullup resistor attached to it internally and that could affect the sensor reading.

4) The code to tell the microcontroller to set PIN A5 to be a virtual ground (zero volts) is

   `pinMode(QTR_GND_PIN, OUTPUT); digitalWrite(QTR_GND_PIN, LOW);` This code does two things. It sets the pin as an output, then it writes a low value to the pin bringing the pin to 0 volts (nearly the same as ground).

5) The code to tell the microcontroller to set PIN A4 to be at +5 volts is:

   `pinMode(QTR_5V_PIN, OUTPUT); digitalWrite(QTR_5V_PIN, HIGH);` This code does two things. It sets the pin as an output, then it writes a high value to the pin bringing the pin to 5 volts.

6) Add the `pinMode()` definitions to the code snippet below and observe the value of the ADC reading in the serial monitor window. Place your finger above the sensor and bring it slowly closer and watch the ADC reading value change.

7) Right now the serial monitor is printing the values very quickly and it is hard to read. Add a `delay()` statement in the main loop so that the loop waits `MSEC_SAMPLE` milliseconds before running again. This will make it easier to see the values.

8) Demonstrate the operation of your program to the lab instructor and get a sign-off.

9) Do the results seem reasonable?

10) In addition to printing values to the Serial Monitor, it is possible for the Arduino IDE to plot the numeric values coming from the Arduino. Just for fun, try using the graphing tool in the Arduino

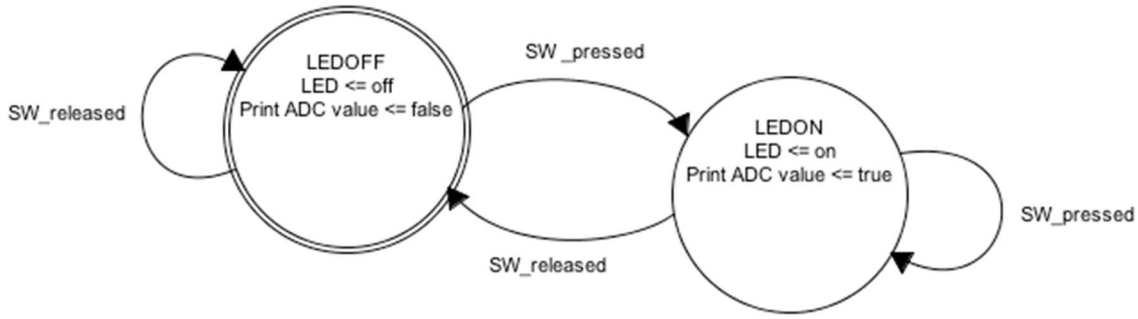
IDE. Go to Tools→ Serial Plotter and then move your finger up and down above the QTR reflectance sensor and watch the plotted values change.

Code Snippet (Incomplete)

```
#define SW1_PIN 5
#define LED1_PIN 11
#define QTR_SIG_PIN A3
#define QTR_5V_PIN A4
#define QTR_GND_PIN A5

#define MSEC_SAMPLE 200

boolean isSwPressed;
unsigned int adcQTR;

void setup(){
  pinMode(SW1_PIN, INPUT_PULLUP);

  pinMode(LED1_PIN, OUTPUT);  digitalWrite(LED1_PIN, LOW);

  pinMode(QTR_SIG_PIN, INPUT);
  //
  //

  Serial.begin(9600);
  Serial.println(F("Lab 2: Analog Sensor Reading\n"));
}

void loop(){
  // scan input and condition it (low to hi true)
  isSwPressed = !digitalRead(SW1_PIN);
  digitalWrite(LED1_PIN, isSwPressed);

  adcQTR = analogRead(QTR_SIG_PIN);  // 0..1023 output
  Serial.println(adcQTR);
} // loop()
```

SECTION 5 - Creating a simple state machine.

1) Instead of printing the ADC readings all the time, modify the code from the last section so that the ADC readings are only printed when the switch is pressed. Use an if-statement to do this. Also when the switch is pressed, turn on the LED. When the switch is not pressed, turn off the LED. All of this can be done with one if-else statement.
2) **STATE MACHINE:** What you have just built is a very simple state machine-- what the code does depends upon the state of the switch. First, if the switch is pressed, you enter a state where the LED is on and serial data is being printed to the serial monitor (or console).
3) **EXIT HOUSEKEEPING** When you leave the switch pressed state, you turn off the LED. Turning off the LED is part of the exit housekeeping that you must do. If you don't turn off the LED, then it will stay on even after you release the switch and have left the printing state. A diagram of the state machine is shown below with two states (LEDOFF, LEDON).



4)
5) Upload and test your modified code. Demonstrate the operation of your program to the lab instructor and get a sign-off.

SECTION 6 - A State Machine Using Enumerated States

The idea of using state machines so valuable when writing microcontroller programs that we will use it often throughout the rest of the course. The next several sections introduce you to the state machine in more depth. It is essential that you understand how state machines work for you to be able to complete later labs in this course.

Take your time and understand what is going on in each of the next sections.

Intended Function of the Program:

The next program has a simple function. When you press the button, the LED will flash on and off continuously. When you the press the button again, the LED will stop flashing. If you press the button again, the LED will start flashing again.

The program is organized around the concept of a state machine. There are three states of the system.

1) STOP - the system is stopped and the LED is off. Only a press and release of the button will start flashing.

2) LED_ON- the led is on. Whenever the LED_ON state is first entered, a timer is set to zero and starts counting up until 250 milliseconds is reached and then the state will change to LED_OFF. Note that the 250 milliseconds is determined by a timer, not a delay function.

From a state machine perspective, there are two ways to leave the LED_ON state. First, if the timer has reached 250 milliseconds, the state will change to LED_OFF. Second, if the button is pressed and released at any time, the state is changed to STOP.

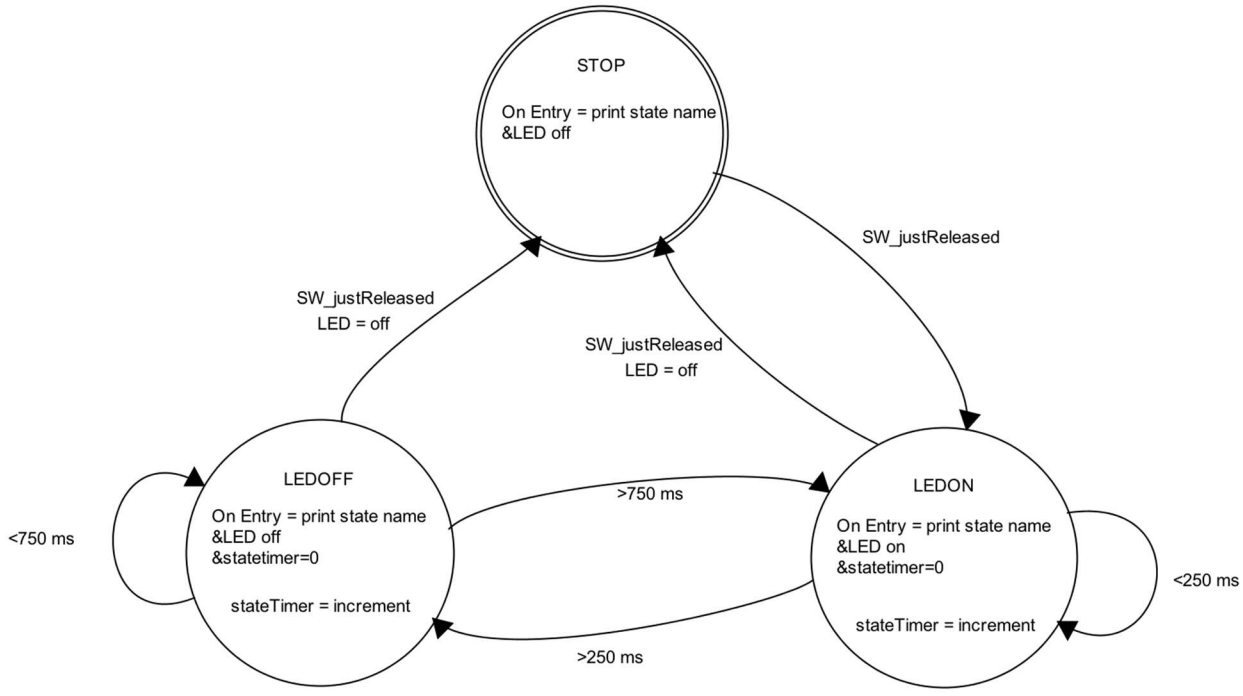
2) LED_OFF- the led is off. Whenever the LED_OFF state is first entered, a timer is set to zero and starts counting up until 750 milliseconds is reached and then the state will change to LED_ON. Note that the 750 milliseconds is determined by a timer, not a delay function.

From a state machine perspective, there are two ways to leave the LED_OFF state. First, if the timer has timed out, the state will change to LED_ON. Second, if the button is pressed and released at any time, the state is changed to STOP.

Procedure:

1) Copy and paste the code from the textbox and run it to examine how it behaves, then carefully read the description of the code below. You will be building state machines soon, so take the time to understand everything in this code. Try pressing the button once and watch the serial monitor for several seconds. Note what the LED is doing. Wait at least 10 seconds in each state. Then press the button again and watch the serial monitor. Each time you press the button, you are changing the state of the state machine. Notice too that the state machine is changing on its own from LED_ON to LED_OFF and back.

2)



```
#define SW1_PIN 5
#define LED1_PIN 11
#define MSEC_SAMPLE 1

enum {STOP, LED_ON, LED_OFF};

boolean isSwPressed, prevIsSwPressed, isSwJustReleased, isSwJustPressed, isSwChange;
int state = STOP, prevState = !state;
int stateTimer;
boolean isNewState;

void setup() {
  pinMode(SW1_PIN, INPUT_PULLUP);
  pinMode(LED1_PIN, OUTPUT);  digitalWrite(LED1_PIN, LOW);
  Serial.begin(9600);  Serial.println(F("Lab 2: if-then state machine\n"));
}

void loop() {
  // Input scan, signal conditioning, history evolution
  prevIsSwPressed = isSwPressed;
  isSwPressed = !digitalRead(SW1_PIN);
  isSwJustPressed = (isSwPressed && !prevIsSwPressed);  // switch edge detection
  isSwJustReleased = (!isSwPressed && prevIsSwPressed);
  isSwChange = (isSwJustReleased || isSwJustPressed);

  // update state information
  isNewState = (state != prevState); // if state != prevState, then isNewState
  prevState = state;

  if (state == STOP) {
    // Entry housekeeping
    if (isNewState) Serial.println("STOP");
    // State business
    digitalWrite(LED1_PIN, LOW);
    // Exit condition is based only on switch changing
    if (isSwJustReleased) state = LED_ON;
  }

  else if (state == LED_ON) {
    // Entry housekeeping
    if (isNewState) {
      stateTimer = 0; // reset state timer to zero
      Serial.println("LED_ON");
      digitalWrite(LED1_PIN, HIGH);
    }
    // State business
    stateTimer++;
    // Exit condition 1 is based on switch changing
    if (isSwJustReleased) {
      digitalWrite(LED1_PIN, LOW);
      state = STOP;
    }
    // Exit condition 2 is based on timer expiring
    if (stateTimer >= 250) state = LED_OFF;
  }

  else if (state == LED_OFF) {
    // Entry housekeeping
    if (isNewState) {
      stateTimer = 0; // reset state timer to zero
      Serial.println("LED_OFF");
      digitalWrite(LED1_PIN, LOW);
    }
    // State business
    stateTimer++;
    // Exit condition 1 is based on switch changing
    if (isSwJustReleased) {
      digitalWrite(LED1_PIN, LOW);
      state = STOP;
    }
    // Exit condition 2 is based on timer expiring
    if (stateTimer >= 750) state = LED_ON;
  }
  else state = STOP;

  delay(MSEC_SAMPLE);
} //loop()
```

Description of the Code

The code uses an enumerated datatype to define the three states:

```
enum {STOP, LED_ON, LED_OFF};
```

3) In the `setup()` section, add three serial print statements as follows:

```
Serial.println(STOP);
Serial.println(LED_ON);
Serial.println(LED_OFF);
```

What values are printed? Record these on the signoff sheet. Once you have these values, delete the print statements.

The code defines a variable called `stateTimer` that counts up by one every time through the look. At the end of the code there is a delay statement that waits for 1 millisecond. This creates an approximately 1 millisecond clock for the state machine. This clock is not very accurate and in future labs you will make a much more accurate timer using interrupts.

A set of if-then-else statements is used to determine which state the code is in. The general pattern for each state is the same. There are things that must be done only when the state is first entered, called **entry housekeeping**. There are tasks, called **state business**, that are done everytime that the code goes through the loop and is in that particular state. Finally, if an exit condition is met, and the code is leaving the state, then there are **exit housekeeping** tasks that are done.

The entry housekeeping tasks include setting the timer back to zero for the state, writing the name of the state to the serial port and turning the LED on or off. Entry tasks are done only when first entering the state. This is done only once by checking the boolean variable called `isNewState`. The value of `isNewState` is true only when the current state does not equal the previous state. Note that the expression below in parenthesis evaluates to either true or false.

```
isNewState = (state != prevState);
```

The state business tasks include incrementing the stateTimer by one.

```
stateTimer++;
```

The exit tasks include turning the LED on or off, and very importantly, changing the state to whatever the new state should be based on the exit condition. For example, if the state exited because the button was pressed, then the state must be set equal to STOP. If the state exited due to a timer, the state needs to be set to either LED_ON or LED_off.

It may seem like a subtle aspect of the exit condition, but note that the exit condition does not check for whether the switch is pressed, instead it checks for `isSwJustReleased`. The difference is important. Try changing the exit condition for all three states to be `isSwPressed` and observe what happens.

Procedure (part 2):

4) Change the exit condition in all three states from `if (isSwJustReleased)` to be `if(isSwPressed)`. Demonstrate the resulting operation to the lab instructor.

SECTION 7 - Using SWITCH CASE Statements for State Machines

When implementing state machines, it is much better coding practice to use a SWITCH CASE structure instead of a large set of if-then-else statements. To illustrate this, consider the code below the performs the same function as before, but uses a switch case structure. Each case clearly represents a state. Within a state, the entry, state business, and exit actions can be seen. Each case ends with a `break` statement. The entire switch statement ends with a `default` case.

```
#define SW1_PIN 5
#define LED1_PIN 11
#define MSEC_SAMPLE 1

enum {STOP, LED_ON, LED_OFF};

boolean isSwPressed, prevIsSwPressed, isSwJustReleased, isSwJustPressed, isSwChange;
int state = STOP, prevState = !state;
int stateTimer;
boolean isNewState;

void setup() {
  pinMode(SW1_PIN, INPUT_PULLUP);
  pinMode(LED1_PIN, OUTPUT);  digitalWrite(LED1_PIN, LOW);
  Serial.begin(9600);  Serial.println(F("Lab 2: switch-case state machine\n"));
}

void loop() {
  // Input scan, signal conditioning, history evolution
  prevIsSwPressed = isSwPressed;
  isSwPressed = !digitalRead(SW1_PIN);
  isSwJustReleased = (isSwPressed && !prevIsSwPressed);  // switch edge detection
  isSwJustPressed = (!isSwPressed && prevIsSwPressed);
  isSwChange = (isSwJustReleased || isSwJustPressed);

  // update state information
  isNewState = (state != prevState); // if state != prevState, then isNewState
  prevState = state;

  switch (state) {
    case STOP:
      if (isNewState) { // Entry
          Serial.println("STOP");
          digitalWrite(LED1_PIN, LOW);
      }
      // No business to do for the STOP state
      if (isSwJustReleased) state = LED_ON; // Exit
      break;

    case LED_ON:
      if (isNewState) {  // Entry
        stateTimer = 0;
        Serial.println("LED_ON");
        digitalWrite(LED1_PIN, HIGH);
      }
      stateTimer++; // State Business
      if (isSwJustReleased) { // Exit 1
        digitalWrite(LED1_PIN, LOW);
        state = STOP;
      }
      if (stateTimer >= 250) state = LED_OFF; // Exit 2
      break;

    case LED_OFF:
      if (isNewState) { // Entry
        stateTimer = 0;
        Serial.println("LED_OFF");
        digitalWrite(LED1_PIN, LOW);
      }
      stateTimer++; // State Business
      if (isSwJustReleased) { // Exit 1
        digitalWrite(LED1_PIN, LOW);
        state = STOP;
      }
      if (stateTimer >= 750) state = LED_ON; // Exit 2
      break;

    default: state = STOP; // In switch case, always include a default
  } // switch-case
  delay(MSEC_SAMPLE);
} //loop()
```

SECTION 8 - Making a More Complex State Machine

Using what you have learned so far, it is possible to make the microcontroller to sequence through a complex set of behaviors when the button is pressed.

In this section, the state machine will have several states:

```
enum {LED_OFF, BLINK_G, BLINK_R, BLINK_GR, BLINK_RATE};
```

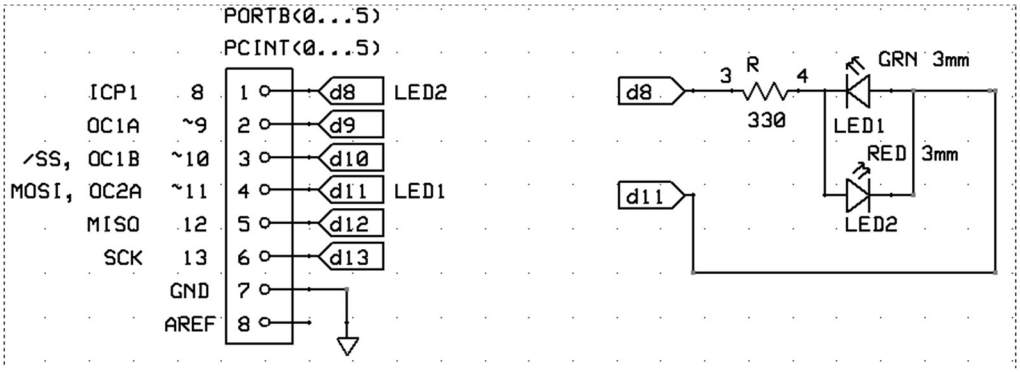
First the LED's are off (`LED_OFF`). Then when the button is pressed, the green LED will blink (`BLINK_G`). On the next button press, the red LED will blink (`BLINK_R`). On the next button press, the green and red LEDs will alternately blink (`BLINK_GR`). On the next button press, the green and red LEDs will alternately blink at a rate that changes when you place your finger near the QTR reflectance sensor (`BLINK_RATE`).

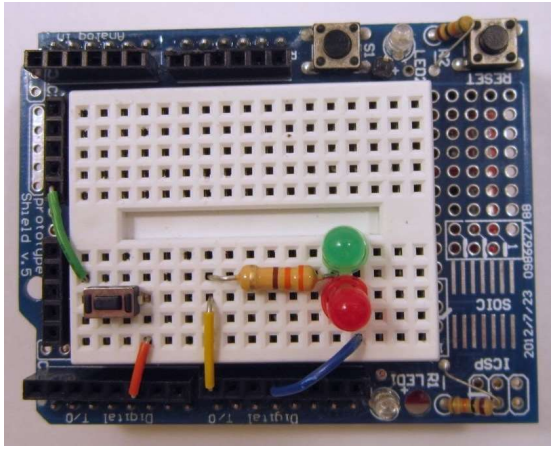
For now, the state for `BLINK_R` is empty.

Procedure:

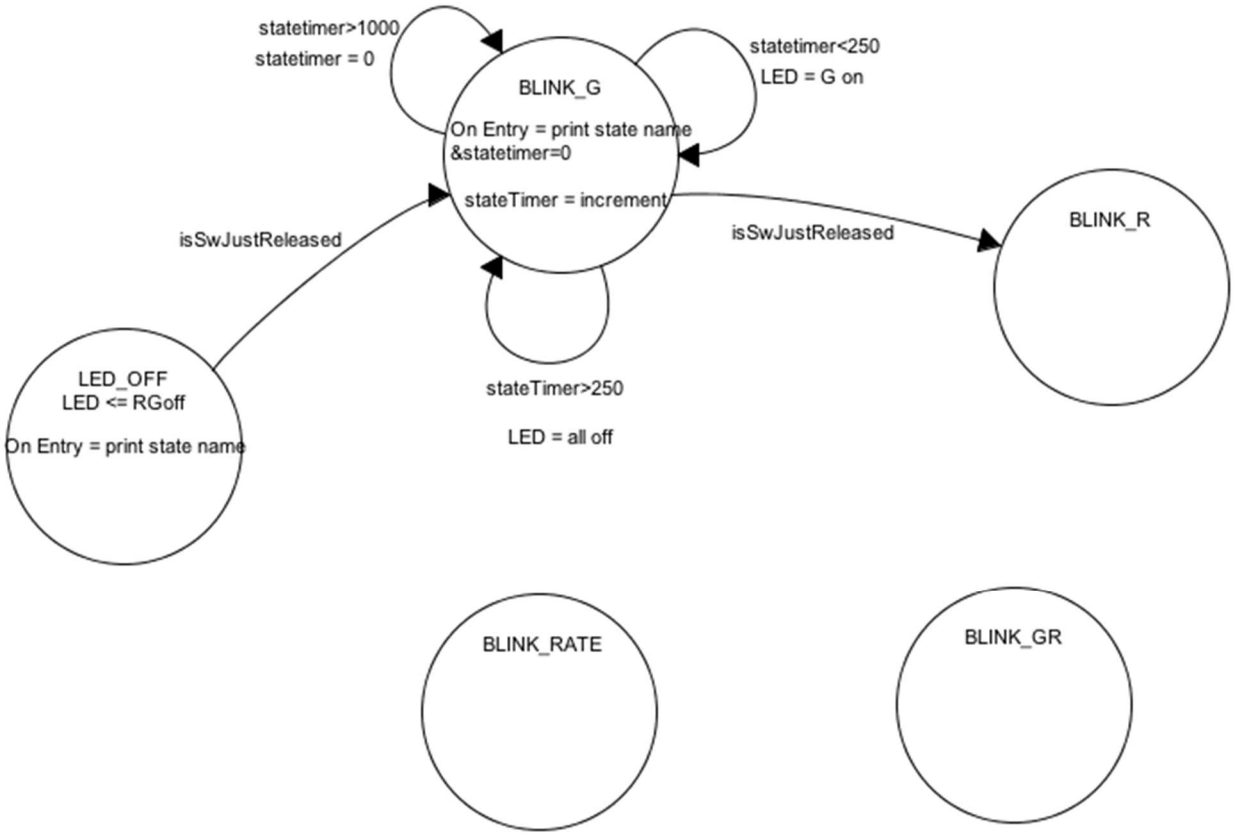
1) Modify the wiring so that the LEDs are connected to PIN8 and PIN 11. Refer to the schematic on the next page. Leave the green LED and resistor in the protoboard exactly as before. Add the red LED in parallel with the green LED, but place the red LED so that it is installed with the opposite polarity. Because the LED is truly a diode, current can only flow one way.

   Referring to the schematic, the green LED is turned on when digital PIN 11 is high and digital PIN 8 is low. When these are reversed, and PIN 11 is low and PIN 8 is high, the red LED is turned on. The LED's are connected back to back with a common current limiting resistor. Check that the LEDs are no longer connected to GND.





2) The code has the same general layout with sections for declarations, `setup()` and `loop()` but there are also function definitions below the main loop. These functions are called in the switch-case section and help to make the code more readable. Copy the code from the following **two text boxes.** The first text box has the declarations, setup and function calls. The second text box has the main loop(). Merge these files to make one program. Upload and run the code (CTRL+U) and open the serial monitor (CTRL+SHIFT+M). Notice how the serial monitor allows you to debug which state the program is currently in. Press the button and try out the different states.
3) Press the button to set the state to BLINK_RATE and place your finger near the QTR reflectance sensor. Notice how the blink rate changes in response to the analog input voltage from the reflectance sensor.
4) Notice that there is no BLINK_R state when you press the button. Edit the code and insert code for the BLINK_R state in the switch-case statement. You can copy the code from the BLINK_G state and then modify it so that blinks the red led. Because you have inserted a new state, you must modify the BLINK_G code slightly so that when the button is pressed to exit the BLINK_G state, that the next state is BLINK_R not BLINK_GR. Also, make sure that within the BLINK_R state that you print the state name to the serial port to help you debug the code. Make sure that you also turn on the correct LED in the BLINK_R state.
5) Demonstrate your new state machine to your lab instructor. The sequence of states should be {LED_OFF, BLINK_G, BLINK_R, BLINK_GR, BLINK_RATE};
6) A partial state diagram showing the state machine (with only BLINK_G state fully shown) is given below. Draw in the rest of the diagram for the BLINK_R state only.



State diagram showing state machine (with only BLINK_G state fully shown)

```
#define SW1_PIN 5
#define LED1_PIN 8
#define LED2_PIN 11
#define QTR_SIG_PIN A3
#define QTR_5V_PIN A4
#define QTR_GND_PIN A5
#define MSEC_SAMPLE 1

enum {LED_OFF, BLINK_G, BLINK_R, BLINK_GR, BLINK_RATE};

boolean isSwPressed, prevIsSwPressed, isSwJustReleased, isSwJustPressed, isSwChange;
int state = LED_OFF, prevState = !state;
int stateTimer, adcQTR;
boolean isNewState;

void setup() {
  pinMode(SW1_PIN, INPUT_PULLUP);//  pinMode(SW1_PIN, INPUT); won't work

  pinMode(LED1_PIN, OUTPUT);  digitalWrite(LED1_PIN, LOW);
  pinMode(LED2_PIN, OUTPUT);  digitalWrite(LED2_PIN, LOW);

  pinMode(QTR_SIG_PIN, INPUT);
  pinMode(QTR_5V_PIN, OUTPUT);   digitalWrite(QTR_5V_PIN, HIGH);
  pinMode(QTR_GND_PIN, OUTPUT);  digitalWrite(QTR_GND_PIN, LOW);

  Serial.begin(9600);
  Serial.println(F("Lab 2 Complex State Machine"));
} // setup()

// ADD loop() HERE - IT IS GIVEN ON THE NEXT PAGE

//*************************************************************************
void redOn(void) {
  PORTB = PORTB |(1 << 0);  // sets Uno dig_8, PORTB.0, pin to 1 (HIGH)
                            // physical pin 14 (28 pin DIP)
//  digitalWrite(LED1_PIN, HIGH);  // alternative to PORTB setting
  digitalWrite(LED2_PIN, LOW);
}

//*************************************************************************
void greenOn(void) {
  digitalWrite(LED1_PIN, LOW);
  digitalWrite(LED2_PIN, HIGH);
}

//*************************************************************************
void allOff(void) {
  digitalWrite(LED1_PIN, LOW);
  digitalWrite(LED2_PIN, LOW);
}
```

```
void loop() {
  prevIsSwPressed = isSwPressed;
  isSwPressed = !digitalRead(SW1_PIN);
  isSwJustPressed = (isSwPressed && !prevIsSwPressed);  // switch edge detection
  isSwJustReleased = (!isSwPressed && prevIsSwPressed);
  isSwChange = (isSwJustReleased || isSwJustPressed);

  isNewState = (state != prevState);
  prevState = state;

  switch (state) {

    case LED_OFF:
      if (isNewState) Serial.println("LED_OFF");
      allOff();
      if (isSwJustReleased) state = BLINK_G;
    break;

    case BLINK_G:
      if (isNewState) {
      stateTimer = 0;
      Serial.println("BLINK_G");
      }
      stateTimer++;
      if (stateTimer < 250) greenOn();
      else allOff();
      if (stateTimer >= 1000) stateTimer = 0;
      if (isSwJustReleased) {
        allOff();
        state = BLINK_GR;
      }
    break;

  // ADD BLINK_R STATE HERE. ************


    case BLINK_GR:
      if (isNewState) {
        stateTimer = 0;
        Serial.println("BLINK_GR");
      }
      stateTimer++;
      if (stateTimer < 500) redOn();
      else greenOn();
      if (stateTimer >= 1000) stateTimer = 0;
      if (isSwJustReleased) {
        allOff();
        state = BLINK_RATE;
      }
    break;

    case BLINK_RATE:
      if (isNewState) {
        stateTimer = 0;
        Serial.println("BLINK_RATE");
      }
      stateTimer++;
      adcQTR = analogRead(QTR_SIG_PIN);
      if (stateTimer < adcQTR/2) redOn();
      else greenOn();
      if (stateTimer >= adcQTR) stateTimer = 0;
      if (isSwJustReleased) {
        allOff();
        state = LED_OFF;
      }
    break;

    default: state = LED_OFF;
  } // switch (state)

  delay(MSEC_SAMPLE);
} // loop()
```

SECTION 9 – Faster Code Execution and Measurement of Code Speed

In the code so far, digital pins have been set to high or low states using the Arduino library function `digitalWrite()`. Using library functions like `digitalWrite();` makes the code easier to create and to read but the code does not execute as fast as using direct manipulation of the port registers.

In this section you will run a piece of test code that sets a pin high and then low using the `digitalWrite()` function and compares that to setting another pin high and low using direct manipulation of the port registers.

Code:

```
#define LED1_PIN 11   // Port B3
#define LED2_PIN 8    // Port B0

void setup() {
  pinMode(LED1_PIN, OUTPUT);  digitalWrite(LED1_PIN, LOW);
  pinMode(LED2_PIN, OUTPUT);  digitalWrite(LED2_PIN, LOW);

  Serial.begin(9600);
  Serial.println(F("Lab 2 Faster I/O"));
}

void loop() {

// Set pin high then low using slow library function

  digitalWrite(LED2_PIN,LOW);
  digitalWrite(LED2_PIN,HIGH);
  digitalWrite(LED2_PIN,LOW);

// Set pin high then low using fast direct setting of using bit manipulation

  // Three ways to do the same thing

  // Binary
  PORTB = PORTB | 0b00000001; // set pin HIGH
  PORTB = PORTB & 0b11111110; // set pin LOW

  // Hex
  PORTB |= 0x01; // set pin HIGH
  PORTB &= 0xFE; // set pin LOW

  // bit shifting
  PORTB |= (1 << 0); // set pin HIGH
  PORTB &= ~(1 << 0); // set pin LOW
}
```

Description of the code:

The code defines two pins as outputs and sets them to low values. The code then writes LED1_PIN to a low value then high value using the digital write function. Then the code sets LED2_PIN to low and then high using direct port manipulation. The same pin is written low then high using three equivalent methods, one using binary bit masks, one using hex bit masks and one using bit shift operations. In practice you will see all three methods used, as they represent different coding styles.

Procedure:

1) Copy the code and run it. Connect an oscilloscope's probes to PIN 8 and PIN 11 and display the digital signals on the oscilloscope.
2) Record how long (in microseconds) the voltage at PIN 8 stays high for an individual pulse.
3) Record how long (in microseconds) the voltage at PIN 11 stays high for an individual pulse. You should see three quick pulses because of the three different ways that the PIN is written to.

Write Up:

There is no write up for this lab, just get the items on the cover sheet signed off by your lab instructor and submit just the cover sheet in class.

Homework: (Total estimate: ~ 1 hours, 0 minutes.)

1) (30 minutes) Try out the ultrasonic distance sensor in your kit. 

Note that the samples code in the ARDUINO IDE is for a different sensor that has a single wire interface instead of the two wire interface your sensor has. For the code sample, use the code given below and wire up the sensor accordingly.

Try running the sensor code and observe how the sensor works to measure distance. You can read about what the distance sensor does on this page:
http://arduino-info.wikispaces.com/UltraSonicDistance

```
const int PING_TRIG_PIN = A2; //pin which triggers ultrasonic sound
const int ECHO_PIN = A1; //pin which delivers time to receive echo using pulseIn()
int MaxDistanceTimeout=18000;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  pinMode(PING_TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
}

void loop() {
  //sending the signal, starting with LOW for a clean signal
  digitalWrite(PING_TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(PING_TRIG_PIN, HIGH);
  delayMicroseconds(5);
  digitalWrite(PING_TRIG_PIN, LOW);
  int distance=pulseIn(ECHO_PIN, HIGH, MaxDistanceTimeout) / 29 / 2; // timeout after 18000
microseconds (300 cm)
  if (distance==0) distance= MaxDistanceTimeout / 29 / 2;  // if no ping, assume object is
at farthest distance
  Serial.println(distance);
  delay(100);
}
```

2) (30 minutes) Read the lab handout for next week.